Red Hat OpenStack Platform 9

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 9 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

法律通告

摘要

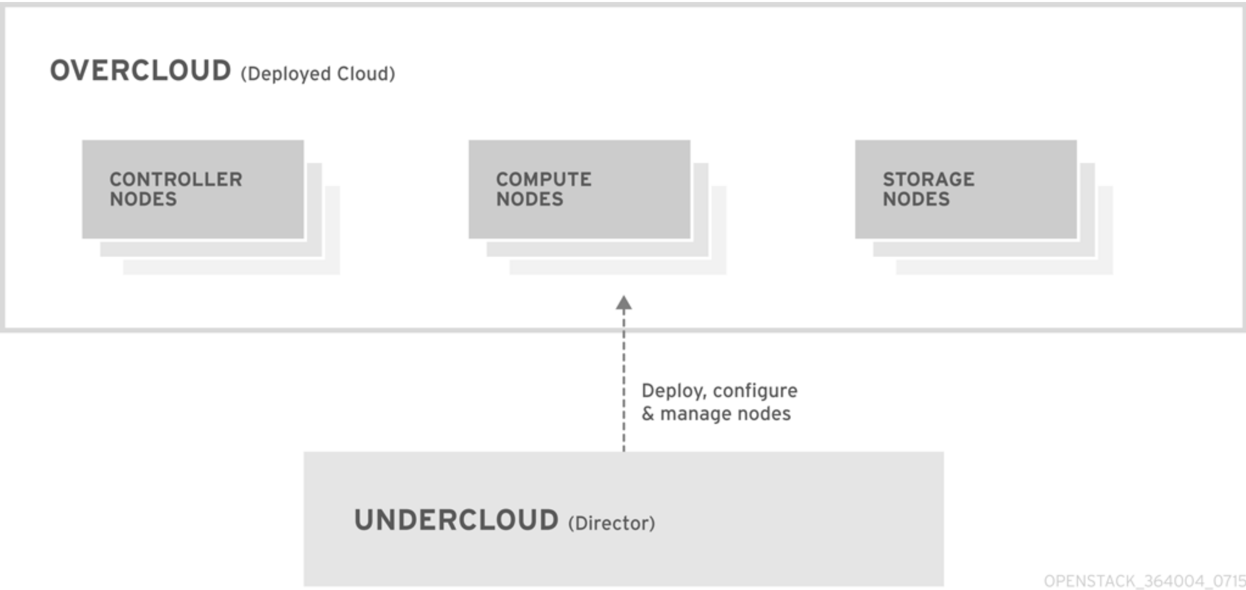
本指南包括了如何使用 Red Hat OpenStack Platform director 在企业级环境中安装 Red Hat OpenStack Platform 9 的信息。这包括安装 director、规划您的环境以及使用 director 创建一个 OpenStack 环境。

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 的组件具有以下功能：

- 环境规划 - Undercloud 提供了为用户分配 Red Hat OpenStack Platform 角色（Compute、Controller 和不同的存储节点）的功能。

- 逻辑系统控制 - Undercloud 使用每个节点的智能平台管理界面（Platform Management Interface，简称 IPMI）来进行电源管理控制，并使用一个基于 PXE 的服务来发现硬件的属性来在每个节点上安装 OpenStack。通过这个功能，可以把裸机系统部署为 OpenStack 节点。

- 编配（Orchestration） - Undercloud 提供了一组 YAML 模板来创建一个 OpenStack 环境。

Red Hat OpenStack Platform director 通过基于终端命令行的接口来使用这些 Undercloud 的功能。

Undercloud 包括以下组件：

- 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 Identity（keystone） - 提供 director 组件的验证和授权功能。
- MariaDB - director 的后台数据库。
- RabbitMQ - director 组件的消息队列。

1.2. OVERCLOUD

Overcloud 是一个通过 Undercloud 创建的 Red Hat OpenStack Platform 环境，它包括一个或多个以下类型的节点:

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)
- 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

- 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。

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 需要控制电源管理，所以全部 Overcloud 裸机系统都需要一个智能平台管理界面（IPMI）。

