Red Hat OpenStack Platform 16.2

director 的安装和使用

使用 Red Hat OpenStack Platform director 创建 OpenStack 云环境

Last Updated: 2024-07-30

Red Hat OpenStack Platform 16.2 director 的安装和使用

使用 Red Hat OpenStack Platform director 创建 OpenStack 云环境

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

使用 Red Hat OpenStack Platform director 在企业环境中安装 Red Hat OpenStack Platform 16。其中包括安装 director、规划您的环境以及使用 director 创建 OpenStack 环境。

目录

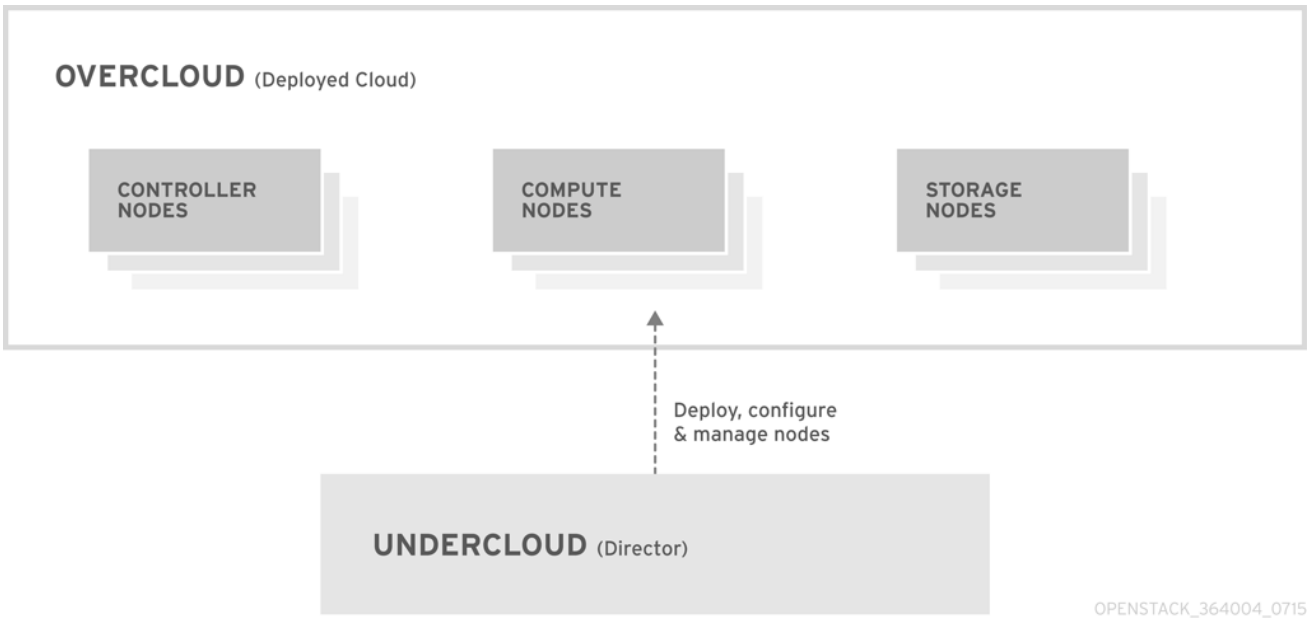
让开源更具包容性

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。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 DIRECTOR 简介

Red Hat OpenStack Platform (RHOSP) director是一个安装和管理完整的 RHOSP 环境的工具组。director 主要基于 OpenStack 项目 TripleO。您可以使用 director 安装全面运行、精益且稳定的 RHOSP 环境，该环境可置备和控制裸机系统以用作 OpenStack 节点。

director 使用两个主要概念：undercloud 和 overcloud。首先，您要安装 undercloud，然后使用 undercloud 作为安装和配置 overcloud 的工具。



1.1. 了解 UNDERCLOUD

undercloud 是包含 Red Hat OpenStack Platform director 工具组的主要管理节点。它是一个单一系统的 OpenStack 安装，包括置备和管理组成 OpenStack 环境 (overcloud) 的 OpenStack 节点的组件。组成 undercloud 的组件具有多个功能：

环境规划

undercloud 包括用户可用于创建和分配某些节点角色的规划功能。undercloud 包括一组可分配给特定节点的默认节点角色：计算 (Compute)、控制器 (Controller) 和各种存储角色。您也可以设计自定义角色。此外，您还可以选择每个节点角色包括哪些 Red Hat OpenStack Platform 服务。这提供了对新节点类型建模，或把特定组件隔离在其自己的主机中的方法。

裸机系统控制

undercloud 使用每个节点的带外管理接口（通常是智能平台管理接口 (IPMI)）进行电源管理控制，使用基于 PXE 的服务发现硬件属性并在每个节点上安装 OpenStack。您可以使用此功能将裸机系统置备为 OpenStack 节点。有关电源管理驱动程序的完整列表，请参阅 第 30 章 *电源管理驱动*。

编配

undercloud 包含一组 YAML 模板，这些模板代表您环境中的一系列计划。undercloud 导入这些计划，并根据其中的指示创建所需的 OpenStack 环境。这些计划还可以包括 hook。通过使用 hook，可以在环境创建过程中的特定点上融入您自己的自定义设置。

undercloud 组件

undercloud 使用 OpenStack 组件作为它的基本工具组。每个组件都在 undercloud 中的一个独立容器内运行：

- OpenStack Identity (keystone) - 提供 director 组件的验证和授权功能。
- OpenStack Bare Metal (ironic) 和 OpenStack Compute (nova) - 管理裸机节点。

- OpenStack Networking (neutron) 和 Open vSwitch - 控制裸机节点的网络。
- OpenStack Image Service (glance) - 存储 director 写入裸机的镜像。
- OpenStack Orchestration (heat) 和 Puppet - 提供对节点的编配功能，并在 director 把 overcloud 镜像写入到磁盘后配置节点。
- OpenStack Workflow Service (mistral) - 为特定 director 操作（如导入和部署计划）提供一组工作流程。
- OpenStack Messaging Service（zaqar） - 为 OpenStack Workflow Service 提供一个消息服务。
- OpenStack Object Storage（swift） - 为不同的 OpenStack 平台组件提供对象存储，包括：
  - 为 OpenStack Image Service 提供镜像存储
  - 为 OpenStack Bare Metal 提供内省数据
  - 为 OpenStack Workflow Service 提供部署计划

1.2. 了解 OVERCLOUD

overcloud 是 undercloud 创建的 Red Hat OpenStack Platform (RHOSP) 环境。overcloud 包括多个具有不同角色的节点，这些角色是基于您要创建的 OpenStack Platform 环境定义的。undercloud 包括一组默认的 overcloud 节点角色：

Controller

Controller 节点为 OpenStack 环境提供管理、网络和高可用性功能。建议的 OpenStack 环境是在一个高可用性集群中包含三个 Controller 节点。
一个默认 Controller（控制器）节点角色支持以下组件。不是所有这些服务都默认启用。其中一些组件需要自定义或者预打包环境文件才能启用：

- OpenStack Dashboard (horizon)
- OpenStack Identity (keystone)
- OpenStack Compute (nova) API
- OpenStack Networking (neutron)
- OpenStack Image Service (glance)
- OpenStack Block Storage (cinder)
- OpenStack Object Storage (swift)
- OpenStack Orchestration (heat)
- OpenStack Shared File Systems (manila)
- OpenStack Bare Metal (ironic)
- OpenStack Load Balancing-as-a-Service (octavia)
- OpenStack Key Manager (barbican)

- MariaDB
- Open vSwitch
- 高可用性服务的 Pacemaker 和 Galera。

计算

Compute 节点为 OpenStack 环境提供计算资源。随着时间的推移，可以通过添加更多节点来扩展您的环境。一个默认 Compute （计算）节点包括以下组件：

- OpenStack Compute (nova)
- KVM/QEMU
- OpenStack Telemetry (ceilometer) 代理
- Open vSwitch

存储

存储节点为 OpenStack 环境提供存储。以下列表包含有关 RHOSP 中存储节点的各种类型的信息：

- Ceph Storage 节点 - 用来组成存储集群。每个节点包含一个 Ceph Object Storage Daemon (OSD)。此外，当您部署 Ceph Storage 节点作为环境一部分时，director 将 Ceph Monitor 安装到 Controller 节点上。
- Block storage (cinder) - 用作高可用性 Controller 节点的外部块存储。这类节点包括以下组件：
  - OpenStack Block Storage (cinder) 卷
  - OpenStack Telemetry 代理
  - Open vSwitch.
- Object Storage (swift) - 这些节点为 OpenStack Swift 提供了一个外部存储层。Controller 节点通过 Swift 代理访问对象存储节点。对象存储节点包含以下组件：
  - OpenStack Object Storage (swift) 存储
  - OpenStack Telemetry 代理
  - Open vSwitch.

1.3. 了解 RED HAT OPENSTACK PLATFORM 中的高可用性

Red Hat OpenStack Platform（RHOSP）director 使用 Controller 节点集群向 OpenStack Platform 环境提供高可用性服务。对于每个服务，director 在所有 Controller 节点上安装相同的组件，并作为单个服务一起管理这些 Controller 节点。在单个 Controller 节点上出现运行故障时，这种集群配置提供回退（fallback）机制。这可以使 OpenStack 用户实现某种程度的不中断运行。

OpenStack Platform director 使用以下软件来管理 Controller 节点上的组件：

- Pacemaker - Pacemaker 是集群资源管理器。Pacemaker 管理和监控集群中所有节点上的 OpenStack 组件的可用性。

- HA Proxy（HA 代理） - 为集群提供负载均衡和代理服务。
- Galera - 在集群中复制 RHOSP 数据库。
- Memcached - 提供数据库缓存服务。

注意

- 自版本 13 起，您可以使用 director 部署计算实例的高可用性 (Instance HA)。使用 Instance HA，当 Compute 节点出现故障时，可以自动清空该 Compute 节点上的实例。

1.4. 了解 RED HAT OPENSTACK PLATFORM 中的容器化

undercloud 和 overcloud 上的各个 OpenStack Platform 服务在对应节点的单独 Linux 容器内运行。这种容器化提供了一种隔离服务、维护环境和升级 Red Hat OpenStack Platform (RHOSP) 的方法。

Red Hat OpenStack Platform 16.2 支持安装在 Red Hat Enterprise Linux 8.4 操作系统上。Red Hat Enterprise Linux 8.4 不再包含 Docker，并提供一组新的工具替换 Docker 生态系统。这意味着 OpenStack Platform 16.2 使用这些新工具替换 Docker 进行 OpenStack Platform 部署和升级。

Podman

Pod Manager (Podman) 是容器管理工具。它几乎实现所有 Docker CLI 命令，但不包括与 Docker Swarm 相关的命令。Podman 管理 pod、容器和容器镜像。Podman 和 Docker 之间的一个主要差异是 Podman 可以在后台没有运行守护进程的情况下管理资源。
有关 Podman 的更多信息，请访问 Podman 网站。

Buildah

Buildah 专门构建您与 Podman 一起使用的 Open Containers Initiative (OCI) 镜像。Buildah 命令复制 Dockerfile 的内容。Buildah 还提供一个较低级别的 **coreutils** 接口以构建容器镜像，因此您无需 Dockerfile 即可构建容器。Buildah 还使用其他脚本语言在无需守护进程的情况下构建容器镜像。
有关 Buildah 的更多信息，请访问 Buildah 网站。

Skopeo

Skopeo 使操作员能够检查远程容器镜像，帮助 director 在拉取镜像时收集数据。其他功能包括在 registry 间复制容器镜像，以及从 registry 中删除镜像。

红帽支持使用以下方式为您的 overcloud 管理容器镜像：

- 将容器镜像从 Red Hat Container Catalog 拉取到 undercloud 上的 **image-serve** registry，然后从 **image-serve** registry 拉取镜像。当您首先将镜像拉取到 undercloud 时，要避免多个 overcloud 节点同时通过外部连接拉取容器镜像。
- 从 Satellite 6 服务器拉取容器镜像。您可以直接从 Satellite 拉取这些镜像，因为网络流量是内部流量。

本指南包含有关配置容器镜像 registry 的详细信息和执行基本容器操作的信息。

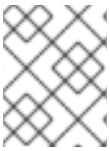
1.5. 在 RED HAT OPENSTACK PLATFORM 中使用 CEPH STORAGE

对于使用 Red Hat OpenStack Platform (RHOSP) 为成千上万的客户端提供服务的大型机构来说，这很常见。每个 OpenStack 客户端在消耗块存储资源时可能会有自己的独特需求。在单个节点上部署 glance（镜像）、cinder（卷）和 nova（计算）可能无法管理具有数千客户端的大型部署。在外部扩展

OpenStack 可解决此问题。

但是，在实际的环境中，仍然需要一个存储层的虚拟化解决方案（如 Red Hat Ceph Storage）来扩展 RHOSP 的存储层，使它可以支持 terabyte、petabyte 甚至 exabyte 数量级的存储要求。Red Hat Ceph Storage 提供了这样一个存储虚拟化层，在商用硬件上运行时具有高可用性和高性能。尽管虚拟化可能看似会影响性能，Ceph 会将块设备镜像以条带化的形式作为对象分布在集群中上的对象，这意味着大型 Ceph Block Device 镜像的性能实际上比独立磁盘性能更好。另外，Cept Block 设备还支持缓存、copy-on-write cloning 和 copy-on-read cloning 功能来提高性能。

有关 Red Hat Ceph Storage 的更多信息，请参阅 Red Hat Ceph Storage 。



注意

对于多架构云，红帽仅支持预安装或外部 Ceph 实施。有关更多信息，请参阅将 Overcloud 与现有 Red Hat Ceph 集群集成，并为 overcloud 配置 CPU 架构 。

第 2 章 规划您的 UNDERCLOUD

在 undercloud 上配置并安装 director 之前，您必须规划 undercloud 主机以确保它满足某些要求。

2.1. 容器化 UNDERCLOUD

undercloud 是控制最终 Red Hat OpenStack Platform (RHOSP) 环境（称为 overcloud）的配置、安装和管理的节点。undercloud 将每个 RHOSP 组件服务作为容器运行。undercloud 使用这些容器化服务创建名为 director 的工具集，用于创建和管理 overcloud。

undercloud 和 overcloud 都使用容器，它们使用相同的架构来拉取、配置和运行容器。此架构基于用于置备节点的 OpenStack Orchestration 服务 (heat) 并使用 Ansible 配置服务和容器。熟悉 heat 和 Ansible 有助于诊断您可能遇到的问题。

2.2. 准备 UNDERCLOUD 网络

undercloud 需要访问两个主要网络：

- **Provisioning 网络或 Control Plane 网络**，这是 director 用于置备节点和在执行 Ansible 配置时通过 SSH 访问这些节点的网络。此网络还支持从 undercloud 到 overcloud 节点的 SSH 访问。undercloud 包含在此网络上内省和置备其他节点的 DHCP 服务，这意味着此网络上不应存在其他 DHCP 服务。director 为此网络配置接口。

- **External 网络**，支持访问 OpenStack Platform 软件仓库、容器镜像源和 DNS 服务器或 NTP 服务器等的其他服务器。使用此网络从您的工作站对 undercloud 进行标准访问。您必须在 undercloud 上手动配置一个接口以访问外部网络。

undercloud 至少需要 2 个 1 Gbps 网络接口卡：一个用于 **Provisioning 或 Control Plane 网络**，一个用于 **External 网络**。

规划网络时，请查看以下准则：

- 红帽建议使用一个网络来置备，为数据平面使用 control plane 和另一个网络。不要在 OVS 网桥上创建置备和 control plane 网络。

- provisioning 和 control plane 网络可以在 Linux 绑定或独立接口上配置。如果您使用 Linux 绑定，请将其配置为 active-backup 绑定类型。

  - 在非控制器节点上，在置备和 control plane 网络中流量数量相对较低，它们不需要高带宽或负载均衡。

  - 在 Controller 上，provisioning 和 control plane 网络需要额外的带宽。增加带宽的原因是控制器为其它角色中的多个节点提供服务。当对环境进行频繁更改时，还需要更多的带宽。为获得最佳性能，当控制器有超过 50 个计算节点 - 或者同时置备了四个以上的裸机节点，则应该同时置备 4-10 倍，则其带宽应为非控制器节点上的接口的 4-10 倍。

- 当置备超过 50 个 overcloud 节点时，undercloud 应该具有更高的带宽与 provisioning 网络的连接。

- 不要使用与您从工作站访问 director 机器相同的 Provisioning 或 Control Plane NIC。director 安装会使用 Provisioning NIC 创建一个网桥，它会忽略所有远程连接。使用 External NIC 来远程连接 director 系统。

- Provisioning 网络需要一个与您的环境大小相匹配的 IP 范围。使用以下原则来决定包括在这个范围内的 IP 地址总数量：

  - 要为每个监听 IP 地址（用于在发期间连接到 Provisioning 网络的每个节点

  - 至少包括一个临时 IP 地址，用于内省期间连接到 Provisioning 网络的每个节点。
  - 至少包括一个永久 IP 地址，用于部署期间连接到 Provisioning 网络的每个节点。
  - 包括一个额外的 IP 地址，用于 Provisioning 网络上 overcloud 高可用性集群的虚拟 IP。
  - 包括此范围内的额外 IP 地址，以用于扩展环境。

- 为防止 Controller 节点网卡或网络交换机故障破坏 overcloud 服务可用性，请确保 keystone 管理端点位于使用绑定网卡或网络硬件冗余的网络中。如果将 Keystone 端点移到不同的网络，如 **internal_api**，请确保 undercloud 可以访问 VLAN 或子网。有关更多信息，请参阅红帽知识库解决方案如何将 Keystone 管理端点迁移到 internal_api 网络 。

2.3. 确定环境规模

在安装 undercloud 前，请确定环境的规模。规划环境时应考虑到以下因素：

想要在 overcloud 中部署多少个节点？

undercloud 管理 overcloud 中的每个节点。置备 overcloud 节点会消耗 undercloud 上的资源。您必须为 undercloud 提供足够的资源用来置备和控制所有 overcloud 节点。

您希望 undercloud 同步执行多少个操作？

undercloud 上的大部分 OpenStack 服务会使用一组 worker。每个 worker 执行特定于该服务的一个操作。多个 worker 提供同步操作。undercloud 上 worker 的默认数量是 undercloud 上 CPU 线程总数的一半在本实例中，线程数是指 CPU 内核数乘以超线程值。例如，如果 undercloud 的 CPU 具有 16 个线程，则 director 服务默认生成 8 个 worker。director 还默认使用一组最小值和最大值限制

服务	最小值	最大值
OpenStack Orchestration (heat)	4	24
所有其他服务	2	12

undercloud 具有以下最小 CPU 和内存要求：

- 支持 Intel 64 或 AMD64 CPU 扩展的 8 线程 64 位 x86 处理器。这可为每个 undercloud 服务提供 4 个 worker。
- 最少 24 GB RAM。
  - 每 10 个由 undercloud 部署的主机，**ceph-ansible** playbook 需要占用 1 GB 常驻集合大小 (RSS)。如果要在部署中使用新的或现有的 Ceph 集群，您必须相应地置备 undercloud RAM。

要使用更多 worker，使用以下建议增加 undercloud 的 vCPU 和内存：

- **最小值：**每个线程使用 1.5 GB 内存。例如，一台有 48 个线程的机器需要 72 GB RAM 来为 24 个 heat worker 和 12 个其他服务的 worker 提供最小的覆盖范围。
- **建议值：**每个线程使用 3 GB 内存。例如，一台有 48 个线程的机器需要 144 GB RAM 来为 24 个 heat worker 和 12 个其他服务的 worker 提供建议的覆盖范围。

2.4. UNDERCLOUD 磁盘大小调整

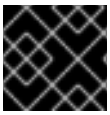
建议的最小 undercloud 磁盘大小为在根磁盘上有 **100 GB** 可用磁盘空间：

- 20 GB 用于容器镜像
- 10 GB 在节点部署过程中用于 QCOW2 镜像转换和缓存
- 70 GB 或更大空间供常规使用、保存日志和指标以及满足增长需要

2.5. 虚拟化支持

红帽仅支持以下平台上的虚拟化 undercloud：

平台	备注
基于内核的虚拟机 (KVM)	由 Red Hat Enterprise Linux 8.4 托管，如认证的虚拟机监控程序上所列。
Red Hat Virtualization	由 Red Hat Virtualization 4.x 托管，如认证的虚拟机监控程序上所列。
Microsoft Hyper-V	由各种版本的 Hyper-V 托管，如红帽客户门户认证目录上所列。
VMware ESX 和 ESXi	由各种版本的 ESX 和 ESXi 托管，如红帽客户门户认证目录上所列。



重要

确保您的管理程序支持 Red Hat Enterprise Linux 8.4 客户机。

虚拟机要求

虚拟 undercloud 的资源要求与裸机 undercloud 的资源要求类似。在置备时考虑各种调优选项，如网络模型、客户机 CPU 功能、存储后端、存储格式和缓存模式。

网络注意事项

电源管理

undercloud 虚拟机(VM)需要访问 overcloud 节点的电源管理设备。这是注册节点时为 **pm_addr** 参数设置的 IP 地址。

Provisioning 网络

用于 provisioning 网络的 NIC **ctlplane** 需要向 overcloud 裸机节点的 NIC 广播和提供 DHCP 请求。创建一个网桥，将虚拟机的 NIC 连接到与裸机 NIC 相同的网络。

允许来自未知地址的流量

您必须配置虚拟 undercloud 管理程序，以防止虚拟机监控程序阻止 undercloud 从未知地址传输流量。配置取决于您用于虚拟 undercloud 的平台：

- Red Hat Enterprise Virtualization：禁用 **anti-mac-spoofing** 参数。
- VMware ESX 或 ESXi:

- 在 IPv4 **ctlplane** 网络中：允许伪传输。
- 在 IPv6 **ctlplane** 网络中：允许伪传输、MAC 地址更改和适当的模式操作。
  有关如何配置 VMware ESX 或 ESXi 的更多信息，请参阅 VMware 文档网站上的 vSphere 标准交换机。

应用这些设置后，必须关闭再打开 director 虚拟机。重新引导虚拟机还远远不够。

2.6. 字符编码配置

Red Hat OpenStack Platform 有特殊的字符编码要求，作为本地设置的一部分：

- 在所有节点上使用 UTF-8 编码。确保在所有节点上将 **LANG** 环境变量设置为 **en_US.UTF-8**。
- 如果您使用 Red Hat Ansible Tower 自动创建 Red Hat OpenStack Platform 资源，请避免使用非 ASCII 字符。

2.7. 使用代理运行 UNDERCLOUD 时的注意事项

使用代理运行 undercloud 有某些限制，红帽建议您使用 Red Hat Satellite 进行 registry 和软件包管理。

但是，如果您的环境使用代理，请查看这些注意事项，以透彻理解将 Red Hat OpenStack Platform OpenStack 的组件与代理集成的不同配置方法，以及每种方法的限制。

系统范围的代理配置

使用此方法为 undercloud 上的所有网络流量配置代理通信。要配置代理设置，请编辑 **/etc/environment** 文件并设置以下环境变量：

http_proxy
: 用于标准 HTTP 请求的代理。

https_proxy
: 用于 HTTP 请求的代理。

no_proxy
: 从代理通信中排除的、以逗号分隔的域列表。

系统范围的代理方法有以下限制：

- 由于 **pam_env** PAM 模块中的固定大小缓冲区，**no_proxy** 的最大长度为 1024 个字符。
- 有些容器错误地绑定并解析 **/etc/environments** 中的环境变量，这会导致运行这些服务时出现问题。有关更多信息，请参阅 BZ#1916070 - 未在容器中正确获取 /etc/environment 文件中的代理配置更新和 BZ#1918408 - mistral_executor 容器无法正确设置 no_proxy 环境参数 。

dnf 代理配置

使用此方法将 **dnf** 配置为通过代理运行所有流量。要配置代理设置，请编辑 **/etc/dnf/dnf.conf** 文件并设置以下参数：

proxy
: 代理服务器的 URL。

proxy_username
: 用于连接到代理服务器的用户名。

proxy_password

用于连接到代理服务器的密码。

proxy_auth_method

代理服务器使用的身份验证方法。

有关这些选项的更多信息，请运行 **man dnf.conf**。

dnf 代理方法有以下限制：

- 此方法仅对 **dnf** 提供代理支持。
- **dnf** 代理方法不包括用于从代理通信中排除某些主机的选项。

Red Hat Subscription Manager 代理

使用此方法将 Red Hat Subscription Manager 配置为通过代理运行所有流量。要配置代理设置，请编辑 **/etc/rhsm/rhsm.conf** 文件并设置以下参数：

proxy_hostname

用于代理的主机。

proxy_scheme

在将代理写入软件仓库定义时，代理的方案。

proxy_port

代理的端口。

proxy_username

用于连接到代理服务器的用户名。

proxy_password

用于连接到代理服务器的密码。

no_proxy

要从代理通信中排除的、以逗号分隔的特定主机的主机名后缀列表。

有关这些选项的更多信息，请运行 **man rhsm.conf**。

Red Hat Subscription Manager 代理方法有以下限制：

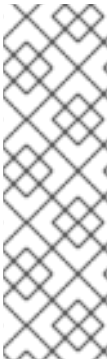
- 此方法仅对 Red Hat Subscription Manager 提供代理支持。
- Red Hat Subscription Manager 代理配置的值覆盖为系统范围环境变量设置的任何值。

透明代理

如果您的网络使用透明代理来管理应用层流量，则不需要将 undercloud 本身配置为与代理交互，因为代理管理会自动发生。透明代理可帮助克服 Red Hat OpenStack Platform 中与基于客户端的代理配置相关的限制。

2.8. UNDERCLOUD 软件仓库

RHOSP (RHOSP) 16.2 在 Red Hat Enterprise Linux (RHEL) 8.4 上运行。因此，您必须将这些软件仓库中的内容锁定到相应的 RHEL 版本。



注意

- 如果使用 Red Hat Satellite 同步软件仓库，您可以启用 RHEL 软件仓库的特定版本。但是，无论您选择的版本，存储库标签仍保持不变。例如，如果您启用 BaseOS 存储库的 8.4 版本，存储库名称包括您启用的特定版本，但存储库标签仍然是 **rhel-8-for-x86_64-baseos-eus-rpms**。
- 不再需要 **advanced-virt-for-rhel-8-x86_64-rpms** 和 **advanced-virt-for-rhel-8-x86_64-eus-rpms** 存储库。要禁用这些软件仓库，请参阅 OSP 16.2 不再需要红帽知识库解决方案 advanced-virt-for-rhel-8-x86_64-rpms。

警告

此处指定的软件仓库之外的任何软件仓库都不受支持。除非建议，请勿启用下表中所列之外的任何其他产品或软件仓库，否则可能会遇到依赖软件包问题。请勿启用 Extra Packages for Enterprise Linux (EPEL)。

核心软件仓库

下表列出了用于安装 undercloud 的核心软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 系统的基本操作系统仓库。
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-eus-rpms**	包含 RHOSP 依赖项。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	RHEL 的高可用性工具。用于 Controller 节点的高可用性功能。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	RHEL 的 Ansible Engine。用于提供最新版本的 Ansible。
RHOSP 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-x86_64-rpms**	Core RHOSP 存储库，其中包含 RHOSP director 的软件包。
Red Hat Fast datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。

Ceph 软件仓库

下表列出了用于 undercloud 的与 Ceph Storage 有关的软件仓库。

名称	软件仓库	要求说明
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	提供节点与 Ceph Storage 集群进行通信的工具。如果您计划在 overcloud 中使用 Ceph Storage，或者要与现有 Ceph Storage 集群集成，undercloud 需要此软件仓库的 **ceph-ansible** 软件包。

IBM POWER 软件仓库

下表包含 POWER PC 架构上 RHOSP 的软件仓库列表。使用这些软件仓库来替代核心软件仓库中的同等库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux for IBM Power，little endian - BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le 系统的基本操作系统软件仓库。
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	包含 RHOSP 依赖项。
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-rpms**	RHEL 的高可用性工具。用于 Controller 节点的高可用性功能。
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。
Red Hat Ansible Engine 2.9 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.9-for-rhel-8-ppc64le-rpms**	RHEL 的 Ansible Engine。提供最新版本的 Ansible。
Red Hat OpenStack Platform 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-ppc64le-rpms**	ppc64le 系统的核心 RHOSP 存储库。

第 3 章 DIRECTOR 安装准备

要安装和配置 director，您必须完成一些准备任务，以确保已将 undercloud 注册到红帽客户门户网站或 Red Hat Satellite 服务器中，已安装了 director 软件包，并且您配置了 director 的容器镜像源，以便在安装过程中拉取容器镜像。

3.1. 准备 UNDERCLOUD

在安装 director 前，您必须在主机上完成一些基本配置。

步骤

1. 以 **root** 用户身份登录 undercloud。
2. 创建 **stack** 用户：

```
[root@director ~]# useradd stack
```

3. 为该用户设置密码：

```
[root@director ~]# passwd stack
```

4. 进行以下操作，以使用户在使用 **sudo** 时无需输入密码：

```
[root@director ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a /etc/sudoers.d/stack
[root@director ~]# chmod 0440 /etc/sudoers.d/stack
```

5. 切换到新的 **stack** 用户：

```
[root@director ~]# su - stack
[stack@director ~]$
```

6. 为系统镜像和 heat 模板创建目录：

```
[stack@director ~]$ mkdir ~/images
[stack@director ~]$ mkdir ~/templates
```

director 使用系统镜像和 heat 模板来创建 overcloud 环境。红帽建议创建这些目录来帮助您组织本地文件系统。

7. 检查 undercloud 的基础和完整主机名：

```
[stack@director ~]$ hostname
[stack@director ~]$ hostname -f
```

如果上述命令没有显示正确的完全限定主机名或报告错误，则使用 **hostnamectl** 设置主机名：

```
[stack@director ~]$ sudo hostnamectl set-hostname undercloud.example.com
[stack@director ~]$ sudo hostnamectl set-hostname --transient undercloud.example.com
```

8. 如果您所使用的 DNS 服务器无法解析 undercloud 主机完全限定域名 (FQDN)，请编辑 **/etc/hosts** 并为系统主机名包含一个条目。**/etc/hosts** 中的 IP 地址必须与您计划用于 undercloud

公共 API 的地址匹配。例如，如果系统使用 **undercloud.example.com** 作为 FQDN，使用 **10.0.0.1** 作为 IP 地址，则将以下行添加到 **/etc/hosts**：

```
10.0.0.1  undercloud.example.com undercloud
```

9. 如果您计划让 Red Hat OpenStack Platform director 位于 overcloud 或其身份提供程序之外的独立域，则必须将额外的域添加到 /etc/resolv.conf：

```
search overcloud.com idp.overcloud.com
```

重要

您必须为 DNS 启用端口扩展(**dns_domain_ports**)的 DNS 域，以便内部解析 RHOSP 环境中端口的名称。使用 **NeutronDnsDomain** 默认值 **openstacklocal** 意味着网络服务不会内部解析 DNS 的端口名称。如需更多信息，请参阅网络指南中的指定 DNS 分配给端口的名称。

3.2. 注册 UNDERCLOUD 并附加订阅

*在安装 director 前，您必须运行 **subscription-manager** 来注册 undercloud 并附加有效的 Red Hat OpenStack Platform 订阅。*

步骤

1. *以 **stack** 用户身份登录 undercloud。*
2. *在红帽 Content Delivery Network 或 Red Hat Satellite 注册您的系统。例如，运行以下命令在 Content Delivery Network 中注册系统。根据提示输入您的客户门户网站用户名和密码：*

```
[stack@director ~]$ sudo subscription-manager register
```

3. *查找 Red Hat OpenStack Platform (RHOSP) director 的权利池 ID：*

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
                     Red Hat Virtualization
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             Valid-Pool-Number-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

4. 找到 **Pool ID** 值并附加 Red Hat OpenStack Platform 16.2 权利：

```
[stack@director ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

5. 将 undercloud 锁定到 Red Hat Enterprise Linux 8.4：

```
$ sudo subscription-manager release --set=8.4
```

3.3. 为 UNDERCLOUD 启用软件仓库

启用 undercloud 所需的软件仓库，并将系统软件包更新至最新版本。

步骤

1. 以 **stack** 用户身份登录 undercloud。
2. 禁用所有默认的软件仓库，然后启用所需的 Red Hat Enterprise Linux 软件仓库：

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos \
--enable=rhel-8-for-x86_64-baseos-tus-rpms \
--enable=rhel-8-for-x86_64-appstream-tus-rpms \
--enable=rhel-8-for-x86_64-highavailability-tus-rpms \
--enable=ansible-2.9-for-rhel-8-x86_64-rpms \
--enable=openstack-16.2-for-rhel-8-x86_64-rpms \
--enable=fast-datapath-for-rhel-8-x86_64-rpms
```

这些仓库包括了安装 director 所需的软件包。

3. 将 **container-tools** 软件仓库模块设置为版本 **3.0**：

```
[stack@director ~]$ sudo dnf module reset container-tools
[stack@director ~]$ sudo dnf module enable -y container-tools:3.0
```

4. 在系统上执行更新，确保您有最新的基本系统软件包：

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

3.4. 安装 DIRECTOR 软件包

安装与 Red Hat OpenStack Platform director 相关的软件包。

步骤

1. 安装用于安装和配置 director 的命令行工具：

```
[stack@director ~]$ sudo dnf install -y python3-tripleoclient
```

3.5. 安装 CEPH-ANSIBLE

*当在 Red Hat OpenStack Platform 中使用 Ceph Storage 时，需要 **ceph-ansible** 软件包。*

流程

1. *启用 Ceph 工具存储库：*

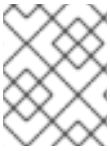
```
[stack@director ~]$ sudo subscription-manager repos --enable=rhceph-4-tools-for-rhel-8-x86_64-rpms
```

2. *安装 **ceph-ansible** 软件包：*

```
[stack@director ~]$ sudo dnf install -y ceph-ansible
```

3.6. 准备容器镜像

undercloud 安装需要一个环境文件来确定从何处获取容器镜像以及如何存储它们。生成并自定义此环境文件，您可以使用这个文件准备容器镜像。



注意

如果需要为您的 undercloud 配置特定的容器镜像版本，必须将镜像固定到特定的版本。有关更多信息，请参阅为 undercloud 固定容器镜像。

步骤

1. *以 **stack** 用户身份登录 undercloud 主机。*

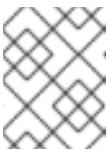
2. *生成默认的容器镜像准备文件：*

```
$ sudo openstack tripleo container image prepare default \
  --local-push-destination \
  --output-env-file containers-prepare-parameter.yaml
```

此命令包括以下附加选项：

- ***--local-push-destination**，在 undercloud 上设置 registry 作为存储容器镜像的位置。这意味着 director 从 Red Hat Container Catalog 拉取必要的镜像并将其推送到 undercloud 上的 registry 中。director 将该 registry 用作容器镜像源。如果直接从 Red Hat Container Catalog 拉取镜像，请忽略这个选项。*

- ***--output-env-file** 是环境文件名称。此文件的内容包括用于准备您的容器镜像的参数。在本例中，文件的名称是 **containers-prepare-parameter.yaml**。*



注意

*您可以使用相同的 **containers-prepare-parameter.yaml** 文件为 undercloud 和 overcloud 定义容器镜像源。*

3. *修改 **containers-prepare-parameter.yaml** 以符合您的需求。*

3.7. 容器镜像准备参数

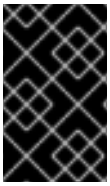
*用于准备您的容器的默认文件（**containers-prepare-parameter.yaml**）包含 **ContainerImagePrepare** heat 参数。此参数定义一个用于准备一系列镜像的策略列表：*

```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

*每一策略接受一组子参数，它们定义要使用哪些镜像以及对这些镜像执行哪些操作。下表包含有关您可与每个 **ContainerImagePrepare** 策略配合使用的子参数的信息：*

参数	描述
excludes	用于从策略中排除镜像名称的正则表达式列表。
includes	用于在策略中包含的正则表达式列表。至少一个镜像名称必须与现有镜像匹配。如果指定了 **includes**，将忽略所有 **excludes**。
modify_append_tag	要附加到目标镜像标签的字符串。例如，如果使用标签 16.2.3-5.161 拉取镜像并将 **modify_append_tag** 设置为 **-hotfix**，director 会将最终镜像标记为 16.2.3-5.161-hotfix。
modify_only_with_labels	过滤想要修改的镜像的镜像标签字典。如果镜像与定义的标签匹配，则 director 将该镜像包括在修改过程中。
modify_role	在上传期间但在将镜像推送到目标 registry 之前运行的 ansible 角色名称字符串。
modify_vars	要传递给 **modify_role** 的变量的字典。

参数	描述
push_destination	定义用于在上传过程中要将镜像推送到的 registry 的命名空间。 ● 如果设为 **true**，则使用主机名将 **push_destination** 设置为 undercloud registry 命名空间，这是建议的方法。 ● 如果设置为 **false**，则不会推送到本地 registry，节点直接从源拉取镜像。 ● 如果设置为自定义值，则 director 将镜像推送到外部本地 registry。 当直接从 Red Hat Container Catalog 拉取镜像时，如果在生产环境中将此参数设置为 **false**，则所有 overcloud 节点都将通过外部连接同时从 Red Hat Container Catalog 拉取镜像，这可能会导致出现带宽问题。仅使用 **false** 直接从托管容器镜像的 Red Hat Satellite Server 拉取。 如果 **push_destination** 参数设置为 **false** 或未定义，且远程 registry 需要身份验证，请将 **ContainerImageRegistryLogin** 参数设置为 **true**，并使用 **ContainerImageRegistryCredentials** 参数包含凭据。
pull_source	从中拉取原始容器镜像的源 registry 。
set	用于定义从何处获取初始镜像的 **key: value** 定义的字典。
tag_from_label	使用指定容器镜像元数据标签的值为每个镜像创建标签，并拉取该标记的镜像。例如，如果设置了 **tag_from_label: {version}-{release}**，则 director 会使用 **version** 和 **release** 标签来构造新标签。对于一个容器，**version** 可能会设置为 16.2.3，**release** 可能会设置为 **5.161**，这会产生标签 16.2.3-5.161。仅当未在 **set** 字典中定义 **tag** 时，director 才使用此参数。

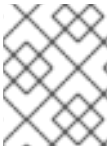


重要

*将镜像推送到 undercloud 时，请使用 **push_destination: true** 而不是 **push_destination: UNDERCLOUD_IP:PORT**。**push_destination: true** 方法在 IPv4 和 IPv6 地址之间提供了一定程度的一致性。*

***set** 参数接受一组 **key: value** 定义：*

键	描述
ceph_image	Ceph Storage 容器镜像的名称。
ceph_namespace	Ceph Storage 容器镜像的命名空间。
ceph_tag	Ceph Storage 容器镜像的标签。
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	Ceph Storage Alert Manager 容器镜像的名称、命名空间和标签。
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	Ceph Storage Grafana 容器镜像的名称、命名空间和标签。
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	Ceph Storage Node Exporter 容器镜像的名称、命名空间和标签。
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	Ceph Storage Prometheus 容器镜像的名称、命名空间和标签。
name_prefix	各个 OpenStack 服务镜像的前缀。
name_suffix	各个 OpenStack 服务镜像的后缀。
namespace	各个 OpenStack 服务镜像的命名空间。
neutron_driver	用于确定要使用的 OpenStack Networking (neutron) 容器的驱动程序。null 值代表使用标准的 **neutron-server** 容器。设为 **ovn** 可使用基于 OVN 的容器。
tag	为来自源的所有镜像设置特定的标签。如果没有定义，director 将 Red Hat OpenStack Platform 版本号用作默认值。此参数优先于 **tag_from_label** 值。



注意

*容器镜像使用基于 Red Hat OpenStack Platform 版本的多流标签。这意味着不再有 **latest** 标签。*

3.8. 容器镜像标记准则

Red Hat Container Registry 使用特定的版本格式来标记所有 Red Hat OpenStack Platform 容器镜像。此格式遵循每个容器的标签元数据，即 **version-release**。

version

对应于 Red Hat OpenStack Platform 的主要和次要版本。这些版本充当包含一个或多个发行版本的流。

release

对应于版本流中特定容器镜像版本的发行版本。

例如，如果 Red Hat OpenStack Platform 的最新版本为 16.2.3，容器镜像的发行版本为 **5.161**，则生成的容器镜像标签为 16.2.3-5.161。

Red Hat Container Registry 还使用一组主要和次要 **version** 标签，链接到该容器镜像版本的最新发行版本。例如，16.2 和 16.2.3 链接到 16.2.3 容器流中的最新 **release**。如果出现 16.2 的新次要发行版本，16.2 标签链接到新次要发行版本流的最新 **release**，而 16.2.3 标签则继续链接到 16.2.3 流中的最新 **release**。

ContainerImagePrepare 参数包含两个子参数，可用于确定要下载的容器镜像。这些子参数是 **set** 字典中的 **tag** 参数，以及 **tag_from_label** 参数。使用以下准则来确定要使用 **tag** 还是 **tag_from_label**。

- **tag** 的默认值是您的 OpenStack Platform 版本的主要版本。对于此版本，它是 16.2。这始终对应于最新的次要版本和发行版本。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.2
      ...
```

- 要更改为 OpenStack Platform 容器镜像的特定次要版本，请将标签设置为次要版本。例如，若要更改为 16.2.2，可将 **tag** 设置为 16.2.2。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.2.2
      ...
```

- 在设置 **tag** 时，director 始终会在安装和更新期间下载 **tag** 中设置的版本的最新容器镜像 **release**。

- 如果没有设置 **tag**，则 director 会结合使用 **tag_from_label** 的值和最新的主要版本。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      # tag: 16.2
      ...
    tag_from_label: '{version}-{release}'
```

- ***tag_from_label** 参数根据其从 Red Hat Container Registry 中检查到的最新容器镜像发行版本的标签元数据生成标签。例如，特定容器的标签可能会使用以下 **version** 和 **release** 元数据：*

```
"Labels": {
  "release": "5.161",
  "version": "16.2.3",
  ...
}
```

- ***tag_from_label** 的默认值为 **{version}-{release}**，对应于每个容器镜像的版本和发行版本元数据标签。例如，如果容器镜像的 **version** 设置为 16.2.3，**release** 设置为 5.161，生成的容器镜像标签为 16.2.3-5.161。*

- ***tag** 参数始终优先于 **tag_from_label** 参数。要使用 **tag_from_label**，在容器准备配置中省略 **tag** 参数。*

- ***tag** 和 **tag_from_label** 之间的一个关键区别是：director 仅基于主要或次要版本标签使用 **tag** 拉取镜像，Red Hat Container Registry 将这些标签链接到版本流中的最新镜像发行版本，而 director 使用 **tag_from_label** 对每个容器镜像执行元数据检查，以便 director 生成标签并拉取对应的镜像。*

3.9. 从私有 REGISTRY 获取容器镜像

***registry.redhat.io** registry 需要身份验证才能访问和拉取镜像。要通过 **registry.redhat.io** 和其他私有 registry 进行身份验证，请在 **containers-prepare-parameter.yaml** 文件中包括 **ContainerImageRegistryCredentials** 和 **ContainerImageRegistryLogin** 参数。*

ContainerImageRegistryCredentials

*有些容器镜像 registry 需要进行身份验证才能访问镜像。在这种情况下，请使用您的 **containers-prepare-parameter.yaml** 环境文件中的 **ContainerImageRegistryCredentials** 参数。**ContainerImageRegistryCredentials** 参数使用一组基于私有 registry URL 的键。每个私有 registry URL 使用其自己的键和值对定义用户名（键）和密码（值）。这提供了一种为多个私有 registry 指定凭据的方法。*

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      my_username: my_password
```

*在示例中，用身份验证凭据替换 **my_username** 和 **my_password**。红帽建议创建一个 registry 服务帐户并使用这些凭据访问 **registry.redhat.io** 内容，而不使用您的个人用户凭据。*

*要指定多个 registry 的身份验证详情，请在 **ContainerImageRegistryCredentials** 中为每个 registry 设置多个键对值：*

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
```

```
    namespace: registry.redhat.io/...
    ...
  - push_destination: true
    set:
      namespace: registry.internalsite.com/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
    registry.internalsite.com:
      myuser2: '0th3rp@55w0rd!'
    '192.0.2.1:8787':
      myuser3: '@n0th3rp@55w0rd!'
```

重要

默认 **ContainerImagePrepare** 参数从需要进行身份验证的 **registry.redhat.io** 拉取容器镜像。

如需更多信息，请参阅 Red Hat Container Registry 身份验证。

ContainerImageRegistryLogin

ContainerImageRegistryLogin 参数用于控制 overcloud 节点系统是否需要登录到远程 registry 来获取容器镜像。当您想让 overcloud 节点直接拉取镜像，而不是使用 undercloud 托管镜像时，会出现这种情况。

如果 **push_destination** 设置为 false 或未用于给定策略，则必须将 **ContainerImageRegistryLogin** 设置为 **true**。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      namespace: registry.redhat.io/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: true
```

但是，如果 overcloud 节点没有与 **ContainerImageRegistryCredentials** 中定义的 registry 主机的网络连接，并将此 **ContainerImageRegistryLogin** 设置为 **true**，则尝试进行登录时部署可能会失败。如果 overcloud 节点没有与 **ContainerImageRegistryCredentials** 中定义的 registry 主机的网络连接，请将 **push_destination** 设置为 **true**，将 **ContainerImageRegistryLogin** 设置为 **false**，以便 overcloud 节点从 undercloud 拉取镜像。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  ...
```

```
ContainerImageRegistryCredentials:
  registry.redhat.io:
    myuser: 'p@55w0rd!'
ContainerImageRegistryLogin: false
```

3.10. 分层镜像准备条目

ContainerImagePrepare 参数的值是一个 YAML 列表。这意味着您可以指定多个条目。

以下示例演示了两个条目，director 使用所有镜像的最新版本，**nova-api** 镜像除外，该镜像使用标记为 **16.2.1-hotfix** 的版本：

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: "{version}-{release}"
    push_destination: true
    excludes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel8
      name_prefix: openstack-
      name_suffix: ''
      tag:16.2
  - push_destination: true
    includes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel8
      tag: 16.2.1-hotfix
```

includes 和 **excludes** 参数使用正则表达式来控制每个条目的镜像筛选。匹配 **includes** 策略的镜像的优先级高于 **excludes** 匹配项。镜像名称必须与 **includes** 或 **excludes** 正则表达式值匹配才能被认为匹配。

如果您的 Block Storage (cinder)驱动程序需要供应商提供的 cinder-volume 镜像（称为插件），则会使用类似的技术。如果您的块存储驱动程序需要插件，请参阅高级 Overcloud 自定义指南中的 部署供应商插件。

3.11. 排除 CEPH STORAGE 容器镜像

默认的 overcloud 角色配置使用默认的 Controller、Compute 和 Ceph Storage 角色。但是，如果使用默认角色配置来部署不含 Ceph Storage 节点的 overcloud，director 仍然会从 Red Hat Container Registry 拉取 Ceph Storage 容器镜像，因为这些镜像是作为默认配置的一部分包含在其中。

如果您的 overcloud 不需要 Ceph Storage 容器，则可以将 director 配置为不从 Red Hat Container Registry 拉取 Ceph Storage 容器镜像。

步骤

1. 编辑 **containers-prepare-parameter.yaml** 文件以排除 Ceph Storage 容器：

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    excludes:
```

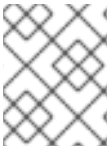
```
    - ceph
    - prometheus
  set:
    ...
```

excludes 参数使用正则表达式来排除包含 **ceph** 或 **prometheus** 字符串的任何容器镜像。

2. 保存 **containers-prepare-parameter.yaml** 文件。

3.12. 准备期间修改镜像

可在准备镜像期间修改镜像，然后立即使用修改的镜像部署 overcloud。



注意

Red Hat OpenStack Platform (RHOSP) Director 支持在准备 RHOSP 容器（而非 Ceph 容器）期间修改镜像。

修改镜像的情况包括：

- 作为连续集成管道的一个部分，在部署之前使用要测试的更改修改镜像。
- 作为开发工作流的一个部分，必须部署本地更改以进行测试和开发。
- 必须部署更改，但更改没有通过镜像构建管道提供。例如，添加专有附加组件或紧急修复。

要在准备期间修改镜像，可在您要修改的每个镜像上调用 Ansible 角色。该角色提取源镜像，进行请求的更改，并标记结果。准备命令可将镜像推送到目标 registry，并设置 heat 参数以引用修改的镜像。

Ansible 角色 **tripleo-modify-image** 与所需的角色接口相一致，并提供修改用例必需的行为。使用 **ContainerImagePrepare** 参数中与修改相关的键控制修改：

- **modify_role** 指定要为每个镜像调用的 Ansible 角色进行修改。
- **modify_append_tag** 将字符串附加到源镜像标签的末尾。这可以标明生成的镜像已被修改过。如果 **push_destination** registry 已包含修改的镜像，则使用此参数跳过修改。在每次修改镜像时都更改 **modify_append_tag**。
- **modify_vars** 是要传递给角色的 Ansible 变量的字典。

要选择 **tripleo-modify-image** 角色处理的用例，将 **tasks_from** 变量设置为该角色中所需的文件。

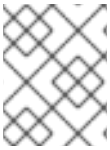
在开发和测试修改镜像的 **ContainerImagePrepare** 条目时，运行镜像准备命令（无需任何其他选项），以确认镜像已如期修改：

```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

重要

要使用 **openstack tripleo container image prepare** 命令，您的 undercloud 必须包含一个正在运行的 **image-serve** registry。这样，在新的 undercloud 安装之前您将无法运行此命令，因为 **image-serve** registry 将不会被安装。您可以在成功安装 undercloud 后运行此命令。

3.13. 更新容器镜像的现有软件包



注意

Red Hat OpenStack Platform (RHOSP) director 支持更新 RHOSP 容器的容器镜像上的现有软件包，不适用于 Ceph 容器。

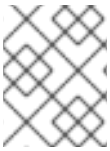
步骤

- 以下示例 **ContainerImagePrepare** 条目使用 undercloud 主机的 dnf 软件仓库配置在容器镜像的所有软件包中更新：

```
ContainerImagePrepare:
- push_destination: true
  ...
  modify_role: tripleo-modify-image
  modify_append_tag: "-updated"
  modify_vars:
    tasks_from: yum_update.yml
    compare_host_packages: true
    yum_repos_dir_path: /etc/yum.repos.d
  ...
```

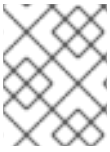
3.14. 将额外的 RPM 文件安装到容器镜像中

您可以在容器镜像中安装 RPM 文件的目录。这对安装修补程序、本地软件包内部版本或任何通过软件包仓库无法获取的软件包都非常有用。



注意

Red Hat OpenStack Platform (RHOSP) Director 支持将额外的 RPM 文件安装到 RHOSP 容器的容器镜像，而不是 Ceph 容器。



注意

在现有部署中修改容器镜像时，您必须执行次要更新，以将更改应用到 overcloud。如需更多信息，请参阅 *保持 Red Hat OpenStack Platform 更新*。

步骤

- 以下示例 **ContainerImagePrepare** 条目仅在 **nova-compute** 镜像上安装一些热修复软件包：

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

3.15. 通过自定义 DOCKERFILE 修改容器镜像

您可以指定包含 Dockerfile 的目录，以进行必要的更改。调用 **tripleo-modify-image** 角色时，该角色生成 **Dockerfile.modified** 文件，而该文件更改 **FROM** 指令并添加额外的 **LABEL** 指令。

注意

Red Hat OpenStack Platform (RHOSP) Director 支持使用 RHOSP 容器（而非 Ceph 容器）的自定义 Dockerfile 修改容器镜像。

步骤

1. 以下示例在 **nova-compute** 镜像上运行自定义 Dockerfile ：

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: modify_image.yml
    modify_dir_path: /home/stack/nova-custom
  ...
```

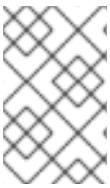
2. 以下示例显示了 **/home/stack/nova-custom/Dockerfile** 文件。运行任何 **USER** 根指令后，必须切换回原始镜像默认用户：

```
FROM registry.redhat.io/rhosp-rhel8/openstack-nova-compute:latest

USER "root"

COPY customize.sh /tmp/
RUN /tmp/customize.sh

USER "nova"
```

3.16. 为容器镜像准备 SATELLITE 服务器

Red Hat Satellite 6 提供了注册表同步功能。通过该功能可将多个镜像提取到 Satellite 服务器中，作为应用程序生命周期的一部分加以管理。Satellite 也可以作为 registry 供其他启用容器功能的系统使用。有关管理容器镜像的更多信息，请参阅 Red Hat Satellite 6 内容管理指南 中的管理容器镜像。

以下操作过程示例中使用了 Red Hat Satellite 6 的 **hammer** 命令行工具和一个名为 **ACME** 的示例组织。请将该组织替换为您自己 Satellite 6 中的组织。



注意

此过程需要身份验证凭据以从 **registry.redhat.io** 访问容器镜像。红帽建议创建一个 registry 服务帐户并使用这些凭据访问 **registry.redhat.io** 内容，而不使用您的个人用户凭据。有关更多信息，请参阅"红帽容器 registry 身份验证"。

步骤

1. 创建所有容器镜像的列表：

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{ .Name }}" | sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-dashboard-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/openshift" | grep ose-prometheus
```

   - 如果您计划安装 Ceph 并启用 Ceph 仪表板，则需要以下 ose-prometheus 容器：

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.6
registry.redhat.io/openshift4/ose-prometheus:v4.6
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.6
```

2. 将 **satellite_images** 文件复制到包含 Satellite 6 **hammer** 工具的系统中。或者，根据 Hammer CLI 指南中的说明将 **hammer** 工具安装到 undercloud 中。

3. 运行以下 **hammer** 命令在 Satellite 机构中创建新产品 **(OSP 容器)**：

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP Containers"
```

   该定制产品将会包含您的镜像。

4. 添加 **satellite_images** 文件中的 overcloud 容器镜像：

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io\///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. 添加 Ceph Storage 4 容器镜像：

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```

注意

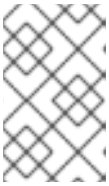
如果要安装 Ceph 仪表板，请在 **hammer repository create** 命令中包含 **--name rhceph-4-dashboard-rhel8** ：

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-dashboard-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-dashboard-rhel8
```

6. 同步容器镜像：

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP Containers"
```

等待 Satellite 服务器完成同步。



注意

根据具体配置情况，**hammer** 可能会询问您的 Satellite 服务器用户名和密码。您可以使用配置文件将 **hammer** 配置为自动登录。有关更多信息，请参阅 Hammer CLI 指南中的身份验证章节。

7. 如果您的 Satellite 6 服务器使用内容视图，请创建一个用于纳入镜像的新内容视图版本，并在应用生命周期的不同环境之间推进这个视图。这在很大程度上取决于您如何构建应用程序的生命周期。例如，如果您的生命周期中有一个称为 **production** 的环境，并且希望容器镜像在该环境中可用，则创建一个包含容器镜像的内容视图，并将该内容视图推进到 **production** 环境中。有关更多信息，请参阅管理内容视图。

8. 检查 **base** 镜像的可用标签：

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --lifecycle-environment "production" \
  --product "OSP Containers"
```

