Red Hat OpenStack Platform 11

director 的安装和使用

使用 Red Hat OpenStack Platform director 创建 OpenStack 云环境

OpenStack 团队

Red Hat OpenStack Platform 11 director 的安装和使用

使用 Red Hat OpenStack Platform director 创建 OpenStack 云环境

OpenStack 团队
rhos-docs@redhat.com

摘要

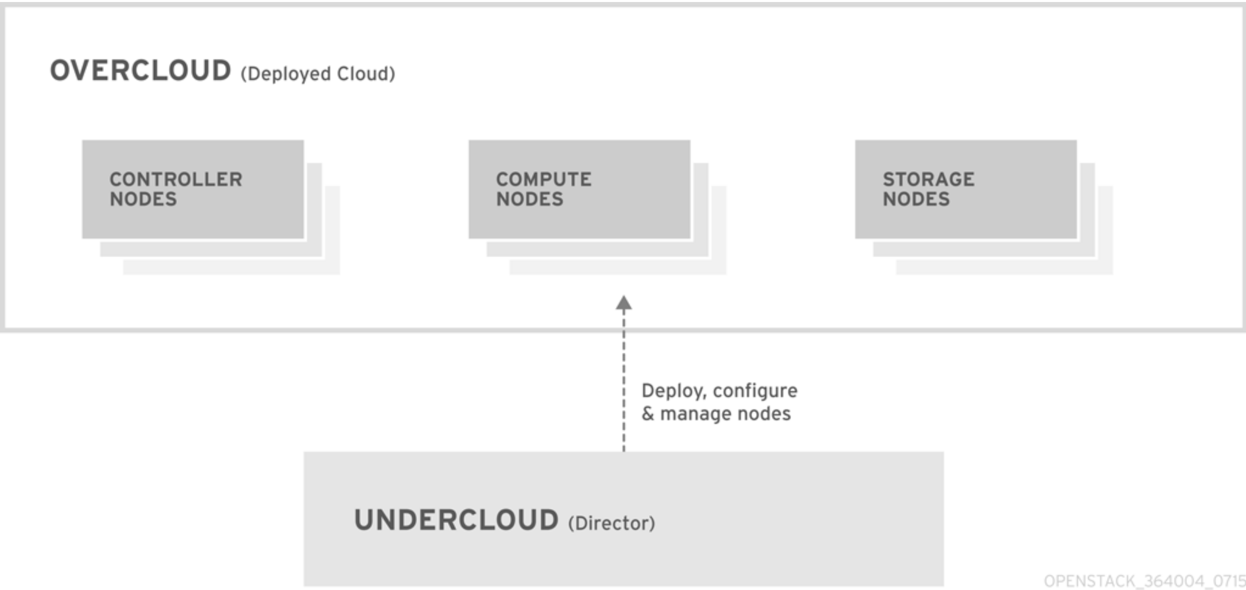
本指南包括了如何使用 Red Hat OpenStack Platform director 在企业级环境中安装 Red Hat OpenStack Platform 11 的信息。这包括安装 director、规划您的环境以及使用 director 创建一个 OpenStack 环境。

目录

第 1 章 简介

Red Hat OpenStack Platform director 是一个安装和管理完整的 OpenStack 环境的工具组，它主要基于 OpenStack 项目 - TripleO（"OpenStack-On-OpenStack" 的缩写）。这个项目利用 OpenStack 组件来安装一个可以完全工作的 OpenStack 环境。它包括了新的 OpenStack 组件来部署并控制裸机系统作为 OpenStack 的节点，这为安装一个完整、稳定、高效的 Red Hat OpenStack Platform 环境提供了一个简洁的方法。

Red Hat OpenStack Platform director 使用两个主要概念：undercloud 和 overcloud。undercloud 可以安装并配置 overcloud。在以下的几个小节中会对这两个概念进行介绍。



1.1. UNDERCLOUD

undercloud 是主要的 director 节点，它是由单一系统组成的一个 OpenStack 安装，并包括了部署和管理 OpenStack 环境（overcloud）所需的组件。undercloud 的组件具有多个功能：

环境规划

undercloud 为用户提供了创建并分配节点角色的规划功能。undercloud 包括了一组默认节点，如 Compute、Controller 和不同存储角色。另外，它还提供了使用自定义节点的功能。您可以选择在每个节点上包括哪些 OpenStack Platform 服务，这提供了一个模式化新节点类型，或在它们自己的主机上分离特定组件的方法。

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，即 Intelligent Platform Management Interface，简称 IPMI）来进行电源管理控制，并使用一个基于 PXE 的服务来发现硬件属性并在每个节点上安装 OpenStack。通过这个功能，可以把裸机系统部署为 OpenStack 节点。如需获得完整的电源管理驱动程序列表，请参阅 附录 B, *电源管理驱动*。

编配

undercloud 提供了一组 YAML 模板作为您的环境的一组计划。undercloud 导入这些计划，并根据其中的指示创建所需的 OpenStack 环境。在这些计划中还包括了一些 hook，您可以使用它们在创建环境的某些特定阶段对环境进行定制。

命令行工具和 Web 用户界面

Red Hat OpenStack Platform director 通过基于终端命令行接口或基于 web 的用户接口来使用 undercloud 的功能。

undercloud 组件

undercloud 使用 OpenStack 组件作为它的基本工具集。它包括以下组件：

- 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 Server（glance） - 存储写到裸机上的镜像。
- 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 Telemetry（ceilometer） - 监控并采集数据。这还包括：
    - OpenStack Telemetry Metrics（gnocchi） - 为 metrics 提供一个时间序列数据库。
    - OpenStack Telemetry Alarming（aodh） - 为监控提供一个警告组件。
    - OpenStack Telemetry Event Storage（panko）- 为监控提供事件存储。
- 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 Platform 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. OVERCLOUD

overcloud 是一个通过 undercloud 创建的 Red Hat OpenStack Platform 环境，它包括了一组基于所要创建的 OpenStack Platform 环境的不同节点角色。undercloud 包括了一组默认的 overcloud 节点角色：

Controller

为 OpenStack 环境提供管理、网络和高可用性服务的节点。在一个理想的 OpenStack 环境中，推荐在一个高可用性集群中使用 3 个 Controller 节点。

一个默认 Controller（控制器）节点包括以下组件：

- 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 Telemetry (ceilometer)
- OpenStack Telemetry Metrics (gnocchi)
- OpenStack Telemetry Alarming (aodh)
- OpenStack Clustering (sahara)
- OpenStack Shared File Systems（manila）
- OpenStack Bare Metal（ironic）
- MariaDB
- Open vSwitch
- 高可用性服务的 Pacemaker 和 Galera。

Compute

为 OpenStack 环境提供计算资源的节点。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

Storage

为 OpenStack 环境提供存储的节点。这包括：

- Ceph Storage 节点 - 用来组成存储集群，每个节点包括一个 Ceph Object Storage Daemon（OSD）。另外，director 会在实施 Ceph Storage 节点的 Controller 节点上安装 Ceph Monitor。
- Block storage（cinder）- 作为 HA 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 (ceilometer) 代理
  - Open vSwitch.
- Object storage (swift) - 这些节点为 Openstack Swift 提供了一个外部的存储层。Controller 节点通过 Swift 代理访问这些节点。这个节点包括以下组件：
  - OpenStack Object Storage (swift) 存储

- OpenStack Telemetry (ceilometer) 代理
- Open vSwitch.

1.3. 高可用性

Red Hat OpenStack Platform director 使用一个 Controller 节点集群来为 OpenStack Platform 环境提供高可用性功能。director 在每个 Controller 节点上安装一组重复的组件，这种集群配置在单一 Controller 节点出现问题时，提供了一个故障转移的功能，从而在一定程度上为 OpenStack 用户提供了不间断的服务。

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

- Pacemaker - Pacemaker 是集群资源的管理者，它会管理并监控一个集群中所有节点上的 OpenStack 组件的可用性。
- HA Proxy（HA 代理） - 为集群提供负载平衡和代理服务。
- Galera - 提供在集群中复制 OpenStack Platform 数据库的服务。
- Memcached - 提供数据库缓存服务。

注意

Red Hat OpenStack Platform director 会在 Controller 节点上自动配置大多数的高可用性设置。但是，仍然需要手工配置节点来启用隔离（fencing）和电源管理功能。这个指南中包括了相关的内容。

1.4. CEPH 存储

大型的机构通常需要使用 OpenStack 来为成千上万的客户端系统提供服务，而每个 OpenStack 客户端系统都会有自己对块存储资源的要求。在一个单一的节点上安装 glance (images)、cinder (volumes) 和/或 nova (compute) 来管理一个大型系统中的成千上万的客户系统将变得非常不现实。这个问题可以通过外部扩展 OpenStack 加以解决。

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来扩展 Red Hat OpenStack Platform 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，它具有高可用性和高效性。虽然虚拟化可能会在牺牲系统性能的情况下实现，但是 Ceph 会把集群中的块设备镜像作为对象来处理，这意味着大型的 Ceph Block 设备镜像有比独立磁盘更好的性能。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。

如需了解更多与 Red Hat Ceph Storage 相关的信息，请参阅 Red Hat Ceph Storage。

第 2 章 配置要求

设置一个环境来使用 director 部署 Red Hat OpenStack Platform 对相关系统有一定的配置要求。本章包括了与设置和访问 director 相关的系统配置要求信息，以及对使用 direct 部署用来提供 OpenStack 服务的主机的硬件配置要求。

注意

在部署 Red Hat OpenStack Platform 前，务必要考虑可用部署方法的特征。如需了解更多相关信息，请参阅安装和管理 Red Hat OpenStack Platform。

2.1. 环境配置要求

最低要求

- 一个作为 Red Hat OpenStack Platform director 的主机
- 一个作为 Red Hat OpenStack Platform Compute 节点的主机
- 一个作为 Red Hat OpenStack Platform Controller 节点的主机

推荐的配置要求：

- 一个作为 Red Hat OpenStack Platform director 的主机
- 3 个作为 Red Hat OpenStack Platform Compute 节点的主机
- 3 个作为一个集群中的 Red Hat OpenStack Platform Controller 节点的主机
- 3 个作为一个集群中的 Red Hat Ceph Storage 节点的主机

请注意：

- 推荐所有主机都使用裸机系统。最起码，Compute 节点需要使用裸机系统。
- 因为 director 控制电源管理，所以所有 overcloud 裸机系统都需要一个智能平台管理接口（IPMI）。

警告

如果不同时从 Open vSwitch (OVS) 2.4.0 升级到 OVS 2.5.0，请勿升级到 Red Hat Enterprise Linux 7.3 内核。倘若仅升级内核，OVS 将停止工作。

2.2. UNDERCLOUD 配置要求

托管 director 的 undercloud 系统为 overcloud 中所有节点提供部署和管理服务。

- 支持 Intel 64 或 AMD64 CPU 扩展的 8 核 64 位 x86 处理器。
- 最少 16GB 内存。

- 根磁盘上需要至少 40 GB 的可用磁盘空间。在进行 overcloud 部署或更新前，需要保证至少有 10 GB 的可用磁盘空间。这些可用磁盘空间用来保存节点部署过程中的镜像转换和缓存。
- 最少两张 1 Gbps 网卡。推荐为 Provisioning 网络流量使用 10 Gbps 接口，特别是部署 overcloud 环境中大量节点时。
- 安装 Red Hat Enterprise Linux 7.3 作为主机操作系统。
- 已在主机上启用 SELinux。

重要

如果使用逻辑卷管理（LVM），请确保 undercloud 的文件系统仅包含根分区和交换分区。如需更多信息，请参阅红帽客户门户上的文章“安装 undercloud 后 Director 节点无法启动”。

2.2.1. 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）	托管于 Red Hat Enterprise Linux 5、6 和 7，详见认证虚拟机管理器名单
Red Hat Enterprise Virtualization	托管于 Red Hat Enterprise Virtualization 3.0、3.1、3.2、3.3、3.4、3.5 、3.6 和 4.0，详见认证虚拟机管理器名单
Microsoft Hyper-V	托管于各种版本的 Hyper-V，详见红帽客户门户认证目录。
VMware ESX 和 ESXi	托管于各种版本的 ESX 和 ESXi，详见红帽客户门户认证目录。

重要

Red Hat OpenStack Platform director 11 需要使用最新版本的 Red Hat Enterprise Linux 作为主机操作系统。这表示，您的虚拟化平台也必须支持底层 Red Hat Enterprise Linux 版本。

虚拟机要求

虚拟 undercloud 的要求与裸机 undercloud 的相似。在部署时，您应当考虑各种不同的调优选项，如网络模型、客户机 CPU 配置、存储后端、存储格式和缓存模式等。

网络注意事项

注意虚拟化 undercloud 的下列事项：

电源管理

undercloud 虚拟机要求访问 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

用于部署（**`ctlplane`**）网络的 NIC 必须能够对 overcloud 裸机节点的 DHCP 请求进行广播和服务。建议创建一个网桥，将虚拟机的 NIC 连接到与裸机 NIC 相同的网络。

> **注意**
>
> 虚拟机管理器技术阻止 undercloud 从自未知地址传输流量时存在一个常见问题。如果使用 Red Hat Enterprise Virtualization，可通过禁用 **`anti-mac-spoofing`** 来避免此问题。如果使用 VMware ESX 或 ESXi，可通过允许伪传输来避免此问题。

示例架构

这仅仅是利用 KVM 服务器的基本 undercloud 虚拟化架构的示例。您可以根据自己的网络和资源要求，在此基础上进行构建。

KVM 主机使用两个 Linux 网桥：

br-ex（eth0）

- 提供对 undercloud 的外部访问
- 外部网络上的 DHCP 服务器利用虚拟 NIC（eth0）把网络配置分配到 undercloud
- 为 undercloud 提供对裸机服务器的电源管理接口的访问

br-ctlplane（eth1）

- 连接到裸机 overcloud 节点所在的网络
- undercloud 通过虚拟 NIC（eth1）处理 DHCP 和 PXE 引导请求
- overcloud 的裸机服务器在这一网络上通过 PXE 来引导

KVM 主机需要以下软件包：

```
$ yum install libvirt-client libvirt-daemon qemu-kvm libvirt-daemon-
driver-qemu libvirt-daemon-kvm virt-install bridge-utils rsync
```

以下命令会在 KVM 主机上创建 undercloud 虚拟机，同时创建连接对应网桥的两个虚拟 NIC：

```
$ virt-install --name undercloud --memory=16384 --vcpus=4 --location
/var/lib/libvirt/images/rhel-server-7.3-x86_64-dvd.iso --disk size=100
--network bridge=br-ex --network bridge=br-ctlplane --graphics=vnc --
hvm --os-variant=rhel7
```

这将启动 **libvirt** 域。使用 **virt-manager** 进行连接，并逐步完成安装过程。此外，您也可通过以下选项包含 kickstart 文件来执行无人看守安装：

```
--initrd-inject=/root/ks.cfg --extra-args "ks=file:/ks.cfg"
```

安装完成后，以 **root** 用户身份通过 SSH 连接实例，再按照???一章中的说明操作。

备份

若要备份虚拟 undercloud，可以从多个解决方案中选择：

- **选项 1：**按照备份和恢复 Director undercloud 指南中的说明。
- **选项 2：**关闭 undercloud，再复制 undercloud 虚拟机存储后备文件。
- **选项 3：** 如果您的虚拟机管理器支持实时或原子快照，可创建 undercloud 虚拟机的快照。

如果使用的是 KVM 服务器，请通过以下步骤来创建快照：

1. 确保 undercloud 客户机虚拟机上正在运行 **qemu-guest-agent**。
2. 创建正在运行的虚拟机的实时快照：

```
$ virsh snapshot-create-as --domain undercloud --disk-only --atomic --
quiesce
```

1. 复制 QCOW 后备文件（现为只读）

```
$ rsync --sparse -avh --progress
/var/lib/libvirt/images/undercloud.qcow2 1.qcow2
```

1. 将 QCOW 覆盖文件合并到后备文件，再将 undercloud 虚拟机切回到使用原始的文件：

```
$ virsh blockcommit undercloud vda --active --verbose --pivot
```

2.3. 网络要求

undercloud 主机最少需要两个网络：

- Provisioning 网络 - 提供 DHCP 和 PXE 引导功能来帮助发现裸机系统以在 overcloud 中使用。通常情况下，此网络需要在一个主干接口中使用一个原生的 VLAN，从而使 director 处理 PXE 引导和 DHCP 请求。虽然一些服务器硬件的 BIOS 支持从 VLAN 进行 PXE 引导，但 BIOS 必须同时支持在引导后把 VLAN 转换为原生 VLAN，否则将无法访问 undercloud。当前，只有一小部分的服务器硬件完整支持此功能。另外，此网络还用于通过智能平台管理接口（IPMI）控制所有 overcloud 节点的电源管理。
- External 网络 - 用来远程连接到所有节点的一个独立网络。连接到这个网络的接口需要一个可路由的 IP 地址（静态定义或通过一个外部 DHCP 服务动态分配）。

以上为至少需要的网络数量。不过，director 可以将其他 Red Hat OpenStack Platform 网络流量隔离到其他网络。Red Hat OpenStack Platform 支持将物理接口和 tagged VLAN 用于网络隔离。

请注意：

- 典型的最小 overcloud 网络配置可以包括：
  - 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）。
  - 双 NIC 配置 - 一个 NIC 用于 Provisioning 网络，另外一个 NIC 作为 External 网络。
  - 双 NIC 配置 - 一个 NIC 在原生 VLAN 中用于 Provisioning 网络，另外一个用于 tagged VLAN（使用子网处理不同的 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。
- 额外的物理 NIC 可以用来分离不同的网络、创建绑定的接口或处理 tagged VLAN 的网络数据。
- 如果使用 VLAN 分离网络流量类型，使用支持 802.1Q 标准的交换机来提供 tagged VLAN。
- 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。
- 确保 Provisioning 网络的 NIC 和在 director 机器上用来进行远程连接的 NIC 不同。director 会使用 Provisioning NIC 创建一个网桥，它会忽略所有远程连接。在 director 系统上需要使用 External NIC 进行远程连接。
- 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 地址数量：
  - 最少为每个连接到 Provisioning 网络的节点包括一个 IP。
  - 如果有高可用性配置，则需要包括一个额外的 IP 地址来作为集群的虚拟 IP。
  - 为扩展环境准备额外的 IP 地址。

  注意

  在 Provisioning 网络中需要避免重复的 IP 地址。相关信息，请参阅 第 3.2 节 “规划网络”。

  注意

  如需了解更多与 IP 地址使用规划相关的信息（如用于存储网络、供应商网络和租户网络），请参阅联网指南。

- 把所有 overcloud 系统设置为使用 Provisioning NIC 进行 PXE 引导，并在 External NIC 以及系统的所有其他 NIC 上禁用 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）。这将使得 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。稍后设置 overcloud 节点时需要用到这些信息。
- 如果一个实例需要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，并把它和这个实例相关联。这个实例仍然会保留它的私人 IP，但网络数据可以通过 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 地址只会为一个租户预留以供使用，租户可以根据需要关联或取消关联一个特定的实例。这个配置会使您的环境暴露于外部网络，您需要确保使用了适当的安全保护措施。

重要

OpenStack Platform 环境的安全性在一定程度上取决于网络的安全性。在您的网络环境中使用适当的安全性措施来确保可以正确地控制网络访问。例如：

- 使用网络分段（network segmentation）技术来控制网络数据并隔离敏感数据。扁平化网络（flat network）通常不是非常安全。
- 限制对服务和端口的访问。
- 使用适当的防火墙设置以及密码。
- 启用 SELinux。

如需了解更多与系统安全相关的信息，请参阅：

- Red Hat Enterprise Linux 7 Security Guide
- Red Hat Enterprise Linux 7 SELinux User’s and Administrator’s Guide

2.4. OVERCLOUD 的配置要求

以下小节详细介绍 overcloud 安装中各个系统和节点的要求。

注意

当前还不支持从 SAN（FC-AL、FCoE、iSCSI）引导 overcloud 节点。

2.4.1. Compute 节点的配置要求

Compute 节点负责运行虚拟机实例。它们必须支持硬件虚拟化，并需要有足够的内存和磁盘空间来支持它们所运行的虚拟机。

处理器

支持带有 Intel 64 或 AMD64 CPU 扩展并启用了 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。

内存

最少 6 GB 内存，再加上提供给虚拟机实例使用的内存。

磁盘空间

最少具有 40GB 可用磁盘空间。

网据接口卡

最少一个 1 Gbps 网络接口卡。但在生产环境中，推荐最少使用两个网卡。额外的网卡可以组成绑定接口，或处理标记的 VLAN 网络（tagged VLAN）流量。

电源管理

每个 Controller 节点都需要在服务器的主板上有一个被支持的电源管理接口（如 IPMI）。

2.4.2. Controller 节点的要求

Controller 节点用来托管 RHEL OpenStack Platform 环境中的核心服务，如 Horizon 仪表板、后端数据库服务器、Keystone 认证和高可用性服务。

处理器

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

内存

至少 16 GB 的内存。不过，建议根据 CPU 内核数量来决定内存大小。请参考以下计算方式：

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

有关测量内存要求的更多信息，请参阅红帽客户门户上的“Red Hat OpenStack Platform 高可用控制器硬件要求”。

磁盘空间

最少具有 40GB 可用磁盘空间。

网据接口卡

最少两个 1 Gbps 网络接口卡。额外的网卡可以组成绑定接口，或处理标记的 VLAN 网络（tagged VLAN）流量。

电源管理

每个 Controller 节点都需要在服务器的主板上有一个被支持的电源管理接口（如 IPMI）。

2.4.3. Ceph 存储节点的要求

Ceph 存储节点负责在 RHEL OpenStack Platform 环境中提供对象存储。

处理器

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

内存

所需的内存数量取决于存储空间的数量。理想情况下，每 1TB 硬盘空间需要最少 1GB 内存。

磁盘空间

所需的存储数量取决于内存空间的数量。理想情况下，每 1TB 硬盘空间需要最少 1GB 内存。

磁盘配置

推荐的 Red Hat Ceph Storage 节点配置需要和以下磁盘配置类似：

- **`/dev/sda`** - root 磁盘。director 把主 overcloud 镜像复制到这个磁盘。
- **`/dev/sdb`** - journal 磁盘。这个磁盘被分为不同的分区来保存 Ceph OSD 的日志信息。例如，**`/dev/sdb1`**、**`/dev/sdb2`**、**`/dev/sdb3`** 等。 journal 磁盘通常需要是一个固态磁盘（SSD）来保证系统的性能。
- **`/dev/sdc`** 和后续 - OSD 磁盘。可以根据您的存储需要使用多个磁盘。

  本文档包括了把您的 Ceph 存储磁盘映射到 director 的方法。

网据接口卡

最少一个 1 Gbps 网络接口卡。但在生产环境中，推荐最少使用两个网卡。额外的网卡可以组成绑定接口，或处理标记的 VLAN 网络（tagged VLAN）流量。推荐为存储节点使用10 Gbps 接口，特别是所创建的 OpenStack Platform 环境需要处理大量网络数据时。

电源管理

每个 Controller 节点都需要在服务器的主板上有一个被支持的电源管理接口（如 IPMI）。

重要

director 不会在 journal 磁盘上创建分区。在 Director 部署 Ceph Storage 节点前，您需要手工创建这些 journal 分区。

Ceph Storage OSD 和 journals 分区需要 GPT 磁盘标签，您可以提前对这些标签进行配置。例如，使用以下命令，在潜在的 Ceph Storage 主机上为一个磁盘或分区创建一个 GPT 磁盘标签：

```
# parted [device] mklabel gpt
```

2.4.4. 对象存储节点要求

对象存储节点提供 overcloud 的对象存储层。对象存储代理安装在控制器节点上。存储层要求每一个裸机节点装有多个磁盘。

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。理想状态下，每 1TB 硬盘空间需要至少 1GB 内存。为获得最佳性能，建议每 1TB 硬盘空间使用 2GB 内存，尤其是小文件（100GB 以下）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 驱动器存储帐户和容器数据。帐户和容器数据大约占对象数据的 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。如果存储的大部分是小对象，则需要提供较多的 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

所推荐的节点配置需要类似于如下的磁盘布局：

- **/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。

网据接口卡

最少两个 1 Gbps 网络接口卡。额外的网卡可以组成绑定接口，或处理标记的 VLAN 网络（tagged VLAN）流量。

电源管理

每个 Controller 节点都需要在服务器的主板上有一个被支持的电源管理接口（如 IPMI）。

2.5. 软件仓库的要求

undercloud 和 overcloud 都需要通过 Red Hat Content Delivery Network 或者 Red Hat Satellite 5 或 6 来访问红帽存储库。如果使用 Red Hat Satellite Server，请将所需的存储库与您的 OpenStack Platform 环境进行同步。请参考下方的 CDN 频道名列表：

警告

如果不同时从 Open vSwitch (OVS) 2.4.0 升级到 OVS 2.5.0，请勿升级到 Red Hat Enterprise Linux 7.3 内核。倘若仅升级内核，OVS 将停止工作。

表 2.1. OpenStack Platform 软件仓库

名称	软件仓库	描述
Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rpms**	基本操作系统的软件仓库。
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	**rhel-7-server-extras-rpms**	包括 Red Hat OpenStack Platform 的依赖软件包。
Red Hat Enterprise Linux 7 Server - RH Common (RPMs)	**rhel-7-server-rh-common-rpms**	包括部署和配置 Red Hat OpenStack Platform 的工具程序。

Red Hat Satellite Tools for RHEL 7 Server RPMs x86_64	**`rhel-7-server-satellite-tools-6.2-rpms`**	使用 Red Hat Satellite 6 管理主机的工具。
Red Hat Enterprise Linux High Availability (for RHEL 7 Server) (RPMs)	**`rhel-ha-for-rhel-7-server-rpms`**	Red Hat Enterprise Linux 的高可用性工具。用于 Controller 节点的高可用性功能。
Red Hat Enterprise Linux OpenStack Platform 11 for RHEL 7 (RPMs)	**`rhel-7-server-openstack-11-rpms`**	Red Hat OpenStack Platform 核心存储库。也包含 Red Hat OpenStack Platform director 的软件包。
Red Hat Ceph Storage OSD 2 for Red Hat Enterprise Linux 7 Server (RPMs)	**`rhel-7-server-rhceph-2-osd-rpms`**	（Ceph Storage 节点）Ceph Storage Object Storage 守护进程的软件仓库。在 Ceph Storage 节点上安装。
Red Hat Ceph Storage MON 2 for Red Hat Enterprise Linux 7 Server (RPMs)	**`rhel-7-server-rhceph-2-mon-rpms`**	（Ceph Storage 节点）Ceph Storage Monitor 守护进程的软件仓库。在使用 Ceph Storage 节点的 OpenStack 环境的 Controller 节点上安装。
Red Hat Ceph Storage Tools 2 for Red Hat Enterprise Linux 7 Workstation (RPMs)	**`rhel-7-server-rhceph-2-tools-rpms`**	（适用于 Ceph Storage 节点）为 Ceph 对象存储提供所需的 Rados REST 网关。

注意

如需在一个离线网络中为 Red Hat OpenStack Platform 环境配置软件仓库，请参阅红帽客户门户网站所提供的 "Configuring Red Hat OpenStack Platform Director in an Offline Environment"。

第 3 章 规划您的 OVERCLOUD

以下一节提供了与规划您的 Red Hat OpenStack Platform 环境中的各个环境相关的信息。这包括定义节点角色、规划您的网络拓扑结构和存储。

3.1. 规划节点的实施角色

director 为构建 overcloud 提供了多个默认的节点类型。这些节点类型是：

Controller

为控制您的环境提供关键服务。它包括 dashboard 服务（horizon）、用户验证服务（keystone）、镜像存储服务（glance）、网络服务（neutron）和编配服务（heat），以及在使用多个 Controller 节点时的高可用性服务。一个基本的 Red Hat OpenStack Platform 环境中需要最少一个 Controller 节点。

Compute

一个作为虚拟机监控程序（hypervisor）的物理服务器，它为环境中运行的虚拟机提供处理能力。一个基本的 Red Hat OpenStack Platform 环境中需要最少一个 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。这个实施角色是可选的。

Cinder-Storage

为 OpenStack 的 cinder 服务提供外部块存储的主机。这个实施角色是可选的。

Swift-Storage

为 OpenStack 的 Swift 服务提供外部对象存储的主机。这个实施角色是可选的。

下表提供了几个不同的 overcloud 示例，以及每种情况中的节点数量。

表 3.1. 节点部署

	Controller	Compute	Ceph-Storage	Swift-Storage	Cinder-Storage	总计
小型 overcloud	1	1	-	-	-	2
中型 overcloud	1	3	-	-	-	4

带有额外对象存储和块存储的中型 overcloud	1	3	-	1	1	6
带有高可用性功能的中型 overcloud	3	3	-	-	-	6
带有高可用性和 Ceph 存储的中型 overcloud	3	3	3	-	-	9

此外，还需思考是否要将个别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 *overcloud 高级自定义*指南中的 第 8 章：可编辑角色和服务。

3.2. 规划网络

规划环境中的网络拓扑结构和子网非常重要，您需要把角色和服务进行正确的映射，从而使它们可以进行正确的通信。Red Hat OpenStack Platform 使用 neutron 网络服务，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。budirector 在overcloud 环境的每个 Controller 节点上实施此服务。

Red Hat OpenStack Platform 把不同的服务映射到分配给环境中的不同子网的独立网络流量类型中。这些网络类型包括：

表 3.2. 网络类型分配

网络类型	描述	用于
IPMI	用于节点电源管理的网络。此网络在安装 undercloud 之前预先定义。	所有节点
Provisioning / Control Plane	director 使用此网络类型来通过 PXE 引导实施新的节点，并编配在 overcloud 裸机服务器上进行的 OpenStack Platform 安装。此网络在安装 undercloud 之前预先定义。	所有节点