2.2. UNDERCLOUD 的配置要求

运行 director 的 Undercloud 系统会对 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。
- 在根磁盘上需要最少 40 GB 的可用磁盘空间。在进行 Overcloud 部署或更新前，需要保证最少有 10 GB 的可用磁盘空间。这些磁盘空间被用来保存镜像转换和缓存。
- 最少两个 1 Gbps 网卡。推荐使用 10 Gbps 网卡来作为 Provisioning 网络的接口，特别是您的 Overcloud 环境中存在大量节点。
- 安装 Red Hat Enterprise Linux 7.2 作为主机操作系统。

重要

如果使用 LVM（Logical Volume Management），请确保 Undercloud 的文件系统只包括 root 和 swap 分区。如需了解更多相关信息，请参阅红帽客户门户网站中的 "Director node fails to boot after undercloud installation"。

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以在 Overcloud 中使用。通常情况下，这个网络需要在一个主干（trunk）接口中使用一个原生的 VLAN，从而使 director 可用处理 PXE 引导和 DHCP 请求。虽然一些服务器硬件的 BIOS 支持从一个 VLAN 进行 PXE 引导，但 BIOS 必须同时支持在引导后把 VLAN 翻译为一个原生的 VLAN，否则将无法访问 Undercloud。当前，只有一小部分的服务器硬件支持这个功能。另外，这个网络还被用来通过 IPMI 对所有 Overcloud 节点进行电源管理。
- 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）。

以上只代表了所需网络的最少数量。director 还可以把其它 Red Hat OpenStack Platform 的网络流量分离到其它网络中。Red Hat OpenStack Platform 支持使用物理的网络接口或 tagged VLAN 进行网络分离。如需了解更多相关信息，请参阅 第 6.2 节 “分离网络”。

请注意：

- 典型的最小 Overcloud 网络配置可以是：
  - 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 - 不同的 Overcloud 网络类型使用不同的 NIC。
- 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 创建节点时，在所有 Overcloud 机器间使用一个名称来代表 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

如需了解更多与配置 IP 地址相关的信息（如用于存储网络、供应商网络和租户网络），请参阅网络指南。

- 把所有 Overcloud 系统设置为使用 Provisioning NIC 进行 PXE 引导，并在 External NIC 以及系统的所有其它 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 驱动之前引导）。
- 所有 Overcloud 裸机系统都需要一个被支持的电源管理接口（如 IPMI）。director 需要使用它来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。在设置 Overcloud 节点时需要使用这些信息。
- 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

以下小节包括了 Overcloud 中的独立系统和节点的配置要求信息。

注意

当前还不支持从 SAN（FC-AL、FCoE、iSCSI）引导一个 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 节点用来处理 Red Hat OpenStack Platform 环境中的核心服务，如 Horizon 仪表板、后台数据库服务器、Keystone 验证和高可用性服务。

处理器

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

内存

推荐的内存数量取决于 CPU 内核的数量。CPU 内核的数量越多，所需的内存也越大。如需了解更多相关信息，请参阅红帽客户门户网站中的 "Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers"。

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 存储节点为 Red Hat OpenStack Platform 环境提供项存储。