此命令显示内容视图中特定环境的 OpenStack Platform 容器镜像的标签。

9. 返回到 undercloud，并生成默认的环境文件，它将您的 Satellite 服务器用作源来准备镜像。运行以下示例命令以生成环境文件：

```
$ sudo openstack tripleo container image prepare default \
  --output-env-file containers-prepare-parameter.yaml
```

- **--output-env-file** 是环境文件名称。此文件的内容包括用于为 undercloud 准备容器镜像的参数。在本例中，文件的名称是 **containers-prepare-parameter.yaml**。

10. 编辑 **containers-prepare-parameter.yaml** 文件并修改以下参数：

- ***push_destination*** - *根据您选择的容器镜像管理策略，将此参数设置为* ***true*** *或* ***false****。如果将此参数设置为* ***false****，则 overcloud 节点直接从 Satellite 拉取镜像。如果将此参数设置为* ***true****，则 director 将镜像从 Satellite 拉取到 undercloud registry，overcloud 从 undercloud registry 拉取镜像。*

- ***namespace*** - *Satellite 服务器上 registry 的 URL 和端口。Red Hat Satellite 上的默认 registry 端口为 443。*

- ***name_prefix*** - *该前缀基于 Satellite 6 规范。它的值根据您是否使用了内容视图而不同：*

  - *如果您使用了内容视图，则前缀的结构为* ***[组织]-[环境]-[内容视图]-[产品]-****。例如：* ***acme-production-myosp16-osp_containers-****。*

  - *如果不使用内容视图，则前缀的结构为* ***[组织]-[产品]-****。例如：* ***acme-osp_containers-****。*

- ***ceph_namespace***、***ceph_image***、***ceph_tag*** - *如果使用 Ceph Storage，请额外纳入这些参数以定义 Ceph Storage 容器镜像的位置。请注意，* ***ceph_image*** *现包含特定于 Satellite 的前缀。这个前缀与* ***name_prefix*** *选项的值相同。*

以下示例环境文件包含特定于 Satellite 的参数：

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:443
      neutron_driver: null
      tag: '16.2'
      ...
```

注意

要使用存储在 Red Hat Satellite Server 上的特定容器镜像版本，请将 ***标签*** *键值对设置为* ***set*** *字典中的特定版本。例如，要使用 16.2.2 镜像流，请在* ***set*** *字典中设置* ***tag: 16.2.2****。*

您必须在 ***undercloud.conf*** *配置文件中定义* ***containers-prepare-parameter.yaml*** *环境文件，否则 undercloud 将使用默认值：*

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```

第 4 章 在 UNDERCLOUD 上安装 DIRECTOR

要配置并安装 director，请在 **undercloud.conf** 文件中设置适当的参数并运行 undercloud 安装命令。安装 director 后，导入 director 在节点置备期间用于写入裸机节点的 overcloud 镜像。

4.1. 配置 DIRECTOR

director 安装过程需要 **undercloud.conf** 配置文件中的某些设置，director 从 **stack** 用户的主目录中读取该文件。完成以下步骤，复制默认模板作为基础进行配置。

步骤

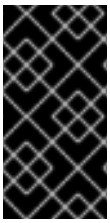
1. 将默认模板复制到 **stack** 用户的主目录：

```
[stack@director ~]$ cp \
  /usr/share/python-tripleoclient/undercloud.conf.sample \
  ~/undercloud.conf
```

2. 编辑 **undercloud.conf** 文件。这个文件包含用于配置 undercloud 的设置。如果忽略或注释掉某个参数，undercloud 安装将使用默认值。

4.2. DIRECTOR 配置参数

以下列表包含用于配置 **undercloud.conf** 文件的参数的相关信息。将所有参数保留在相关部分内以避免出错。



重要

您至少必须将 **container_images_file** 参数设置为包含容器镜像配置的环境文件。如果没有将此参数正确设置为适当的文件，则 director 无法从 **ContainerImagePrepare** 参数获取容器镜像规则集，也无法从 **ContainerImageRegistryCredentials** 参数获取容器 registry 身份验证详情。

默认值

以下参数会在 **undercloud.conf** 文件的 **[DEFAULT]** 部分中进行定义：

additional_architectures

overcloud 支持的附加（内核）架构的列表。目前，除了默认的 **x86_64** 架构外，overcloud 还支持 **ppc64le** 架构。

注意

当启用对 ppc64le 的支持时，还必须将 **ipxe_enabled** 设置为 **False**。有关使用多个 CPU 架构配置 undercloud 的更多信息，请参阅 配置多个 CPU 架构 overcloud 。

certificate_generation_ca

为所请求证书签名的 CA 的 **certmonger** 别名。仅在设置了 **generate_service_certificate** 参数的情况下使用此选项。如果您选择 **local** CA，certmonger 会将本地 CA 证书提取到 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** 并将该证书添加到信任链中。

clean_nodes

确定是否在部署之间和内省之后擦除硬盘。

cleanup

删除临时文件。把它设置为 **False** 以保留部署期间使用的临时文件。如果出现错误，临时文件可帮助您调试部署。

container_cli

用于容器管理的 CLI 工具。将此参数设置为 **podman**。Red Hat Enterprise Linux 8.4 仅支持 **podman**。

container_healthcheck_disabled

禁用容器化服务运行状况检查。红帽建议您启用运行状况检查，并将此选项设置为 **false**。

container_images_file

含有容器镜像信息的 Heat 环境文件。此文件可能包含以下条目：

- 所有需要的容器镜像的参数
- **ContainerImagePrepare** 参数（用于推动必要的镜像准备）。通常，含有此参数的文件被命名为 **containers-prepare-parameter.yaml**。

container_insecure_registries

供 **podman** 使用的不安全 registry 列表。如果您想从其他来源（如私有容器 registry）拉取镜像，则使用此参数。在大多数情况下，如果在 Satellite 中注册了 undercloud，**podman** 就有从 Red Hat Container Catalog 或 Satellite Server 拉取容器镜像的证书。

container_registry_mirror

配置的 **podman** 使用的可选 **registry-mirror**。

custom_env_files

要添加到 undercloud 安装中的其他环境文件。

deployment_user

安装 undercloud 的用户。如果此参数保留为不设置，则使用当前的默认用户 **stack**。

discovery_default_driver

为自动注册的节点设置默认驱动程序。需要启用 **enable_node_discovery** 参数，且必须在 **enabled_drivers** 列表中包含驱动程序。

enable_ironic; enable_ironic_inspector; enable_mistral; enable_nova; enable_tempest; enable_validations; enable_zaqar

定义要为 director 启用的核心服务。保留这些参数设为 **true**。

enable_node_discovery

自动注册通过 PXE 引导内省虚拟内存盘 (ramdisk) 的所有未知节点。新节点使用 **fake** 作为默认驱动程序，但您可以设置 **discovery_default_driver** 覆盖它。您也可以使用内省规则为新注册的节点指定驱动程序信息。

enable_novajoin

定义是否在 undercloud 中安装 **novajoin** 元数据服务。

enable_routed_networks

定义是否支持路由的 control plane 网络。

enable_swift_encryption

定义是否启用 Swift 加密。

enable_telemetry

定义是否在 undercloud 中安装 OpenStack Telemetry 服务（gnocchi、aodh、panko）。如果您想自动安装和配置 telemetry 服务，则将 **enable_telemetry** 参数设为 **true**。默认值为 **false**，即在

undercloud 中禁用 telemetry。如果使用要利用指标数据的其他产品，如 Red Hat CloudForms，则需要这个参数。

警告

每个组件都不支持 RBAC。Alarming 服务(aodh)和 Gnocchi 不考虑安全 RBAC 规则。

enabled_hardware_types

要为 undercloud 启用的硬件类型的列表。

generate_service_certificate

定义 undercloud 安装期间是否生成 SSL/TLS 证书，此证书用于 **undercloud_service_certificate** 参数。undercloud 安装会保存生成的证书 **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem**。**certificate_generation_ca** 参数中定义的 CA 将为此证书签名。

heat_container_image

要使用的 heat 容器镜像的 URL。请保留不设置。

heat_native

使用 **heat-all** 运行基于主机的 undercloud 配置。请保留为 **true**。

hieradata_override

在 director 上配置 Puppet hieradata 的 **hieradata** 覆盖文件的路径，为 **undercloud.conf** 参数外的服务提供自定义配置。如果设置此参数，undercloud 安装会将此文件复制到 **/etc/puppet/hieradata** 目录并将其设为层次结构中的第一个文件。有关使用此功能的更多信息，请参阅在 undercloud 上配置 hieradata。

inspection_extras

指定在内省的过程中是否启用额外的硬件集合。此参数在内省镜像上需要 **python-hardware** 或 **python-hardware-detect** 软件包。

inspection_interface

该 director 用来进行节点内省的网桥。这是 director 配置创建的自定义网桥。**LOCAL_INTERFACE** 会附加到这个网桥。请保留使用默认的值（**br-ctlplane**）。

inspection_runbench

在节点内省过程中运行一组基准测试。将此参数设为 **true** 以启用基准测试。如果您需要在检查注册节点的硬件时执行基准数据分析操作，则需要使用这个参数。

ipa_otp

定义在 IPA 服务器注册 undercloud 节点使用的一次性密码。启用 **enable_novajoin** 之后需要提供此密码。

ipv6_address_mode

undercloud 置备网络的 IPv6 地址配置模式。以下列表包含这个参数的可能值：

- dhcpv6-stateless - 使用路由器公告 (RA) 的地址配置以及使用 DHCPv6 的可选信息。
- DHCPv6-stateful - 地址配置和使用 DHCPv6 的可选信息。

ipxe_enabled

定义使用 iPXE 还是标准的 PXE。默认为 **true**，其启用 iPXE。将此参数设置为 **false** 以使用标准 PXE。对于 PowerPC 部署，或混合了 PowerPC 和 x86 的部署，请将此值设置为 **false**。

local_interface

指定 director Provisioning NIC 的接口。这也是该 director 用于 DHCP 和 PXE 引导服务的设备。把这个项的值改为您选择的设备。使用 **ip addr** 命令可以查看连接了哪些设备。以下是一个 **ip addr** 命令的结果输出示例：

```
2: em0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic em0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

在这个例子中，External NIC 使用 **em0**，Provisioning NIC 使用 **em1**（当前没有被配置）。在这种情况下，将 **local_interface** 设置为 **em1**。配置脚本会把这个接口附加到一个自定义的网桥（由 **inspection_interface** 参数定义）上。

local_ip

为 director Provisioning NIC 定义的 IP 地址。这也是 director 用于 DHCP 和 PXE 引导服务的 IP 地址。除非 Provisioning 网络需要使用其他子网（如该 IP 地址与环境中的现有 IP 地址或子网冲突）保留默认值 **192.168.24.1/24**。
对于 IPv6，本地 IP 地址前缀长度必须是 **/64**，以支持有状态和无状态连接。

local_mtu

要用于 **local_interface** 的最大传输单元 (MTU)。对于 undercloud 不要超过 1500。

local_subnet

要用于 PXE 引导和 DHCP 接口的本地子网。**local_ip** 地址应该属于这个子网。默认值为 **ctlplane-subnet**。

net_config_override

网络配置覆盖模板的路径。如果设置此参数，undercloud 将使用 JSON 或 YAML 格式的模板以使用 **os-net-config** 配置网络，并忽略 **undercloud.conf** 中设置的网络参数。当您要配置绑定或向接口添加一个选项时，请使用此参数。有关自定义 undercloud 网络接口的更多信息，请参阅配置 undercloud 网络接口。

networks_file

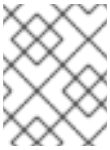
覆盖用于 **heat** 的网络文件。

output_dir

输出状态目录、处理的 heat 模板和 Ansible 部署文件。

overcloud_domain_name

要在部署 overcloud 时使用的 DNS 域名。



注意

配置 overcloud 时，必须将 **CloudDomain** 参数设置为匹配的值。配置 overcloud 时，在环境文件中设置此参数。

roles_file

要用来覆盖用于 undercloud 安装的默认角色文件的角色文件。强烈建议您将此参数保留为不设置，以便 director 安装使用默认的角色文件。

scheduler_max_attempts

调度程序尝试部署实例的次数上限。此值必须大于或等于您期望一次部署的裸机节点数，以避免调度时的潜在争用情形。

service_principal

使用该证书的服务的 Kerberos 主体。仅在您的 CA 需要 Kerberos 主体（如在 FreeIPA 中）时使用此参数。

subnets

用于置备和内省的路由网络子网的列表。默认值仅包括 **ctlplane-subnet** 子网。如需更多信息，请参阅 子网。

templates

要覆盖的 heat 模板文件。

undercloud_admin_host

通过 SSL/TLS 为 director Admin API 端点定义的 IP 地址或主机名。director 配置将 IP 地址作为路由的 IP 地址附加到 director 软件网桥，其使用 **/32** 子网掩码。
如果 **undercloud_admin_host** 不在与 **local_ip** 相同的 IP 网络中，您必须将 **ControlVirtualInterface** 参数设置为您希望 undercloud 上的 admin API 侦听的接口。默认情况下，admin API 侦听 **br-ctlplane** 接口。在自定义环境文件中设置 **ControlVirtualInterface** 参数，并通过配置 **custom_env_files** 参数在 **undercloud.conf** 文件中包含自定义环境文件。

有关自定义 undercloud 网络接口的详情，请参考配置 undercloud 网络接口。

undercloud_debug

把 undercloud 服务的日志级别设置为 **DEBUG**。将此值设置为 **true** 以启用 **DEBUG** 日志级别。

undercloud_enable_selinux

在部署期间启用或禁用 SELinux。除非调试问题，否则强烈建议保留此值设为 **true**。

undercloud_hostname

定义 undercloud 的完全限定主机名。如果设置，undercloud 安装将配置所有系统主机名设置。如果保留未设置，undercloud 将使用当前的主机名，但您必须相应地配置所有主机名设置。

undercloud_log_file

用于存储 undercloud 安装和升级日志的日志文件的路径。默认情况下，日志文件是主目录中的 **install-undercloud.log**。例如，**/home/stack/install-undercloud.log**。

undercloud_nameservers

用于 undercloud 主机名解析的 DNS 名称服务器列表。

undercloud_ntp_servers

帮助同步 undercloud 日期和时间的网络时间协议服务器列表。

undercloud_public_host

通过 SSL/TLS 为 director Public API 端点定义的 IP 地址或主机名。director 配置将 IP 地址作为路由的 IP 地址附加到 director 软件网桥，其使用 **/32** 子网掩码。
如果 **undercloud_public_host** 不在与 **local_ip** 相同的 IP 网络中，您必须将 **PublicVirtualInterface** 参数设置为您希望 undercloud 上公共 API 侦听的公共接口。默认情况下，公共 API 侦听 **br-ctlplane** 接口。在自定义环境文件中设置 **PublicVirtualInterface** 参数，并通过配置 **custom_env_files** 参数在 **undercloud.conf** 文件中包含自定义环境文件。

有关自定义 undercloud 网络接口的详情，请参考配置 undercloud 网络接口。

undercloud_service_certificate

用于 OpenStack SSL/TLS 通信的证书的位置和文件名。理想的情况是从一个信任的证书认证机构获得这个证书。否则，生成自己的自签名证书。

undercloud_timezone

undercloud 的主机时区。如果未指定时区，director 将使用现有时区配置。

undercloud_update_packages

定义是否在安装 undercloud 期间更新软件包。

子网

每个置备子网在 **undercloud.conf** 文件中都有一个对应的同名部分。例如，要创建称为 **ctlplane-subnet** 的子网，在 **undercloud.conf** 文件中使用以下示例：

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

您可以根据自身环境所需来指定相应数量的置备网络。

重要

director 在创建子网后无法更改子网的 IP 地址。

cidr

director 用来管理 overcloud 实例的网络。这是 undercloud **neutron** 服务管理的 Provisioning 网络。保留其默认值 **192.168.24.0/24**，除非您需要 Provisioning 网络使用其他子网。

masquerade

定义是否伪装 **cidr** 中定义的用于外部访问的网络。这为 Provisioning 网络提供了网络地址转换 (NAT)，使 Provisioning 网络能够通过 director 进行外部访问。

注意

director 配置还使用相关 **sysctl** 内核参数自动启用 IP 转发。

dhcp_start; dhcp_end

overcloud 节点 DHCP 分配范围的开始值和终止值。确保此范围包含足够的 IP 地址，以分配给您的节点。如果没有为子网指定，director 通过删除为 **local_ip**、**gateway**、**undercloud_admin_host**、undercloud_public_host、**undercloud_public_host** 以及 **inspection_iprange** 参数的值来确定分配池。
您可以通过指定启动和结束地址对列表，为 undercloud control plane 子网配置非持续分配池。另外，您可以使用 **dhcp_exclude** 选项排除 IP 地址范围中的 IP 地址。例如，以下配置同时创建分配池 **172.20.0.100-172.20.0.150** 和 **172.20.0.200-172.20.0.250** ：

选项1

```
dhcp_start = 172.20.0.100,172.20.0.200
dhcp_end = 172.20.0.150,172.20.0.250
```

- 选项 2

```
dhcp_start = 172.20.0.100
dhcp_end = 172.20.0.250
dhcp_exclude = 172.20.0.151-172.20.0.199
```

dhcp_exclude

DHCP 分配范围中排除的 IP 地址。例如，以下配置排除 IP 地址 **172.20.0.105** 和 IP 地址范围 **172.20.0.210-172.20.0.219** ：

```
dhcp_exclude = 172.20.0.105,172.20.0.210-172.20.0.219
```

dns_nameservers

特定于子网的 DNS 名称服务器。如果没有为子网定义名称服务器，子网将使用 **undercloud_nameservers** 参数中定义的名称服务器。

gateway

overcloud 实例的网关。它是 undercloud 主机，会把网络流量转发到外部网络。保留其默认值 **192.168.24.1**，除非您需要 director 使用其他 IP 地址，或想直接使用外部网关。

host_routes

此网络上 overcloud 实例的 Neutron 管理的子网的主机路由。这也为 undercloud 上的 **local_subnet** 配置主机路由。

inspection_iprange

在检查过程中，此网络上的节点要使用的临时 IP 范围。这个范围不得与 **dhcp_start** 和 **dhcp_ end** 定义的范围重叠，但必须位于同一个 IP 子网中。

根据您的配置修改这些参数的值。完成后，请保存文件。

4.3. 使用环境文件配置 UNDERCLOUD

您通过 **undercloud.conf** 文件配置 undercloud 的主要参数。您还可以使用包含 heat 参数的环境文件来执行额外的 undercloud 配置。

步骤

1. 创建名为 **/home/stack/templates/custom-undercloud-params.yaml** 的环境文件。
2. 编辑此文件并包括您的 heat 参数。例如，要为特定的 OpenStack Platform 服务启用调试功能，请在 **custom-undercloud-params.yaml** 文件中包括以下代码段：

   ```
   parameter_defaults:
     Debug: True
   ```

   完成后保存此文件。
3. 编辑 **undercloud.conf** 文件，找到 **custom_env_files** 参数。编辑该参数以指向 **custom-undercloud-params.yaml** 环境文件：

   ```
   custom_env_files = /home/stack/templates/custom-undercloud-params.yaml
   ```

注意

您可以使用逗号分隔列表指定多个环境文件。

director 安装在下次安装或升级 undercloud 的操作过程中包括此环境文件。

4.4. 用于 UNDERCLOUD 配置的常见 HEAT 参数

下表包含您可能在自定义环境文件中为 undercloud 设置的一些常见 heat 参数。

参数	描述
AdminPassword	设置 undercloud **admin** 用户密码。
AdminEmail	设置 undercloud **admin** 用户电子邮件地址。
Debug	启用调试模式。

在自定义环境文件的 **parameter_defaults** 部分下设置这些参数：

```
parameter_defaults:
  Debug: True
  AdminPassword: "myp@ssw0rd!"
  AdminEmail: "admin@example.com"
```

4.5. 在 UNDERCLOUD 上配置 HIERADATA

您可以通过在 director 上配置 Puppet hieradata，为可用 **undercloud.conf** 参数之外的服务提供自定义配置。

步骤

1. 创建一个 hieradata 覆盖文件，例如 **/home/stack/hieradata.yaml**。
2. 将自定义的 hieradata 添加到该文件。例如，添加以下代码段，将 Compute (nova) 服务参数 **force_raw_images** 从默认值 **True** 改为 **False**：

   ```
   nova::compute::force_raw_images: False
   ```

   如果没有为您要设置的参数实施 Puppet，则使用以下方法配置该参数：

   ```
   nova::config::nova_config:
     DEFAULT/<parameter_name>:
       value: <parameter_value>
   ```

   例如：

   ```
   nova::config::nova_config:
     DEFAULT/network_allocate_retries:
       value: 20
   ```

```
  ironic/serial_console_state_timeout:
    value: 15
```

3. 将 **undercloud.conf** 文件中的 **hieradata_override** 参数设置为新 **/home/stack/hieradata.yaml** 文件的路径：

```
hieradata_override = /home/stack/hieradata.yaml
```

4.6. 为使用 IPV6 的裸机置备配置 UNDERCLOUD

如果有使用 IPv6 的节点和基础架构，您可以将 undercloud 和置备网络配置为使用 IPv6 而不是 IPv4，以便 director 能够在 IPv6 节点上置备和部署 Red Hat OpenStack Platform。但是，有一些注意事项：

- 双堆栈 IPv4/6 不可用。
- Tempest 验证可能无法正确执行。
- 在升级过程中，无法进行 IPv4 到 IPv6 的迁移。

修改 **undercloud.conf** 文件，以便在 Red Hat OpenStack Platform 中启用 IPv6 置备。

先决条件

- undercloud 上的 IPv6 地址。有关更多信息，请参阅 Overcloud 的 IPv6 网络指南中的在 undercloud 上配置 IPv6 地址。

流程

1. 打开 **undercloud.conf** 文件。
2. 将 IPv6 地址模式指定为无状态或有状态：

```
[DEFAULT]
ipv6_address_mode = <address_mode>
...
```

根据 NIC 支持的模式，将 < **address_mode** > 替换为 **dhcpv6-stateless** 或 **dhcpv6-stateful**。

注意

当您使用有状态地址模式时，固件、链加载程序和操作系统可能会使用不同的算法来生成 DHCP 服务器跟踪的 ID。DHCPv6 不会按 MAC 跟踪地址，如果来自请求者的标识符值变化，则不会提供相同的地址，但 MAC 地址保持不变。因此，当使用有状态 DHCPv6 时，还必须完成下一步来配置网络接口。

3. 如果您将 undercloud 配置为使用有状态 DHCPv6，请指定用于裸机节点的网络接口：

```
[DEFAULT]
ipv6_address_mode = dhcpv6-stateful
ironic_enabled_network_interfaces = neutron,flat
...
```

4. 为裸机节点设置默认网络接口：

```
[DEFAULT]
...
ironic_default_network_interface = neutron
...
```

5. 指定 undercloud 是否应该在 provisioning 网络中创建路由器：

```
[DEFAULT]
...
enable_routed_networks: <true/false>
...
```

- 将 **<true/false>** 替换为 **true** 以启用路由网络，并防止 undercloud 在 provisioning 网络中创建路由器。为 **true** 时，数据中心路由器必须提供路由器公告。
- 将 **<true/false>** 替换为 **false** 来禁用路由网络，并在 provisioning 网络中创建一个路由器。

6. 配置本地 IP 地址，以及 director Admin API 和公共 API 端点通过 SSL/TLS 的 IP 地址：

```
[DEFAULT]
...
local_ip = <ipv6_address>
undercloud_admin_host = <ipv6_address>
undercloud_public_host = <ipv6_address>
...
```

将 **<ipv6_address>** 替换为 undercloud 的 IPv6 地址。

7. 可选：配置 director 用来管理实例的 provisioning 网络：

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
...
```

- 将 **<ipv6_address>** 替换为不使用默认 provisioning 网络时用于管理实例的网络的 IPv6 地址。
- 将 **<ipv6_prefix>** 替换为不使用默认置备网络时用于管理实例的网络的 IP 地址前缀。

8. 为置备节点配置 DHCP 分配范围：

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
...
```

- 将 **<ipv6_address_dhcp_start>** 替换为要用于 overcloud 节点的网络范围的 IPv6 地址。
- 将 **<ipv6_address_dhcp_end>** 替换为要用于 overcloud 节点的网络范围末尾的 IPv6 地址。

9. 可选：配置将流量转发到外部网络的网关：

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
```

```
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

在不使用默认网关时，将 **<ipv6_gateway_address>** 替换为网关的 IPv6 地址。

10. 配置在检查过程中使用的 DHCP 范围：

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
...
```

- 将< **ipv6_address_inspection_** start> 替换为检查过程中要使用的网络范围内的 IPv6 地址。
- 将 **<ipv6_address_inspection_end** > 替换为检查过程中要使用的网络范围末尾的 IPv6 地址。

注意

这个范围不得与 **dhcp_start** 和 **dhcp_end** 定义的范围重叠，但必须位于同一 IP 子网中。

11. 为子网配置 IPv6 名称服务器：

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
dns_nameservers = <ipv6_dns>
```

将 **<ipv6_dns>** 替换为特定于子网的 DNS 名称服务器。

4.7. 配置 UNDERCLOUD 网络接口

在 **undercloud.conf** 文件中包含自定义网络配置，以使用特定的网络功能安装 undercloud。例如，一些接口可能没有 DHCP。在这种情况下，您必须在 **undercloud.conf** 文件中禁用这些接口的 DHCP，以便 **os-net-config** 可在 undercloud 安装过程中应用配置。

步骤

1. 登录 undercloud 主机。
2. 创建新文件 **undercloud-os-net-config.yaml**，并包含所需的网络配置。
   有关更多信息，请参阅高级 Overcloud 自定义指南中的网络接口参考。

   下面是一个示例：

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

要为特定接口创建网络绑定，请使用以下示例：

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
    - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
      name: nic2
    - type: interface
      name: nic3
```

3. *将 **undercloud-os-net-config.yaml** 文件的路径包含在 **undercloud.conf** 文件的 **net_config_override** 参数中：*

```
[DEFAULT]
...
net_config_override=undercloud-os-net-config.yaml
...
```

注意

*director 使用您在 **net_config_override** 参数中包含的文件作为模板来生成 **/etc/os-net-config/config.yaml** 文件。**os-net-config** 管理您在模板中定义的接口，因此您必须在此文件中执行所有 undercloud 网络接口自定义。*

4. 安装 undercloud。

验证

- 在 undercloud 安装成功完成后，验证 **/etc/os-net-net-config/config.yaml** 文件是否包含相关配置：

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
    - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

4.8. 安装 DIRECTOR

完成以下步骤以安装 director 并执行一些基本安装后任务。

步骤

1. 运行以下命令，以在 undercloud 上安装 director：

```
[stack@director ~]$ openstack undercloud install
```

此命令会启动 director 配置脚本。director 安装附加软件包，根据 **undercloud.conf** 中的配置配置其服务，并启动所有 RHOSP 服务容器。这个脚本会需要一些时间来完成。

此脚本会生成两个文件：

- ***undercloud-passwords.conf*** - director 服务的所有密码列表。
- ***stackrc*** - 用来访问 director 命令行工具的一组初始变量。

2. 确认 RHOSP 服务容器正在运行：

```
[stack@director ~]$ sudo podman ps -a --format "{{.Names}} {{.Status}}"
```

以下命令输出显示 RHOSP 服务容器正在运行(**Up**)：

```
memcached Up 3 hours (healthy)
haproxy Up 3 hours
rabbitmq Up 3 hours (healthy)
mysql Up 3 hours (healthy)
iscsid Up 3 hours (healthy)
keystone Up 3 hours (healthy)
keystone_cron Up 3 hours (healthy)
```

```
neutron_api Up 3 hours (healthy)
logrotate_crond Up 3 hours (healthy)
neutron_dhcp Up 3 hours (healthy)
neutron_l3_agent Up 3 hours (healthy)
neutron_ovs_agent Up 3 hours (healthy)
ironic_api Up 3 hours (healthy)
ironic_conductor Up 3 hours (healthy)
ironic_neutron_agent Up 3 hours (healthy)
ironic_pxe_tftp Up 3 hours (healthy)
ironic_pxe_http Up 3 hours (unhealthy)
ironic_inspector Up 3 hours (healthy)
ironic_inspector_dnsmasq Up 3 hours (healthy)
neutron-dnsmasq-qdhcp-30d628e6-45e6-499d-8003-28c0bc066487 Up 3 hours
...
```

3. 运行以下命令初始化 **stack** 用户来使用命令行工具：

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

提示现在指示 OpenStack 命令对 undercloud 进行验证并执行；

```
(undercloud) [stack@director ~]$
```

director 的安装已完成。您现在可以使用 director 命令行工具了。

4.9. 为 OVERCLOUD 配置 CPU 架构

Red Hat OpenStack Platform (RHOSP)默认将 overcloud 的 CPU 架构配置为 x86_64。您还可以在 POWER (ppc64le)硬件上部署 overcloud Compute 节点。对于 Compute 节点集群，可使用相同的架构，或使用 x86_64 和 ppc64le 系统的组合。

注意

undercloud、Controller 节点、Ceph Storage 节点及所有其他系统仅在 x86_64 硬件上才受支持。

4.9.1. 将 POWER (ppc64le)配置为 overcloud 的单一 CPU 架构

overcloud 上 Compute 节点的默认 CPU 架构是 x86_64。要在 POWER (ppc64le)硬件上部署 overcloud Compute 节点，您可以将架构更改为 ppc64le。

流程

1. 在 **undercloud.conf** 文件中禁用 iPXE：

```
[DEFAULT]
ipxe_enabled = False
```

注意

对于 RHOSP 16.2.1 及更早版本，此配置会导致部署中的任何 x86_64 节点在 PXE/legacy 模式中启动。要为 overcloud 配置多个 CPU 架构，请参阅 配置多个 CPU 架构 overcloud。

2. 安装 undercloud：

```
[stack@director ~]$ openstack undercloud install
```

有关更多信息，请参阅在 undercloud 上安装 director。

3. 等待安装脚本完成。

4. 获取并上传 overcloud 节点的镜像。有关更多信息，请参阅 获取 overcloud 节点的镜像。

4.9.2. 配置多个 CPU 架构 overcloud

对于 RHOSP 16.2.2 及更新的版本，您可以将 undercloud 配置为在架构包含 POWER (ppc64le) 和 x86_64 UEFI 节点时支持 PXE 和 iPXE 引导模式。

注意

当您的架构包含 POWER (ppc64le) 节点时，RHOSP 16.2.1 及更早版本只支持 PXE 引导。

流程

1. 在 **undercloud.conf** 文件中启用 iPXE：

```
[DEFAULT]
ipxe_enabled = True
```

2. 为 undercloud、**undercloud_noIronicIPXEEnabled.yaml** 创建自定义环境文件。

3. 要将默认裸机置备服务(ironic) iPXE 设置更改为 **PXE**，请在 **undercloud_noIronicIPXEEnabled.yaml** 中添加以下配置：

```
parameter_defaults:
  IronicIPXEEnabled: false
  IronicInspectorIPXEEnabled: true
```

4. 如果您的架构包含 **ppc64le** 节点，请在 **undercloud_noIronicIPXEEnabled.yaml** 中添加以下配置来禁用引导超时：

```
parameter_defaults:
  ExtraConfig:
    ironic::config::ironic_config:
      ipmi/disable_boot_timeout:
        value: 'false'
```

5. 在 **undercloud.conf** 文件中包含自定义环境文件：

```
[DEFAULT]
...
custom_env_files = undercloud_noIronicIPXEEnabled.yaml
```

6. 安装 undercloud：

```
[stack@director ~]$ openstack undercloud install
```

有关更多信息，请参阅在 undercloud 上安装 director。

7. 等待安装脚本完成。
8. 注册 overcloud 节点：

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```

有关注册 overcloud 节点的更多信息，请参阅为 overcloud 注册节点。

9. 等待节点完成注册和配置。
10. 确认 director 已成功注册节点：

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

11. 检查每个注册的节点的现有功能：

```
$ openstack baremetal node show <node> -f json -c properties | jq -r .properties.capabilities
```

12. 通过将 **boot_mode:uefi** 添加到节点的现有容量中，将每个注册节点的引导模式设置为 **uefi**：

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
<capability_n>" <node>
```

- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<capability_1&** gt; 以及所有功能（直到 & **lt;capability_n&** gt;）替换为您在第 6 步中获得的每个功能。

13. 获取并上传 overcloud 节点的镜像。如需更多信息，请参阅 多个 CPU 架构 overcloud 镜像。
14. 为每个节点设置引导模式：

- 对于旧的/PXE：

```
$ openstack baremetal node set --boot-interface pxe <node_name>
```

- 对于 iPXE：

```
$ openstack baremetal node set --boot-interface ipxe <node_name>
```

4.9.3. 在多架构 overcloud 中使用 Ceph Storage

在多架构云中配置对外部 Ceph 的访问时，请将 **CephAnsiblePlaybook** 参数设置为 **/usr/share/ceph-ansible/site.yml.sample**，同时也包括您的客户端密钥和其他 Ceph 相关参数。

例如：

```
parameter_defaults:
  CephAnsiblePlaybook: /usr/share/ceph-ansible/site.yml.sample
  CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
  CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
  CephExternalMonHost: 172.16.1.7, 172.16.1.8
```

4.9.4. 在多架构 overcloud 中使用可组合服务

以下服务通常是组成 Controller 节点的一部分，在自定义角色中使用它们当前还是技术预览：

- Block Storage 服务 (cinder)
- Image 服务 (glance)
- Identity 服务 (keystone)
- Networking 服务 (neutron)
- Object Storage 服务 (swift)

注意

红帽不支持技术预览功能。

有关可组合服务的更多信息，请参阅高级 Overcloud 自定义指南中的可组合服务和自定义角色。通过以下示例了解如何将列出的服务从 Controller 节点移至专用 ppc64le 节点：

```
(undercloud) [stack@director ~]$ rsync -a /usr/share/openstack-tripleo-heat-templates/. ~/templates
(undercloud) [stack@director ~]$ cd ~/templates/roles
(undercloud) [stack@director roles]$ cat <<EO_TEMPLATE >ControllerPPC64LE.yaml
###############################################################################
# Role: ControllerPPC64LE                                                     #
###############################################################################
- name: ControllerPPC64LE
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
  networks:
    - External
    - InternalApi
    - Storage
    - StorageMgmt
    - Tenant
  # For systems with both IPv4 and IPv6, you may specify a gateway network for
  # each, such as ['ControlPlane', 'External']
  default_route_networks: ['External']
  HostnameFormatDefault: '%stackname%-controllerppc64le-%index%'
  ImageDefault: ppc64le-overcloud-full
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackendDellPs
```

```
- OS::TripleO::Services::CinderBackendDellSc
- OS::TripleO::Services::CinderBackendDellEMCUnity
- OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
- OS::TripleO::Services::CinderBackendDellEMCVNX
- OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
- OS::TripleO::Services::CinderBackendNetApp
- OS::TripleO::Services::CinderBackendScaleIO
- OS::TripleO::Services::CinderBackendVRTSHyperScale
- OS::TripleO::Services::CinderBackup
- OS::TripleO::Services::CinderHPELeftHandISCSI
- OS::TripleO::Services::CinderScheduler
- OS::TripleO::Services::CinderVolume
- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Docker
- OS::TripleO::Services::Fluentd
- OS::TripleO::Services::GlanceApi
- OS::TripleO::Services::GlanceRegistry
- OS::TripleO::Services::Ipsec
- OS::TripleO::Services::Iscsid
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Keystone
- OS::TripleO::Services::LoginDefs
- OS::TripleO::Services::MySQLClient
- OS::TripleO::Services::NeutronApi
- OS::TripleO::Services::NeutronBgpVpnApi
- OS::TripleO::Services::NeutronSfcApi
- OS::TripleO::Services::NeutronCorePlugin
- OS::TripleO::Services::NeutronDhcpAgent
- OS::TripleO::Services::NeutronL2gwAgent
- OS::TripleO::Services::NeutronL2gwApi
- OS::TripleO::Services::NeutronL3Agent
- OS::TripleO::Services::NeutronLbaasv2Agent
- OS::TripleO::Services::NeutronLbaasv2Api
- OS::TripleO::Services::NeutronLinuxbridgeAgent
- OS::TripleO::Services::NeutronMetadataAgent
- OS::TripleO::Services::NeutronML2FujitsuCfab
- OS::TripleO::Services::NeutronML2FujitsuFossw
- OS::TripleO::Services::NeutronOvsAgent
- OS::TripleO::Services::NeutronVppAgent
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCrond
- OS::TripleO::Services::OpenDaylightOvs
- OS::TripleO::Services::Rhsm
- OS::TripleO::Services::RsyslogSidecar
- OS::TripleO::Services::Securetty
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::SkydiveAgent
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Sshd
- OS::TripleO::Services::SwiftProxy
- OS::TripleO::Services::SwiftDispersion
- OS::TripleO::Services::SwiftRingBuilder
- OS::TripleO::Services::SwiftStorage
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
```

```
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
EO_TEMPLATE
(undercloud) [stack@director roles]$ sed -i~ -e '/OS::TripleO::Services::\
(Cinder\|Glance\|Swift\|Keystone\|Neutron\)/d' Controller.yaml
(undercloud) [stack@director roles]$ cd ../
(undercloud) [stack@director templates]$ openstack overcloud roles generate \
    --roles-path roles -o roles_data.yaml \
    Controller Compute ComputePPC64LE ControllerPPC64LE BlockStorage ObjectStorage
CephStorage
```

4.10. 为 OVERCLOUD 节点获取镜像

director 需要几个磁盘镜像用于置备 overcloud 节点：

- 一个内省内核和 ramdisk 用于通过 PXE 引导进行裸机系统内省。
- 一个部署内核和 ramdisk 用于系统置备和部署。
- overcloud 内核、ramdisk 和完整镜像形成 director 写入节点硬盘的基本 overcloud 系统。

您可以根据 CPU 架构获取并安装所需的镜像。在不想运行其他 Red Hat OpenStack Platform (RHOSP)服务或消耗您的一项订阅授权时，您还可以获取并安装基本镜像来置备裸机操作系统。

4.10.1. 单个 CPU 架构 overcloud 镜像

您的 Red Hat OpenStack Platform (RHOSP)安装包括了为 director 提供以下 overcloud 镜像的软件包：

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

这些镜像是使用默认 CPU 架构 x86-64 部署 overcloud 所必需的。将这些镜像导入到 director 也会在 director PXE 服务器上安装内省镜像。

流程

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

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

3. 安装 **rhosp-director-images** 和 **rhosp-director-images-ipa-x86_64** 软件包：

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images rhosp-director-
images-ipa-x86_64
```

4. 在 **stack** 用户的主目录(**/home/stack/ images**)中创建 images 目录。

```
(undercloud) [stack@director ~]$ mkdir /home/stack/images
```

5. 将镜像存档提取到 **images** 目录中：

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-16.2.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-16.2.tar; do tar -xvf $i; done
```

6. 将镜像导入 director：

```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/
```

7. 验证镜像是否已上传：

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

8. 验证 director 是否已将内省 PXE 镜像复制到 **/var/lib/ironic/httpboot** ：

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/httpboot
total 417296
-rwxr-xr-x. 1 root  root    6639920 Jan 29 14:48 agent.kernel
-rw-r--r--. 1 root  root  420656424 Jan 29 14:48 agent.ramdisk
-rw-r--r--. 1 42422 42422       758 Jan 29 14:29 boot.ipxe
-rw-r--r--. 1 42422 42422       488 Jan 29 14:16 inspector.ipxe
```

4.10.2. 多个 CPU 架构 overcloud 镜像

您的 Red Hat OpenStack Platform (RHOSP) 安装包括为您提供使用默认 CPU 架构 x86-64 部署 overcloud 所需的以下镜像的软件包：

- ***overcloud-full***
- ***overcloud-full-initrd***
- ***overcloud-full-vmlinuz***

您的 RHOSP 安装还包括为以下镜像提供使用 POWER (ppc64le) CPU 架构部署 overcloud 所需的软件包：

- ***ppc64le-overcloud-full***

将这些镜像导入到 director 也会在 director PXE 服务器上安装内省镜像。

流程

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

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

3. 安装 **rhosp-director-images-all** 软件包：

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-all
```

4. 将存档解包到特定于架构的目录中，该目录位于 **stack** 用户的主目录下的 **images** 目录 (**/home/stack/images**) 中：

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do mkdir $arch ; done
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do for i in
/usr/share/rhosp-director-images/overcloud-full-latest-16.1-${arch}.tar /usr/share/rhosp-
director-images/ironic-python-agent-latest-16.1-${arch}.tar ; do tar -C $arch -xf $i ; done ;
done
```

5. 将镜像导入 director：

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --http-boot
/var/lib/ironic/tftpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --image-type ironic-python-agent --
http-boot /var/lib/ironic/httpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64/ --architecture x86_64 --http-boot /var/lib/ironic/tftpboot
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64 --architecture x86_64 --image-type ironic-python-agent --http-boot
/var/lib/ironic/httpboot
```

6. 验证镜像是否已上传：

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+--------+
| ID                                   | Name                    | Status |
+--------------------------------------+-------------------------+--------+
| 6a6096ba-8f79-4343-b77c-4349f7b94960 | overcloud-full          | active |
| de2a1bde-9351-40d2-bbd7-7ce9d6eb50d8 | overcloud-full-initrd   | active |
| 67073533-dd2a-4a95-8e8b-0f108f031092 | overcloud-full-vmlinuz  | active |
| f0fedcd0-3f28-4b44-9c88-619419007a03 | ppc64le-overcloud-full  | active |
+--------------------------------------+-------------------------+--------+
```

7. 验证 director 是否已将内省 PXE 镜像复制到 **/var/lib/ironic/tftpboot** ：

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/tftpboot
/var/lib/ironic/tftpboot/ppc64le/
/var/lib/ironic/tftpboot:
total 422624
-rwxr-xr-x. 1 root  root     6385968 Aug  8 19:35 agent.kernel
```

```
-rw-r--r--. 1 root  root   425530268 Aug  8 19:35 agent.ramdisk
-rwxr--r--. 1 42422 42422      20832 Aug  8 02:08 chain.c32
-rwxr--r--. 1 42422 42422     715584 Aug  8 02:06 ipxe.efi
-rw-r--r--. 1 root  root          22 Aug  8 02:06 map-file
drwxr-xr-x. 2 42422 42422         62 Aug  8 19:34 ppc64le
-rwxr--r--. 1 42422 42422      26826 Aug  8 02:08 pxelinux.0
drwxr-xr-x. 2 42422 42422         21 Aug  8 02:06 pxelinux.cfg
-rwxr--r--. 1 42422 42422      69631 Aug  8 02:06 undionly.kpxe

/var/lib/ironic/tftpboot/ppc64le/:
total 457204
-rwxr-xr-x. 1 root  root  19858896 Aug  8 19:34 agent.kernel
-rw-r--r--. 1 root  root  448311235 Aug  8 19:34 agent.ramdisk
-rw-r--r--. 1 42422 42422       336 Aug  8 02:06 default
```

4.10.3. 在容器镜像上启用多个 CPU 架构

如果您的 Red Hat OpenStack Platform (RHOSP) 部署有多个 CPU 架构，并且它使用容器镜像，您必须更新容器镜像以启用多个架构。

流程

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

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

3. 在 **containers-prepare-parameter.yaml** 文件中添加额外的架构以启用多个架构：

```
parameter_defaults:
  ContainerImageRegistryLogin: true
  AdditionalArchitectures: [<list_of_architectures>]
  ContainerImagePrepare:
  - push_destination: true
    ...
```

将 **<list_of_architectures>** 替换为 overcloud 环境支持的架构列表，例如 **[ppc64le]**。

4. 准备并上传容器：

```
$ openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

4.10.4. 最小 overcloud 镜像

如果您不希望运行其他 Red Hat OpenStack Platform (RHOSP) 服务或消耗您的一项订阅授权，您可以使用 **overcloud-minimal** 镜像来置备裸机操作系统。

您的 RHOSP 安装包括 **overcloud-minimal** 软件包，它为您提供了 director 的以下 overcloud 镜像：

- **overcloud-minimal**
- **overcloud-minimal-initrd**

- **overcloud-minimal-vmlinuz**

注意

默认的 **overcloud-full.qcow2** 镜像是一种平面分区镜像。但是，您仍可以导入和使用完整的磁盘镜像。更多信息请参阅 第 24 章 创建完整磁盘镜像。

步骤

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

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

3. 安装 **overcloud-minimal** 软件包：

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-minimal
```

4. 将镜像存档解包到 **stack** 用户主目录 (**/home/stack/images**) 中的 **images** 目录中：

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ tar xf /usr/share/rhosp-director-images/overcloud-minimal-latest-16.2.tar
```

5. 将镜像导入 director：

```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/ --image-type os --os-image-name overcloud-minimal.qcow2
```

6. 验证镜像是否已上传：

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+
| ID                                   | Name                    |
+--------------------------------------+-------------------------+
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full          |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd   |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz  |
| 32cf6771-b5df-4498-8f02-c3bd8bb93fdd | overcloud-minimal       |
| 600035af-dbbb-4985-8b24-a4e9da149ae5 | overcloud-minimal-initrd |
| d45b0071-8006-472b-bbcc-458899e0d801 | overcloud-minimal-vmlinuz |
+--------------------------------------+-------------------------+
```

4.11. 为 CONTROL PLANE 设置名称服务器

如果您希望 overcloud 解析外部主机名，如 **cdn.redhat.com**，请在 overcloud 节点上设置名称服务器。对于没有进行网络隔离的标准 overcloud，名称服务器会使用 undercloud 的 control plane 子网来定义。完成以下过程为环境定义名称服务器。

步骤

1. 查找 **stackrc** 文件，以启用 director 命令行工具：

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

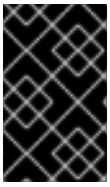
2. 为 **ctlplane-subnet** 子网设置名称服务器：

```
(undercloud) [stack@director images]$ openstack subnet set --dns-nameserver
[nameserver1-ip] --dns-nameserver [nameserver2-ip] ctlplane-subnet
```

请为每一个名称服务器使用 **--dns-nameserver** 选项。

3. 查看子网来验证名称解析服务器：

```
(undercloud) [stack@director images]$ openstack subnet show ctlplane-subnet
+-------------------+-----------------------------------------------+
| Field             | Value                                         |
+-------------------+-----------------------------------------------+
| ...               |                                               |
| dns_nameservers   | 8.8.8.8                                       |
| ...               |                                               |
+-------------------+-----------------------------------------------+
```



重要

如果要将服务流量隔离到单独的网络，overcloud 节点必须使用网络环境文件中的 **DnsServer** 参数。您还必须将 control plane 名称服务器和 **DnsServers** 参数设置为相同的 DNS 服务器。

4.12. 更新 UNDERCLOUD 配置

如果需要更改 undercloud 配置以适应新要求，则可在安装后更改 undercloud 配置，请编辑相关配置文件并重新运行 **openstack undercloud install** 命令。

步骤

1. 修改 undercloud 配置文件。例如，编辑 **undercloud.conf** 文件并将 **idrac** 硬件类型添加到已启用硬件类型列表中：

```
enabled_hardware_types = ipmi,redfish,idrac
```

2. 运行 **openstack undercloud install** 命令以使用新更改刷新 undercloud：

```
[stack@director ~]$ openstack undercloud install
```

等待命令运行完成。

3. 初始化 **stack** 用户以使用命令行工具：

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

提示现在指示 OpenStack 命令对 undercloud 进行验证并执行：

```
(undercloud) [stack@director ~]$
```

4. 确认 director 已应用新配置。在此示例中，检查已启用硬件类型列表：

```
(undercloud) [stack@director ~]$ openstack baremetal driver list
+---------------------+---------------------+
| Supported driver(s) | Active host(s)      |
+---------------------+---------------------+
| idrac               | director.example.com |
| ipmi                | director.example.com |
| redfish             | director.example.com |
+---------------------+---------------------+
```

undercloud 重新配置完成。

4.13. UNDERCLOUD 容器 REGISTRY

Red Hat Enterprise Linux 8.4 不再包括 **docker-distribution** 软件包，该软件包安装了 Docker Registry v2。为了保持兼容性和相同的功能级别，director 安装使用名为 **image-serve** 的 vhost 创建 Apache Web 服务器以提供 registry。该 registry 也使用禁用了 SSL 的端口 8787/TCP。基于 Apache 的 registry 未容器化，这意味着您必需运行以下命令以重启 registry：

```
$ sudo systemctl restart httpd
```

您可以在以下位置找到容器 registry 日志：

- /var/log/httpd/image_serve_access.log
- /var/log/httpd/image_serve_error.log。

镜像内容来自 **/var/lib/image-serve**。此位置使用特定目录布局和 **apache** 配置来实施 registry REST API 的拉取功能。

基于 Apache 的 registry 不支持 **podman push** 或 **buildah push** 命令，这意味着您无法使用传统方法推送容器镜像。要在部署过程中修改镜像，请使用容器准备工作流，如 **ContainerImagePrepare** 参数。要管理容器镜像，请使用容器管理命令：

OpenStack tripleo 容器镜像列表

列出 registry 上存储的所有镜像。

OpenStack tripleo 容器镜像显示

显示 registry 上特定镜像的元数据。

OpenStack tripleo container image push

将镜像从远程 registry 推送到 undercloud registry。

OpenStack tripleo container image delete

从 registry 中删除镜像。

第 5 章 安装 UNDERCLOUD MINION

您可以部署额外的 undercloud minion 以在多个主机之间扩展 OpenStack Platform director 服务，这有助于在部署大型 overcloud 时提高性能。该功能为可选。

重要

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

5.1. UNDERCLOUD MINION

undercloud minion 在单独的主机上提供额外的 **heat-engine** 和 **ironic-conductor** 服务。这些额外的服务支持 undercloud 的编配和置备操作。跨多个主机的 undercloud 操作分布提供了更多资源来运行 overcloud 部署，这可实现更快和更大型的部署。

5.2. UNDERCLOUD MINION 要求

服务要求

在 undercloud minion 上扩展的 **heat-engine** 和 **ironic-conductor** 服务使用一组 worker。每个 worker 执行特定于该服务的操作。多个 worker 提供同步操作。minion 上 worker 的默认数量是 minion 主机的 CPU 线程总数的一半在本实例中，线程总数是 CPU 内核数乘以超线程值。例如，如果您的 minion 有 16 个线程的 CPU，则 minion 会默认为每个服务生成 8 个 worker。minion 还默认使用一组最小值和最大值：

服务	最小值	最大值
heat-engine	4	24
ironic-conductor	2	12

undercloud minion 具有以下最小 CPU 和内存要求：

- 支持 Intel 64 或 AMD64 CPU 扩展的 8 线程 64 位 x86 处理器。此处理器可为每个 undercloud 服务提供 4 个 worker。
- 至少 16 GB RAM。

要使用更多 worker，请按照每个 CPU 线程需要 2 GB RAM 的比例增加 undercloud 上的 vCPU 和内存数。例如，一台有 48 个线程的计算机必须具有 96 GB RAM。这可支持 24 个 **heat-engine** worker 和 12 个 **ironic-conductor** worker。

容器镜像要求

undercloud minon 不托管内部容器镜像 registry。因此，您必须将 minion 配置为使用以下方法之一来获取容器镜像：

- 直接从 Red Hat Container Image Registry (**registry.redhat.io**) 拉取镜像。
- 拉取您在 Red Hat Satellite Server 上托管的镜像。

对于这两种方法，您必须将 **push_destination: false** 设置为 **containers-prepare-parameter.yaml** 文件中的 **ContainerImagePrepare** heat 参数的一部分。

5.3. 准备一个 MINION

在安装 minion 前，您必须在主机上完成一些基本配置。

- 一个用来执行命令的非 root 用户。
- 一个可解析的主机名
- 一个红帽订阅
- 用于准备镜像和安装 minion 的命令行工具

步骤

1. 以 **root** 用户身份登录到 minion 主机。
2. 创建 **stack** 用户：

```
[root@minion ~]# useradd stack
```

3. 为 **stack** 用户设置密码：

```
[root@minion ~]# passwd stack
```

4. 进行以下操作，以使用户在使用 **sudo** 时无需输入密码：

```
[root@minion ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a /etc/sudoers.d/stack
[root@minion ~]# chmod 0440 /etc/sudoers.d/stack
```

5. 切换到新的 **stack** 用户：

```
[root@minion ~]# su - stack
[stack@minion ~]$
```

6. 检查 minion 的基础和完整主机名：

```
[stack@minion ~]$ hostname
[stack@minion ~]$ hostname -f
```

如果上述命令没有显示正确的完全限定主机名或报告错误，则使用 **hostnamectl** 设置主机名：

```
[stack@minion ~]$ sudo hostnamectl set-hostname minion.example.com
[stack@minion ~]$ sudo hostnamectl set-hostname --transient minion.example.com
```

7. 编辑 **/etc/hosts** 文件并包括系统主机名的条目。例如：如果系统名是 **minion.example.com**，并使用 IP 地址 **10.0.0.1**，请在 **/etc/hosts** 文件中添加以下行：

```
10.0.0.1  minion.example.com manager
```

8. 在红帽 Content Delivery Network 或 Red Hat Satellite 注册您的系统。例如，运行以下命令在 Content Delivery Network 中注册系统。根据提示输入您的客户门户网站用户名和密码：

```
[stack@minion ~]$ sudo subscription-manager register
```

9. 查找 Red Hat OpenStack Platform (RHOSP) director 的权利池 ID：

```
[stack@minion ~]$ sudo subscription-manager list --available --all --matches="Red Hat
OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
                     Red Hat Virtualization
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             Valid-Pool-Number-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

10. 找到 **Pool ID** 值并附加 Red Hat OpenStack Platform 16.2 权利：

```
[stack@minion ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

11. 禁用所有默认的仓库，然后启用 Red Hat Enterprise Linux 仓库：

```
[stack@minion ~]$ sudo subscription-manager repos --disable=*
[stack@minion ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-
eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-
highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-
16.2-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

这些仓库包括了安装 minion 所需的软件包。

12. 在系统上执行更新，确保您有最新的基本系统软件包：

```
[stack@minion ~]$ sudo dnf update -y
[stack@minion ~]$ sudo reboot
```

13. 安装用于安装和配置 minion 的命令行工具：

```
[stack@minion ~]$ sudo dnf install -y python3-tripleoclient
```

5.4. 将 UNDERCLOUD 配置文件复制到 MINION 中

minion 需要 undercloud 中的一些配置文件，以便 minion 安装可以配置 minion 服务，并将它们注册到 director：

- ***tripleo-undercloud-outputs.yaml***
- ***tripleo-undercloud-passwords.yaml***

步骤

1. 以 **stack** 用户身份登录 undercloud。
2. 将 undercloud 中的文件复制到 minion：

```
$ scp ~/tripleo-undercloud-outputs.yaml ~/tripleo-undercloud-passwords.yaml
stack@<minion-host>:~/.
```

   - 将 **<minion-host>** 替换为 minion 的主机名或 IP 地址。

5.5. 复制 UNDERCLOUD 证书颁发机构

如果 undercloud 使用 SSL/TLS 进行端点加密，则 minion 主机必须包含签发 undercloud SSL/TLS 证书的证书颁发机构。根据 undercloud 配置，这个证书颁发机构是以下其中之一：

- 在 minion 主机上预载入证书的外部证书颁发机构。不需要操作。
- director 生成的自签名证书颁发机构（**/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**）。将这个文件复制到 minion 主机，并将该文件作为 minion 主机的可信证书颁发机构的一部分。这个过程使用这个文件作为示例。
- 使用 OpenSSL 创建的自定义自签名证书颁发机构。本文档中的示例将该文件称为 **ca.crt.pem**。将这个文件复制到 minion 主机，并将该文件作为 minion 主机的可信证书颁发机构的一部分。

步骤

1. 以 **root** 用户身份登录到 minion 主机。
2. 将证书颁发机构文件从 undercloud 复制到 minion：