Internal API	Internal API 网络被用来处理经过 API 、RPC 消息和数据库进行的 OpenStack 服务间的通讯。	Controller、Compute、Cinder Storage、Swift Storage
Tenant	Neutron 为每个租户提供自己的网络。这可以通过使用 VLAN 隔离（VLAN segregation，每个租户网络都是一个网络 VLAN）实现，也可以使用 VXLAN 或 GRE 通道（tunneling）实现。每个租户网络的网络数据会被相互隔离，并都有一个相关联的 IP 子网。通过网络命名空间，多个租户子网可以使用相同的地址。	Controller、Compute
Storage	块存储、NFS、iSCSI 和其它存储。在理想情况下，因为性能的原因，这个网络应该位于一个完全独立的网络交换环境中。	所有节点
Storage Management	OpenStack Object Storage（swift）使用这个网络来在相关的副本节点中同步数据项。代理服务（proxy service）在用户请求和底层的存储层间起到一个主机接口的作用。这个代理会接收用户的请求，并找到所需的副本来获得所需的数据。使用 Ceph 作为后端的服务会通过 Storage Management 网络进行连接，因为它们不会和 Ceph 直接进行交流，而是使用前端的服务。请注意，RBD 驱动是个例外，它会直接连接到 Ceph。	Controller、Ceph Storage、Cinder Storage、Swift Storage
External	运行 OpenStack Dashboard（horizon）来进行图形化的系统管理、OpenStack 服务的公共 API 以及对入站网络流量进行 SNAT 处理来把它们导向正确的目标。如果 external 网络使用私有 IP 地址（RFC-1918），还需要对来自于互联网的流量进行额外的 NAT 处理。	Controller

Floating IP	允许入站的网络流量到达相关实例，使用 1 对 1 的 IP 地址映射把浮动 IP 地址和在租户网络中实际分配给实例的 IP 地址相关联。如果在一个与外部网络隔离的 VLAN 中托管浮动 IP，您可以把浮动 IP VLAN 中继到控制器节点，并在 overcloud 创建后通过 Neutron 添加 VLAN。这为创建多个浮动 IP 网络并附加到多个网桥提供了途径。VLAN 会被中继，但不会配置为接口。相反，neutron 会为每个浮动 IP 网络在所选的网桥上创建一个带有 VLAN 分段 ID 的 OVS 端口。	Controller
Management	提供与系统管理相关的功能，如 SSH 访问、DNS 流量和 NTP 流量。这个网络也作为非 Controller 节点的一个网关。	所有节点

在典型的 Red Hat OpenStack Platform 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 标签（VLAN tagging）来为每个接口提供多个网络。大多数的网络都是相互分离的子网，但部分网络需要第 3 层网关来提供路由功能以便接入互联网或连接基础架构网络。

注意

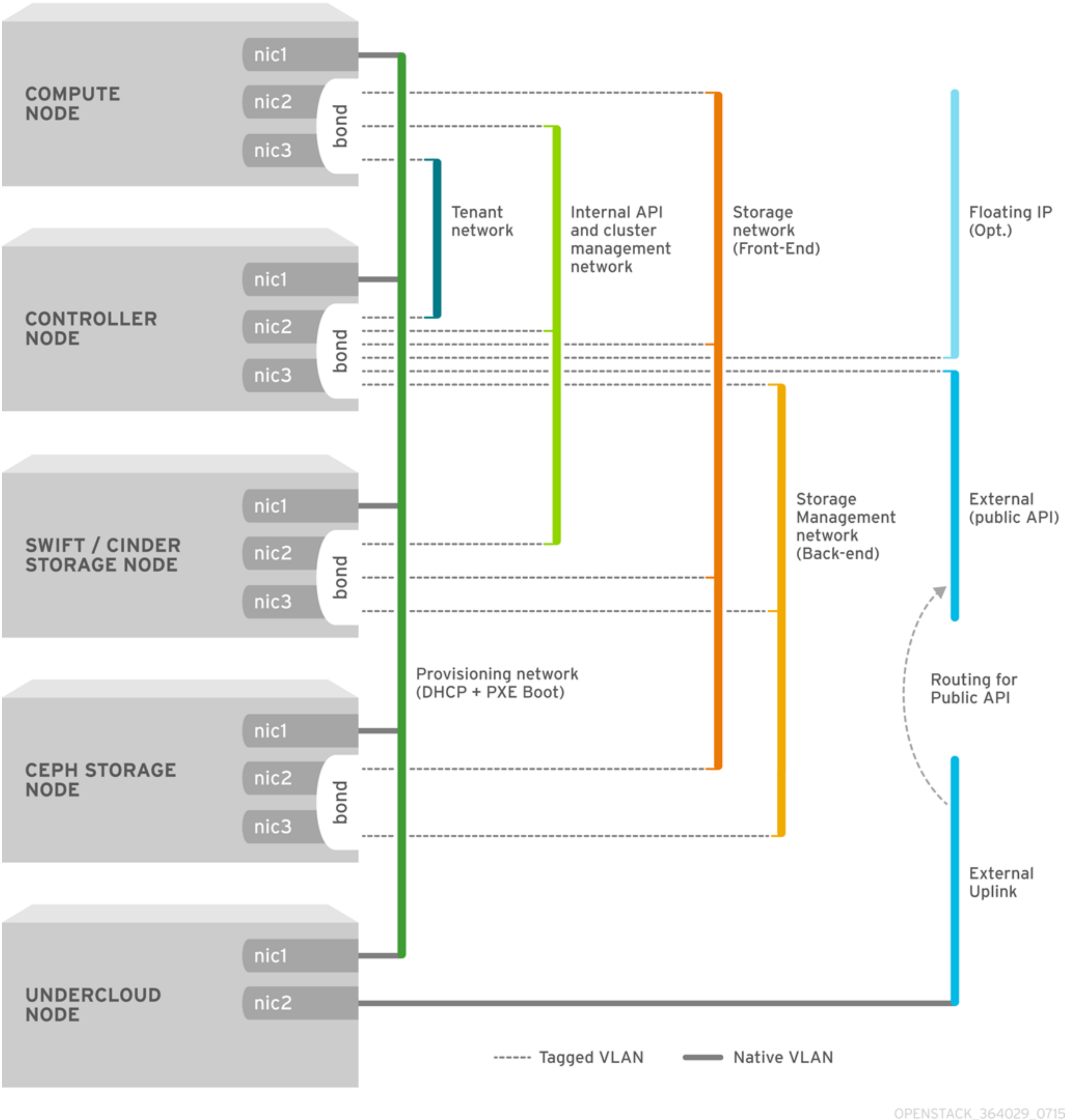
我们推荐，即使在部署时使用了禁用隧道功能的 neutron VLAN，您最好仍然部署一个项目网络（利用 GRE 或 VXLAN 进行隧道连接）。这只需要在部署时进行一些微小的自定义，便可为以后使用网络隧道功能实现工具网络或虚拟化网络留下选择余地。您仍然需要使用 VLAN 创建租户网络，但同时也可为特殊用途网络创建 VXLAN 隧道，而不需要消耗租户 VLAN。VXLAN 功能可以添加到带有租户 VLAN 的部署中，而租户 VLAN 却无法在不对系统运行造成干扰的情况下添加到现有的 overcloud 中。

director 提供了一个为其中的 6 种网络流量类型映射到特定子网或 VLAN 的方法。这些流量类型包括：

- Internal API
- Storage
- Storage Management
- Tenant
- External
- Management

所有没有被分配的网络都会被自动分配到和 Provisioning 网络相同的网络中。

下图显示了一个通过独立的 VLAN 进行分离的网络拓扑结构示例。每个 overcloud 节点都使用绑定的两个接口（**nic2** 和 **nic3**）来通过各自的 VLAN 提供网络功能。同时，所有 overcloud 节点都使用 **nic1** 并借助原生 VLAN 通过 Provisioning 网络跟 undercloud 进行通信。



下表提供了把网络流量映射到不同网络结构中的示例：

表 3.3. 网络映射

	映射	接口总数	VLAN 总数

带有外部连接的平面网络	网络 1 - Provisioning、Internal API、Storage、Storage Management、Tenant Networks 网络 2 - 外部、浮动 IP（在 overcloud 创建后进行映射）	2	2
隔离的网络	Network 1 - Provisioning Network 2 - Internal API Network 3 - Tenant Network Network 4 - Storage Network 5 - Storage Management Network 6 - Storage Management 网络 7 - 外部、浮动 IP（在 overcloud 创建后进行映射）	3（包括 2 个绑定接口）	7

3.3. 规划存储

director 为 overcloud 环境提供不同的存储选项，它们包括：

Ceph Storage 节点

director 使用 Red Hat Ceph Storage 创建一组可扩展存储节点。overcloud 将这些节点用于：

- **镜像** - Glance 管理虚拟机的镜像。镜像是不可变的，OpenStack 把镜像看做为二进制数据块，并根据它们的实际情况进行下载。您可以使用 glance 在一个 Ceph 块设备中存储镜像。
- **卷** - Cinder 卷是块设备。OpenStack 使用卷来引导虚拟机，或把卷附加到运行的虚拟机上。OpenStack 使用 Cinder 服务管理卷，您可以使用 Cinder，通过一个镜像的写时复制（copy-on-write，简称 COW） 克隆引导虚拟机。
- **客户端磁盘** - 客户端磁盘就是客户端（虚拟机）操作系统磁盘。在默认情况下，当使用 nova 引导虚拟机时，它的磁盘会以一个文件的形式出现在虚拟机监控程序（hypervisor）上，通常位于 `/var/lib/nova/instances/<uuid>/`。现在，可以直接在 Ceph 中引导每个虚拟机，而不用使用 cinder，这可以使您在实时迁移时更容易地执行维护操作。例如，如果您的 hypervisor 出现问题，它还可以方便地触发 `nova evacuate`，从而使在其它地方运行虚拟机的过程几乎可以“无缝”地实现。

重要

Ceph 不支持提供 QCOW2 虚拟机磁盘。如果您需要在 Ceph 中引导虚拟机（使用临时后端或从卷引导），glance 镜像的格式必须是 **`RAW`**。

如需了解更详细的信息，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 环境中的控制器节点，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。

第 4 章 安装 UNDERCLOUD

创建 Red Hat OpenStack Platform 环境的第一步是在 undercloud 系统上安装 director。这需要进行一些先期的步骤来启用所需的订阅和存储库。

4.1. 创建 DIRECTOR 安装用户

director 的安装过程需要一个非 root 的用户来执行命令。使用以下命令创建一个名为 **stack** 的用户并设置密码：

```
[root@director ~]# useradd stack
[root@director ~]# passwd stack  # specify a password
```

进行以下操作以使这个用户使用 **sudo** 时不需要输入密码：

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

切换到新的 **stack** 用户：

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

使用 **stack** 用户继续安装的过程。

4.2. 为模板和镜像创建目录

director 使用系统镜像和 Heat 模板来创建 overcloud 环境。为保持文件的有序性，我们推荐创建以下目录来组织镜像和模板文件：

```
$ mkdir ~/images
$ mkdir ~/templates
```

本指南中的其它部分会使用这两个目录来保存特定的文件。

4.3. 为系统设置主机名

director 的安装和配置过程需要一个完全限定域名（FQDN），这意味着您需要为 director 的主机设置主机名（hostname）。使用以下命令检查主机的主机名：

```
$ hostname    # Checks the base hostname
$ hostname -f # Checks the long hostname (FQDN)
```

如果需要，使用 **hostnamectl** 设置主机名：

```
$ sudo hostnamectl set-hostname manager.example.com
$ sudo hostnamectl set-hostname --transient manager.example.com
```

另外，director 还需要在**/etc/hosts** 文件中包括一个带有系统主机名的项。例如，系统名是 **manager.example.com**，**/etc/hosts** 则需要包括一个和以下类似的项：

```
127.0.0.1   manager.example.com manager localhost localhost.localdomain
localhost4 localhost4.localdomain4
```

4.4. 注册系统

若要安装 Red Hat OpenStack Platform director，首先需要使用 Red Hat Subscription Manager 注册主机系统，并且订阅所需的频道。

在 Content Delivery Network 中注册您的系统，在出现提示时输入您的用户门户网站的用户名和密码：

```
$ sudo subscription-manager register
```

找到 Red Hat OpenStack Platform director 所在的权利池。

```
$ sudo subscription-manager list --available --all
```

使用上一步中获得的池 ID 添加 Red Hat OpenStack Platform 11 权力：

```
$ sudo subscription-manager attach --pool=pool_id
```

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

```
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos --enable=rhel-7-server-rpms --
enable=rhel-7-server-extras-rpms --enable=rhel-7-server-rh-common-rpms
--enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-server-
openstack-11-rpms
```

这些仓库包括了安装 director 所需的软件包。

重要

仅启用 第 2.5 节 “软件仓库的要求”中列出的存储库。其他存储库都可能会造成软件包和软件冲突，请勿启用任何其他存储库。

对系统上的软件进行一个更新来确保使用了最新的基本系统软件包：

```
$ sudo yum update -y
$ sudo reboot
```

现在，这个系统已经准备好可以进行 director 安装了。

4.5. 安装 DIRECTOR 软件包

使用以下命令安装 director 所需的命令行工具：

```
$ sudo yum install -y python-tripleoclient
```

▪

这个命令会安装 director 所需的所有软件包。

4.6. 配置 DIRECTOR

director 的安装过程需要特定的设置来决定您的网络配置。这些设置保存在 **stack** 用户的家目录中的一个模板中（**undercloud.conf**）。

红帽会提供一个基本的模板来帮助您设置安装所需的配置。把这个模板复制到 **stack** 用户的家目录中：

```
$ cp /usr/share/instack-undercloud/undercloud.conf.sample
~/undercloud.conf
```

模板包含两个部分： **[DEFAULT]** 和 **[auth]**。

[DEFAULT] 部分包含下列参数：

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，但用户必须相应地配置所有主机名设置。

local_ip

director 的 Provisioning NIC 的 IP 地址。它同时还是 director 用来作为它的 DHCP 和 PXE 引导服务的 IP 地址。除非您需要为 Provisioning 网络使用不同的子网（例如，默认值与环境中存在的 IP 地址或子网有冲突），否则请保留默认值 **192.168.24.1/24**。

network_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。除非您的 director 使用不同的 IP 地址，或直接使用外部网关，否则请保留默认值 **192.168.24.1**。

注意

director 的配置脚本也可以使用相关的 **sysctl** 内核参数自动启用 IP 转发功能。

undercloud_public_host

使用 SSL/TLS 时 director 的 Public API 的 IP 地址。该 IP 地址用于通过 SSL/TLS 从外部访问 director 端点。director 的配置会把这个 IP 地址附加到它的软件网桥上作为一个路由的 IP 地址（使用 **/32** 网络掩码）。

undercloud_admin_host

使用 SSL/TLS 时 director 的 Admin API 的 IP 地址。该 IP 地址用于通过 SSL/TLS 访问管理端点。director 的配置会把这个 IP 地址附加到它的软件网桥上作为一个路由的 IP 地址（使用 **/32** 网络掩码）。

undercloud_nameservers

用于 undercloud 主机名解析的 DNS 名称服务器列表。

undercloud_ntp_servers

用于帮助同步 undercloud 的日期和时间的网络时间协议服务器列表。

undercloud_service_certificate

用于 OpenStack SSL/TLS 通讯的证书的位置和文件名。最理想的情况是从一个信任的证书认证机构获得这个证书。或者，您也可以根据 附录 A, *SSL/TLS 证书配置* 中的指南生成一个自签发的证书。另外，它还包括了为您的证书（无论是自签发证书还是从证书认证机构获得的证书）设置 SELinux 上下文的方法。

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。

certificate_generation_ca

为所请求证书签名的 CA 的 **`certmonger`** 别名。只有设置了 **`generate_service_certificate`** 参数时才应使用此参数。如果选择 **`local`** CA，certmonger 会把本地 CA 证书提取至 **`/etc/pki/ca-trust/source/anchors/cm-local-ca.pem`**，并将它添加到信任链中。

service_principal

使用该证书的服务的 Kerberos 主体。只有您的 CA 要求 Kerberos 主体时才应使用此参数，例如在 FreeIPA 中。

local_interface

指定 director 的 Provisioning NIC 的接口。它同时还是 director 用来作为它的 DHCP 和 PXE 引导服务的设备。把这个项的值改为您需要使用的值。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。

local_mtu

要用于 **`local_interface`** 的 MTU。

network_cidr

director 用来管理 overcloud 实例的网络，这是由 undercloud 的 **neutron** 服务管理的 Provisioning 网络。除非您的 Provisioning 网络使用了不同的子网，否则请保留默认值 **192.168.24.0/24**。

masquerade_network

定义用于外部访问的网络伪装。这为 Provisioning 网络提供了一定程度的网络地址转换（network address translation，简称 NAT）功能，从而可以通过 director 实现外部访问。除非 Provisioning 网络使用了不同的子网，否则请保留默认值**192.168.24.0/24**。

dhcp_start; dhcp_end

overcloud 节点 DHCP 分配范围的开始值和终止值。请确保此范围可以为节点提供足够的 IP 地址。

hieradata_override

hieradata 覆盖文件的路径。如果设置此参数，undercloud 安装会将此文件复制到 **/etc/puppet/hieradata** 下，并将它设置为层次结构中的第一个文件。此参数可用于在 **undercloud.conf** 参数以外提供服务自定义配置。

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 格式的模板来利用 **os-net-config** 参数配置网络。这将忽略 **undercloud.conf** 中设置的网络参数。如需示例，可参见 **/usr/share/instack-undercloud/templates/net-config.json.template**。

inspection_interface

director 用来进行节点内省的网桥。这是 director 配置创建的一个自定义网桥。**LOCAL_INTERFACE** 会附加到这个网桥。请保留使用默认的值（**br-ctlplane**）。

inspection_iprange

在 PXE 引导和部署过程中，director 内省服务使用的 IP 地址范围（使用逗号分隔范围的起始值和终止值）。例如，**192.168.24.100,192.168.24.120**。请确保这个范围有足够的 IP 地址用于节点，并与 **dhcp_start** 和 **dhcp_end** 指定的范围不冲突。

inspection_extras

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

inspection_runbench

在节点发现过程中运行一组基准数据。把它设置为 **true** 来启用这个功能。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。更详细的相关信息，请参阅 第 5.2 节 “检查节点硬件”。

inspection_enable_uefi

定义是否支持对仅带有 UEFI 固件的节点进行内省

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_pxe** 作为默认驱动器，但您可以设置 **discovery_default_driver** 进行覆盖。您也可以使用内省规则为新注册的节点指定驱动器信息。

discovery_default_driver

为自动注册的节点设置默认驱动器。需要启用 **`enable_node_discovery`** 且必须在**`enabled_drivers`** 列表中包含驱动器。请参阅 附录 B, *电源管理驱动* 了解支持的驱动器列表。

undercloud_debug

把 undercloud 服务的日志级别设置为 **`DEBUG`**。把值设为 **`true`** 来启用它。

undercloud_update_packages

定义是否在安装 undercloud 期间更新软件包。

enable_tempest

定义是否安装检查工具。默认设置是 **`false`**，您可以把它设为 **`true`**。

enable_telemetry

定义是否在 undercloud 中安装 OpenStack Telemetry 服务（ceilometer、aodh、panko、gnocchi）。在 Red Hat OpenStack Platform 11 中，遥测的指标后端由 gnocchi 提供。将 **`enable_telemetry`** 参数设置为 **`true`** 将会自动安装和设置遥测服务。如果您不想在此节点上使用遥测，请将值设置为 **`false`**。

enable_ui

定义是否安装 director 的 Web UI。安装后您可以通过图形 Web 界面来执行 overcloud 规划和部署。如需更多信息，请参阅???。请注意，只有通过 **`undercloud_service_certificate`** 或**`generate_service_certificate`** 启用了 SSL/TLS，才能使用此 UI。

enable_validations

定义是否安装所需项目来运行验证。

enable_legacy_ceilometer_api

定义是否在 undercloud 中启用传统的 OpenStack Telemetry 服务（Ceilometer）API。注意，传统 API 已被弃用，将从未来的版本中移除。请使用通过 **`enable_telemetry`** 安装的更新组件。

enable_novajoin

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

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。

store_events

定义是否在 undercloud 上存储 OpenStack Telemetry 服务 (ceilometer) 中的事件。

ipxe_enabled

定义使用 iPXE 还是标准的 PXE。默认值是 **`true`**，这会使用 iPXE；设置为 **`false`** 将使用标准的 PXE。如需了解更多相关信息，请参阅用户门户网站中的 "Changing from iPXE to PXE in Red Hat OpenStack Platform director" 。

scheduler_max_attempts

调度程序尝试部署实例的次数上限。请将此值设为大于等于您要立即部署的裸机节点数量，从而在调度时避免可能出现的争用情形。

clean_nodes

确定是否在部署之间和内省之后擦除硬盘。

enabled_drivers

为 undercloud 启用的裸机驱动器列表。请参考 附录 B, *电源管理驱动* 了解受支持的驱动器列表。

[auth] 部分包含以下参数：

undercloud_db_password; undercloud_admin_token; undercloud_admin_password; undercloud_glance_password; etc

剩下的参数用来定义 director 服务的访问信息。这些值不需要改变。如果 **undercloud.conf** 为空，director 的配置脚本会自动产生这些值。当配置脚本完成后，您就可以获得所有这些值。

根据网络的具体情况修改这些参数的值。完成后，保存这个文件并运行以下命令：

```
$ openstack undercloud install
```

这会启动 director 的配置脚本。director 会安装额外的软件包，并把它的服务配置为和 **undercloud.conf** 中的设置相符合的情况。这个脚本会需要一些时间来完成。

完成后，配置脚本会产生两个文件：

- **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。

此配置也会自动启用所有 OpenStack Platform 服务。使用以下命令检查已启用的服务：

```
$ sudo systemctl list-units openstack-*
```

运行以下命令初始化 **stack** 用户来使用命令行工具：

```
$ source ~/stackrc
```

您现在可以使用 director 的命令行工具了。

4.7. 为 OVERCLOUD 节点获得镜像

director 需要以下几个磁盘镜像来部署 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。
- 一个实施内核和 ramdisk - 用于系统部署和实施。
- overcloud 内核、ramdisk 和完整镜像 - 写到节点硬盘中的基本 overcloud 系统。

从 **rhosp-director-images** 和 **rhosp-director-images-ipa** 软件包中获得这些镜像：

```
$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

- 把压缩文件展开到 **stack** 用户的家目录下的 **images** 目录中（**/home/stack/images**）：

```
$ cd ~/images
$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-
11.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-
11.0.tar; do tar -xvf $i; done
```

把这些镜像导入到 director：

```
$ openstack overcloud image upload --image-path /home/stack/images/
```

这会把 **bm-deploy-kernel**、**bm-deploy-ramdisk**、**overcloud-full**、**overcloud-full-initrd** 和 **overcloud-full-vmlinuz** 镜像上传到 director。部署操作和 overcloud 需要这些镜像。另外，此脚本还会在 director 的 PXE 服务器上安装内省镜像。

在 CLI 中查看镜像列表：

```
$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk      |
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel       |
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

这个列表没有显示内省 PXE 镜像。director 会把这些文件复制到 **/httpboot**。

```
[stack@host1 ~]$ ls -l /httpboot
total 341460
-rwxr-xr-x. 1 root root   5153184 Mar 31 06:58 agent.kernel
-rw-r--r--. 1 root root 344491465 Mar 31 06:59 agent.ramdisk
-rw-r--r--. 1 root root       337 Mar 31 06:23 inspector.ipxe
```

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。请参阅 附录 C, *完整的磁盘镜像* 了解更多信息。

4.8. 在 UNDERCLOUD 的 NEUTRON 子网中设置名称解析服务器

overcloud 节点需要名称服务器来提供 DNS 主机名解析。对于没有网络隔离的标准 overcloud，此名称服务器在 undercloud 的 **neutron** 子网中定义。使用以下命令定义环境的名称服务器：

```
$ openstack subnet list
$ openstack subnet set --dns-nameserver [nameserver1-ip] --dns-
nameserver [nameserver2-ip] [subnet-uuid]
```

查看子网来验证名称解析服务器：

```
$ openstack subnet show [subnet-uuid]
+-------------------+-----------------------------------------------
-+
| Field             | Value
|
+-------------------+-----------------------------------------------
-+
| ...               |
|
| dns_nameservers   | 8.8.8.8
|
| ...               |
|
+-------------------+-----------------------------------------------
-+
```

重要

如果要将服务流量分离到单独的网络，overcloud 节点将使用网络环境配置文件中的 **DnsServer** 参数。

4.9. 备份 UNDERCLOUD

红帽提供了一个对 undercloud 主机的重要数据以及 Red Hat OpenStack Platform director 进行备份的方法。如需了解更多与 undercloud 备份相关的信息，请参阅“备份和恢复 Director undercloud” 指南。

4.10. 完成 UNDERCLOUD 的配置

undercloud 的配置到此结束。下一章将介绍基本的 overcloud 配置，包括注册节点和检查节点，并把它们标记为不同的节点角色。

第 5 章 使用 CLI 工具配置基本的 OVERCLOUD 要求

本章介绍使用 CLI 工具配置 OpenStack Platform 环境的基本步骤。overcloud 基本配置中不含任何自定义功能。但是，您可以参考 overcloud 高级自定义指南中的说明，添加高级配置选项到这一基本 overcloud 中，并按照您的具体规格进行自定义。

在本章使用的示例中，所有节点都是使用 IPMI 作为电源管理的裸机。如需了解关于其它支持的电源管理类型和它们的选项，请参阅 附录 B, *电源管理驱动*。

流程

1. 在 director 中创建一个节点定义模板并注册空白节点。
2. 检查所有节点的硬件。
3. 为节点添加标签（tag）来标记为角色。
4. 定义额外的节点属性

配置要求

- ??? 中创建的 director 节点
- 一组作为节点的裸机。所需的节点数量由您需要创建的 overcloud 类型所决定（请参阅 第 3.1 节 “规划节点的实施角色”）。这些机器需要满足每个节点类型对系统的要求。相关信息，请参阅 ???。这些节点不需要操作系统，director 会把一个 Red Hat Enterprise Linux 7 镜像复制到每个节点。
- Provisioning 网络的一个网络连接（被配置为一个原生 VLAM）。所有节点必须都连接到这个网络，并需要满足 第 2.3 节 “网络要求” 中的要求。在本章的示例中，我们使用 192.168.24.0/24 作为 Provisioning 子网，分配的 IP 地址信息如下：

表 5.1. Provisioning 网络 IP 分配信息

节点名	IP 地址	MAC 地址	IPMI IP 地址
Director	192.168.24.1	aa:aa:aa:aa:aa:aa	不需要
Controller	DHCP	bb:bb:bb:bb:bb:bb	192.168.24.205
Compute	DHCP	cc:cc:cc:cc:cc:cc	192.168.24.206

- 所有其他网络类型使用 Provisioning 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。

5.1. 为 OVERCLOUD 注册节点

director 需要一个节点定义模板，您可以手动创建该模板。这个文件（**instackenv.json**）是一个 JSON 格式的文件，它包括了节点的详细硬件和电源管理信息。例如，注册两个节点的模板会和以下类似：

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.168.24.205"
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.168.24.206"
        }
    ]
}
```

这个模板使用以下属性：

pm_type

使用的电源管理驱动。在这个示例中使用 IPMI 驱动（**pxe_ipmitool**）。

pm_user; pm_password

IPMI 的用户名和密码。

pm_addr

IPMI 设备的 IP 地址。

mac

（可选）节点上网络接口的 MAC 地址列表。对于每个系统的 Provisioning NIC，只使用 MAC 地址。

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）

> **注意**
>
> 如需了解更多与所支持的电源管理类型和选项相关的信息，请参阅 附录 B, *电源管理驱动*。

在创建完模板后，把这个文件保存到 **stack** 用户的家目录（**/home/stack/instackenv.json**），然后把它导入到 director。使用以下命令：

```
$ openstack overcloud node import ~/instackenv.json
```

这会导入模板，并把模板中的每个节点注册到 director。

完成节点注册和配置之后，在 CLI 中查看这些节点的列表：

```
$ openstack baremetal node list
```

5.2. 检查节点硬件

director 可以在每个节点上运行内省进程。这个进程会使每个节点通过 PXE 引导一个内省代理。这个代理从节点上收集硬件数据，并把信息发送回 director，director 把这些信息保存在运行于 director 上的 OpenStack Object Storage (swift) 服务中。director 使用硬件信息用于不同目的，如进行 profile tagging、benchmarking、手工引导磁盘分配等。

> **注意**
>
> 您也可以创建策略文件来在进行内省后自动把节点标记为（tag）配置集（profile）。如需了解更多与创建策略文件并把它们包括在内省过程中的信息，请参阅 附录 D, *自动配置集标记*。获得，参阅 第 5.3 节 “为节点添加标签（tag）来标记为配置集” 中的内容把节点手工标记为配置集。

运行以下命令检查每个节点的属性：

```
$ openstack overcloud node introspect --all-manageable --provide
```

- **--all-manageable** 选项仅内省处于受管理状态的节点。本例中为所有节点。
- **--provide** 选项将所有节点在内省后重置为 **active** 状态。

在一个单独的终端窗口中运行以下命令来监测内省的进程：

```
$ sudo journalctl -l -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq -u openstack-ironic-conductor -f
```

重要

确保这个过程成功完成。它可能需要 15 分钟来检查这些裸机节点。

内省完成后，所有节点变更为 **active** 状态。

执行单个节点内省

要在 **active** 节点上执行一个内省操作，请将节点设置为管理模式，然后执行内省操作：

```
$ openstack baremetal node manage [NODE UUID]
$ openstack overcloud node introspect [NODE UUID] --provide
```

内省完成后，节点变更为 **active** 状态。

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应进入 **active** 状态。要在初始内省后对所有节点执行内省操作，请将所有节点设置为 **manageable** 状态，然后运行批量内省命令