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
```

2.5. 软件仓库的要求

Undercloud 和 Overcloud 都需要通过 Red Hat Content Delivery Network 或 Red Hat Satellite 5 或 6 来访问红帽软件仓库。如果使用 Red Hat Satellite Server，把所需的软件仓库与您的 OpenStack Platform 环境进行同步。请参阅以下的 CDN 频道名列表：

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-6.1-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 9 director for RHEL 7 (RPMs)	**`rhel-7-server-openstack-9-director-rpms`**	Red Hat OpenStack Platform director 的软件仓库。它还提供了在 director 部署的 Overclouds 中使用的一些工具程序。
Red Hat Enterprise Linux OpenStack Platform 9 for RHEL 7 (RPMs)	**`rhel-7-server-openstack-9-rpms`**	核心 Red Hat OpenStack Platform 软件仓库。
Red Hat Ceph Storage OSD 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	**`rhel-7-server-rhceph-1.3-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 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	**`rhel-7-server-rhceph-1.3-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。

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。

下表提供了不同 Overcloud 配置的示例，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 Object 和 Block 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

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讯。Red Hat OpenStack Platform 使用 neutron 网络服务，这个服务会独立运行，并管理基于软件的网络、静态和浮动 IP 地址以及 DHCP。director 在 Overcloud 环境的每个节点上实施这个服务。

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	节点电源管理的网络。这个网络在安装 Undercloud 前被预先定义。	所有节点
Provisioning	director 使用这个网络类型来通过 PXE 引导实施新的节点，并调配在 Overcloud 裸机服务器上进行的 OpenStack 安装。这个网络在安装 Undercloud 前被预先定义。	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，它使用 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。如果提供在一个与 External 分离的 VLAN 中的浮动 IP，您可以把 Floating IP VLAN trunk 到 Controller 节点，并在 Overcloud 创建后通过 Neutron 添加 VLAN。这意味着，创建多个 Floating IP 网络附加到多个网桥。VLAN 会被 trunk，但不会被配置为接口。neutron 会为每个 Floating 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

在一个典型的 Red Hat OpenStack Platform 安装中，网络类型的数量通常会超过物理网络连接的数量。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，但以下网络需要一个第 3 层的网关来提供到互联网和其它网络的路由功能。

注意

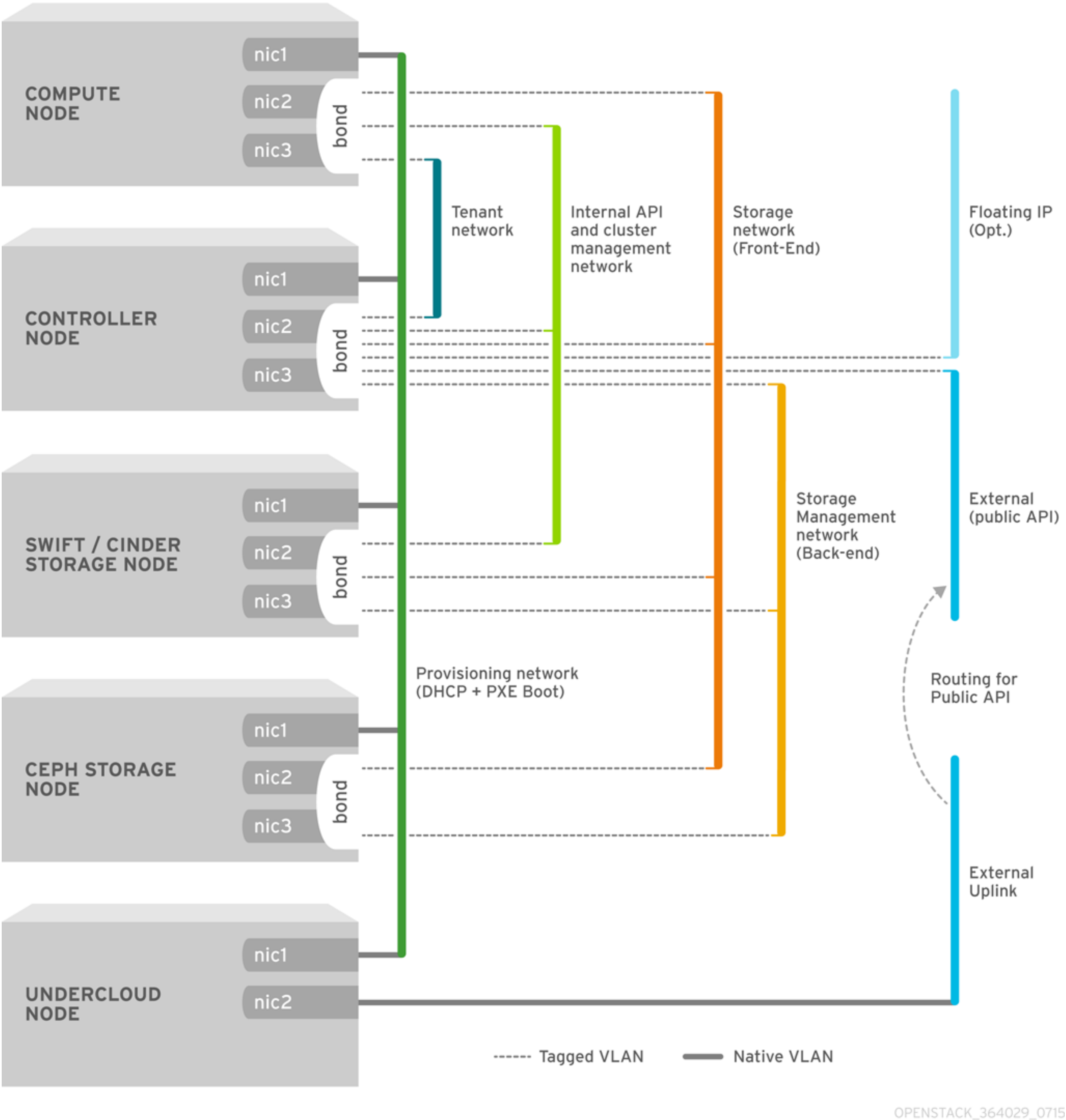
我们推荐，即使在部署时使用了禁用隧道功能（tunneling）的 neutron VLAN，您最好仍然部署一个 Tenant VLAN（用于 GRE 和 VXLAN 隧道）。这只需要在部署时进行一些微小的定制，从而为以后留下了使用网络隧道功能实现工具网络或虚拟化网络的选择。您仍然需要使用 VLAN 创建 Tenant 网络，但同时也为特殊目的的网络创建了 VXLAN 隧道功能，而不需要消耗租户 VLAN。VXLAN 功能可以被添加到带有 Tenant VLAN 的部署中，而 Tenant VLAN 却无法在不对系统运行造成影响的情况下添加到一个存在的 Overcloud 中。

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。每个 Overcloud 节点都使用两个接口（**`nic2`** 和 **`nic3`**）的绑定来通过相关的 VLAN 提供网络功能。同时，所有 Overcloud 节点都使用 **`nic1`** 通过原生的 VLAN 来通过 Provisioning 网络和 Undercloud 进行通讯。



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 - External、Floating 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 Network 7 - External 和 Floating IP（在创建 Overcloud 后被映射）	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 创建一组可扩展的存储节点。Overcloud 使用这些节点用于：

- **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 Architecture Guide。

Swift 存储节点

director 会创建一个外部的对象存储节点。当您需要扩展或替换 Overcloud 环境中的 controller 节点，同时需要在一个高可用性集群外保留块存储时，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（subscription）和软件仓库（repository）。

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。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

要安装 Red Hat OpenStack Platform director，首先需要使用 Red Hat Subscription Manager 注册系统，并订阅所需频道。

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
```