```
[root@minion ~]# scp \
    root@<undercloud-host>:/etc/pki/ca-trust/source/anchors/cm-local-ca.pem \
    /etc/pki/ca-trust/source/anchors/undercloud-ca.pem
```

   - 将 **<undercloud-host>** 替换为 undercloud 的主机名或 IP 地址。

3. 为 minion 主机更新可信证书颁发机构：

```
[root@minion ~]# update-ca-trust enable
[root@minion ~]# update-ca-trust extract
```

5.6. 配置 MINION

minion 安装过程需要 **minion.conf** 配置文件中的某些设置，minion 从 **stack** 用户的主目录中读取该文件。完成以下步骤，使用默认模板作为基础进行配置。

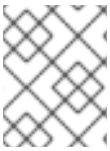
步骤

1. 以 **stack** 用户身份登录到 minion 主机。
2. 将默认模板复制到 **stack** 用户的主目录：

```
[stack@minion ~]$ cp \
 /usr/share/python-tripleoclient/minion.conf.sample \
 ~/minion.conf
```

3. 编辑 **minion.conf** 文件。这个文件包含用于配置 minion 的设置。如果忽略或注释掉某个参数，minion 安装将使用默认值。查看以下推荐参数：

   - **minion_hostname**，将其设置为 minion 的主机名。
   - **minion_local_interface**，将其设置为通过置备网络连接到 undercloud 的接口。
   - **minion_local_ip**，将其设置为置备网络上的空闲 IP 地址。
   - **minion_nameservers**，将其设置为 DNS 名称服务器以便 minion 可以解析主机名。
   - **enable_ironic_conductor**，定义是否启用 **ironic-conductor** 服务。
   - **enable_heat_engine**，定义是否启用 **heat-engine** 服务。



注意

默认 **minion.conf** 文件只在 minion 上启用 **heat-engine** 服务。要启用 **ironic-conductor** 服务，请将 **enable_ironic_conductor** 参数设置为 **true**。

5.7. MINION 配置参数

以下列表包含用于配置 **minion.conf** 文件的参数的相关信息。将所有参数保留在相关部分内以避免出错。

默认值

以下参数会在 **minion.conf** 文件的 **[DEFAULT]** 部分中进行定义：

cleanup

清理临时文件。将此参数设置为 **False**，可在该命令运行后保留部署期间使用的临时文件。这可用于调试生成的文件或者确定是否发生了错误。

container_cli

用于容器管理的 CLI 工具。将此参数设置为 **podman**。Red Hat Enterprise Linux 8.4 仅支持 **podman**。

container_healthcheck_disabled

禁用容器化服务运行状况检查。红帽建议您启用运行状况检查，并将此选项设置为 **false**。

container_images_file

含有容器镜像信息的 Heat 环境文件。此文件可能包含以下条目：

- 所有需要的容器镜像的参数
- **ContainerImagePrepare** 参数（用于推动必要的镜像准备）。通常，含有此参数的文件被命名为 **containers-prepare-parameter.yaml**。

container_insecure_registries

供 **podman** 使用的不安全 registry 列表。如果您想从其他来源（如私有容器 registry）拉取镜像，则使用此参数。在大多数情况下，如果在 Satellite 中注册了 minion，**podman** 就有从红帽容器目录或 Satellite 服务器拉取容器镜像的证书。

container_registry_mirror

配置的 **podman** 使用的可选 **registry-mirror**。

custom_env_files

要添加到 minion 安装中的其他环境文件。

deployment_user

安装 minion 的用户。如果此参数保留为不设置，则使用当前的默认用户 **stack**。

enable_heat_engine

定义是否在 minion 上安装 heat 引擎。默认值是 **true**。

enable_ironic_conductor

定义是否在 minion 上安装 ironic conductor 服务。默认值为：**false**。将此值设置为 **true** 以启用 ironic conductor 服务。

heat_container_image

要使用的 heat 容器镜像的 URL。请保留为不设置。

heat_native

使用原生 heat 模板。请保留为 **true**。

hieradata_override

在 director 上配置 Puppet hieradata 的 **hieradata** 覆盖文件的路径，为 **minion.conf** 参数外的服务提供自定义配置。如果设置此参数，minion 安装会将此文件复制到 **/etc/puppet/hieradata** 目录并将其设为层次结构中的第一个文件。

minion_debug

将此值设置为 **true**，为 minion 服务启用 **DEBUG** 日志级别。

minion_enable_selinux

在部署期间启用或禁用 SELinux。除非调试问题，否则强烈建议保留此值设为 **true**。

minion_enable_validations

在 minion 上启用验证服务。

minion_hostname

定义 minion 的完全限定主机名。如果设置，minion 安装将配置所有系统主机名设置。如果保留未设置，minion 将使用当前的主机名，但您必须相应地配置所有系统主机名设置。

minion_local_interface

在 undercloud 上为 Provisioning NIC 选择的接口。这也是 minion 用于 DHCP 和 PXE 引导服务的设备。把这个项的值改为您选择的设备。使用 **ip addr** 命令可以查看连接了哪些设备。以下是一个 **ip addr** 命令的结果输出示例：

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic eth0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

在这个例子中，External NIC 使用 **eth0**，Provisioning NIC 使用 **eth1**（当前没有被配置）。在这种情况下，把 **local_interface** 设置为 **eth1**。配置脚本会把这个接口附加到一个自定义的网桥（由 **inspection_interface** 参数定义）上。

minion_local_ip

在 undercloud 上为 Provisioning NIC 定义的 IP 地址。这也是 minion 用于 DHCP 和 PXE 引导服务的 IP 地址。除非 Provisioning 网络需要使用其他子网（如默认 IP 地址与环境中的现有 IP 地址或子网冲突）保留默认值 **192.168.24.1/24**。

minion_local_mtu

要用于 **local_interface** 的最大传输单元 (MTU)。对于 minion 不要超过 1500。

minion_log_file

要存储 minion 安装和升级日志的日志文件的路径。默认情况下，日志文件是主目录中的 **install-minion.log**。例如：**/home/stack/install-minion.log**。

minion_nameservers

用于 minion 主机名解析的 DNS 名称服务器列表。

minion_ntp_servers

帮助同步 minion 日期和时间的网络时间协议服务器列表。

minion_password_file

包含 minion 用于连接 undercloud 服务的密码的文件。将此参数设置为从 undercloud 中复制的 **tripleo-undercloud-passwords.yaml** 文件。

minion_service_certificate

用于 OpenStack SSL/TLS 通信的证书的位置和文件名。理想的情况是从一个信任的证书认证机构获得这个证书。否则，生成自己的自签名证书。

minion_timezone

minion 的主机时区。如果未指定时区，minion 将使用现有时区配置。

minion_undercloud_output_file

包含 minion 可以用来连接到 undercloud 服务的 undercloud 配置信息的文件。将此参数设置为从 undercloud 中复制的 **tripleo-undercloud-outputs.yaml** 文件。

net_config_override

网络配置覆盖模板的路径。如果设置此参数，minion 将使用 JSON 格式的模板以使用 **os-net-config** 配置网络，并忽略 **minion.conf** 中设置的网络参数。如需示例，请参阅 **/usr/share/python-tripleoclient/minion.conf.sample**。

networks_file

覆盖用于 **heat** 的网络文件。

output_dir

输出状态目录、处理的 heat 模板和 Ansible 部署文件。

roles_file

要用来覆盖 minion 安装的默认角色文件的角色文件。强烈建议您保留为不设置，以便 minion 安装使用默认的角色文件。

templates

要覆盖的 heat 模板文件。

5.8. 安装 MINION

完成以下步骤来安装 minion。

步骤

1. 以 **stack** 用户身份登录到 minion 主机。

2. 运行以下命令来安装 minion：

```
[stack@minion ~]$ openstack undercloud minion install
```

该命令启动 minion 的配置脚本，安装附加软件包并根据 **minion.conf** 文件中的配置来配置 minion 服务。这个脚本会需要一些时间来完成。

5.9. 验证 MINION 安装

完成以下步骤确认成功安装 minion。

步骤

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

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

3. 如果您在 minion 上启用了 heat engine 服务，请验证 minion 中的 **heat-engine** 服务会出现在 undercloud 服务列表中：

```
[stack@director ~]$ $ openstack orchestration service list
```

命令输出显示一个表，其中包含 undercloud 和所有 minion 的 **heat-engine** worker。

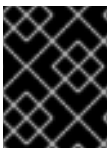
4. 如果您在 minion 上启用了 ironic-conductor 服务，请验证 minion 中的 **ironic-conductor** 服务出现在 undercloud 服务列表中：

```
[stack@director ~]$ $ openstack baremetal conductor list
```

命令输出显示一个表，其中包含 undercloud 和所有 minion 的 **ironic-conductor** 服务。

第 6 章 规划您的 OVERCLOUD

以下部分包含在规划 Red Hat OpenStack Platform（RHOSP）环境的各个方面的指导信息。这包括定义节点角色、规划您的网络拓扑结构和存储。



重要

部署 overcloud 节点后，请勿重命名这些节点。在部署后重命名节点会导致实例管理问题。

6.1. 节点角色

director 包含以下默认节点类型用于构建 overcloud：

Controller

提供用于控制环境的关键服务。它包括仪表板服务 (horizon)、认证服务 (keystone)、镜像存储服务 (glance)、联网服务 (neutron)、编配服务 (heat) 以及高可用性服务。Red Hat OpenStack Platform（RHOSP）环境需要三个 Controller 节点以实现高可用生产级环境。

注意

将只有一个 Controller 节点的环境用于测试目的，不应该用于生产环境。不支持由两个 Controller 节点或由三个以上 Controller 节点组成的环境。

计算

用作虚拟机监控程序并包含在环境中运行虚拟机所需的处理能力的物理服务器。基本 RHOSP 环境需要至少一个 Compute 节点。

Ceph Storage

提供 Red Hat Ceph Storage 的一个主机。额外的 Ceph Storage 主机可以在一个集群中扩展。这个部署角色是可选的。

Swift Storage

为 OpenStack Object Storage (swift) 服务提供外部对象存储的主机。这个部署角色是可选的。

下表包含一些不同 overcloud 的示例并为每个场景定义节点类型。

表 6.1. 场景的节点部署角色

	Controller	计算	Ceph 存储	Swift Storage	总计
小型 overcloud	3	1	-	-	4
中型 overcloud	3	3	-	-	6
带有额外对象存储的中型 overcloud	3	3	-	3	9
带有 Ceph Storage 集群的中型 overcloud	3	3	3	-	9

此外，还需思考是否要将各个服务划分成不同的自定义角色。有关可组合角色架构的更多信息，请参阅高级 Overcloud 自定义指南中的"可组合服务和自定义角色"。

表 6.2. 用于概念验证部署的节点部署角色

	undercloud	Controller	计算	Ceph Storage	总计
概念验证	1	1	1	1	4

警告

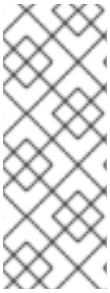
Red Hat OpenStack Platform 在第 2 天运维中维护一个可正常运行的 Ceph Storage 集群。因此，在少于三个 MON 或三个存储节点的部署中，无法进行某些第 2 天运维，如 Ceph Storage 集群的升级或次要更新。如果使用单个 Controller 节点或单个 Ceph Storage 节点，则第 2 天运维将失败。

6.2. OVERCLOUD 网络

规划环境的网络拓扑和子网非常重要，它可以确保映射角色和服务，以使其可以正确地相互通信。Red Hat OpenStack Platform (RHOSP) 使用 Openstack Networking (neutron) 服务，此服务可自主运行，并可管理基于软件的网络、静态和浮动 IP 地址以及 DHCP。

默认情况下，director 配置节点以使用 **Provisioning/Control Plane** 获得连接。不过，可以将网络流量隔离到一系列的可组合网络，供您自定义和分配服务。

在典型的 RHOSP 安装中，网络类型的数量通常会超过物理网络链路的数量。为了可以把所有网络都连接到正确的主机，overcloud 使用 VLAN 标记（VLAN tagging）来为每个接口提供多个网络。大多数网络都是隔离的子网，但有些网络需要第 3 层网关来提供路由用于互联网访问或基础架构网络连接。如果使用 VLAN 来隔离网络流量类型，则必需使用支持 802.1Q 标准的交换机来提供 tagged VLAN。



注意

您可以使用 VLAN 创建项目（租户）网络。您可以为特殊使用网络创建 Geneve 或 VXLAN 隧道，而无需消耗项目 VLAN。红帽建议您使用 Geneve 或 VXLAN 部署项目网络隧道，即使您打算在禁用隧道的 neutron VLAN 模式中部署 overcloud。如果您使用 Geneve 或 VXLAN 部署项目网络隧道，您仍然可以更新您的环境，以使用隧道网络作为实用程序网络或虚拟化网络。可以将 Geneve 或 VXLAN 功能添加到带有项目 VLAN 的部署中，但无法将项目 VLAN 添加到现有的 overcloud 中，而不会造成中断。

director 还包括一组模板，可用于使用隔离的可组合网络配置 NIC。以下配置为默认配置：

- 单 NIC 配置 - 一个 NIC 在原生 VLAN 中用于 Provisioning 网络，并用于 tagged VLAN（使用子网处理不同的 overcloud 网络类型）。

- *绑定 NIC 配置 - 一个 NIC 在原生 VLAN 中用于 Provisioning 网络，tagged VLAN 绑定中的两个 NIC 用于不同的 overcloud 网络类型。*
- *多 NIC 配置 - 每个 NIC 都使用一个子网来分别处理 overcloud 中不同的网络类型。*

您也可以创建自己的模板来映射特定的 NIC 配置。

在考虑网络配置时，以下详细信息也很重要：

- *在 overcloud 创建过程中，在所有 overcloud 机器间使用同一个名称来代表 NIC。理想情况下，您应该在每个 overcloud 节点上对每个相关的网络都使用相同的 NIC 来避免混淆。例如，Provisioning 网络使用主（primary）NIC，OpenStack 服务使用从（secondary）NIC。*
- *设置所有 overcloud 系统为从 Provisioning NIC 进行 PXE 引导，并禁用通过外部 NIC 和系统上的任何其他 NIC 进行 PXE 引导。另外，还需要确保 Provisioning NIC 在 PXE 引导设置中位于引导顺序的最上面（在硬盘和 CD/DVD 驱动器之前）。*
- *所有 overcloud 裸机系统都需要一个受支持的电源管理接口，如智能平台管理接口 (IPMI)，以便 director 能够控制每个节点的电源管理。*
- *请记录下每个 overcloud 系统的以下信息：Provisioning NIC 的 MAC 地址、IPMI NIC 的 IP 地址、IPMI 用户名和 IPMI 密码。稍后配置 overcloud 节点时，这些信息很有用。*
- *如果一个实例必须可以被外部互联网访问，则需要从公共网络中分配一个浮动 IP 地址，并把浮动 IP 和这个实例相关联。这个实例仍然会保留它的私人 IP，但网络流量可以通过 NAT 到达浮动 IP 地址。请注意，一个浮动 IP 地址只能分配给一个接口，而不能分配给多个私人 IP 地址。但是，浮动 IP 地址只保留给一个租户使用，这意味着租户可以根据需要将浮动 IP 地址与特定实例相关联或取消关联。此配置会使您的基础架构暴露于外部互联网，您必须使用了适当的安全保护措施。*
- *为了减少 Open vSwitch 中出现网络环路的风险，只能有一个接口或一个绑定作为给定网桥的成员。如果需要多个绑定或接口，可以配置多个网桥。*
- *红帽建议使用 DNS 主机名解析，以便您的 overcloud 节点能够连接到外部服务，如 Red Hat Content Delivery Network 和网络时间服务器。*
- *红帽建议将置备接口、外部接口和任何浮动 IP 接口保留在 1500 的默认 MTU。否则可能会发生连接问题。这是因为路由器通常无法跨第 3 层边界转发巨型帧。*

注意

如果您使用 Red Hat Virtualization (RHV)，则可以对 overcloud control plane 进行虚拟化。如需更多信息，请参阅创建虚拟化 control planes。

6.3. OVERCLOUD 存储

注意

在使用任何驱动程序的后端 cinder-volume 或后端类型的虚拟客户机实例上使用 LVM 会导致性能、卷可见性和可用性以及数据损坏问题。使用 LVM 筛选减少可见性、可用性和数据损坏问题。有关更多信息，请参阅存储指南中的第 2 节块存储和卷和 KCS 文章 3213311"在 cinder 卷上使用 LVM 会将数据公开给计算主机。"

director 包括面向 overcloud 环境的不同存储选项：

Ceph Storage 节点

director 使用 Red Hat Ceph Storage 创建一组可扩展存储节点。overcloud 针对以下存储类型使用这些节点：

- **镜像** - Image 服务 (glance) 管理虚拟机的镜像。镜像是不可变的，OpenStack 将镜像视为二进制 blob 并相应地进行下载。您可以使用 Image 服务 (glance) 在 Ceph Block Device 中存储镜像。
- **卷** - OpenStack 使用 Block Storage 服务 (cinder) 管理卷。Block Storage 服务 (cinder) 卷是块设备。OpenStack 使用卷来引导虚拟机，或将卷附加到正在运行的虚拟机上。您可以使用 Block Storage 服务通过镜像的 copy-on-write clone 来引导虚拟机。
- **文件系统 -** Openstack 使用 Shared File Systems 服务 (manila) 管理共享文件系统。共享由文件系统支持。您可以使用 manila 来管理由 CephFS 文件系统（数据保存在 Ceph Storage 节点上）支持的共享。
- **客户机磁盘** - 客户机磁盘就是客户机操作系统磁盘。默认情况下，当您使用 Compute 服务 (nova) 引导虚拟机时，虚拟机磁盘会在虚拟机监控程序的文件系统上显示为一个文件（通常在 **/var/lib/nova/instances/<uuid>/** 下）。Ceph 中的每个虚拟机都可以在不使用 Block Storage 服务 (cinder) 的情况下引导。因此，您可以使用实时迁移过程轻松执行维护操作。此外，如果您的虚拟机监控程序出现故障，它还可以方便地触发 **nova evacuate**，并在其他位置运行虚拟机。

重要

有关支持的镜像格式的信息，请参阅实例和管理镜像指南中的镜像服务。

有关 Ceph Storage 的更多信息，请参阅 Red Hat Ceph Storage 架构指南 。

Swift 存储节点

director 会创建外部对象存储节点。当您需要扩展或替换 overcloud 环境中的 Controller 节点，同时需要在一个高可用性集群外保留对象存储时，这将非常有用。

6.4. OVERCLOUD 安全性

OpenStack 平台环境的安全性在一定程度上取决于网络的安全性。遵循网络环境中的良好安全原则，确保正确控制网络访问：

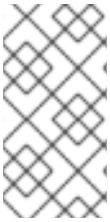
- 使用网络分段缓解网络移动并隔离敏感数据。扁平网络的安全性要低得多。
- 限制对服务和端口的访问。
- 强制执行正确的防火墙规则并使用密码。
- 确保启用 SELinux。

有关保护系统安全的更多信息，请参阅以下红帽指南：

- Red Hat Enterprise Linux 8 的安全加固
- 对 Red Hat Enterprise Linux 8 使用 SELinux

6.5. OVERCLOUD 高可用性

要部署高度可用的 overcloud，director 将配置多个 Controller、Compute 和 Storage 节点，并以单一集群的形式协同工作。出现节点故障时，根据故障的节点类型来触发自动隔离和重新生成流程。有关 overcloud 高可用性架构和服务的更多信息，请参阅高可用性部署和使用。



注意

不支持在不使用 STONITH 的情况下部署高可用性 overcloud。您必须在高可用性 overcloud 中为作为 Pacemaker 集群一部分的每个节点配置 STONITH 设备。有关 STONITH 和 Pacemaker 的信息，请参阅红帽高可用性集群中的隔离和 RHEL 高可用性集群的支持策略。

您也可以通过 director 为 Compute 实例配置高可用性 (Instance HA)。使用此高可用性机制，可在节点出现故障时在 Compute 节点上自动清空并重新生成实例。对 Instance HA 的要求与一般的 overcloud 要求相同，但必须执行一些附加步骤以准备好环境进行部署。有关 Instance HA 和安装说明的更多信息，请参阅 Compute 实例的高可用性指南。

6.6. CONTROLLER 节点要求

Controller 节点在 Red Hat OpenStack Platform 环境中托管核心服务，如 Dashboard (horizon)、后端数据库服务器、Identity 服务 (keystone) 和高可用性服务。

处理器

支持 Intel 64 或 AMD64 CPU 扩展的 64 位 x86 处理器。

内存

最小内存为 32 GB。不过，建议根据 vCPU 数量（CPU 内核数乘以超线程值）来决定内存大小。使用以下计算确定 RAM 要求：

- **控制器 RAM 最小值计算：**
  - 每个 vCPU 使用 1.5 GB 内存。例如，拥有 48 个 vCPU 的计算机应当具有 72 GB RAM。
- **控制器 RAM 建议值计算：**
  - 每个 vCPU 使用 3 GB 内存。例如，拥有 48 个 vCPU 的计算机应当具有 144 GB RAM。

有关衡量内存要求的更多信息，请参阅红帽客户门户网站上的"高可用性控制器的 Red Hat OpenStack Platform 硬件要求"。

磁盘存储和布局

如果 Object Storage 服务 (swift) 不在 Controller 节点上运行，则需要最小 50 GB 的存储。但是，Telemetry 和 Object Storage 服务都安装在 Controller 上，且二者均配置为使用根磁盘。这些默认值适合部署在商用硬件上构建的小型 overcloud。这些环境通常用于概念验证和测试环境。您可以使用这些默认布局，只需最少的规划即可部署 overcloud，但它们只能提供很低的工作负载容量和性能。然而在企业环境中，默认布局可能造成很大的瓶颈。这是因为 Telemetry 会不断地访问存储资源，导致磁盘 I/O 使用率很高，从而严重影响所有其他 Controller 服务的性能。在这种环境中，必须规划 overcloud 并进行相应的配置。

红帽为 Telemetry 和 Object Storage 提供了一些配置建议方案。有关更多信息，请参阅特定 Red Hat OpenStack Platform 服务的部署建议。

网络接口卡

最少两个 1 Gbps 网络接口卡。对绑定的接口使用额外的网络接口卡，或代理标记的 VLAN 流量。

电源管理

每个 Controller 节点在服务器的主板上都要有一个受支持的电源管理接口，如智能平台管理接口 (IPMI) 功能。

虚拟化支持

红帽仅在 Red Hat Virtualization 平台上支持虚拟化 Controller 节点。如需更多信息，请参阅创建虚拟化 control planes。

6.7. COMPUTE 节点要求

Compute 节点负责在启动虚拟机实例后运行虚拟机实例。Compute 节点需要支持硬件虚拟化的裸机系统。Compute 节点还必须有足够的内存和磁盘空间来支持其托管的虚拟机实例的要求。

处理器

- 支持 Intel 64 或 AMD64 CPU 扩展并启用了 AMD-V 或 Intel VT 硬件虚拟扩展的 64 位 x86 处理器。我们推荐所使用的处理器最少有 4 个内核。
- IBM POWER 8 处理器。

内存

主机操作系统最少需要 6 GB RAM，以及满足以下注意事项的额外内存：

- 添加您要提供给虚拟机实例的额外内存。
- 添加额外内存以在主机上运行特殊功能或其他资源，如附加内核模块、虚拟交换机、监控解决方案和其他额外的后台任务。
- 如果要使用非统一内存访问 (NUMA)，红帽建议每个 CPU 插槽节点使用 8GB，或如果您有 256 GB 的物理 RAM，则建议每插槽节点使用 16GB。
- 至少配置 4 GB 交换空间。

磁盘空间

最少具有 50GB 可用磁盘空间。

网络接口卡

最少一个 1 Gbps 网络接口卡。对绑定的接口使用额外的网络接口卡，或代理标记的 VLAN 流量。

电源管理

每个 Compute 节点在服务器的主板上都要有一个受支持的电源管理接口，如智能平台管理接口 (IPMI) 功能。

6.8. CEPH STORAGE 节点要求

如果使用 Red Hat OpenStack Platform (RHOSP) director 创建 Red Hat Ceph Storage 节点，则还有额外的要求。

有关如何为 Ceph Storage 节点选择处理器、内存、网络接口卡 (NIC) 和磁盘布局的信息，请参阅 Red Hat Ceph Storage 硬件指南中的 Red Hat Ceph Storage 的硬件选择建议 。

每个 Ceph Storage 节点还需要在服务器的主板上有一个受支持的电源管理接口，如智能平台管理接口 (IPMI) 功能。

注意

RHOSP director 使用 **ceph-ansible**，不支持在 Ceph Storage 节点的根磁盘上安装 OSD。这意味着所支持的每个 Ceph Storage 节点需要至少两个磁盘。

Ceph Storage 节点和 RHEL 兼容性

- RHEL 8.4 支持 RHOSP 16.2。在升级到 RHOSP 16.1 及之后的版本前，请参阅红帽知识库文章 Red Hat Ceph Storage: 支持的配置。

Red Hat Ceph Storage 兼容性

- RHOSP 16.2 支持 Red Hat Ceph Storage 4。

放置组 (PG)

- Ceph Storage 使用放置组 (PG) 大规模推动动态高效的对象跟踪。如果 OSD 出现故障或集群进行重新平衡，Ceph 可移动或复制放置组及其内容，这意味着 Ceph Storage 集群可以有效地重新平衡并恢复。

- director 创建的默认放置组数量并非始终最佳，因此一定要根据要求计算正确的放置组数量。您可以使用放置组计算器计算正确的数量。要使用 PG 计算器，请输入每个服务的预计存储使用量（百分比表示），以及 Ceph 集群的其他属性，如 OSD 数量。计算器返回每个池的最佳 PG 数量。有关更多信息，请参阅每个池计算器的放置组 (PG)。

- 自动扩展是管理放置组的替代方法。借助自动扩展功能，您可以将每个服务的预期 Ceph Storage 要求设置为百分比而不是特定数量的放置组。Ceph 根据集群的使用方式自动扩展放置组。有关更多信息，请参阅 Red Hat Ceph Storage 策略指南 中的自动扩展放置组。

处理器

- 支持 Intel 64 或 AMD64 CPU 扩展的 64 位 x86 处理器。

网络接口卡

- 最少一个 1 Gbps 网络接口卡 (NIC)，但红帽建议您在生产环境中至少使用两个 NIC。对绑定的接口使用额外的 NIC，或代理标记的 VLAN 流量。为存储节点使用 10 Gbps 接口，特别是所创建的 Red Hat OpenStack Platform (RHOSP) 环境需要处理大量流量时。

电源管理

- 每个 Controller 节点在服务器的主板上都要有一个受支持的电源管理接口，如智能平台管理接口 (IPMI) 功能。

有关使用 Ceph Storage 集群安装 overcloud 的更多信息，请参阅使用容器化 Red Hat Ceph 部署 Overcloud 指南。

6.9. OBJECT STORAGE 节点要求

Object Storage 节点提供 overcloud 的对象存储层。Object Storage 代理安装在 Controller 节点上。存储层需要每个节点上有多个磁盘的裸机节点。

处理器

支持 Intel 64 或 AMD64 CPU 扩展的 64 位 x86 处理器。

内存

内存要求取决于存储空间大小。每 1TB 硬盘空间需要至少 1GB 内存。要获得最佳性能，建议每 1 TB 硬盘空间使用 2 GB，特别是用于文件小于 100GB 的工作负载。

磁盘空间

存储要求取决于工作负载需要的容量。建议使用 SSD 驱动器来存储帐户和容器数据。帐户和容器数据与对象的容量比约为 1%。例如，对于每 100TB 硬盘容量，请提供 1TB 容量来存储帐户和容器数据。不过，这还取决于所存储数据的类型。如果主要是要存储小对象，则提供更多 SSD 空间。而对于大对象（视频和备份等），可提供较少的 SSD 空间。

磁盘配置

建议的节点配置需要类似以下示例的磁盘布局：

- **/dev/sda** - 根磁盘。director 把主 overcloud 镜像复制到该磁盘。
- **/dev/sdb** - 用于帐户数据。
- **/dev/sdc** - 用于容器数据。
- **/dev/sdd 及后续** - 对象服务器磁盘。可以根据您的存储需要使用多个磁盘。

网络接口卡

最少两个 1 Gbps 网络接口卡。对绑定的接口使用额外的网络接口卡，或代理标记的 VLAN 流量。

电源管理

每个 Controller 节点在服务器的主板上都要有一个受支持的电源管理接口，如智能平台管理接口 (IPMI) 功能。

6.10. OVERCLOUD 软件仓库

Red Hat OpenStack Platform (RHOSP) 16.2 在 Red Hat Enterprise Linux (RHEL) 8.4 上运行。因此，您必须将这些软件仓库中的内容锁定到相应的 RHEL 版本。

注意

- 如果使用 Red Hat Satellite 同步软件仓库，您可以启用 RHEL 软件仓库的特定版本。但是，无论您选择的版本，存储库标签仍保持不变。例如，如果您启用 BaseOS 存储库的 8.4 版本，存储库名称包括您启用的特定版本，但存储库标签仍然是 **rhel-8-for-x86_64-baseos-eus-rpms**。
- 不再需要 **advanced-virt-for-rhel-8-x86_64-rpms** 和 **advanced-virt-for-rhel-8-x86_64-eus-rpms** 存储库。要禁用这些软件仓库，请参阅 OSP 16.2 不再需要红帽知识库解决方案 advanced-virt-for-rhel-8-x86_64-rpms。

警告

此处指定的软件仓库之外的任何软件仓库都不受支持。除非建议，请勿启用下表中所列之外的任何其他产品或软件仓库，否则可能会遇到依赖软件包问题。请勿启用 Extra Packages for Enterprise Linux (EPEL)。

Controller 节点软件仓库

下表列出了用于 overcloud 中 Controller 节点的核心软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 系统的基本操作系统仓库。
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-eus-rpms**	包含 RHOSP 依赖项。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	RHEL 的高可用性工具。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	RHEL 的 Ansible Engine。用于提供最新版本的 Ansible。
Red Hat OpenStack Platform 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-x86_64-rpms**	核心 RHOSP 存储库。
Red Hat Fast datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	用于 RHEL 8 的 Red Hat Ceph Storage 4 的工具。

Compute 和 ComputeHCI 节点软件仓库

下表列出了 overcloud 中 Compute 和 ComputeHCI 节点的核心软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-baseos-eus-rpms**	x86_64 系统的基本操作系统仓库。
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-eus-rpms**	包含 RHOSP 依赖项。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	RHEL 的高可用性工具。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	RHEL 的 Ansible Engine。用于提供最新版本的 Ansible。

名称	软件仓库	要求说明
Red Hat OpenStack Platform 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-x86_64-rpms**	核心 RHOSP 存储库。
Red Hat Fast datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	用于 RHEL 8 的 Red Hat Ceph Storage 4 的工具。

Real Time Compute 软件仓库

下表列出了 Real Time Compute (RTC) 功能的软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - Real Time (RPMs)	**rhel-8-for-x86_64-rt-rpms**	Real Time KVM (RT-KVM) 的软件仓库。包含用于启用实时内核的软件包。对 RT-KVM 为目标的所有 Compute 节点启用此软件仓库。注意：您需要单独订阅 **Red Hat OpenStack Platform for Real Time** SKU，才能访问此软件仓库。
Red Hat Enterprise Linux 8 for x86_64 - Real Time for NFV (RPMs)	**rhel-8-for-x86_64-nfv-rpms**	适用于 NFV 的实时 KVM (RT-KVM) 的软件仓库。包含用于启用实时内核的软件包。对以 RT-KVM 为目标的所有 NFV Compute 节点启用此软件仓库。注意：您需要单独订阅 **Red Hat OpenStack Platform for Real Time** SKU，才能访问此软件仓库。

Ceph Storage 节点软件仓库

下表列出了用于 overcloud 的与 Ceph Storage 有关的软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)	**rhel-8-for-x86_64-baseos-rpms**	x86_64 系统的基本操作系统仓库。
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-rpms**	包含 RHOSP 依赖项。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Extended Update Support (EUS)	**rhel-8-for-x86_64-highavailability-eus-rpms**	RHEL 的高可用性工具。注意：如果您将 **overcloud-full** 镜像用于 Ceph Storage 角色，则必须启用此存储库。Ceph Storage 角色应使用 **overcloud-minimal** 镜像，该镜像不需要此存储库。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	RHEL 的 Ansible Engine。用于提供最新版本的 Ansible。
Red Hat OpenStack Platform 16.2 Director Deployment Tools for RHEL 8 x86_64 (RPMs)	**openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms**	帮助 director 配置 Ceph Storage 节点的软件包。此软件仓库包含在单机 Ceph Storage 订阅中。如果您使用组合的 OpenStack Platform 和 Ceph Storage 订阅，请使用 **openstack-16.2-for-rhel-8-x86_64-rpms** 存储库。
Red Hat OpenStack Platform 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-x86_64-rpms**	帮助 director 配置 Ceph Storage 节点的软件包。此软件仓库包含在 OpenStack Platform 和 Ceph Storage 订阅中。如果使用的是单机 Ceph Storage 订阅，则请使用 **openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms** 软件仓库。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	提供节点与 Ceph Storage 集群进行通信的工具。
Red Hat Fast datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。如果您在 Ceph Storage 节点上使用 OVS，请将此存储库添加到网络接口配置(NIC)模板中。

IBM POWER 软件仓库

下表列出了 POWER PC 架构上 RHOSP 的软件仓库。使用这些软件仓库来替代核心软件仓库中的同等库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux for IBM Power，little endian - BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le 系统的基本操作系统软件仓库。

名称	软件仓库	要求说明
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	包含 RHOSP 依赖项。
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-rpms**	RHEL 的高可用性工具。用于 Controller 节点的高可用性功能。
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	为 OpenStack Platform 提供 Open vSwitch (OVS) 软件包。
Red Hat Ansible Engine 2.9 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.9-for-rhel-8-ppc64le-rpms**	RHEL 的 Ansible Engine。用于提供最新版本的 Ansible。
Red Hat OpenStack Platform 16.2 for RHEL 8 (RPMs)	**openstack-16.2-for-rhel-8-ppc64le-rpms**	ppc64le 系统的核心 RHOSP 存储库。

6.11. 置备方法

您可以使用三种主要方法为 Red Hat OpenStack Platform 环境置备节点：

使用 director 置备

*Red Hat OpenStack Platform director 是标准置备方法。在这种情况下，**openstack overcloud deploy** 命令同时执行部署的置备和配置。有关标准置备和部署方法的更多信息，请参阅 第 7 章 配置基本 overcloud。*

使用 OpenStack Bare Metal (ironic) 服务置备

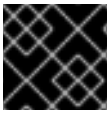
在这种情况下，您可以将标准 director 部署的置备和配置阶段分成两个不同的过程。如果要降低标准 director 部署带来的一些风险，并更有效地识别故障点，这非常有用。有关这种情况的详情请参考 第 8 章 部署 overcloud 前置备裸机节点。

重要

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

使用外部工具置备

*在这种情况下，director 会使用外部工具控制预置备节点上的 overcloud 配置。如果您要在没有电源管理控制的情况下创建 overcloud，使用具有 DHCP/PXE 引导限制的网络，或者使用具有不依赖 QCOW2 **overcloud-full** 镜像的自定义分区布局的节点，这非常有用。此情况不使用 OpenStack Compute (nova)、OpenStack Bare Metal (ironic) 或者 OpenStack Image (glance) 服务来管理节点。有关这种情况的详情请参考 第 9 章 使用预置备节点配置基本 overcloud。*



重要

您无法将预置备节点与 director 置备的节点合并。

第 7 章 配置基本 OVERCLOUD

overcloud 基本配置中不含任何自定义功能。要配置基本的 Red Hat OpenStack Platform (RHOSP) 环境，您必须执行以下任务：

- 为您的 overcloud 注册裸机节点。
- 为 director 提供裸机节点硬件清单。
- 使用与指定角色匹配的资源类标记每个裸机节点。

提示

您可以在此基本 overcloud 中添加高级配置选项，并根据您的规格进行自定义。有关更多信息，请参阅高级 Overcloud 自定义。

7.1. 为 OVERCLOUD 注册节点

director 需要一个节点定义模板，用于指定节点的硬件和电源管理详情。您可以使用 JSON 格式、node **.json** 或 YAML 格式创建此模板 **nodes.yaml**。

流程

1. 创建名为 **nodes.json** 或 **nodes.yaml** 的模板，它将列出您的节点。使用以下 JSON 和 YAML 模板示例了解如何创建节点定义模板的结构：

   示例 JSON 模板

```
{
  "nodes": [{
    "name": "node01",
    "ports": [{
      "address": "aa:aa:aa:aa:aa:aa",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.205"
  },
  {
    "name": "node02",
    "ports": [{
      "address": "bb:bb:bb:bb:bb:bb",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
```

```
        "port_id": "p0"
      }
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.206"
  }]
}
```

示例 YAML 模板

```
nodes:
  - name: "node01"
    ports:
      - address: "aa:aa:aa:aa:aa:aa"
        physical_network: ctlplane
        local_link_connection:
          switch_id: "52:54:00:00:00:00"
          port_id: p0
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.205"
  - name: "node02"
    ports:
      - address: "bb:bb:bb:bb:bb:bb"
        physical_network: ctlplane
        local_link_connection:
          switch_id: "52:54:00:00:00:00"
          port_id: p0
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.206"
```

此模板包含以下属性：

name

节点的逻辑名称。

ports

访问特定 IPMI 设备的端口。您可以定义以下可选端口属性：

- ***地址***：节点上网络接口的 MAC 地址。对于每个系统的 Provisioning NIC，只使用 MAC 地址。
- ***physical_network***：连接到 Provisioning NIC 的物理网络。
- ***local_link_connection***：如果您使用 IPv6 置备，且 LLDP 在内省过程中没有正确填充本地链路连接，您必须在 local_link_connection 参数中包含带有 **switch_id** 和 **port_id** 字段的虚拟数据。有关如何包含虚拟数据的更多信息，请参阅使用 director 内省来收集裸机节点硬件信息。

cpu

节点上的 CPU 数量。（可选）

memory

以 MB 为单位的内存大小。（可选）

disk

以 GB 为单位的硬盘的大小。（可选）

arch

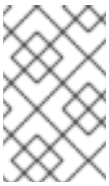
系统架构。（可选）

重要

在构建多架构云时，**arch** 键是必需的，用于区分使用 **x86_64** 和 **ppc64le** 架构的节点。

pm_type

要使用的电源管理驱动程序。此示例使用 IPMI 驱动程序 (**ipmi**)。



注意

IPMI 是首选的受支持电源管理驱动程序。有关支持的电源管理类型及其选项的更多信息，请参阅 电源管理驱动程序。如果这些电源管理驱动程序不能正常工作，请将 IPMI 用于电源管理。

pm_user; pm_password

IPMI 的用户名和密码。

pm_addr

IPMI 设备的 IP 地址。

2. 创建模板后，运行以下命令验证格式和语法：

```
$ source ~/stackrc
(undercloud)$ openstack overcloud node import --validate-only ~/nodes.json
```

重要

您还必须为多架构节点包含 **--http-boot /var/lib/ironic/tftpboot/** 选项。

3. 将文件保存到 **stack** 用户的主目录(**/home/stack/nodes.json**)。

4. 将模板导入到 director，将每个节点从模板注册到 director：

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```

注意

如果使用 UEFI 引导模式，还必须在每个节点上设置引导模式。如果您在不设置 UEFI 引导模式的情况下内省节点，节点以旧模式引导。如需更多信息，请参阅将引导模式设置为 UEFI 引导模式。

5. 等待节点完成注册和配置。完成后，确认 director 已成功注册节点：

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

7.2. 创建裸机节点硬件的清单

director 需要 Red Hat OpenStack Platform (RHOSP) 部署中节点的硬件清单进行配置集标记、基准测试和手动根磁盘分配。

您可以使用以下方法之一向 director 提供硬件清单：

- 自动：您可以使用 director 的内省过程，从每个节点收集硬件信息。这个过程在每个节点上启动内省代理。内省代理从节点收集硬件数据并将数据送回 director。director 将硬件数据存储在 undercloud 节点上运行的对象存储服务(swift)中。
- 手动：您可以手动配置每个裸机的基本硬件清单。此清单存储在裸机置备服务(ironic)中，用于管理和部署裸机机器。

注意

如果您的 overcloud 使用 derived **_params.yaml**，则需要使用 director 的自动内省过程，这需要存在内省数据。如需有关 derived **_params.yaml** 的更多信息，请参阅 工作流和派生参数。

director 自动内省流程比设置裸机置备服务端口的手动方法提供以下优点：

- 内省记录了硬件信息中的所有连接端口，包括在 **nodes.yaml** 中尚未配置 PXE 引导的端口。
- 如果属性可以使用 LLDP 发现，内省会为每个端口设置 **local_link_connection** 属性。使用手动方法时，您必须在注册节点时为每个端口配置 **local_link_connection**。
- 在部署 spine-and-leaf 或 DCN 架构时，内省为裸机置备服务端口设置 **physical_network** 属性。

7.2.1. 使用 director 内省来收集裸机节点硬件信息

将物理机注册为裸机节点后，您可以使用 director 内省自动添加其硬件详情并为每个以太网 MAC 地址创建端口。

提示

作为自动内省的替代选择，您可以手动为 director 提供裸机节点的硬件信息。如需更多信息，请参阅 手动配置裸机节点硬件信息。

先决条件

- 您已为 overcloud 注册了裸机节点。

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. 运行 pre-introspection 验证组来检查内省要求：

```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 查看验证报告的结果。
5. 可选：查看特定验证的详细输出：

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

- 将 **<**validation> 替换为您要查看报告的特定验证 UUID。

重要

FAILED 验证不会阻止您部署或运行 Red Hat OpenStack Platform。但是，**FAILED** 验证可能会显示生产环境中潜在的问题。

6. 检查每个节点的属性。您可以检查所有节点的硬件属性或特定节点：
   - 检查所有节点的硬件属性：

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

     - 使用 **--all-manageable** 选项仅内省处于受管理状态的节点。在此示例中，所有节点都处于受管理状态。
     - 使用 **--provide** 选项在内省后将所有节点重置为 **available** 状态。

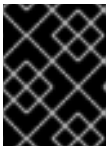
   - 检查特定节点的硬件属性：

```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

     - 使用 **--provide** 选项，在内省后将所有指定节点重置为 **available** 状态。
     - 将 **<node1>**, **[node2]**, 及所有节点（直到 **[noden]**）替换为您要内省的每个节点的 UUID。

7. 在一个单独的终端窗口中监控内省进度日志：

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```



重要

确保内省进程运行完。内省通常需要 15 分钟时间用于裸机节点。但是，不正确的内省网络可能会导致它花费更长的时间，这可能会导致内省失败。

8. 可选：如果您已将 undercloud 配置为通过 IPv6 进行裸机置备，那么您还需要检查 LLDP 是否已为裸机置备服务(ironic)端口设置了 **local_link_connection** ：

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

- 如果裸机节点上的端口的 Local Link Connection 字段为空，则必须使用虚拟数据手动填充 **local_link_connection** 值。以下示例将虚拟交换机 ID 设置为 **52:54:00:00:00:00**，并将虚拟端口 ID 设置为 **p0** ：

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- 验证 "Local Link Connection" 字段是否包含虚拟数据：

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

内省完成后，所有节点都会变为 **available** 状态。

7.2.2. 手动配置裸机节点硬件信息

将物理机注册为裸机节点后，您可以手动添加其硬件详情，并为每个以太网 MAC 地址创建裸机端口。在部署 overcloud 前，必须至少创建一个裸机端口。

提示

作为手动内省的替代选择，您可以使用自动 director 内省流程来收集裸机节点的硬件信息。如需更多信息，请参阅使用 director 内省来收集裸机节点硬件信息。

先决条件

- 您已为 overcloud 注册了裸机节点。
- 您已为 node **.json** 中注册的节点上的每个端口配置了 **local_link_connection**。有关更多信息，请参阅为 overcloud 注册节点。

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. 通过在节点的功能中添加 **boot_option':'local** 来将每个注册的节点的引导选项设置为 **local**：

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

- 将 **<node>** 替换为裸机节点的 ID。

4. 指定部署内核并为节点驱动程序部署 ramdisk：

```
(undercloud)$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<kernel_file>** 替换为 **.kernel** 镜像的路径，例如 **file:///var/lib/ironic/httpboot/agent.kernel**。
- 将 **<initramfs_file>** 替换为 **.initramfs** 镜像的路径，例如 **file:///var/lib/ironic/httpboot/agent.ramdisk**。

5. 更新节点属性以匹配节点上的硬件规格：

```
(undercloud)$ openstack baremetal node set <node> \
  --property cpus=<cpu> \
  --property memory_mb=<ram> \
  --property local_gb=<disk> \
  --property cpu_arch=<arch>
```

- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<cpu>** 替换为 CPU 数量。
- 将 &**lt;ram&**gt; 替换为 RAM （以 MB 为单位）。
- 将 **<disk>** 替换为磁盘大小（以 GB 为单位）。
- 将 &**lt;arch&**gt; 替换为构架类型。

6. 可选：为每个节点指定 IPMI 密码套件：

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

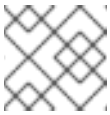
- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<version>** 替换为节点上要使用的密码套件版本。设置为以下有效值之一：
  - **3** - 节点使用带有 SHA1 密码套件的 AES-128。
  - **17** - 节点使用带有 SHA256 密码套件的 AES-128。

7. 可选：如果您有多个磁盘，请设置 root 设备提示来告知部署 ramdisk 哪个磁盘用于部署：

```
(undercloud)$ openstack baremetal node set <node> \
  --property root_device='{"<property>": "<value>"}'
```

- 将 **<node>** 替换为裸机节点的 ID。

- 将 **<property > 和 <value** > 替换为您要用于部署的磁盘的详情，如 **root_device='{"size": "128"}'**
  RHOSP 支持以下属性：
  - **model**（字符串）：设备识别码。
  - **vendor**（字符串）：设备厂商。
  - **serial**（字符串）：磁盘序列号。
  - **hctl**（字符串）：SCSI 的 Host:Channel:Target:Lun。
  - **size**（整数）：设备的大小（以 GB 为单位）。
  - **wwn**（字符串）：唯一的存储 ID。
  - **wwn_with_extension**（字符串）：唯一存储 ID 附加厂商扩展名。
  - **wwn_vendor_extension**（字符串）：唯一厂商存储标识符。
  - **rotational**（布尔值）：旋转磁盘设备为 true (HDD)，否则为 false (SSD)。
  - **name** （字符串）：设备名称，例如：/dev/sdb1 仅将此属性用于具有持久名称的设备。



注意

如果您指定多个属性，该设备必须与所有这些属性匹配。

8. 通过在 provisioning 网络中创建带有 NIC 的 MAC 地址的端口来通知节点网卡的裸机置备服务：

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- 将 **<node_uuid>** 替换为裸机节点的唯一 ID。
- 将 **<mac_address** > 替换为用于 PXE 引导的 NIC 的 MAC 地址。

9. 验证节点的配置：

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| Interface  | Result | Reason                                     |
+------------+--------+--------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| console    | None   | not supported                              |
| deploy     | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| inspect    | None   | not supported                              |
| management | True   |                                            |
| network    | True   |                                            |
```

```
| power      | True   |                                      |
| raid       | True   |                                      |
| storage    | True   |                                      |
+------------+--------+--------------------------------------+
```

*验证输出 **Result** 表示以下内容：*

- ***false** ：接口失败验证。如果提供的原因包括缺少 **instance_info** 参数 **[\'ramdisk', \'kernel', \'image_source']**，这可能是因为计算服务在部署过程开始时填充缺少的参数，因此在此时没有设置它们。如果您使用完整磁盘镜像，则可能需要设置 **image_source** 才能通过验证。*
- ***true** ：接口已通过验证。*
- ***None**: 接口不支持您的驱动。*

7.3. 为节点添加标签以加入到配置集

注册并检查每个节点的硬件后，为节点添加标签以加入到特定配置集中。这些配置集标签将您的节点与类别相匹配，这会将类别分配给部署角色。以下示例显示 Controller 节点的角色、类别、配置集和节点间的关系：

类型	描述
角色	**Controller** 角色定义 director 配置 Controller 节点的方式。
类型（Flavor）	**control** 类型定义了用作控制器的节点的硬件配置集。将此类型分配给 **Controller** 角色，以便 director 能够决定使用哪些节点。
配置集	**control** 配置集是应用至 **control** 类型的标签。它定义了属于该类型的节点。
节点	您也可以对单个节点应用 **control** 配置集标签，这样会将这些节点分组至 **control** 类型，因此，director 会使用 **Controller** 角色来配置它们。

*默认的配置集类型 **compute**、**control**、**swift-storage**、**ceph-storage** 和 **block-storage** 会在 undercloud 的安装过程中创建，多数环境中可不经修改直接使用。*

步骤

1. *为了通过添加标签把节点标记为特定的配置集，把 **profile** 选项添加到每个节点的 **properties/capabilities** 参数中。例如，要标记特定节点以使用特定配置集，请使用以下命令：*

```
(undercloud) $ NODE=<NODE NAME OR ID>
(undercloud) $ PROFILE=<PROFILE NAME>
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local" $NODE
```

   - *将 **$NODE** 变量设置为节点的名称或 UUID。*

- 将 **$PROFILE** 变量设置为特定的配置集，如 **control** 或 **compute**。
- **properties/capabilities** 中的 **profile** 选项包含 **$PROFILE** 变量，用于使用对应的配置集标记节点，如 **profile:control** 或 **profile:compute**。
- 设置 **boot_option:local** 选项，以定义每个节点的引导方式。

您还可以使用额外的 **openstack baremetal node show** 命令和 **jq** 筛选来保留现有 **capabilities** 值：

```
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local,$(openstack baremetal node show $NODE
-f json -c properties | jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

2. 在标记完节点后，检查分配的配置集或可能的配置集：

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

7.4. 将引导模式设置为 UEFI 模式

默认引导模式是传统的 BIOS 模式。您可以在 RHOSP 部署中将节点配置为使用 UEFI 引导模式，而不是传统的 BIOS 引导模式。

警告

有些硬件不支持旧的 BIOS 引导模式。如果您在不支持旧 BIOS 引导模式的硬件中使用旧的 BIOS 引导模式，则部署可能会失败。要确保您的硬件成功部署，请使用 UEFI 引导模式。

注意

如果启用 UEFI 引导模式，您必须构建自己的完整磁盘镜像，其中包括分区布局和引导装载程序，以及用户镜像。有关创建完整磁盘镜像的更多信息，请参阅创建完整磁盘镜像。

流程

1. 在 **undercloud.conf** 文件中设置下列参数：

```
ipxe_enabled = True
```

2. 保存 **undercloud.conf** 文件并运行 undercloud 安装：

```
$ openstack undercloud install
```

等待安装脚本完成。

3. 检查每个注册的节点的现有功能：

```
$ openstack baremetal node show <node> -f json -c properties | jq -r .properties.capabilities
```

- 将 **<node>** 替换为裸机节点的 ID。

4. 通过将 **boot_mode:uefi** 添加到节点的现有容量中，将每个注册节点的引导模式设置为 **uefi**：

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
<capability_n>" <node>
```

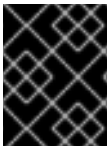
- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<capability_1&** gt; 以及所有功能（直到 **<capability_n** >）替换为您在第 3 步中获得的每个功能。
  例如，使用以下命令将引导模式设置为带有本地引导的 **uefi** ：

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,boot_option:local"
<node>
```

5. 将每个裸机类型的引导模式设置为 **uefi** ：

```
$ openstack flavor set --property capabilities:boot_mode='uefi' <flavor>
```

7.5. 启用虚拟介质引导



重要

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

您可以使用 Redfish 虚拟介质引导，向节点的 Baseboard Management Controller (BMC) 提供引导镜像，以便 BMC 可将镜像插入到其中一个虚拟驱动器中。然后，节点可以从虚拟驱动器引导到镜像中存在的操作系统。

Redfish 硬件类型支持通过虚拟介质引导部署、救援和用户镜像。Bare Metal 服务 (ironic) 使用与节点关联的内核和 ramdisk 镜像，在节点部署时为 UEFI 或 BIOS 引导模式构建可引导的 ISO 镜像。虚拟介质引导的主要优点是可以消除 PXE 的 TFTP 镜像传输阶段，并使用 HTTP GET 或其他方法。

要通过虚拟介质使用 **redfish** 硬件类型引导节点，请将引导接口设置为 **redfish-virtual-media**，对于 UEFI 节点，请定义 EFI 系统分区 (ESP) 镜像。然后将注册节点配置为使用 Redfish 虚拟介质引导。

前提条件

- 在 **undercloud.conf** 文件的 **enabled_hardware_types** 参数中启用 redfish 驱动程序。
- 注册并登记的裸机节点。
- Image Service (glance) 中的 IPA 和实例镜像。
- 对于 UEFI 节点，还必须在 Image Service (glance) 中有一个 EFI 系统分区镜像 (ESP)。
- 裸机类型。
- 用于清理和置备的网络。

流程

1. 将 Bare Metal 服务 (ironic) 引导接口设置为 **redfish-virtual-media**：

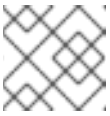
```
$ openstack baremetal node set --boot-interface redfish-virtual-media $NODE_NAME
```

- 将 **$NODE_NAME** 替换为节点的名称。

2. 对于 UEFI 节点，将引导模式设置为 **uefi**：

```
NODE=<NODE NAME OR ID> ; openstack baremetal node set --property
capabilities="boot_mode:uefi,$(openstack baremetal node show $NODE -f json -c properties
| jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

- 将 **$NODE** 替换为节点的名称。



注意

对于 BIOS 节点，请不要完成此步骤。

3. 对于 UEFI 节点，定义 EFI 系统分区 (ESP) 镜像：

```
$ openstack baremetal node set --driver-info bootloader=$ESP $NODE_NAME
```

- 将 **$ESP** 替换为 glance 镜像 UUID 或 ESP 镜像的 URL，并将 **$NODE_NAME** 替换为节点的名称。

注意

对于 BIOS 节点，请不要完成此步骤。

4. 在裸机节点上创建一个端口，并将端口与裸机节点上 NIC 的 MAC 地址关联：

```
$ openstack baremetal port create --pxe-enabled True --node $UUID $MAC_ADDRESS
```

- 将 **$UUID** 替换为裸机节点的 UUID，并将 **$MAC_ADDRESS** 替换为裸机节点上 NIC 的 MAC 地址。

7.6. 为多磁盘集群定义根磁盘

大多数 Ceph Storage 节点会使用多个磁盘。当节点使用多个磁盘时，director 必须识别根磁盘。默认情况下，director 在置备过程中将 overcloud 镜像写入根磁盘

使用这个流程通过序列号识别根设备。有关可以用来识别根磁盘的其他属性的更多信息，请参阅 第 7.7 节 "识别根磁盘的属性"。

流程

1. 验证每个节点的硬件内省的磁盘信息。以下命令显示节点的磁盘信息：