```
$ for node in $(openstack baremetal node list --fields uuid -f value) ;
do openstack baremetal node manage $node ; done
$ openstack overcloud node introspect --all-manageable --provide
```

内省完成后，所有节点变更为 **active** 状态。

5.3. 为节点添加标签（TAG）来标记为配置集

在注册并检查完每个节点的硬件后，需要为它们添加标签，加入特定的配置文件。这些配置文件标签会把节点和类型（flavor）相匹配，从而使类型分配到部署角色。下方的示例显示了 Controller 节点的角色、类型、配置文件和节点之间的关系：

类型	描述
角色	**Controller** 角色定义了配置控制器节点的方式。
类型	**control** 类型定义了用作控制器的节点的硬件配置文件。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。

类型	描述
节点	您也对单个节点应用 **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。

默认的配置文件类型 **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。

注意

如果有大量的节点，可以使用自动为配置集添加标签的功能。相关信息，请参阅 附录 D, *自动配置集标记*。

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，把环境中的两个节点分别标记为使用 controller 配置集和 compute 配置集，使用以下命令：

```
$ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' 58c3d07e-24f2-48a7-
bbb6-6843f0e8ee13
$ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 1a4e30da-b6dc-499d-
ba87-0bd8a3819bc0
```

其中的 **profile:compute** 和 **profile:control** 选项会把节点标记为相关的配置集。

这些命令同时也设置了 **boot_option:local** 参数，它定义了每个节点的引导模式。

在标记完节点后，检查分配的配置集或可能的配置集：

```
$ openstack overcloud profiles list
```

自定义角色配置集

如果使用自定义角色，您可能需要额外创建类别和配置文件，以便容纳这些新角色。例如，运行以下命令，为 Networker 角色创建新类别：

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --vcpus 1
networker
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="networker" networker
```

分配采用这种新配置集的节点：

```
$ openstack baremetal node set --property
capabilities='profile:networker,boot_option:local' dad05b82-0c74-40bf-
9d12-193184bfc72d
```

5.4. 为节点定义 ROOT DISK

一些节点可能会使用多个磁盘。这意味着，director 需要可以区分在 provisioning 时作为 root 磁盘使用的磁盘。以下的几个属性可以被用来帮助 director 区分 root 磁盘：

- **model**（字符串）：设备 ID。
- **vendor**（字符串）：设备厂商。
- **serial**（字符串）：磁盘序列号。
- **wwn**（字符串）：唯一的存储 ID。
- **hctl**（字符串）：SCSI 的 Host:Channel:Target:Lun。
- **size**（整数）：设备的大小（以 GB 为单位）。

在本例中，使用磁盘的序列号指定根设备来部署 overcloud 镜像。

通过对每个节点的硬盘执行内省，检查磁盘信息。以下命令显示了一个节点的磁盘信息：

```
$ 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"
  }
]
```

在本例中，把根设备设置成磁盘 2（序列号为 **61866da04f380d001ea4e13c12e36ad6**）。这需要在节点定义中修改 **root_device** 参数：

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

这将帮助 director 识别特定的磁盘来用作根设备。当开始创建 overcloud 时，director 会部署这一节点，把 overcloud 镜像写入到这个磁盘。

注意

把每个节点的 BIOS 配置为包括从所选 root 磁盘引导。推荐的引导顺序是：网络引导，然后是 root 磁盘引导。

重要

不要使用 **name** 来指定 root 磁盘，因为这个值可能会在节点引导后改变。

5.5. 自定义 OVERCLOUD

undercloud 包含一组 Heat 模板，可充当 overcloud 的创建计划。您可以参照 overcloud 高级自定义指南，自定义 overcloud 的高级功能。

或者，您可以继续操作并部署基本的 overcloud。如需更多信息，请参阅???。

重要

基本 overcloud 将本地 LVM 存储用作块存储，这不是受支持的配置。建议您使用外部存储解决方案来设置块存储。

5.6. 使用 CLI 工具创建 OVERCLOUD

创建 OpenStack 环境的最后一个环节是运行 **openstack overcloud deploy** 命令进行创建。在运行此命令前，您应当已经熟悉关键的选项，以及如何加入自定义环境文件。以下小节探讨 **openstack overcloud deploy** 命令以及与它相关的选项。

> **警告**
>
> 不要以后台进程的形式运行 **`openstack overcloud deploy`**，因为这可能会造成在 overcloud 的创建过程中出现进程无法继续的问题。

设置 overcloud 参数

下表列出了 **`openstack overcloud deploy`** 命令的额外参数。

表 5.2. 部署参数

参数	描述
`--templates [TEMPLATES]`	包括在部署过程中使用的 Heat 模板的目录。如果为空，命令会使用位于 **`/usr/share/openstack-tripleo-heat-templates/`** 的默认模板。
`--stack STACK`	创建或更新的栈的名称
`-t [TIMEOUT]`，**`--timeout [TIMEOUT]`**	部署超时时间（分钟）
`--libvirt-type [LIBVIRT_TYPE]`	hypervisor 使用的虚拟类型
`--ntp-server [NTP_SERVER]`	用来同步时间的 NTP 服务器
`--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讯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。
`-e [EXTRA HEAT TEMPLATE]`，**`--extra-template [EXTRA HEAT TEMPLATE]`**	传递给 overcloud 部署的额外环境文件。此参数可以指定多次。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。

参数	描述
--environment-directory	需要在部署中包括的环境文件所在的目录。这个命令会使用数字顺序而不是字母顺序处理这些环境文件。
--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。
--dry-run	对 overcloud 进行验证检查，但不实际创建 overcloud。
--skip-postconfig	跳过 overcloud 部署后配置。
--force-postconfig	强制进行 overcloud 部署后配置。
--answers-file ANSWERS_FILE	到带有选项和参数的 YAML 文件的路径。
--rhel-reg	把 overcloud 节点注册到客户门户或 Satellite 6。
--reg-method	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。
--reg-org [REG_ORG]	用于注册的组织。
--reg-force	强制注册系统（即使已经注册过）。

参数	描述
--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，http://satellite.example.com，而不是 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。如果是 Red Hat Satellite 6 服务器，overcloud 会获取 **katello-ca-consumer-latest.noarch.rpm** 文件，使用 **subscription-manager** 进行注册，并安装 **katello-agent**。如果是 Red Hat Satellite 5 服务器，overcloud 会获取 **RHN-ORG-TRUSTED-SSL-CERT** 文件，并使用 **rhnreg_ks** 进行注册。
--reg-activation-key [REG_ACTIVATION_KEY]	用于注册的激活码。

某些命令行参数被弃用，以支持使用环境文件的 **parameter_defaults** 部分中包含的 Heat 模板参数。下表将被弃用的参数映射到 Heat 模板的等同参数。

表 5.3. 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 Cinder 节点数量	**BlockStorageCount**
--swift-storage-scale	扩展的 Swift 节点数量	**ObjectStorageCount**
--control-flavor	Controller 节点使用的 flavor	**overcloudControlFlavor**

参数	描述	Heat 模板参数
--compute-flavor	Compute 节点使用的 flavor	**overcloudComputeFlavor**
--ceph-storage-flavor	Ceph 节点使用的 flavor	**overcloudCephStorageFlavor**
--block-storage-flavor	Cinder 节点使用的 flavor	**overcloudBlockStorageFlavor**
--swift-storage-flavor	Swift 存储节点使用的 flavor	**overcloudSwiftStorageFlavor**
--neutron-flat-networks	定义在 neutron 插件中配置的平面网络。默认是 "datacentre"，允许外部网络创建	**NeutronFlatNetworks**
--neutron-physical-bridge	在每个虚拟机管理器上创建的 Open vSwitch 网桥。默认值是 "br-ex"，一般情况下不需要修改它	**HypervisorNeutronPhysicalBridge**
--neutron-bridge-mappings	要使用的逻辑网络到物理网桥的映射。默认情况是把主机上的外部网桥（br-ex）映射到一个物理名（datacentre）。您可以使用它作为默认的浮动网络	**NeutronBridgeMappings**
--neutron-public-interface	定义网络节点的 br-ex 中的网桥接口	**NeutronPublicInterface**
--neutron-network-type	neutron 的租户网络类型	**NeutronNetworkType**
--neutron-tunnel-types	Neutron 租户网络的隧道类型。使用逗号分隔的字符串可以指定多个值	**NeutronTunnelTypes**

参数	描述	Heat 模板参数
--neutron-tunnel-id-ranges	可以用来进行租户网络分配的 GRE 隧道 ID 的范围	**NeutronTunnelIdRanges**
--neutron-vni-ranges	可以用来进行租户网络分配的 VXLAN VNI ID 范围	**NeutronVniRanges**
--neutron-network-vlan-ranges	支持的 Neutron ML2 和 Open vSwitch VLAN 映射范围。默认是在 *datacentre* 物理网络中允许任何 VLAN	**NeutronNetworkVLANRanges**
--neutron-mechanism-drivers	neutron 租户网络的机制驱动。默认值是 "openvswitch"。使用逗号分隔的字符串可以指定多个值	**NeutronMechanismDrivers**
--neutron-disable-tunneling	禁用隧道功能以在 Neutron 中使用 VLAN 分段网络或平面网络	无参数映射。
--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

这些参数预定会从未来的 Red Hat OpenStack Platform 版本中移除。

注意

运行以下命令获得选项的完整列表：

```
$ openstack help overcloud deploy
```

5.7. 在 OVERCLOUD 创建中包括环境文件

使用 **-e** 来包含用于自定义 overcloud 的环境文件。您可以根据需要包含多个环境文件。但是，环境文件的顺序非常重要，后续环境文件中定义的参数和资源会覆盖前面环境文件中定义的相同参数和资源。下表是环境文件顺序的一个示例：

- 每个角色及其类型的节点数量。包含此信息对于创建 overcloud 至关重要。

- 任何网络隔离文件（包括 Heat 模板集合中的 **environments/network-isolation.yaml**），然后是您的自定义 NIC 配置文件。
- 任何外部的负载平衡环境文件。
- 任何存储环境文件，如 Ceph Storage、NFS、iSCSI 等。
- 任何用于 Red Hat CDN 或 Satellite 注册的环境文件。
- 任何其它自定义环境文件。

使用 **-e** 选项添加的所有环境文件都会成为 overcloud 栈定义的一部分。下例中的命令演示如何启动包含有自定义环境文件的 overcloud 创建过程：

```
$ 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 \
```

这个命令包括以下的额外选项：

- **--templates** - 使用 **/usr/share/openstack-tripleo-heat-templates** 中的 Heat 模板集合创建 overcloud。
- **-e ~/templates/node-info.yaml** - **-e** 选项为 overcloud 部署添加了额外的环境文件。在此处，它是用来定义每个角色使用多少个节点和多少种类型的自定义环境文件。例如：

```
parameter_defaults:
  overcloudControlFlavor: control
  overcloudComputeFlavor: compute
  overcloudCephStorageFlavor: ceph-storage
  ControllerCount: 3
  ComputeCount: 3
  CephStorageCount: 3
```

- **-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml** - **-e** 选项为 overcloud 部署添加了额外的环境文件。在此处，它是对网络隔离配置进行初始化的环境文件。
- **-e ~/templates/network-environment.yaml** - **-e** 选项添加额外的环境文件到 overcloud 部署中。在本例中，它是包含网络隔离自定义的网络环境文件。
- **-e ~/templates/storage-environment.yaml** - **-e** 选项为 overcloud 部署添加了额外的环境文件。在此处，它是用来初始化存储配置的自定义环境文件。
- **--ntp-server pool.ntp.org** - 使用一个 NTP 服务器进行时间同步。这可以保持 Controller 节点集群的同步。

director 需要这些环境文件来进行重新部署并使用部署后的功能（请参阅 ???）。没有正确包含这些文件可能会破坏您的 overcloud。

如果计划在以后修改 overcloud 配置，您应该：

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 的配置，因为在使用 director 对 overcloud 栈进行更新时，这种手动配置会被 director 的配置覆盖。

包含环境文件目录

您可以使用 **--environment-directory** 选项添加包含环境文件的完整目录。部署命令按照数字的顺序而不是字母顺序处理这个目录中的环境文件。因此，在使用这个方法时，推荐在文件名中包括一个数字前缀以排列其处理顺序。例如：

```
$ ls -1 ~/templates
00-node-info.yaml
10-network-isolation.yaml
20-network-environment.yaml
30-storage-environment.yaml
40-rhel-registration.yaml
```

运行以下部署命令，以包含目录：

```
$ openstack overcloud deploy --templates --environment-directory
~/templates
```

使用一个应答文件

应答文件是 YAML 格式的文件，用于简化模板和环境文件的包含方式。应答文件使用以下参数：

templates

要使用的 Heat 核心模板集。它用于替代 **--templates** 命令行选项。

environments

要包含的环境文件列表。它用于替代 **--environment-file** (**-e**) 命令行选项。

例如，应答文件可能包含以下内容：

```
templates: /usr/share/openstack-tripleo-heat-templates/
environments:
  - ~/templates/00-node-info.yaml
  - ~/templates/10-network-isolation.yaml
  - ~/templates/20-network-environment.yaml
  - ~/templates/30-storage-environment.yaml
  - ~/templates/40-rhel-registration.yaml
```

运行下列部署命令，以包含应答文件：

```
$ openstack overcloud deploy --answers-file ~/answers.yaml
```

5.8. 管理 OVERCLOUD 计划

作为 **openstack overcloud deploy** 命令的一种替代，director 也可以管理导入的计划。

如要创建新计划，请以 **stack** 用户的身份运行以下命令：

```
$ openstack overcloud plan create --templates /usr/share/openstack-
tripleo-heat-templates my-overcloud
```

这会从 **/usr/share/openstack-tripleo-heat-templates** 的 Heat 核心模板集创建一个计划。director 根据您的输入为计划命名。在本例中，名称即是 **my-overcloud**。director 将这个名称用作对象存储容器、工作流环境以及 overcloud 堆栈名称的标签。

使用以下命令，从环境文件添加参数：

```
$ openstack overcloud parameters set my-overcloud ~/templates/my-
environment.yaml
```

使用以下命令，部署您的计划：

```
$ openstack overcloud plan deploy my-overcloud
```

使用以下命令，删除现有的计划：

```
$ openstack overcloud plan delete my-overcloud
```

注意

基本上，**openstack overcloud deploy** 命令使用所有这些命令来移除现有的计划，上传包含环境文件的新计划并部署该计划。

5.9. 验证 OVERCLOUD 模板和计划

创建 overcloud 或更新堆栈之前，先验证您的 Heat 模板和环境文件，确认是否存在任何错误。

创建一个呈现的模板

overcloud 的 Heat 核心模板采用 Jinja2 格式。要验证您的模板，请使用以下命令呈现未采用 Jinja2 格式的版本：

```
$ openstack overcloud plan create --templates /usr/share/openstack-
tripleo-heat-templates overcloud-validation
$ mkdir ~/overcloud-validation
$ cd ~/overcloud-validation
$ swift download overcloud-validation
```

使用 **~/overcloud-validation** 中的呈现模板执行后续的验证测试。

验证模板语法

使用下列命令来验证模板语法：

```
$ openstack orchestration template validate --show-nested --template
~/overcloud-validation/overcloud.yaml -e ~/overcloud-
validation/overcloud-resource-registry-puppet.yaml -e [ENVIRONMENT
FILE] -e [ENVIRONMENT FILE]
```

注意

验证过程需要 **overcloud-resource-registry-puppet.yaml** 环境文件包括针对于 overcloud 的资源。使用 **-e** 选项为这个命令添加其他额外的环境文件。此外，还需包含 **--show-nested** 选项，以解析来自嵌套模板的参数。

此命令可以确认模板中的任何语法错误。如果模板语法验证成功，输出会显示生成的 overcloud 模板的预览。

5.10. 监控 OVERCLOUD 的创建过程

overcloud 创建过程开始，director 部署您的节点。此过程需要一些时间完成。要查看 overcloud 创建的状态，请在另外一个终端中以 **stack** 用户身份运行：

```
$ source ~/stackrc                # Initializes the stack user to use
the CLI commands
$ heat stack-list --show-nested
```

heat stack-list --show-nested 命令会显示 overcloud 创建过程的当前状态。

5.11. 访问 OVERCLOUD

director 会生成脚本来配置和帮助认证 director 主机与 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：

```
$ source ~/overcloudrc
```

这会加载从 director 主机的 CLI 访问 overcloud 所需的环境变量。若要返回到与 director 主机交互，请运行以下命令：

```
$ source ~/stackrc
```

overcloud 中的每个节点还会包括一个名为 **heat-admin** 的用户。**stack** 用户有到每个节点上的此用户的 SSH 访问权限。要通过 SSH 访问某一节点，可查找相关节点的 IP 地址：

```
$ nova list
```

使用 **heat-admin** 用户和节点的 IP 地址连接到节点：

```
$ ssh heat-admin@192.168.24.23
```

5.12. 完成 OVERCLOUD 的创建

使用命令行工具创建 overcloud 的步骤到此结束。有关创建后的功能，请参阅???。

第 6 章 使用 WEB UI 配置基本 OVERCLOUD

本章介绍使用 Web UI 配置 OpenStack Platform 环境的基本步骤。overcloud 基本配置中不含任何自定义功能。但是，您可以参考 overcloud 高级自定义指南中的说明，添加高级配置选项到这一基本 overcloud 中并按照您的具体规格进行自定义。

在本章使用的示例中，所有节点都是使用 IPMI 作为电源管理的裸机。如需了解关于其它支持的电源管理类型和它们的选项，请参阅 附录 B, *电源管理驱动*。

流程

1. 使用节点定义模板并通过手动注册来注册空节点。
2. 检查所有节点的硬件。
3. 把一个 overcloud 计划上传到 director。
4. 将节点分配到角色。

配置要求

- ???中安装的 director 节点已启用 UI
- 一组作为节点的裸机。所需的节点数量由您需要创建的 overcloud 类型所决定（请参阅 第 3.1 节 “规划节点的实施角色”）。这些机器需要满足每个节点类型对系统的要求。相关信息，请参阅 ???。这些节点不需要操作系统，director 会把一个 Red Hat Enterprise Linux 7 镜像复制到每个节点。
- Provisioning 网络的一个网络连接，需配置为原生 VLAN。所有节点都必须连接此网络，并且满足第 2.3 节 “网络要求”中规定的要求。
- 所有其他网络类型使用 Provisioning 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。

6.1. 访问 WEB UI

UI 以一个独立应用的形式在浏览器中运行。不过，您的客户端系统需要访问 Public API 端点来使用 undercloud 上的以下组件：

组件	**UI** 用途
OpenStack Identity（**`keystone`**）	用于进行 UI 认证和其他服务的端点发现。
OpenStack Orchestration（**`heat`**）	查看部署的状态。
OpenStack Bare Metal（**`ironic`**）	用于控制节点。

组件	UI 用途
OpenStack Object Storage（**swift**）	用于存储供 overcloud 创建使用的 Heat 模板集合或计划。
OpenStack Workflow（**mistral**）	访问和执行 director 任务。
OpenStack Messaging（**zaqar**）	基于 Websocket 的服务，可用于查找特定任务的状态。

UI 直接与这些 Public API 交互，所以您的客户端系统需要能够访问其端点。

重要

若要通过 Mozilla Firefox 访问 director 的 Web UI，需要针对这些 OpenStack Platform Public API 进行服务器身份例外处理。有关这些例外处理的信息，请参阅附录 E, *通过 Firefox 访问 UI 时的服务器例外*。

用户通过 SSL 访问 director 的 Web UI。例如，如果 undercloud 的 IP 地址是 192.168.24.1，则用于访问 UI 的地址是 **https://192.168.24.1**。Web UI 首先将显示一个登录屏幕，屏幕中含有以下字段：

- **Username** - director 的管理员用户。默认值为 **admin**。
- **Password** - 管理员用户的密码。在 undercloud 主机终端中以 **stack** 用户身份运行 **sudo hiera admin_password** 命令，可查找此密码。

登录 UI 后，UI 将访问 OpenStack Identity Public API 并获取其他 Public API 服务的端点。不过，如果您计划更改端点或使用不同的 IP 来访问端点，director UI 可读取 **/var/www/openstack-tripleo-ui/dist/tripleo_ui_config.js** 文件中的设置。此文件使用以下参数：

参数	描述
keystone	OpenStack Identity (**keystone**) 服务的 Public API。UI 将自动通过此服务发现其他服务的端点；这意味着，您只需要定义此参数。不过，您可以根据需要为其他端点定义自定义 URL。
heat	OpenStack Orchestration (**heat**) 服务的 Public API。其 URL 需要使用以下格式：**https://<IP>:13004//v1/<tenant_id>**，其中 **tenant_id** 是 undercloud 的 **admin** 租户。

参数	描述
ironic	OpenStack Bare Metal (**ironic**) 服务的 Public API。
swift	OpenStack Object Storage (**swift**) 服务的 Public API。其 URL 需要使用以下格式：**https://<IP>:8080/v1/AUTH_<tenant_id>** ，其中 **tenant_id** 是 undercloud 的 **admin** 租户。
mistral	OpenStack Workflow（**mistral**）服务的 Public API。
zaqar-websocket	OpenStack Messaging（**zaqar**）服务的 Websocket。
zaqar_default_queue	用于 OpenStack Messaging（**zaqar**）服务的消息传递队列。默认为 **tripleo**。

下方为 **tripleo_ui_config.js** 文件的示例：

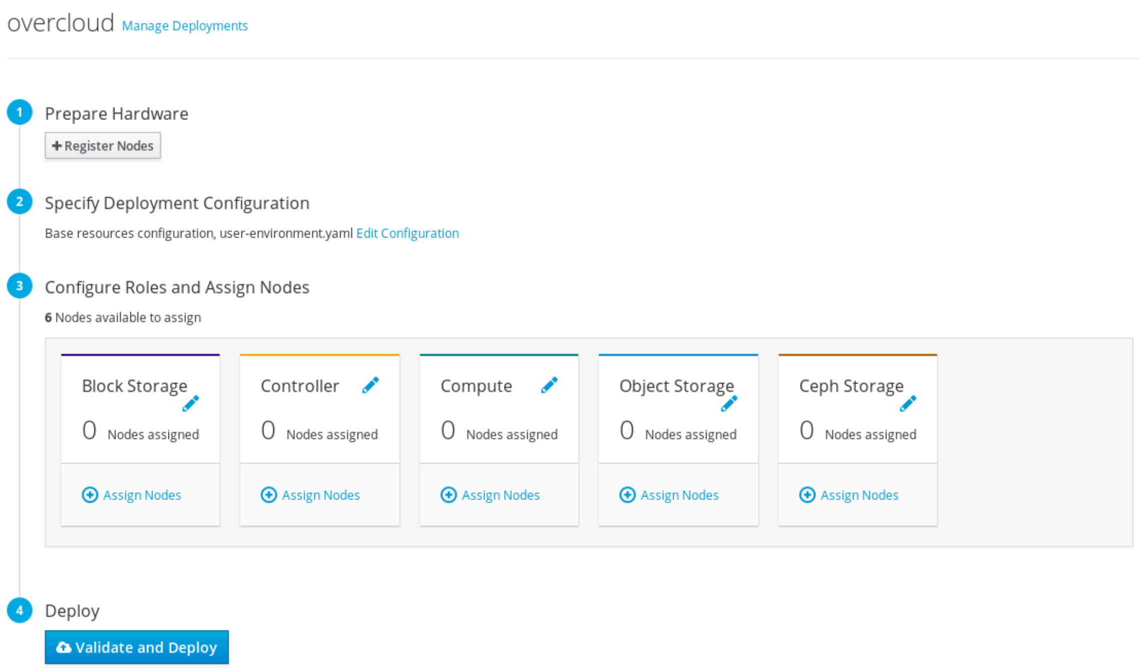
```
window.tripleOUiConfig = {
  "keystone": "https://192.168.24.2:13000/v2.0",
  "heat":
"https://192.168.24.2:13004/v1/f30d815864be456dbc2a00f4e9200e9e",
  "ironic": "https://192.168.24.2:13385",
  "mistral": "https://192.168.24.2:13989/v2",
  "swift":
"https://192.168.24.2:13808/v1/AUTH_f30d815864be456dbc2a00f4e9200e9e",
  "zaqar-websocket": "wss://192.168.24.2:9000",
  "zaqar_default_queue": "tripleo"
};
```

6.2. 浏览 WEB UI

UI 包含三大区域：

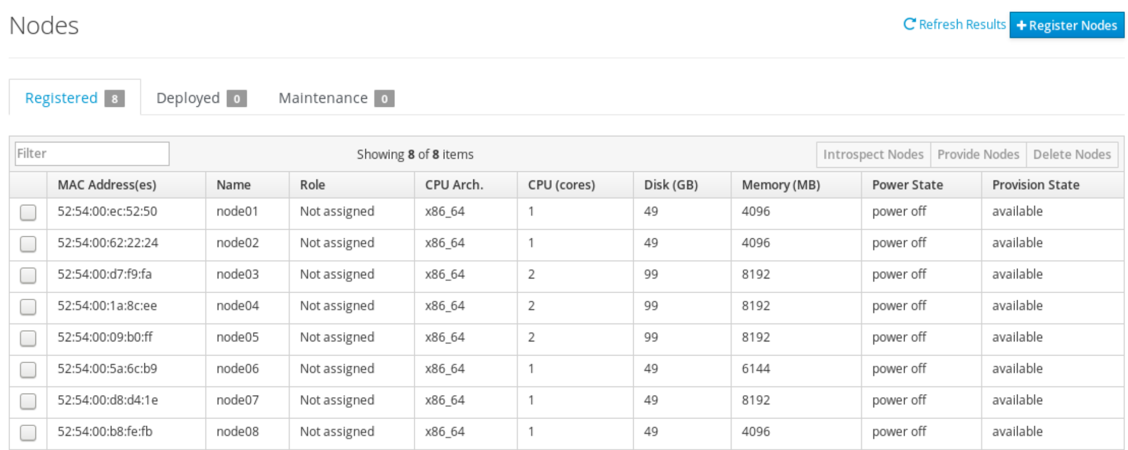
Deployment Plan

位于 UI 顶部的菜单项。此页面充当主 UI 区域，您可以在其中定义 overcloud 创建计划、分配至各个角色的节点和当前 overcloud 的状态。此区域也提供一个部署工作流，逐步引导您完成 overcloud 创建过程的每个步骤，如设置部署参数和分配节点到角色等。



Nodes

位于 UI 顶部的菜单项。此页面充当节点配置区域，提供用于注册新节点和内省已注册节点的各种方式。此区域还定义可部署的节点、当前部署的节点，以及维护中的节点。



Validations

位于页面右侧的面板。此区域提供一组部署前和部署后系统检查，它们可确保 overcloud 创建过程成功完成。这些验证任务在部署期间的特定时点上自动运行，但您也可以手动执行它们。若要运行某一项验证任务，可单击对应的 **Play** 按钮。单击各项验证任务的标题即可运行该任务，或者单击验证标题来查看更多相关信息。

6.3. 在 WEB UI 中导入 OVERCLOUD 计划

director UI 需要先加载计划，才能配置 overcloud。此计划通常是一个 Heat 模板集，比如 undercloud 上的 **`/usr/share/openstack-tripleo-heat-templates`**。此外，您也可以根据自己的硬件和环境要求自定义计划。有关自定义 overcloud 的更多信息，请参阅 overcloud 高级自定义指南。

计划中包含配置 overcloud 的四个主要步骤：

1. **准备硬件** - 节点注册和内省。
2. **指定部署配置** - 配置 overcloud 参数，定义要包含的环境文件。
3. **配置角色和分配节点** - 分配节点到角色，修改角色相关参数。
4. **部署** - 启动 overcloud 创建过程。

undercloud 的创建和配置过程会自动上传计划。您也可以在 Web UI 中导入多个计划。单击 **Deployment Plan** 屏幕中的 **Manage Deployments**。这将显示当前的 **Plans** 表。

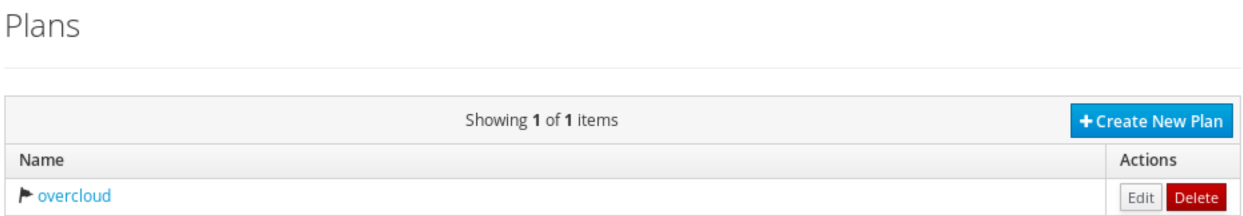
单击 **Create New Plan**；这时会显示一个窗口，要求您提供以下信息：

- **Plan Name** - 计划的纯文本名称。例如，**`overcloud`**。
- **Upload Type** - 选择要上传 **Tar Archive (tar.gz)** 或整个 **Local Folder**（仅限 Google Chrome）。
- **Plan Files** - 单击浏览按钮来选择本地文件系统中的计划。

如果需要复制 director 的 Heat 模板集到客户端机器，可以存档文件再复制它们：

```
$ cd /usr/share/openstack-tripleo-heat-templates/
$ tar -cf ~/overcloud.tar *
$ scp ~/overcloud.tar user@10.0.0.55:~/.
```

一旦 director UI 上传了计划，该计划就会显示在 **Plans** 表中；此时您可以进行配置。单击 **Deployment Plan**。

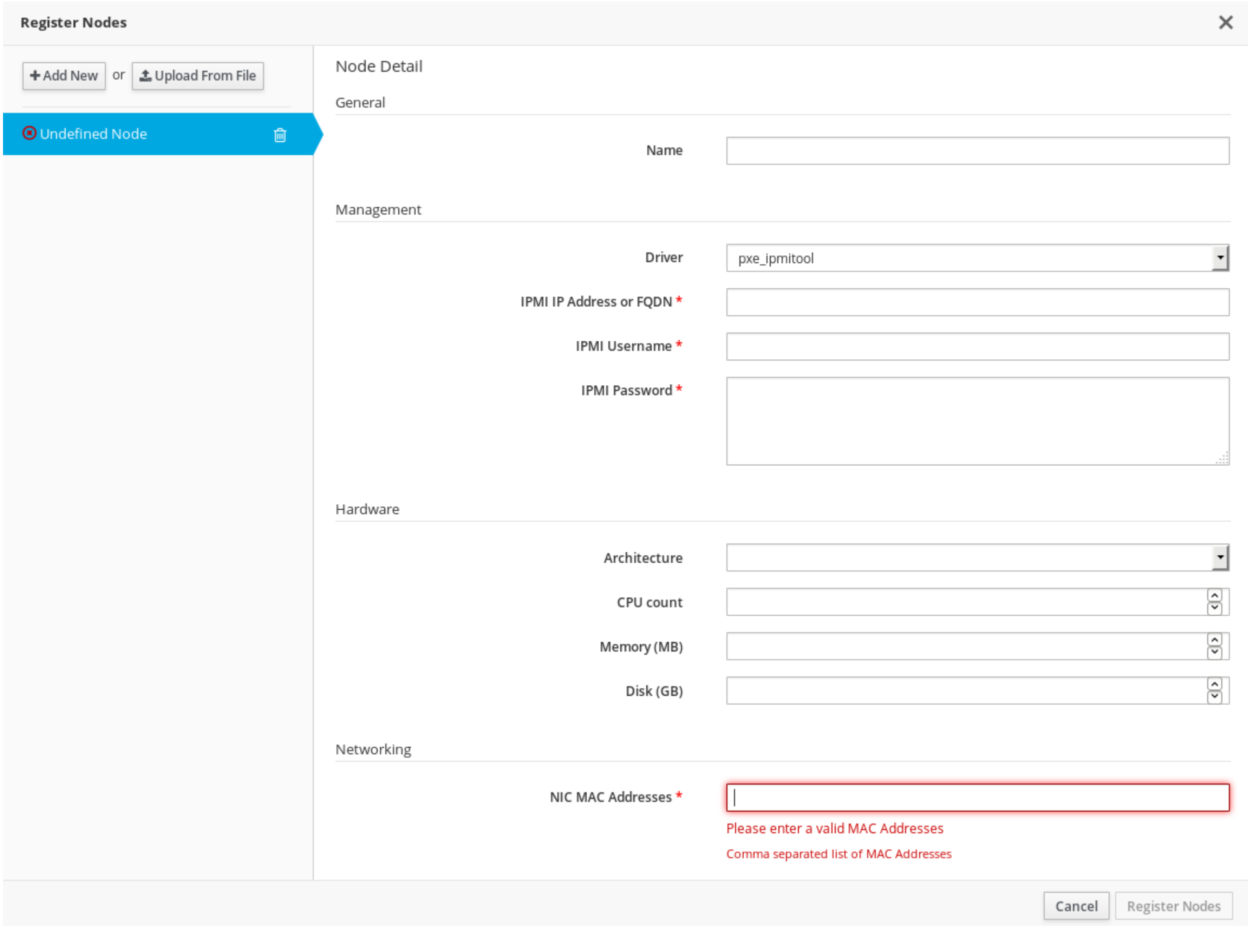


6.4. 在 WEB UI 中注册节点

配置 overcloud 节点的第一步是注册您的节点。您可以通过以下方式之一来开始注册节点：

- 在 **Deployment Plan** 屏幕中，单击 **1 Prepare Hardware** 下的 **Register Nodes**。
- 单击 **Nodes** 屏幕上的 **Register Nodes**。

这将显示 **Register Nodes** 窗口。



Director 需要一份要注册的节点列表，您可以通过以下两种方式之一来提供：

1. **上传节点定义模板** - 单击 **Upload from File** 按钮，再选择一个文件。如需节点定义模板的语法，请参阅???。
2. **手动注册各个节点** - 单击 **Add New**，然后提供节点的一系列详细信息。

如下为手动注册时要提供的详细信息：

名称

节点的纯文本名称。仅可使用 *RFC3986* 非保留字符。

Driver

使用的电源管理驱动。在这个示例中使用 IPMI 驱动器（**`pxe_ipmitool`**），但其他驱动器也可用。请参阅附录 B, *电源管理驱动*了解可用的驱动。

IPMI IP 地址

IPMI 设备的 IP 地址。

IPMI Username; IPMI Password

IPMI 的用户名和密码。

Architecture

系统架构。（可选）

CPU count

节点上的 CPU 数量。（可选）

Memory (MB)

以 MB 为单位的内存大小。（可选）

Disk (GB)

以 GB 为单位的硬盘的大小。（可选）

NIC MAC Addresses

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。

注意

UI 也支持通过使用 **pxe_ssh** 驱动程序从 KVM 主机注册节点。请注意，此选项仅可用于测试和评估目的，不建议将它用于 Red Hat OpenStack Platform 企业环境。如需更多信息，请参阅第 B.6 节 “SSH 和 Virsh”。

在输入节点信息后，请单击窗口底部的 **Register Nodes**。

Director 将注册节点。完成时，您可以使用 UI 来执行节点内省。

6.5. 在 WEB UI 中检查节点硬件

director 可以在每个节点上运行内省过程。此过程会使每个节点通过 PXE 引导一个内省代理。此代理从节点上收集硬件数据，并把信息发送回 director，director 把这些信息存储到运行于 director 上的 OpenStack Object Storage（swift）服务中。director 将这些硬件信息用于不同的目的，如进行配置文件标记、基准测试和手动分配根磁盘等。

注意

您也可以通过创建策略文件，在内省后立即将节点标记到配置文件中。如需有关创建策略文件并将它们纳入到内省过程中的信息，请参阅附录 D, *自动配置集标记*。另外，您还可以通过 UI 将节点标记到配置文件中。如需关于手动标记节点的详细信息，请参阅第 6.7 节 “在 Web UI 中将节点标记到角色”。

若要启动内省过程：

1. 前往 **Nodes** 屏幕。
2. 选择您想要内省的所有节点。
3. 单击 **Introspect Nodes**。

重要

确保这个过程成功完成。它可能需要 15 分钟来检查这些裸机节点。

内省过程完成时，选择 **Provision State** 为 **`manageable`** 的所有节点，再单击 **Provide Nodes** 按钮。等待 **Provision State** 更改为 **`available`**。

节点现在已做好部署准备。

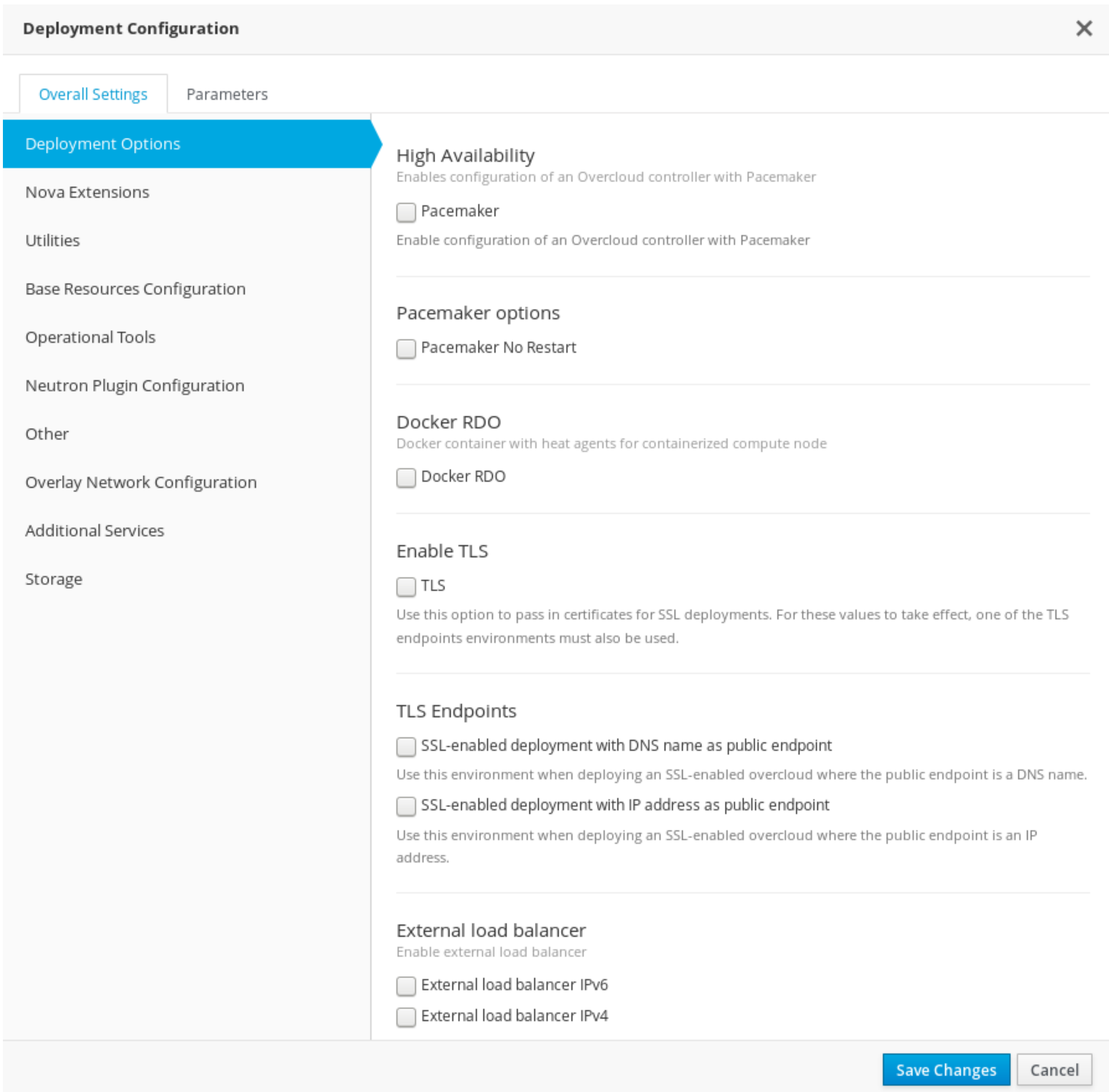
6.6. 在 WEB UI 中编辑 OVERCLOUD 计划参数

Deployment Plan 屏幕中提供了一种方式，供您自定义上传的计划。在 **2 Specify Deployment Configuration** 下，单击 **Edit Configuration** 连接，来修改您的基本 overcloud 配置。

这时将出现一个窗口，其包含两个主要选项卡：

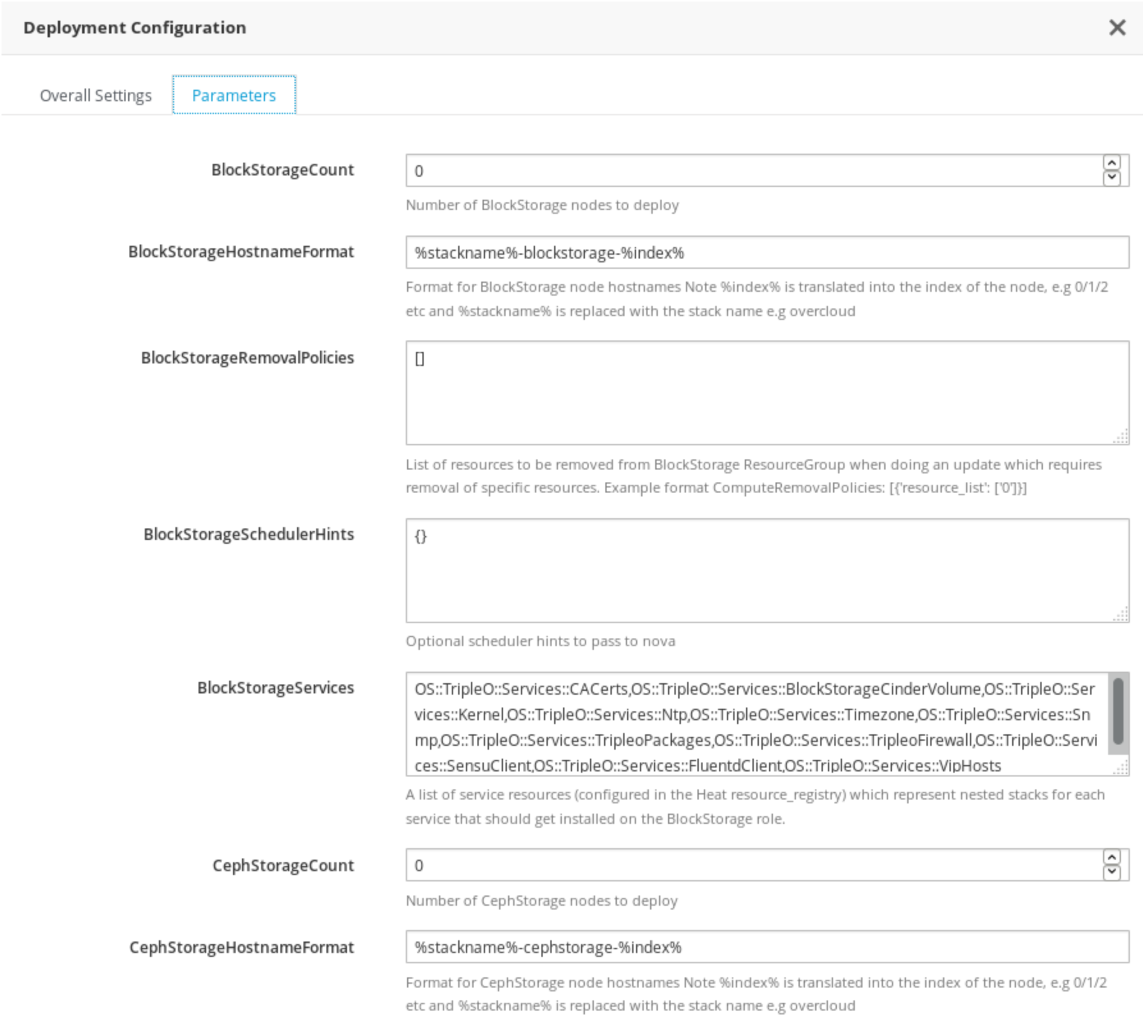
Overall Settings

此选项卡提供了包括 overcloud 中不同功能的方式。这些功能在计划的 **`capabilities-map.yaml`** 文件中定义，每项功能各自使用一个不同的环境文件。例如，在 **Storage** 下，您可以选择 **Storage Environment**，其计划映射到 **`environments/storage-environment.yaml`** 文件，允许您配置 overcloud 的 NFS、iSCSI 或 Ceph 设置。**Other** 选项卡包含在计划中检测到但未在 **`capabilities-map.yaml`** 中列出的环境文件，对于添加计划中包含的自定义环境文件，这非常有用。一旦选定要包含的功能，请单击 **Save**。



Parameters

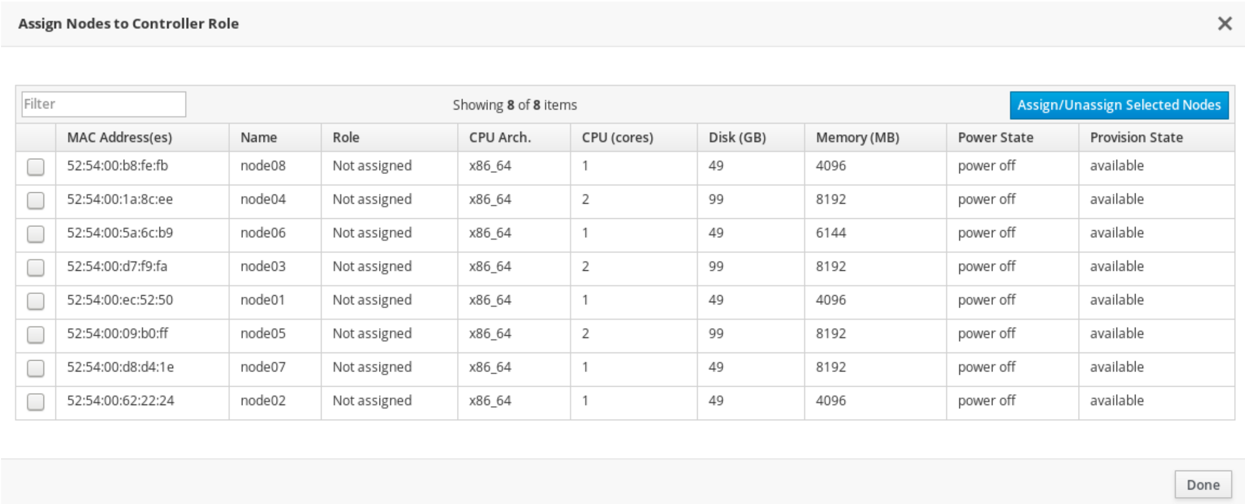
这包括 overcloud 的各种基础级参数和环境文件参数。例如，您可以在此区域中更改每个节点的节点计数。如果您计划使用 3 个控制器节点，可将 **`ControllerCount`** 更改为 3。修改了基础级参数后，请单击 **Save**。



6.7. 在 WEB UI 中将节点标记到角色

注册并检查了各个节点的硬件后，您要将它们标记到特定的配置文件中。这些配置文件将您的节点与特定的类型和部署角色匹配。

要将节点分配到角色，请滚动到 **Deployment Plan** 屏幕的 **3 Configure Roles and Assign Nodes** 区域。单击所选角色的 **Assign Nodes**。这时会出现一个窗口，从中您可以选择要分配到该角色的节点。选择了角色的节点后，请单击 **Assign/Unassign Selected Nodes**。



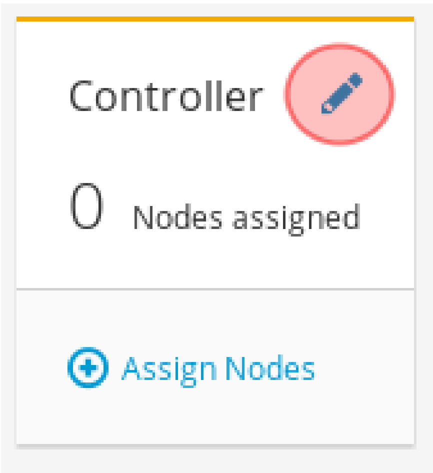
	MAC Address(es)	Name	Role	CPU Arch.	CPU (cores)	Disk (GB)	Memory (MB)	Power State	Provision State
	52:54:00:b8:fe:fb	node08	Not assigned	x86_64	1	49	4096	power off	available
	52:54:00:1a:8c:ee	node04	Not assigned	x86_64	2	99	8192	power off	available
	52:54:00:5a:6c:b9	node06	Not assigned	x86_64	1	49	6144	power off	available
	52:54:00:d7:f9:fa	node03	Not assigned	x86_64	2	99	8192	power off	available
	52:54:00:ec:52:50	node01	Not assigned	x86_64	1	49	4096	power off	available
	52:54:00:09:b0:ff	node05	Not assigned	x86_64	2	99	8192	power off	available
	52:54:00:d8:d4:1e	node07	Not assigned	x86_64	1	49	8192	power off	available
	52:54:00:62:22:24	node02	Not assigned	x86_64	1	49	4096	power off	available

当这些节点分配到角色时，请单击 **Done** 以返回到 **Deployment Plan** 屏幕。

为您要在 overcloud 中包含的每个角色完成此任务。

6.8. 在 WEB UI 中编辑节点

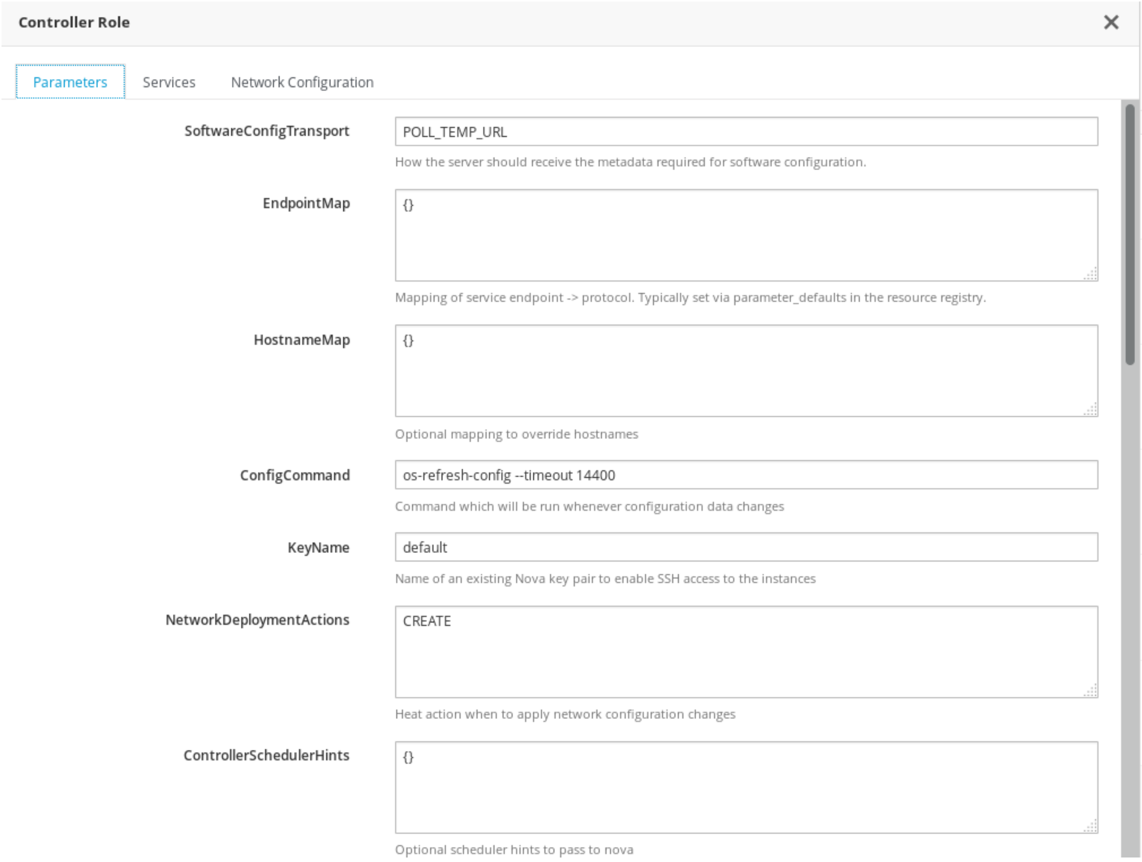
每个节点角色都提供配置角色相关参数的途径。滚动到 **Deployment Plan** 屏幕的 **3 Configure Roles and Assign Nodes** 区域。单击角色名称旁边的 **Edit Role Parameters** 图标（铅笔图标）。



此时将出现一个窗口，其中显示了两个主要选项卡：

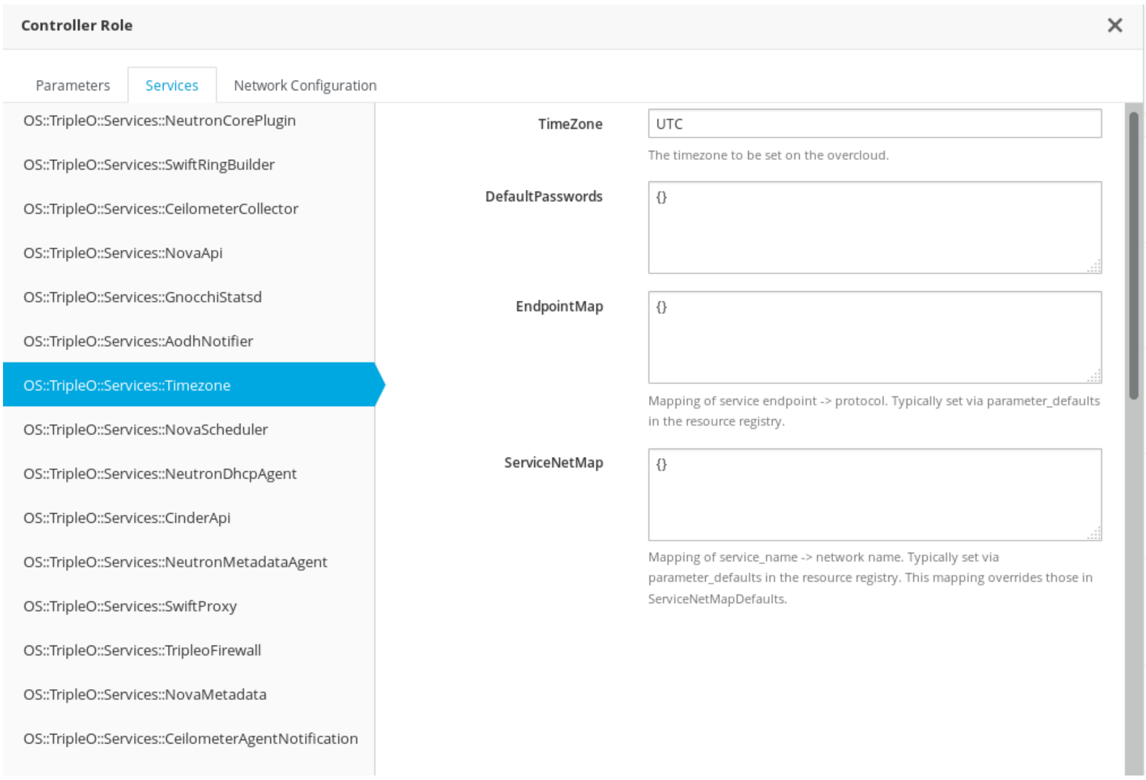
Parameters

> 此选项卡中含有各种角色相关的参数。例如，如果您在编辑控制器节点，可以使用 **`overcloudControlFlavor`** 参数更改角色的默认类型。修改了角色相关参数后，请单击 **Save Changes**。



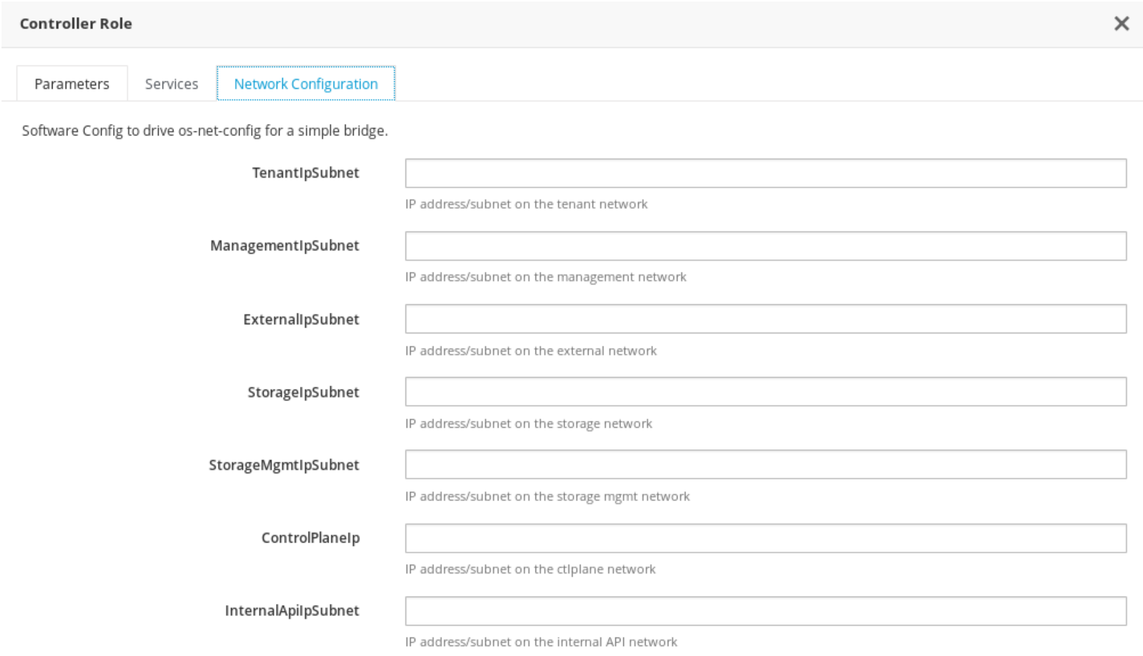
Services

此选项卡可定义所选角色的服务相关参数。左侧面板中显示您可选择并修改的服务列表。例如，若要更改时区，可单击 `OS::TripleO:Services:Timezone` 服务，再根据您想要的时区更改 `TimeZone` 参数。修改了服务相关参数后，请单击 **Save Changes**。



Network Configuration

此选项卡可用于定义 overcloud 中各种网络的 IP 地址或子网范围。



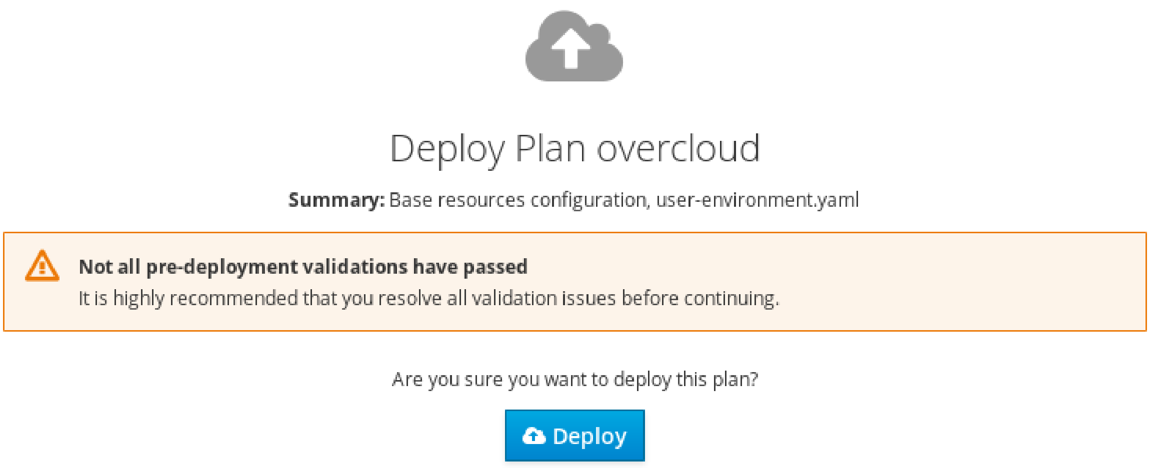
重要

尽管角色的服务参数显示在 UI 中，但一部分服务可能默认为禁用状态。您可以按照 ??? 中的说明启用它们。也可参阅 overcloud 高级自定义指南的*Composable Roles*部分，了解启用这些服务的相关信息。