```
(undercloud)$ openstack baremetal introspection data save 1a4e30da-b6dc-499d-ba87-
0bd8a3819bc0 | jq ".inventory.disks"
```

例如，一个节点的数据可能会显示 3 个磁盘：

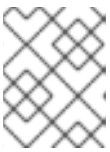
```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
```

2. 在 undercloud 上，为节点设置根磁盘。包括用于定义根磁盘的最合适的硬件属性值。

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":"
<serial_number>"}' <node-uuid>
```

例如：要将根设备设定为磁盘 2，其序列号为 **61866da04f380d001ea4e13c12e36ad6**，输入以下命令：

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":
"61866da04f380d001ea4e13c12e36ad6"}' 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```



注意

将每个节点的 BIOS 配置为从您选择的根磁盘引导。将引导顺序配置为首先从网络引导，然后从根磁盘引导。

director 识别特定磁盘以用作根磁盘。运行 **openstack overcloud deploy** 命令时，director 置备 overcloud 镜像并将其写入根磁盘。

7.7. 识别根磁盘的属性

您可以定义多个属性以帮助 director 识别根磁盘：

- **model**（字符串）：设备识别码。
- **vendor**（字符串）：设备厂商。
- **serial**（字符串）：磁盘序列号。
- **hctl**（字符串）：SCSI 的 Host:Channel:Target:Lun。
- **size**（整数）：设备的大小（以 GB 为单位）。
- **wwn**（字符串）：唯一的存储 ID。
- **wwn_with_extension**（字符串）：唯一存储 ID 附加厂商扩展名。
- **wwn_vendor_extension**（字符串）：唯一厂商存储标识符。
- **rotational**（布尔值）：旋转磁盘设备为 true (HDD)，否则为 false (SSD)。
- **name**（字符串）：设备名称，例如：/dev/sdb1。

重要

仅对具有持久名称的设备使用 **name** 属性。不要使用 **name** 来设置任何其他设备的根磁盘，因为此值在节点引导时可能会改变。

7.8. 使用 OVERCLOUD-MINIMAL 镜像来避免使用红帽订阅授权

默认情况下，director 在置备过程中将 QCOW2 **overcloud-full** 镜像写入根磁盘。**overcloud-full** 镜像使用有效的红帽订阅。但是，如果您不希望运行其他 OpenStack 服务或消耗您的订阅授权，您还可以使用 **overcloud-minimal** 镜像来置备裸机操作系统。

在您希望只使用 Ceph 守护进程来置备节点时，会发生此情况的常见用例。对于此情况和类似用例，使用 **overcloud-minimal** 镜像选项以避免达到您购买的红帽订阅的极限。有关如何获取 **overcloud-minimal** 镜像的信息，请参阅获取 overcloud 节点的镜像。

注意

Red Hat OpenStack Platform (RHOSP) 订阅包含 Open vSwitch (OVS)，但使用 **overcloud-minimal** 镜像时核心服务（如 OVS）不可用。部署 Ceph Storage 节点不需要 OVS。使用 **linux_bond** 定义绑定，而不使用 **ovs_bond**。有关 **linux_bond** 的更多信息，请参阅 Linux 绑定选项。

步骤

1. 要配置 director 使用 **overcloud-minimal** 镜像，创建包含以下镜像定义的环境文件：

```
parameter_defaults:
  <roleName>Image: overcloud-minimal
```

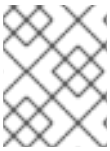
2. 将 **<roleName>** 替换为角色的名称，并将 **Image** 加到角色名称的后面。以下示例显示了 Ceph 存储节点的 **overcloud-minimal** 镜像：

```
parameter_defaults:
  CephStorageImage: overcloud-minimal
```

3. 在 **roles_data.yaml** 角色定义文件中，将 **rhsm_enforce** 参数设置为 **False**。

```
rhsm_enforce: False
```

4. 将环境文件传递给 **openstack overcloud deploy** 命令。



注意

overcloud-minimal 镜像仅支持标准 Linux 网桥，不支持 OVS，因为 OVS 是需要 Red Hat OpenStack Platform 订阅权利的 OpenStack 服务。

7.9. 创建特定于架构的角色

构建多架构云时，必须将所有架构特定的角色添加到 **roles_data.yaml** 文件中。以下示例包括 **ComputePPC64LE** 角色和默认角色：

```
openstack overcloud roles generate \
  --roles-path /usr/share/openstack-tripleo-heat-templates/roles -o ~/templates/roles_data.yaml \
  Controller Compute ComputePPC64LE BlockStorage ObjectStorage CephStorage
```

创建自定义角色文件部分包含角色的相关信息。

7.10. 环境文件

undercloud 包括一组构成您的 overcloud 创建计划的 heat 模板。您可以使用环境文件来自定义 overcloud 的各个方面，这些文件是 YAML 格式的文件，其内容可覆盖核心 heat 模板集合中的参数和资源。您可以根据需要纳入多个环境文件。但是，环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源更为优先。以下列表是环境文件顺序的示例：

- 节点数和每个角色的类别。包含此信息对于创建 overcloud 至关重要。
- 容器化 OpenStack 服务的容器镜像位置。
- 任何网络隔离文件，首先是 heat 模板集合中的初始化文件 (**environments/network-isolation.yaml**)，然后是您自定义的 NIC 配置文件，最后是任何额外的网络配置。有关更多信息，请参阅高级 Overcloud 自定义指南中的以下章节：
  - "基本网络隔离"
  - "自定义可组合网络"
  - "自定义网络接口模板"
- 使用外部负载均衡器时的所有外部负载均衡环境文件。有关更多信息，请参阅 Overcloud 的外部负载均衡。
- 任何存储环境文件，如 Ceph Storage、NFS 或 iSCSI。

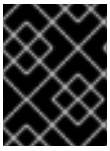
- 任何用于红帽 CDN 或 Satellite 注册的环境文件。
- 任何其它自定义环境文件。

注意

Open Virtual Networking (OVN) 是 Red Hat OpenStack Platform 16.2 中的默认网络机制驱动程序。如果要将 OVN 与分布式虚拟路由 (DVR) 搭配使用，则必须在 **openstack overcloud deploy** 命令中包含 **environments/services/neutron-ovn-dvr-ha.yaml** 文件。如果要在没有 DVR 的情况下使用 OVN，则必须在 **openstack overcloud deploy** 命令中包含 **environment/services/neutron-ovn-ha.yaml** 文件。

红帽建议将自定义环境文件组织在一个单独目录中，比如 **templates** 目录。

有关自定义 overcloud 高级功能的更多信息，请参阅高级 Overcloud 自定义指南。



重要

一个基本的 overcloud 会使用本地 LVM 存储作为块存储，这种配置不受支持。建议您使用外部存储解决方案（如 Red Hat Ceph Storage）来实现块存储。

注意

环境文件扩展名必须为 **.yaml** 或 **.template**，否则不会被视为自定义模板资源。

接下来的几个部分包含有关创建 overcloud 所需的一些环境文件的信息。

7.11. 创建定义节点数目和类型的环境文件

默认情况下，director 部署具有 1 个 Controller 节点和 1 个 Compute 节点的 overcloud，节点的类型是 **baremetal**。不过，这仅适用于概念验证部署。您可以通过指定不同的节点数目和类型来覆盖默认配置。对于小规模的生产环境，部署至少 3 个 Controller 节点和 3 个 Compute 节点，并分配特定类别以确保节点具有正确的资源规范。完成以下步骤以创建名为 **node-info.yaml** 的环境文件，用于存储节点计数和类别分配。

步骤

1. 在 **/home/stack/templates/** 目录中创建 **node-info.yaml** 文件：

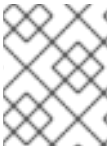
```
(undercloud) $ touch /home/stack/templates/node-info.yaml
```

2. 编辑这个文件，使其包含您需要的节点数目和类型。此示例包含 3 个 Controller 节点和 3 个 Compute 节点：

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  ControllerCount: 3
  ComputeCount: 3
```

7.12. 创建 UNDERCLOUD CA 信任的环境文件

如果您的 undercloud 使用 TLS，而证书颁发机构 (CA) 未获得公开信任，可将此 CA 用于 undercloud 操作的 SSL 端点加密。为确保其余部署可以访问 undercloud 端点，请将您的 overcloud 节点配置成信任 undercloud CA。



注意

要使这种方法奏效，overcloud 节点必须有一个指向 undercloud 公共端点的网络路由。依赖于脊叶型网络的部署可能必须应用这种配置。

有两种自定义证书可用于 undercloud：

- **用户提供的证书** - 当您自行提供证书时，会应用此定义。证书可能来自于您自己的 CA，也可能是自签名的。这通过使用 **undercloud_service_certificate** 选项来传递。在这种情况下，您必须信任自签名证书，或 CA（具体取决于您的部署）。
- **自动生成的证书** - 当您通过 **certmonger** 生成使用自己的本地 CA 的证书时，会应用此定义。使用 **undercloud.conf** 文件中的 **generate_service_certificate** 选项启用自动生成的证书。在这种情况下，director 在 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** 生成 CA 证书，并配置 undercloud 的 HAProxy 实例以使用服务器证书。将 CA 证书添加到 **inject-trust-anchor-hiera.yaml** 文件中以将其呈现给 OpenStack Platform。

本例中使用了位于 **/home/stack/ca.crt.pem** 的一个自签名证书。如果您使用自动生成的证书，请改为使用 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**。

步骤

1. 打开证书文件，仅复制证书部分。不要包括其密钥：

```
$ vi /home/stack/ca.crt.pem
```

您需要的证书部分与下方简写的示例类似：

```
-----BEGIN CERTIFICATE-----
MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECgwH
UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
-----END CERTIFICATE-----
```

2. 创建一个名为 **/home/stack/inject-trust-anchor-hiera.yaml** 的 YAML 文件，其包含下列内容以及您从 PEM 文件复制而来的证书：

```
parameter_defaults:
  CAMap:
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECgwH

UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
        -----END CERTIFICATE-----
```

注意

证书字符串必须采用 PEM 格式。

注意

CAMap 参数可能包含其他与 SSL/TLS 配置相关的证书。

在部署 overcloud 的过程中，director 将 CA 证书复制到每个 overcloud 节点。因此，每个节点都会信任 undercloud 的 SSL 端点提供的加密。有关环境文件的详情请参考 第 7.16 节 "在 overcloud 部署中包括环境文件"。

7.13. 在新部署中禁用 TSX

从 Red Hat Enterprise Linux 8.3 开始，内核默认禁用对 Intel 事务同步扩展 (TSX) 功能的支持。

您必须为新的 overcloud 显式禁用 TSX，除非您特别需要将其用于工作负载或第三方供应商。

在环境文件中设置 **KernelArgs** heat 参数。

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "tsx=off"
```

在运行 **openstack overcloud deploy** 命令时包含该环境文件：

其他资源

- "Intel TSX 对 OpenStack 虚拟客户机的影响的指南（适用于 RHEL 8.3 及更高版本）"

7.14. 部署命令

创建 OpenStack 环境的最后一个阶段是，运行 **openstack overcloud deploy** 命令以创建 overcloud。在运行此命令前，熟悉关键的选项，以及如何纳入自定义的环境文件。

警告

请勿将 **openstack overcloud deploy** 作为后台进程运行。如果将其作为后台进程运行，overcloud 创建可能在部署过程中挂起。

7.15. 部署命令选项

下表列出 **openstack overcloud deploy** 命令的其他参数。

重要

一些选项在此发行版本中作为技术预览提供，因此不享有红帽的全面支持。它们只应用于测试，不应在生产环境中使用。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

表 7.1. 部署命令选项

参数	描述
--templates [TEMPLATES]	包含您要部署的 heat 模板的目录。如果为空，部署命令会使用位于 **/usr/share/openstack-tripleo-heat-templates/** 的默认模板。
--stack STACK	要创建或更新的堆栈的名称
-t [TIMEOUT]，--timeout [TIMEOUT]	以分钟为单位的部署超时持续时间
--libvirt-type [LIBVIRT_TYPE]	要用于虚拟机监控程序的虚拟化类型
--ntp-server [NTP_SERVER]	要用于同步时间的网络时间协议 (NTP) 服务器。您也可以在以逗号分隔的列表中指定多个 NTP 服务器，例如：**--ntp-server 0.centos.pool.org，1.centos.pool.org**。对于高可用性集群部署，重要的是各个 Controller 节点始终引用同一时间源。但请注意，通常的环境可能已经指定了符合公认规范的 NTP 时间源。
--no-proxy [NO_PROXY]	为环境变量 **no_proxy** 指定自定义值。 这个环境变量被用来在代理通信中排除特定的主机名。
--overcloud-ssh-user OVERCLOUD_SSH_USER	定义访问 overcloud 节点的 SSH 用户。SSH 访问通常使用 **heat-admin** 用户。
--overcloud-ssh-key OVERCLOUD_SSH_KEY	定义用于 SSH 访问 overcloud 节点的密钥路径。
--overcloud-ssh-network OVERCLOUD_SSH_NETWORK	定义要用于 SSH 访问 overcloud 节点的网络名称。
-e [EXTRA HEAT TEMPLATE]，--environment-file [ENVIRONMENT FILE]	要传递给 overcloud 部署的额外环境文件。您可以多次指定此选项。请注意，传递到 **openstack overcloud deploy** 命令的环境文件顺序是非常重要的。例如，如果一个参数在多个环境文件中出现，则后续环境文件中的参数将覆盖前面文件中的同一参数。
--environment-directory	包含要在部署中包括的环境文件的目录。部署命令以数字顺序，然后以字母顺序处理这些环境文件。

参数	描述
-r ROLES_FILE	定义角色文件并覆盖 **--templates** 目录里的默认 **roles_data.yaml** 。文件位置可以是绝对路径或者相对于 **--templates** 的路径。
-n NETWORKS_FILE	定义网络文件并覆盖 **--templates** 目录里的默认 network_data.yaml。文件位置可以是绝对路径或者相对于 **--templates** 的路径。
-p PLAN_ENVIRONMENT_FILE	定义计划环境文件，并覆盖 **--templates** 目录里的默认 **plan-environment.yaml** 。文件位置可以是绝对路径或者相对于 **--templates** 的路径。
--no-cleanup	如果您不希望在部署后删除临时文件，并记录其位置，则使用此选项。
--update-plan-only	如果您要在不执行实际部署的情况下更新计划，则使用此选项。
--validation-errors-nonfatal	overcloud 在创建过程中会执行一组部署前检查。如果部署前检查出现任何非严重错误，则此选项会退出创建。我们推荐使用此选项，因为任何错误都有可能造成部署失败。
--validation-warnings-fatal	overcloud 在创建过程中会执行一组部署前检查。如果部署前检查出现任何非关键警告，则此选项会退出创建。openstack-tripleo-validations
--dry-run	如果您要在不创建 overcloud 的情况下对 overcloud 执行验证检查，则使用此选项。
--run-validations	使用此选项从 **openstack-tripleo-validations** 软件包运行外部验证。
--skip-postconfig	使用此选项跳过 overcloud 部署后配置。
--force-postconfig	使用此选项强制进行 overcloud 部署后配置。
--skip-deploy-identifier	如果您不希望部署命令为 **DeployIdentifier** 参数生成唯一标识符，则使用此选项。软件配置部署步骤仅当配置发生实际更改时才会触发。使用此选项要非常谨慎，仅当您确信不需要运行软件配置（如扩展某些角色）时方可使用。
--answers-file ANSWERS_FILE	带有选项和参数的 YAML 文件的路径。

参数	描述
--disable-password-generation	如果要禁用 overcloud 服务的密码生成，则使用此选项。
--deployed-server	如果要部署预置备 overcloud 节点，则使用此选项。与 **--disable-validations** 结合使用。
--no-config-download， --stack-only	如果您要禁用 **config-download** 工作流，仅创建堆栈和相关 OpenStack 资源，则使用此选项。此命令不会对 overcloud 应用软件配置。
--config-download-only	如果您要禁用 overcloud 栈创建，并仅运行 **config-download** 工作流以应用软件配置，则使用此选项。
--output-dir OUTPUT_DIR	要用于保存 **config-download** 输出的目录。该目录必须可由 mistral 用户写入。如果没有指定，director 将使用默认值，即 **/var/lib/mistral/overcloud**。
--override-ansible-cfg OVERRIDE_ANSIBLE_CFG	Ansible 配置文件的路径。该文件的配置会覆盖 **config-download** 默认生成的所有配置。
--config-download-timeout CONFIG_DOWNLOAD_TIMEOUT	您要用于 **config-download** 步骤的超时持续时间（以分钟为单位）。如果未设置，director 会在堆栈部署操作后将默认值设置为从 **--timeout** 参数中保留的时间。
--limit NODE1，NODE2	将此选项与以逗号分隔的节点列表一起使用，将 config-download playbook 执行限制到特定的节点或一组节点。例如，当想要仅在新节点上运行 config-download 时，**--limit** 选项对扩展操作很有用。此参数可能会导致在主机间实时迁移实例失败，请参阅使用 ansible-playbook-command.sh 脚本运行 config-download
--tags TAG1，TAG2	**（技术预览）**将此选项与 config-download playbook 中的以逗号分隔的标签列表一起使用，通过一组特定 config-download 任务来运行部署。
--skip-tags TAG1，TAG2	**（技术预览）**将此选项与您要从 config-download playbook 中跳过的以逗号分隔的标签列表一起使用。

运行以下命令查看完整选项列表：

```
(undercloud) $ openstack help overcloud deploy
```

*某些命令行参数已过时或已弃用，它们的功能可以通过环境文件的 **parameter_defaults** 部分中所包含的 heat 模板参数实现。下表将已弃用的参数与 heat 模板中的等效参数对应了起来。*

表 7.2. 将被弃用的 CLI 参数映射到 heat 模板参数

参数	描述	Heat 模板参数
--control-scale	扩展的 Controller 节点数量	**ControllerCount**
--compute-scale	扩展的 Compute 节点数量	**ComputeCount**
--ceph-storage-scale	扩展的 Ceph 节点数量	**CephStorageCount**
--block-storage-scale	扩展的 Block Storage (cinder) 节点数量	**BlockStorageCount**
--swift-storage-scale	扩展的 Object Storage (swift) 节点数量	**ObjectStorageCount**
--control-flavor	要用于 Controller 节点的 flavor	**OvercloudControllerFlavor**
--compute-flavor	要用于 Compute 节点的 flavor	**OvercloudComputeFlavor**
--ceph-storage-flavor	要用于 Ceph Storage 节点的 flavor	**OvercloudCephStorageFlavor**
--block-storage-flavor	要用于 Block Storage (cinder) 节点的 flavor	**OvercloudBlockStorageFlavor**
--swift-storage-flavor	要用于 Object Storage (swift) 节点的 flavor	**OvercloudSwiftStorageFlavor**
--validation-errors-fatal	overcloud 在创建过程中会执行一组部署前检查。在使用这个选项时，如果部署前检查出现任何严重错误，则会退出创建。我们推荐使用此选项，因为任何错误都有可能造成部署失败。	未进行参数映射
--disable-validations	完全禁用部署前验证。这些验证是内置部署前验证，已由 **openstack-tripleo-validations** 软件包中的外部验证替代。	未进行参数映射
--config-download	使用 **config-download** 机制运行部署。现在这是默认选项，以后可以删除该 CLI 选项。	未进行参数映射
--rhel-reg	使用此选项把 overcloud 节点注册到客户门户或 Satellite 6。	**RhsmVars**

参数	描述	Heat 模板参数
--reg-method	使用此选项定义您要用于 overcloud 节点的注册方法。**satellite** 代表 Red Hat Satellite 6 或 Red Hat Satellite 5，**portal** 代表客户门户（Customer Portal）。	**RhsmVars**
--reg-org [REG_ORG]	要用于注册的组织。	**RhsmVars**
--reg-force	使用此选项注册系统（即使已经注册过）。	**RhsmVars**
--reg-sat-url [REG_SAT_URL]	注册 overcloud 节点的 Satellite 服务器的基本 URL。此参数需要使用 Satellite 的 HTTP URL 而不是 HTTPS URL。例如，使用 http://satellite.example.com 而不使用 https://satellite.example.com。overcloud 的创建过程会使用此 URL 来确定服务器是 Red Hat Satellite 5 还是 Red Hat Satellite 6 服务器。如果服务器是 Red Hat Satellite 6 服务器，则 overcloud 获取 **katello-ca-consumer-latest.noarch.rpm** 文件，使用 **subscription-manager** 注册，并安装 **katello-agent**。如果服务器是 Red Hat Satellite 5 服务器，则 overcloud 获取 **RHN-ORG-TRUSTED-SSL-CERT** 文件并使用 **rhnreg_ks** 注册。	**RhsmVars**
--reg-activation-key [REG_ACTIVATION_KEY]	使用此选项定义您要用于注册的激活码。	**RhsmVars**

这些参数计划从未来的 Red Hat OpenStack Platform 版本中移除。

7.16. 在 OVERCLOUD 部署中包括环境文件

*使用 **-e** 选项包括环境文件以自定义您的 overcloud。您可以根据需要纳入多个环境文件。但是，环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源更为优先。*

*使用 **-e** 选项添加到 overcloud 的所有环境文件都会成为 overcloud 堆栈定义的一部分。*

下例中的命令演示如何使用此场景中早前定义的环境文件来启动 overcloud 创建过程：

```
(undercloud) $ openstack overcloud deploy --templates \
```

```
-e /home/stack/templates/node-info.yaml\
-e /home/stack/containers-prepare-parameter.yaml \
-e /home/stack/inject-trust-anchor-hiera.yaml \
-r /home/stack/templates/roles_data.yaml \
```

这个命令包括以下额外选项：

--templates

以 **/usr/share/openstack-tripleo-heat-templates** 中的 heat 模板集合为基础来创建 overcloud。

-e /home/stack/templates/node-info.yaml

添加环境文件以定义每种角色有多少个节点以及使用哪些类型。

-e /home/stack/containers-prepare-parameter.yaml

添加容器镜像准备环境文件。您在安装 undercloud 的过程中生成了此文件，可使用此文件创建 overcloud。

-e /home/stack/inject-trust-anchor-hiera.yaml

添加用于在 undercloud 中安装自定义证书的环境文件。

-r /home/stack/templates/roles_data.yaml

（可选）如果使用自定义角色或要启用多架构云，这是生成的角色数据。更多信息请参阅 第 7.9 节 "创建特定于架构的角色"。

director 需要这些环境文件用于重新部署和部署后功能。没有正确包含这些文件可能会破坏您的 overcloud。

要在稍后阶段修改 overcloud 配置，请执行以下操作：

1. 修改定制环境文件和 heat 模板中的参数。
2. 使用相同的环境文件再次运行 **openstack overcloud deploy** 命令

不要直接编辑 overcloud 配置，因为 director 在更新 overcloud 栈时会覆盖任何手动配置。

7.17. 运行部署前验证

运行 **pre-deployment** 验证组检查部署要求。

步骤

1. source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 此验证需要 overcloud 计划的副本。使用所有必要的环境文件上传 overcloud 计划。如果仅上传您的计划，请使用 **--update-plan-only** 选项运行 **openstack overcloud deploy** 命令：

```
$ openstack overcloud deploy --templates \
    -e environment-file1.yaml \
    -e environment-file2.yaml \
    ...
    --update-plan-only
```

3. 使用 **--group pre-deployment** 选项 运行 **openstack tripleo validator run** 命令：

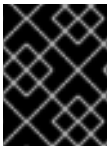
```
$ openstack tripleo validator run --group pre-deployment
```

4. 如果 overcloud 使用与默认 **overcloud** 名称不同的计划名称，请使用 **--plan** 选项来设置计划名称：

```
$ openstack tripleo validator run --group pre-deployment \
  --plan myovercloud
```

5. 查看验证报告的结果。要查看特定验证的详细输出，请根据报告中具体验证的 UUID 运行 **openstack tripleo validator show run --full** 命令：

```
$ openstack tripleo validator show run --full <UUID>
```



重要

FAILED 验证不会阻止您部署或运行 Red Hat OpenStack Platform。但是， **FAILED** 验证可能会显示生产环境中潜在的问题。

7.18. OVERCLOUD 部署输出

overcloud 创建完毕后，director 会提供为配置 overcloud 而执行的 Ansible play 的总结：

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director 还会提供用于访问 overcloud 的详细信息。

```
Ansible passed.
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

7.19. 访问 OVERCLOUD

director 会生成脚本来配置和帮助认证 undercloud 与 overcloud 的交互。director 将此文件保存为 **stack** 用户主目录的 **overcloudrc** 文件。运行以下命令来使用此文件：

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

此命令加载从 undercloud CLI 与 overcloud 交互所需的环境变量。命令提示符的变化会显示当前状态：

```
(overcloud) $
```

要返回与 undercloud 进行交互的状态，请运行以下命令：

```
-
```

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

7.20. 运行部署后验证

运行**post-deployment**验证组检查部署后的状态。

步骤

1. source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 使用 **--group post-deployment** 选项运行 **openstack tripleo validator run** 命令：

```
$ openstack tripleo validator run --group post-deployment
```

3. 如果 overcloud 使用与默认 **overcloud** 名称不同的计划名称，请使用 **--plan** 选项来设置计划名称：

```
$ openstack tripleo validator run --group post-deployment \
    --plan myovercloud
```

4. 查看验证报告的结果。要查看特定验证的详细输出，请根据报告中具体验证的 UUID 运行 **openstack tripleo validator show run --full** 命令：

```
$ openstack tripleo validator show run --full <UUID>
```

重要

FAILED 验证不会阻止您部署或运行 Red Hat OpenStack Platform。但是，**FAILED** 验证可能会显示生产环境中潜在的问题。

第 8 章 部署 OVERCLOUD 前置备裸机节点

重要

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

overcloud 部署过程包含两个主要操作：

- 置备节点
- 部署 overcloud

如果您将这些操作划分为不同的过程，您可以降低这个过程带来的风险，并更有效地识别故障点：

1. 置备裸机节点。
   a. 以 yaml 格式创建节点定义文件。
   b. 运行 provisioning 命令，包括节点定义文件。
2. 部署 overcloud。
   a. 运行部署命令，包括置备命令生成的 heat 环境文件。

置备过程会置备节点并生成包含各种节点规格的 heat 环境文件，包括节点数、预先节点放置、自定义镜像和自定义 NIC。当您部署 overcloud 时，请将此文件包括在部署命令中。

重要

您无法将预置备节点与 director 置备的节点合并。

8.1. 为 OVERCLOUD 注册节点

director 需要一个节点定义模板，用于指定节点的硬件和电源管理详情。您可以使用 JSON 格式、node **.json** 或 YAML 格式创建此模板 **nodes.yaml**。

流程

1. 创建名为 **nodes.json** 或 **nodes.yaml** 的模板，它将列出您的节点。使用以下 JSON 和 YAML 模板示例了解如何创建节点定义模板的结构：

   示例 JSON 模板

```
{
  "nodes": [{
    "name": "node01",
    "ports": [{
      "address": "aa:aa:aa:aa:aa:aa",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
```

```
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.205"
  },
  {
    "name": "node02",
    "ports": [{
      "address": "bb:bb:bb:bb:bb:bb",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.206"
  }]
}
```

示例 YAML 模板

```
nodes:
  - name: "node01"
    ports:
      - address: "aa:aa:aa:aa:aa:aa"
        physical_network: ctlplane
        local_link_connection:
          switch_id: "52:54:00:00:00:00"
          port_id: p0
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.205"
  - name: "node02"
    ports:
      - address: "bb:bb:bb:bb:bb:bb"
        physical_network: ctlplane
        local_link_connection:
          switch_id: "52:54:00:00:00:00"
          port_id: p0
```

```
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.206"
```

此模板包含以下属性：

name

节点的逻辑名称。

ports

访问特定 IPMI 设备的端口。您可以定义以下可选端口属性：

- **地址**：节点上网络接口的 MAC 地址。对于每个系统的 Provisioning NIC，只使用 MAC 地址。
- **physical_network**：连接到 Provisioning NIC 的物理网络。
- **local_link_connection**：如果您使用 IPv6 置备，且 LLDP 在内省过程中没有正确填充本地链路连接，您必须在 local_link_connection 参数中包含带有 **switch_id** 和 **port_id** 字段的虚拟数据。有关如何包含虚拟数据的更多信息，请参阅使用 director 内省来收集裸机节点硬件信息。

cpu

节点上的 CPU 数量。（可选）

memory

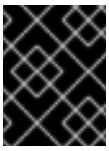
以 MB 为单位的内存大小。（可选）

disk

以 GB 为单位的硬盘的大小。（可选）

arch

系统架构。（可选）

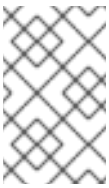


重要

在构建多架构云时，**arch** 键是必需的，用于区分使用 **x86_64** 和 **ppc64le** 架构的节点。

pm_type

要使用的电源管理驱动程序。此示例使用 IPMI 驱动程序（**ipmi**）。



注意

IPMI 是首选的受支持电源管理驱动程序。有关支持的电源管理类型及其选项的更多信息，请参阅 电源管理驱动程序。如果这些电源管理驱动程序不能正常工作，请将 IPMI 用于电源管理。

pm_user; pm_password

IPMI 的用户名和密码。

pm_addr

IPMI 设备的 IP 地址。

2. 创建模板后，运行以下命令验证格式和语法：

```
$ source ~/stackrc
(undercloud)$ openstack overcloud node import --validate-only ~/nodes.json
```

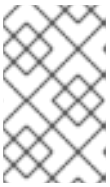
重要

您还必须为多架构节点包含 **--http-boot /var/lib/ironic/tftpboot/** 选项。

3. 将文件保存到 **stack** 用户的主目录(**/home/stack/nodes.json**)。

4. 将模板导入到 director，将每个节点从模板注册到 director：

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```



注意

如果使用 UEFI 引导模式，还必须在每个节点上设置引导模式。如果您在不设置 UEFI 引导模式的情况下内省节点，节点以旧模式引导。如需更多信息，请参阅将引导模式设置为 UEFI 引导模式。

5. 等待节点完成注册和配置。完成后，确认 director 已成功注册节点：

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

8.2. 创建裸机节点硬件的清单

director 需要 Red Hat OpenStack Platform (RHOSP)部署中节点的硬件清单进行配置集标记、基准测试和手动根磁盘分配。

您可以使用以下方法之一向 director 提供硬件清单：

- 自动：您可以使用 director 的内省过程，从每个节点收集硬件信息。这个过程在每个节点上启动内省代理。内省代理从节点收集硬件数据并将数据送回 director。director 将硬件数据存储在 undercloud 节点上运行的对象存储服务(swift)中。
- 手动：您可以手动配置每个裸机的基本硬件清单。此清单存储在裸机置备服务(ironic)中，用于管理和部署裸机机器。

注意

如果您的 overcloud 使用 derived **_params.yaml**，则需要使用 director 的自动内省过程，这需要存在内省数据。如需有关 derived **_params.yaml** 的更多信息，请参阅 工作流和派生参数。

director 自动内省流程比设置裸机置备服务端口的手动方法提供以下优点：

- 内省记录了硬件信息中的所有连接端口，包括在 **nodes.yaml** 中尚未配置 PXE 引导的端口。
- 如果属性可以使用 LLDP 发现，内省会为每个端口设置 **local_link_connection** 属性。使用手动方法时，您必须在注册节点时为每个端口配置 **local_link_connection**。
- 在部署 spine-and-leaf 或 DCN 架构时，内省为裸机置备服务端口设置 **physical_network** 属性。

8.2.1. 使用 director 内省来收集裸机节点硬件信息

将物理机注册为裸机节点后，您可以使用 director 内省自动添加其硬件详情并为每个以太网 MAC 地址创建端口。

提示

作为自动内省的替代选择，您可以手动为 director 提供裸机节点的硬件信息。如需更多信息，请参阅 *手动配置裸机节点硬件信息*。

先决条件

- 您已为 overcloud 注册了裸机节点。

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
```

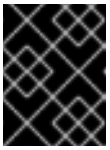
3. 运行 pre-introspection 验证组来检查内省要求：

```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 查看验证报告的结果。
5. 可选：查看特定验证的详细输出：

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

- 将 **<**validation> 替换为您要查看报告的特定验证 UUID。



重要

FAILED 验证不会阻止您部署或运行 Red Hat OpenStack Platform。但是，**FAILED** 验证可能会显示生产环境中潜在的问题。

6. 检查每个节点的属性。您可以检查所有节点的硬件属性或特定节点：
   - 检查所有节点的硬件属性：

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

- 使用 **--all-manageable** 选项仅内省处于受管理状态的节点。在此示例中，所有节点都处于受管理状态。
- 使用 **--provide** 选项在内省后将所有节点重置为 **available** 状态。

- 检查特定节点的硬件属性：

```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

- 使用 **--provide** 选项，在内省后将所有指定节点重置为 **available** 状态。
- 将 **<node1>**, **[node2]**, 及所有节点（直到 **[noden]**）替换为您要内省的每个节点的 UUID。

7. 在一个单独的终端窗口中监控内省进度日志：

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```

重要

确保内省进程运行完。内省通常需要 15 分钟时间用于裸机节点。但是，不正确的内省网络可能会导致它花费更长的时间，这可能会导致内省失败。

8. 可选：如果您已将 undercloud 配置为通过 IPv6 进行裸机置备，那么您还需要检查 LLDP 是否已为裸机置备服务(ironic)端口设置了 **local_link_connection** ：

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

- 如果裸机节点上的端口的 Local Link Connection 字段为空，则必须使用虚拟数据手动填充 **local_link_connection** 值。以下示例将虚拟交换机 ID 设置为 **52:54:00:00:00:00**，并将虚拟端口 ID 设置为 **p0** ：

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- 验证 "Local Link Connection" 字段是否包含虚拟数据：

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

内省完成后，所有节点都会变为 **available** 状态。

8.2.2. 手动配置裸机节点硬件信息

将物理机注册为裸机节点后，您可以手动添加其硬件详情，并为每个以太网 MAC 地址创建裸机端口。在部署 overcloud 前，必须至少创建一个裸机端口。

提示

作为手动内省的替代选择，您可以使用自动 director 内省流程来收集裸机节点的硬件信息。如需更多信息，请参阅使用 director 内省来收集裸机节点硬件信息。

先决条件

- 您已为 overcloud 注册了裸机节点。
- 您已为 node **.json** 中注册的节点上的每个端口配置了 **local_link_connection**。有关更多信息，请参阅为 overcloud 注册节点。

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. 通过在节点的功能中添加 **boot_option':'local** 来将每个注册的节点的引导选项设置为 **local**：

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

   - 将 **<node>** 替换为裸机节点的 ID。

4. 指定部署内核并为节点驱动程序部署 ramdisk：

```
(undercloud)$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

   - 将 **<node>** 替换为裸机节点的 ID。
   - 将 **<kernel_file** > 替换为 **.kernel** 镜像的路径，例如 **file:///var/lib/ironic/httpboot/agent.kernel**。
   - 将 **<initramfs_file** > 替换为 **.initramfs** 镜像的路径，例如 **file:///var/lib/ironic/httpboot/agent.ramdisk**。

5. 更新节点属性以匹配节点上的硬件规格：

```
(undercloud)$ openstack baremetal node set <node> \
  --property cpus=<cpu> \
  --property memory_mb=<ram> \
  --property local_gb=<disk> \
  --property cpu_arch=<arch>
```

   - 将 **<node>** 替换为裸机节点的 ID。
   - 将 **<cpu** > 替换为 CPU 数量。
   - 将 & **lt;ram&** gt; 替换为 RAM （以 MB 为单位）。
   - 将 **<disk>** 替换为磁盘大小（以 GB 为单位）。
   - 将 & **lt;arch&** gt; 替换为构架类型。

6. 可选：为每个节点指定 IPMI 密码套件：

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<version>** 替换为节点上要使用的密码套件版本。设置为以下有效值之一：
  - **3** - 节点使用带有 SHA1 密码套件的 AES-128。
  - **17** - 节点使用带有 SHA256 密码套件的 AES-128。

7. 可选：如果您有多个磁盘，请设置 root 设备提示来告知部署 ramdisk 哪个磁盘用于部署：

```
(undercloud)$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- 将 **<node>** 替换为裸机节点的 ID。
- 将 **<property>** 和 **<value>** 替换为您要用于部署的磁盘的详情，如 **root_device='{"size": "128"}'**
  RHOSP 支持以下属性：
  - **model**（字符串）：设备识别码。
  - **vendor**（字符串）：设备厂商。
  - **serial**（字符串）：磁盘序列号。
  - **hctl**（字符串）：SCSI 的 Host:Channel:Target:Lun。
  - **size**（整数）：设备的大小（以 GB 为单位）。
  - **wwn**（字符串）：唯一的存储 ID。
  - **wwn_with_extension**（字符串）：唯一存储 ID 附加厂商扩展名。
  - **wwn_vendor_extension**（字符串）：唯一厂商存储标识符。
  - **rotational**（布尔值）：旋转磁盘设备为 true (HDD)，否则为 false (SSD)。
  - **name**（字符串）：设备名称，例如：/dev/sdb1 仅将此属性用于具有持久名称的设备。

注意

如果您指定多个属性，该设备必须与所有这些属性匹配。

8. 通过在 provisioning 网络中创建带有 NIC 的 MAC 地址的端口来通知节点网卡的裸机置备服务：

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- 将 **<node_uuid>** 替换为裸机节点的唯一 ID。
- 将 **<mac_address>** 替换为用于 PXE 引导的 NIC 的 MAC 地址。

9. 验证节点的配置：

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| Interface  | Result | Reason                                     |
+------------+--------+--------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| console    | None   | not supported                              |
| deploy     | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| inspect    | None   | not supported                              |
| management | True   |                                            |
| network    | True   |                                            |
| power      | True   |                                            |
| raid       | True   |                                            |
| storage    | True   |                                            |
+------------+--------+--------------------------------------------+
```

验证输出 **Result** 表示以下内容：

- **false** ：接口失败验证。如果提供的原因包括缺少 **instance_info** 参数 **[\'ramdisk', \'kernel', \'image_source']**，这可能是因为计算服务在部署过程开始时填充缺少的参数，因此在此时没有设置它们。如果您使用完整磁盘镜像，则可能需要设置 **image_source** 才能通过验证。
- **true** ：接口已通过验证。
- **None**: 接口不支持您的驱动。

8.3. 置备裸机节点

创建一个新的 YAML 文件 ~/**overcloud-baremetal-deploy.yaml**，定义您要部署的裸机节点的数量和属性，并为这些节点分配 overcloud 角色。置备过程会创建一个 heat 环境文件，您可以将其包括在 **openstack overcloud deploy** 命令中。

先决条件

- 已安装 undercloud。如需更多信息，请参阅安装 director。
- 裸机节点已内省并可用于置备和部署。有关更多信息，请参阅为 overcloud 注册节点 和 创建裸机节点硬件清单。

流程

1. 查找 **stackrc** undercloud 凭据文件：

```
$ source ~/stackrc
```

2. 创建新的 ~/**overcloud-baremetal-deploy.yaml** 文件，并为您要置备的每个角色定义节点数。例如，要置备三个 Controller 节点和三个 Compute 节点，请使用以下语法：

```
- name: Controller
  count: 3
- name: Compute
  count: 3
```

3. 在 ~/**overcloud-baremetal-deploy.yaml** 文件中，定义您要分配给节点的任何预先节点放置、自定义 NIC 或其他属性。例如，使用以下示例语法在节点 **node00**、**node01** 和 **node02** 上置备三个 Controller 节点，在 **node04**、**node05** 和 **node06** 上置备三个 Compute 节点：

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  - hostname: overcloud-novacompute-1
    name: node05
  - hostname: overcloud-novacompute-2
    name: node06
```

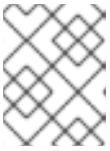
您还可以用 **defaults** 参数覆盖默认参数值，以避免为每个节点条目手动定义节点：

```
- name: Controller
  count: 3
  defaults:
    nics:
      network: custom-network
      subnet: custom-subnet
  instances:
  - hostname: overcloud-controller-0
    name: node00
  ...
```

有关您可以在节点定义文件中使用的参数、属性和值的更多信息，请参阅 *裸机节点置备属性*。

4. 可选：默认情况下，置备过程使用 **overcloud-full** 镜像。您可以使用 **image** 属性定义自定义镜像，以用于角色的所有节点，或用于特定的节点实例：

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
    image:
      href: overcloud-custom
```



注意

具有大于 2 TiB 的根(/)分区的裸机节点需要使用整个磁盘镜像。有关完整磁盘镜像的更多信息，请参阅创建整个磁盘镜像。

5. 运行置备命令，指定 **~/overcloud-baremetal-deploy.yaml** 文件，并使用 **--output** 选项定义输出文件：

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

置备过程会生成一个 heat 环境文件，其名称是您在 **--output** 选项中指定的名称。此文件包含节点定义。当您部署 overcloud 时，请将此文件包括在部署命令中。

6. 在一个单独的终端中对节点进行监控以验证它们是否成功置备。置备过程将节点状态从 **available** 改为 **active**：

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

使用 **metalsmith** 工具获取节点的统一视图，包括分配和 neutron 端口：

```
(undercloud)$ metalsmith list
```

您还可以使用 **openstack baremetal allocation** 命令来验证节点与主机名的关联：

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

当节点置备成功时，您可以部署 overcloud。如需更多信息，请参阅使用预置备节点配置基本 overcloud。

8.4. 扩展裸机节点

要增加现有 overcloud 中的裸机节点数量，请在 **~/overcloud-baremetal-deploy.yaml** 文件中增加节点数并重新部署 overcloud。

先决条件

- 成功安装 undercloud。如需更多信息，请参阅安装 director。
- 成功部署 overcloud。如需更多信息，请参阅使用预置备节点配置基本 overcloud。
- 裸机节点已内省并可用于置备和部署。有关更多信息，请参阅为 overcloud 注册节点 和 创建裸机节点硬件清单。

流程

1. 查找 **stackrc** undercloud 凭据文件：

```
$ source ~/stackrc
```

2. 编辑用于置备裸机节点的 **~/overcloud-baremetal-deploy.yaml** 文件，并为您要扩展的角色增加 **count** 参数。例如，如果 overcloud 包含三个计算节点，请使用以下内容将 Compute 节点数增加到 10：

```
- name: Controller
  count: 3
- name: Compute
  count: 10
```

您还可以在 **instances** 参数中添加预先节点放置。有关可用参数和属性的更多信息，请参阅裸机节点置备属性。

3. 运行置备命令，指定 **~/overcloud-baremetal-deploy.yaml** 文件，并使用 **--output** 选项定义输出文件：

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

4. 使用 **openstack baremetal node list** 命令监控置备进度。

5. 部署 overcloud，包括置备命令生成的 **~/overcloud-baremetal-deployed.yaml** 文件，以及其他与部署相关的环境文件：

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

8.5. 缩减裸机节点

在 **~/overcloud-baremetal-deploy.yaml** 文件中标记您要从堆栈中删除的节点，重新部署 overcloud，然后使用 **--baremetal-deployment** 选项将该文件包含在 **openstack overcloud node delete** 命令中。

先决条件

- 成功安装 undercloud。更多信息请参阅 第 4 章 在 undercloud 上安装 director。
- 成功部署 overcloud。更多信息请参阅 第 9 章 使用预置备节点配置基本 overcloud。
- 至少一个要从堆栈中删除的裸机节点。

步骤

1. 查找 **stackrc** undercloud 凭据文件：

```
$ source ~/stackrc
```

2. 编辑用于置备裸机节点的 **~/overcloud-baremetal-deploy.yaml** 文件，并减少您要缩减的角色的 **count** 参数。还必须为要从堆栈中删除的每个节点定义以下属性：

   - 节点的名称。
   - 与节点关联的主机名。
   - 属性 **provisioned: false**
     例如，要从堆栈中删除节点 **overcloud-controller-1**，请在 **~/overcloud-baremetal-deploy.yaml** 文件中包括以下内容：

     ```
     - name: Controller
       count: 2
       instances:
       - hostname: overcloud-controller-0
         name: node00
       - hostname: overcloud-controller-1
         name: node01
         # Removed from cluster due to disk failure
         provisioned: false
       - hostname: overcloud-controller-2
         name: node02
     ```

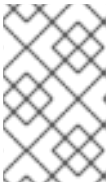
3. 运行置备命令，指定 **~/overcloud-baremetal-deploy.yaml** 文件，并使用 **--output** 选项定义输出文件：

   ```
   (undercloud)$ openstack overcloud node provision \
   --stack stack \
   --output ~/overcloud-baremetal-deployed.yaml \
   ~/overcloud-baremetal-deploy.yaml
   ```

4. 重新部署 overcloud，并包含置备命令生成的 **~/overcloud-baremetal-deployed.yaml** 文件，以及与部署相关的任何其他环境文件：

   ```
   (undercloud)$ openstack overcloud deploy \
     ...
     -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
     -e ~/overcloud-baremetal-deployed.yaml \
     --deployed-server \
     --disable-validations \
     ...
   ```

   重新部署 overcloud 后，堆栈中不再存在使用 **provisioned: false** 属性定义的节点。但是，这些节点仍然以置备状态运行。

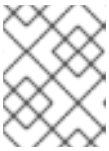
   

   注意

   如果要临时从堆栈中删除节点，您可以使用 **provisioned: false** 属性部署 overcloud，然后使用 **provisioned: true** 属性重新部署 overcloud，以将节点返回到堆栈。

5. 运行 **openstack overcloud node delete** 命令，包括带有 **--baremetal-deployment** 选项的 **~/overcloud-baremetal-deploy.yaml** 文件。

```
(undercloud)$ openstack overcloud node delete \
--stack stack \
--baremetal-deployment ~/overcloud-baremetal-deploy.yaml
```



注意

不要将您要从堆栈中删除的节点作为命令参数包括在 **openstack overcloud node delete** 命令中。

8.6. 裸机节点置备属性

使用下面的表格来了解在使用 **openstack baremetal node provision** 命令置备裸机节点时可使用的参数、属性和值。

表 8.1. 角色参数

参数	价值
name	必填角色名称
数量	要为这个角色置备的节点数量。默认值为：**1**。
默认值	**instances**条目属性的默认值字典。**instances**条目属性覆盖您在 **defaults** 参数中指定的任何默认值。
实例	用于为特定节点指定属性的值的字典。有关 **instances** 参数中支持的属性的更多信息，请参阅表 8.2 “**instances** 和 **defaults** 参数”。此列表的长度不得超过 **count** 参数的值。
hostname_format	覆盖这个角色的默认主机名格式。默认格式使用小写角色名称。例如，Controller 角色的默认格式为 **%stackname%-controller-%index%**。只有 Compute 角色不会遵循角色名称规则。Compute 的默认格式为 **%stackname%-novacompute-%index%**

示例语法

在以下示例中，**name** 指的是节点的逻辑名称，**hostname** 是指从 overcloud 堆栈名称、角色和增量索引中生成的主机名。所有 Controller 服务器都使用默认自定义镜像 **overcloud-full-custom**，并位于预测的节点上。其中一个计算服务器可预测放置在 **node04** 上，其自定义主机名为 **overcloud-compute-special**，其他 99 个计算服务器则位于从可用节点池中自动分配的节点上：

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
```

```
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 100
  instances:
  - hostname: overcloud-compute-special
    name: node04
```

表 8.2. instances 和 defaults 参数

参数	价值
主机名	如果主机名与 **hostname_format** 模式兼容，那么其他属性则适用于分配给这个主机名的节点。否则，您可以对该节点使用自定义主机名。
name	要置备的节点的名称。
镜像	要在节点上置备的镜像的详情。有关 **image** 参数中支持的属性的更多信息，请参阅 表 8.3 “**image**参数”。
功能	选择与节点功能匹配的条件。
NIC	代表请求的 NIC 的字典列表。有关在 **nics** 参数中支持的属性的更多信息，请参阅 表 8.4 “**nic** 参数”。
配置集	使用高级配置集匹配的选择条件。
已置备	确定此节点置备或取消置备的布尔值。默认值为 **true**。使用 **false** 取消置备节点。如需更多信息，请参阅 缩减裸机节点。
resource_class	与节点的资源类匹配的选择条件。默认值为 **baremetal**。
root_size_gb	GiB 中根分区的大小。默认值为 **49**
swap_size_mb	MiB 中 swap 分区的大小。
遍历	作为与节点遍历匹配的选择条件的遍历列表。

示例语法

在以下示例中，所有 Controller 服务器都使用自定义的默认 overcloud 镜像 ***overcloud-full-custom****。Controller 服务器* ***overcloud-controller-0*** *可预先放置在* ***node00*** *上，并具有自定义的根和 swap 大小。其他两个 Controller 服务器位于从可用节点池中自动分配的节点上，并且具有默认的根和 swap 大小：*

```
- name: Controller
  count: 3
  defaults:
    image:
```

```
    href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
instances:
- hostname: overcloud-controller-0
  name: node00
  root_size_gb: 140
  swap_size_mb: 600
```

*表 8.3. **image参数***

参数	价值
href	glance 镜像引用，或根分区，或整个磁盘镜像的 URL。支持的 URL 方案有 **file:**//、**http:**// 和 **https:**//。如果该值不是有效的 URL，则该值必须是有效的 glance 镜像引用。
checksum	当 href 是 URL 时，这个值必须是根分区的 SHA512 校验和或者整个磁盘镜像。
内核	内核镜像的 glance 镜像引用或者 URL。仅在分区镜像中使用此属性。
ramdisk	ramdisk 镜像的 glance 镜像引用或 URL。仅在分区镜像中使用此属性。

示例语法

*在以下示例中，所有三个 Controller 服务器都位于从可用节点池中自动分配的节点上。此环境中的所有 Controller 服务器都使用默认自定义镜像 **overcloud-full-custom**：*

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      checksum: 1582054665
      kernel: file:///var/lib/ironic/images/overcloud-full-custom.vmlinuz
      ramdisk: file:///var/lib/ironic/images/overcloud-full-custom.initrd
```

*表 8.4. **nic 参数***

参数	价值
fixed_ip	要用于此 NIC 的特定 IP 地址。
network	要为此 NIC 创建端口的 neutron 网络。
子网	要为此 NIC 创建端口的 neutron 子网。
端口	使用现有的 Neutron 端口而不是创建新端口。

示例语法

在以下示例中，所有三个 Controller 服务器都位于从可用节点池中自动分配的节点上。此环境中的所有 Controller 服务器都使用默认的自定义镜像 ***overcloud-full-custom****，并具有特定的网络要求：*

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      nics:
        network: custom-network
        subnet: custom-subnet
```

第 9 章 使用预置备节点配置基本 OVERCLOUD

本章包含基本配置流程，可用于使用预置备节点配置 Red Hat OpenStack Platform (RHOSP) 环境。这种情境与标准 overcloud 创建情境有所不同，具体体现在以下几个方面：

- 您可以使用外部工具置备节点，让 director 仅控制 overcloud 的配置。
- 您无需依赖 director 的置备方法，可直接使用节点。如果您想创建没有电源管理控制的 overcloud 或使用具有 DHCP/PXE 引导限制的网络，该方法很有用。
- director 不使用 OpenStack Compute (nova)、OpenStack Bare Metal (ironic) 或者 OpenStack Image (glance) 来管理节点。
- 预置备的节点可以使用不依赖于 QCOW2 overcloud-full 镜像的自定义分区布局。

这种情境仅包括没有自定义功能的基本配置。但是，您可以在这个基本的 overcloud 中添加高级配置选项并使用高级 Overcloud 自定义指南中的说明进行自定义，以符合您的规格。

重要

您无法将预置备节点与 director 置备的节点合并。

9.1. 预置备节点要求

在开始使用预置备节点部署 overcloud 前，请确保环境中存在以下配置：

- 您在 第 4 章 在 undercloud 上安装 director 中创建的 director 节点。
- 节点的一组裸机。所需的节点数量由您需要创建的 overcloud 类型所决定。这些机器必须符合为每个节点类型设定的要求。这些节点需要 Red Hat Enterprise Linux 8.4 作为主机操作系统安装。红帽建议使用最新的可用版本。
- 用于管理预置备节点的一个网络连接。本情境需要具备对节点的不间断 SSH 访问权限以进行编配代理配置。
- Control Plane 网络的一个网络连接。这种网络具备两种主要情境：
  - 将 Provisioning 网络用作 Control Plane，这种是默认情境。这个网络通常是从预置备节点到 director 的第 3 层 (L3) 可路由网络连接。本情境示例使用以下 IP 地址分配：

    表 9.1. Provisioning 网络 IP 分配信息

节点名	IP 地址
Director	192.168.24.1
控制器 0	192.168.24.2
计算 0	192.168.24.3

  - 使用单独的网络。如果 director 的 Provisioning 网络是私有非路由网络，则可从任何子网定义节点的 IP 地址，通过公共 API 端点与 director 通信。有关此情况的要求的详情请参考 第 9.6 节 "为预置备节点使用单独网络"。

- 本示例中的所有其他网络类型使用 Control Plane 网络来提供 OpenStack 服务。不过，您可以为其他网络流量类型创建额外的网络。
- 如果有任何节点使用 Pacemaker 资源，服务用户 **hacluster** 和服务组 **haclient** 必须具有 189 的 UID/GID。这是因为 CVE-2018-16877。如果与操作系统一起安装了 Pacemaker，则安装会自动创建这些 ID。如果错误设置 ID 值，请按照文章 OpenStack 小幅更新/快进升级可能在 controller 节点上在 pacemaker 步骤失败，显示消息"Could not evaluate: backup_cib" 中的步骤来更改 ID 值。
- 要防止某些服务绑定到不正确的 IP 地址并导致部署失败，请确保 **/etc/hosts** 文件不包含 **node-name=127.0.0.1** 映射。

9.2. 在预置备节点上创建用户

使用预置备节点配置 overcloud 时，director 需要对 overcloud 节点进行 SSH 访问。在预置备的节点上，您必须创建一个使用 SSH 密钥身份验证的用户，并为该用户配置免密码 sudo 访问。在预置备的节点上创建用户后，可以结合使用 **--overcloud-ssh-user** 和 **--overcloud-ssh-key** 选项及 **openstack overcloud deploy** 命令，创建具有预置备节点的 overcloud。

默认情况下，overcloud SSH 用户和 overcloud SSH 密钥的值是 **stack** 用户和 **~/.ssh/id_rsa**。要创建 **stack** 用户，请完成以下步骤。

步骤

1. 在每个 overcloud 节点上，创建 **stack** 用户并设定密码。例如，在 Controller 节点上运行以下命令：

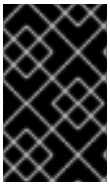
```
[root@controller-0 ~]# useradd stack
[root@controller-0 ~]# passwd stack  # specify a password
```

2. 进行以下操作以使这个用户使用 **sudo** 时不需要输入密码：

```
[root@controller-0 ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
/etc/sudoers.d/stack
[root@controller-0 ~]# chmod 0440 /etc/sudoers.d/stack
```

3. 在所有预置备节点上创建并配置 **stack** 用户后，将 **stack** 用户的公共 SSH 密钥从 director 节点复制到每个 overcloud 节点。例如，要将 director 的公共 SSH 密钥复制到 Controller 节点，请运行以下命令：

```
[stack@director ~]$ ssh-copy-id stack@192.168.24.2
```



重要

要复制 SSH 密钥，您可能必须在每个 overcloud 节点的 SSH 配置中临时设置 **PasswordAuthentication Yes**。复制 SSH 密钥后，设置 **PasswordAuthentication No**，并在以后使用 SSH 密钥进行身份验证。

9.3. 为预置备节点注册操作系统

每个节点需要具备访问红帽订阅的权限。在每个节点上完成以下步骤，将节点注册到 Red Hat Content Delivery Network 中。

重要

仅启用列出的软件仓库。其他软件仓库可能导致软件包和软件冲突。请勿启用任何其他软件仓库。

步骤

1. 运行注册命令，按照提示输入您的客户门户用户名和密码：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager register
```

2. 查找 Red Hat OpenStack Platform 16.2 的权利池：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager list --available \
 --all --matches="Red Hat OpenStack"
```

3. 使用上一步中的池 ID 以附加 Red Hat OpenStack Platform 16 权利：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager attach --pool=pool_id
```

4. 禁用所有默认软件仓库：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --disable=*
```

5. 启用所需的 Red Hat Enterprise Linux 软件仓库。

- 对于 x86_64 系统：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos \
 --enable=rhel-8-for-x86_64-baseos-eus-rpms \
 --enable=rhel-8-for-x86_64-appstream-eus-rpms \
 --enable=rhel-8-for-x86_64-highavailability-eus-rpms \
 --enable=ansible-2.9-for-rhel-8-x86_64-rpms \
 --enable=openstack-16.2-for-rhel-8-x86_64-rpms \
 --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

- 对于 POWER 系统：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos \
 --enable=rhel-8-for-ppc64le-baseos-rpms \
 --enable=rhel-8-for-ppc64le-appstream-rpms \
 --enable=rhel-8-for-ppc64le-highavailability-rpms \
 --enable=ansible-2.8-for-rhel-8-ppc64le-rpms \
 --enable=openstack-16-for-rhel-8-ppc64le-rpms \
 --enable=fast-datapath-for-rhel-8-ppc64le-rpms
```

6. 将 **container-tools** 软件仓库模块设置为版本 **3.0**：

```
[heat-admin@controller-0 ~]$ sudo dnf module disable -y container-tools:rhel8
[heat-admin@controller-0 ~]$ sudo dnf module enable -y container-tools:3.0
```

7. 如果 overcloud 使用 Ceph Storage 节点，请启用相关的 Ceph Storage 软件仓库：

```
[heat-admin@cephstorage-0 ~]$ sudo subscription-manager repos \
 --enable=rhel-8-for-x86_64-baseos-rpms \
 --enable=rhel-8-for-x86_64-appstream-rpms \
 --enable=ansible-2.9-for-rhel-8-x86_64-rpms \
 --enable=openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms
```

8. 锁定除 Red Hat Ceph Storage 节点外的所有 overcloud 节点上的 RHEL 版本：

```
[heat-admin@controller-0 ~]$ sudo subscription-manager release --set=8.4
```

9. 更新您的系统，确保您具备最新的基础系统软件包：

```
[heat-admin@controller-0 ~]$ sudo dnf update -y
[heat-admin@controller-0 ~]$ sudo reboot
```

节点现在可供您的 overcloud 使用。

9.4. 配置对 DIRECTOR 的 SSL/TLS 访问权限

如果 director 使用 SSL/TLS，那么预置备节点需要用于签署 director 的 SSL/TLS 证书的证书授权机构文件。如果使用自己的证书颁发机构（CA），请在每个 overcloud 节点上执行以下操作。

步骤

1. 将证书授权机构文件复制到各预置备节点上的 **/etc/pki/ca-trust/source/anchors/** 目录中。
2. 在每个 overcloud 节点上运行以下命令：

```
[root@controller-0 ~]#  sudo update-ca-trust extract
```

这些步骤将确保 overcloud 节点可以通过 SSL/TLS 访问 director 的公共 API。

9.5. 配置 CONTROL PLANE 网络

预置备 overcloud 节点使用标准 HTTP 请求从 director 获取元数据。这表示所有 overcloud 节点都需要 L3 权限来访问以下之一：

- director 的 Control Plane 网络，这是在 **undercloud.conf** 文件中使用 **network_cidr** 参数定义的子网。overcloud 节点需要对该子网的直接访问权限或对该子网的可路由访问权限。
- director 的公共 API 端点，使用 **undercloud.conf** 文件中的 **undercloud_public_host** 参数指定。如果没有指向 Control Plane 的 L3 路由或希望使用 SSL/TLS 通信，则此选项可用。有关将 overcloud 节点配置为使用公共 API 端点的更多信息，请参阅 第 9.6 节 "为预置备节点使用单独网络"。

director 使用 Control Plane 网络管理和配置标准 overcloud。对于具有预置备节点的 overcloud，可能需要对您的网络配置进行修改以适应 director 和预置备节点之间的通信。

使用网络隔离

您可以使用网络隔离分组服务以使用特定网络，包括 Control Plane。高级 Overcloud 自定义指南中有多个网络隔离策略。您可以为 Control Plane 上的节点定义特定 IP 地址。有关隔离网络和创建可预测节点放置策略的更多信息，请参阅高级 Overcloud 自定义指南中的以下章节：

- *"基本网络隔离"*
- *"Controlling Node Placement"*

注意

如果使用网络隔离，请确保您的 NIC 模板不包括用于 undercloud 访问的 NIC。这些模板可能会重新配置 NIC，这会导致在部署过程中出现连接和配置问题。

分配 IP 地址

*如果不使用网络隔离，则可使用一个 Control Plane 网络管理所有服务。这需要手动配置每个节点的 Control Plane NIC，以便使用 Control Plane 网络范围内的 IP 地址。如果将 director 的 Provisioning 网络用作 Control Plane，请确保所选的 overcloud IP 地址不在置备（**dhcp_start** 和 **dhcp_end**）和内省（**inspection_iprange**）DHCP 范围之内。*

在创建标准 overcloud 期间，director 将创建 OpenStack Networking (neutron) 端口并为 Provisioning/Control Plane 网络上的 overcloud 节点自动分配 IP 地址。但是，这可能导致 director 分配的地址与您为每个节点手动配置的 IP 地址不同。在这种情况下，使用可预测的 IP 地址策略来强制 director 使用 Control Plane 上的预置备 IP 分配。

*例如，您可以使用具有以下 IP 分配的环境文件 **ctlplane-assignments.yaml** 来实现一个可预测的 IP 策略：*

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml

parameter_defaults:
  DeployedServerPortMap:
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.2
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          192.168.24.0/24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          - 192.168.24.0/24
```

*在本示例中，**OS::TripleO::DeployedServer::ControlPlanePort** 资源将一组参数传递至 director，并为预置备的节点定义 IP 分配。使用 **DeployedServerPortMap** 参数定义与每个 overcloud 节点对应的 IP 地址和子网 CIDR。映射定义以下属性：*

1. *分配名称，其格式为 **<node_hostname>-<network>**，其中 **<node_hostname>** 值与节点的短主机名匹配，**<network>** 与网络的小写名称匹配。例如：**controller-0-ctlplane** 用于 **controller-0.example.com**，**compute-0-ctlplane** 用于 **compute-0.example.com**。*

2. *IP 分配，它采用以下参数形式：*

- **fixed_ips/ip_address** - 为 control plane 指定固定的 IP 地址。使用 **ip_address** 列表中的多个参数来定义多个 IP 地址。
- **subnets/cidr** - 定义子网的 CIDR 值。

本章后面部分使用生成的环境文件 (**ctlplane-assignments.yaml**) 作为 **openstack overcloud deploy** 命令的一部分。

9.6. 为预置备节点使用单独网络

默认情况下，director 使用 Provisioning 网络作为 overcloud Control Plane。但是，如果此网络被隔离且不可路由，则节点在配置期间不能与 director 的内部 API 通信。在这种情况下，您可能需要为节点指定一个单独的网络，并进行配置，以便通过公共 API 与 director 通信。

对于此情境，有几个需要满足的要求：

- overcloud 节点必须容纳来自 第 9.5 节 "配置 control plane 网络" 的基本网络配置。
- 您必须在 director 上启用 SSL/TLS，以便使用公共 API 端点。如需更多信息，请参阅 第 4.2 节 "Director 配置参数" 和 第 20 章 配置自定义 SSL/TLS 证书。
- 您必须为 director 指定一个可访问的完全限定域名（FQDN）。此 FQDN 必须解析为 director 的可路由 IP 地址。使用 **undercloud.conf** 文件中的 **undercloud_public_host** 参数来设置这个 FQDN。

本节中的示例使用了与主要情境不同的 IP 地址分配：

表 9.2. Provisioning 网络 IP 分配信息

节点名	IP 地址或 FQDN
Director（内部 API）	192.168.24.1 （Provisioning 网络和 Control Plane）
Director（公共 API）	10.1.1.1 / director.example.com
overcloud 虚拟 IP	192.168.100.1
控制器 0	192.168.100.2
计算 0	192.168.100.3

以下章节为需要单独的 overcloud 节点网络的情境提供额外配置。

IP 地址分配

IP 分配方法类似于 第 9.5 节 "配置 control plane 网络"。但是，由于无法从部署的服务器路由到 Control Plane，您必须使用 **DeployedServerPortMap** 参数从所选的 overcloud 节点子网分配 IP 地址，包括访问 Control Plane 的虚拟 IP 地址。以下示例是容纳此网络架构的 第 9.5 节 "配置 control plane 网络" 中的 **ctlplane-assignments.yaml** 环境文件的修改版本：

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-templates/deployed-server/deployed-neutron-port.yaml
```

```
  OS::TripleO::Network::Ports::ControlPlaneVipPort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml (1)

parameter_defaults:
  NeutronPublicInterface: eth1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
      subnets:
        - cidr: 24
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
      subnets:
        - cidr: 24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.3
      subnets:
        - cidr: 24
```

(1) **RedisVipPort** 和 **OVNDBsVipPort** 资源映射到 **network/ports/noop.yaml**。此映射是必需的，因为默认的 Redis 和 OVNDB VIP 地址来自 Control Plane。在这种情况下，使用 **noop** 来禁用这种 Control Plane 映射。

9.7. 映射预置备的节点主机名

配置预置备节点时，必须将基于 heat 的主机名映射到实际主机名，以便 **ansible-playbook** 能够到达可解析主机。请使用 **HostnameMap** 来映射这些值。

步骤

1. 创建环境文件，例如 **hostname-map.yaml**，并包括 **HostnameMap** 参数和主机名映射。请遵循以下语法：

```
parameter_defaults:
  HostnameMap:
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
```

[HEAT HOSTNAME] 通常符合以下惯例：**[STACK NAME]-[ROLE]-[INDEX]**：

```
parameter_defaults:
  HostnameMap:
    overcloud-controller-0: controller-00-rack01
    overcloud-controller-1: controller-01-rack02
    overcloud-controller-2: controller-02-rack03
```

```
overcloud-novacompute-0: compute-00-rack01
overcloud-novacompute-1: compute-01-rack01
overcloud-novacompute-2: compute-02-rack01
```

2. 保存 **hostname-map.yaml** 文件。

9.8. 为预置备节点配置 CEPH STORAGE

在 undercloud 主机上完成以下步骤，为已部署的节点配置 **ceph-ansible**。

步骤

1. 在 undercloud 主机上，创建环境变量 **OVERCLOUD_HOSTS**，并将该变量设置为您要用作 Ceph 客户端的 overcloud 主机的 IP 地址列表（以空格分隔）：

```
export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"
```

2. 默认的 overcloud 计划名称是 **overcloud**。如果使用其他名称，请创建一个环境变量 **OVERCLOUD_PLAN** 来存储您的自定义名称：

```
export OVERCLOUD_PLAN="<custom-stack-name>"
```

   - 将 **<custom-stack-name>** 替换为堆栈的名称。

3. 运行 **enable-ssh-admin.sh** 脚本，以在 Ansible 可用于配置 Ceph 客户端的 overcloud 节点上配置用户：

```
bash /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/enable-ssh-admin.sh
```

运行 **openstack overcloud deploy** 命令时，Ansible 配置您在 **OVERCLOUD_HOSTS** 变量中定义为 Ceph 客户端的主机。

9.9. 利用预置备节点创建 OVERCLOUD

overcloud 部署使用 第 7.14 节 "部署命令" 中的标准 CLI 方法。对于预置备的节点，该部署命令需要使用来自核心 heat 模板集的一些额外选项和环境文件：

- **--disable-validations** - 使用此选项禁止对未用于预置备基础架构的服务执行基本的 CLI 验证功能。如果不禁止这些验证，部署将失败。
- **environments/deployed-server-environment.yaml** - 包含此环境文件以创建和配置预置备基础架构。这种环境文件用 **OS::Heat::DeployedServer** 资源代替 **OS::Nova::Server** 资源。

以下命令是示例 overcloud 部署命令，且环境文件特定于预置备的架构：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user stack \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  <OTHER OPTIONS>
```

-

--overcloud-ssh-user 和 **--overcloud-ssh-key** 选项用于在配置阶段通过 SSH 连接每个 overcloud 节点，创建初始 **tripleo-admin** 用户，以及将 SSH 密钥注入 **/home/tripleo-admin/.ssh/authorized_keys**。要注入 SSH 密钥，请为初始 SSH 连接指定包含 **--overcloud-ssh-user** 和 **--overcloud-ssh-key**（默认为 **~/.ssh/id_rsa**）的凭证。为了避免暴露使用 **--overcloud-ssh-key** 选项指定的私钥，director 不会将该密钥传递给任何 API 服务，如 heat 或 Workflow 服务 (mistral)，而只有 director 的 **openstack overcloud deploy** 命令使用此密钥为 **tripleo-admin** 用户启用访问权限。

9.10. OVERCLOUD 部署输出

overcloud 创建完毕后，director 会提供为配置 overcloud 而执行的 Ansible play 的总结：

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director 还会提供用于访问 overcloud 的详细信息。

```
Ansible passed.
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

9.11. 访问 OVERCLOUD

director 会生成脚本来配置和帮助认证 undercloud 与 overcloud 的交互。director 将此文件保存为 **stack** 用户主目录的 **overcloudrc** 文件。运行以下命令来使用此文件：

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

此命令加载从 undercloud CLI 与 overcloud 交互所需的环境变量。命令提示符的变化会显示当前状态：

```
(overcloud) $
```

要返回与 undercloud 进行交互的状态，请运行以下命令：

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

9.12. 扩展预置备节点

扩展预置备节点的流程与 第 16 章 扩展 overcloud 节点 中的标准扩展流程类似。但是，添加新预置备节点的流程却不相同，这是因为预置备节点不从 OpenStack Bare Metal (ironic) 和 OpenStack Compute (nova) 使用标准注册和管理流程。

扩展预置备节点

使用预置备节点扩展 overcloud 时，必须在每个节点上配置编配代理以对应 director 的节点计数。

执行以下操作以扩展 overcloud 节点：

1. 根据 第 9.1 节 "预置备节点要求" 准备新的预置备节点。
2. 扩展节点。更多信息请参阅 第 16 章 扩展 overcloud 节点。
3. 在执行部署命令后，等待 director 创建新的节点资源并启动配置。

缩减预置备节点

使用预置备节点缩减 overcloud 时，请按照 第 16 章 扩展 overcloud 节点 中的缩减说明操作。

在缩减操作中，您可以为 OSP 置备或预置备节点使用主机名。您也可以将 UUID 用于 OSP 置备的节点。但是，预先提供的节点没有 UUID，因此您始终使用主机名。将 hostname 或 UUID 值传递给 **openstack overcloud node delete** 命令。

流程

1. 找出您要删除的节点的名称。

```
$ openstack stack resource list overcloud -n5 --filter
type=OS::TripleO::ComputeDeployedServerServer
```

2. 将 **stack_name** 列中对应的节点名称传递给 **openstack overcloud node delete** 命令：

```
$ openstack overcloud node delete --stack <overcloud> <stack>
```

   - 将 **<overcloud>** 替换为 overcloud 堆栈的名称或 UUID。
   - 将 &lt **;stack_name** > 替换为您要删除的节点的名称。您可以在 **openstack overcloud node delete** 命令中包含多个节点名称。

3. 确保 **openstack overcloud node delete** 命令已运行完：

```
$ openstack stack list
```

   当删除操作完成后，**overcloud** 堆栈的状态会显示 **UPDATE_COMPLETE**。

从堆栈中移除 overcloud 节点之后，关闭这些节点。在标准部署中，director 上的裸机服务控制此功能。但是，如果使用预置备节点，您必须手动关闭这些节点，或使用每个物理系统的电源管理控制。从堆栈中移除节点之后，如果您不关闭它们，它们可能保持运行，并作为 overcloud 环境的组成部分重新连接。

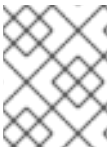
关闭移除的节点后，将其重新置备为基础操作系统配置，以免它们未来意外加入 overcloud。

注意

在将之前已经从 overcloud 移除的节点重新部署到新的基础操作系统之前，不要尝试再次使用它们。缩减流程只从 overcloud 堆栈移除节点，不会卸载任何软件包。

移除预置备 overcloud

要删除使用预置备节点的整个 overcloud，请参阅 第 12.6 节 "移除 overcloud" 获取标准 overcloud 移除流程。移除 overcloud 后，关闭所有节点并将其重新置备为基础操作系统配置。



注意

在将之前已经从 overcloud 移除的节点重新部署到新的基础操作系统之前，不要尝试再次使用它们。移除流程只删除 overcloud 堆栈，不会卸载任何软件包。

第 10 章 部署多个 OVERCLOUD

重要

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

您可以使用一个 undercloud 节点部署并管理多个 overcloud。每个 overcloud 都是唯一的 heat 栈，不共享堆栈资源。如果环境中 undercloud 和 overcloud 的比率为 1:1 时会造成无法管理的 overclound 数量时，可以使用该方法。例如，Edge、多站点和多产品环境。

多个 overcloud 部署中的 overcloud 环境是完全独立的，您可以使用 **source** 命令在环境之间进行切换。每个 overcloud 都有一个唯一的凭据文件，它由部署过程创建。要与 overcloud 交互，您必须提供适当的凭据文件。

如果将裸机置备服务(ironic)用于裸机置备，则所有 overcloud 必须位于同一置备网络中。如果无法使用同一置备网络，则可使用部署的服务器方法使用路由的网络部署多个 overcloud。在这种情况下，您必须确保 **HostnameMap** 参数中的值与每个 overcloud 的堆栈名称匹配。

要在单个 undercloud 上部署多个 overcloud，您必须执行以下任务：

1. 部署 undercloud。如需更多信息，请参阅 section I. Director 安装和配置。
2. 部署第一个 overcloud。如需更多信息，请参阅 合作伙伴 II。基本的 overcloud 部署。
3. 通过为新 overcloud 创建新的环境文件，并在部署命令中指定核心 heat 模板以及新的配置文件和新堆栈 **名称来** 部署额外的 overcloud。

10.1. 部署额外 OVERCLOUD

您可以在单个 undercloud 上部署多个 overcloud。以下流程演示了如何在现有 overcloud (**overcloud-one**)的现有 Red Hat OpenStack Platform (RHOSP) 部署中创建和部署新的 **overcloud-two**。

先决条件

- undercloud。
- 一个或多个 overcloud。
- 可用于额外 overcloud 的节点。
- 为额外 overcloud 自定义网络，使得每个 overcloud 在生成的堆栈中都有唯一的网络。

步骤

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

```
$ source ~/stackrc
```

3. 为您想部署的额外 overcloud 创建新目录：

```
(undercloud)$ mkdir ~/overcloud-two
```

4. *将 **network_data.yaml** 文件从现有 overcloud 复制到额外 overcloud 的新目录中：*

```
(undercloud)$ cp network_data.yaml ~/overcloud-two/network_data.yaml
```

5. *打开 ~/**overcloud-two**/**network_data.yaml** 文件，并将 **name_lower** 更新至额外 overcloud 网络的唯一名称：*

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  ...
```

6. *添加 **service_net_map_replace** （如果不存在），并将值设为额外 overcloud 网络的默认值：*

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  service_net_map_replace: internal_api
```

7. *指定额外 overcloud 上每个子网的 VLAN ID：*

```
- name: InternalApi
  ...
  vip: true
  vlan: 21
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '172.21.0.4', 'end': '172.21.0.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2001::10', 'end':
'fd00:fd00:fd00:2001:ffff:ffff:ffff:fffe'}]
  mtu: 1500
- name: Storage
  ...
```

8. *指定 **overcloud-two** 外部网络网关的 IP 地址：*

```
- name: External
  ...
  gateway_ip: <ip_address>
  ...
```

   - *将 **<ip_address>** 替换为 **overcloud-two** 外部网络网关的 IP 地址，例如 **10.0.10.1**。*

9. *为额外 overcloud 创建网络配置文件，该文件会覆盖 **/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml** 文件中提供的默认隔离网络配置，如 **network_overrides.yaml**。*

10. *打开 ~/**overcloud-two**/**network_overrides.yaml** 文件，并添加 **overcloud-two** DNS 服务器的 IP 地址：*

```
parameter_defaults:
  ...
  DnsServers:
    - <ip_address>
  ...
```

- 将 **<ip_address>** 替换为 **overcloud-two** DNS 服务器的 IP 地址，例如 **10.0.10.2**。

11. 如果您的部署使用可预测的 IP 地址，请在新的网络 IP 地址映射文件 **ips-from-pool- overcloud-two.yaml** 中为 **overcloud-two** 节点配置 IP 地址：

```
parameter_defaults:
  ControllerIPs:
  ...
    internal_api_cloud_2:
    - 192.168.1.10
    - 192.168.1.11
    - 192.168.1.12
    ...
    external_cloud_2:
    - 10.0.1.41
    ...
```

12. 使用其他环境文件将 **overcloud-two** 环境文件添加到堆栈中，并部署额外的 overcloud：

```
(undercloud)$ openstack overcloud deploy --templates \
  --stack overcloud-two \
  -n ~/overcloud-two/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml
\
  -e ~/overcloud-two/network_overrides.yaml \
  -e [your environment files] \
  ...
```

部署过程创建 **overcloud-tworc**，用于与 **overcloud-two** 交互。

13. 要与其他 overcloud 交互，请提供 overcloud 凭证文件：

```
$ source overcloud-tworc
```

10.2. 管理多个 OVERCLOUD

您部署的每个 overcloud 都使用一组相同的核心 heat 模板 **/usr/share/openstack-tripleo-heat-templates**。红帽建议不要修改或复制这些模板，因为使用非标准核心模板集可能引入更新和升级问题。

相反，在部署或维护多个 overcloud 时，为了便于管理，创建特定于每个云的环境文件的单独目录。为每个云运行部署命令时，将核心 heat 模板与您单独创建的特定于云的环境文件一起包括在内。例如，为 undercloud 和两个 overcloud 创建以下目录：

~stack/undercloud

包含特定于 undercloud 的环境文件。

~stack/overcloud-one

包含特定于第一个 overcloud 的环境文件。

~stack/overcloud-two

包含特定于第二个 overcloud 的环境文件。

*部署或重新部署 **overcloud-one** 或 **overcloud-two** 时，使用 **--templates** 选项在部署命令中包括核心 heat 模板，然后从云特定的环境文件目录中指定任何额外环境文件。*

或者，在版本控制系统中创建软件仓库，并对每个部署使用分支。有关更多信息，请参阅高级 Overcloud 自定义指南中的使用自定义核心 Heat 模板部分。

使用以下命令查看可用的 overcloud 计划列表：

```
$ openstack overcloud plan list
```

使用以下命令查看当前部署的 overcloud 列表：

```
$ openstack stack list
```

第 11 章 执行 OVERCLOUD 安装后任务

本章介绍创建 overcloud 后要立即执行的任务。这些任务可确保您的 overcloud 随时可用。

11.1. 检查 OVERCLOUD 部署状态

要检查 overcloud 的部署状态，可使用 **openstack overcloud status** 命令。此命令返回所有部署步骤的结果。

步骤

1. Source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 运行部署状态命令：

```
$ openstack overcloud status
```

此命令的输出显示 overcloud 的状态：

```
+-----------+---------------------+---------------------+-------------------+
| Plan Name |       Created       |       Updated       | Deployment Status |
+-----------+---------------------+---------------------+-------------------+
| overcloud | 2018-05-03 21:24:50 | 2018-05-03 21:27:59 |   DEPLOY_SUCCESS  |
+-----------+---------------------+---------------------+-------------------+
```

如果您的 overcloud 使用其他名称，请使用 **--plan** 参数来选择具有不同名称的 overcloud：

```
$ openstack overcloud status --plan my-deployment
```

11.2. 创建基本 OVERCLOUD 类别

本指南中的步骤假定您的安装中包含有类型 (flavor)。如果您尚未创建任何 flavor，请完成以下步骤，创建一组具有多种存储和处理功能的基本默认 flavor：

步骤

1. 获取 **overcloudrc** 文件：

```
$ source ~/overcloudrc
```

2. 运行 **openstack flavor create** 命令以创建类别。使用以下选项指定每种类别的硬件要求：

--disk
为虚拟机卷定义硬盘空间。

--ram
定义虚拟机所需的 RAM。

--vcpus
定义虚拟机的虚拟 CPU 的数量。

3. 以下示例创建默认 overcloud 类别：

```
$ openstack flavor create m1.tiny --ram 512 --disk 0 --vcpus 1
$ openstack flavor create m1.smaller --ram 1024 --disk 0 --vcpus 1
$ openstack flavor create m1.small --ram 2048 --disk 10 --vcpus 1
$ openstack flavor create m1.medium --ram 3072 --disk 10 --vcpus 2
$ openstack flavor create m1.large --ram 8192 --disk 10 --vcpus 4
$ openstack flavor create m1.xlarge --ram 8192 --disk 10 --vcpus 8
```

注意

使用 **$ openstack flavor create --help** 来进一步了解 **openstack flavor create** 命令。

11.3. 创建默认租户网络

overcloud 需要默认租户网络，以便虚拟机在内部通信。

步骤

1. 获取 **overcloudrc** 文件：

```
$ source ~/overcloudrc
```

2. 创建默认租户网络：

```
(overcloud) $ openstack network create default
```

3. 在网络中创建一个子网：

```
(overcloud) $ openstack subnet create default --network default --gateway 172.20.1.1 --
subnet-range 172.20.0.0/16
```

4. 确认所创建的网络：

```
(overcloud) $ openstack network list
+----------------------+-------------+--------------------------------------+
| id                   | name        | subnets                              |
+----------------------+-------------+--------------------------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+----------------------+-------------+--------------------------------------+
```

这些命令创建名为 **default** 的基本 Networking 服务 (neutron) 网络。overcloud 自动使用内部 DHCP 机制将此网络中的 IP 地址分配给虚拟机。

11.4. 创建默认浮动 IP 网络

要从 overcloud 以外访问您的虚拟机，您必须配置一个外部网络，为虚拟机提供浮动 IP 地址。

此流程包含两个示例。使用最适合您环境的示例：

- 原生 VLAN（扁平网络）

- 非原生 VLAN（VLAN 网络）

这两个示例都涉及使用名称 **public** 创建网络。overcloud 需要将这一特定名称用于默认的浮动 IP 池。这个名称对于 第 11.7 节 "验证 overcloud" 中的验证测试也非常重要。

默认情况下，Openstack Networking (neutron) 将名为 **datacentre** 的物理网络名称映射到主机节点上的 **br-ex** 网桥。您将 **public** overcloud 网络连接到物理 **datacentre**，这将通过 **br-ex** 网桥提供网关。

先决条件

- 专用接口或原生 VLAN 用于浮动 IP 网络。

步骤

1. 获取 **overcloudrc** 文件：

```
$ source ~/overcloudrc
```

2. 创建**公共**网络：
   - 创建**扁平**网络获取原生 VLAN 连接：

```
(overcloud) $ openstack network create public --external --provider-network-type flat --provider-physical-network datacentre
```

   - 创建 **vlan** 网络获取非原生 VLAN 连接：

```
(overcloud) $ openstack network create public --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201
```

   使用 **--provider-segment** 选项定义您要使用的 VLAN。在本例中，VLAN 是 **201**。

3. 使用浮动 IP 地址分配池创建子网。在本示例中，IP 范围为 **10.1.1.51** 到 **10.1.1.250**：

```
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range 10.1.1.0/24
```

   确保此范围与外部网络中的其他 IP 地址不冲突。

11.5. 创建默认提供商网络

提供商网络是另一类型的外部网络连接，将流量从私有租户网络路由到外部基础架构网络。该提供商网络类似于浮动 IP 网络，但提供商网络使用逻辑路由器将私有网络连接到提供商网络。

此流程包含两个示例。使用最适合您环境的示例：

- 原生 VLAN（扁平网络）
- 非原生 VLAN（VLAN 网络）

默认情况下，Openstack Networking (neutron) 将名为 **datacentre** 的物理网络名称映射到主机节点上的 **br-ex** 网桥。您将 **public** overcloud 网络连接到物理 **datacentre**，这将通过 **br-ex** 网桥提供网关。

步骤

1. 获取 **overcloudrc** 文件：

```
$ source ~/overcloudrc
```

2. 创建**提供商**网络。

- 创建**扁平**网络获取原生 VLAN 连接：

```
(overcloud) $ openstack network create provider --external --provider-network-type flat --provider-physical-network datacentre --share
```

- 创建 **vlan** 网络获取非原生 VLAN 连接：

```
(overcloud) $ openstack network create provider --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201 --share
```

使用 **--provider-segment** 选项定义您要使用的 VLAN。在本例中，VLAN 是 **201**。

这些示例命令创建共享网络。也可以指定租户而不是指定 **--share**，这样只有租户才能访问新网络。

+ 如果将提供商网络标记为外部网络，则只有运营商可在该网络上创建端口。

3. 向**提供商**网络添加一个子网以提供 DHCP 服务：

```
(overcloud) $ openstack subnet create provider-subnet --network  provider --dhcp --allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254 --subnet-range 10.9.101.0/24
```

4. 创建一个路由器，使其他网络能够通过提供商网络路由流量：

```
(overcloud) $ openstack router create external
```

5. 将路由器的外部网关设置为**提供商**网络：

```
(overcloud) $ openstack router set --external-gateway provider external
```

6. 将其他网络添加到该路由器。例如，运行以下命令以将子网 **subnet1** 附加到路由器上：

```
(overcloud) $ openstack router add subnet external subnet1
```

此命令将 **subnet1** 添加到路由表中，并允许使用 **subnet1** 的虚拟机中的流量路由到提供商网络。

11.6. 创建其他网桥映射

只要部署期间映射其他网桥，浮动 IP 网络可以使用任何网桥，而不只是 **br-ex**。

例如，要将名为 **br-floating** 的新网桥映射到 **floating** 物理网络，可在环境文件中包括 **NeutronBridgeMappings** 参数：

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

使用此方法，您可以在创建 overcloud 后创建单独的外部网络。例如，要创建映射到 **floating** 物理网络的浮动 IP 网络，请运行以下命令：

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-physical-network floating --
provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --subnet-range 10.1.2.0/24
```

11.7. 验证 OVERCLOUD

Overcloud 会使用 OpenStack Integration Test Suite (tempest) 工具集来执行一系列集成测试。本节介绍运行集成测试的准备工作。有关如何使用 OpenStack Integration Test Suite 的完整说明，请参阅 OpenStack Integration Test Suite 指南。

Integration Test Suite 需要执行一些安装后步骤以确保成功测试。

步骤

1. 如果在 undercloud 上运行这个测试，请确保 undercloud 主机能够访问 overcloud 的内部 API 网络。例如，在 undercloud 主机上添加一个临时的 VLAN，用于使用 172.16.0.201/24 地址访问内部 API 网络 (ID: 201)：

   ```
   $ source ~/stackrc
   (undercloud) $ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface vlan201
   type=internal
   (undercloud) $ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev vlan201
   ```

2. 运行集成测试，如 OpenStack Integration Test Suite 指南 中所述。

3. 在验证完成后，删除所有到 overcloud 的内部 API 的临时连接。在这个示例中，使用以下命令删除以前在 undercloud 中创建的 VLAN：

   ```
   $ source ~/stackrc
   (undercloud) $ sudo ovs-vsctl del-port vlan201
   ```

11.8. 防止 OVERCLOUD 被移除

为 heat 设置自定义策略，以防止 overcloud 被删除。

步骤

1. 创建名为 **prevent-stack-delete.yaml** 的环境文件。

2. 设置 **HeatApiPolicies** 参数：

   ```
   parameter_defaults:
     HeatApiPolicies:
       heat-deny-action:
         key: 'actions:action'
         value: 'rule:deny_everybody'
   ```

```
heat-protect-overcloud:
  key: 'stacks:delete'
  value: 'rule:deny_everybody'
```

重要

heat-deny-action 是您必须在 undercloud 安装中包含的默认策略。

3. 将 **prevent-stack-delete.yaml** 环境文件添加到 **undercloud.conf** 文件中的 **custom_env_files** 参数：

```
custom_env_files = prevent-stack-delete.yaml
```

4. 运行 undercloud 安装命令以刷新配置：

```
$ openstack undercloud install
```

此环境文件会防止您删除 overcloud 中的任何堆栈，这意味着您无法执行以下功能：

- 删除 overcloud
- 移除单独的 Compute 或 Ceph Storage 节点
- 替换 Controller 节点

要启用堆栈删除，请从 **custom_env_files** 参数中移除 **prevent-stack-delete.yaml** 文件，再运行 **openstack undercloud install** 命令。

第 12 章 执行基本 OVERCLOUD 管理任务

本章介绍在 overcloud 生命周期过程中可能需要执行的基本任务。

12.1. 通过 SSH 访问 OVERCLOUD 节点

您可以通过 SSH 协议访问每个 overcloud 节点。

- 每个 overcloud 节点都包含一个 **heat-admin** 用户。
- undercloud 上的 **stack** 用户对每个 overcloud 节点上的 **heat-admin** 用户具有基于密钥的 SSH 访问权限。
- 所有 overcloud 节点都有一个短主机名，undercloud 将其解析为 control plane 网络上的 IP 地址。每个短主机名都使用 **.ctlplane** 后缀。例如，**overcloud-controller-0** 的短名称为 **overcloud-controller-0.ctlplane**

先决条件

- 具有可正常工作的 control plane 网络的已部署 overcloud。

步骤

1. 以 **stack** 用户身份登录 undercloud。
2. 获取 **overcloudrc** 文件：

```
$ source ~/stackrc
```

3. 查找要访问的节点的名称：

```
(undercloud) $ openstack server list
```

4. 以 **heat-admin** 用户身份连接到节点，并使用节点的短主机名：

```
(undercloud) $ ssh heat-admin@overcloud-controller-0.ctlplane
```

12.2. 管理容器化服务

Red Hat OpenStack (RHOSP) 平台在 undercloud 和 overcloud 节点上的容器中运行服务。在某些情况下，您可能需要控制主机上的单个服务。本节介绍了可在节点上运行的用于管理容器化服务的一些常见命令。

列出容器和镜像

要列出运行中的容器，请运行以下命令：

```
$ sudo podman ps
```

要在命令输出中包括停止的或失败的容器，将 **--all** 选项添加到命令中：

```
$ sudo podman ps --all
```

要列出容器镜像，请运行以下命令：

```
$ sudo podman images
```

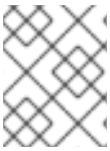
检查容器属性

要查看容器或容器镜像的属性，请使用 **podman inspect** 命令。例如，要检查 **keystone** 容器，请运行以下命令：

```
$ sudo podman inspect keystone
```

使用 Systemd 服务管理容器

早期版本的 OpenStack Platform 使用 Docker 及其守护进程管理容器。在 OpenStack Platform 16 中，Systemd 服务接口管理容器的生命周期。每个容器都是一个服务，您运行 Systemd 命令，为每个容器执行特定操作。



注意

不建议使用 Podman CLI 停止、启动和重启容器，因为 Systemd 会应用重启策略。请使用 Systemd 服务命令。

要检查容器状态，请运行 **systemctl status** 命令：

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
   Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
 Main PID: 29012 (podman)
   CGroup: /system.slice/tripleo_keystone.service
           └─29012 /usr/bin/podman start -a keystone
```

要停止容器，请运行 **systemctl stop** 命令：

```
$ sudo systemctl stop tripleo_keystone
```

要启动容器，请运行 **systemctl start** 命令：

```
$ sudo systemctl start tripleo_keystone
```

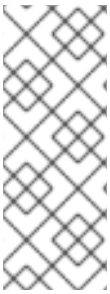
要重启容器，请运行 **systemctl restart** 命令：

```
$ sudo systemctl restart tripleo_keystone
```

由于没有守护进程监控容器状态，Systemd 在以下情况下自动重启大多数容器：

- 清除退出代码或信号，如运行 **podman stop** 命令。
- 取消清除退出代码，如启动后的 podman 容器崩溃。
- 取消清除信号。
- 如果容器启动时间超过 1 分 30 秒，则超时。

有关 Systemd 服务的更多信息，请参阅 **systemd.service** 文档。



注意

在重启容器后，针对其中的服务配置文件所做的所有更改都会恢复。这是因为容器基于 **/var/lib/config-data/puppet-generated/** 中节点的本地文件系统上的文件重新生成服务配置。例如，如果您编辑了 **keystone** 容器中的 **/etc/keystone/keystone.conf**，并重启了该容器，则该容器会使用节点的本地文件系统上的 **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** 来重新生成配置，以覆盖重启之前在该容器中所做的所有更改。

使用 Systemd 计时器监控 podman 容器

Systemd 计时器接口管理容器运行健康检查。每个容器都有一个计时器，它会运行一个服务单员来执行健康检查脚本。

要列出所有 OpenStack Platform 容器计时器，请运行 **systemctl list-timers** 命令并将输出限制为包含 **tripleo** 的行：

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left       Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer          tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:33 UTC  4s left       Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer         tripleo_mistral_engine_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left       Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer               tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left       Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer              tripleo_memcached_healthcheck.service
(...)
```

要检查特定容器计时器的状态，请对运行状况检查服务运行 **systemctl status** 命令：

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

要停止、启动、重启和显示容器计时器的状态，请根据 **.timer** Systemd 资源运行相关 **systemctl** 命令。例如，要检查 **tripleo_keystone_healthcheck.timer** 资源的状态，可运行以下命令：

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

-

如果健康状况检查服务被禁用，但该服务的计时器存在并启用，则意味着检查当前超时，但将根据计时器运行。您还可以手动启动检查。

注意

podman ps 命令不显示容器运行状态。

检查容器日志

OpenStack Platform 16 引入了一个新的日志记录目录 **/var/log/containers/stdout**，其中包含所有容器的标准输出 (stdout) 以及每个容器合并到一个文件中的标准错误 (stderr)。

Paunch 和 **container-puppet.py** 脚本配置 podman 容器以将其输出推送至 **/var/log/containers/stdout** 目录，这将创建所有日志的集合，甚至是删除的容器，如 **container-puppet-*** 容器。

主机会对此目录进行日志轮转以防止产生巨大的文件及占用太多磁盘空间的问题。

如果替换了容器，新的容器将输出到同一日志文件中，因为 **podman** 会使用容器名而非容器 ID。

您也可以使用 **podman logs** 命令检查容器化服务的日志。例如，要查看 **keystone** 容器的日志，请运行以下命令：

```
$ sudo podman logs keystone
```

访问容器

要进入容器化服务的 shell，请使用 **podman exec** 命令以启动 **/bin/bash**。例如，要进入 **keystone** 容器的 shell，请运行以下命令：

```
$ sudo podman exec -it keystone /bin/bash
```

要以根用户身份进入 **keystone** 容器的 shell，请运行以下命令：

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

要退出容器，请运行以下命令：

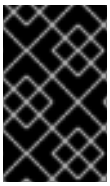
```
# exit
```

12.3. 修改 OVERCLOUD 环境

您可以修改 overcloud 以添加额外功能或更改现有操作。要修改 overcloud，请在自定义环境文件和 heat 模板中进行修改，然后从您的初始 overcloud 创建中重新运行 **openstack overcloud deploy** 命令。例如，如果您使用 第 7.14 节 "部署命令" 创建 overcloud，请重新运行以下命令：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/node-info.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --ntp-server pool.ntp.org
```

director 会在 heat 中检查 **overcloud** 栈，然后根据环境文件和 heat 模板更新栈中的每一项目。director 不会重新创建 overcloud，而是更改现有 overcloud。



重要

从自定义环境文件中移除参数不会将参数值恢复到默认配置。您必须从 **/usr/share/openstack-tripleo-heat-templates** 中的核心 heat 模板集合中获得默认值，然后在自定义环境文件中手动设置这些值。

如果您要包括新的环境文件，则使用 `-e` 选项将其添加到 **openstack overcloud deploy** 命令中。例如：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/new-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -e ~/templates/node-info.yaml \
  --ntp-server pool.ntp.org
```

此命令将环境文件中的新参数和资源纳入堆栈。

重要

不建议手动修改 overcloud 配置，因为 director 稍后可能会覆盖这些修改。

12.4. 将虚拟机导入 OVERCLOUD

您可以将虚拟机从现有 OpenStack 环境迁移到 Red Hat OpenStack Platform (RHOSP) 环境。

步骤

1. 在现有 OpenStack 环境中，通过对一个运行的服务器进行快照并下载镜像来创建一个新镜像：

```
$ openstack server image create instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

2. 将导出的镜像复制到 undercloud 节点：

```
$ scp exported_vm.qcow2 stack@192.168.0.2:~/.
```

3. 以 **stack** 用户的身份登录 undercloud。

4. 获取 **overcloudrc** 文件：

```
$ source ~/overcloudrc
```

5. 将导出的镜像上传到 overcloud 中：

```
(overcloud) $ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2 --container-format bare
```

6. 启动新实例：

```
(overcloud) $ openstack server create  imported_instance --key-name default --flavor
m1.demo --image imported_image --nic net-id=net_id
```

重要

这些命令将每个虚拟机磁盘从现有 OpenStack 环境复制到新的 Red Hat OpenStack Platform 环境中。QCOW 快照丢掉了原始的层系统。

此过程从一个 Compute 节点上迁移所有实例。现在即可在该节点上执行维护，而无需让任何实例停机。要让 Compute 节点恢复启用状态，运行以下命令：

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set [hostname] nova-compute --enable
```

12.5. 运行动态清单脚本

director 可在 Red Hat OpenStack Platform (RHOSP) 环境中运行基于 Ansible 的自动化。director 使用 **tripleo-ansible-inventory** 命令在您的环境中生成动态的节点清单。

步骤

1. 要查看动态的节点清单，请在 source **stackrc** 之后运行 **tripleo-ansible-inventory** 命令：

```
$ source ~/stackrc
(undercloud) $ tripleo-ansible-inventory --list
```

使用 **--list** 选项返回有关所有主机的详细信息。此命令以 JSON 格式输出动态清单：

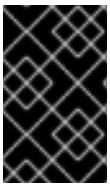
```
{"overcloud": {"children": ["controller", "compute"], "vars": {"ansible_ssh_user": "heat-admin"}},
"controller": ["192.168.24.2"], "undercloud": {"hosts": ["localhost"], "vars":
{"overcloud_horizon_url": "http://192.168.24.4:80/dashboard", "overcloud_admin_password":
"abcdefghijklm12345678", "ansible_connection": "local"}}, "compute": ["192.168.24.3"]}
```

2. 要在您的环境中实施 Ansible playbook，请运行 **ansible** 命令，并使用 **-i** 选项包括动态清单工具的完整路径。例如：

```
(undercloud) $ ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- 用您要使用的主机类型替换 **[HOSTS]**：
  - **controller**，适用于所有 Controller 节点
  - **compute**，适用于所有 Compute 节点
  - **overcloud**，适用于所有 overcloud 子节点。例如， **controller** 和 **compute** 节点
  - **undercloud**，适用于 undercloud
  - **"*"**，适用于所有节点
- 用其他 Ansible 选项替换 **[OTHER OPTIONS]**。

- 使用 **--ssh-extra-args='-o StrictHostKeyChecking=no'** 选项跳过主机密钥检查确认操作。
- 使用 **-u [USER]** 选项更改执行 Ansible 自动化的 SSH 用户。默认的 overcloud SSH 用户由动态清单中的 **ansible_ssh_user** 参数自动定义。**-u** 选项会覆盖此参数。
- 使用 **-m [MODULE]** 选项使用特定的 Ansible 模块。默认为 **command**，用于执行 Linux 命令。
- 使用 **-a [MODULE_ARGS]** 选项为选定的模块定义参数。



重要

overcloud 上的自定义 Ansible 自动化不是标准 overcloud 堆栈的一部分。后续执行 **openstack overcloud deploy** 命令可能会覆盖 overcloud 节点上的 OpenStack Platform 服务的基于 Ansible 的配置。

12.6. 移除 OVERCLOUD

若要移除 overcloud，请运行 **openstack overcloud delete** 命令。

流程

1. 删除现有的 overcloud：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud delete overcloud
```

2. 确认 **openstack stack list** 命令的输出中不再存在 overcloud:

```
(undercloud) $ openstack stack list
```

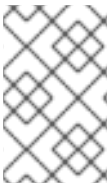
删除操作会需要几分钟完成。

3. 在删除操作完成后，请按照部署情景中的标准步骤来重新创建 overcloud。

第 13 章 使用 ANSIBLE 配置 OVERCLOUD

Ansible 是应用 overcloud 配置的主要方法。本章介绍了有关如何与 overcloud Ansible 配置进行交互的信息。

尽管 director 会自动生成 Ansible playbook，您最好熟悉 Ansible 语法。有关使用 Ansible 的更多信息，请参见 https://docs.ansible.com/。



注意

Ansible 还使用了角色这一概念，但这有别于 OpenStack Platform director 中的角色。**Ansible 角色**形成了 playbook 的可重复使用的组件，而 director 角色包含 OpenStack 服务到节点类型的映射。

13.1. 基于 ANSIBLE 的 OVERCLOUD 配置 (CONFIG-DOWNLOAD)

config-download 功能是 director 用于配置 overcloud 的方法。director 使用 **config-download** 以及 OpenStack Orchestration (heat) 和 OpenStack Workflow Service (mistral) 生成软件配置并将配置应用到每个 overcloud 节点。尽管 heat 从 **SoftwareDeployment** 资源创建所有部署数据以执行 overcloud 安装和配置，但 heat 并不应用任何配置。Heat 仅通过 heat API 提供配置数据。当 director 创建堆栈时，mistral 工作流查询 heat API 以获取配置数据，生成一组 Ansible playbook，并将 Ansible playbook 应用到 overcloud。

作为结果，在运行 **openstack overcloud deploy** 命令时后会发生以下操作：

- director 根据 **openstack-tripleo-heat-templates** 创建新的部署计划，并通过包含环境文件和参数来自定义该计划。
- director 使用 heat 来解释部署计划，并创建 overcloud 堆栈和所有下级资源。这包括使用 OpenStack Bare Metal 服务 (ironic) 置备节点。
- Heat 也会从部署计划创建软件配置。director 从这一软件配置中编译 Ansible playbook。
- director 在 overcloud 节点上生成一个临时用户 (**tripleo-admin**)，专门用于进行 Ansible SSH 访问。
- director 下载 heat 软件配置，并使用 heat 输出生成一系列 Ansible playbook。
- director 通过 **ansible-playbook** 将 Ansible playbook 应用到 overcloud 节点。

13.2. CONFIG-DOWNLOAD 工作目录

director 生成一组 Ansible playbook 以用于 **config-download** 过程。这些 playbook 存储在 **/var/lib/mistral/** 下的工作目录中。此目录依据 overcloud 的名称来命名，默认为 **overcloud**。

工作目录中包含一组按照各个 overcloud 角色命名的子目录。这些子目录包含与 overcloud 角色中节点配置相关的所有任务。这些子目录也包含以每个特定节点命名的其他子目录。这些子目录包含可应用到 overcloud 角色任务的节点相关变量。因此，工作目录中的 overcloud 角色采用下列结构：

```
— /var/lib/mistral/overcloud
  |
  ├── Controller
  |   ├── overcloud-controller-0
  |   ├── overcloud-controller-1
  |   └── overcloud-controller-2
```

```
├── Compute
│   ├── overcloud-compute-0
/   ├── overcloud-compute-1
│   └── overcloud-compute-2
...
```

每个工作目录都是记录每次部署操作后进行的更改的本地 Git 软件仓库。使用本地 Git 软件仓库跟踪各部署之间的配置更改。

13.3. 启用对于 CONFIG-DOWNLOAD 工作目录的访问权限

OpenStack Workflow service (mistral) 容器中的 **mistral** 用户拥有 **/var/lib/mistral/** 工作目录中的所有文件。您可以向 undercloud 上的 **stack** 用户授予访问这个目录中所有文件的权限。这有助于在该目录中执行特定的操作。

步骤

1. 使用 **setfacl** 命令授予 undercloud 上的 **stack** 用户访问 **/var/lib/mistral** 目录中文件的权限：

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
$ sudo chmod -R og-rwx /var/lib/mistral/.ssh
```

此命令保留 **mistral** 用户对该目录的访问权限。

13.4. 检查 CONFIG-DOWNLOAD 日志

在 **config-download** 过程上，Ansible 在 undercloud 上的 **config-download** 工作目录下创建日志文件。

步骤

1. 通过在 **config-download** 工作目录中执行 **less** 命令来查看日志。下例使用了 **overcloud** 工作目录：

```
$ less /var/lib/mistral/overcloud/ansible.log
```

13.5. 对工作目录执行 GIT 操作

config-download 工作目录是本地 Git 软件仓库。每次运行部署操作时，director 会向工作目录添加一个包含相关更改的 Git 提交。可以执行 Git 操作以查看不同阶段部署的配置，并将此配置与不同部署进行比较。

请注意工作目录的限制。例如，如果您使用 Git 恢复到早期版本的 **config-download** 工作目录，则此操作仅影响工作目录中的配置。不会影响以下配置：

- **overcloud 数据架构：**应用上一版本的工作目录软件配置不会撤消数据迁移和架构更改。
- **overcloud 的硬件布局：**恢复到之前的软件配置不会撤消与 overcloud 硬件相关的更改，如扩展或缩减。
- **heat 堆栈：**恢复到以前版本的工作目录不会影响 heat 堆栈中存储的配置。heat 堆栈创建软件配置的新版本，并应用到 overcloud。要对 overcloud 进行永久更改，在重新运行 **openstack overcloud deploy** 之前修改应用到 overcloud 堆栈的环境文件。

完成以下步骤，比较 **config-download** 工作目录的不同提交项。

步骤

1. 更改为您的 overcloud 的 **config-download** 工作目录。在本例中，overcloud 的工作目录的名称是 **overcloud**：

```
$ cd /var/lib/mistral/overcloud
```

2. 运行 **git log** 命令，以列出工作目录中的提交。您也可以通过格式化日志输出来显示日期：

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

默认情况下，最近的提交显示在前面。

3. 针对两个提交哈希运行 **git diff**，以查看部署之间的所有变化：

```
$ git diff a7e9063 dfb9d12
```

13.6. 使用 CONFIG-DOWNLOAD 的部署方法

在 overcloud 部署上下文中有四种使用 **config-download** 的主要方法：

标准部署

运行 **openstack overcloud deploy** 命令，以在置备阶段后自动运行配置阶段。这是运行 **openstack overcloud deploy** 命令时的默认方法。

分离置备和配置

使用特定选项运行 **openstack overcloud deploy** 命令，以分离置备和配置阶段。

部署后运行 ansible-playbook-command.sh 脚本

使用结合或分离的置备和配置阶段运行 **openstack overcloud deploy** 命令，然后运行 **config-download** 工作目录中提供的 **ansible-playbook-command.sh** 脚本，以重新应用配置阶段。

置备节点，手动创建 config-download，并运行 Ansible

使用特定选项运行 **openstack overcloud deploy** 命令，以置备节点，然后使用 **deploy_steps_playbook.yaml** playbook 运行 **ansible-playbook** 命令。

13.7. 在标准部署上运行 CONFIG-DOWNLOAD

执行 **config-download** 的默认方法是运行 **openstack overcloud deploy** 命令。此方法适合大多数环境。

先决条件

- 成功安装 undercloud。
- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

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

```
$ source ~/stackrc
```

3. 运行部署命令。包括 overcloud 所需的任何环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
```

4. 等待部署过程完成。

在部署过程中，director 在 **/var/lib/mistral/** 工作目录中生成 **config-download** 文件。在部署过程完成后，查看工作目录中的 Ansible playbook，以查看为配置 overcloud 而执行的任务 director。

13.8. 使用分离的置备和配置运行 CONFIG-DOWNLOAD

openstack overcloud deploy 命令运行基于 heat 的置备过程，然后运行 **config-download** 配置过程。您还可以运行该部署命令来单独执行每个过程。使用此方法将 overcloud 节点置备为不同的过程，以便您可以在运行 overcloud 配置过程之前在节点上执行任何手动预配置任务。

先决条件

- 成功安装 undercloud。
- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

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

```
$ source ~/stackrc
```

3. 使用 **--stack-only** 选项运行部署命令。包括 overcloud 所需的任何环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. 等待置备过程完成。

5. 为 **tripleo-admin** 用户启用从 undercloud 到 overcloud 的 SSH 访问。 **config-download** 进程使用 **tripleo-admin** 用户来执行基于 Ansible 的配置：

```
$ openstack overcloud admin authorize
```

6. 在节点上执行任何手动预配置任务。如果使用 Ansible 进行配置，请使用 **tripleo-admin** 用户来访问节点。

7. 使用 **--config-download-only** 选项运行部署命令。包括 overcloud 所需的环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --config-download-only
```

8. 等待配置过程完成。

在配置阶段，director 在 **/var/lib/mistral/** 工作目录中生成 **config-download** 文件。在部署过程完成后，查看工作目录中的 Ansible playbook，以查看为配置 overcloud 而执行的任务 director。

13.9. 使用 ANSIBLE-PLAYBOOK-COMMAND.SH 脚本运行 CONFIG-DOWNLOAD

当您使用标准方法或分离的置备和配置过程部署 overcloud 时，director 会在 **/var/lib/mistral/** 中生成工作目录。此目录包含再次运行配置过程所需的 playbook 和脚本。

先决条件

- 使用以下方法之一部署的 overcloud：
  - 结合置备和配置过程的标准方法
  - 分离置备和配置过程

步骤

1. 以 **stack** 用户身份登录 undercloud 主机。

2. 更改为 Ansible playbook 的目录：

```
$ cd /var/lib/mistral/overcloud/
```

3. 将 **/var/lib/mistral/.ssh** 目录的所有者更改为 **stack** 用户。

```
$ sudo chown stack. -R /var/lib/mistral/.ssh/
```

4. 运行 **ansible-playbook-command.sh** 命令以运行 overcloud 配置：

```
$ sudo ./ansible-playbook-command.sh
```

5. 将 **/var/lib/mistral/.ssh** 目录的所有者更改为 **mistral** 用户。这需要确保 **mistral_executor** 容器中运行的 ansible-playbook 命令成功。

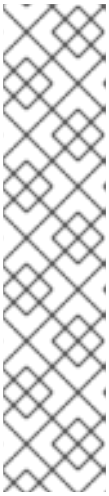
```
$ sudo chown 42430:42430 -R /var/lib/mistral/.ssh/
```

6. 以 **mistral** 用户身份再次运行该脚本。
可以将额外的 Ansible 参数传递给该脚本，再将其原封不动地传递给 **ansible-playbook** 命令。这意味着您可以使用其他 Ansible 功能，如检查模式 (**--check**)、限制主机 (**--limit**) 或覆盖变量 (**-e**)。例如：