6.9. 在 WEB UI 中启动 OVERCLOUD 创建过程

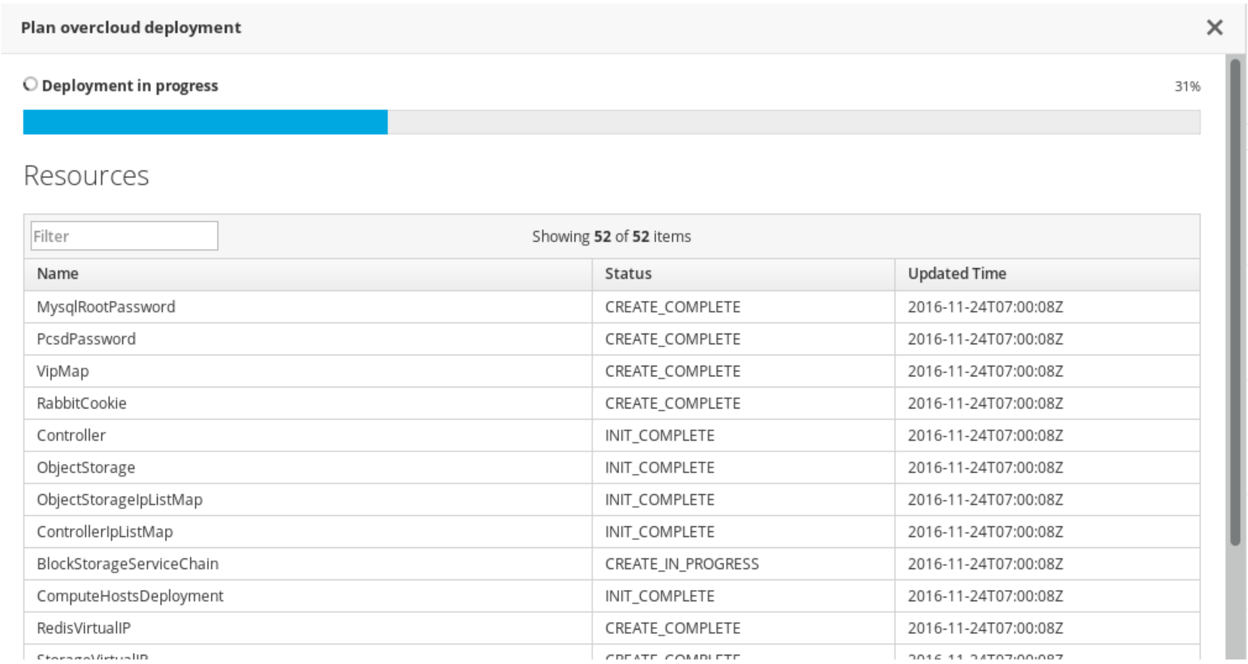
配置了 overcloud 计划后，您可以开始部署 overcloud。具体操作包括滚动到 **4 Deploy** 区域，并单击 **Validate and Deploy**。

如果没有运行 undercloud 验证或未通过验证，屏幕中会显示警告消息。请确保您的 undercloud 主机满足相关的要求，然后再进行部署。



准备好部署时，请单击 **Deploy**。

UI 会定期监控 overcloud 的创建进度，并通过进度条来指示当前的进度百分比。单击**View detailed information** 链接，可显示 overcloud 中当前 OpenStack Orchestration 栈的日志。

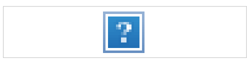


等待 overcloud 部署完成。

在 overcloud 创建过程完成后，**4 Deploy** 区域显示当前的 overcloud 状态和以下详细信息：

- **IP address** - 用于访问 overcloud 的 IP 地址。
- **Password** - ondercloud 上 OpenStack **`admin`** 用户的密码。

使用以上信息来访问您的 overcloud。



6.10. 完成 OVERCLOUD 创建

通过 director UI 创建 overcloud 的步骤到此结束。如需了解创建后功能，请参阅???。

第 7 章 使用预配置节点配置基本 OVERCLOUD

本章介绍了使用预配置节点来配置 OpenStack Platform 环境的基本步骤。这种配置情境与标准的 overcloud 创建场景之间存在多项差异：

- 您可以使用外部工具配置节点，让 director 仅控制 overcloud 的配置。
- 您无需依赖 director 的配置方法，可直接使用节点。在不实施电源管理控制的情况下创建 overcloud，或者使用具有 DHCP/PXE 引导限制的网络时，这一点非常有用。
- director 不使用 OpenStack Compute (nova)、OpenStack Bare Metal (ironic) 或者 OpenStack Image (glance) 来管理节点。
- 预配置节点使用自定义分区布局。

本情境提供基本配置，不包含任何自定义功能。但是，您可以参考 overcloud 高级自定义指南中的说明，添加高级配置选项到这一基本 overcloud 中并按照您的具体规格进行自定义。

重要

不支持在 overcloud 中混用预配置节点和 director 配置节点。

配置要求

- ??? 中创建的 director 节点。
- 一组作为节点的裸机。所需的节点数量由您需要创建的 overcloud 类型所决定（请参阅第 3.1 节 “规划节点的实施角色”了解与overcloud 角色有关的信息）。这些机器还需要满足每个节点类型的要求。相关信息，请参阅???。这些节点需要采用 Red Hat Enterprise Linux 7.3 操作系统。
- 用于管理预配置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。这种网络通常是预配置节点至 director 的 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 7.1. Provisioning 网络 IP 分配信息

节点名	IP 地址
Director	192.168.24.1
Controller	192.168.24.2
Compute	192.168.24.3

- 使用单独的网络。当 director 的 Provisioning 网络是一种专用的不可路由的网络时，您可以从任何子网定义节点的 IP 地址，通过公共 API 端点与 director 通讯。本情境有一些需要注意的地方，详情请参阅???章节。

- 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。

7.1. 创建配置节点的用户

在本流程稍后的阶段，director 需要具备以 **stack** 用户的身份访问 overcloud 节点的 SSH 权限。

1. 在每个 overcloud 节点上，创建名为 **stack** 的用户，并为每个节点设置密码。例如，在 Controller 节点上运行以下命令：

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

3. 在所有预配置节点上创建和配置 **stack** 用户之后，将 **stack** 用户的公共 SSH 密钥从 director 节点复制到每个 overcloud 节点。例如，将 director 的公共 SSH 密钥复制到 Controller 节点：

```
[stack@director ~]$ ssh-copy-id stack@192.168.24.2
```

7.2. 为节点注册操作系统

每个节点需要具备访问红帽订阅的权限。以下步骤显示了如何将每个节点注册到 Red Hat Content Delivery Network。对每个节点执行以下操作：

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：

```
[root@controller ~]# sudo subscription-manager register
```

2. 找到 Red Hat OpenStack Platform 11 所在的权利池。

```
[root@controller ~]# sudo subscription-manager list --available -
-all
```

3. 使用上一步中获得的池 ID 添加 Red Hat OpenStack Platform 11 权力：

```
[root@controller ~]# sudo subscription-manager attach --
pool=pool_id
```

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

```
[root@controller ~]# sudo subscription-manager repos --disable=*
```

```
[root@controller ~]# sudo subscription-manager repos --
enable=rhel-7-server-rpms --enable=rhel-7-server-extras-rpms --
enable=rhel-7-server-rh-common-rpms --enable=rhel-ha-for-rhel-7-
server-rpms --enable=rhel-7-server-openstack-11-rpms --
enable=rhel-7-server-rhceph-2-osd-rpms --enable=rhel-7-server-
rhceph-2-mon-rpms --enable=rhel-7-server-rhceph-2-tools-rpms
```

重要

仅启用 第 2.5 节 “软件仓库的要求”中列出的存储库。其他存储库都可能会造成软件包和软件冲突，请勿启用任何其他存储库。

5. 更新您的系统，确保您具备最新的基础系统软件包：

```
[root@controller ~]# sudo yum update -y
[root@controller ~]# sudo reboot
```

节点现在可供您的 overcloud 使用。

7.3. 在节点上安装用户代理

各预配置节点使用 OpenStack Orchestration (heat) 代理与 director 通讯。各节点上的代理对 director 轮询，获得针对各节点定制的元数据。该元数据允许代理配置各节点。

在各节点上为编配代理安装初始软件包：

```
[root@controller ~]# sudo yum -y install python-heat-agent*
```

7.4. 为 DIRECTOR 配置 SSL/TLS 访问权限

如果 director 使用 SSL/TLS，那么预配置节点需要用于签署 director 的 SSL/TLS 证书的证书授权机构文件。如果使用您自己的证书授权机构，则在各 overcloud 节点上执行以下操作：

1. 将证书授权机构文件复制到各预配置节点上的 **/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 ~]#  sudo update-ca-trust extract
```

这可以确保 overcloud 节点能够通过 SSL/TLS 访问 director 的 Public API。

7.5. 为 CONTROL PLANE 配置网络

预配置 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，这是使用 **undercloud.conf** 文件中的 **network_cidr** 参数定义的子网。该节点需要直接访问此子网，或者需要通过路由访问此子网。

- director 的 Public API 端点，由 **undercloud.conf** 文件中的 **undercloud_public_host** 参数指定。当您无法通过 L3 路由至 Control Plane 时，或者在轮询 director 的元数据期间计划使用 SSL/TLS 通信时，此选项都可用。请参阅 ???了解配置 overcloud 节点以使用 Public API 端点的更多步骤。

director 使用 Control Plane 网络来管理和配置标准 overcloud。对于具备预配置节点的 overcloud，您可能需要对网络配置做一些修改，以适应 director 与预配置节点通讯的方式。

使用网络隔离

网络隔离允许您对服务分组，以使用特定的网络，包括 Control Plane。请参阅 overcloud 高级自定义指南，了解多种网络隔离策略。此外，您也可以为 control plane 上的节点指定特定的 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，请参考 *overcloud 高级自定义*指南中的以下章节：

- "Isolating Networks"
- "Controlling Node Placement"

> **注意**
>
> 使用网络隔离时，确保您的 NIC 模板不包含用于访问 undercloud 的 NIC。这些模板可能会重新配置 NIC，从而导致部署期间出现连接和配置问题。

分配 IP 地址

如果不使用网络隔离，您可以使用单个 Control Plane 网络来管理所有服务。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，确保选定的 overcloud IP 地址不在 DHCP 所提供的置备范围（**dhcp_start** 和 **dhcp_end**）和内省范围（**inspection_iprange**）之内。

创建标准 overcloud 期间，director 创建 OpenStack Networking (neutron) 端口，用于自动为 Provisioning / Control Plane 网络池上的 overcloud 节点分配 IP 地址。但是，这可能导致 director 为手动配置的节点分配不同的 IP 地址。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 上的预配置 IP 分配。

可预测 IP 策略的一个示例是使用 IP 地址分配如下的环境文件 (**ctlplane-assignments.yaml**)：

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-
tripleo-heat-templates/deployed-server/deployed-neutron-port.yaml

parameter_defaults:
  DeployedServerPortMap:
    controller-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.2
      subnets:
        - cidr: 24
    compute-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 24
```

在本示例中，**OS::TripleO::DeployedServer::ControlPlanePort** 资源将一组参数传递至 director，并为预配置的节点定义 IP 分配。**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：

1. 分配的名称，该名称采用 **<node_hostname>-<network>** 格式
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。

本章之后的步骤将生成的环境文件 (**ctlplane-assignments.yaml**) 用作 **openstack overcloud deploy** 命令的组成部分。

7.6. 使用单独的 OVERCLOUD 节点网络

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 通讯。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 通讯。

对于此情境，有几个需要满足的要求：

- overcloud 节点必须适应 第 7.5 节 “为 Control Plane 配置网络” 中的基础网络配置。
- 您必须在 director 上启用 SSL/TLS，以便使用公共 API 端点。如需了解更多信息，请参阅第 4.6 节 “配置 director”和附录 A, *SSL/TLS 证书配置*。
- 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 7.2. 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
Controller	192.168.100.2

节点名	**IP** 地址或 **FQDN**
Compute	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。

编配配置

在 undercloud 上启用 SSL/TLS 通讯之后，director 为大部分服务提供公共 API 端点。但是，OpenStack Orchestration (heat) 使用内部端点作为默认的元数据供应商。这表示，需要对 undercloud 做一些修改，以便 overcloud 节点可以在公共端点访问 OpenStack Orchestration。此修改涉及更改 director 上的某些 Puppet hieradata。

您的 **undercloud.conf** 中的 **hieradata_override** 允许您指定用于 undercloud 配置的额外 Puppet hieradata。使用以下步骤修改与 OpenStack 编配相关的 hieradata：

1. 如果您尚未使用 **hieradata_override** 文件，请创建一个新文件。本示例使用了位于 **/home/stack/hieradata.yaml** 的文件。

2. 在 **/home/stack/hieradata.yaml** 中包含下列 hieradata：

   ```
   heat_clients_endpoint_type: public
   heat::engine::default_deployment_signal_transport:
   TEMP_URL_SIGNAL
   ```

   此操作会将端点类型从默认的 **internal** 更改为 **public**，同时将信令方法更改为使用 OpenStack Object Storage（swift）中的 TempURL。

3. 在您的 **undercloud.conf** 中，将 **hieradata_override** 参数设置为 hieradata 文件的路径：

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

4. 重新运行 **openstack overcloud install** 命令，以实施新配置选项。

这会使编配元数据服务器切换为使用 director 的公共 API 上的 URL。

IP 地址分配

IP 分配方法与第 7.5 节 “为 Control Plane 配置网络”类似。但是，由于 Control Plane 无法从部署的服务器路由，您将使用 **DeployedServerPortMap** 参数从您选定的 overcloud 节点子网分配 IP 地址，包括用于访问 Control Plane 的虚拟 IP 地址。以下是从 第 7.5 节 “为 Control Plane 配置网络” 修改的 **ctlplane-assignments.yaml** 环境文件版本，融合了这种网络架构：

```
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 1

parameter_defaults:
  NeutronPublicInterface: eth1
  EC2MetadataIp: 192.168.100.1 2
  ControlPlaneDefaultRoute: 192.168.100.1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
    controller0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
    compute0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.3
```

1

RedisVipPort 资源被映射至 **network/ports/noop.yaml**。出现这种映射的原因是默认的 Redis VIP 地址来自 Control Plane。在这种情况下，我们使用 **noop** 来禁用这种 Control Plane 映射。

2

EC2MetadataIp 和 **ControlPlaneDefaultRoute** 参数被设置为 Control Plane 虚拟 IP 地址的值。默认的 NIC 配置模板需要这些参数，且您必须将它们设置为可 ping 通的 IP 地址，以便通过部署期间执行的验证。或者，自定义 NIC 配置，使它们无需使用这些参数。

7.7. 利用预配置节点创建 OVERCLOUD

overcloud 部署使用来自 ??? 的标准 CLI 方法。对于预配置节点，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** - 禁止对未配合使用预配置基础架构的服务使用基本 CLI 验证。
- **environments/deployed-server-environment.yaml** - 用于创建和配置预配置基础架构的主要环境文件。这种环境文件用 **OS::Heat::DeployedServer** 资源代替 **OS::Nova::Server** 资源。
- **environments/deployed-server-bootstrap-environment-rhel.yaml** -用于在预配置服务器上执行启动引导脚本的环境文件。此脚本安装额外的软件包，并为 overcloud 节点提供基本配置。
- **environments/deployed-server-pacemaker-environment.yaml** - 用于在预配置 Controller 节点上进行 Pacemaker 配置的环境文件。 此文件的注册资源的命名空间使用来自 **deployed-server/deployed-server-roles-data.yaml** 的 Controller 角色名称，默认为 **ControllerDeployedServer**。

- **deployed-server/deployed-server-roles-data.yaml** - 一个示例自定义角色文件。此文件复制了默认的 **roles_data.yaml**，也包括各角色的 **disable_constraints: True** 参数。此参数会禁用生成的角色模板中的编配限制。这些限制适用于未搭配使用预配置基础架构的服务。

  使用您自己的自定义角色文件时，确保将各角色的 **disable_constraints: True** 参数包含在内。例如：

```
- name: ControllerDeployedServer
  disable_constraints: True
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    ...
```

以下是一个 overcloud 部署命令命令，采用了针对预配置架构的特定环境文件：

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack overcloud deploy \
  [other arguments] \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-
server-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-
server-bootstrap-environment-rhel.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-
server-pacemaker-environment.yaml \
  -r /usr/share/openstack-tripleo-heat-templates/deployed-
server/deployed-server-roles-data.yaml
```

由此开始 overcloud 配置。但是，当 overcloud 节点资源进入 **CREATE_IN_PROGRESS** 阶段时，部署堆栈暂停：

```
2017-01-14 13:25:13Z [overcloud.Compute.0.Compute]: CREATE_IN_PROGRESS
state changed
2017-01-14 13:25:14Z [overcloud.Controller.0.Controller]:
CREATE_IN_PROGRESS  state changed
```

出现这种暂停是因为 director 等待 overcloud 节点上的编配代理来轮询元数据服务器。下一节将介绍如何配置节点以开始轮询元数据服务器。

7.8. 轮询元数据服务器

部署目前正在进行，但会在进入 **CREATE_IN_PROGRESS** 阶段后暂停。下一步是在 overcloud 节点上配置编配代理，以便在 director 上轮询元数据服务器。可采用两种方式完成此操作：

重要

仅在初始部署时使用自动配置。扩展节点时不要使用自动配置。

自动配置

director 的核心 Heat 模板集包含一个脚本，在 overcloud 节点上对 Heat 代理执行自动配置。此脚本要求您以 **stack** 用户的身份 source **stackrc** 文件，以便进行 director 验证并查询编配服务：

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

此外，该脚本也要求使用一些额外的环境变量来定义节点的角色及其 IP 地址。这些环境变量包括：

OVERCLOUD_ROLES

待配置的以空格隔开的角色列表。这些角色与您的角色数据文件中定义的角色相关联。

[ROLE]_hosts

每个角色都需要一个环境变量，包括以空格隔开的、此角色中的节点 IP 地址列表。

以下命令演示了如何设置这些环境变量：

```
[stack@director ~]$ export OVERCLOUD_ROLES="ControllerDeployedServer
ComputeDeployedServer"
[stack@director ~]$ export
ControllerDeployedServer_hosts="192.168.100.2"
[stack@director ~]$ export ComputeDeployedServer_hosts="192.168.100.3"
```

运行该脚本，在各 overcloud 节点上配置编配代理：

```
[stack@director ~]$ /usr/share/openstack-tripleo-heat-
templates/deployed-server/scripts/get-occ-config.sh
```

注意

该脚本使用执行脚本的同一用户身份，通过 SSH 访问预配置节点。在这种情况下，该脚本以 **stack** 用户的身份进行验证。

该脚本完成以下操作：

- 查询 director 的编配服务获取各节点的元数据 URL 。
- 访问节点，使用各节点特定的元数据 URL 配置各节点的代理。
- 重新启动编配代理服务。

脚本完成之后，overcloud 节点开始在 director 上轮询编配服务。堆栈部署继续进行。

手动配置

如果您更愿意在预配置节点上手动配置编配代理，请使用下列命令在 director 上查询编配服务以获取各节点的元数据 URL：

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ for STACK in $(openstack stack resource list -n5 --
filter name=deployed-server -c stack_name -f value overcloud) ; do
STACKID=$(echo $STACK | cut -d '-' -f2,4 --output-delimiter " ") ; echo
```

```
"== Metadata URL for $STACKID ==" ; openstack stack resource metadata
$STACK deployed-server | jq -r '.["os-collect-
config"].request.metadata_url' ; echo ; done
```

这会显示堆栈名称和各节点的元数据 URL：

```
== Metadata URL for ControllerDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-
edServer-ts6lr4tm5p44-deployed-server-td42md2tap4g/43d302fa-d4c2-40df-
b3ac-624d6075ef27?
temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_expires=
2147483586

== Metadata URL for ComputeDeployedServer 0 ==
http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b8283e7125f05b764/ov-
edServer-wdpk7upmz3eh-deployed-server-ghv7ptfikz2j/0a43e94b-fe02-427b-
9bfe-71d2b7bb3126?
temp_url_sig=8a50d8ed6502969f0063e79bb32592f4203a136e&temp_url_expires=
2147483586
```

在各 overcloud 节点上：

1. 删除现有的 **os-collect-config.conf** 模板。此操作可以确保代理不会覆盖我们手动做出的更改：

   ```
   $ sudo /bin/rm -f /usr/libexec/os-apply-config/templates/etc/os-
   collect-config.conf
   ```

2. 配置 **/etc/os-collect-config.conf** 文件，使其使用对应的元数据 URL。例如，Controller 节点使用以下命令：

   ```
   [DEFAULT]
   collectors=request
   command=os-refresh-config
   polling_interval=30

   [request]
   metadata_url=http://192.168.24.1:8080/v1/AUTH_6fce4e6019264a5b828
   3e7125f05b764/ov-edServer-ts6lr4tm5p44-deployed-server-
   td42md2tap4g/43d302fa-d4c2-40df-b3ac-624d6075ef27?
   temp_url_sig=58313e577a93de8f8d2367f8ce92dd7be7aac3a1&temp_url_ex
   pires=2147483586
   ```

3. 保存这个文件。

4. 重新启动 **os-collect-config** 服务：

   ```
   [stack@controller ~]$ sudo systemctl restart os-collect-config
   ```

在配置和重新启动之后，这些编配代理会轮询 director 的编配服务以获取 overcloud 配置。部署堆栈继续创建过程，各节点的堆栈最终变更为 **CREATE_COMPLETE**。

7.9. 监控 OVERCLOUD 的创建过程

overcloud 配置过程开始。此过程需要一些时间完成。要查看 overcloud 创建的状态，请以 **stack** 用户身份打开另外一个终端并运行：

```
$ source ~/stackrc
$ heat stack-list --show-nested
```

heat stack-list --show-nested 命令会显示 overcloud 创建过程的当前状态。

7.10. 访问 OVERCLOUD

director 会生成脚本来配置和帮助认证 director 主机与 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：

```
$ source ~/overcloudrc
```

这会加载所需的环境变量，以便从 director 节点上的 CLI 与 overcloud 交互。要回到与 director 节点交互的状态，请运行：

```
$ source ~/stackrc
```

7.11. 扩展预配置节点

扩展预配置节点的流程与???中的标准扩展流程类似。但是，添加新预配置节点的流程却不相同，这是因为预配置节点不从 OpenStack Bare Metal (ironic) 和 OpenStack Compute (nova) 使用标准注册和管理流程。

扩展预配置节点

利用预配置节点扩展 overcloud 时，您需要配置各节点上的编配代理，以匹配 director 的节点数量。

扩展节点的常规流程如下：

1. 依照配置要求准备新的预配置节点。
2. 扩展节点。请参阅???了解这些说明。
3. 执行部署命令之后，等稍等片刻，直至 director 创建新的节点资源。依照 第 7.8 节 "轮询元数据服务器" 中的说明，手动配置预配置节点以轮询 director 的编配服务器元数据 URL。

缩减预配置节点

使用预配置节点缩减 overcloud 时，请遵循 ??? 中显示的常规缩减说明。

从堆栈中移除 overcloud 节点之后，关闭这些节点。在标准部署中，此功能由 director 的裸机服务管控。但是，使用预配置节点时，您应当手动关闭这些节点，或者使用各物理系统的电源管理控制功能。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。

关闭已移除的节点之后，重新将它们部署到基础操作系统配置中，避免它们将来意外加入到 overcloud 中

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。

7.12. 移除预配置 OVERCLOUD

使用与标准 overcloud 相同的流程，移除使用预配置节点的整个 overcloud。请参阅???了解更多详细信息。

移除 overcloud 之后，关闭所有节点，并将它们重新部署到基础操作系统配置中。

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。

7.13. 完成 OVERCLOUD 的创建

通过预配置节点创建 overcloud 的流程到此结束。如需了解创建之后的功能，请参阅???。

第 8 章 创建 OVERCLOUD 后执行的任务

本章介绍了在创建 overcloud 后需要执行的一些功能。

8.1. 创建 OVERCLOUD TENANT 网络

overcloud 需要为实例提供租户网络。对 **overcloud** 执行 source 命令并在 Neutron 中创建一个初始租户网络。例如：

```
$ source ~/overcloudrc
$ openstack network create default
$ openstack subnet create default --network default --gateway
172.20.1.1 --subnet-range 172.20.0.0/16
```

这会创建一个名为 **default** 的基本 Neutron 网络。overcloud 会使用内部 DHCP 机制从这个网络中自动分配 IP 地址。

使用 **neutron net-list** 来确定创建的网络：

```
$ openstack network list
+-----------------------+-------------+----------------------------
----------+
| id                    | name        | subnets
|
+-----------------------+-------------+----------------------------
----------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-
1c6f8f4f332f |
+-----------------------+-------------+----------------------------
----------+
```

8.2. 创建 OVERCLOUD EXTERNAL 网络

您需要在 overcloud 上创建外部网络，以便您可以分配 IP 地址到实例。

使用原生的 VLAN

这个步骤假设 External 网络有一个专用的接口，或一个原生的 VLAN。

source **overcloud**，并在 Neutron 中创建一个 External 网络：

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type
flat --provider-physical-network datacentre
$ 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
```

在本例中，创建一个名为 **public** 的网络。overcloud 需要将这一特定名称用于默认的浮动 IP 池。另外，它对???中所述的验证测试也非常重要。

此命令还会将网络映射到 **datacentre** 物理网络。因此，**datacentre** 将映射到 **br-ex** 网桥。除非创建 overcloud 过程中使用了自定义 neutron 设置，否则此选项可保留默认设置。

使用非原生的 VLAN

如果没有使用原生的 VLAN，使用以下命令把网络分配给一个 VLAN：

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type
vlan --provider-physical-network datacentre --provider-segment 104
$ 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
```

provider:segmentation_id 的值定义了要使用的 VLAN。在这个示例中，使用 104。

使用 **neutron net-list** 来确定创建的网络：

```
$ openstack network list
+-----------------------+-------------+---------------------------
----------+
| id                    | name        | subnets
|
+-----------------------+-------------+---------------------------
----------+
| d474fe1f-222d-4e32... | public      | 01c5f621-1e0f-4b9d-9c30-
7dc59592a52f |
+-----------------------+-------------+---------------------------
----------+
```

8.3. 创建额外的浮动 IP 地址网络

只要满足以下条件，浮动 IP 网络可以使用任何网桥，而不只局限于 **br-ex**：

- 在网络环境文件中，**NeutronExternalNetworkBridge** 被设置为 **"''"**。
- 在部署的过程中已映射了额外的网桥。例如，把一个名为 **br-floating** 的新网桥映射到 **floating** 物理网络：

  ```
  $ openstack overcloud deploy --templates -e /usr/share/openstack-
  tripleo-heat-templates/environments/network-isolation.yaml -e
  ~/templates/network-environment.yaml --neutron-bridge-mappings
  datacentre:br-ex,floating:br-floating
  ```

在创建 overcloud 后创建浮动 IP 网络：

```
$ openstack network create ext-net --external --provider-physical-
network floating --provider-network-type vlan --provider-segment 105
$ openstack network create ext-subnet --network ext-net --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
```

8.4. 创建 OVERCLOUD 供应商网络

供应商网络就是物理附加到位于部署的 overcloud 以外的网络。它可以是现有的基础架构网络，或是通过路由而非浮动 IP 为实例提供直接外部访问的网络。

在创建一个供应商网络时，可以使用网桥映射把它和一个物理网络相关联。这和创建浮动 IP 网络相似。您需要把供应商网络添加到 Controller 节点和 Compute 节点中，这是因为 Compute 节点会把虚拟机的虚拟网络接口直接附加到附加的网络接口上。

例如，供应商网络是 br-ex bridge 网桥上的一个 VLAN，使用以下命令在 VLAN 201 上添加一个供应商网络：

```
$ openstack network create provider_network --provider-physical-network
datacentre --provider-network-type vlan --provider-segment 201 --share
```

此命令会创建一个共享网络。另外，也可以指定租户，而不指定 **--share**。此网络只对特定的租户有效。如果将供应商网络标记为外部，则只有操作员可以在该网络上创建端口。

如果需要 neutron 为租户实例提供 DHCP 服务，则需要向供应商网络添加一个子网：

```
$ openstack network create provider-subnet --network  provider_network
--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
```

8.5. 验证 OVERCLOUD

overcloud 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信息。有关使用 OpenStack Integration Test Suite 的完整说明，请参阅 OpenStack Integration Test Suite 指南。

运行 Integration Test Suite 之前

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
$ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface
vlan201 type=internal
$ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev
vlan201
```

运行 OpenStack Integration Test Suite 之前，检查确认您的 overcloud 中具有 **heat_stack_owner** 角色：

```
$ source ~/overcloudrc
$ openstack role list
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 6226a517204846d1a26d15aae1af208f | swiftoperator    |
| 7c7eb03955e545dd86bbfeb73692738b | heat_stack_owner |
+----------------------------------+------------------+
```

如果角色不存在，则创建它：

```
$ openstack role create heat_stack_owner
```

运行 Integration Test Suite 之后

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
$ sudo ovs-vsctl del-port vlan201
```

8.6. 隔离 CONTROLLER 节点

隔离（fencing）就是把一个节点和集群中的其它节点进行隔离来保护整个集群和它的资源。如果没有隔离功能，一个有问题的节点可能会导致集群中的数据被破坏。

director 使用 Pacemaker 来为 Controller 节点提供高可用性集群。Pacemaker 使用 STONITH（Shoot-The-Other-Node-In-The-Head）进程来隔离有问题的节点。在默认情况下，STONITH 在集群中被禁用，您需要手工配置来使 Pacemaker 可以控制集群中的每个节点的电源管理。

注意

使用 director 上的 **stack** 用户，以 **heat-admin** 用户身份登录到每个节点。overcloud 的创建过程会自动把 **stack** 用户的 SSH 密钥复制给每个节点的 **heat-admin** 用户。

运行 **pcs status** 验证您有一个正在运行的集群：

```
$ sudo pcs status
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

使用 **pcs property show** 验证 stonith 被禁用：

```
$ sudo pcs property show
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

Controller 节点为 director 支持的不同电源管理设备包括了一组隔离代理。这包括：

表 8.1. 隔离代理

设备	类型
fence_ipmilan	Intelligent Platform Management Interface (IPMI)
fence_idrac、**fence_drac5**	Dell Remote Access Controller (DRAC)
fence_ilo	Integrated Lights-Out (iLO)
fence_ucs	Cisco UCS - 如需了解更多信息，请参阅 Configuring Cisco Unified Computing System (UCS) Fencing on an OpenStack High Availability Environment
fence_xvm、**fence_virt**	Libvirt 和 SSH

这一节的剩余部分会使用 IPMI 代理（**fence_ipmilan**）作为示例。

查看 Pacemaker 支持的完整 IPMI 选项列表：

```
$ sudo pcs stonith describe fence_ipmilan
```

每个节点都需要配置 IPMI 设备来控制电源管理。这包括为每个节点在 pacemaker 中添加一个 **stonith** 设备。在集群中使用以下命令：

注意

每个示例中的第二个命令保证了节点不会隔离自己。

对于 Controller 节点 0：

```
$ sudo pcs stonith create my-ipmilan-for-controller-0 fence_ipmilan
pcmk_host_list=overcloud-controller-0 ipaddr=192.168.24.205 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-0 avoids
overcloud-controller-0
```

对于 Controller 节点 1：

```
$ sudo pcs stonith create my-ipmilan-for-controller-1 fence_ipmilan
pcmk_host_list=overcloud-controller-1 ipaddr=192.168.24.206 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-1 avoids
overcloud-controller-1
```

对于 Controller 节点 2：

```
$ sudo pcs stonith create my-ipmilan-for-controller-2 fence_ipmilan
pcmk_host_list=overcloud-controller-2 ipaddr=192.168.24.207 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-2 avoids
overcloud-controller-2
```

运行以下命令来设置所有 stonith 资源：

```
$ sudo pcs stonith show
```

运行以下命令来查看一个特定的 stonith 资源：

```
$ sudo pcs stonith show [stonith-name]
```

最后，把 **stonith** 属性设置为 **true** 来启用隔离功能：

```
$ sudo pcs property set stonith-enabled=true
```

验证属性：

```
$ sudo pcs property show
```

8.7. 修改 OVERCLOUD 环境

有些时候，您可能要修改 overcloud 来添加一些功能，或改变它的运行方式。要修改 overcloud，请在您的自定义环境文件和 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，如果根据 ??? 创建了一个 overcloud，您应该重新运行以下命令：

```
$ 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。它不会重新创建 overcloud，而是更改现有的 overcloud。

如果需要包括一个新的环境文件，在 **openstack overcloud deploy** 命令中使用 **-e** 选项添加它。例如：

```
$ 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
```

这包括从环境文件中读入到栈中的新参数和资源。

重要

建议您不要手动修改 overcloud 的配置，因为 director 可能会在以后覆盖这些修改。

8.8. 把虚拟机导入到 OVERCLOUD

如果您有一个已存在的 OpenStack 环境，并希望把它的虚拟机迁移到 Red Hat OpenStack Platform 环境中，请进行以下操作。

为一个运行的服务器进行快照来创建一个新的镜像，然后下载这个镜像。

```
$ openstack server image create instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

把导出的镜像上传到 overcloud，然后启动一个新实例。

```
$ openstack image create imported_image --file exported_vm.qcow2 --
disk-format qcow2 --container-format bare
$ openstack server create  imported_instance --key-name default --
flavor m1.demo --image imported_image --nic net-id=net_id
```

重要

每个虚拟机磁盘都需要从存在的 OpenStack 环境中复制到新的 Red Hat OpenStack Platform。使用 QCOW 的快照将会丢掉它原始的层系统。

8.9. 从一个 OVERCLOUD COMPUTE 节点中迁移虚拟机

在一些情况下，您可以在 overcloud 计算节点上执行维护操作。为了避免停机，请按照以下方法把计算节点上的虚拟机迁移到 overcloud 中的另一个计算节点。

所有计算节点都需要一个共享的 SSH，让每一个主机的 **nova** 用户在迁移过程中具有相应的访问权限。以下脚本在 undercloud 上创建 SSH 密钥，并将它复制到所有计算节点，然后设置适当的配置和权限：

```
#!/bin/bash

ssh-keygen -f ~/nova_rsa -t rsa -N ''

ALLCOMPUTES=$(openstack server list --name ".*compute.*" -c Networks -f
csv --quote none | tail -n+2 | awk -F '=' '{ print $2 }')

for COMPUTE in $ALLCOMPUTES
do
  ssh heat-admin@$COMPUTE "sudo usermod -s /bin/bash nova"
  ssh heat-admin@$COMPUTE "sudo mkdir -p /var/lib/nova/.ssh"
  scp ~/nova* heat-admin@$COMPUTE:~/
  ssh heat-admin@$COMPUTE "sudo cp ~/nova_rsa
/var/lib/nova/.ssh/id_rsa"
```

```
  ssh heat-admin@$COMPUTE "sudo cp ~/nova_rsa.pub
/var/lib/nova/.ssh/id_rsa.pub"
  ssh heat-admin@$COMPUTE "sudo rm ~/nova_rsa*"
  ssh heat-admin@$COMPUTE "echo 'StrictHostKeyChecking no'|sudo tee -a
/var/lib/nova/.ssh/config"
  ssh heat-admin@$COMPUTE "sudo cat /var/lib/nova/.ssh/id_rsa.pub|sudo
tee -a /var/lib/nova/.ssh/authorized_keys"
  ssh heat-admin@$COMPUTE "sudo chown -R nova: /var/lib/nova/.ssh"
  ssh heat-admin@$COMPUTE "sudo su -c 'chmod 600 /var/lib/nova/.ssh/*'
nova"
  ssh heat-admin@$COMPUTE "sudo chmod 700 /var/lib/nova/.ssh"
done

rm ~/nova_rsa*
```

运行此脚本后，请测试计算节点之间的 SSH 连接。从 undercloud 登录计算节点：

```
$ ssh heat-admin@192.0.1.10
```

切换到 **nova** 用户：

```
$ sudo su - nova
```

尝试以 **nova** 用户身份登录另一个计算节点：

```
$ ssh nova@192.0.1.11
```

返回到 director 节点，对 **overcloudrc** 执行 source 命令，并获取当前的 nova 服务列表：

```
$ source ~/stack/overcloudrc
$ openstack compute service list
```

在要迁移的节点上禁用 **nova-compute** 服务。

```
$ openstack compute service set [hostname] nova-compute --disable
```

这会防止新的虚拟机在它上面运行。

开始从节点迁移实例。以下命令可迁移一个实例：

```
$ openstack server migrate [server-name]
```

为您需要从已禁用计算节点迁移的每个实例运行此命令。

使用以下命令检索迁移过程的当前状态：

```
$ nova migration-list
```

当一个实例的迁移过程完成后，它在 nova 中的状态将变为 **VERIFY_RESIZE**。此时，您可以选择确认迁移已成功完成，或把它恢复到原来的状态。要确认迁移已成功完成，运行以下命令：

```
$ nova resize-confirm [server-name]
```

为具有 **VERIFY_RESIZE** 状态的每个实例运行此命令。

这会从一个主机上迁移所有实例。现在，您就可以在没有实例下线的情况下执行维护操作。要把主机返回到启用的状态，运行以下命令：

```
$ openstack compute service set [hostname] nova-compute --enable
```

8.10. 运行 ANSIBLE 自动化

director 使您能够在 OpenStack Platform 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。

重要

动态清单工具仅包括 undercloud 以及默认的 **`controller`** 和 **`compute`** overcloud 节点。不支持其他角色。

要查看动态的节点清单，请在 source **`stackrc`** 之后运行 **`tripleo-ansible-inventory`** 命令：

```
$ souce ~/stackrc
$ tripleo-ansible-inventory --list
```

`--list` 选项提供与所有主机有关的详情。

此选项以 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"]}
```

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：

```
ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- 将 **`[HOSTS]`** 更换为要使用的主机类型。例如：
  - **`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`**
  - **`undercloud`**，适用于 undercloud
  - **`"*"`**，适用于所有节点
- 将 **`[OTHER OPTIONS]`** 更换为额外的 Ansible 选项。以下是一些有用的选项：
  - **`--ssh-extra-args='-o StrictHostKeyChecking=no'`**，用于跳过主机密钥检查确认操作。

- **-u [USER]**，用于更改执行 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。
- **-m [MODULE]**，使用特定的 Ansible 模块。默认为 **command**，用于执行 Linux 命令。
- **-a [MODULE_ARGS]**，为选定的模块定义参数。

重要

overcloud 上的 Ansible 自动化不属于标准 overcloud 堆栈范围之内。这表示后续执行的 **openstack overcloud deploy** 命令可能会覆盖 overcloud 节点上基于 Ansible 的 OpenStack Platform 服务配置。

8.11. 防止 OVERCLOUD 被删除

为了避免无意中使用 **heat stack-delete overcloud** 命令造成 overcloud 被删除，Heat 包括了一组策略来限制某些操作。打开 **/etc/heat/policy.json** 并找到以下参数：

```
"stacks:delete": "rule:deny_stack_user"
```

把它改为：

```
"stacks:delete": "rule:deny_everybody"
```

保存这个文件。

这会避免使用 **heat** 客户端删除 overcloud。若要允许删除 overcloud，请把策略恢复为原先的值。

8.12. 删除 OVERCLOUD

在需要时，可以删除整个 overcloud。

删除任何现有的 overcloud：

```
$ openstack stack delete overcloud
```

确认删除 overcloud：

```
$ openstack stack list
```

删除操作会需要几分钟完成。

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 9 章 扩展 OVERCLOUD

警告

为实例实施高可用性期间无法执行任何升级或扩展操作（如计算实例高可用性中所述）。尝试进行此类操作都将会失败。

此外，为实例启用高可用性也会阻碍您未来使用 director 升级 overcloud，不论升级的幅度如何。如需更多信息，请参阅 https://access.redhat.com/solutions/2661641。

在某些情况下，您可能需要在创建 overcloud 后添加或删除节点。例如，可能需要为 overcloud 添加计算节点。这样的情形需要更新 overcloud。

下表介绍了对每个节点类型进行扩展的支持信息：

表 9.1. 每个节点类型的扩展支持

节点类型	扩充	缩小	备注
Controller	N	N	
Compute	Y	Y	
Ceph Storage 节点	Y	N	在初始创建的 overcloud 中必须至少有一个 Ceph 存储节点。
Block Storage 节点	N	N	
Object Storage 节点	Y	Y	需要手工进行 ring 管理。相关详情请参阅第 9.6 节 “替换 Object 存储节点”。

重要

在进行 overcloud 扩展前，确保至少有 10 GB 的可用空间。这些可用空间将在节点部署过程中用于保存镜像转换和缓存。

9.1. 添加额外节点

为 director 的节点池添加更多节点，创建一个包括用来注册新节点信息的 JSON 文件（例如，**newnodes.json**）：

```
{
  "nodes":[
    {
        "mac":[
            "dd:dd:dd:dd:dd:dd"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"pxe_ipmitool",
        "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":"pxe_ipmitool",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.168.24.208"
    }
  ]
}
```

如需了解与这些参数相关的信息，请参阅 ???。

运行以下命令注册这些节点：

```
$ openstack baremetal import --json newnodes.json
```

在注册完这些节点后，为它们启动内省进程。为每个新节点运行以下命令：

```
$ openstack baremetal node manage [NODE UUID]
$ openstack overcloud node introspect [NODE UUID] --provide
```

这会发现节点，并为它们创建硬件属性的基准数据。

在内省操作完成后，把新节点标记为相应的角色。例如，使用以下命令把节点标记为一个 Compute 节点：

```
$ openstack baremetal node set --property
capabilities='profile:compute,boot_option:local' [NODE UUID]
```

设置部署时使用的引导镜像。找到 **bm-deploy-kernel** 和 **bm-deploy-ramdisk** 镜像的 UUID：

```
$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel       |
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk      |
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

在新节点的 **deploy_kernel** 和 **deploy_ramdisk** 设置中使用这些 UUID：

```
$ openstack baremetal node set --driver-info deploy_kernel='09b40e3d-
0382-4925-a356-3a4b4f36b514' [NODE UUID]
$ openstack baremetal node set --driver-info deploy_ramdisk='765a46af-
4417-4592-91e5-a300ead3faf6' [NODE UUID]
```

若要扩展 overcloud，需要使用角色所需的节点数量，重新运行 **openstack overcloud deploy**。例如，扩展到 5 个计算节点：

```
$ openstack overcloud deploy --templates --compute-scale 5
[OTHER_OPTIONS]
```

这会更新整个 overcloud 栈。请注意，这只会更新栈，而不会删除 overcloud 或替换栈。

重要

务必包含初始 overcloud 创建中的所有环境文件和选项。其中包括非计算节点的相同扩展参数。

9.2. 删除 COMPUTE 节点

在某些情况下，您可能需要从 overcloud 中删除计算节点。例如，需要替换有问题的计算节点。

重要

在从 overcloud 中删除计算节点前，先将该节点上的工作负载迁移到其他计算节点。请参阅???了解详细信息。

接下来，在 overcloud 中禁用节点的计算服务。这会停止在此节点上调度新的实例。

```
$ source ~/stack/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
$ source ~/stack/stackrc
```

若要删除 overcloud 节点，需要使用本地模板文件对 director 中的 **overcloud** 栈进行更新。首先，确定 overcloud 栈的 UUID：

```
$ openstack stack list
```

找到要被删除的节点的 UUID：

```
$ openstack server list
```

运行以下命令来从栈中删除节点，并相应地更新计划：

```
$ openstack overcloud node delete --stack [STACK_UUID] --templates -e
[ENVIRONMENT_FILE] [NODE1_UUID] [NODE2_UUID] [NODE3_UUID]
```

重要

如果在创建 overcloud 时传递了额外的环境文件，请使用 **`-e`** 或 **`--environment-file`** 选项再次传递它们来避免对 overcloud 进行不必要的手动更改。

重要

在继续进行操作前，请确认 **`openstack overcloud node delete`** 命令已运行完。使用 **`openstack stack list`** 命令检查 **`overcloud`** 栈的状态是否已变为 **`UPDATE_COMPLETE`**。

最后，删除节点的 Compute 服务：

```
$ source ~/stack/overcloudrc
$ openstack compute service delete [service-id]
$ source ~/stack/stackrc
```

删除节点的 Open vSwitch 代理：

```
$ source ~/stack/overcloudrc
$ openstack network agent list
$ openstack network agent delete [openvswitch-agent-id]
$ source ~/stack/stackrc
```

现在，可以安全地把节点从 overcloud 中删除，并将它部署用于其他目的。

9.3. 替换 COMPUTE 节点

当一个 Compute 节点出现问题时，您可以使用一个正常的节点替换它。使用以下步骤替换 Compute 节点：

- 迁移 Compute 节点上的负载并关闭节点。详细信息，请参阅 ???。
- 从 overcloud 中删除计算节点。有关此过程的信息，请参阅第 9.2 节 “删除 Compute 节点”。
- 使用新的计算节点扩展 overcloud。有关此过程的信息，请参阅 第 9.1 节 “添加额外节点”。

这个过程确保了替换节点的操作不会影响到任何实例的可用性。

9.4. 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，并使用一个新 Controller 节点替换它。另外，还要保证节点可以和集群中的其它节点进行连接。

本节介绍了如何替换控制器节点。此过程包括运行 **`openstack overcloud deploy`** 命令来通过替换控制器节点的请求更新 overcloud。请注意，此过程不是完全自动的；在 overcloud 栈更新过程中，**`openstack overcloud deploy`** 命令会在某个时点上报告失败，overcloud 栈更新过程随之停止。这时，需要一些人工干预，然后 **`openstack overcloud deploy`** 进程才会继续。

9.4.1. 预检查

在尝试替换 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。参照以下初步检查列表，确定是否可以安全地执行控制器节点替换。在 undercloud 中执行这些检查的所有命令。

1. 在 undercloud 中检查 **`overcloud`** 栈的当前状态：

   ```
   $ source stackrc
   $ 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. 对 undercloud 数据库进行备份：

   ```
   $ mkdir /home/stack/backup
   $ sudo mysqldump --all-databases --quick --single-transaction |
   gzip > /home/stack/backup/dump_db_undercloud.sql.gz
   $ sudo systemctl stop openstack-ironic-api.service openstack-
   ironic-conductor.service openstack-ironic-inspector.service
   openstack-ironic-inspector-dnsmasq.service
   $ sudo cp /var/lib/ironic-inspector/inspector.sqlite
   /home/stack/backup
   $ sudo systemctl start openstack-ironic-api.service openstack-
   ironic-conductor.service openstack-ironic-inspector.service
   openstack-ironic-inspector-dnsmasq.service
   ```

3. 确认您的 undercloud 拥有 10 GB 的可用存储空间，以容纳部署新节点期间的镜像缓存和转换。

4. 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，使用以下命令获得 Pacemaker 的状态：

   ```
   $ ssh heat-admin@192.168.0.47 'sudo pcs status'
   ```

   这个命令的输出应该显示，所有服务都在存在的节点上运行，并已在出现故障的节点上停止运行。

5. 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`**

使用以下命令在每个运行的 Controller 节点上检查这些参数（IP 地址分别为 192.168.0.47 和 192.168.0.46）：

```
$ for i in 192.168.0.47 192.168.0.46 ; do echo "*** $i ***" ;
ssh heat-admin@$i "sudo mysql --exec=\"SHOW STATUS LIKE
'wsrep_local_state_comment'\" ; sudo mysql --exec=\"SHOW
STATUS LIKE 'wsrep_cluster_size'\""; done
```

6. 检查 RabbitMQ 的状态。例如，一个运行节点的 IP 地址是 192.168.0.47，使用以下命令获得状态值

```
$ ssh heat-admin@192.168.0.47 "sudo rabbitmqctl cluster_status"
```

running_nodes 应该只显示两个可用的节点，而不会显示有故障的节点。

7. 如果启用了隔离服务，需要禁用它。例如，一个运行的 Controller 节点的 IP 地址是 192.168.0.47，使用以下命令禁用隔离服务：

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-
enabled=false"
```

使用以下命令检查隔离服务的状态：

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-
enabled"
```

8. 在 director 节点上检查 **nova-compute** 服务：

```
$ sudo systemctl status openstack-nova-compute
$ 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**。

9. 确保所有 undercloud 服务都在运行：

```
$ sudo systemctl -t service
```

9.4.2. 替换节点

找出要被删除节点的索引。节点索引是 **nova list** 输出中的实例名的一个后缀。

```
[stack@director ~]$ openstack server list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 861408be-4027-4f53-87a6-cd3cf206ba7a | overcloud-compute-0    |
| 0966e9ae-f553-447a-9929-c4232432f718 | overcloud-compute-1    |
| 9c08fa65-b38c-4b2e-bd47-33870bff06c7 | overcloud-compute-2    |
| a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af | overcloud-controller-0 |
| cfefaf60-8311-4bc3-9416-6a824a40a9ae | overcloud-controller-1 |
| 97a055d4-aefd-481c-82b7-4a5f384036d2 | overcloud-controller-2 |
+--------------------------------------+------------------------+
```

本例中的目标是删除 **overcloud-controller-1** 节点，并替换为 **overcloud-controller-**

3。首先，把节点设置为维护模式，从而使 director 不再重新部署故障节点。把 **nova list** 中的实例 ID 与 **ironic node-list** 中的节点 ID 相关联

```
[stack@director ~]$ openstack baremetal node list
+--------------------------------------+------+--------------------
------------------+
| UUID                                 | Name | Instance UUID
|
+--------------------------------------+------+--------------------
------------------+
| 36404147-7c8a-41e6-8c72-a6e90afc7584 | None | 7bee57cf-4a58-4eaf-
b851-2a8bf6620e48 |
| 91eb9ac5-7d52-453c-a017-c0e3d823efd0 | None | None
|
| 75b25e9a-948d-424a-9b3b-f0ef70a6eacf | None | None
|
| 038727da-6a5c-425f-bd45-fda2f4bd145b | None | 763bfec2-9354-466a-
ae65-2401c13e07e5 |
| dc2292e6-4056-46e0-8848-d6e96df1f55d | None | 2017b481-706f-44e1-
852a-2ee857c303c4 |
| c7eadcea-e377-4392-9fc3-cf2b02b7ec29 | None | 5f73c7d7-4826-49a5-
b6be-8bfd558f3b41 |
| da3a8d19-8a59-4e9d-923a-6a336fe10284 | None | cfefaf60-8311-4bc3-
9416-6a824a40a9ae |
| 807cb6ce-6b94-4cd1-9969-5c47560c2eee | None | c07c13e6-a845-4791-
9628-260110829c3a |
+--------------------------------------+------+--------------------
------------------+
```

把节点设为维护模式：

```
[stack@director ~]$ openstack baremetal node maintenance set da3a8d19-
8a59-4e9d-923a-6a336fe10284
```

使用 **control** 配置集标记新节点。

```
[stack@director ~]$ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-
9b3b-f0ef70a6eacf
```

创建一个 YAML 文件（**~/templates/remove-controller.yaml**），它定义了要被删除的节点索引：

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['1']}]
```

在确定节点索引后，重新部署 overcloud 并包含 **remove-controller.yaml** 环境文件：

```
[stack@director ~]$ openstack overcloud deploy --templates --control-
scale 3 -e ~/templates/remove-controller.yaml [OTHER OPTIONS]
```

重要

如果在创建 overcloud 时传递了额外的环境文件或选项，现在请再次传递它们来避免对 overcloud 进行不必要的更改。

请注意，**-e ~/templates/remove-controller.yaml** 在这个实例中只需要一次。

director 会删除旧节点，创建一个新节点并更新 overcloud 栈。您可以使用以下命令检查 overcloud 栈的状态：

```
[stack@director ~]$ openstack stack list --nested
```

9.4.3. 手工干预

在 **ControllerNodesPostDeployment** 阶段，overcloud 栈更新会在 **ControllerLoadBalancerDeployment_Step1** 的步骤中出现因 **UPDATE_FAILED** 错误而终止执行的情况。这是因为一些 Puppet 模块不支持节点替换。因此，这一时点上需要一些人工干预。请执行以下配置步骤：

1. 获得 Controller 节点的 IP 地址列表。例如：

```
[stack@director ~]$ openstack server list
... +------------------------+ ... +-------------------------+
... | 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   |
... +------------------------+ ... +-------------------------+
```

2. 在一个存在节点的 **/etc/corosync/corosync.conf** 文件中检查删除节点的 **nodeid** 值。例如，节点 **overcloud-controller-0**（IP 地址为 192.168.0.47）：

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo cat
/etc/corosync/corosync.conf"
```

这个命令会显示一个 **nodelist**，它包括了被删除节点（**overcloud-controller-1**）的 ID：

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-1
    nodeid: 2
  }
  node {
```

```
    ring0_addr: overcloud-controller-2
    nodeid: 3
  }
}
```

记录下被删除节点的 **nodeid** 值以供以后使用。在这个示例中，它的值是 2。

3. 从每个节点的 Corosync 配置中删除失败的节点，并重启 Corosync。对于这个示例，登录到 **overcloud-controller-0** 和 **overcloud-controller-2** 并运行以下命令：

```
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
reload corosync"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
reload corosync"
```

4. 登录到剩下的节点之一，使用 **crm_node** 命令从集群中删除节点：

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ sudo crm_node -R
overcloud-controller-1 --force
```

保持在这个节点的登录。

5. 从 RabbitMQ 集群中删除故障节点：

```
[heat-admin@overcloud-controller-0 ~]$ sudo rabbitmqctl
forget_cluster_node rabbit@overcloud-controller-1
```

6. 从 MongoDB 中删除失败的节点。首先，找到节点 Interal API 连接的 IP 地址。

```
[heat-admin@overcloud-controller-0 ~]$ sudo netstat -tulnp | grep
27017
tcp        0      0 192.168.0.47:27017    0.0.0.0:*
LISTEN      13415/mongod
```

检查这个节点是否是 **primary** 副本：

```
[root@overcloud-controller-0 ~]# echo "db.isMaster()" | mongo --
host 192.168.0.47:27017
MongoDB shell version: 2.6.11
connecting to: 192.168.0.47:27017/echo
{
  "setName" : "tripleo",
  "setVersion" : 1,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "192.168.0.47:27017",
    "192.168.0.46:27017",
    "192.168.0.45:27017"
  ],
```

```
    "primary" : "192.168.0.47:27017",
    "me" : "192.168.0.47:27017",
    "electionId" : ObjectId("575919933ea8637676159d28"),
    "maxBsonObjectSize" : 16777216,
    "maxMessageSizeBytes" : 48000000,
    "maxWriteBatchSize" : 1000,
    "localTime" : ISODate("2016-06-09T09:02:43.340Z"),
    "maxWireVersion" : 2,
    "minWireVersion" : 0,
    "ok" : 1
}
bye
```

从输出中可以判断当前节点是否是 primary 节点。如果不是，使用 **primary** 所指定节点的 IP 地址。

从 primary 节点连接到 MongoDB：

```
[heat-admin@overcloud-controller-0 ~]$ mongo --host 192.168.0.47
MongoDB shell version: 2.6.9
connecting to: 192.168.0.47:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
tripleo:PRIMARY>
```

检查 MongoDB 集群的状态：

```
tripleo:PRIMARY> rs.status()
```

_id 代表节点，在删除失败的节点时需要使用 **name**。在这里，我们删除 Node 1，它的 **name** 是 **192.168.0.45:27017**：

```
tripleo:PRIMARY> rs.remove('192.168.0.45:27017')
```

重要

您需要针对 **PRIMARY** 副本集运行这个命令。如果您可以看到以下信息：

```
"replSetReconfig command must be sent to the current
replica set primary."
```

重新登录到 **PRIMARY** 节点上的 MongoDB。

> **注意**
>
> 在删除失败节点的副本集时出现以下输出是正常的：
>
> ```
> 2016-05-07T03:57:19.541+0000 DBClientCursor::init call()
> failed
> 2016-05-07T03:57:19.543+0000 Error: error doing query:
> failed at src/mongo/shell/query.js:81
> 2016-05-07T03:57:19.545+0000 trying reconnect to
> 192.168.0.47:27017 (192.168.0.47) failed
> 2016-05-07T03:57:19.547+0000 reconnect 192.168.0.47:27017
> (192.168.0.47) ok
> ```

退出 MongoDB：

```
tripleo:PRIMARY> exit
```

7. 在 Galera 集群中更新节点列表：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource update
galera wsrep_cluster_address=gcomm://overcloud-controller-
0,overcloud-controller-3,overcloud-controller-2
```

8. 配置新节点上的 Galera 集群检查。将现有节点的 **/etc/sysconfig/clustercheck** 复制到新节点的同一位置上。

9. 配置新节点上 **root** 用户的 Galera 访问权限。将现有节点的 **/root/.my.cnf** 复制到新节点的同一位置上。

10. 为集群添加新节点：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster node add
overcloud-controller-3
```

11. 检查每个节点上的 **/etc/corosync/corosync.conf** 文件。如果新节点的 **nodeid** 和已删除节点的相同，则需要把它更新为一个新的 nodeid 值。例如，**/etc/corosync/corosync.conf** 文件包括一个新节点（**overcloud-controller-3**）的项：

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-2
    nodeid: 3
  }
  node {
    ring0_addr: overcloud-controller-3
    nodeid: 2
  }
}
```

请注意，新节点使用和被删除节点相同的 **nodeid**。把这个值更新为一个没有被使用过的节点 ID 值。例如：

```
node {
  ring0_addr: overcloud-controller-3
  nodeid: 4
}
```

在包括新节点在内的所有 Controller 节点的 **/etc/corosync/corosync.conf** 文件中更新这个 **nodeid** 值。

12. 只在存在的节点上重启 Corosync 服务。例如，在 **overcloud-controller-0** 上：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster reload
corosync
```

在 **overcloud-controller-2** 上：

```
[heat-admin@overcloud-controller-2 ~]$ sudo pcs cluster reload
corosync
```

不要在新节点上运行这个命令。

13. 启动新的 Controller 节点：

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster start
overcloud-controller-3
```

14. 通过 Pacemaker 启用并重启一些服务。集群当前处于维护模式，您需要临时禁用它来启用服务。例如：

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=false --wait
```

15. 等待 Galera 服务在所有节点上都已启动。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep
galera -A1
Master/Slave Set: galera-master [galera]
Masters: [ overcloud-controller-0 overcloud-controller-2
overcloud-controller-3 ]
```

如果需要，在新节点上执行 **cleanup**：

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs resource cleanup
galera --node overcloud-controller-3
```

16. 把集群切换为维护模式：

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=true --wait
```

手动配置已完成。重新运行 overcloud 部署命令来继续栈的更新：

```
[stack@director ~]$ openstack overcloud deploy --templates --control-
scale 3 [OTHER OPTIONS]
```

重要

如果在创建 overcloud 时传递了额外的环境文件或选项，现在请再次传递它们来避免对 overcloud 进行不必要的更改。但请注意，此时不再需要 **remove-controller.yaml** 文件。