```
$ ./ansible-playbook-command.sh --limit Controller
```

警告

当 **--limit** 用于大规模部署时，只有执行中包含的主机才会添加到跨节点的 SSH **known_hosts** 文件中。因此，一些操作（如实时迁移）可能无法在没有在 **known_hosts** 文件中的节点间工作。



注意

要确保 **/etc/hosts** 文件在所有节点上都为最新版本，请以 **root** 用户身份运行以下命令：

```
(undercloud)$ sudo -i
(undercloud)$ cd /var/lib/mistral/overcloud
(undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible
allovercloud \
  -i tripleo-ansible-inventory.yaml \
  -m include_role \
  -a name=tripleo-hosts-entries \
  -e @global_vars.yaml
```

7. 等待配置过程完成。

其他信息

- 这个工作目录中包含一个名为 **deploy_steps_playbook.yaml** 的 playbook，用于管理 overcloud 配置任务。要查看此 playbook，请运行以下命令：

```
$ less deploy_steps_playbook.yaml
```

这个 playbook 会使用工作目录中所含的各种任务文件。某些任务文件是所有 OpenStack 平台角色通用的，某些任务文件则特定于某些 OpenStack 平台角色和服务器。

- 这个工作目录中还包含与您在 overcloud 的 **roles_data** 文件中定义的各个角色相对应的子目录。例如：

```
$ ls Controller/
```

每个 OpenStack 平台角色目录中还包含相应角色类型的各个服务器的子目录。这些目录采用可组合角色主机名格式：

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml** 中的 Ansible 任务已标记。要查看完整的标记列表，在 **ansible-playbook** 中使用 CLI 选项 **--list-tags**：

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

然后，在 **ansible-playbook-command.sh** 脚本中通过 **--tags**、**--skip-tags** 或 **--start-at-task** 来应用已标记的配置：

```
$ ./ansible-playbook-command.sh --tags overcloud
```

1. 对 overcloud 运行 **config-download** playbook 时，可能会收到有关每个主机的 SSH 指纹的消息。要避免这些消息，请在运行 **ansible-playbook-command.sh** 脚本时包含 **--ssh-common-args="-o StrictHostKeyChecking=no"**：

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

13.10. 使用手动创建的 PLAYBOOK 运行 CONFIG-DOWNLOAD

您可以在标准工作流之外创建自己的 **config-download** 文件。例如，您可以使用 **--stack-only** 选项运行 **openstack overcloud deploy** 命令以置备节点，然后单独手动应用 Ansible 配置。

先决条件

- 成功安装 undercloud。
- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

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

```
$ source ~/stackrc
```

3. 使用 **--stack-only** 选项运行部署命令。包括 overcloud 所需的环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. 等待置备过程完成。

5. 为 **tripleo-admin** 用户启用从 undercloud 到 overcloud 的 SSH 访问。 **config-download** 进程使用 **tripleo-admin** 用户来执行基于 Ansible 的配置：

```
$ openstack overcloud admin authorize
```

6. 生成 **config-download** 文件：

```
$ openstack overcloud config download \
  --name overcloud \
  --config-dir ~/config-download
```

   - **--name** 是要用于 Ansible 文件导出的 overcloud 名称。
   - **--config-dir** 是要保存 **config-download** 文件的位置。

7. 切换到包含 **config-download** 文件的目录：

```
$ cd ~/config-download
```

8. 生成静态清单文件：

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user heat-admin \
  --static-yaml-inventory inventory.yaml
```

   - 用您的 overcloud 的名称替换 **<overcloud>**。

9. 使用 **config-download** 文件和静态清单文件执行配置。要执行部署 playbook，请运行 **ansible-playbook** 命令：

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/config-download/deploy_steps_playbook.yaml
```

10. 等待配置过程完成。

11. 要从该配置手动生成 **overcloudrc** 文件，请运行以下命令：

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. 手动将部署状态设置为成功：

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<OVERCLOUD>"}'
```

- 用您的 overcloud 的名称替换 **<OVERCLOUD>**。

其他信息

- 这个 **config-download** 目录中包含一个名为 **deploy_steps_playbook.yaml** 的 playbook，用于运行 overcloud 配置。要查看此 playbook，请运行以下命令：

```
$ less deploy_steps_playbook.yaml
```

这个 playbook 会使用工作目录中所含的各种任务文件。某些任务文件是所有 OpenStack 平台角色通用的，某些任务文件则特定于某些 OpenStack 平台角色和服务器。

- 这个 **config-download** 目录中还包含与您在 overcloud 的 **roles_data** 文件中定义的各个角色相对应的子目录。例如：

```
$ ls Controller/
```

每个 OpenStack 平台角色目录中还包含相应角色类型的各个服务器的子目录。这些目录采用可组合角色主机名格式：

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml** 中的 Ansible 任务已标记。要查看完整的标记列表，在 **ansible-playbook** 中使用 CLI 选项 **--list-tags**：

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

然后，在 **ansible-playbook-command.sh** 脚本中通过 **--tags**、**--skip-tags** 或 **--start-at-task** 来应用已标记的配置：

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

1. 对 overcloud 运行 **config-download** playbook 时，可能会收到有关每个主机的 SSH 指纹的消息。要避免这些消息，请将 **--ssh-common-args="-o StrictHostKeyChecking=no"** 包含到 **ansible-playbook** 命令中：

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

13.11. CONFIG-DOWNLOAD 的限制

config-download 功能有一些限制：

- 在使用 ansible-playbook CLI 参数（如 **--tags**、**--skip-tags** 或 **--start-at-task**）时，请勿随意更改部署配置的运行或应用顺序。利用这些 CLI 参数，可以轻松地重新运行先前失败的任务或在初始部署的基础上进行迭代。但是，为了保证部署的一致性，您必须通过 **deploy_steps_playbook.yaml** 依序运行所有任务。
- 您不能将 **--start-at-task** 参数用于在任务名称中使用变量的某些任务。例如，**--start-at-task** 参数不适用于以下 Ansible 任务：

```
- name: Run puppet host configuration for step {{ step }}
```

- 如果您的 overcloud 部署包含 director 部署的 Ceph Storage 集群，则在使用 **--check** 选项时您无法跳过 **step1** 任务，除非您也跳过 **external_deploy_steps** 任务。
- 您可以使用 **--forks** 选项设置并行 Ansible 任务的数量。但是，在 25 个并行任务后 **config-download** 操作的性能会下降。因此，**--forks** 选项请勿超过 25。

13.12. CONFIG-DOWNLOAD 顶层文件

以下文件是 **config-download** 工作目录内的重要顶级文件。

Ansible 配置和执行。

以下文件专用于在 **config-download** 工作目录中配置和执行 Ansible。

ansible.cfg
: 运行 **ansible-playbook** 时所使用的配置文件。

ansible.log
: 来自最近一次 **ansible-playbook** 运行的日志文件。

ansible-errors.json
: 包含任何部署错误的 JSON 结构文件。

ansible-playbook-command.sh
: 从上次部署操作重新运行 **ansible-playbook** 命令的可执行脚本。

ssh_private_key
: Ansible 用于访问 overcloud 节点的 SSH 私钥。

tripleo-ansible-inventory.yaml
: 包含所有 overcloud 节点的主机和变量的 Ansible 清单文件。

overcloud-config.tar.gz
: 工作目录的存档。

Playbook

下列文件是 **config-download** 工作目录中的 playbook。

deploy_steps_playbook.yaml
: 主要的部署步骤。此 playbook 为您的 overcloud 执行主要的配置操作。

pre_upgrade_rolling_steps_playbook.yaml

主要升级的升级前步骤

upgrade_steps_playbook.yaml

主要升级步骤。

post_upgrade_steps_playbook.yaml

主要升级的升级后步骤。

update_steps_playbook.yaml

次要更新步骤。

fast_forward_upgrade_playbook.yaml

快进升级任务。仅在要从一个长生命版本的 Red Hat OpenStack Platform 升级到下一个版本时使用此 playbook。

13.13. CONFIG-DOWNLOAD 标签

playbook 使用标记的任务控制其应用到 overcloud 的任务。使用包含 **ansible-playbook** CLI 参数的标签 **--tags** 或 **--skip-tags** 控制要执行的任务。以下列表包含有关默认启用的标签的信息：

facts

fact 收集操作。

common_roles

适用于所有节点的 Ansible 角色。

overcloud

用于 overcloud 部署的所有 play。

pre_deploy_steps

在 **deploy_steps** 操作之前发生的部署。

host_prep_steps

主机准备步骤。

deploy_steps

部署步骤。

post_deploy_steps

在 **deploy_steps** 操作后进行的步骤。

external

所有外部部署任务。

external_deploy_steps

仅在 undercloud 中运行的外部部署任务。

13.14. CONFIG-DOWNLOAD 部署步骤

deploy_steps_playbook.yaml playbook 配置 overcloud。此 playbook 应用所有必需的软件配置，以基于 overcloud 部署计划部署完整的 overcloud。

本节包含此 playbook 内使用的不同 Ansible play 的总结。本节中的 play 名称是 playbook 中使用的相同名称，也显示在 **ansible-playbook** 输出中。本节还包含有关在每个 play 上设置的 Ansible 标签的信息。

从 undercloud 收集事实

为 undercloud 节点收集的 fact。
标签： facts

从 overcloud 收集 fact

为 overcloud 节点收集的 fact。
标签： ***facts***

加载全局变量

从 ***global_vars.yaml*** *加载所有变量。*
标签： ***always***

TripleO 服务器的通用角色

将常见 Ansible 角色应用于所有 overcloud 节点，包括用于安装 bootstrap 软件包的 tripleo-bootstrap 和用于配置 ssh 已知主机的 tripleo-ssh-known-hosts。
标签： ***common_roles***

Overcloud 部署步骤任务 - 步骤 0

从 deploy_steps_tasks 模板接口应用任务。
标签： ***overcloud***、***deploy_steps***

服务器部署

应用特定于服务器的 heat 部署，以进行网络和基础架构数据等配置。包括 NetworkDeployment、<Role>Deployment、<Role>AllNodesDeployment，等等。
标签： ***overcloud***、***pre_deploy_steps***

主机准备步骤

从 host_prep_steps 模板接口应用任务。
标签： ***overcloud***、***host_prep_steps***

外部部署步骤 [1,2,3,4,5]

应用来自 external_deploy_steps_tasks 模板接口的任务。Ansible 仅针对 undercloud 节点运行这些任务。
标签： ***external***、***external_deploy_steps***

Overcloud 部署步骤任务 [1,2,3,4,5]

从 deploy_steps_tasks 模板接口应用任务。
标签： ***overcloud***、***deploy_steps***

Overcloud 通用部署步骤任务 [1,2,3,4,5]

应用每个步骤执行的常见任务，包括 puppet 主机配置 ***container-puppet.py*** *和 paunch（容器配置）。*
标签： ***overcloud***、***deploy_steps***

部署后服务器

5 步部署过程后执行的配置应用特定于服务器的 heat 部署。
标签： ***overcloud***、***post_deploy_steps***

部署之后的外部部署任务

应用来自 external_post_deploy_steps_tasks 模板接口的任务。Ansible 仅针对 undercloud 节点运行这些任务。
标签： ***external***、***external_deploy_steps***

第 14 章 使用 ANSIBLE 管理容器

注意

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

Red Hat OpenStack Platform 16.2 使用 Paunch 管理容器。但是，您还可以使用 Ansible 角色 **tripleo-container-manage** 在容器上执行管理操作。如果要使用 **tripleo-container-manage** 角色，您必须首先禁用 Paunch。禁用 Paunch 后，director 会自动使用 Ansible 角色，您还可以编写自定义 playbook 来执行特定的容器管理操作：

- 收集 heat 生成的容器配置数据。**tripleo-container-manage** 角色使用此数据来编配容器部署。
- 启动容器。
- 停止容器。
- 更新容器。
- 删除容器。
- 使用特定配置运行容器。

虽然 director 会自动执行容器管理，但您可能想要自定义容器配置，或者在不重新部署 overcloud 的情况下对容器应用热修复。

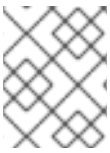
注意

此角色仅支持 Podman 容器管理。

先决条件

- 成功安装 undercloud。更多信息请参阅 第 4.8 节 “安装 director”。

14.1. 在 UNDERCLOUD 上启用 TRIPLEO-CONTAINER-MANAGE ANSIBLE 角色



注意

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

Paunch 是 Red Hat OpenStack Platform 16.2 中的默认容器管理机制。但是，您也可以使用 **tripleo-container-manage** Ansible 角色。如果要使用此角色，则必须禁用 Paunch。

先决条件

- 安装了基本操作系统和 **python3-tripleoclient** 软件包的主机。更多信息请参阅 第 3 章 director 安装准备。

流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 在 **undercloud.conf** 文件中将 **undercloud_enable_paunch** 参数设置为 **false**：

```
undercloud_enable_paunch: false
```

3. 运行 **openstack undercloud install** 命令：

```
$ openstack undercloud install
```

14.2. 在 OVERCLOUD 上启用 TRIPLEO-CONTAINER-MANAGE ANSIBLE 角色

注意

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

Paunch 是 Red Hat OpenStack Platform 16.2 中的默认容器管理机制。但是，您也可以使用 **tripleo-container-manage** Ansible 角色。如果要使用此角色，则必须禁用 Paunch。

先决条件

- 成功安装 undercloud。更多信息请参阅 第 4 章 在 undercloud 上安装 director。

流程

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

```
$ source ~/stackrc
```

3. 在 overcloud 部署命令中包含 **/usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml** 文件，以及与部署相关的任何其他环境文件：

```
(undercloud) [stack@director ~]$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml
  -e <other_environment_files>
  ...
```

14.3. 对单个容器执行操作

注意

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

*您可以使用 **tripleo-container-manage** 角色来管理所有容器或特定容器。如果要管理特定容器，您必须找到容器部署步骤和容器配置 JSON 文件的名称，以便可以利用自定义 Ansible playbook 针对特定容器进行管理。*

先决条件

- *成功安装 undercloud。更多信息请参阅 第 4 章 在 undercloud 上安装 director。*

步骤

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

```
$ source ~/overcloudrc
```

3. *找到容器部署步骤。您可在 **/var/lib/tripleo-config/container-startup-config/step_{1,2,3,4,5,6}** 目录中找到每个步骤的容器配置。*
4. *找到容器的 JSON 配置文件。您可以在相关的 **step_*** 目录中找到容器配置文件。例如，第 1 步中的 HAProxy 容器的配置文件为 **/var/lib/tripleo-config/container-startup-config/step_1/haproxy.json**。*
5. *编写合适的 Ansible playbook。例如，要替换 HAProxy 容器镜像，请使用以下示例 playbook：*

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo-container-manage
          vars:
            tripleo_container_manage_systemd_order: true
            tripleo_container_manage_config_patterns: 'haproxy.json'
            tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
            tripleo_container_manage_config_id: "tripleo_step1"
            tripleo_container_manage_config_overrides:
              haproxy:
                image: registry.redhat.io/tripleomaster/<HAProxy-container>:hotfix
```

   *有关您可以在 **tripleo-container-manage** 角色中使用的变量的更多信息，请参阅 第 14.4 节 "tripleo-container-manage 角色变量"。*

6. *运行 playbook：*

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml
```

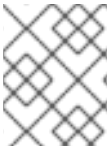
   *如果要在不应用任何更改的情况下执行 playbook，请在 **ansible-playbook** 命令中包含 **--check** 选项：*

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check
```

*如果要在不应用更改的情况下识别 playbook 对容器的更改，请在 **ansible-playbook** 命令中包含 **--check** 和 **--diff**选项：*

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check --diff
```

14.4. TRIPLEO-CONTAINER-MANAGE 角色变量



注意

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

***tripleo-container-manage** Ansible 角色包含以下变量：*

表 14.1. 角色变量

名称	默认值	描述
tripleo_container_manage_check_puppet_config	false	如果您希望 Ansible 检查 Puppet 容器配置，则使用此变量。Ansible 可使用配置 hash 来识别更新的容器配置。如果容器有一个来自 Puppet 的新配置，请将此变量设置为 **true**，以便 Ansible 能够检测到新配置，并将容器添加到 Ansible 必须重启的容器列表中。
tripleo_container_manage_cli	podman	使用此变量设置要用于管理容器的命令行界面。**tripleo-container-manage** 角色仅支持 Podman。
tripleo_container_manage_concurrency	1	使用此变量设置要同时管理的容器数量。
tripleo_container_manage_config	/var/lib/tripleo-config/	使用此变量设置容器配置目录的路径。
tripleo_container_manage_config_id	tripleo	使用此变量设置具体配置步骤的 ID。例如，将此值设置为 **tripleo_step2**，以管理部署的第二步的容器。
tripleo_container_manage_config_patterns	*.json	使用此变量设置用于标识容器配置目录中配置文件的 bash 正则表达式。

名称	默认值	描述
tripleo_container_manage_debug	false	使用此变量启用或禁用调试模式。如果要运行带有特定一次性配置的容器，请在调试模式下运行 **tripleo-container-manage** 角色，以输出管理容器生命周期的容器命令，或运行 no-op 容器管理操作进行测试和验证。
tripleo_container_manage_healthcheck_disable	false	使用此变量启用或禁用健康检查。
tripleo_container_manage_log_path	/var/log/containers/stdouts	使用此变量为容器设置 stdout 日志路径。
tripleo_container_manage_systemd_order	false	使用此变量启用或禁用 Ansible 的 systemd 关闭顺序。
tripleo_container_manage_systemd_teardown	true	使用此变量触发已弃用容器的清理。
tripleo_container_manage_config_overrides	{}	使用此变量覆盖任何容器配置。此变量的值来自一个字典，其中每个键都是容器名称和您要覆盖的参数，如容器镜像或用户。此变量不会将自定义覆盖写入 JSON 容器配置文件，并且任何新的容器部署、更新或升级都会恢复到 JSON 配置文件的内容。
tripleo_container_manage_valid_exit_code	[]	使用此变量检查容器是否返回退出代码。这个值必须是列表，例如 **[0， 3]**。

第 15 章 使用验证框架

Red Hat OpenStack Platform 包含一个验证框架，可以用于验证 undercloud 和 overcloud 的要求和功能。该框架包括两种验证类型：

- 基于 Ansible 的手动验证，通过 **openstack tripleo validator** 命令集执行。
- 自动动态验证，在部署过程中执行。

您必须了解您要运行哪些验证，并跳过与您的环境无关的验证。例如，部署前验证包括 TLS-everywhere 的测试。如果您不打算为 TLS-everywhere 配置环境，此测试会失败。使用 **openstack tripleo validator run** 命令中的 **--validation** 选项来根据您的环境重新定义验证。

15.1. 基于 ANSIBLE 的验证

在安装 Red Hat OpenStack Platform director 的过程中，director 还会从 **openstack-tripleo-validations** 软件包中安装一组 playbook。每个 playbook 包含对某些系统要求的测试，以及定义何时运行测试的一系列组：

no-op

运行 no-op（无操作）任务的验证，以验证工作流功能正常。这些验证在 undercloud 和 overcloud 上运行。

Prep

检查 undercloud 节点的硬件配置的验证。在运行 **openstack undercloud install** 命令前运行这些验证。

openshift-on-openstack

检查环境是否满足可在 OpenStack 上部署 OpenShift 的要求的验证。

pre-introspection

使用 Ironic 检查程序进行节点内省前要运行的验证。

pre-deployment

在 **openstack overcloud deploy** 命令前要运行的验证。

post-deployment

在 overcloud 部署完成后要运行的验证。

pre-upgrade

在升级前验证 OpenStack 部署的验证。

升级后

在升级后验证 OpenStack 部署的验证。

15.2. 列出验证

运行 **openstack tripleo validator list** 命令来列出不同类型的可用验证。

步骤

1. source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 运行 **openstack tripleo validator list** 命令：

- 要列出所有验证，请在没有任何选项的情况下运行命令：

```
$ openstack tripleo validator list
```

- 要列出组中的验证，请使用 **--group** 选项运行该命令：

```
$ openstack tripleo validator list --group prep
```

注意

如需完整的选项列表，请运行 **openstack tripleo validator list --help**。

15.3. 运行验证

要运行验证或验证组，请使用 **openstack tripleo validator run** 命令。要查看完整的选项列表，请使用 **openstack tripleo validator run --help** 命令。

步骤

1. Source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 创建并验证名为 **inventory.yaml** 的静态清单文件。

```
$ tripleo-ansible-inventory --static-yaml-inventory inventory.yaml
$ openstack tripleo validator run  --group pre-introspection -i inventory.yaml
```

3. 输入 **openstack tripleo validator run** 命令：

- 要运行单个验证，请输入带 **--validation-name** 选项的命令和验证的名称。例如，要检查每个节点的内存要求，请输入 **--validation check-ram**：

```
$ openstack tripleo validator run --validation check-ram
```

如果 overcloud 使用与默认 **overcloud** 名称不同的计划名称，请使用 **--plan** 选项来设置计划名称：

```
$ openstack tripleo validator run --validation check-ram --plan myovercloud
```

要运行多个特定的验证，请使用 **--validation** 选项以及您要运行的验证的逗号分隔列表。有关查看可用验证列表的更多信息，请参阅 列出验证。

- 要运行组中的所有验证，请输入带 **--group** 选项的命令：

```
$ openstack tripleo validator run --group prep
```

要查看特定验证的详细输出，请根据报告中具体验证的 UUID 运行 **openstack tripleo validator show run --full** 命令：

```
$ openstack tripleo validator show run --full <UUID>
```

15.4. 查看验证历史记录

director 在运行一个验证或一组验证后保存每个验证的结果。使用 **openstack tripleo validator show history** 命令查看以前的验证结果。

先决条件

- 您已运行了一个验证或一组验证。

步骤

1. Source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 查看所有验证的列表：

```
$ openstack tripleo validator show history
```

要查看特定验证类型的历史记录，请使用 **--validation** 选项运行相同的命令：

```
$ openstack tripleo validator show history --validation ntp
```

3. 使用 **openstack tripleo validator show run --full** 命令查看特定验证 UUID 的日志：

```
$ openstack tripleo validator show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

15.5. 验证框架日志格式

在运行一个验证或一组验证后，director 将每个验证的 JSON 格式日志保存在 **/var/logs/validations** 目录中。您可以手动查看文件，也可以使用 **openstack tripleo validator show run --full** 命令显示特定验证 UUID 的日志。

每个验证日志文件都遵循特定的格式：

- **<UUID>_<Name>_<Time>**

  UUID
  验证的 Ansible UUID。

  名称
  验证的 Ansible 名称。

  Time
  运行验证时的开始日期和时间。

每个验证日志包含三个主要部分：

- *plays*
- *stats*
- *validation_output*

plays

plays 部分包含有关 director 在验证过程中执行的任务的信息：

play

一个 play 就是一组任务。每个 **play** 部分包含关于这一特定组任务的信息，包括开始和结束时间、持续时间、play 的主机组，以及验证 ID 和路径。

tasks

director 为执行验证运行的个别 Ansible 任务。每个 **tasks** 部分包含一个 **hosts** 部分，其中包含了在每个单独主机上执行的操作以及执行操作的结果。**tasks** 部分还包含一个 **task** 部分，其中包含了任务的持续时间。

stats

stats 部分包含每个主机上所有任务结果的基本摘要，如成功和失败的任务。

validation_output

如果任何任务在验证过程中失败或导致警告消息，则 **validation_output** 会包含该失败或警告的输出。

15.6. 验证框架日志输出格式

验证框架的默认行为是以 JSON 格式保存验证日志。您可以使用 **ANSIBLE_STDOUT_CALLBACK** 环境变量更改日志输出。

要更改验证输出日志格式，请运行验证并包含 **--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** 选项：

```
$ openstack tripleo validator run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- 将 **<callback>** 替换为 Ansible 输出回调。要查看标准 Ansible 输出回调列表，请运行以下命令：

```
$ ansible-doc -t callback -l
```

验证框架包括以下额外回调：

validation_json

框架将 JSON 格式的验证结果保存为 **/var/logs/validations** 中的日志文件。这是验证框架的默认回调。

validation_stdout

框架在屏幕上显示 JSON 格式的验证结果。

http_json

框架将 JSON 格式的验证结果发送到外部日志记录服务器。您还必须包含此回调的额外环境变量：

HTTP_JSON_SERVER

外部服务器的 URL。

HTTP_JSON_PORT

外部服务器的 API 入口点的端口。8989 中的默认端口。

使用额外的 **--extra-env-vars** 选项设置这些环境变量：

```
$ openstack tripleo validator run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
```

```
--extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
--extra-env-vars HTTP_JSON_PORT=8989 \
--validation check-ram
```

重要

在使用 **http_json** 回调前，必须将 **http_json** 添加到 **ansible.cfg** 文件中的 **callback_whitelist** 参数中：

```
callback_whitelist = http_json
```

15.7. 动态验证

Red Hat OpenStack Platform 在可组合服务的模板中包括了动态验证功能。动态验证在 overcloud 部署过程的关键步骤中验证服务操作状态。

在部署过程中自动运行动态验证。有些动态验证也使用 **openstack-tripleo-validations** 软件包中的角色。

第 16 章 扩展 OVERCLOUD 节点

如果要在创建 overcloud 后添加或移除节点，您必须更新 overcloud。

警告

*不要使用 **openstack server delete** 从 overcloud 中删除节点。按照本节中的步骤正确地删除和替换节点。*

注意

在开始横向扩展或移除 overcloud 节点之前，请确保您的裸机节点未处于维护模式。

下表介绍了对每个节点类型进行扩展的支持信息：

表 16.1. 每个节点类型的扩展支持

节点类型	扩展？	缩减？	备注
Controller	N	N	您可以使用 第 17 章 *替换 Controller 节点* 中的步骤替换 Controller 节点。
计算	Y	Y	
Ceph Storage 节点	Y	N	在初始创建的 overcloud 中必须至少有一个 Ceph Storage 节点。
Object Storage 节点	Y	Y	

重要

在扩展 overcloud 前，请确保至少有 10 GB 的可用空间。这些可用空间将在节点置备过程中用于保存镜像转换和缓存。

16.1. 向 OVERCLOUD 添加节点

完成下列步骤，向 director 节点池添加更多节点。

注意

全新的 Red Hat OpenStack Platform 安装不包括某些更新，如安全勘误和程序错误修复。因此，如果您使用红帽客户门户网站或 Red Hat Satellite Server 扩展连接的环境，RPM 更新不会应用到新节点。要将最新的更新应用到 overcloud 节点，您必须执行以下操作之一：

- 在扩展操作后，完成节点的 overcloud 更新。
- 在 scale-out 操作前，使用 **virt-customize** 工具将软件包修改为基本 overcloud 镜像。有关更多信息，请参阅红帽知识库解决方案，使用 virt-customize 修改 Red Hat Linux OpenStack Platform Overcloud 镜像。

流程

1. 创建一个名为 **newnodes.json** 的新 JSON 文件，其中包含您要注册的新节点的详情：

```
{
  "nodes":[
    {
        "mac":[
            "dd:dd:dd:dd:dd:dd"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"ipmi",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.168.24.207"
    },
    {
        "mac":[
            "ee:ee:ee:ee:ee:ee"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"ipmi",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.168.24.208"
    }
  ]
}
```

2. 注册新节点：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud node import newnodes.json
```

3. 注册新节点后，为每个新节点启动内省过程：

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

此过程将检测和基准测试节点的硬件属性。

4. 配置该节点的镜像属性：

```
(undercloud) $ openstack overcloud node configure [NODE UUID]
```

16.2. 增加角色的节点数

完成以下步骤来扩展特定角色的 overcloud 节点，如 Compute 节点。

步骤

1. 给每个新节点标记您想要的角色。例如，如要为节点标上 Compute 角色，可运行以下命令：

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' [NODE UUID]
```

2. 若要扩展 overcloud，您必须编辑包含节点数目的环境文件并重新部署 overcloud。例如，若要将 overcloud 扩展到 5 个 Compute 节点，可编辑 **ComputeCount** 参数：

```
parameter_defaults:
  ...
  ComputeCount: 5
  ...
```

3. 使用更新后的文件（在本示例中，该文件名为 **node-info.yaml**），重新运行部署命令：

```
(undercloud) $ openstack overcloud deploy --templates -e /home/stack/templates/node-
info.yaml [OTHER_OPTIONS]
```

务必包含初始 overcloud 创建中的所有环境文件和选项。这包括非 Compute 节点的相同缩放参数。

4. 等待部署操作完成。

16.3. 删除或替换 COMPUTE 节点

在某些情况下，您需要从 overcloud 中删除 Compute 节点。例如，需要替换有问题的 Compute 节点。当您删除 Compute 节点时，节点索引默认添加到 denylist 中，以防止在扩展操作期间重复使用索引。

从 overcloud 部署中移除节点后，您可以替换移除的 Compute 节点。

先决条件

- 您要删除的节点上禁用了 Compute 服务，以防止节点调度新实例。要确认禁用了 Compute 服务，请使用以下命令：

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

如果没有禁用 Compute 服务，则禁用它：

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

提示

使用 **--disable-reason** 选项添加有关为何要禁用该服务的简短说明。如果您打算重新部署 Compute 服务，这非常有用。

- Compute 节点上的工作负载已迁移到其他 Compute 节点。如需更多信息，请参阅在 Compute 节点之间迁移虚拟机实例。
- 如果启用了实例 HA，请选择以下选项之一：
  - 如果可以访问计算节点，请以 **root** 用户身份登录计算节点，并使用 **shutdown -h now** 命令执行清理关闭。
  - 如果无法访问计算节点，以 **root** 用户身份登录控制器节点，为计算节点禁用 STONITH 设备，并关闭裸机节点：

```
[root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
[stack@undercloud ~]$ source stackrc
[stack@undercloud ~]$ openstack baremetal node power off <UUID>
```

流程

1. 查找 undercloud 配置：

```
(overcloud)$ source ~/stackrc
```

2. 确定 overcloud 堆栈的 UUID：

```
(undercloud)$ openstack stack list
```

3. 识别您要删除的 Compute 节点的 UUID 或主机名：

```
(undercloud)$ openstack server list
```

4. 可选：使用 **--update-plan-only** 选项运行 **overcloud deploy** 命令，以使用模板中的最新配置来更新计划。这样可确保在删除任何 Compute 节点前 overcloud 配置为最新版本：

```
$ openstack overcloud deploy --update-plan-only \
  --templates  \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml \
  [-e |...]
```

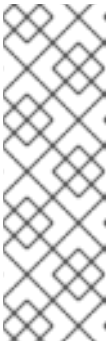
注意

如果您更新了 overcloud 节点 denylist，则需要这一步。有关将 overcloud 节点添加到 denylist 的更多信息，请参阅 将节点列入黑名单。

5. 从堆栈中删除 Compute 节点：

```
$ openstack overcloud node delete --stack <overcloud> \
 <node_1> ... [node_n]
```

- 将 **<overcloud>** 替换为 overcloud 堆栈的名称或 UUID。
- 将 **<node_1>** 以及选择性地将直至 **[node_n]** 的全部节点替换为要删除的 Compute 节点的 Compute 服务主机名或 UUID。不要混合使用 UUID 和主机名。只使用 UUID 或只使用主机名。



注意

如果节点已经关闭，这个命令会返回一个 **WARNING** 信息：

```
Ansible failed, check log at /var/lib/mistral/overcloud/ansible.log
WARNING: Scale-down configuration error. Manual cleanup of some
actions may be necessary. Continuing with node removal.
```

您可以忽略此消息。

6. 等待 Compute 节点删除。

7. 删除您删除的每个节点的网络代理：

```
(overcloud)$ for AGENT in $(openstack network agent list \
 --host <scaled_down_node> -c ID -f value) ; \
 do openstack network agent delete $AGENT ; done
```

8. 在节点删除完成后，检查 overcloud 栈的状态：

```
(undercloud)$ openstack stack list
```

表 16.2. 结果

状态	描述
UPDATE_COMPLETE	删除操作成功完成。
UPDATE_FAILED	删除操作失败。 失败删除操作的常见原因是您要删除的节点上的 IPMI 接口无法访问的 IPMI 接口。 删除操作失败时，您必须手动删除 Compute 节点。有关更多信息，请参阅从 overcloud 手动删除 Compute 节点。

9. 如果启用了 Instance HA，请执行以下操作：

   a. 为节点清理 Pacemaker 资源：

```
$ sudo pcs resource delete <scaled_down_node>
$ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
$ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
force
```

b. 删除节点的 STONITH 设备：

```
$ sudo pcs stonith delete <device-name>
```

10. 如果您不替换 overcloud 上移除的 Compute 节点，则减少包含节点数的环境文件中的 **ComputeCount** 参数。此文件通常被命名为 **node-info.yaml**。例如，如果您删除了一个节点，将节点数从四个节点减小到三个节点：

```
parameter_defaults:
  ...
  ComputeCount: 3
```

减少节点数可确保 director 在运行 **openstack overcloud deploy** 时不置备任何新节点。

如果您要替换 overcloud 部署上移除的 Compute 节点，请参阅 替换移除的 Compute 节点。

16.3.1. 手动删除 Compute 节点

如果因为无法访问的节点而导致 **openstack overcloud node delete** 命令失败，则必须手动从 overcloud 中删除 Compute 节点。

先决条件

- 执行 删除或替换 Compute 节点 流程会返回 **UPDATE_FAILED** 状态。

流程

1. 确定 overcloud 堆栈的 UUID：

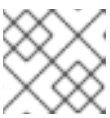
```
(undercloud)$ openstack stack list
```

2. 识别您要删除的节点的 UUID：

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

3. 将您要删除的节点移到维护模式：

```
(undercloud)$ openstack baremetal node maintenance set <node_uuid>
```



注意

等待 Compute 服务将其状态与 Bare Metal 服务同步。这可能需要四分钟。

4. 查找 overcloud 配置：

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

5. 停止您要删除的 Compute 节点上的所有 podman 容器：

```
$ sudo systemctl stop tripleo_*
```

6. 删除您删除的节点的网络代理：

```
(overcloud)$ for AGENT in $(openstack network agent list \
  --host <scaled_down_node> -c ID -f value) ; \
  do openstack network agent delete $AGENT ; done
```

- 将 **<scaled_down_node>** 替换为要移除的节点的名称。

7. 确认 overcloud 上移除的节点上禁用了 Compute 服务，以防止节点调度新实例：

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

如果没有禁用 Compute 服务，请禁用它：

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

提示

使用 **--disable-reason** 选项添加有关为何要禁用该服务的简短说明。如果您打算重新部署 Compute 服务，这非常有用。

8. 从节点中删除 Compute 服务：

```
(overcloud)$ openstack compute service delete <service_id>
```

9. 将删除的 Compute 服务作为资源提供商从放置服务中移除：

```
(overcloud)$ openstack resource provider list
(overcloud)$ openstack resource provider delete <uuid>
```

10. 查找 undercloud 配置：

```
(overcloud)$ source ~/stackrc
```

11. 从堆栈中删除 Compute 节点：

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> <node>
```

- 将 **<overcloud>** 替换为 overcloud 堆栈的名称或 UUID。
- 将 & **lt;node** > 替换为您要删除的 Compute 节点的 Compute 服务主机名或 UUID。

注意

如果节点已经关闭，这个命令会返回一个 **WARNING** 信息：

```
Ansible failed, check log at `/var/lib/mistral/overcloud/ansible.log`
WARNING: Scale-down configuration error. Manual cleanup of some actions may be necessary. Continuing with node removal.
```

您可以忽略此消息。

12. 等待 overcloud 节点删除。

13. 在节点删除完成后，检查 overcloud 栈的状态：

```
(undercloud)$ openstack stack list
```

表 16.3. 结果

状态	描述
UPDATE_COMPLETE	删除操作成功完成。
UPDATE_FAILED	删除操作失败。 如果在维护模式下 overcloud 节点无法删除，则问题可能是硬件。

14. 如果启用了 Instance HA，请执行以下操作：

    a. 为节点清理 Pacemaker 资源：

```
$ sudo pcs resource delete <scaled_down_node>
$ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
$ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
force
```

    b. 删除节点的 STONITH 设备：

```
$ sudo pcs stonith delete <device-name>
```

15. 如果您不替换 overcloud 上移除的 Compute 节点，则减少包含节点数的环境文件中的 **ComputeCount** 参数。此文件通常被命名为 **node-info.yaml**。例如，如果您删除了一个节点，将节点数从四个节点减小到三个节点：

```
parameter_defaults:
  ...
  ComputeCount: 3
  ...
```

减少节点数可确保 director 在运行 **openstack overcloud deploy** 时不置备任何新节点。

如果您要替换 overcloud 部署上移除的 Compute 节点，请参阅 替换移除的 Compute 节点。

16.3.2. 替换删除的 Compute 节点

要替换 overcloud 部署上移除的 Compute 节点，您可以注册并检查新的 Compute 节点，或者重新添加移除的 Compute 节点。您还必须配置 overcloud 以置备节点。

流程

1. 可选：要重复使用已删除 Compute 节点的索引，在删除 Compute 节点时为角色配置 **RemovalPoliciesMode** 和 **RemovalPolicies** 来替换拒绝列表：

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: update
  <RoleName>RemovalPolicies: [{'resource_list': []}]
```

2. 替换删除的 Compute 节点：

   - 若要添加新的 Compute 节点，请注册、检查和标记新节点，以准备它以进行调配。如需更多信息，请参阅配置基本 overcloud。
   - 要重新添加您手动删除的 Compute 节点，从维护模式中删除该节点：

```
(undercloud)$ openstack baremetal node maintenance unset <node_uuid>
```

3. 重新运行用于部署现有 overcloud 的 **openstack overcloud deploy** 命令。
4. 等待部署过程完成。
5. 确认 director 已成功注册了新的 Compute 节点：

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

6. 如果您执行了第 1 步，当计算节点被删除时，为角色将 **RemovalPoliciesMode** 设置为 **update**，然后您您需要为角色将 **RemovalPoliciesMode** 重新设置为默认值（**append**），把计算节点索引添加到当前的拒绝列表中：

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: append
```

7. 重新运行用于部署现有 overcloud 的 **openstack overcloud deploy** 命令。

16.4. 在替换使用可预测的 IP 地址和 HOSTNAMEMAP 的节点时保留主机名

如果将 overcloud 配置为使用可预测的 IP 地址，并且 **HostNameMap** 将基于 heat 的主机名映射到预置备节点的主机名，则必须配置 overcloud，以将新的替换节点索引映射到 IP 地址和主机名。

流程

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

```
$ source ~/stackrc
```

3. 检索您要替换资源的 **physical_resource_id** 和 **removed_rsrc_list** ：

```
(undercloud)$ openstack stack resource show <stack> <role>
```

- 将 **<stack>** 替换为资源所属的堆栈的名称，如 **overcloud**。
- 将 **&** lt;role> 替换为您要替换节点的角色的名称，如 **Compute**。
  输出示例：

```
+------------------------+--------------------------------------------------------+
| Field                  | Value                                                  |
+------------------------+--------------------------------------------------------+
| attributes             | {u'attributes': None, u'refs': None, u'refs_map': None,   |
|                        | u'removed_rsrc_list': [u'2', u'3']}          | 1
| creation_time          | 2017-09-05T09:10:42Z                                   |
| description            |                                                        |
| links                  | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
|                        | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-       |
|                        | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':     |
|                        | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
|                        | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
|                        | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
|                        | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
|                        | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-     |
|                        | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]      |
| logical_resource_id    | Compute                                                |
| physical_resource_id   | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                   |
| required_by            | [u'AllNodesDeploySteps',                               |
|                        | u'ComputeAllNodesValidationDeployment',                |
|                        | u'AllNodesExtraConfig', u'ComputeIpListMap',           |
|                        | u'ComputeHostsDeployment', u'UpdateWorkflow',          |
|                        | u'ComputeSshKnownHostsDeployment', u'hostsConfig',     |
|                        | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']  |
| resource_name          | Compute                                                |
| resource_status        | CREATE_COMPLETE                                        |
| resource_status_reason | state changed                                          |
| resource_type          | OS::Heat::ResourceGroup                                |
| updated_time           | 2017-09-05T09:10:42Z                                   |
+------------------------+--------------------------------------------------------+
```

1 **removed_rsrc_list** 列出了已经为资源删除的节点索引。

4. 检索 **resource_name**，以确定 heat 应用到此资源节点的最大索引：

```
(undercloud)$ openstack stack resource list <physical_resource_id>
```

- 使用在第 3 步中获取的 ID 替换 **<physical_resource_id>**。

5. 使用 **resource_name** 和 **removed_rsrc_list** 确定 heat 将应用到新节点的下一个索引：

- 如果 **removed_rsrc_list** 为空，则下一个索引将是(current_maximum_index)+ 1。

- 如果 **removed_rsrc_list** 包含值(current_maximum_index)+1，则下一个索引将是下一个可用索引。

6. 检索替换裸机节点的 ID：

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

7. 使用新索引更新替换节点的功能：

```
openstack baremetal node set --property capabilities='node:<role>-<index>,boot_option:local'
<node>
```

- 将 **<role>** 替换为您要替换节点的角色的名称，如 **compute**。
- 将 **<index>** 替换为在第 5 步中计算的索引。
- 将 **<node>** 替换为裸机节点的 ID。

计算调度程序使用节点功能与部署时的节点匹配。

8. 通过在 **HostnameMap** 配置中添加索引来为新节点分配主机名，例如：

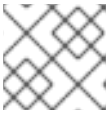
```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0 (1)
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0 (2)
    overcloud-compute-0: overcloud-compute-prod-abc-0
    overcloud-compute-3: overcloud-compute-prod-abc-3 (3)
    overcloud-compute-8: overcloud-compute-prod-abc-3 (4)
    ....
```

(1) 您要删除并替换为新节点的节点。

(2) 新节点。

(3) 您要删除并替换为新节点的节点。

(4) 新节点。



注意

不要删除从 **HostnameMap** 中删除的节点映射。

9. 将替换节点的 IP 地址添加到网络 IP 地址映射文件 **ips-from-pool-all.yaml** 中的每个网络 IP 地址列表的末尾。在以下示例中，新索引 **overcloud-controller-3** 的 IP 地址添加到每个 **ControllerIPs** 网络的 IP 地址列表中，并分配与 **overcloud-controller-1** 相同的 IP 地址，因为它

*替换了 **overcloud-controller-1**。新索引 **overcloud-compute-8** 的 IP 地址也添加到每个 **ComputeIPs** 网络的 IP 地址列表中，并分配与它替换的索引相同的 IP 地址：*

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10 1
      - 192.168.1.11 2
      - 192.168.1.12 3
      - 192.168.1.11 4
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
    internal_api:
      - 172.17.0.10 5
      - 172.17.0.11 6
      - 172.17.0.11 7
    ...
    storage:
      - 172.17.0.10
      - 172.17.0.11
      - 172.17.0.11
    ...
```

1

*分配给索引 0 的 IP 地址，主机名为 **overcloud-controller-prod-123-0**。*

2

*分配给索引 1 的 IP 地址，主机名 **overcloud-controller-prod-456-0**。此节点由索引 3 替代。不要删除此条目。*

3

*分配给索引 2 的 IP 地址，主机名为 **overcloud-controller-prod-789-0**。*

4

*分配给索引 3 的 IP 地址，主机名为 **overcloud-controller-prod-456-0**。这是替换索引 1 的新节点。*

5

分配给索引 0 的 IP 地址，主机名为 overcloud-compute-0。

6

分配给索引 1 的 IP 地址，主机名为 overcloud-compute-3。此节点由索引 2 替代。不要删除此条目。

7

分配给索引 2 的 IP 地址，主机名为 overcloud-compute-8。这是替换索引 1 的新节点。

16.5. 替换 CEPH STORAGE 节点

您可以使用 director 来替换 director 创建的集群中的 Ceph Storage 节点。有关更多信息，请参阅*使用容器化 Red Hat Ceph 部署 Overcloud* 指南。

16.6. 替换 OBJECT STORAGE 节点

参考本节中的说明，了解如何在不影响集群完整性的情况下替换 Object Storage 节点。此示例涉及您要替换 overcloud-objectstorage-1 节点的三节点 Object Storage 集群。此操作过程的目标是添加一个节点，然后删除 overcloud-objectstorage-1 节点。新节点替换 overcloud-objectstorage-1 节点。

步骤

1. 通过 ObjectStorageCount 参数增加 Object Storage 数量。此参数通常位于 node-info.yaml 中，这是包含节点数的环境文件：

```
parameter_defaults:
  ObjectStorageCount: 4
```

ObjectStorageCount 参数定义环境中 Object Storage 节点的数量。在本例中，将 Object Storage 节点的数量从 3 扩展到 4。

2. 使用更新的 ObjectStorageCount 参数，运行部署命令：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates -e node-info.yaml
<environment_files>
```

部署命令完成后，overcloud 将包含新增的 Object Storage 节点。

3. 将数据复制到新节点。在删除节点（本例中为 overcloud-objectstorage-1）前，先等待新节点上完成复制传递。在 /var/log/swift/swift.log 文件中检查复制传递进度。当传递完成时，Object Storage 服务应该会记录类似于以下示例的日志条目：

```
Mar 29 08:49:05 localhost *object-server: Object replication complete.*
Mar 29 08:49:11 localhost *container-server: Replication run OVER*
Mar 29 08:49:13 localhost *account-server: Replication run OVER*
```

4. 若要从环中删除旧节点，可减小 ObjectStorageCount 参数来省略旧节点。在本例中，将 ObjectStorageCount 参数减小到 3:

```
parameter_defaults:
  ObjectStorageCount: 3
```

5. 创建一个新环境文件，命名为 remove-object-node.yaml。此文件将确认并移除指定的 Object Storage 节点。以下内容指定了 overcloud-objectstorage-1 的移除：

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

6. 在部署命令中包含 node-info.yaml 和 remove-object-node.yaml 文件：

```
(undercloud) $ openstack overcloud deploy --templates -e node-info.yaml
<environment_files> -e remove-object-node.yaml
```

director 从 overcloud 中删除 Object Storage 节点，并更新 overcloud 中的其他节点来使删除生效。

重要

请包含初始 overcloud 创建中的所有环境文件和选项。这包括非 Compute 节点的相同缩放参数。

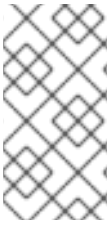
16.7. 使用 SKIP 部署标识符

在堆栈更新操作 puppet 中，默认获取所有清单。这可能导致时间消耗操作，这可能并不是必需的。

要覆盖默认操作，请使用 skip-deploy-identifier 选项。

```
openstack overcloud deploy --skip-deploy-identifier
```

如果您不希望部署命令为 DeployIdentifier 参数生成唯一标识符，则使用此选项。软件配置部署步骤仅当配置发生实际更改时才会触发。使用此选项要非常谨慎，仅当您确信不需要运行软件配置（如扩展某些角色）时方可使用。



注意

如果对 puppet 清单或 hierdata 的更改，则 puppet 将重新应用所有清单，即使指定了 --skip-deploy-identifier。

16.8. 将节点列入黑名单

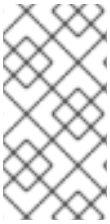
您可以阻止 overcloud 节点获得更新的部署内容。这在某些情况下非常有用，比如，您想要扩展新节点，并阻止现有节点获得核心 heat 模板集合中更新的参数和资源集合。这意味着列入黑名单的节点将完全不受栈操作的影响。

在环境文件中使用 DeploymentServerBlacklist 参数可创建黑名单。

设置黑名单

DeploymentServerBlacklist 参数是服务器名称列表。可以将其写入新的环境文件，或将参数值添加到现有的自定义环境文件，然后将此文件传递给部署命令：

```
parameter_defaults:
  DeploymentServerBlacklist:
    - overcloud-compute-0
    - overcloud-compute-1
    - overcloud-compute-2
```



注意

参数值中的服务器名称是由 OpenStack Orchestration (heat) 规定的名称，并非实际的服务器主机名。

将此环境文件包含到 **openstack overcloud deploy** 命令中：

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e server-blacklist.yaml \
  [OTHER OPTIONS]
```

heat 会将列表中的任何服务器列入黑名单，阻止其获得更新的 heat 部署内容。在栈操作完成后，黑名单中的服务器不会发生任何变化。您也可以在操作过程中关闭或停止 os-collect-config 代理。

警告

- 将节点列入黑名单时要非常谨慎。在使用黑名单前，必须完全清楚在有黑名单的情况下如何应用所要求的更改。在使用黑名单功能时，有可能造成栈停止工作，或对 overcloud 执行不正确的配置。例如，如果集群配置更改应用到 Pacemaker 集群的所有成员，那么在执行更改时将 Pacemaker 集群的某个成员列入黑名单就会导致集群出现问题。

- 不要在更新或升级过程中使用黑名单。这些过程本身有一些方法可将更改操作与特定服务器进行隔离。

- 将服务器加入黑名单后，不允许再对这些节点进行更改操作，除非将服务器从黑名单中移除。这包括更新、升级、扩展、缩减和节点替换等操作。例如，如果在使用新的 Compute 节点扩展 overcloud 时将现有 Compute 节点列入黑名单，则列入黑名单的节点会错过添加到 /etc/hosts 和 /etc/ssh/ssh_known_hosts 中的信息。这可能导致实时迁移失败，具体取决于目标主机。在下一次 overcloud 部署过程中，利用添加到 /etc/hosts 和 /etc/ssh/ssh_known_hosts 中的信息更新 Compute 节点，这些节点不再列入黑名单。不要手动修改 /etc/hosts 和 /etc/ssh/ssh_known_hosts 文件。要修改 /etc/hosts 和 /etc/ssh/ssh_known_hosts 文件，请运行 overcloud 部署命令，如清除黑名单一节中所述。

清除黑名单

要清除黑名单以便对其中节点执行后续的栈操作，可编辑 **DeploymentServerBlacklist**，使其成为空阵列：

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

警告

不要省略 **DeploymentServerBlacklist** 参数。如果省略该参数，overcloud 部署将使用先前保存的参数值。

第 17 章 替换 CONTROLLER 节点

在一些情况下，高可用性集群中的 Controller 节点可能会出现故障。在这种情况下，您需要把这个节点从集群中删除，并替换为一个新的 Controller 节点。

完成本节中的步骤来替换 Controller 节点。在 Controller 节点替换过程中，需要运行 openstack overcloud deploy 命令，以使用替换 Controller 节点的请求来更新 overcloud。

重要

以下操作过程仅适用于高可用性环境。在只使用一个 Controller 节点的情况下不要使用此过程。

17.1. 准备替换 CONTROLLER 节点

在替换 overcloud 控制器节点前，务必要检查Red Hat OpenStack Platform 环境的当前状态；此检查有助于避免在替换控制器节点的过程中出现混乱。使用以下初步检查列表，确定是否可以安全地执行 Controller 节点替换。在 undercloud 上对这些检查运行所有命令。

步骤

1. 在 undercloud 中检查 overcloud 栈的当前状态：

```
$ source stackrc
(undercloud)$ openstack stack list --nested
```

overcloud 栈以及它们的子栈的状态需要是 CREATE_COMPLETE 或 UPDATE_COMPLETE。

2. 安装数据库客户端工具：

```
(undercloud)$ sudo dnf -y install mariadb
```

3. 配置数据库的 root 用户访问权限：

```
(undercloud)$ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
```

4. 对 **undercloud 数据**库进行备份：

```
(undercloud)$ mkdir /home/stack/backup
(undercloud)$ sudo mysqldump --all-databases --quick --single-transaction | gzip >
/home/stack/backup/dump_db_undercloud.sql.gz
```

5. 检查您的 **undercloud 是否包含 10 GB 可用存储，可在置备新**节点时容纳镜像缓存和转换：

```
(undercloud)$ df -h
```

6. **如果您要为新 Controller** 节点重复使用 **IP 地址，请确保删除旧 Controller 使用的端口：**

```
(undercloud)$ openstack port delete <port>
```

7. **在运行的 Controller** 节点上检查 **Pacemaker 的状态。例如，运行的 Controller** 节点的 **IP 地址是 192.168.0.47，使用以下命令查看 Pacemaker 的状态：**

```
(undercloud)$ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

输出显示所有在现有节点上运行且在故障节点上停止的服务。

8. 检查 **overcloud 的 MariaDB 集群中各个**节点的以下参数：

- **wsrep_local_state_comment: Synced**
- **wsrep_cluster_size: 2**

使用以下命令检查各个运行的 **Controller** 节点的这些参数。在本例中，**Controller** 节点 **IP 地址是 192.168.0.47 和 192.168.0.46：**

```
(undercloud)$ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh heat-
admin@$i "sudo podman exec \$(sudo podman ps --filter name=galera-bundle -q) mysql
-e \"SHOW STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE
'wsrep_cluster_size';\""; done
```

9. 检查 **RabbitMQ 状态。例如，运行的 Controller** 节点的 **IP 地址是 192.168.0.47，使用以下**

命令查看 RabbitMQ 的状态：

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo podman exec \$(sudo podman ps -f
name=rabbitmq-bundle -q) rabbitmqctl cluster_status"
```

running_nodes 键应该只显示两个可用的节点，而不会显示有故障的节点。

10. 如果启用了隔离服务，则将其禁用。例如，如果 192.168.0.47 是运行中 Controller 节点的 IP 地址，则使用以下命令检查隔离服务的状态：

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
```

运行以下命令可禁用隔离服务：

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
```

11. 检查 director 节点上的 Compute 服务是否活跃：

```
(undercloud)$ openstack hypervisor list
```

以上命令的输出应该显示所有没有处于维护模式的节点的状态为 up。

12. 确保所有 undercloud 容器都在运行：

```
(undercloud)$ sudo podman ps
```

13. 停止在故障 Controller 节点上运行的所有 nova the 容器：

```
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api_cron.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_compute.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_conductor.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_metadata.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_placement.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_scheduler.service
```

14. 可选：如果您使用 Bare Metal Service (ironic)作为 virt 驱动程序，您必须为您要删除的实例的任何裸机实例手动更新单元数据库中的服务条目。联系红帽支持以获得帮助。

注意

当使用 Bare Metal Service (ironic)作为 virt 驱动程序时，这个单元数据库手动更新是一个临时临时解决方案，以确保节点被重新平衡，直到 BZ2017980 完成为止。

17.2. 删除 CEPH MONITOR 守护进程

如果 Controller 节点正在运行 Ceph 监控服务，请完成以下步骤以删除 ceph-mon 守护进程。

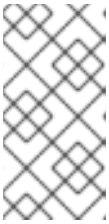
注意

在集群中添加新的 Controller 节点，也会自动添加新的 Ceph 监控器守护进程。

步骤

1. 连接到您要替换的 Controller 节点，并改为 root 用户身份：

```
# ssh heat-admin@192.168.0.47
# sudo su -
```



注意

如果无法连接到该 Controller 节点，请跳过第 1 步和第 2 步，然后在能够正常工作的任意 Controller 节点上从第 3 步开始继续执行这个操作过程。

2. 停止该监控器：

```
# systemctl stop ceph-mon@<monitor_hostname>
```

例如：

```
# systemctl stop ceph-mon@overcloud-controller-1
```

3. 从要替换的 Controller 节点断开连接。

4. 连接到现有 Controller 节点中的一个。

```
# ssh heat-admin@192.168.0.46
# sudo su -
```

5. 从集群中删除该监控器：

```
# sudo podman exec -it ceph-mon-controller-0 ceph mon remove overcloud-controller-1
```

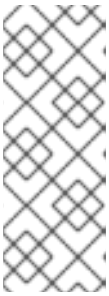
6. 在所有 Controller 节点上，从 /etc/ceph/ceph.conf 中删除 v1 和 v2 监控条目。例如，如果删除 controller-1，则删除 controller-1 的 IP 和主机名。

删除前：

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.22:3300,v1:172.18.0.22:6789],[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
```

删除后：

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-0
```



注意

在添加替换 Controller 节点时，director 会更新相关 overcloud 节点上的 ceph.conf 文件。通常，这个配置文件由 director 独占管理，您不应手动编辑。不过，如果在您添加新节点前其他节点重新启动，若要确保一致性，您可以手动编辑该文件。

7. （可选）归档监控器数据，并将其保存到其他服务器上：

```
# mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-
<daemon_id>
```

17.3. 为 CONTROLLER 节点替换准备集群

在替换旧节点前，您必须确保节点上没有运行 Pacemaker，然后将该节点从 Pacemaker 集群中删除。

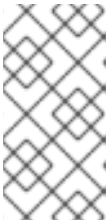
步骤

1. 要查看 Controller 节点的 IP 地址列表，请运行以下命令：

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+------------------------+
| Name                   | Networks               |
+------------------------+------------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-1 | ctlplane=192.168.0.45 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
+------------------------+------------------------+
```

2. 如果旧节点仍然可以连接，请登录其中一个剩余的节点，并停止旧节点上的 pacemaker。在本例中，请停止 overcloud-controller-1 上的 pacemaker：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs status | grep -w Online | grep -w overcloud-controller-1"
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster stop overcloud-controller-1"
```



注意

如果旧节点在物理上不可用或者已经停止，则不必进行前面的操作，因为该节点上 pacemaker 已经停止。

3. 在旧节点上停止 Pacemaker 后，从 pacemaker 集群中删除旧节点。以下示例命令登录到 overcloud-controller-0 以删除 overcloud-controller-1：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-controller-1"
```

如果要替换的节点无法访问（例如，由于硬件故障），请运行 pcs 命令并使用附加 --skip-offline 和 --force 选项从集群中强制删除该节点：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-controller-1 --skip-offline --force"
```

4. 从 pacemaker 集群中删除旧节点后，再从 pacemaker 中的已知主机列表中删除该节点：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs host deauth overcloud-controller-1"
```

无论节点是否可访问，都可运行该命令。

5. 要确保新 Controller 节点在替换后使用正确的 STONITH 隔离设备，请输入以下命令从节点中删除旧设备：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs stonith delete <stonith_resource_name>"
```

- 将 <stonith_resource_name> 替换为与旧节点对应的 STONITH 资源的名称。资源名称使用 <resource_agent>-<host_mac> 格式。您可以在 fence.yaml 文件的 FencingConfig 部分查找资源代理和主机 MAC 地址。

6. overcloud 数据库必须在替换过程中继续运行。为了确保 Pacemaker 不会在此过程中停止 Galera，可选择一个运行中的 Controller 节点，然后使用该 Controller 节点的 IP 地址在 undercloud 上运行以下命令：

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs resource unmanage galera-bundle"
```

17.4. 替换 CONTROLLER 节点

要替换 Controller 节点，请确定您要替换的节点的索引。

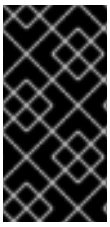
- 如果节点是虚拟节点，请确定包含故障磁盘的节点，然后从备份中恢复磁盘。确保用于故障服务器上 PXE 引导的 NIC 的 MAC 地址在磁盘替换后保持不变。

- 如果该节点是裸机节点，请替换磁盘，利用您的 overcloud 配置准备新磁盘，然后对新硬件上执行节点内省。

- 如果节点是带有隔离功能的高可用性集群的一部分，您可能需要单独恢复 Galera 节点。有关更多信息，请参阅文章 Galera 的工作原理以及如何在 Red Hat OpenStack Platform 中救援

Galera 集群。

完成以下的示例步骤，将 overcloud-controller-1 节点替换为 overcloud-controller-3 节点。overcloud-controller-3 节点的 ID 是 75b25e9a-948d-424a-9b3b-f0ef70a6eacf。



重要

要将节点替换为现有的裸机节点，请在传出节点上启用维护模式，以便 director 不会自动重新置备节点。

步骤

1. *Source stackrc 文件：*

```
$ source ~/stackrc
```

2. *确定 overcloud-controller-1 节点的索引：*

```
$ INSTANCE=$(openstack server list --name overcloud-controller-1 -f value -c ID)
```

3. *确定与实例关联的裸机节点：*

```
$ NODE=$(openstack baremetal node list -f csv --quote minimal | grep $INSTANCE | cut -f1 -d,)
```

4. *把节点设为维护模式：*

```
$ openstack baremetal node maintenance set $NODE
```

5. *如果 Controller 节点是虚拟节点，请在 Controller 主机上运行以下命令，从备份中替换虚拟磁盘：*

```
$ cp <VIRTUAL_DISK_BACKUP> /var/lib/libvirt/images/<VIRTUAL_DISK>
```

- *将 <VIRTUAL_DISK_BACKUP> 替换为故障虚拟磁盘备份的路径，然后将 <VIRTUAL_DISK> 替换为要替换的虚拟磁盘的名称。*

如果您没有传出节点的备份，必须使用新的虚拟化节点。

如果 Controller 节点是裸机节点，请完成下列步骤，将磁盘替换为新的裸机磁盘：

a. 更换物理硬盘或固态硬盘驱动器。

b. 使用与故障节点相同的配置来准备节点。

6. 列出未关联的节点，并确定新节点的 ID：

```
$ openstack baremetal node list --unassociated
```

7. 使用 control 配置集标记新节点：

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

17.5. 替换 BOOTSTRAP CONTROLLER 节点

如果要替换用于 bootstrap 操作的 Controller 节点并保留节点名称，请完成以下步骤，以在替换过程后设置 bootstrap Controller 节点的名称。

步骤

1. 查找 bootstrap Controller 节点的名称：

```
$ ssh heat-admin@<controller_ip> "sudo hiera -c /etc/puppet/hiera.yaml
pacemaker_short_bootstrap_node_name"
```

- 将 <controller_ip> 替换为任何活跃 Controller 节点的 IP 地址。

2. 检查您的环境文件是否包含 ExtraConfig 部分。如果 ExtraConfig 参数不存在，请创建以下环境文件 ~/templates/bootstrap-controller.yaml 并添加以下内容：

```
parameter_defaults:
  ExtraConfig:
    pacemaker_short_bootstrap_node_name: <node_name>
    mysql_short_bootstrap_node_name: <node_name>
```

- 将 < node_name > 替换为您要在替换过程后在 bootstrap 操作中使用的现有 Controller 节点的名称。

  如果您的环境文件已经包含 ExtraConfig 参数，请只添加设置 pacemaker_short_bootstrap_node_name 和 mysql_short_bootstrap_node_name 参数的行。

3. 按照以下步骤触发 Controller 节点替换，并在 overcloud deploy command 中包含环境文件。如需更多信息，请参阅 触发 Controller 节点替换。

有关对 bootstrap Controller 节点替换进行故障排除的信息，请参阅文章如果相同的主机名用于新节点，则第 1 步中替换第一个 Controller 节点会失败。

17.6. 在替换使用可预测的 IP 地址和 HOSTNAMEMAP 的节点时保留主机名

如果将 overcloud 配置为使用可预测的 IP 地址，并且 HostNameMap 将基于 heat 的主机名映射到预置备节点的主机名，则必须配置 overcloud，以将新的替换节点索引映射到 IP 地址和主机名。

流程

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

```
$ source ~/stackrc
```

3. 检索您要替换资源的 physical_resource_id 和 removed_rsrc_list ：

```
(undercloud)$ openstack stack resource show <stack> <role>
```

- 将 <stack> 替换为资源所属的堆栈的名称，如 overcloud。

- 将 & lt;role> 替换为您要替换节点的角色的名称，如 Compute。

  输出示例：

```
+----------------------+--------------------------------------------------------------+
| Field                | Value                                                        |
+----------------------+--------------------------------------------------------------+
| attributes           | {u'attributes': None, u'refs': None, u'refs_map': None,      |
|                      | u'removed_rsrc_list': [u'2', u'3']}                          | 1
| creation_time        | 2017-09-05T09:10:42Z                                         |
| description          |                                                              |
| links                | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9    |
|                      | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-             |
|                      | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':           |
|                      | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da    |
|                      | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-    |
|                      | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h    |
|                      | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874    |
|                      | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-           |
|                      | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]            |
| logical_resource_id  | Compute                                                      |
| physical_resource_id | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                         |
| required_by          | [u'AllNodesDeploySteps',                                     |
|                      | u'ComputeAllNodesValidationDeployment',                      |
|                      | u'AllNodesExtraConfig', u'ComputeIpListMap',                 |
|                      | u'ComputeHostsDeployment', u'UpdateWorkflow',                |
|                      | u'ComputeSshKnownHostsDeployment', u'hostsConfig',           |
|                      | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']        |
| resource_name        | Compute                                                      |
| resource_status      | CREATE_COMPLETE                                              |
| resource_status_reason | state changed                                              |
| resource_type        | OS::Heat::ResourceGroup                                      |
| updated_time         | 2017-09-05T09:10:42Z                                         |
+----------------------+--------------------------------------------------------------+
```

1

removed_rsrc_list 列出了已经为资源删除的节点索引。

4. 检索 resource_name，以确定 heat 应用到此资源节点的最大索引：

```
(undercloud)$ openstack stack resource list <physical_resource_id>
```

- 使用在第 3 步中获取的 ID 替换 <physical_resource_id>。

5. 使用 resource_name 和 removed_rsrc_list 确定 heat 将应用到新节点的下一个索引：

   - 如果 removed_rsrc_list 为空，则下一个索引将是(current_maximum_index)+ 1。
   - 如果 removed_rsrc_list 包含值(current_maximum_index)+ 1，则下一个索引将是下一个可用索引。

6. 检索替换裸机节点的 ID：

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

7. 使用新索引更新替换节点的功能：

```
openstack baremetal node set --property capabilities='node:<role>-<index>,boot_option:local'
<node>
```

   - 将 **<role>** 替换为您要替换节点的角色的名称，如 **compute**。
   - 将 **& lt;index& gt;** 替换为在第 5 步中计算的索引。
   - 将 **<node>** 替换为裸机节点的 ID。

   计算调度程序使用节点功能与部署时的节点匹配。

8. 通过在 **HostnameMap** 配置中添加索引来为新节点分配主机名，例如：

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0 (1)
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0 (2)
```

```
overcloud-compute-0: overcloud-compute-prod-abc-0
overcloud-compute-3: overcloud-compute-prod-abc-3 3
overcloud-compute-8: overcloud-compute-prod-abc-3 4
....
```

1 *您要删除并替换为新节点的节点。*

2 *新节点。*

3 *您要删除并替换为新节点的节点。*

4 *新节点。*

注意

不要删除从 HostnameMap 中删除的节点映射。

9. **将替换节点的 IP 地址添加到网络 IP 地址映射文件 ips-from-pool-all.yaml 中的每个网络 IP 地址列表的末尾。在以下示例中，新索引 overcloud-controller-3 的 IP 地址添加到每个 ControllerIPs 网络的 IP 地址列表中，并分配与 overcloud-controller-1 相同的 IP 地址，因为它替换了 overcloud-controller-1。新索引 overcloud-compute-8 的 IP 地址也添加到每个 ComputeIPs 网络的 IP 地址列表中，并分配与它替换的索引相同的 IP 地址：**

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10 1
      - 192.168.1.11 2
      - 192.168.1.12 3
      - 192.168.1.11 4
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
```

```
...

ComputeIPs:
  ...
  internal_api:
    - 172.17.0.10 5
    - 172.17.0.11 6
    - 172.17.0.11 7
  ...
  storage:
    - 172.17.0.10
    - 172.17.0.11
    - 172.17.0.11
  ...
```

1

分配给索引 0 的 IP 地址，主机名为 overcloud-controller-prod-123-0。

2

分配给索引 1 的 IP 地址，主机名 overcloud-controller-prod-456-0。此节点由索引 3 替代。不要删除此条目。

3

分配给索引 2 的 IP 地址，主机名为 overcloud-controller-prod-789-0。

4

分配给索引 3 的 IP 地址，主机名为 overcloud-controller-prod-456-0。这是替换索引 1 的新节点。

5

分配给索引 0 的 IP 地址，主机名为 overcloud-compute-0。

6

分配给索引 1 的 IP 地址，主机名为 overcloud-compute-3。此节点由索引 2 替代。不要删除此条目。

7

分配给索引 2 的 IP 地址，主机名为 overcloud-compute-8。这是替换索引 1 的新节点。

17.7. 触发 CONTROLLER 节点替换

完成以下步骤，以删除旧的 Controller 节点，并将它替换为新的 Controller 节点。

流程

1. 确定您要删除的 Controller 节点的 UUID，并将其存储在 < NODEID& gt; 变量中。确保将 & lt;node_name > 替换为您要删除的节点的名称：

```
(undercloud)[stack@director ~]$ NODEID=$(openstack server list -f value -c ID --name
<node_name>)
```

2. 要识别 Heat 资源 ID，请输入以下命令：

```
(undercloud)[stack@director ~]$ openstack stack resource show overcloud ControllerServers
-f json -c attributes | jq --arg NODEID "$NODEID" -c '.attributes.value | keys[] as $k | if .[$k]
== $NODEID then "Node index \($k) for \(.[$k])" else empty end'
```

3. 创建以下环境文件 ~/templates/remove-controller.yaml，并包含您要删除的 Controller 节点的节点索引：

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['<node_index>']}]
```

4. 输入 overcloud 部署命令，在命令中包含 remove-controller.yaml 环境文件以及所有与您环境相关的其他环境文件：

```
(undercloud) $ openstack overcloud deploy --templates \
    -e /home/stack/templates/remove-controller.yaml \
    [OTHER OPTIONS]
```

注意

- 仅对部署命令的这个实例包含 **-e ~/templates/remove-controller.yaml**。从后续的部署操作中移除此环境文件。
- 如果要替换 **bootstrap Controller** 节点并希望保留节点名称，请包含 **~/templates/bootstrap-controller.yaml**。如需更多信息，请参阅 *替换 bootstrap Controller 节点*。

5. **director** 会移除旧节点，创建一个新节点并更新 **overcloud** 堆栈。您可以使用以下命令检查 **overcloud** 栈的状态：

```
(undercloud)$ openstack stack list --nested
```

6. 部署命令完成后，确认旧节点已替换为新节点：

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+-----------------------+
| Name                   | Networks              |
+------------------------+-----------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
| overcloud-controller-3 | ctlplane=192.168.0.48 |
+------------------------+-----------------------+
```

新节点现在托管运行的 **control plane** 服务。

17.8. CONTROLLER 节点替换后清理

完成节点替换后，执行以下步骤来完善 **Controller** 集群。

步骤

1. 登录 **Controller** 节点。
2. 启用 **Galera** 集群的 **Pacemaker** 管理，并在新节点上启动 **Galera**：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
```

3. 执行最后的状态检查来确保服务在正确运行：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs status
```

注意

如果有任何服务失败，请使用 **pcs resource refresh** 命令来解决问题并重新启动失败的服务。

4. 退出 **director**：

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

5. 查找 **overcloudrc** 文件，以便您可以跟 **overcloud** 交互：

```
$ source ~/overcloudrc
```

6. 检查 **overcloud** 环境中的网络代理：

```
(overcloud) $ openstack network agent list
```

7. 如果出现任何旧节点的代理，请删除它们：

```
(overcloud) $ for AGENT in $(openstack network agent list --host overcloud-controller-
1.localdomain -c ID -f value) ; do openstack network agent delete $AGENT ; done
```

8. 如有必要，将您的路由器添加到新节点上的 **3** 层代理主机。使用以下示例命令，通过 **UUID 2d1c1dc1-d9d4-4fa9-b2c8-f29cd1a649d4** 将名为 **r1** 的路由器添加到 **L3** 代理中：

```
(overcloud) $ openstack network agent add router --l3 2d1c1dc1-d9d4-4fa9-b2c8-
f29cd1a649d4 r1
```

9. 清理 **cinder** 服务。

a. ***列出 cinder 服务：***

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

b. ***登录到控制器节点，连接到 cinder-api 容器，并使用 cinder-manage service remove 命令删除左侧服务：***

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
service remove cinder-backup <host>
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
service remove cinder-scheduler <host>
```

10. ***清理 RabbitMQ 集群。***

a. ***登录 Controller 节点。***

b. ***使用 podman exec 命令启动 bash，并验证 RabbitMQ 集群的状态：***

```
[heat-admin@overcloud-controller-0 ~]$ podman exec -it rabbitmq-bundle-podman-0
bash
[heat-admin@overcloud-controller-0 ~]$ rabbitmqctl cluster_status
```

c. ***使用 rabbitmqctl 命令忘记替换的控制器节点：***

```
[heat-admin@overcloud-controller-0 ~]$ rabbitmqctl forget_cluster_node <node_name>
```

11. ***如果替换了 bootstrap Controller 节点，则必须在替换过程后移除环境文件 ~/templates/bootstrap-controller.yaml，或者从现有环境文件中移除 pacemaker_short_bootstrap_node_name 和 mysql_short_bootstrap_node_name 参数。此步骤可防止 director 在后续替换中尝试覆盖 Controller 节点名称。如需更多信息，请参阅 替换 bootstrap 控制器节点。***

第 18 章 重新引导节点

您可能需要在 undercloud 和 overcloud 中重新引导节点。使用以下步骤了解如何重新引导不同的节点类型。

- 如果重新引导一个角色中的所有节点，建议单独重新引导各节点。如果您同时重新引导角色中的所有节点，在重新引导操作过程中可能会发生服务停机。
- 如果重新引导 OpenStack Platform 环境中的所有节点，则按照以下顺序重新引导节点：

建议的节点重新引导顺序

1. 重新引导 undercloud 节点。
2. 重新引导 Controller 节点和其他可组合节点。
3. 重新引导独立 Ceph MON 节点。
4. 重新引导 Ceph Storage 节点。
5. 重新引导 Object Storage 服务(swift)节点。
6. 重新引导 Compute 节点。

18.1. 重新引导 UNDERCLOUD 节点

完成以下步骤以重新引导 undercloud 节点。

步骤

1. 以 stack 用户的身份登录 undercloud。

2. 重新引导 undercloud：

```
$ sudo reboot
```

3. 稍等片刻，直到节点启动。

18.2. 重新引导 CONTROLLER 和可组合节点

根据可组合角色重新引导 Controller 节点和独立节点，并排除 Compute 节点和 Ceph Storage 节点。

流程

1. 登录您要重新引导的节点。

2. 可选：如果节点使用 Pacemaker 资源，请停止集群：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster stop
```

3. 重新引导节点：

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

4. 稍等片刻，直到节点启动。

verfication

1. 验证服务是否已启用。

   a. 如果该节点使用 Pacemaker 服务，请检查该节点是否已重新加入集群：

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo pcs status
   ```

   b. 如果该节点使用 Systemd 服务，请检查是否所有服务都已启用：

```
[heat-admin@overcloud-controller-0 ~]$ sudo systemctl status
```

c. 如果该节点使用容器化服务，则检查节点上的所有容器是否已激活：

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman ps
```

18.3. 重新引导独立 CEPH MON 节点

完成以下步骤以重新引导独立 Ceph MON 节点。

步骤

1. 登录到一个 Ceph MON 节点。
2. 重新引导节点：

```
$ sudo reboot
```

3. 等待节点被引导并重新加入 MON 集群。

对集群中的每个 MON 节点重复这些步骤。

18.4. 重新引导 CEPH STORAGE (OSD) 集群

完成以下步骤以重新引导 Ceph Storage (OSD) 节点集群。

流程

1. 登录到 Ceph MON 或 Controller 节点，并临时禁用 Ceph 存储集群重新平衡：

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
```

注意

如果您有多堆栈或分布式计算节点(DCN)架构，您必须在设置 noout 和 norebalance 标志时指定集群名称。例如： sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>

2. 选择第一个要重新引导的 Ceph Storage 节点并登录到该节点。

3. 重新引导节点：

```
$ sudo reboot
```

4. 稍等片刻，直到节点启动。

5. 登录到节点并检查集群状态：

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

确认 pgmap 报告的所有 pgs 的状态是否都正常 (active+clean)。

6. 注销节点，重新引导下一个节点，并检查其状态。重复此过程，直到您已重新引导所有 Ceph Storage 节点。

7. 完成后，登录 Ceph MON 或 Controller 节点并重新启用集群重新平衡：

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
```

注意

如果您有多堆栈或分布式计算节点(DCN)架构，您必须在取消设置 noout 和 norebalance 标志时指定集群名称。例如： sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>

8. 执行最后的状态检查，确认集群报告 HEALTH_OK：

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

18.5. 重新引导 COMPUTE 节点

为确保 Red Hat OpenStack Platform 环境中实例的最小停机时间，迁移实例工作流 概述了您要从您要重新引导的 Compute 节点迁移实例所需的步骤。

注意

如果您没有将实例从源 Compute 节点迁移到另一个 Compute 节点，则实例可能会在源 Compute 节点上重启，这可能会导致升级失败。这与 Podman 和 libvirt 服务更改的已知问题相关：

- BZ the106 - tripleo_nova_libvirt 重启两次后的 podman panic
- BZ11410135 - tripleo_nova_libvirt 重启两次后的 podman panic

迁移实例工作流

1. 决定是否在重新引导节点前将实例迁移到另一个 Compute 节点。
2. 选择并禁用您要重新引导的 Compute 节点，使其不调配新实例。
3. 将实例迁移到另一个 Compute 节点中。
4. 重新引导空的 Compute 节点。
5. 启用空的 Compute 节点。

先决条件

- 重启 Compute 节点之前，必须决定是否在节点重启过程中将实例迁移到另一个 Compute 节

点。

查看在 Compute 节点之间迁移虚拟机实例时可能会遇到的迁移约束列表。如需更多信息，请参阅为实例创建配置 Compute Service 中的迁移限制。

- 如果您无法迁移实例，则可设置以下核心模板参数以在 Compute 节点重启后控制实例的状态：

  NovaResumeGuestsStateOnHostBoot

  确定重新引导后是否将实例返回 Compute 节点上的相同状态。设为 False 时，实例保持关闭，必须手动启动。默认值为 False。

  NovaResumeGuestsShutdownTimeout

  重启前等待实例被关闭的时间（以秒为单位）。建议不要将此值设置为 0。默认值为 300。

  有关 overcloud 参数及其用法的更多信息，请参阅 Overcloud 参数。

流程

1. 以 stack 用户的身份登录 undercloud。

2. 列出所有的 Compute 节点及其 UUID：

```
$ source ~/stackrc
(undercloud) $ openstack server list --name compute
```

   识别您要重新引导的 Compute 节点的 UUID。

3. 从 undercloud 中，选择一个 Compute 节点并禁用它：

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service list
(overcloud) $ openstack compute service set <hostname> nova-compute --disable
```

4. 列出 Compute 节点上的所有实例：

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

5. 可选：如果您决定将实例迁移到另一个 Compute 节点，请完成以下步骤：

a. 如果您决定将实例迁移至另一个 Compute 节点，则使用以下命令之一：

- 要将实例迁移到其他主机，请运行以下命令：

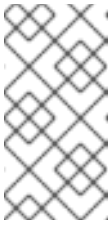
```
(overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
```

- 让 nova-scheduler 自动选择目标主机：

```
(overcloud) $ nova live-migration <instance_id>
```

- 一次性实时迁移所有实例：

```
$ nova host-evacuate-live <hostname>
```



注意

nova 命令可能会引发一些弃用警告，这些警告信息可以被安全忽略。

b. 稍等片刻，直至迁移完成。

c. 确认迁移成功完成：

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

d. 继续迁移实例，直到 Compute 节点上不剩任何实例。

6. *登录到 Compute 节点并重新引导节点：*

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

7. *稍等片刻，直到节点启动。*

8. *重新启用 Compute 节点：*

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

9. *确认是否已启用 Compute 节点：*

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

第 19 章 关闭并启动 UNDERCLOUD 和 OVERCLOUD

如果必须对 undercloud 和 overcloud 执行维护，您必须按照特定顺序关闭和启动 undercloud 和 overcloud 节点，以确保启动 overcloud 时出现的问题很少。

先决条件

- 运行 undercloud 和 overcloud

19.1. UNDERCLOUD 和 OVERCLOUD 关闭顺序

要关闭 Red Hat OpenStack Platform 环境，您必须按照以下顺序关闭 overcloud 和 undercloud：

1. 关闭 overcloud Compute 节点上的实例
2. 关闭 Compute 节点
3. 停止 Controller 节点上的所有高可用性和 OpenStack Platform 服务
4. 关闭 Ceph Storage 节点
5. 关闭 Controller 节点
6. 关闭 undercloud

19.2. 关闭 OVERCLOUD COMPUTE 节点上的实例

作为关闭 Red Hat OpenStack Platform 环境的一部分，在关闭 Compute 节点之前关闭 Compute 节点上的所有实例。

先决条件

- 具有活跃 Compute 服务的 overcloud

步骤

1. 以 stack 用户身份登录 undercloud。
2. 提供 overcloud 的凭据文件：

```
$ source ~/overcloudrc
```

3. 查看 overcloud 中运行的实例：

```
$ openstack server list --all-projects
```

4. 停止 overcloud 中的每个实例：

```
$ openstack server stop <INSTANCE>
```

对每个实例重复这一步，直到停止 overcloud 中的所有实例。

19.3. 关闭 COMPUTE 节点

作为关闭 Red Hat OpenStack Platform 环境的一部分，登录并关闭每个 Compute 节点。

先决条件

- 关闭 Compute 节点上的所有实例：

步骤

1. 以 root 用户身份登录 Compute 节点。
2. 关闭该节点：

```
# shutdown -h now
```

3. 对每个 Compute 节点执行这些步骤，直到关闭所有 Compute 节点。

19.4. 停止 CONTROLLER 节点上的服务

作为关闭 Red Hat OpenStack Platform 环境的一部分，在关闭 Controller 节点前停止节点上的服务。这包括 Pacemaker 和 systemd 服务。

先决条件

- 具有活跃 Pacemaker 服务的 overcloud

步骤

1. 以 root 用户身份登录 Controller 节点。

2. 停止 Pacemaker 集群。

```
# pcs cluster stop --all
```

此命令停止所有节点上的集群。

3. 等待 Pacemaker 服务停止并检查服务是否已停止。

   a. 检查 Pacemaker 状态：

```
# pcs status
```

   b. 检查 Podman 中没有 Pacemaker 服务在运行：

```
# podman ps --filter "name=.*-bundle.*"
```

4. 停止 Red Hat OpenStack Platform 服务：

```
# systemctl stop 'tripleo_*'
```

5. 等待服务停止，检查 Podman 中服务不再运行：

```
# podman ps
```

19.5. 关闭 CEPH STORAGE 节点

作为关闭 Red Hat OpenStack Platform 环境的一部分，禁用 Ceph Storage 服务，然后登录并关闭每个 Ceph Storage 节点。

先决条件

- 正常运行的 Ceph Storage 集群
- Ceph MON 服务在单机 Ceph MON 节点或 Controller 节点上运行

步骤

1. 以 root 用户身份登录运行 Ceph MON 服务的节点，如 Controller 节点或单机 Ceph MON 节点。

2. 检查集群的运行状况。在以下示例中，podman 命令在 Controller 节点上的 Ceph MON 容器中运行状态检查：

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

确保状态为 HEALTH_OK。

3. 为集群设置 noout、norecover、norebalance、nobackfill、nodown 和 pause 标志。在以下示例中，podman 命令通过 Controller 节点上的 Ceph MON 容器设置这些标志：

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd set pause
```

4. 关闭每个 Ceph Storage 节点：

   a. 以 root 用户身份登录 Ceph Storage 节点。

   b. 关闭该节点：

   ```
   # shutdown -h now
   ```

   c. 对每个 Ceph Storage 节点执行这些步骤，直到关闭所有 Ceph Storage 节点。

5. 关闭任何单机 Ceph MON 节点：

   a. 以 root 用户身份登录单机 Ceph MON 节点。

   b. 关闭该节点：

   ```
   # shutdown -h now
   ```

   c. 对每个单机 Ceph MON 节点执行这些步骤，直到关闭所有单机 Ceph MON 节点。

其他资源

- “关闭并启动整个 Ceph 集群的步骤是什么？”

19.6. 关闭 CONTROLLER 节点

作为关闭 Red Hat OpenStack Platform 环境的一部分，登录并关闭每个 Controller 节点。

先决条件

- 停止 Pacemaker 集群

- 停止 Controller 节点上的所有 Red Hat OpenStack Platform 服务

步骤

1. 以 root 用户身份登录 Controller 节点。
2. 关闭该节点：

```
# shutdown -h now
```

3. 对每个 Controller 节点执行这些步骤，直到关闭所有 Controller 节点。

19.7. 关闭 UNDERCLOUD

作为关闭 Red Hat OpenStack Platform 环境的一部分，登录到 undercloud 节点并关闭 undercloud。

先决条件

- 正在运行的 undercloud

步骤

1. 以 stack 用户身份登录 undercloud。
2. 关闭 undercloud：

```
$ sudo shutdown -h now
```

19.8. 执行系统维护

在完全关闭 undercloud 和 overcloud 后，对环境中的系统执行任何维护，然后启动 undercloud 和 overcloud。

19.9. UNDERCLOUD 和 OVERCLOUD 启动顺序

要启动 Red Hat OpenStack Platform 环境，您必须按照以下顺序启动 undercloud 和 overcloud：

1. *启动 undercloud。*
2. *启动 Controller 节点。*
3. *启动 Ceph Storage 节点。*
4. *启动 Compute 节点。*
5. *启动 overcloud Compute 节点上的实例。*

19.10. 启动 UNDERCLOUD

作为启动 Red Hat OpenStack Platform 环境的一部分，启动 undercloud 节点，登录到 undercloud，再检查 undercloud 服务。

先决条件

- *undercloud 已关机。*

流程

- *打开 undercloud 并等待 undercloud 引导。*

验证

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

```
$ source ~/stackrc
```

3. 检查 **undercloud** 上的服务：

```
$ systemctl list-units 'tripleo_*'
```

4. 创建并验证名为 **inventory.yaml** 的静态清单文件：

```
$ tripleo-ansible-inventory --static-yaml-inventory inventory.yaml
$ openstack tripleo validator run --group pre-introspection \
 -i inventory.yaml
```

5. 检查所有服务和容器是否活跃且健康：

```
$ openstack tripleo validator run --validation service-status \
 --limit undercloud -i inventory.yaml
```

其他资源

- 使用验证框架

19.11. 启动 CONTROLLER 节点

作为启动 **Red Hat OpenStack Platform** 环境的一部分，打开每个 **Controller** 节点电源，并检查节点上的非 **Pacemaker** 服务。

先决条件

- **Controller** 节点已关闭。

流程

- 打开每个 **Controller** 节点电源。

验证

1. 以 **root** 用户身份登录每个 **Controller** 节点。

2. 检查 Controller 节点上的服务：

```
$ systemctl -t service
```

只有基于非 Pacemaker 的服务正在运行。

3. 等待 Pacemaker 服务启动并检查服务是否已启动：

```
$ pcs status
```

注意

如果您的环境使用 Instance HA，Pacemaker 资源不会启动，直到您启动 Compute 节点，或使用 pcs stonith confirm <compute_node> 命令执行手动取消隔离操作。您必须在使用 Instance HA 的每个 Compute 节点上运行此命令。

19.12. 启动 CEPH STORAGE 节点

作为启动 Red Hat OpenStack Platform 环境的一部分，打开 Ceph MON 和 Ceph Storage 节点电源，并启用 Ceph Storage 服务。

先决条件

- 已关闭电源的 Ceph Storage 集群
- Ceph MON 服务在已关闭电源的单机 Ceph MON 节点或已打开电源的 Controller 节点上启用

步骤

1. 如果您的环境有单机 Ceph MON 节点，请打开每个 Ceph MON 节点电源。
2. 打开每个 Ceph Storage 节点电源。
3. 以 root 用户身份登录运行 Ceph MON 服务的节点，如 Controller 节点或单机 Ceph MON

节点。

4. 检查集群节点的状态：在以下示例中，podman 命令在 Controller 节点上的 Ceph MON 容器中运行状态检查：

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

确保每个节点都已打开电源并连接。

5. 为集群取消设置 noout、norecover、norebalance、nobackfill、nodown 和 pause 标志。在以下示例中，podman 命令通过 Controller 节点上的 Ceph MON 容器取消设置这些标志：

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset pause
```

验证

1. 检查集群的运行状况。在以下示例中，podman 命令在 Controller 节点上的 Ceph MON 容器中运行状态检查：

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

确保状态为 HEALTH_OK。

其他资源

- “关闭并启动整个 Ceph 集群的步骤是什么？”

19.13. 启动 COMPUTE 节点

作为启动 Red Hat OpenStack Platform 环境的一部分，打开每个 Compute 节点电源并检查节点上的服务。

先决条件

- 关闭 Compute 节点电源

步骤

1. 打开每个 Compute 节点电源。

验证

1. 以 root 用户身份登录每个 Compute。
2. 检查 Compute 节点上的服务：

```
$ systemctl -t service
```

19.14. 启动 OVERCLOUD COMPUTE 节点上的实例

作为启动 Red Hat OpenStack Platform 环境的一部分，启动 Compute 节点上的实例。

先决条件

- 具有活跃节点的活跃 overcloud

步骤

1. 以 stack 用户身份登录 undercloud。
2. 提供 overcloud 的凭据文件：

```
$ source ~/overcloudrc
```

3. 查看 overcloud 中运行的实例：

```
$ openstack server list --all-projects
```

4. *启动 **overcloud** 中的实例：*

```
$ openstack server start <INSTANCE>
```

第 20 章 配置自定义 SSL/TLS 证书

您可以手动配置 undercloud，以使用 SSL/TLS 进行公共端点的通信。当您使用 SSL/TLS 手动配置 undercloud 端点时，您要创建安全端点作为概念验证。红帽建议使用证书颁发机构解决方案。

当您使用证书颁发机构(CA)解决方案时，您有生产就绪解决方案，如证书续订、证书撤销列表(CRL)和行业可接受的加密。有关使用 Red Hat Identity Manager (IdM)作为 CA 的详情，请参考在 Ansible 实施 TLS。

如果要使用具有您自己的证书颁发机构的 SSL 证书，您必须完成以下配置步骤。

20.1. 初始化签名主机

签名主机是使用证书颁发机构生成并签名新证书的主机。如果您从未在所选签名主机上创建 SSL 证书，您可能需要初始化该主机，让它能够为新证书签名。

流程

1. /etc/pki/CA/index.txt 文件包含所有签名证书的记录。请确定文件系统路径和 index.txt 文件已存在：

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. /etc/pki/CA/serial 文件标识下一个序列号，以用于下一个要签名的证书。检查是否存在此文件。如果此文件不存在，则使用新启动值创建新文件：

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

20.2. 创建证书颁发机构

一般情况下，您需要使用一个外部的证书认证机构来签发您的 SSL/TLS 证书。在某些情况下，您可能想使用自己的证书颁发机构。例如，您可能想拥有仅限内部使用的证书颁发机构。

流程

1. 生成密钥和证书对以充当证书颁发机构：

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

2. **openssl req 命令会要求输入认证机构的详细信息。根据提示输入相关信息。这些命令创建一个称为 ca.crt.pem 的证书颁发机构文件。**

3. **将证书位置设置为 enable-tls.yaml 文件中的 PublicTLSCAFile 参数的值。当您将证书位置设置为 PublicTLSCAFile 参数的值时，您可以确保 CA 证书路径添加到 clouds.yaml 身份验证文件中。**

```
parameter_defaults:
  PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

20.3. 将此证书颁发机构添加到客户端

对于旨在使用 SSL/TLS 通信的所有外部客户端，将证书颁发机构文件复制到需要访问 Red Hat OpenStack Platform 环境的每个客户端。

流程

1. **将证书颁发机构复制到客户端系统：**

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
```

2. **将证书颁发机构文件复制到每个客户端后，在每个客户端上运行以下命令，将证书添加到证书颁发机构信任捆绑包中：**

```
$ sudo update-ca-trust extract
```

20.4. 创建 SSL/TLS 密钥

在 OpenStack 环境中启用 SSL/TLS 需要一个 SSL/TLS 密钥来生成证书。

流程

1. **运行以下命令以生成 SSL/TLS 密钥 (server.key.pem)：**

```
$ openssl genrsa -out server.key.pem 2048
```

20.5. 创建 SSL/TLS 证书签名请求

完成以下步骤以创建证书签名请求。

流程

1. 复制默认 OpenSSL 配置文件：

```
$ cp /etc/pki/tls/openssl.cnf .
```

2. 编辑新的 openssl.cnf 文件并配置要用于 director 的 SSL 参数。一个要修改的参数类型的示例包括：

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

将 commonName_default 设置为以下条目之一：

- 如果使用 IP 地址通过 SSL/TLS 访问 director，则使用 undercloud.conf 文件中的 undercloud_public_host 参数。

- 如果使用完全限定域名通过 SSL/TLS 访问 director，则使用此域名。

编辑 alt_names 部分，使其包含以下条目：

- IP - 客户端用于通过 SSL 访问 director 的 IP 地址列表。

- DNS - 客户端用于通过 SSL 访问 director 的域名列表。其中也包含公共 API IP 地址作为在 alt_names 部分末尾的 DNS 条目。

注意

有关 openssl.cnf 的更多信息，请运行 man openssl.cnf 命令。

3. 运行以下命令以生成证书签名请求 (server.csr.pem)：

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

确保使用 -key 选项包括 OpenStack SSL/TLS 密钥。

此命令生成 server.csr.pem 文件，这是证书签名请求。使用此文件创建 OpenStack SSL/TLS 证书。

20.6. 创建 SSL/TLS 证书

要为 OpenStack 环境生成 SSL/TLS 证书，必须存在以下文件：

openssl.cnf

指定 v3 扩展的自定义配置文件。

server.csr.pem

生成证书并使用 CA 对证书进行签名的证书签名请求。

ca.crt.pem

对证书进行签名的证书颁发机构。

ca.key.pem

证书颁发机构私钥。

流程

1. *如果尚未存在，请创建 **newcerts** 目录：*

```
sudo mkdir -p /etc/pki/CA/newcerts
```

2. *运行以下命令，为 **undercloud 或 overcloud** 创建证书：*

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

这个命令使用以下选项：

-config

*使用一个自定义配置文件，它是带有 **v3** 扩展的 **openssl.cnf** 文件。*

-extensions v3_req

*已启用的 **v3** 扩展。*

-days

定义证书到期前的天数。

-in'

证书签名请求。

-out

生成的签名证书。

-cert

证书颁发机构文件。

-keyfile

证书颁发机构私钥。

此命令创建名为 **server.crt.pem** 的新证书。将此证书与 **OpenStack SSL/TLS** 密钥一起使用

20.7. 将证书添加到 UNDERCLOUD

完成以下步骤，将 **OpenStack SSL/TLS** 证书添加到 **undercloud** 信任捆绑包中。

流程

1. 运行以下命令以组合证书和密钥：

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

此命令创建 **undercloud.pem** 文件。

2. 将 **undercloud.pem** 文件复制到 **/etc/pki** 目录内的位置，并设置必要的 **SELinux** 上下文，以便 **HAProxy** 能够读取：

```
$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp ~/undercloud.pem /etc/pki/undercloud-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs
```

3. 将 **undercloud.pem** 文件位置添加到 **undercloud.conf** 文件的 **undercloud_service_certificate** 选项中：

```
undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem
```

不要设置或启用 **generate_service_certificate** 和 **certificate_generation_ca** 参数。**director** 使用这些参数自动生成证书，而不使用您手动创建的 **undercloud.pem** 证书。

4. 将签名证书的证书颁发机构添加到 **undercloud** 的可信证书颁发机构列表中，以便

undercloud 内的不同服务能够访问证书颁发机构：

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

要验证证书颁发机构是否已添加到 undercloud，请使用 openssl 检查信任捆绑包：

```
$ openssl crl2pkcs7 -nocrl -certfile /etc/pki/tls/certs/ca-bundle.crt | openssl pkcs7 -print_certs
-text | grep <CN of the CA issuer> -A 10 -B 10
```

- **将 <CN of the CA issuer> 替换为 CA 颁发者的通用名称。此命令输出主要证书详细信息，包括有效期日期。**

第 21 章 其他内省操作

在某些情况下，您可能想要在标准 overcloud 部署工作流外执行内省。例如，在替换现有未使用节点上的硬件后，您可能想要内省新节点或刷新内省数据。

21.1. 执行单个节点内省

要在可用节点上执行单个内省，请将节点设置为管理模式并执行内省。

流程

1. 将所有节点设置为 **manageable** 状态：

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. 执行内省：

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

内省完成后，节点切换为 **available** 状态。

21.2. 在初始内省后执行节点内省操作

因为 **--provide** 选项的原因，所有节点在初始内省后都进入 **available** 状态。要在初始内省后在所有节点上执行内省，请将节点设置为管理模式并执行内省。

流程

1. 将所有节点设置为 **manageable** 状态

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do
openstack baremetal node manage $node ; done
```

2. 运行批量内省命令：

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

内省完成后，所有节点都会变为 available 状态。

21.3. 执行网络内省以查看接口信息

网络内省会从网络交换机获取链路层发现协议 (LLDP) 数据。以下命令可显示某个节点上所有接口的某个 LLDP 信息子集，或显示某个节点和接口的全部信息。这对故障排除非常有用。director 默认会启用 LLDP 数据收集。

流程

1. 要获取节点上的接口列表，请运行以下命令：

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

例如：

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-79f8b86c7cd9
+-----------+-------------------+-----------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs  | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+-----------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                 | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                 | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]            | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+-----------------------+-------------------+----------------+
```

2. 要查看接口数据和交换机端口信息，请运行以下命令：

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID] [INTERFACE]
```

例如：

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-907d-79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------------------------
-----------------------------------------------+
| Field                                | Value
|
+--------------------------------------+------------------------------------------------------------------------------------------------
-----------------------------------------------+
```

```
| interface                    | p2p1
|
| mac                          | 00:0a:f7:79:93:18
|
| node_ident                   | c89397b7-a326-41a0-907d-79f8b86c7cd9
|
| switch_capabilities_enabled  | [u'Bridge', u'Router']
|
| switch_capabilities_support  | [u'Bridge', u'Router']
|
| switch_chassis_id            | 64:64:9b:31:12:00
|
| switch_port_autonegotiation_enabled  | True
|
| switch_port_autonegotiation_support  | True
|
| switch_port_description      | ge-0/0/2.0
|
| switch_port_id               | 507
|
| switch_port_link_aggregation_enabled | False
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type         | Unknown
|
| switch_port_mtu              | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids         | [101]
|
| switch_port_vlans            | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities   | None
|
| switch_system_name           | rhos-compute-node-sw1
|
+--------------------------------------+-----------------------------------------------------------------------
-----------------------------------------------+
```

21.4. 获取硬件内省详细信息

裸机服务 hardware-inspection-extras 功能默认启用，您可以使用它来检索 overcloud 配置的硬件详情。有关 undercloud.conf 文件中的 inspection_extras 参数的更多信息，请参阅配置 director。

例如，numa_topology 收集程序就是硬件检查额外功能的一部分，包括每个 NUMA 节点的以下信息：

- RAM（单位为 KB）
- 物理 CPU 内核数和同级线程数
- 和 NUMA 节点关联的 NIC

流程

- 要获得以上列出的信息，请使用裸机节点的 UUID 替换 <UUID> 来完成以下命令：

```
$ openstack baremetal introspection data save \
<UUID> | jq .numa_topology
```

以下示例显示获取的裸机节点 NUMA 信息：

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
```

```
      0,
      16
    ],
    "numa_node": 0
  },
  {
    "cpu": 5,
    "thread_siblings": [
      13,
      29
    ],
    "numa_node": 1
  },
  {
    "cpu": 7,
    "thread_siblings": [
      15,
      31
    ],
    "numa_node": 1
  },
  {
    "cpu": 7,
    "thread_siblings": [
      7,
      23
    ],
    "numa_node": 0
  },
  {
    "cpu": 1,
    "thread_siblings": [
      9,
      25
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "numa_node": 1
  },
  {
    "cpu": 5,
    "thread_siblings": [
```

```
        5,
        21
      ],
      "numa_node": 0
    },
    {
      "cpu": 4,
      "thread_siblings": [
        12,
        28
      ],
      "numa_node": 1
    },
    {
      "cpu": 4,
      "thread_siblings": [
        4,
        20
      ],
      "numa_node": 0
    },
    {
      "cpu": 0,
      "thread_siblings": [
        8,
        24
      ],
      "numa_node": 1
    },
    {
      "cpu": 6,
      "thread_siblings": [
        14,
        30
      ],
      "numa_node": 1
    },
    {
      "cpu": 3,
      "thread_siblings": [
        3,
        19
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        2,
        18
      ],
      "numa_node": 0
    }
  ],
  "ram": [
    {
```

```
      "size_kb": 66980172,
      "numa_node": 0
     },
     {
      "size_kb": 67108864,
      "numa_node": 1
     }
    ],
    "nics": [
     {
      "name": "ens3f1",
      "numa_node": 1
     },
     {
      "name": "ens3f0",
      "numa_node": 1
     },
     {
      "name": "ens2f0",
      "numa_node": 0
     },
     {
      "name": "ens2f1",
      "numa_node": 0
     },
     {
      "name": "ens1f1",
      "numa_node": 0
     },
     {
      "name": "ens1f0",
      "numa_node": 0
     },
     {
      "name": "eno4",
      "numa_node": 0
     },
     {
      "name": "eno1",
      "numa_node": 0
     },
     {
      "name": "eno3",
      "numa_node": 0
     },
     {
      "name": "eno2",
      "numa_node": 0
     }
    ]
}
```

第 22 章 自动发现裸机节点

您可以使用 auto-discovery 来注册 overcloud 节点并生成它们的元数据，而无需创建 instackenv.json 文件。这种改进可有助于缩短收集节点信息所需时间。例如，如果您使用 auto-discovery，则不核对 IPMI IP 地址，然后创建 instackenv.json。

22.1. 启用自动发现

启用并配置裸机自动发现，以便在使用 PXE 引导时自动发现和导入加入您的置备网络的节点。

流程

1. 在 undercloud.conf 文件中启用裸机自动发现：

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

- enable_node_discovery - 启用之后，任何使用 PXE 来引导内省虚拟内存盘的节点都在 Bare Metal 服务 (ironic) 中自动注册。
- discovery_default_driver - 设置用于已发现节点的驱动程序。例如，ipmi。

2. 将您的 IPMI 凭证添加到 ironic：

   a. 将您的 IPMI 凭证添加到名为 ipmi-credentials.json 的文件。请替换本例中的 SampleUsername、RedactedSecurePassword 和 bmc_address 值，以适应您的环境：

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
```

```
        ]
    }
]
```

3. 将 IPMI 凭证文件导入 ironic：

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

22.2. 测试自动发现

PXE 引导连接到置备网络的节点，以测试裸机自动发现功能。

流程

1. 启动所需节点。

2. 运行 openstack baremetal node list 命令。应该看到新节点以 enrolled 状态列出：

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. 为各个节点设置资源类：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. 为各个节点配置内核和 ramdisk：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. 将所有节点设置为 **available**：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal node provide $NODE ; done
```

22.3. 使用规则发现不同供应商的硬件

如果拥有异构硬件环境，您可以使用内省规则来分配凭证和远程管理凭证。例如，您可能需要单独的发现规则来处理使用 **DRAC** 的 **Dell** 节点。

流程

1. 创建名为 **dell-drac-rules.json** 并包含以下内容的文件：

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true},
      {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
       "value": "Dell Inc."}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  },
  {
    "description": "Set the vendor driver for Dell hardware",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true},
      {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
       "value": "Dell Inc."}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver", "value": "idrac"},
      {"action": "set-attribute", "path": "driver_info/drac_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/drac_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/drac_address",
       "value": "{data[inventory][bmc_address]}"}
```

```
        ]
    }
]
```

- *请替换此例中的用户名和密码值以适合您的环境：*

2. *将规则导入 ironic：*

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

第 23 章 配置自动配置集标记

内省操作会执行一系列的基准数据测试，director 将保存这些测试数据。您可以创建一组策略来以不同方式使用这些数据。可使用多种方式创建使用此数据的策略集：

- 这些策略可识别性能不佳或不稳定的节点并隔离这些节点，使其不在 overcloud 中使用。
- 这些策略可定义是否将节点自动标记到特定配置集。

23.1. 策略文件语法

策略文件使用 JSON 格式，它包括了一组规则。每个规则都定义一个 description、一个 condition 和一个 action。description 是规则的纯文本描述，condition 使用键值模式定义一个评估，action 是条件的执行。

Description

description 是规则的纯文本描述。

例如：

```
"description": "A new rule for my node tagging policy"
```

Conditions

condition 就是使用以下键-值来定义评估：

field

定义要评估的字段：

- memory_mb - 节点的内存大小 (MB)。
- cpus - 节点 CPU 的总线程数。

- *cpu_arch - 节点 CPU 的架构。*
- *local_gb - 节点根磁盘的总存储空间。*

op

指定测试所使用的操作。这包括如下属性：

- *eq - 等于*
- *ne - 不等于*
- *lt - 少于*
- *gt - 多于*
- *le - 少于或等于*
- *ge - 多于或等于*
- *in-net - 检查一个 IP 地址是否在指定的网络中*
- *matches - 需要完全和提供的正则表达式相匹配*
- *contains - 需要一个包括和提供的正则表达式相匹配的值；*
- *is-empty - 检查该字段是否为空*

invert

一个布尔值，用来指定是否对检查结果进行反向处理。

multiple

在存在多个结果的情况下，定义使用的测试。此参数包括如下属性：

- any - 只需要任何一个结果匹配
- all - 需要所有结果都匹配
- first - 需要第一个结果匹配

value

测试中的值。如果项和操作结果为这个值，则条件返回为一个“true”的结果。否则，条件返回 false 的结果。

例如：

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

如果条件为 true，策略将执行一个操作。此操作使用 action 密钥和其他密钥，具体取决于 action 的值：

- fail - 使内省失败。需要一个 message 参数来包括失败的信息。
- set-attribute - 在一个 ironic 节点上设置一个属性。需要一个 path 项，它是到一个 ironic 属性（如 /driver_info/ipmi_address）的路径，以及一个 value 值。
-

- ***set-capability - 在一个 ironic 节点上设置一个能力。需要 name 和 value 字段，它们是新能力的名称和值。这将替换这个功能的现有值。例如，使用它来定义节点配置集。***

- ***extend-attribute - 与 set-attribute 相似，只是在存在相同能力时把这个值附加到当前的值后面。如果同时使用了 unique 参数，则在相同值已存在时不进行任何操作。***

例如：

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

23.2. 策略文件示例

以下是一个包含内省规则的 JSON 文件示例（rules.json）：

```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
```

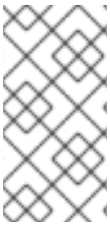
```
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

这个示例包括 3 个规则：

- *如果内存低于 4096 MiB，内省失败。如果您想将某些节点排除在云之外，则可应用这些类型的规则。*
- *硬盘容量大于或等于 1 TiB 的节点会被无条件地分配 swift-storage 配置集。*
- *硬盘容量在 1 TiB 和 40 GiB 间的节点可以作为 Compute 节点或 Controller 节点。您可以分配两个能力（compute_profile 和 control_profile），使 openstack overcloud profiles match*

命令稍后可以作出最终选择。要使此过程成功，必须删除现有配置集能力，否则现有配置集能力具有优先级。

配置集匹配规则不更改任何其他节点。



注意

使用内省规则分配配置集总会覆盖存在的值。但是，对于已经具有配置集能力的节点，会忽略 [PROFILE]_profile 能力。

23.3. 导入策略文件

要将策略文件导入 director，请完成以下步骤。

步骤

1. 将策略文件导入 director：

```
$ openstack baremetal introspection rule import rules.json
```

2. 运行内省进程：

```
$ openstack overcloud node introspect --all-manageable
```

3. 在内省结束后，检查节点以及分配给它们的配置集：

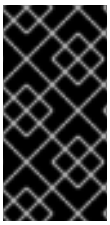
```
$ openstack overcloud profiles list
```

4. 如果在内省规则中出了错，请运行以下命令删除所有规则：

```
$ openstack baremetal introspection rule purge
```

第 24 章 创建完整磁盘镜像

主 overcloud 镜像是不包含分区信息或引导加载程序的平面分区镜像。director 在引导节点时使用单独的内核和 ramdisk，并在将 overcloud 镜像写入磁盘时创建基本分区布局。但是，您可以创建一个完整磁盘镜像，其中包含分区布局、引导加载程序和强化的安全性。



重要

以下过程会使用 director 的镜像构建功能。红帽只支持使用本节中所含的准则构建的镜像。未按这些规范构建的自定义镜像不受支持。

24.1. 安全强化措施

完整磁盘镜像（whole disk image）包括了额外的安全强化措施，可用于在需要高安全性的环境中部署 Red Hat OpenStack Platform。

用于创建镜像的安全建议

- /tmp 目录会挂载到独立的卷或分区上，并包含 rw、nosuid、nodev、noexec 和 relatime 标志。
- /var、/var/log 和 /var/log/audit 目录挂载到单独的卷或分区上，并包含 rw 和 relatime 标志。
- /home 目录挂载到单独的分区或卷上，并包含 rw、nodev 和 relatime 标志。
- 对 GRUB_CMDLINE_LINUX 设置做出以下更改：
  - 要启用审核，可添加 audit=1 内核引导标志。
  - 要禁用使用引导加载程序配置的 USB 的内核支持，请添加 nousb。
  - 要移除不安全的引导标志，请移除 crashkernel=auto。
-

将不安全的模块（usb-storage、cramfs、freevxfs、jffs2、hfs、hfsplus、squashfs、udf、vfat）列入黑名单并防止这些模块加载。

- 从镜像中删除任何不安全的 软件包，因为它们会被默认安装。

24.2. 完整磁盘镜像工作流

要构建完整磁盘镜像，请完成以下工作流：

1. 下载基本 Red Hat Enterprise Linux 8.4 镜像。
2. 设置专门用于注册的环境变量。
3. 通过修改分区的模式和大小来自定义镜像。
4. 创建镜像。
5. 将镜像上传到 director。

24.3. 下载基本云镜像

在构建完整磁盘镜像之前，必须先下载 Red Hat Enterprise Linux 的现有云镜像用作基础。

流程

1. 进入 Red Hat Enterprise Linux 8.4 下载页面。Red Hat OpenStack Platform 16.2 支持 Red Hat Enterprise Linux 8.4。

   - https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.4/x86_64/product-software

注意

如果出现提示，请输入您的客户门户网站登录详情。

2. 点 Red Hat Enterprise Linux 8.4 KVM Guest Image 旁边的 Download Now。

24.4. 启用一致的接口命名

默认情况下，KVM 客户机镜像中禁用一致的网络接口设备命名。使用 virt-customize 启用一致的命名。

流程

1. 将 KVM 客户机镜像移动到 /var/lib/libvirt/images 中：