9.4.4. 完成配置 overcloud 服务

在 overcloud 栈更新完成后，还需要一些最终配置。登录到其中一个控制器节点，刷新 Pacemaker 中所有停止的服务：

```
[heat-admin@overcloud-controller-0 ~]$ for i in `sudo pcs status|grep -
B2 Stop |grep -v "Stop\|Start"|awk -F"[" '/\[/ {print
substr($NF,0,length($NF)-1)}'`; do echo $i; sudo pcs resource cleanup
$i; done
```

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

如果服务失败，在解决相关问题后使用 **pcs resource cleanup** 命令重启它们。

退出 director

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

9.4.5. 完成 L3 代理路由器的配置

对 **overcloudrc** 文件执行 source 命令，从而可以和 overcloud 进行交互。检查您的路由器，确保 overcloud 环境中的 L3 代理正确托管了路由器。本例中使用了名为 **r1** 的路由器：

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

这个命令可能仍然会显示旧节点而不是新节点。要替换它，列出环境中的 L3 网络代理：

```
[stack@director ~]$ neutron agent-list | grep "neutron-l3-agent"
```

找出新节点和旧节点上代理的 UUID。把路由添加到新节点上的代理，并从旧节点上删除。例如：

```
[stack@director ~]$ neutron l3-agent-router-add fd6b3d6e-7d8c-4e1a-
831a-4ec1c9ebb965 r1
[stack@director ~]$ neutron l3-agent-router-remove b40020af-c6dd-4f7a-
b426-eba7bac9dbc2 r1
```

-

在路由器上进行最后的检查，确认所有都已激活：

```
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

删除那些指向旧的 Controller 节点的 Neutron 代理。例如：

```
[stack@director ~]$ neutron agent-list -F id -F host | grep overcloud-
controller-1
| ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb | overcloud-controller-
1.localdomain |
[stack@director ~]$ neutron agent-delete ddae8e46-3e8e-4a1b-a8b3-
c87f13c294eb
```

9.4.6. 完成 Compute 服务配置

已删除节点的计算服务仍然存在于 overcloud 中，需要删除。对 **overcloudrc** 文件执行 source 命令，从而可以和 overcloud 进行交互。检查已删除节点的计算服务：

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ nova service-list | grep "overcloud-controller-
1.localdomain"
```

删除节点的 compute 服务。例如，**overcloud-controller-1.localdomain** 节点的 **nova-scheduler** 服务的 ID 是 5，运行以下命令：

```
[stack@director ~]$ nova service-delete 5
```

针对被删除节点的每个服务执行这个操作。

在新节点上检查 **openstack-nova-consoleauth** 服务。

```
[stack@director ~]$ nova service-list | grep consoleauth
```

如果服务没有运行，登录到一个 Controller 节点并重启服务：

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-
nova-consoleauth
```

9.4.7. 结果

失败的 Controller 节点和它的相关服务被一个新节点替代。

重要

如果禁用了 Object Storage 的自动 ring 构建（如 第 9.6 节 “替换 Object 存储节点”），则需要手工为新节点构建 Object Storage ring 文件。如需了解更多与手工构建 ring 文件相关的信息，请参阅 第 9.6 节 “替换 Object 存储节点”。

9.5. 替换 CEPH 存储节点

director 提供了一个在 director 创建的集群中替换 Ceph 存储节点的方法。相关说明可通过适用于 overcloud 的 Red Hat Ceph Storage找到。

9.6. 替换 OBJECT 存储节点

本节介绍了如何在保持集群完整性的情况下替换 Object Storage 节点。在这个示例中，有一个带有 2 个节点 的 Object Storage 集群，它需要更换 **overcloud-objectstorage-1** 节点。我们需要添加一个额外的节点，然后移除 **overcloud-objectstorage-1**（实际上是替换它）。

1. 使用以下内容，创建一个名为 ~/templates/swift-upscale.yaml 的环境文件：

   ```
   parameter_defaults:
     ObjectStorageCount: 3
   ```

   ObjectStorageCount 定义了环境中存在的 Object Storage 节点数量。在这种情况下，我们将节点从 2 个扩展到 3 个。

2. 在 **openstack overcloud deploy** 命令中包含 **swift-upscale.yaml** 文件及 overcloud 的其余环境文件 (*ENVIRONMENT_FILES*)：

   ```
   $ openstack overcloud deploy --templates ENVIRONMENT_FILES -e
   swift-upscale.yaml
   ```

   注意

   将 **swift-upscale.yaml** 添加到环境文件列表的最后，从而使它的参数可以覆盖前面的环境文件参数。

   重新部署完成后，overcloud 中现在包含一个额外的 Object Storage 节点。

3. 现在需要将数据复制到新节点中。移除节点（在本例中，为 **overcloud-objectstorage-1**）之前，应等待 *replication pass* 在新节点操作完成。您可以通过 **/var/log/swift/swift.log** 查看复制传递的进度。传递完成后，Object Storage 服务应记录类似以下的条目：

   ```
   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. 为了从 ring 中删除旧节点，减少 **swift-upscale.yaml** 中 **ObjectStorageCount** 的值。在这个示例中，我们把它减为 2：

   ```
   parameter_defaults:
     ObjectStorageCount: 2
   ```

5. 创建一个新环境文件，命名为 **remove-object-node.yaml**。该文件将确认和移除指定的 Object Storage 节点。以下内容指定了 **overcloud-objectstorage-1** 的移除：

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

6. 在部署命令中包括这两个环境文件：

```
$ openstack overcloud deploy --templates ENVIRONMENT_FILES -e
swift-upscale.yaml -e remove-object-node.yaml ...
```

director 从 overcloud 中删除对象存储节点，并更新 overcloud 中的其他节点来使删除生效。

第 10 章 重新引导节点

某些情况要求在 undercloud 和 overcloud 中重新引导节点。以下流程介绍了重新引导不同节点类型的方法。请注意以下几点：

- 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。这有助于在重新引导期间保持该角色的服务。
- 如果在您的 OpenStack Platform 环境中重新引导所有节点，请按照以下列表给出的顺序进行重新引导：

建议的节点重新引导顺序

1. 重新引导 director
2. 重新引导 Controller 节点
3. 重新引导 Ceph Storage 节点
4. 重新引导 Compute 节点
5. 重新引导 Object Storage 节点

10.1. 重新引导 DIRECTOR

要重新引导 director 节点，请遵循此流程：

1. 重新引导节点：

   ```
   $ sudo reboot
   ```

2. 稍等片刻，直到节点启动。

节点启动时，检查所有服务的状态：

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch*"
```

注意

重新引导之后，**`openstack-nova-compute`** 可能大约需要 10 分钟的时间才能生效。

确认 overcloud 及其节点是否存在：

```
$ source ~/stackrc
$ openstack server list
$ openstack baremetal node list
$ openstack stack list
```

10.2. 重新引导 CONTROLLER 节点

要重新引导 Controller 节点，请遵循此流程：

1. 选择要重新引导的节点。登录到该节点，并重新引导：

   ```
   $ sudo reboot
   ```

   集群中剩余的 Controller 节点在重新引导期间保持高可用性服务。

2. 稍等片刻，直到节点启动。

3. 登录到节点，并检查集群的状态：

   ```
   $ sudo pcs status
   ```

   该节点重新加入到集群中。

   注意

   重新引导之后，如果有服务失败，请运行 sudo **pcs resource cleanup** 清除错误，并将各资源的状态设置为 **Started**。如果错误继续存在，请联系红帽获取指导和帮助。

4. 确认 Controller 节点上的所有 **systemd** 服务都有效：

   ```
   $ sudo systemctl list-units "openstack*" "neutron*"
   "openvswitch*"
   ```

5. 注销节点，选择下一个要重新引导的 Controller 节点，然后重复此流程，直至您已重新引导所有 Controller 节点。

10.3. 重新引导 CEPH STORAGE 节点

要重新引导 Ceph Storage 节点，请遵循此流程：

1. 登录到 Ceph MON 或 Controller 节点，然后暂时禁用 Ceph 存储集群重新平衡：

   ```
   $ sudo ceph osd set noout
   $ sudo ceph osd set norebalance
   ```

2. 选择要重新引导的首个 Ceph Storage 节点，然后登录该节点。

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，并检查集群的状态：

   ```
   $ sudo ceph -s
   ```

   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。重复此流程，直到您已重新引导所有 Ceph 存储节点。

7. 完成之后，登录 Ceph MON 或 Controller 节点，然后重新启用集群重新平衡：

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

8. 执行最后的状态检查，确认集群报告 **HEALTH_OK**：

```
$ sudo ceph status
```

10.4. 重新引导 COMPUTE 节点

单独重新引导各 Compute 节点，确保您的 OpenStack Platform 环境中实例出现停机的时间为零。此操作涉及以下工作流：

1. 选择要重新引导的 Compute 节点
2. 将其实例迁移到另一个 Compute 节点中
3. 重新引导空白 Compute 节点

列出所有的 Compute 节点及其 UUID：

```
$ nova list | grep "compute"
```

选择要重新引导的 Compute 节点，然后先按照以下流程迁移其实例：

1. 从 undercloud 中选择要重新引导的 Compute 节点，然后将其禁用：

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
```

2. 列出 Compute 节点上的所有实例：

```
$ openstack server list --host [hostname] --all-projects
```

3. 从禁用的主机上迁移每个实例。使用以下任意一个命令：

    a. 将实例迁移至您选择的特定主机：

    ```
    $ openstack server migrate [instance-id] --live [target-
    host]--wait
    ```

    b. 让 **nova-scheduler** 自动选择目标主机：

    ```
    $ nova live-migration [instance-id]
    ```

> **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。

4. 稍等片刻，直至迁移完成。
5. 确认实例已从 Compute 节点被迁移：

```
$ openstack server list --host [hostname] --all-projects
```

6. 重复此步骤，直至从 Compute 节点迁移完所有实例：

> **重要**
>
> 有关配置和迁移实例的完整说明，请参阅???。

按照以下流程，重新引导 Compute 节点

1. 登录到 Compute 节点并重新引导该节点：

```
$ sudo reboot
```

2. 稍等片刻，直到节点启动。
3. 重新启用 Compute 节点：

```
$ source ~/overcloudrc
$ openstack compute service set [hostname] nova-compute --enable
```

4. 确认是否已启用 Compute 节点：

```
$ openstack compute service list
```

10.5. 重新引导 OBJECT STORAGE 节点

要重新引导 Object Storage 节点，请遵循此流程：

1. 选择要重新引导的 Object Storage 节点。登录到节点，然后重新引导：

```
$ sudo reboot
```

2. 稍等片刻，直到节点启动。
3. 登录到节点并检查其状态：

```
$ sudo systemctl list-units "openstack-swift*"
```

4. 从节点中登出，然后对下一个 Object Storage 节点重复此流程。

第 11 章 对 DIRECTOR 进行故障排除

在进行 director 操作时可能会在特定阶段出现问题。本节提供了对常见问题进行诊断的信息。

查看 director 组件的日志文件：

- **`/var/log`** 目录包括了许多常见 OpenStack Platform 组件的日志文件，以及标准 Red Hat Enterprise Linux 应用的日志文件。
- **`journald`** 服务为多个组件提供日志功能。ironic 使用两个单元：**`openstack-ironic-api`** 和 **`openstack-ironic-conductor`**。同样的，**`ironic-inspector`** 也使用两个单元：**`openstack-ironic-inspector`** 和 **`openstack-ironic-inspector-dnsmasq`**。以下是使用这个服务的示例：

```
$ sudo journalctl -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
```

- **`ironic-inspector`** 还把 ramdisk 的日志保存在 **`/var/log/ironic-inspector/ramdisk/`** 中（gz 压缩的 tar 文件）。它们的文件名中包括日期、时间和节点的 IPMI 地址。使用这些日志来对相关的服务进行诊断。

11.1. 对节点注册进行故障排除

与节点注册相关的问题通常是因为不正确的节点详情造成的。在这种情况下，使用带有正确节点数据的 **`ironic`** 来解决相关的问题。以下是几个示例：

找到分配的端口 UUID：

```
$ ironic node-port-list [NODE UUID]
```

更新 MAC 地址：

```
$ ironic port-update [PORT UUID] replace address=[NEW MAC]
```

运行以下命令：

```
$ ironic node-update [NODE UUID] replace driver_info/ipmi_address=[NEW IPMI ADDRESS]
```

11.2. 对硬件内省的故障排除

内省（introspection）过程需要被正确完成。但是，如果发现 ramdisk 没有响应，ironic 的发现守护进程（**`ironic-inspector`**）会默认在一个小时后出现超时。在一些时候，这意味着发现 ramdisk 有问题，但通常情况下是因为不正确的环境配置，特别是 BIOS 引导设置。

以下是一些常见的不正确的环境配置情况，以及如果诊断和解决它们的建议。

启动节点内省操作错误

通常，内省操作会使用 **`baremetal introspection`** 命令，它是一个安全调用 Ironic 服务的命令。但是，如果直接使用 **`ironic-inspector`**，它可能在发现状态是 AVAILABLE 的节点过程中出现问题。在进行发现操作前，把节点的状态改为 MANAGEABLE：

```
$ ironic node-set-provision-state [NODE UUID] manage
```

当发现操作完成后，在进行部署前把状态改回到 AVAILABLE：

```
$ ironic node-set-provision-state [NODE UUID] provide
```

内省的节点没有在 PXE 中引导

在重新引导节点前，**`ironic-inspector`** 会把节点的 MAC 地址添加到 undercloud 防火墙的 **`ironic-inspector`** 链中。这将允许节点通过 PXE 进行引导。运行以下命令可以验证配置是否正确：

```
$ `sudo iptables -L`
```

这个命令的输出应该包括以下带有 MAC 地址的链列表：

```
Chain ironic-inspector (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere             MAC
xx:xx:xx:xx:xx:xx
ACCEPT     all  --  anywhere             anywhere
```

如果没有 MAC 地址，最常见的可能是 **`ironic-inspector`** 缓存数据（位于 SQLite 数据库中）被破坏。要解决这个问题，删除 SQLite 文件：

```
$ sudo rm /var/lib/ironic-inspector/inspector.sqlite
```

再重新创建它：

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-
inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

停止发现过程

当前，**`ironic-inspector`** 没有提供一个直接停止发现过程的方法。我们推荐的方法是等待发现过程的超时发生。如果需要，可以修改 **`/etc/ironic-inspector/inspector.conf`** 中的 **`timeout`** 设置来设定一个新的超时时间（以分钟为单位）。

在最坏的情况下，您可以使用以下步骤停止所有节点的发现操作：

把每个节点的电源状态改为 off：

```
$ ironic node-set-power-state [NODE UUID] off
```

删除 **`ironic-inspector`** 的缓存数据并重启它：

```
$ rm /var/lib/ironic-inspector/inspector.sqlite
```

重新同步 **`ironic-inspector`** 缓存：

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-
inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

检查内省存储

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：

```
$ sudo systemctl list-units openstack-swift*
```

11.3. 故障排除工作流和执行

OpenStack Workflow（mistral）服务将多项 OpenStack 任务组合成工作流。Red Hat OpenStack Platform 使用一套这样的工作流来执行 CLI 和 Web UI 中的常见功能。其中包括裸机节点控制、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
...
```

工作流对象

OpenStack Workflow 使用以下对象来跟踪工作流：

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。

工作流错误诊断

OpenStack Workflow 还提供了强大的执行日志记录，帮助您辨别与特定命令失败相关的问题。例如，如果某一工作流执行失败，您可以确定其故障点。列出具有已失败状态 **ERROR** 的工作流执行记录：

```
$ mistral execution-list | grep "ERROR"
```

获取已失败工作流执行记录的 UUID（如 3c87a885-0d37-4af8-a471-1b392264a7f5），查看执行情况及其输出：

```
$ mistral execution-get 3c87a885-0d37-4af8-a471-1b392264a7f5
$ mistral execution-get-output 3c87a885-0d37-4af8-a471-1b392264a7f5
```

这可提供执行中已失败任务的相关信息。**mistral execution-get** 也会显示用于该执行的工作流（例如，**tripleo.plan_management.v1.update_deployment_plan**）。您可以使用以下命令查看完整的工作流定义：

```
$ mistral execution-get-definition
tripleo.plan_management.v1.update_deployment_plan
```

这可用于辨别特定任务在工作流中的位置。

您也可以使用类似的命令语法查看操作执行及其结果：

```
$ mistral action-execution-list
$ mistral action-execution-get b59245bf-7183-4fcf-9508-c83ec1a26908
$ mistral action-execution-get-output b59245bf-7183-4fcf-9508-
c83ec1a26908
```

这可用于辨别导致问题的具体操作。

11.4. 对创建 OVERCLOUD 进行故障排除

实施的过程可能会在 3 个层面出现问题：

- 编配（heat 和 nova 服务）
- 裸机部署（ironic 服务）
- 实施后的配置（Puppet）

如果 overcloud 的部署在以上任何层面中出现问题，可使用 OpenStack 客户端和服务日志文件来诊断失败的部署。

11.4.1. 编配

在多数情况下，Heat 会在 overcloud 创建失败后显示出现问题的 overcloud 栈：

```
$ heat stack-list
+-----------------------+------------+--------------------+--------
--------------+
| id                    | stack_name | stack_status       |
creation_time        |
+-----------------------+------------+--------------------+--------
--------------+
| 7e88af95-535c-4a55... | overcloud  | CREATE_FAILED      | 2015-04-
06T17:57:16Z |
+-----------------------+------------+--------------------+--------
--------------+
```

如果栈列表为空，这意味着出现的问题与初始的 Heat 设置相关。检查您的 Heat 模板和配置选项，并检查在运行 **openstack overcloud deploy** 后的错误信息。

11.4.2. 裸机部署

使用 **ironic** 查看所有注册的节点和它们当前的状态：

```
$ ironic 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       |
+----------+------+---------------+-------------+-----------------+-
------------+
```

以下是一些在部署过程中常见的问题。

- 在结果列表中检查 Provision State 和 Maintenance 栏中的数据。检查以下情况：
  - 结果列表为空，或比期望的节点要少
  - Maintenance 被设置为 True
  - Provision State 被设置为 **manageable**。这通常意味着问题是由注册或发现过程造成的。例如，如果 Maintenance 被自动设置为 True，这通常是因为节点使用了错误的电源管理凭证。
- 如果 Provision State 的值是 **available**，这意味着问题发生在裸机部署开始前。
- 如果 Provision State 的值是 **active**，Power State 的值是 **power on**，这意味着裸机部署已成功完成，所出现的问题发生在实施后的配置阶段。
- 如果一个节点的 Provision State 值是 **wait call-back**，这意味着对这个节点的裸机部署还没有完成。等待这个状态改变；或连接到出现问题的节点的虚拟控制台上检查相关的输出。

- 如果 Provision State 的值是 **error** 或 **deploy failed**，则意味着对这个节点的裸机部署失败。检查裸机节点的详情：

  ```
  $ ironic node-show [NODE UUID]
  ```

  查看包括错误描述信息的 **last_error** 项。如果错误信息不明确，您可以查看相应的日志：

  ```
  $ sudo journalctl -u openstack-ironic-conductor -u openstack-ironic-api
  ```

- 如果您看到 **wait timeout error** 信息，节点的 Power State 值是 **power on**，连接到出现问题的节点的虚拟控制台上检查相关的输出。

11.4.3. 实施后的配置

在配置阶段会发生许多事情。例如，特定的 Puppet 模块可能会因为设置的问题出错。本小节提供了诊断相关问题的方法。

列出 overcloud 栈中的所有资源来找出哪个出现了问题：

```
$ heat resource-list overcloud
```

这会显示一个包括所有资源以及它们的状态的列表。查看带有 **CREATE_FAILED** 的资源。

显示出问题的资源：

```
$ heat resource-show overcloud [FAILED RESOURCE]
```

查看 **resource_status_reason** 项中的内容来帮助进行诊断。

使用 **nova** 命令查看 overcloud 节点的 IP 地址。

```
$ nova list
```

以 **heat-admin** 用户身份登录到一个实施的节点上。例如，栈资源列表显示一个 Controller 节点出现问题，您可以登录到那个 Controller 节点。**heat-admin** 用户有 sudo 访问权限。

```
$ ssh heat-admin@192.168.24.14
```

检查 **os-collect-config** 日志找出可能造成故障的原因。

```
$ sudo journalctl -u os-collect-config
```

在某些情况下，会出现 nova 未完整部署节点的情况。在这种情况下，一个 overcloud 角色类型的 **OS::Heat::ResourceGroup** 会出错。这时，可使用 **nova** 来查看问题。

```
$ nova list
$ nova show [SERVER ID]
```

多数常见错误会显示 **No valid host was found** 错误信息，请参阅 第 11.6 节 “对 "No Valid Host Found" 错误进行故障排除” 来获得更多与排除这类错误相关的信息。在其它情况下，查看以下日志文件来进行进一步的故障排除：

- **/var/log/nova/***
- **/var/log/heat/***
- **/var/log/ironic/***

使用 SOS 工具包来收集系统硬件和配置的信息。这些信息可用于进行故障排除和调试。SOS 受到技术支持和开发人员的广泛采用。SOS 在 undercloud 和 overcloud 中都有用。安装 **sos** 软件包：

```
$ sudo yum install sos
```

产生一个报告

```
$ sudo sosreport --all-logs
```

控制器节点部署后的过程由 5 个主要部署步骤组成：

表 11.1. Controller 节点配置步骤

步骤	描述
ControllerDeployment_Step1	初始负载均衡软件配置，包括 Pacemaker、RabbitMQ、Memcached、Redis 和 Galera。
ControllerDeployment_Step2	初始集群配置，包括 Pacemaker 配置、HAProxy、MongoDB、Galera、Ceph Monitor，以及 OpenStack Platform 服务的数据库初始化。
ControllerDeployment_Step3	初始构建 OpenStack 对象存储 (**swift**) 的 ring。配置所有 OpenStack Platform 服务（**nova**、**neutron**、**cinder**、**sahara**、**ceilometer**、**heat**、**horizon**、**aodh** 和 **gnocchi**）。
ControllerDeployment_Step4	在 Pacemaker 中配置服务启动设置，包括决定服务启动顺序的限制，以及服务启动的参数。
ControllerDeployment_Step5	初始配置 OpenStack Identity (**keystone**) 中的项目、角色和用户。

11.5. 排除 PROVISIONING NETWORK 中出现的 IP 地址冲突的问题

当目标主机被分配了一个已在使用的 IP 地址时，发现和部署任务将会失败。为了避免这个问题，可以对 Provisioning 网络进行一个端口扫描，从而决定发现的 IP 范围和主机的 IP 范围是否可用。

在 undercloud 主机上执行以下步骤：

安装 **nmap**:

```
# yum install nmap
```

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：

```
# nmap -sn 192.168.24.0/24
```

检查 **nmap** 扫描的结果输出：

例如，您应该看到 undercloud 的 IP 地址，以及该子网中的任何其他主机。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，则需要在内省或部署overcloud 节点前修改 IP 地址范围或释放一些 IP 地址。

```
# 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
```

11.6. 对 "NO VALID HOST FOUND" 错误进行故障排除

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.
```

这意味着 nova Scheduler 无法找到合适的裸机节点来引导新的实例。造成这个问题的原因通常是 nova 所期望的资源和 Ironic 通知给 Nova 的资源不匹配。检查以下内容：

1. 确保内省可以成功完成。否则，检查每个节点都包括了需要的 ironic 节点属性。对于每个节点：

   ```
   $ ironic 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。

2. 根据 ironic 节点属性检查使用的 nova flavor 没有超过特定数量：

   ```
   $ nova flavor-show [FLAVOR NAME]
   ```

3. 根据 **ironic node-list** 检查有足够状态为 **available** 的节点。节点的状态为 **manageable** 通常意味着内省操作失败。

4. 使用 **ironic node-list** 检查没有处于维护模式的节点。当一个节点被自动变为维护模式时，通常意味着不正确的电源管理凭证。检查它们并删除维护模式：

```
$ ironic node-set-maintenance [NODE UUID] off
```

5. 如果您使用 AHC 工具程序来自动标记节点，请根据每个 flavor 和配置集来检查是否有足够的相关节点。检查 **ironic node-show** 输出中的 **properties** 项的 **capabilities** 值。例如，一个标记为 Compute 角色的节点应该包括 **profile:compute**。

6. 在进行完内省操作后，从 ironic 为 nova 生成节点信息可能会需要一些时间来完成。director 的工具程序通常会把这个时间考虑进去。但是，如果您手工进行了一些操作，节点可能会在一个短时间内对 nova 无效。使用以下命令检查系统中的总资源：

```
$ nova hypervisor-stats
```

11.7. 在创建后对 OVERCLOUD 进行故障排除

在创建完 overcloud 后，可能还会需要在以后执行特定的 overcloud 操作。例如，可能会需要扩展可用节点，或替换出现故障的节点。在执行这些操作时，可能会出现某些问题。本节就如何对出现失败的创建后操作进行诊断和故障排除提供一些建议。

11.7.1. overcloud 栈的修改

当通过 director 修改 **overcloud** 栈时可能会出现问题。对栈进行修改可能包括：

- 扩展节点
- 删除节点
- 替换节点

修改栈的过程和创建栈的过程相似，director 会检查请求的节点数是否有效，部署额外的节点或删除存在的节点，然后应用 Puppet 配置。在修改 **overcloud** 栈时需要遵循以下的一些建议。

在初始设置时，遵循 第 11.4.3 节 “实施后的配置” 中的建议。这些相同的步骤可以帮助排除更新 **overcloud** heat 栈时出现的问题。特别是，使用以下命令帮助查找有问题的资源：

heat stack-list --show-nested

列出所有栈。**--show-nested** 会显示所有子栈以及它们的父栈。这可以帮助判断栈在什么地方出现问题。

heat resource-list overcloud

列出 **overcloud** 栈中的所有资源，以及它们当前的状态。这可以帮助找出哪些资源造成了栈出现问题。您可以通过这些失败的资源追踪到 heat 模板集合和 Puppet 模块中的相关参数和配置。

heat event-list overcloud

以发生的时间顺序列出与 **overcloud** 栈相关的所有事件。这包括初始化事件、操作完成事件以及栈中所有失败的资源。这些信息可以帮助找出造成资源失败的原因。

下面的几个小节介绍了针对特定节点类型的故障诊断建议。

11.7.2. Controller 服务失败

overcloud 控制器节点包括大量 Red Hat OpenStack Platform 服务，您也可能在一个高可用性的集群中使用多个控制器节点。如果一个节点上的特定服务出现问题，高可用性集群会提供一定程度的故障转移功能。但是，您需要对出现问题的服务进行诊断，从而确保 overcloud 能以最大能力运行。

在高可用性集群中，控制器节点使用 Pacemaker 管理资源和服务。Pacemaker Configuration System（**pcs**）命令是一个用来管理 Pacemaker 集群的工具。在集群的控制器节点上运行这个命令来执行配置和监控功能。在高可用性集群中，可以使用以下命令帮助对 overcloud 服务进行故障排除：

pcs status

当前整个集群的状态概况信息，包括启用的资源、失败的资源和在线节点信息。

pcs resource show

显示资源列表，以及与它们相关的节点。

pcs resource disable [resource]

停止一个特定的资源。

pcs resource enable [resource]

启动一个特定的资源。

pcs cluster standby [node]

把节点设置为待机（standby）模式，使这个节点在集群中不再可用。这可以被用来在不影响集群运行的情况下对特定节点进行维护操作。

pcs cluster unstandby [node]

取消一个节点的待机模式。这个节点将可以重新在集群中使用。

使用这些 Pacemaker 命令来找出有问题的组件和节点。当找到有问题的组件时，在 **/var/log/** 中查看相关的组件日志文件。

11.7.3. Compute 服务失败

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。例如：

- 使用 **systemd** 的以下功能查看服务的状态：

  ```
  $ sudo systemctl status openstack-nova-compute.service
  ```

  同样，使用以下命令查看服务的 **systemd** 日志：

  ```
  $ sudo journalctl -u openstack-nova-compute.service
  ```

- Compute 节点的主日志文件是 **/var/log/nova/nova-compute.log**。如果到 Compute 节点的通讯出现问题，从这个文件开始进行故障排除会是一个好的方法。

- 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。如需了解更多节点迁移的信息，请参阅 ???。

11.7.4. Ceph Storage 服务故障

如果 Red Hat Ceph Storage 集群出现故障，参阅“Red Hat Ceph Storage 配置指南”中的 第 10 章：日志和调试。本小节提供与所有 Ceph 存储服务的日志诊断相关的信息。

11.8. 对 UNDERCLOUD 进行性能微调

本节中提供的建议旨在帮助提高 undercloud 的性能。您可以根据自己的需要实施相关的建议。

- OpenStack Authentication 服务（**keystone**）使用一个基于令牌（token）的系统控制对其它 OpenStack 服务的访问。在运行了一定时间后，数据库中会积累大量不再使用的令牌。我们推荐创建一个 cronjob 来定期删除数据库中的令牌表。例如，在每天早上 4 点删除令牌表：

```
0 04 * * * /bin/keystone-manage token_flush
```

- 在每次运行 **openstack overcloud deploy** 命令时，Heat 会把所有模板文件复制到它的数据库中的 **raw_template** 表中。**raw_template** 表会包括过去所有的模板，并随着时间的推移变得非常大。您可以创建一个每日运行的 cronjob 来删除 **raw_templates** 表中那些不再使用的、存在时间超过一天的模板：

```
0 04 * * * /bin/heat-manage purge_deleted -g days 1
```

- 在一些时候，**openstack-heat-engine** 和 **openstack-heat-api** 服务可能会消耗大量资源。如果这个情况发生了，在 **/etc/heat/heat.conf** 中设置 **max_resources_per_stack=-1**，然后重启 heat 服务：

```
$ sudo systemctl restart openstack-heat-engine openstack-heat-api
```

- 在一些时候，director 可能没有足够的资源来执行并行的节点设置（默认是可以同时并行设置 10 个节点）。为了减少并行节点的数量，把 **/etc/nova/nova.conf** 中的 **max_concurrent_builds** 参数设置为一个小于 10 的值，然后重启 nova 服务：