```
$ sudo mv <kvm_guest_image> /var/lib/libvirt/images/
```

2. 启动 libvirtd ：

```
$ sudo systemctl start libvirtd
```

3. 在 KVM 客户机镜像中启用一致的接口命名：

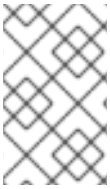
```
$ sudo virt-customize -a /var/lib/libvirt/images/<kvm guest image> --edit
/etc/default/grub:s/net.ifnames=0/net.ifnames=1/
```

4. 停止 libvirtd ：

```
$ sudo systemctl stop libvirtd
```

24.5. 磁盘镜像环境变量

在构建磁盘镜像的过程中，director 需要基础镜像和注册详情，以获取新 overcloud 镜像的软件包。使用以下 Linux 环境变量定义这些属性。



注意

镜像构建过程会利用红帽订阅暂时注册镜像，并在完成构建后取消注册系统。

要构建磁盘镜像，请根据您的环境和要求来设置 Linux 环境变量：

DIB_LOCAL_IMAGE

设置您要用作完整磁盘镜像基础的本地镜像。

REG_ACTIVATION_KEY

使用激活码代替登录详细信息作为注册过程的一部分。

REG_AUTO_ATTACH

定义是否自动附加最兼容的订阅。

REG_BASE_URL

包含镜像软件包的内容交付服务器的基本 URL。默认的客户门户网站订阅管理（Subscription Management）会使用 https://cdn.redhat.com。如果您使用 Red Hat Satellite 6 服务器，则将此参数设置为 Satellite 服务器的基本 URL。

REG_ENVIRONMENT

注册到机构的内部环境中。

REG_METHOD

设置注册方法。使用 portal 可将系统注册到红帽客户门户网站。使用 satellite 可将系统注册到红帽 Satellite 6。

REG_ORG

您想注册镜像的组织。

REG_POOL_ID

产品订阅信息的池 ID。

REG_PASSWORD

为注册镜像的用户帐户设置密码。

REG_RELEASE

设置 Red Hat Enterprise Linux 次要发行版本。您必须将其与 REG_AUTO_ATTACH 或 REG_POOL_ID 环境变量结合使用。

REG_REPOS

包括以逗号分隔的软件仓库名称的字符串。这个字符串中的各个软件仓库会通过 subscription-manager 启用。

对于安全强化型完整磁盘镜像，请使用以下软件仓库：

- rhel-8-for-x86_64-baseos-eus-rpms
- rhel-8-for-x86_64-appstream-eus-rpms
- rhel-8-for-x86_64-highavailability-eus-rpms
- ansible-2.9-for-rhel-8-x86_64-rpms
- fast-datapath-for-rhel-8-x86_64-rpms
- openstack-16.2-for-rhel-8-x86_64-rpms

REG_SAT_URL

注册 overcloud 节点的 Satellite 服务器的基本 URL。此参数需要使用 Satellite 的 HTTP URL 而不是 HTTPS URL。例如，使用 http://satellite.example.com 而不使用 https://satellite.example.com。

REG_SERVER_URL

设置要使用的订阅服务的主机名。默认主机名是红帽客户门户网站（其网址为 subscription.rhn.redhat.com）。如果您使用 Red Hat Satellite 6 服务器，则将此参数设置为 Satellite 服务器的主机名。

REG_USER

为注册镜像的帐户设置用户名。

使用以下示例命令集导出一组环境变量，并将本地 QCOW2 镜像临时注册到红帽客户门户网站：

```
$ export DIB_LOCAL_IMAGE=./rhel-8.4-x86_64-kvm.qcow2
$ export REG_METHOD=portal
$ export REG_USER=<your_name>
$ export REG_PASSWORD=<your_password>
$ export REG_RELEASE="8.4"
$ export REG_POOL_ID=<pool_id>
$ export REG_REPOS="rhel-8-for-x86_64-baseos-eus-rpms \
    rhel-8-for-x86_64-appstream-eus-rpms \
    rhel-8-for-x86_64-highavailability-eus-rpms \
    ansible-2.9-for-rhel-8-x86_64-rpms \
    fast-datapath-for-rhel-8-x86_64-rpms \
    openstack-16.2-for-rhel-8-x86_64-rpms"
```

24.6. 自定义磁盘布局

安全强化型镜像的默认大小为 20G，并会使用预定义的分区大小。但是，必须修改分区布局以容纳 overcloud 容器镜像。完成以下部分中的步骤，将镜像大小增加到 40G。可以修改分区布局和磁盘大小以进一步符合您的需求。

要修改分区布局和磁盘大小，请按照以下步骤操作：

- 使用 DIB_BLOCK_DEVICE_CONFIG 环境变量修改分区模式。
- 通过更新 DIB_IMAGE_SIZE 环境变量，来修改镜像的整体大小。

24.7. 修改分区模式

您可以修改分区模式，以更改分区大小、创建新分区或删除现有分区。使用以下环境变量来定义新的分区模式：

```
$ export DIB_BLOCK_DEVICE_CONFIG='<yaml_schema_with_partitions>'
```

BIOS 示例

以下 YAML 结构展示了修改后的逻辑卷分区布局，该布局拥有充足的空间，可拉取 overcloud 容器镜

像：

```
export DIB_BLOCK_DEVICE_CONFIG='''
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: mbr
    partitions:
      - name: root
        flags: [ boot,primary ]
        size: 40G
- lvm:
    name: lvm
    base: [ root ]
    pvs:
        - name: pv
          base: root
          options: [ "--force" ]
    vgs:
        - name: vg
          base: [ "pv" ]
          options: [ "--force" ]
    lvs:
        - name: lv_root
          base: vg
          extents: 23%VG
        - name: lv_tmp
          base: vg
          extents: 4%VG
        - name: lv_var
          base: vg
          extents: 45%VG
        - name: lv_log
          base: vg
          extents: 23%VG
        - name: lv_audit
          base: vg
          extents: 4%VG
        - name: lv_home
          base: vg
          extents: 1%VG
- mkfs:
    name: fs_root
    base: lv_root
    type: xfs
    label: "img-rootfs"
    mount:
        mount_point: /
        fstab:
            options: "rw,relatime"
            fsck-passno: 1
- mkfs:
    name: fs_tmp
    base: lv_tmp
```

```
    type: xfs
    mount:
      mount_point: /tmp
      fstab:
        options: "rw,nosuid,nodev,noexec,relatime"
        fsck-passno: 2
- mkfs:
    name: fs_var
    base: lv_var
    type: xfs
    mount:
      mount_point: /var
      fstab:
        options: "rw,relatime"
        fsck-passno: 2
- mkfs:
    name: fs_log
    base: lv_log
    type: xfs
    mount:
      mount_point: /var/log
      fstab:
        options: "rw,relatime"
        fsck-passno: 3
- mkfs:
    name: fs_audit
    base: lv_audit
    type: xfs
    mount:
      mount_point: /var/log/audit
      fstab:
        options: "rw,relatime"
        fsck-passno: 4
- mkfs:
    name: fs_home
    base: lv_home
    type: xfs
    mount:
      mount_point: /home
      fstab:
        options: "rw,nodev,relatime"
        fsck-passno: 2
'''
```

使用此示例 YAML 内容作为镜像分区模式的基础。修改分区大小和布局以符合您的需求。

注意

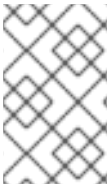
您必须为镜像定义正确的分区大小，因为部署后将无法调整其大小。

UEFI 示例

以下 YAML 结构展示了修改后的逻辑卷分区布局，该布局拥有充足的空间，可拉取 overcloud 容器镜像：

```
export DIB_BLOCK_DEVICE_CONFIG='''
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: gpt
    partitions:
      - name: ESP
        type: 'EF00'
        size: 200MiB
      - name: BSP
        type: 'EF02'
        size: 1MiB
      - name: ROOT
        type: '8300'
        size: 100%
- mkfs:
    name: fs_esp
    base: ESP
    type: vfat
    mount:
      mount_point: /boot/efi
      fstab:
        options: "defaults"
        fsck-passno: 1
- mkfs:
    name: fs_root
    label: "img-rootfs"
    base: ROOT
    type: xfs
    mount:
      mount_point: /
      fstab:
        options: "defaults"
        fsck-passno: 1
'''
```

使用此示例 YAML 内容作为镜像分区模式的基础。修改分区大小和布局以适合您的环境。



注意

在部署前，您必须为镜像定义正确的分区大小，因为部署后无法调整它们的大小。

24.8. 修改镜像大小

修改后的分区模式的空间总量可能会超出默认的磁盘大小 (20G)。在这种情况下，您可能需要修改镜像的大小。要修改镜像大小，可编辑创建镜像的配置文件。

步骤

1. 创建 /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml 的副本：

```
# cp /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml \
/home/stack/overcloud-hardened-images-python3-custom.yaml
```

注意

对于 UEFI 完整磁盘镜像，请使用 /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi-python3.yaml。

2. 编辑配置文件中的 DIB_IMAGE_SIZE 并根据需要调整值：

```
...

environment:
  DIB_PYTHON_VERSION: '3'
  DIB_MODPROBE_BLACKLIST: 'usb-storage cramfs freevxfs jffs2 hfs hfsplus squashfs udf vfat bluetooth'
  DIB_BOOTLOADER_DEFAULT_CMDLINE: 'nofb nomodeset vga=normal console=tty0 console=ttyS0,115200 audit=1 nousb'
  DIB_IMAGE_SIZE: '40' (1)
  COMPRESS_IMAGE: '1'
```

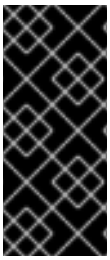
(1) 将该值调整为新的磁盘空间总量。

3. 可选。要配置代理，还必须包含 http_proxy 和 https:_proxy 环境变量：

```
environment:
  http_proxy: <proxy_server>
  https_proxy: <proxy_server>
```

- 将 <proxy_server > 替换为代理的地址。

4. 保存该文件。



重要

部署 overcloud 时，director 创建 overcloud 镜像的 RAW 版本。这意味着 undercloud 必须有足够的可用空间来容纳 RAW 镜像。例如，如果将安全强化型镜像大小设置为 40G，则 undercloud 的硬盘上必须有 40G 可用空间。

重要

当 director 将镜像写入物理磁盘时，它在磁盘末尾创建 64MB 配置驱动主分区。创建完整磁盘镜像时，请确保物理磁盘的大小适合此额外分区。

24.9. 构建完整磁盘镜像

设置环境变量和自定义镜像后，使用 **openstack overcloud image build** 命令创建镜像。

步骤

1. 使用所有必要配置文件运行 **openstack overcloud image build** 命令。

```
# openstack overcloud image build \
--image-name overcloud-hardened-full \ 1
--config-file /home/stack/overcloud-hardened-images-python3-custom.yaml \ 2
--config-file /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-rhel8.yaml 3
```

1 对于 UEFI 完整磁盘镜像，请使用 **overcloud-hardened-uefi-full**。

2 **overcloud-hardened-images-python3-custom.yaml** 文件是包含新磁盘大小的自定义配置文件。如果您不使用其他自定义磁盘大小，请使用原始 **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** 文件。对于标准的 UEFI 完整磁盘镜像，请使用 **overcloud-hardened-images-uefi-python3.yaml**。

3 对于 UEFI 完整磁盘镜像，请使用 **overcloud-hardened-images-uefi-rhel8.yaml**。

该命令会创建一个称为 **overcloud-hardened-full.qcow2** 的镜像，其中包含所有必需的安全功能。

24.10. 上传完整磁盘镜像

将镜像上传至 **OpenStack Image (glance)** 服务，并通过**Red Hat OpenStack Platform director** 开始使用该镜像。要上传安全强化型镜像，请完成以下步骤：

1. 重命名新生成的镜像并将镜像移至 **images** 目录：

```
# mv overcloud-hardened-full.qcow2 ~/images/overcloud-full.qcow2
```

2. 删除所有旧的 **overcloud** 镜像：

```
# openstack image delete overcloud-full
# openstack image delete overcloud-full-initrd
# openstack image delete overcloud-full-vmlinuz
```

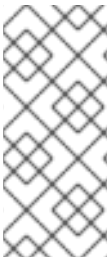
3. 上传新的 **overcloud** 镜像：

```
# openstack overcloud image upload --image-path /home/stack/images --whole-disk
```

如果您想将某个现有镜像替换成安全强化型镜像，请使用 **--update-existing** 标志。此标志使用新安全强化型镜像覆盖原始 **overcloud-full** 镜像。

第 25 章 配置直接部署

置备节点时，director 将 overcloud 基础操作系统镜像挂载到 iSCSI 挂载上，然后将镜像复制到每个节点的磁盘上。直接部署是一个替代的方法，可将 HTTP 位置的磁盘镜像直接写入裸机节点上的磁盘。



注意

对 iSCSI 部署接口 iscsi 的支持将在 Red Hat OpenStack Platform (RHOSP)版本 17.0 中弃用，并将在 RHOSP 18.0 中删除。直接部署（direct）是 RHOSP 17.0 的默认部署接口。

25.1. 在 UNDERCLOUD 上配置直接部署接口

iSCSI 部署接口是默认部署接口。但是，您可以启用直接部署接口，将 HTTP 位置的镜像下载到目标磁盘。

注意

对 iSCSI 部署接口的支持将在 Red Hat OpenStack Platform (RHOSP)版本 17.0 中已弃用，并将在 RHOSP 18.0 中删除。直接部署将是 RHOSP 17.0 的默认部署接口。

前提条件

- 您的 overcloud 节点内存 tmpfs 必须至少有 8GB RAM。

步骤

1. 创建或修改自定义环境文件 /home/stack/undercloud_custom_env.yaml 并指定 IronicDefaultDeployInterface。

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
```

2. 默认情况下，每个节点上的 Bare Metal 服务 (ironic) 代理都包含通过 HTTP 链接存储在 Object Storage 服务 (swift) 中的镜像。或者，ironic 可以通过 ironic-conductor HTTP 服务器将此镜像直接流传输到节点。要更改提供镜像的服务，请在 /home/stack/undercloud_custom_env.yaml 文件中将 IronicImageDownloadSource 设置为 http：

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
  IronicImageDownloadSource: http
```

3. **在 undercloud.conf 文件的 DEFAULT 部分包含自定义环境文件。**

```
custom_env_files = /home/stack/undercloud_custom_env.yaml
```

4. **执行 undercloud 安装：**

```
$ openstack undercloud install
```

第 26 章 创建虚拟化 CONTROL PLANES

一个虚拟化的 control plane 是位于虚拟机 (VM) 而非裸机上的 control plane。使用虚拟化 control plane 可减少 control plane 所需的裸机数。

本章介绍了如何使用 RHOSP 和 Red Hat Virtualization 为 overcloud 虚拟化 Red Hat OpenStack Platform (RHOSP) control plane。

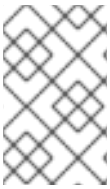
26.1. 虚拟化 CONTROL PLANES 架构

借助 director 使用 Red Hat Virtualization 集群中部署的 Controller 节点置备 overcloud。然后您可以将这些虚拟化控制器部署为虚拟化 control plane 节点。

注意

仅在 Red Hat Virtualization 上支持虚拟化控制器节点。

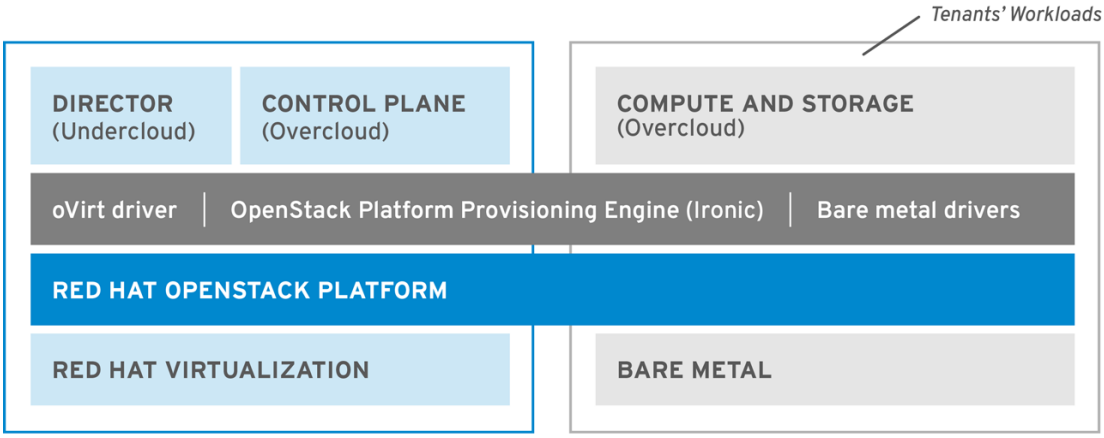
以下构架图演示了如何部署虚拟化 control plane。使用 Red Hat Virtualization 虚拟机上运行的 Controller 节点分发 overcloud，并在裸机上运行 Compute 和 Storage 节点。



注意

在 Red Hat Virtualization 上运行 OpenStack 虚拟化 undercloud。

虚拟化 control planes 架构



OpenStack Bare Metal Provisioning 服务 (ironic) 包括 Red Hat Virtualization 虚拟机 staging-ovirt 的驱动程序。您可以使用此驱动程序在 Red Hat Virtualization 环境中管理虚拟节点。您也可以使用此驱动程序在 Red Hat Virtualization 环境内将 overcloud 控制器部署为虚拟机。

虚拟化 RHOSP overcloud control plane 的优点和限制

虽然虚拟化 RHOSP overcloud control plane 有很多优点，但它并不适用于所有配置。

优点

虚拟化 overcloud control plane 有很多优点，可防止停机并提高性能。

- *您可以使用热添加和热删除以按需要扩展 CPU 和内存，将资源动态分配给虚拟化控制器，从而减少停机时间，并有助于随平台扩展而增加容量。*
- *您可以在同一 Red Hat Virtualization 集群中部署额外的基础架构虚拟机。这可最大程度减少数据中心中的服务器空间，并最大程度提高物理节点的效率。*
- *您可以使用可组合角色来定义更复杂的 RHOSP control plane，并将资源分配给 control plane 的特定组件。*
- *使用 VM 实时迁移功能可在不中断服务的情况下维护系统。*
- *您可以集成 Red Hat Virtualization 支持的第三方工具或自定义工具。*

限制

虚拟化 control plane 对可使用的配置类型有限制。

- 不支持虚拟化 Ceph Storage 节点和 Compute 节点。
- 使用光纤通道的后端不支持 Block Storage (cinder) 镜像到卷。Red Hat Virtualization 不支持 N_Port ID Virtualization (NPIV) 。因此，需要将 LUN 从存储后端映射到控制器（cinder-volume 默认在此运行）的块存储 (cinder) 驱动程序无法工作。您必须为 cinder-volume 创建专用角色，并使用该角色创建物理节点，而不是将其包含在虚拟化控制器中。有关更多信息，请参阅可组合服务和自定义角色。

26.2. 使用 RED HAT VIRTUALIZATION 驱动程序置备虚拟化控制器

完成以下步骤，以使用 RHOSP 和 Red Hat Virtualization 为 overcloud 置备虚拟化 RHOSP control plane。

先决条件

- 您必须具有支持 Intel 64 或 AMD64 CPU 扩展的 64 位 x86 处理器。
- 您必须已安装并配置了以下软件：
  - Red Hat Virtualization。如需更多信息，请参阅 Red Hat Virtualization 文档套件。
  - Red Hat OpenStack Platform (RHOSP)。有关更多信息，请参阅 Director 安装和使用。
- 您必须事先准备虚拟化 Controller 节点。这些要求与裸机 Controller 节点相同。有关更多信息，请参阅 Controller 节点要求。
- 您必须事先准备用作 overcloud Compute 节点和存储节点的裸机节点。有关硬件规范，请参阅 Compute 节点要求和 Ceph Storage 节点要求。要在 POWER (ppc64le) 硬件上部署 overcloud Compute 节点，请参阅针对 POWER 的 Red Hat OpenStack Platform。
-

您必须已创建了逻辑网络，并且主机网络的集群可对多个网络使用网络隔离。如需更多信息，请参阅逻辑网络。

- 您必须将每个节点的内部 BIOS 时钟设置为 UTC，以防止 hwclock 在应用时区偏移前同步 BIOS 时钟时，可能会导致将来的文件时间戳出现问题。

提示

要避免性能瓶颈，请使用可组合角色并在裸机 Controller 节点上保留 data plane 服务。

步骤

1. 要在 director 中启用 staging-ovirt 驱动程序，可将该驱动程序添加到 undercloud.conf 配置文件的 enabled_hardware_types 参数中：

```
enabled_hardware_types = ipmi,redfish,ilo,idrac,staging-ovirt
```

2. 验证 undercloud 是否包含 staging-ovirt 驱动程序：

```
(undercloud) [stack@undercloud ~]$ openstack baremetal driver list
```

如果您正确配置了 undercloud，则命令会返回以下结果：

```
+---------------------+-----------------------+
| Supported driver(s) | Active host(s)        |
+---------------------+-----------------------+
| idrac               | localhost.localdomain |
| ilo                 | localhost.localdomain |
| ipmi                | localhost.localdomain |
| pxe_drac            | localhost.localdomain |
| pxe_ilo             | localhost.localdomain |
| pxe_ipmitool        | localhost.localdomain |
| redfish             | localhost.localdomain |
| staging-ovirt       | localhost.localdomain |
```

3. 更新 overcloud 节点定义模板，例如 nodes.json，以将 Red Hat Virtualization 上托管的虚拟机注册到 director。有关更多信息，请参阅为 Overcloud 注册节点。使用以下键-值对定义要使用 overcloud 部署的虚拟机的各个方面：

表 26.1. 为 overcloud 配置虚拟机

键	设置为该值
pm_type	oVirt/RHV 虚拟机的 OpenStack Bare Metal Provisioning (ironic) 服务驱动程序 **staging-ovirt**。
pm_user	Red Hat Virtualization Manager 用户名。
pm_password	Red Hat Virtualization Manager 密码。
pm_addr	Red Hat Virtualization Manager 服务器的主机名或 IP。
pm_vm_name	在其中创建控制器的 Red Hat Virtualization Manager 中的虚拟机的名称。

例如：

```
{
    "nodes": [
        {
            "name":"osp13-controller-0",
            "pm_type":"staging-ovirt",
            "mac":[
                "00:1a:4a:16:01:56"
            ],
            "cpu":"2",
            "memory":"4096",
            "disk":"40",
            "arch":"x86_64",
            "pm_user":"admin@internal",
            "pm_password":"password",
            "pm_addr":"rhvm.example.com",
            "pm_vm_name":"{osp_curr_ver}-controller-0",
            "capabilities": "profile:control,boot_option:local"
        },
        ...
}
```

在每个 Red Hat Virtualization 主机上配置一个 Controller

4. ***在 Red Hat Virtualization 中使用“soft negative affinity”配置关联性组，以确保为您的控制器虚拟机实施高可用性。如需更多信息，请参阅关联性组。***

5. ***打开 Red Hat Virtualization Manager 界面，使用它将每个 VLAN 映射到控制器虚拟机中的***

单独逻辑 vNIC。如需更多信息，请参阅逻辑网络。

6. 设置 director 和控制器虚拟机的 vNIC 中的 no_filter，并重启虚拟机，可禁用附加到控制器虚拟机的网络上的 MAC 欺骗过滤器。如需更多信息，请参阅虚拟网络接口卡。

7. 部署 overcloud 以在您的环境中包括新的虚拟化控制器节点：

```
(undercloud) [stack@undercloud ~]$ openstack overcloud deploy --templates
```

第 27 章 执行高级容器镜像管理

默认容器镜像配置适合大多数环境。在某些情况下，您的容器镜像配置可能需要一些自定义，如版本固定。

27.1. 为 UNDERCLOUD 固定容器镜像

在某些情况下，您可能需要 undercloud 的一组特定容器镜像版本。在这种情况下，您必须将镜像固定到特定的版本。要固定镜像，您必须生成和修改容器配置文件，然后将 undercloud 角色数据和容器配置文件结合，以生成包含服务到容器镜像映射的环境文件。在 undercloud.conf 文件的 custom_env_files 参数中包含此环境文件。

步骤

1. 以 stack 用户身份登录 undercloud 主机。

2. 使用 --output-env-file 选项运行 openstack tripleo container image prepare default 命令，生成包含默认镜像配置的文件：

```
$ sudo openstack tripleo container image prepare default \
--output-env-file undercloud-container-image-prepare.yaml
```

3. 根据您的环境要求，修改 undercloud-container-image-prepare.yaml 文件。

   a. 移除 tag: 参数，以便 director 可以使用 tag_from_label: 参数。director 使用此参数来标识每个容器镜像的最新版本，拉取每个镜像，并在 director 中的容器 registry 上标记每个镜像。

   b. 移除 undercloud 的 Ceph 标签。

   c. 确保 neutron_driver: 参数为空。不要将此参数设置为 OVN，因为 undercloud 不支持 OVN。

   d. 包含容器镜像 registry 凭据：

```
ContainerImageRegistryCredentials:
  registry.redhat.io
    myser: 'p@55w0rd!'
```

注意

您无法将容器镜像推送到新 undercloud 上的 undercloud registry，因为 image-serve registry 尚未安装。您必须将 push_destination 值设置为 false，或使用自定义值直接从源拉取镜像。有关更多信息，请参阅容器镜像准备参数。

4. 生成新的容器镜像配置文件，该文件结合使用 undercloud 角色文件和自定义 undercloud-container-image-prepare.yaml 文件：

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

undercloud-container-images.yaml 文件是一个环境文件，包含服务参数到容器镜像的映射。例如，OpenStack Identity (keystone) 使用 ContainerKeystoneImage 参数来定义其容器镜像：

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

请注意，容器镜像标签与 {version}-{release} 格式匹配。

5. 将 undercloud-container-images.yaml 文件包含在 undercloud.conf 文件的 custom_env_files 参数中。在运行 undercloud 安装时，undercloud 服务使用来自此文件的固定容器镜像映射。

27.2. 为 OVERCLOUD 固定容器镜像

在某些情况下，您可能需要 overcloud 的一组特定容器镜像版本。在这种情况下，您必须将镜像固定到特定的版本。若要固定镜像，您必须创建 containers-prepare-parameter.yaml 文件，使用此文件将容器镜像拉取到 undercloud registry，并生成包含固定镜像列表的环境文件。

例如，containers-prepare-parameter.yaml 文件可能包含以下内容：

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel8
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
```

ContainerImagePrepare 参数包含单个规则 set。此规则 set 不得包含 tag 参数，且必须依赖 tag_from_label 参数来标识每个容器镜像的最新版本和发行版本。director 使用此规则 set 来标识每个容器镜像的最新版本，拉取每个镜像，并在 director 中的容器 registry 上标记每个镜像。

步骤

1. **运行 openstack tripleo container image prepare 命令，该命令从 containers-prepare-parameter.yaml 文件中定义的源中拉取所有镜像。包含 --output-env-file 以指定将包含固定容器镜像列表的输出文件：**

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

overcloud-images.yaml 文件是一个环境文件，包含服务参数到容器镜像映射。例如，OpenStack Identity (keystone) 使用 ContainerKeystoneImage 参数来定义其容器镜像：

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

请注意，容器镜像标签与 {version}-{release} 格式匹配。

2. **在运行 openstack overcloud deploy 命令时，以特定顺序将 containers-prepare-parameter.yaml 和 overcloud-images.yaml 文件包含在环境文件集合中：**

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/containers-prepare-parameter.yaml \
    -e /home/stack/overcloud-images.yaml \
    ...
```

overcloud 服务使用 overcloud-images.yaml 文件中列出的固定镜像。

第 28 章 DIRECTOR 错误故障排除

在 director 过程的某些阶段可能发生错误。本节包含一些有关诊断常见问题的信息。

28.1. 节点注册故障排除

由于节点详细信不正确的问题而通常导致节点注册问题。在这些情况下，请验证包含节点详细信息的模板文件并更正导入的节点详细信息。

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 使用 --validate-only 选项运行节点导入命令。此选项无需执行导入即可验证您的节点模板：

```
(undercloud) $ openstack overcloud node import --validate-only ~/nodes.json
Waiting for messages on queue 'tripleo' with no timeout.

Successfully validated environment file
```

3. 要修复导入节点的错误详细信息，请运行 openstack baremetal 命令以更新节点详细信息。下例显示如何更改网络详细信息：

   a. 识别导入节点的已分配端口 UUID：

```
$ source ~/stackrc
(undercloud) $ openstack baremetal port list --node [NODE UUID]
```

   b. 更新 MAC 地址：

```
(undercloud) $ openstack baremetal port set --address=[NEW MAC] [PORT UUID]
```

   c. 在节点上配置新的 IPMI 地址：

```
(undercloud) $ openstack baremetal node set --driver-info ipmi_address=[NEW IPMI ADDRESS] [NODE UUID]
```

28.2. 硬件内省故障排除

您必须运行完内省进程。但是，如果检查 ramdisk 没有响应，则 ironic-inspector 将在默认一小时后超时。有时这表示检查 ramdisk 中有错误，但这种超时通常是由于环境错误配置所致，特别是 BIOS 引导设置。

要诊断并解决常见环境错误配置问题，请完成以下步骤：

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. director 使用 OpenStack Object Storage（swift）保存在内省过程中获得的硬件数据。如果这个服务没有运行，内省将失败。检查并确定所有与 OpenStack Object Storage 相关的服务都在运行：

```
(undercloud) $ sudo systemctl list-units tripleo_swift*
```

3. 确保您的节点处于 manageable 状态。内省不检查处于 available 状态的节点，该状态意味着用于部署。如果要检查处于 available 状态的节点，请在内省前将节点状态更改为 manageable 状态：

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

4. 配置对内省 ramdisk 的临时访问权限。可提供临时密码或 SSH 密钥以在内省调试期间访问节点。完成以下步骤以配置 ramdisk 访问权限：

   a. 使用临时密码运行 openssl passwd -1 命令以生成 MD5 hash：

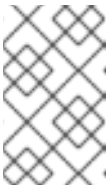
```
(undercloud) $ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

   b. 编辑 /var/lib/ironic/httpboot/inspector.ipxe 文件，找到以 kernel 开头的行，并附加

rootpwd 参数和 MD5 hash：

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-
hardware,logs systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1
ipa-inspection-benchmarks=cpu,mem,disk
rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
```

或者，将公共 SSH 密钥附加到 sshkey 参数。



注意

rootpwd 和 sshkey 参数都需要包括引号。

5. **在节点上运行内省：**

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

内省完成后，使用 --provide 选项将节点状态更改为 available。

6. **从 dnsmasq 日志中识别节点的 IP 地址：**

```
(undercloud) $ sudo tail -f /var/log/containers/ironic-inspector/dnsmasq.log
```

7. **如果出错，则使用根用户和临时访问详细信息访问节点：**

```
$ ssh root@192.168.24.105
```

在内省期间访问节点以运行诊断命令并排除内省故障。

8. **要停止内省过程，请运行以下命令：**

```
(undercloud) $ openstack baremetal introspection abort [NODE UUID]
```

您也可以等待操作过程超时。

注意

Red Hat OpenStack Platform director 在初始中止后重试内省三次。在每次尝试时均运行 openstack baremetal introspection abort 命令以完全中止内省。

28.3. 工作流和执行故障排除

OpenStack Workflow（mistral）服务将多项 OpenStack 任务组合成工作流。Red Hat OpenStack Platform 使用一组这样的工作流在 director 上执行常见功能，包括裸机节点控制、验证、计划管理和 overcloud 部署。

例如，在运行 openstack overcloud deploy 命令时，OpenStack Workflow 服务执行两个工作流。第一个工作流上传部署计划：

```
Removing the current plan files
Uploading new plan files
Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-073744ed5e6b
Plan updated
```

第二个工作流启动 overcloud 部署：

```
Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-heat-templates
Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-1380631c764d
2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS  Stack CREATE started
2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS  state changed
2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS  state
changed
2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS  state changed
...
```

OpenStack Workflow 服务使用以下对象跟踪工作流：

Actions

相关的任务运行时，OpenStack 会执行特定的指令。例如，运行 shell 脚本或执行 HTTP 请求。一些 OpenStack 组件内置有可供 OpenStack Workflow 使用的操作。

任务

定义要运行的操作以及运行该操作的结果。这些任务通常关联有操作或其他工作流。完成一项任务时，工作流会定向到另一任务，这通常取决于前一任务的成败状况。

工作流

分组在一起并以特定顺序执行的一组任务。

执行

定义特定操作、任务或工作流的运行。

OpenStack Workflow 可以提供可靠的执行日志，这有助于识别某些命令失败的问题。例如，如果某一工作流执行失败，您可以确定其故障点。

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 列出具有已失败状态 ERROR 的工作流执行记录：

```
(undercloud) $ openstack workflow execution list | grep "ERROR"
```

3. 获取失败工作流执行的 UUID（例如 dffa96b0-f679-4cd2-a490-4769a3825262）并查看该执行及输出：

```
(undercloud) $ openstack workflow execution show dffa96b0-f679-4cd2-a490-4769a3825262
(undercloud) $ openstack workflow execution output show dffa96b0-f679-4cd2-a490-
4769a3825262
```

4. 这些命令返回有关执行中已失败任务的信息。openstack workflow execution show 命令还会显示用于执行的工作流（例如 tripleo.plan_management.v1.publish_ui_logs_to_swift）。您可以使用以下命令查看完整的工作流定义：

```
(undercloud) $ openstack workflow definition show
tripleo.plan_management.v1.publish_ui_logs_to_swift
```

这可用于辨别特定任务在工作流中的位置。

5. 使用类似命令语法查看操作执行及其结果：

```
(undercloud) $ openstack action execution list
```

```
(undercloud) $ openstack action execution show 8a68eba3-0fec-4b2a-adc9-5561b007e886
(undercloud) $ openstack action execution output show 8a68eba3-0fec-4b2a-adc9-
5561b007e886
```

这对于识别导致问题的具体操作非常有用。

28.4. OVERCLOUD 创建和部署故障排除

使用 OpenStack Orchestration (heat) 服务对 overcloud 进行初始创建。如果 overcloud 部署失败，则使用 OpenStack 客户端和服务日志文件诊断失败的部署。

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 运行部署失败的命令：

```
$ openstack overcloud failures
```

3. 运行以下命令以显示失败的详细信息：

```
(undercloud) $ openstack stack failures list <OVERCLOUD_NAME> --long
```

- 用您的 overcloud 的名称替换 <OVERCLOUD_NAME>。

4. 运行以下命令以识别失败的堆栈：

```
(undercloud) $ openstack stack list --nested --property status=FAILED
```

28.5. 节点置备故障排除

OpenStack Orchestration (heat) 服务控制置备过程。如果节点置备失败，则使用 OpenStack 客户端和服务日志文件诊断问题。

步骤

1. **Source stackrc 文件：**

```
$ source ~/stackrc
```

2. **检查裸机恢复服务以查看所有注册节点及其当前状态：**

```
(undercloud) $ openstack baremetal node list

+----------+------+--------------+-------------+-----------------+-------------+
| UUID     | Name | Instance UUID | Power State | Provision State | Maintenance |
+----------+------+--------------+-------------+-----------------+-------------+
| f1e261...| None | None          | power off   | available       | False       |
| f0b8c1...| None | None          | power off   | available       | False       |
+----------+------+--------------+-------------+-----------------+-------------+
```

可用于置备的所有节点都应设置以下状态：

- **Maintenance 设置为 False。**
- **在置备前，Provision State 设置为 available。**

3. **如果节点没有设置为 False 或 Provision State 设置为 available，则使用下表来识别问题和解决方案：**

问题	原因	解决方案
Maintenance 自动将自身设置为 **True**。	director 无法访问节点的电源管理。	检查节点电源管理的凭据。
Provision State 设置为 **available**，但节点未置备。	此问题在启动裸机部署前发生。	检查包括配置集和类别映射的节点详细信息。检查节点硬件详细信息是否在该类别的要求内。
节点的 Provision State 设置为 **wait call-back**。	此节点的节点置备过程尚未完成。	等到此状态更改。否则，连接到节点的虚拟控制台并检查输出。

问题	原因	解决方案
Provision State 处于 **active**，Power State 处于 **power on**，但节点无响应。	节点置备已成功完成，并在部署后配置步骤中出问题。	诊断节点配置过程。连接到节点的虚拟控制台并检查输出。
Provision State 为 **error** 或 **deploy failed**。	节点置备已失败。	使用 **openstack baremetal node show** 命令查看裸机节点详细信息，并检查 **last_error** 字段，其中包含错误说明。

其他资源

- ***裸机节点置备状态***

28.6. 置备期间 IP 地址冲突故障排除

如果目标主机分配的 IP 地址已经使用，则内省和部署任务将失败。要防止这些失败，可对 Provisioning 网络执行端口扫描，以确定发现 IP 范围和主机 IP 范围是否可用。

步骤

1. ***安装 nmap:***

```
$ sudo dnf install nmap
```

2. ***使用 nmap 命令扫描 IP 地址范围中的活动地址。这个示例会扫描 192.168.24.0/24 这个范围，使用 Provisioning 网络的 IP 子网值（使用 CIDR 位掩码符号）替换它：***

```
$ sudo nmap -sn 192.168.24.0/24
```

3. ***复查 nmap 扫描的输出。例如，您应查看 undercloud 的 IP 地址，以及子网上存在的任何其他主机：***

```
$ sudo nmap -sn 192.168.24.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.168.24.1
Host is up (0.00057s latency).
```

```
Nmap scan report for 192.168.24.2
Host is up (0.00048s latency).
Nmap scan report for 192.168.24.3
Host is up (0.00045s latency).
Nmap scan report for 192.168.24.5
Host is up (0.00040s latency).
Nmap scan report for 192.168.24.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

如果这些活跃的 IP 地址和 undercloud.conf 中指定的 IP 地址范围有冲突，则必须在内省或部署overcloud 节点前修改 IP 地址范围或释放一些 IP 地址。

28.7. “NO VALID HOST FOUND”错误故障排除

在一些情况下，/var/log/nova/nova-conductor.log 包括了以下错误：

```
NoValidHost: No valid host was found. There are not enough hosts available.
```

当计算调度程序找不到适合启动新实例的裸机节点时，会发生此错误。这通常意味着，Compute 服务期望找到的资源与裸机恢复服务向 Compute 建议的资源之间存在不匹配。要检查是否存在不匹配的错误，请完成以下步骤：

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 检查是否在节点上成功执行内省。如果内省失败，则检查每个节点是否包含所需的 ironic 节点属性：

```
(undercloud) $ openstack baremetal node show [NODE UUID]
```

检查 properties JSON 项中的 cpus、cpu_arch、memory_mb 和 local_gb 都有有效的值。

3. 确保映射到节点的 Compute 类型不超过所需节点数的节点属性：

```
(undercloud) $ openstack flavor show [FLAVOR NAME]
```

4. 运行 **openstack baremetal node list** 命令以确保有足够的节点处于可用状态。处于 **manageable** 状态的节点通常表示内省失败。

5. 运行 **openstack baremetal node list** 命令并确保节点不处于维护模式。如果节点自动更改为维护模式，则可能的原因是电源管理凭据不正确。检查电源管理凭据，然后移除维护模式：

```
(undercloud) $ openstack baremetal node maintenance unset [NODE UUID]
```

6. 如果您使用的是自动配置集标记，则检查是否有足够的节点对应于每个类别和配置集。在节点上运行 **openstack baremetal node show** 命令并检查 **properties** 字段中的 **capabilities** 密钥。例如，标记为 **Compute** 角色的节点包含 **profile:compute** 值这样的信息。

7. 内省后，必须等待节点信息从 **Bare Metal** 传播到 **Compute**。但是，如果您手工进行了一些操作，节点可能会短时间内对 **Compute service (nova)** 不可用。使用以下命令检查系统中的总体资源：

```
(undercloud) $ openstack hypervisor stats show
```

28.8. OVERCLOUD 配置故障排除

Red Hat OpenStack Platform director 使用 **Ansible** 配置 **overcloud**。完成以下步骤，对 **overcloud** 上的 **Ansible playbook** 错误 **(config-download)** 进行诊断。

步骤

1. 确保 **stack** 用户能够访问 **undercloud** 的 **/var/lib/mistral** 目录中的文件。

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
```

此命令保留 **mistral** 用户对该目录的访问权限。

2. 切换到 **config-download** 文件的工作目录。该目录通常是 **/var/lib/mistral/overcloud/**。

```
$ cd /var/lib/mistral/overcloud/
```

3. 搜索 **ansible.log** 文件以查找故障点。

```
$ less ansible.log
```

记录失败的步骤。

4. 在工作目录中查找 config-download playbook 中失败的步骤，以确定发生的操作。

28.9. 容器配置故障排除

Red Hat OpenStack Platform director 使用 paunch 启动容器，使用 podman 管理容器，以及使用 puppet 创建容器配置。此步骤显示如何在出错时对容器进行诊断。

访问主机

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 获取包含容器故障的节点的 IP 地址。

```
(undercloud) $ openstack server list
```

3. 登录该节点：

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. 切换到 root 用户：

```
$ sudo -i
```

识别故障容器

1. 查看所有容器：

```
$ podman ps --all
```

识别故障容器。故障容器通常以非零的状态推出。

检查容器日志

1. 每个容器都会保留其主进程的标准输出内容。使用此输出内容作为日志，帮助确定容器运行过程中实际上发生了什么。例如，要查看 **keystone** 容器的日志，请运行以下命令：

```
$ sudo podman logs keystone
```

在大多数情况下，此日志包含有关容器故障原因的信息。

2. 主机还包含已失败服务的 **stdout** 日志。可在 **/var/log/containers/stdouts/** 中查找 **stdout** 日志。例如，要查看出现故障的 **keystone** 容器的日志，请运行以下命令：

```
$ cat /var/log/containers/stdouts/keystone.log
```

检查容器

在某些情况下，您可能需要验证容器的相关信息。例如，请使用以下命令来查看 **keystone** 容器的相关数据：

```
$ sudo podman inspect keystone
```

此命令返回一个包含低级配置数据的 **JSON** 对象。您可以通过管道将这些输出内容传递给 **jq** 命令，以对特定数据进行解析。例如，要查看 **keystone** 容器的加载情况，请运行以下命令：

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

您还可以使用 **--format** 选项将数据解析到一行中，这在针对一组容器数据运行命令时非常有用。例如，要重建用于运行 **keystone** 容器的选项，请使用包含 **--format** 选项的以下 **inspect** 命令：

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}:{{ join .Options "," }}{{end}} -ti {{.Config.Image}}' keystone
```

注意

--format 选项会按照 **Go** 语法来创建查询。

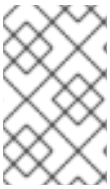
将这些选项和 podman run 命令一起使用以重新创建容器用于故障排除目的：

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

在容器内运行命令

在某些情况下，您可能需要通过特定的 Bash 命令从容器中获取信息。在此情况下，使用以下 podman 命令以在运行中容器内执行命令。例如，运行 podman exec 命令以在 keystone 容器内运行命令：

```
$ sudo podman exec -ti keystone <COMMAND>
```



注意

-ti 选项会通过交互式伪终端来运行命令。

- 用您要运行的命令替换 <COMMAND>。例如，每个容器都有一个健康检查脚本，用于验证服务的连接状况。您可以使用以下命令为 keystone 运行这个健康检查脚本：

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

要访问容器的 shell，请运行 podman exec，并将 /bin/bash 用作您要在容器内运行的命令：

```
$ sudo podman exec -ti keystone /bin/bash
```

查看容器文件系统

1. 要查看故障容器的文件系统，请运行 podman mount 命令。例如，要查看出现故障的 keystone 容器的文件系统，请运行以下命令：

```
$ podman mount keystone
```

这会提供一个挂载位置，用于查看文件系统内容：

```
/var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d7358270e711c610/merged
```

这对于查看容器内的 Puppet 报告很有用。您可以在容器挂载内的 var/lib/puppet/ 目录中查找这些报告。

导出容器

当容器出现故障时，您可能需要调查文件中包含的所有内容。在这种情况下，您可以将容器的整个文件系统导出为 tar 归档。例如，要导出 keystone 容器的文件系统，请运行以下命令：

```
$ sudo podman export keystone -o keystone.tar
```

这个命令会创建 keystone.tar 归档，以供您提取和研究。

28.10. COMPUTE 节点故障排除

Compute 节点使用 Compute 服务来执行基于虚拟机监控程序的操作。这意味着，对 Compute 节点进行故障排除可以解决与这个服务相关的问题。

步骤

1. Source stackrc 文件：

```
$ source ~/stackrc
```

2. 获取包含故障的 Compute 节点的 IP 地址：

```
(undercloud) $ openstack server list
```

3. 登录该节点：

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. 切换到 root 用户：

```
$ sudo -i
```

5. 查看容器状态：

```
$ sudo podman ps -f name=nova_compute
```

6. Compute 节点的主日志文件为 /var/log/containers/nova/nova-compute.log。如果 Compute 节点通信出现问题，请使用此文件开始诊断。

7. 如果需要在 Compute 节点上进行维护工作，把主机上存在的实例迁移到另外一个可以正常工作的 Compute 节点上，然后禁用需要进行维护的节点。

28.11. 创建 SOSREPORT

如果您需要联系红帽以获得 Red Hat OpenStack 平台的产品支持，可能需要生成一份 sosreport。有关创建 sosreport 的更多信息，请参阅：

- "How to collect all required logs for Red Hat Support to investigate an OpenStack issue"

28.12. 日志位置

在故障排除时，使用以下日志手机 undercloud 和 overcloud 的信息。

表 28.1. undercloud 和 overcloud 节点上的日志

信息	日志位置
容器化服务日志	**/var/log/containers/**
容器化服务中的标准输出	**/var/log/containers/stdouts**
Ansible 配置日志	**/var/lib/mistral/overcloud/ansible.log**

表 28.2. undercloud 节点上的其他日志

信息	日志位置
openstack overcloud deploy 的命令历史记录	**/home/stack/.tripleo/history**
Undercloud 安装日志	**/home/stack/install-undercloud.log**

表 28.3. overcloud 节点上的其他日志

信息	日志位置
Cloud-Init 日志	**/var/log/cloud-init.log**
高可用性日志	**/var/log/pacemaker.log**

第 29 章 UNDERCLOUD 和 OVERCLOUD 服务的提示

本节提供有关在 undercloud 上调整和管理特定 OpenStack 服务的建议。

29.1. 调优部署性能

Red Hat OpenStack Platform director 使用 OpenStack Orchestration (heat) 开展主要部署和置备功能。Heat 使用一系列 worker 执行部署任务。为计算默认 worker 数，director 的 heat 配置将 undercloud 的总 CPU 线程计数减半。在本实例中，线程数是指 CPU 内核数乘以超线程值。例如，如果 undercloud 的 CPU 具有 16 个线程，则 heat 默认生成 8 个 worker。director 配置会默认使用最小和最大值：

服务	最小值	最大值
OpenStack Orchestration (heat)	4	24

但是，您可以使用环境文件中的 HeatWorkers 参数手动设置 worker 数：

heat-workers.yaml

```
parameter_defaults:
  HeatWorkers: 16
```

undercloud.conf

```
custom_env_files: heat-workers.yaml
```

29.2. 在容器中运行 SWIFT-RING-BUILDER

要管理您的 Object Storage (swift) 环，请使用服务器容器内的 swift-ring-builder 命令：

- *swift_object_server*
- *swift_container_server*
- *swift_account_server*

例如，要查看有关 swift 对象环的信息，请运行以下命令：

```
$ sudo podman exec -ti -u swift swift_object_server swift-ring-builder /etc/swift/object.builder
```

可同时在 undercloud 和 overcloud 节点上运行此命令。

29.3. 更改 HAPROXY 的 SSL/TLS 密码规则

如果在 undercloud 中启用了 SSL/TLS （请参阅 第 4.2 节 "Director 配置参数"），您可能需要加强与 HAProxy 配置一起使用的 SSL/TLS 密码和规则。这种加强有助于避免 SSL/TLS 漏洞，如 POODLE 漏洞。

使用 hieradata_override undercloud 配置选项，设置下列 hieradata：

tripleo::haproxy::ssl_cipher_suite

HAProxy 中使用的密码套件。

tripleo::haproxy::ssl_options

HAProxy 中使用的 SSL/TLS 规则。

例如，您可能需要使用下列密码和规则：

- *Cipher: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-*

ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS

- *规则：**no-sslv3 no-tls-tickets***

*创建一个包含以下内容的 **hieradata 覆盖文件 (haproxy-hiera-overrides.yaml)：***

```
tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

注意

cipher 集合是一个连续行。

*设置 **undercloud.conf 文件中的 hieradata_override 参数，以便使用在运行 openstack undercloud install 之前创建的 hieradata 覆盖文件：***

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```

第 30 章 电源管理驱动

虽然 IPMI 是 director 用来进行电源管理的主要方法，但是 director 也支持其它电源管理类型。此附录包含 director 支持的电源管理功能列表。为 overcloud 注册节点时使用这些电源管理设置。有关更多信息，请参阅为 overcloud 注册节点。

30.1. 智能平台管理接口 (IPMI)

使用基板管理控制器 (BMC) 时的标准电源管理方法。

pm_type

将这个选项设置为 ipmi。

pm_user; pm_password

IPMI 的用户名和密码。

pm_addr

IPMI 控制器的 IP 地址。

pm_port（可选）

连接到 IPMI 控制器的端口。

30.2. REDFISH

由分布式管理任务组 (DMTF) 开发的，IT 基础架构的标准 RESTful API

pm_type

将这个选项设置为 redfish。

pm_user; pm_password

Redfish 的用户名和密码。

pm_addr

Redfish 控制器的 IP 地址。

pm_system_id

系统资源的规范路径（canonical path）。此路径必须包含根服务、版本以及系统的路径/唯一 ID。例如：/redfish/v1/Systems/CX34R87.

redfish_verify_ca

如果基板管理控制器 (BMC) 中的 Redfish 服务没有配置为使用由认可证书颁发机构 (CA) 签名的有效 TLS 证书，ironic 中的 Redfish 客户端无法连接到 BMC。将 redfish_verify_ca 选项设置为 false 来静默错误。但请注意：禁用 BMC 身份验证会破坏您的 BMC 的访问安全性。

30.3. DELL REMOTE ACCESS CONTROLLER (DRAC)

DRAC 是一个提供远程电源功能的接口，这些功能包括电源管理和服务器监控。

pm_type

将这个选项设置为 idrac。

pm_user; pm_password

DRAC 的用户名和密码。

pm_addr

DRAC 主机的 IP 地址。

30.4. INTEGRATED LIGHTS-OUT (ILO)

iLO 是惠普提供的一个远程电源功能的接口，这些功能包括电源管理和服务器监控。

pm_type

将这个选项设置为 ilo。

pm_user; pm_password

iLO 的用户名和密码。

pm_addr

iLO 接口的 IP 地址。

- 要启用这个驱动程序，请将 ilo 添加到 undercloud.conf 的 enabled_hardware_types

选项中，然后重新运行 openstack undercloud install。

- HP 节点的 ILO 固件版本至少为 1.85（2015 年 5 月 13 日）才能成功内省。director 已成功测试了使用此 ILO 固件版本的节点。

- 不支持使用共享 iLO 端口。

30.5. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu iRMC 是一个 BMC（Baseboard Management Controller），它集成了 LAN 连接以及扩展的功能。这个驱动提供了对连接到 iRMC 上的裸机系统的电源管理功能。

重要

需要 iRMC S4 或更高版本。

pm_type

将这个选项设置为 irmc。

pm_user; pm_password

iRMC 接口的用户名和密码。

重要

iRMC 用户必须具有 ADMINISTRATOR 权限。

pm_addr

iRMC 接口的 IP 地址。

pm_port（可选）

iRMC 操作的端口。默认值是 443。

pm_auth_method（可选）

iRMC 操作的验证方法。使用 basic 或 digest。默认为 basic。

pm_client_timeout（可选）

iRMC 操作的超时（以秒为单位）。默认值是 60 秒。

pm_sensor_method（可选）

获得感应器数据的方法。使用 ipmitool 或 scci。默认值是 ipmitool。

- *要启用此驱动程序，请将 irmc 添加到 undercloud.conf 文件的 enabled_hardware_types 选项中，并重新运行 openstack undercloud install 命令。*

iRMC 使用 UEFI 引导模式

iRMC 需要 ipxe.efi 来使用 UEFI 模式引导。要使用 UEFI 引导，请禁用使用 SNP (Simple Network Protocol) iPXE EFI 的默认行为，并重新安装 undercloud。

步骤

1. *创建自定义环境文件，例如 /home/stack/templates/irmc-uefi-boot.yaml。*

2. *在自定义环境文件中添加以下配置：*

```
parameter_defaults:
  IronicIPXEUefiSnpOnly: false
```

3. *编辑 undercloud.conf 文件中的 custom_env_files 参数，以添加您的自定义环境文件：*

```
custom_env_files = /home/stack/templates/irmc-uefi-boot.yaml
```

注意

您可以使用以逗号分隔的列表指定多个环境文件。

4. *重新安装 undercloud 以应用您的配置更新：*

```
$ OpenStack undercloud install
```

30.6. RED HAT VIRTUALIZATION

这个驱动通过其 RESTful API 控制 Red Hat Virtualization (RHV) 中的虚拟机。

pm_type

将这个选项设置为 **staging-ovirt**。

pm_user; pm_password

RHV 环境的用户名和密码。该用户名中还含有认证供应商。例如：admin@internal。

pm_addr

RHV REST API 的 IP 地址。

pm_vm_name

要控制的虚拟机的名称。

mac

节点上的网络接口的 MAC 地址列表。对于每个系统的 Provisioning NIC，只使用 MAC 地址。

- 要启用此驱动程序，将 **staging-ovirt** 添加到 **undercloud.conf** 的 **enabled_hardware_types** 选项中，并重新运行 **openstack undercloud install** 命令。

30.7. MANUAL-MANAGEMENT 驱动程序

使用 **manual-management** 驱动程序控制没有电源管理的裸机设备。director 无法控制已注册的裸机设备，您必须在内省和部署过程中的特定点执行手动电源操作。

重要

此选项仅适用于测试和评估目的。不建议用于 Red Hat OpenStack Platform 企业环境。

pm_type

将此选项设置为 manual-management。

- *这个驱动不使用任何验证信息，因为它不控制电源管理。*
- *要启用此驱动程序，将 manual-management 添加到 undercloud.conf 的 enabled_hardware_types 选项中，并重新运行 openstack undercloud install 命令。*
- *在 instackenv.json 节点清单文件中，为您要手动管理的节点将 pm_type 设置为 manual-management。*

内省

- *在节点上执行内省时，先手动启动节点，再运行 openstack overcloud node introspect 命令。确保节点通过 PXE 引导。*
- *如果启用了节点清理，在出现 内省完成 消息后手动重新引导节点，并在运行 openstack baremetal node list 命令时 清理 节点状态。确保节点通过 PXE 引导。*
- *在内省和清理过程完成后，关闭节点。*

Deployment

- *执行 overcloud 部署时，请使用 openstack baremetal node list 命令检查节点状态。等待节点状态从 deploying 变为 wait call-back，然后手动启动节点。确保节点通过 PXE 引导。*
- *在 overcloud 置备过程完成后，节点将关闭。您必须从磁盘引导节点才能启动配置过程。要检查置备过程，请使用 openstack baremetal node list 命令检查节点状态，并等待每个节点的节点状态变为 active。当节点状态为 active 时，请手动启动调配的 overcloud 节点。*