```
$ sudo systemctl restart openstack-nova-api openstack-nova-scheduler
```

- 编辑 **/etc/my.cnf.d/server.cnf** 文件。可以用来微调的值包括：

max_connections

可以同时连接到数据库的连接数量。推荐的值是 4096。

innodb_additional_mem_pool_size

数据库用来存储数据字典信息和其他内部数据结构的内存池大小（以字节为单位）。默认值是 8M，undercloud 的理想值是 20M。

innodb_buffer_pool_size

数据库用来缓存表和索引数据的内存区域（即缓冲池）的大小（以字节为单位）。默认值是 128M，undercloud 的理想值是 1000M。

innodb_flush_log_at_trx_commit

commit 操作对 ACID 原则的遵守，与高性能间的平衡控制。把它设置为 1。

innodb_lock_wait_timeout

数据库交易在放弃等待行锁定前等待的时间长度（以秒为单位）。把它设置为 50。

innodb_max_purge_lag

在 purge 操作滞后时，如何延迟 INSERT、UPDATE 和 DELETE 操作。把它设为 10000。

innodb_thread_concurrency

并行操作系统线程的限制。理想情况下，为每个 CPU 和磁盘资源最少提供 2 个线程。例如，具有一个 4 核 CPU 和一个磁盘，则提供 10 个线程。

- 确保 heat 有足够的 worker 来执行 overcloud 创建。通常情况下，这取决于 undercloud 有多少个 CPU。若要手动设置 worker 的数量，可编辑 **/etc/heat/heat.conf** 文件，把 **num_engine_workers** 参数的值设置为所需的 worker 数量（理想值是 4），然后重启 heat 引擎：

```
$ sudo systemctl restart openstack-heat-engine
```

11.9. UNDERCLOUD 和 OVERCLOUD 的重要日志

在故障排除时，使用以下日志查找 undercloud 和 overcloud 的信息。

表 11.2. undercloud 的重要日志

信息	日志位置
OpenStack Compute 日志	**/var/log/nova/nova-compute.log**
OpenStack Compute API 交互	**/var/log/nova/nova-api.log**
OpenStack Compute Conductor 日志	**/var/log/nova/nova-conductor.log**
OpenStack Orchestration 日志	**heat-engine.log**
OpenStack Orchestration API 交互	**heat-api.log**
OpenStack Orchestration CloudFormations 日志	**/var/log/heat/heat-api-cfn.log**

信息	日志位置
OpenStack Bare Metal Conductor 日志	**ironic-conductor.log**
OpenStack Bare Metal API 交互	**ironic-api.log**
内省（Introspection）	**/var/log/ironic-inspector/ironic-inspector.log**
OpenStack Workflow Engine 日志	**/var/log/mistral/engine.log**
OpenStack Workflow Executor 日志	**/var/log/mistral/executor.log**
OpenStack Workflow API 交互	**/var/log/mistral/api.log**

表 11.3. overcloud 的重要日志

信息	日志位置
Cloud-Init 日志	**/var/log/cloud-init.log**
overcloud 配置（最后一次 Puppet 运行的概述）	**/var/lib/puppet/state/last_run_summary.yaml**
overcloud 配置（最后一次 Puppet 运行的报告）	**/var/lib/puppet/state/last_run_report.yaml**
overcloud 配置（所有 Puppet 报告）	**/var/lib/puppet/reports/overcloud-*/***
overcloud 配置（来自每一次 Puppet 运行的 stdout）	**/var/run/heat-config/deployed/*-stdout.log**
overcloud 配置（来自每一次 Puppet 运行的 stderr）	**/var/run/heat-config/deployed/*-stderr.log**

信息	日志位置
高可用性日志	**`/var/log/pacemaker.log`**

附录 A. SSL/TLS 证书配置

您可以将 undercloud 配置为使用 SSL/TLS 进行公共端点上的通信。但是，如果使用自有证书认证机构颁发的 SSL 证书，该证书需要参考下一节中的步骤进行配置。

注意

有关 overcloud SSL/TLS 证书的创建流程，请参阅 *overcloud 高级自定义*指南中的“在 overcloud 中启用 SSL/TLS”部分。

A.1. 初始化签名主机

签名主机是生成新证书并通过证书认证机构进行签名的主机。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。

`/etc/pki/CA/index.txt` 文件存储所有已签名证书的记录。检查是否存在此文件。如果不存在，请创建一个空文件：

```
$ sudo touch /etc/pki/CA/index.txt
```

`/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。如果不存在，请使用新的起始值创建一个新文件：

```
$ sudo echo '1000' | sudo tee /etc/pki/CA/serial
```

A.2. 创建一个证书认证机构（CA）

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。在一些情况下，您可能需要使用自己的证书认证机构。例如，您希望创建一个只对内部有效的证书认证机构。

创建一个密钥和证书对来作为证书认证机构：

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca
-out ca.crt.pem
```

`openssl req` 命令会要求输入认证机构的详细信息。根据提示输入所需信息。

这会创建一个名为 **`ca.crt.pem`** 的证书认证机构文件。

A.3. 把证书认证机构添加到客户端中

对于任何需要使用 SSL/TLS 进行通信的外部客户端，将证书认证机构文件复制到所有需要访问 Red Hat OpenStack Platform 环境的客户端上。在复制完成后，在客户端上运行以下命令，将该文件加入到证书认证机构信任套件中：

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

A.4. 创建一个 SSL/TLS 密钥

A.4. 创建一个 SSL/TLS 密钥

运行以下命令以生成 SSL/TLS 密钥（**server.key.pem**）。我们可以在不同地方使用它来生成自己的 undercloud 或 overcloud 证书：

```
$ openssl genrsa -out server.key.pem 2048
```

A.5. 创建一个 SSL/TLS 证书签发请求

下一步会为 undercloud 或 overcloud 创建一个证书签名请求。

复制默认的 OpenSSL 配置文件用来进行定制。

```
$ cp /etc/pki/tls/openssl.cnf .
```

编辑自定义的 **openssl.cnf** 文件，把 SSL 参数设置为被 director 使用。一个包括相关参数的示例如下：

```
[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** 设置为以下之一：

- 如果使用 IP 地址通过 SSL/TLS 访问，请使用 **undercloud.conf** 中的 **undercloud_public_vip** 参数。
- 如果使用完全限定域名通过 SSL/TLS 访问，则改为使用域名。

编辑 **alt_names** 部分，使其包含以下条目：

- **IP** - 供客户端通过 SSL 访问 director 的 IP 地址列表。
- **DNS** - 供客户端通过 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。

如需了解有关 **openssl.cnf** 的更多信息，请运行 **man openssl.cnf**。

运行以下命令来产生证书签发请求（**server.csr.pem**）：

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
```

确保 **-key** 选项中包括了在 第 A.4 节 “创建一个 SSL/TLS 密钥” 中创建的 SSL/TLS 密钥。

使用 **server.csr.pem** 文件创建 SSL/TLS 证书。

A.6. 创建 SSL/TLS 证书

以下命令为 undercloud 或 overcloud 创建一个证书：

```
$ 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：

- 配置文件来指定 v3 扩展。把它作为 **-config** 选项。
- 第 A.5 节 “创建一个 SSL/TLS 证书签发请求” 中介绍的证书签发请求来产生证书，并通过证书认证机构进行签发。把它作为 **-in** 选项。
- 您在第 A.2 节 “创建一个证书认证机构（CA）”中创建的证书认证机构， 它将为证书签名。请将它包含为 **-cert** 选项。
- 您在第 A.2 节 “创建一个证书认证机构（CA）”中创建的证书认证机构私钥。请将它包含为 **-keyfile** 选项。

这会产生一个名为 **server.crt.pem** 的证书。使用此证书以及在 第 A.4 节 “创建一个 SSL/TLS 密钥” 中创建的 SSL/TLS 密钥来启用 SSL/TLS。

A.7. 在 UNDERCLOUD 中使用证书

运行以下命令来组合证书和密钥：

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

这会创建一个 **undercloud.pem** 文件。在 **undercloud.conf** 中指定这个文件的位置作为 **undercloud_service_certificate** 选项。另外，这个文件还需要一个特殊的 SELinux context，从而使 HAProxy 工具可以读取它。请参照以下示例：

```
$ sudo mkdir /etc/pki/instack-certs
$ sudo cp ~/undercloud.pem /etc/pki/instack-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/instack-certs(/.*)?"
$ sudo restorecon -R /etc/pki/instack-certs
```

把 **undercloud.pem** 文件的位置添加到 **undercloud.conf** 文件的 **undercloud_service_certificate** 选项中。例如：

```
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
```

另外，确保把第 A.2 节 “创建一个证书认证机构（CA）”中创建的证书认证机构添加到 undercloud 的信任证书认证机构列表中，从而使 undercloud 中的不同服务可以访问这个证书认证机构：

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

根据第 4.6 节 “配置 director”中的说明继续安装 undercloud。

附录 B. 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。本附录提供了 director 支持的电源管理功能列表。在 ??? 中都可以使用这些电源管理设置。

B.1. 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

把这个选项设置为 **pxe_drac**。

pm_user; pm_password

DRAC 的用户名和密码。

pm_addr

DRAC 主机的 IP 地址。

B.2. 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

把这个选项设置为 **pxe_ilo**。

pm_user; pm_password

iLO 的用户名和密码。

pm_addr

iLO 接口的 IP 地址。

- 编辑 **/etc/ironic/ironic.conf** 文件，把 **pxe_ilo** 加入到 **enabled_drivers** 选项来启用这个驱动。
- director 需要为 iLo 安装一组额外的工具程序。安装 **python-proliantutils** 软件包并重启 **openstack-ironic-conductor** 服务：

```
$ sudo yum install python-proliantutils
$ sudo systemctl restart openstack-ironic-conductor.service
```

- 为了成功进行内省，HP 节点必须是 2015 的固件版本。director 已经经过测试可以使用固件版本为 1.85（May 13 2015）的节点。
- 不支持使用共享 iLO 端口。

B.3. IBOOT

Dataprobe 提供的 iBoot 是一个为系统提供远程电源管理的电源单元。

pm_type

把这个选项设置为 **pxe_iboot**。

pm_user; pm_password

iBoot 的用户名和密码。

pm_addr

iBoot 接口的 IP 地址。

pm_relay_id（可选）

iBoot 对于主机的中继 ID。默认值是 1。

pm_port（可选）

iBoot 端口。默认值是 9100。

- 编辑 **/etc/ironic/ironic.conf** 文件，把 **pxe_iboot** 加入到 **enabled_drivers** 选项来启用这个驱动。

B.4. CISCO UNIFIED COMPUTING SYSTEM（UCS）

Cisco 的 UCS 是一个数据中心平台，包括计算、网络、存储访问和虚拟化资源。这个驱动提供了对连接到 UCS 上的裸机系统的电源管理功能。

pm_type

把这个选项设置为 **pxe_ucs**。

pm_user; pm_password

UCS 的用户名和密码。

pm_addr

UCS 接口的 IP 地址。

pm_service_profile

使用的 UCS 服务配置集。通常的格式是 **org-root/ls-[service_profile_name]**。例如：

```
"pm_service_profile": "org-root/ls-Nova-1"
```

- 编辑 **/etc/ironic/ironic.conf** 文件，把 **pxe_ucs** 加入到 **enabled_drivers** 选项来启用这个驱动。
- director 需要为 UCS 安装一组额外的工具程序。安装 **python-UcsSdk** 软件包并重启 **openstack-ironic-conductor** 服务：

```
$ sudo yum install python-UcsSdk
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.5. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu 的 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
: 把选项设置为 **`pxe_irmc`**。

pm_user; pm_password
: iRMC 接口的用户名和密码。

pm_addr
: iRMC 接口的 IP 地址。

pm_port（可选）
: iRMC 操作使用的端口。默认值是 443。

pm_auth_method（可选）
: iRMC 操作的验证方法。使用 **`basic`** 或 **`digest`**。默认值是 **`basic`**

pm_client_timeout（可选）
: iRMC 操作的超时值（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`**。

- 编辑 **`/etc/ironic/ironic.conf`** 文件，把 **`pxe_irmc`** 加入到 **`enabled_drivers`** 选项来启用这个驱动。
- 如果使用 SCCI 作为获得感应器数据的方法，则 director 还需要安装一组额外的工具程序。安装 **`python-scciclient`** 软件包并重启 **`openstack-ironic-conductor`** 服务：

```
$ yum install python-scciclient
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.6. SSH 和 VIRSH

director 可以通过 SSH 访问运行 libvirt 的主机。director 使用 virsh 控制这些节点的电源管理。

重要

这个选项当前只用于测试和评估，我们不推荐在 Red Hat OpenStack Platform 企业级环境中使用它。

pm_type

把这个选项设置为 **pxe_ssh**。

pm_user; pm_password

SSH 的用户名和 SSH 私钥的内容。如果使用 CLI 工具来注册节点，私钥必须在一行中，其中的新行以 转义字符 (**\n**) 替代。例如：

```
-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEA ....
kk+WXt9Y=\n-----END RSA PRIVATE KEY-----
```

把 SSH 公共密钥添加到 libvirt 服务器的 **authorized_keys** 集合中。

pm_addr

virsh 主机的 IP 地址。

- 运行 libvirt 的服务器需要一个带有在 **pm_password** 属性中设置的公共密钥的 SSH 密钥对。
- 确保所选的 **pm_user** 可以完全访问 libvirt 环境。

B.7. FAKE PXE 驱动

这个驱动提供了一个在没有电源管理的情况下使用裸机的方法。这意味着，director 不控制注册的裸机设备，而是在内省以及部署过程的特定点上手工控制电源。

重要

这个选项当前只用于测试和评估，我们不推荐在 Red Hat OpenStack Platform 企业级环境中使用它。

pm_type

把这个选项设置为 **fake_pxe**。

- 这个驱动不使用任何验证信息，因为它不控制电源管理。
- 编辑 **/etc/ironic/ironic.conf** 文件，把 **fake_pxe** 加入到 **enabled_drivers** 选项来启用这个驱动。在编辑这个文件后重启 baremetal 服务：

```
$ sudo systemctl restart openstack-ironic-api openstack-
ironic-conductor
```

- 在节点上执行内省（introspection）操作时，运行完 **openstack baremetal introspection bulk start** 命令后需要手工启动节点。
- 在进行 overcloud 部署时，使用 **ironic node-list** 命令检查节点的状态。等待节点状态由 **deploying** 变为 **deploy wait-callback** 后，再手动启动这个节点。
- 在 overcloud 部署完成后，重启节点。使用 **ironic node-list** 命令来检查节点的状态，确定部署过程是否已完成。部署完成后，节点状态会变为 **active**。然后，手动重启所有 overcloud 节点。

附录 C. 完整的磁盘镜像

主要的 overcloud 镜像是一个平面分区镜像。这表示镜像本身不包含分区信息或引导加载程序。director 在引导时使用单独的内核和 ramdisk，在将 overcloud 镜像写入磁盘时创建一个基本的分区布局。但是，您可以创建包含分区布局和引导加载程序的完整磁盘镜像。

C.1. 创建完整的磁盘镜像

从 **overcloud-full.qcow2** 平面分区镜像创建完整的磁盘镜像包含以下步骤：

1. 打开 **overcloud-full** 平面分区，这是完整磁盘镜像的基础。
2. 按照所需的尺寸大小，创建完整的新磁盘镜像。本例使用 10 GB 镜像。
3. 在完整的磁盘镜像上创建分区和卷。根据完整磁盘镜像所需的数量创建分区和卷。本例为 **boot** 创建了一个隔离分区，为文件系统中的其他内容创建了逻辑卷。
4. 在分区和卷上创建初始文件系统。
5. 安装平面分区文件系统，将内容复制到完整磁盘镜像的适当分区上。
6. 生成 **fstab** 内容，并将其保存到完整磁盘镜像的 **/etc/fstab** 中。
7. 卸载所有文件系统。
8. 仅在完整磁盘镜像上挂载分区。先将根分区挂载到 **/** 中，然后将其他分区挂载到相应的目录中。
9. 使用 shell 命令安装引导加载程序，以便在完整磁盘镜像上执行 **grub2-install** 和 **grub2-mkconfig**。此操作会将 **grub2** 引导加载程序安装到完整磁盘镜像中。
10. 更新 **dracut**，以便支持逻辑卷管理
11. 卸载所有文件系统，然后关闭镜像

手动创建一个完整的磁盘镜像

建议使用 **guestfish** 工具创建镜像，运行以下命令可安装此工具：

```
$ sudo yum install -y guestfish
```

安装完成后，运行 **guestfish** 交互式 shell：

```
$ guestfish

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

><fs>
```

有关使用 **guestfish** 的更多信息，请参阅 Red Hat Enterprise Linux 7 *虚拟部署和管理指南*中的 "The Guestfish Shell"。

C.2. 自动创建一个完整的磁盘镜像

以下 Python 脚本使用 **guestfish** 库来自动编写镜像创建脚本。该脚本根据 **logical_volumes**（Python 格式的词典）中的信息创建一系列卷。该词典中的键表示待创建的卷，它的值表示卷的大小。

```
#!/usr/bin/env python
import guestfs
import os

logical_volumes = {
  'root': 4 * 1024,
  'swap': 250,
  'home': 500,
  'tmp':  250,
  'var':  5 * 1024
}

# remove old generated drive
try:
  os.unlink('/home/stack/images/overcloud-full-partitioned.qcow2')
except:
  pass

g = guestfs.GuestFS(python_return_dict=True)

# import old and new images
print('Creating new repartitioned image')
g.add_drive_opts('/home/stack/images/overcloud-full.qcow2',
format='qcow2', readonly=1)
g.disk_create('/home/stack/images/overcloud-full-partitioned.qcow2',
'qcow2', 12 * 1024 * 1024 * 1024)
g.add_drive_opts('/home/stack/images/overcloud-full-partitioned.qcow2',
format='qcow2', readonly=0)
g.launch()

# create the partitions for new image
print('Creating the initial partitions')
g.part_init('/dev/sdb', 'mbr')
g.part_add('/dev/sdb', 'primary', 2048, 616448)
g.part_add('/dev/sdb', 'primary', 616449, -1)
g.pvcreate('/dev/sdb2')
g.vgcreate('vg', ['/dev/sdb2', ])
volume_names = []
for volume,size in logical_volumes.iteritems():
  g.lvcreate(volume, 'vg', size)
  volume_names.append(volume)

volume_names.append('boot')
g.part_set_bootable('/dev/sdb', 1, True)

g.mkmountpoint('/old')
```

```
g.mount('/dev/sda', '/old')

for volume_name in volume_names:
    # add filesystems to volumes
    print('Adding filesystems to %s' % volume_name)
    if volume_name == 'boot':
        g.mkfs('ext4', '/dev/sdb1')
    elif volume_name == 'swap':
        g.mkswap_opts('/dev/vg/%s' % volume_name)
    else:
        g.mkfs('ext4', '/dev/vg/%s' % volume_name)

    # Mount the volume and copy files
    if volume_name not in ['tmp', 'swap']:
        print('Mounting %s' % volume_name)
        g.mkmountpoint('/%s' % volume_name)
        if volume_name == 'boot':
            g.mount('/dev/sdb1', '/%s' % volume_name)
        else:
            g.mount('/dev/vg/%s' % volume_name, '/%s' % volume_name)

        print('Copying files to %s' % volume_name)
        if volume_name == 'root':
            results = [filename for filename in g.ls('/old/') if
filename not in volume_names]
            results.append ('root')
            for result in results:
                print('- Copying %s to root' % result)
                g.cp_a('/old/%s' % result, '/root/')
        else:
            results = g.ls('/old/%s/' % volume_name)
            if results == []:
                print("- No files found")
            else:
                for result in results:
                    print('- Copying %s to %s' % (result,
volume_name))
                    g.cp_a('/old/%s/%s' % (volume_name, result),
'/%s/' % volume_name)

        g.umount('/%s' % volume_name)

g.umount('/old')
g.mount('/dev/vg/root', '/')

for volume_name in volume_names:
    if volume_name not in ['root','swap']:
        g.mkdir('/%s' % volume_name)
        if volume_name == 'boot':
            g.mount('/dev/sdb1', '/%s' % volume_name)
        else:
            g.mount('/dev/vg/%s' % volume_name, '/%s' % volume_name)

# create /etc/fstab file
print('Generating fstab content:')
```

```
fstab_content = []
fstab_content.append('UUID=%s /boot ext4 defaults 0 2' %
g.vfs_uuid('/dev/sdb1'))
fstab_content.append('UUID=%s / ext4 defaults 0 1' %
g.vfs_uuid('/dev/vg/root'))
fstab_content.append('UUID=%s none swap sw 0 0' %
g.vfs_uuid('/dev/vg/swap'))
for volume_name in volume_names:
    if volume_name not in ['boot', 'root', 'swap']:
        fstab_content.append('UUID=%s /%s ext4 defaults 0 2' %
(g.vfs_uuid('/dev/vg/%s' % volume_name), volume_name))

g.write('/etc/fstab', '\n'.join(fstab_content))
print '\n'.join(fstab_content)

# Generate bootloader
print('Installing bootloader')
g.sh('grub2-install --target=i386-pc /dev/sdb')
g.sh('grub2-mkconfig -o /boot/grub2/grub.cfg')
dracut_content = """
add_dracutmodules+="lvm crypt"
"""
g.write('/etc/dracut.conf', dracut_content)

# update initramfs to include lvm and crypt
kernels = g.ls('/lib/modules')
for kernel in kernels:
    print("Updating dracut to include modules in kernel %s" % kernel)
    g.sh('dracut -f /boot/initramfs-%s.img %s --force' % (kernel,
kernel))

for volume_name in volume_names:
    if volume_name not in ['swap', 'root']:
        g.umount("/%s" % volume_name)

g.umount("/")

# close images
print('Finishing image')
g.shutdown()
g.close()
```

将此脚本作为可执行文件保存到 undercloud，然后以 **stack** 用户的身份运行该文件。

```
$ ./whole-disk-image.py
```

此操作会从平面分区镜像自动创建完整的磁盘镜像。完整的磁盘镜像创建完成之后，替换旧的 **overcloud-full.qcow2** 镜像：

```
$ mv ~/images/overcloud-full.qcow2 ~/images/overcloud-full-old.qcow
$ cp ~/images/overcloud-full-partitioned.qcow2 ~/images/overcloud-
full.qcow
```

现在，您可以同时上传完整的磁盘镜像和其他镜像。

C.3. 在完整的磁盘镜像上对卷进行加密

您也可以使用 **guestfish** 在完整的磁盘镜像上对卷进行加密。此操作需要使用 **luks-format** 子命令来擦除现有的卷并创建一个加密卷。

以下 Python 脚本可用于打开之前创建的 **overcloud-full-partitioned.qcow2** 镜像，移除当前的 **home** 卷（为空），并使用加密的 **home** 卷代替：

```
#!/usr/bin/env python
import binascii
import guestfs

g = guestfs.GuestFS(python_return_dict=True)
g.add_drive_opts("/home/stack/images/overcloud-full-partitioned.qcow2",
format="qcow2", readonly=0)
g.launch()

random_content = binascii.b2a_hex(os.urandom(1024))
g.luks_format('/dev/vg/home', random_content, 0)
g.luks_open('/dev/vg/home', random_content, 'cryptedhome')
g.vgscan()
g.vg_activate_all(True)
g.mkfs('ext4', '/dev/mapper/cryptedhome')
g.mount('/dev/vg/root','/')

volumes = lvs()
volumes.remove('/dev/vg/home')
volumes.remove('/dev/vg/root')
volumes.remove('/dev/vg/swap')
fstab_content = []
fstab_content.append('UUID=%s /boot ext4 defaults 0 2' %
g.vfs_uuid('/dev/sda1'))
fstab_content.append('UUID=%s / ext4 defaults 0 1' %
g.vfs_uuid('/dev/vg/root'))
fstab_content.append('UUID=%s none swap sw 0 0' %
g.vfs_uuid('/dev/vg/swap'))
fstab_content.append('/dev/mapper/cryptedhome /home ext4 defaults 0 1')
for volume in volumes:
  volume_name = volume.replace('/dev/vg/', '')
  fstab_content.append('UUID=%s /%s ext4 defaults 0 2' %
(g.vfs_uuid(volume), volume_name))

g.write('/etc/fstab', '\n'.join(fstab_content))
print '\n'.join(fstab_content)

g.write('/root/home_keyfile', random_content)
g.chmod(0400, '/root/home_keyfile')

mapper = """
home UUID={home_id} /root/home_keyfile
""".format(home_id=g.vfs_uuid('/dev/mapper/cryptedhome'))
g.write('/etc/crypttab', mapper)

g.luks_close('/dev/mapper/cryptedhome')
```

```
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts',
'/', force=True)
g.shutdown()
g.close()
```

此脚本也可以：

- 创建密钥 (**random_content**)
- 使用新加密卷重新生成 **/etc/fstab** 文件
- 在 **/root/home_keyfile** 中保存加密密钥
- 生成一个 **crypttab** 文件，以便使用 **/root/home_keyfile**) 自动解密卷

以此脚本为例，创建加密卷，这是完整的磁盘镜像创建流程的组成部分。

C.4. 上传完整的磁盘镜像

要上传完整的磁盘镜像，请使用镜像上传命令的 **--whole-disk-image** 选项。例如：

```
$ openstack overcloud image upload --whole-disk --image-path
/home/stack/images
```

此命令从 **/home/stack/images** 上传镜像，但将 **overcloud-full.qcow2** 文件视为完整的磁盘镜像。这表示，在运行磁盘上传命令之前，您必须先将所需的完整磁盘镜像重命名为 **overcloud-full.qcow2**。

附录 D. 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。例如：

- 策略可以找出并隔离 overcloud 中性能较低或不稳定的节点。
- 策略可以自动使用标签把节点标记为不同的配置集。

D.1. 策略文件语法

策略使用 JSON 格式，它包括了一组规则。每个规则都包括一个 *description*、一个 *condition* 和一个 *action*。

描述

规则描述

Example:

```
"description": "A new rule for my node tagging policy"
```

Conditions

一个 condition 就是使用以下“关键字-值”来定义测试的条件：

field

定义要测试的项。如需了解项的类型，请参阅 第 D.4 节 “自动配置集标记属性”

op

指定测试所使用的操作。包括：

- **`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`** - 完全和提供的正则表达式相匹配
- **`contains`** - 包括和正则表达式匹配的值；
- **`is-empty`** - 检查项是否为空。

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。这包括：

- **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，返回“false”。

Example:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

当条件结果返回“true”时要进行的操作。它使用 **action** 关键字，以及由 **action** 值决定的额外关键字：

- **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** 项，它们分别是新能力的名称和值。当前存在的相同能力的值会被覆盖。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。

Example:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

D.2. 策论文件示例

以下是一个带有要应用的内省规则的 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。通过使用这个规则可以排除那些不应该成为您的云环境组成部分的节点。
- 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。我们分配了两个配置集（**compute_profile** 和 **control_profile**）以便 **openstack overcloud profiles match** 命令可以做最终的决定。另外，在这种情况下，还需要删除存在的配置集（如果不删除，存在的配置集会被优先使用）。

其它节点没有改变

注意

使用内省规则分配**配置集**总会覆盖存在的值。但是，**[PROFILE]_profile** 是一个例外，已存在配置集的节点会忽略它。

D.3. 导入策略文件

使用以下命令把策略文件导入到 director：

```
$ openstack baremetal introspection rule import rules.json
```

然后运行内省进程。

```
$ openstack baremetal introspection bulk start
```

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
```

如果您的内省规则有错误，可以把它们删除：

```
$ openstack baremetal introspection rule purge
```

D.4. 自动配置集标记属性

Automatic Profile Tagging（自动配置集标记）会检查每个条件的 **`field`** 属性的以下节点属性：

属性	描述
memory_mb	节点的内存数量（以 MB 为单位）。
cpus	节点 CPU 的内核总数量。
cpu_arch	节点 CPU 的架构。
local_gb	节点的 root 磁盘的存储总量。如需了解更多为一个节点设置 root 磁盘的信息，请参阅 第 5.4 节 “为节点定义 Root Disk”。

附录 E. 通过 FIREFOX 访问 UI 时的服务器例外

通过 Firefox 访问 UI 时，需要设置特定的服务器身份例外来允许 OpenStack Platform 服务。其中包括以下服务 URL：

URL	服务
`https://<undercloud>:13000`	OpenStack Identity (keystone)
`https://<undercloud>:13004`	OpenStack Orchestration (heat)
`https://<undercloud>:13385`	OpenStack Bare Metal（ironic）
`https://<undercloud>:13989`	OpenStack Workflow 服务（mistral）
`https://<undercloud>:13808`	OpenStack Object Storage (swift)
`https://<undercloud>:9000`	OpenStack Messaging 服务（zaqar）

将 **<undercloud>** 替换为您的 undercloud 的 Public API IP 地址。

在 Firefox 中添加 URL 的服务器身份例外：

1. 前往 **Preferences**。
2. 前往 **Certificates** > **View Certificates**。
3. 单击 **Servers** 选项卡，再单击 **Add Exception**。
4. 此时会显示一个窗口，要求您提供服务器的 URL。输入上表中的服务 URL 和端口，然后单击 **Get Certificate**。
5. 单击 **Confirm Security Exception**。

继续为所有服务 URL 添加例外。

附录 F. 增强安全性

以下部分提供了一些增强 undercloud 安全性的建议。

F.1. 更改 HAPROXY 使用的 SSL/TLS 密码和规则

如果您在 undercloud 中启用了 SSL/TLS（请参阅 第 4.6 节 “配置 director”），您可能需要增强 HAProxy 配置采用的 SSL/TLS 密码和规则。此操作可以帮助避免 SSL/TLS 漏洞，例如 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：

- 密码：**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

密码集合是一个连续行。

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
...
```