使用上个命令中获得的池 ID 添加 Red Hat OpenStack Platform 9 权利：

```
$ 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-openstack-9-
rpms --enable=rhel-7-server-openstack-9-director-rpms --enable=rhel-7-
server-rh-common-rpms
```

这些仓库包括了安装 director 所需的软件包。

重要

只启用以上列出的软件仓库。启用任何其它软件仓库都可能会造成软件冲突。

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
```

这个基本的模板包括以下参数：

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），请保留使用默认值 **192.0.2.1/24**。

network_gateway

Overcloud 实例的网关。它是 Undercloud 主机，会把网络流量转发到 External 网络。除非您的 director 使用不同的 IP 地址，或直接使用一个外部网关，请保留使用默认的值（**192.0.2.1**）。

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_vip

director 的 Public API 的 IP 地址。使用 Provisioning 网络中的一个与其它任何 IP 地址或地址范围都不冲突的 IP 地址。例如，**192.0.2.2**。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_vip

director 的 Admin API 的 IP 地址。使用 Provisioning 网络中的一个与其它任何 IP 地址或地址范围都不冲突的 IP 地址。例如，**192.0.2.3**。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_service_certificate

用于 OpenStack SSL 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* 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）设置上下文的方法。

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。

network_cidr

director 用来管理 Overcloud 实例的网络，它是 Provisioning 网络。除非您的 Provisioning 网络使用了不同的子网，保留使用默认值（**192.0.2.0/24**）。

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，请保留使用默认的值（**192.0.2.0/24**）。

dhcp_start; dhcp_end

Overcloud 节点的 DHCP 分配范围的开始值和终止值。请确保这个范围可以为节点提供足够的 IP 地址。

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，**192.0.2.100,192.0.2.120**。请确保这个范围有足够的 IP 地址，并和 **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”。

undercloud_debug

把 Undercloud 服务的日志级别设置为 **DEBUG**。把它设为 **true** 来启用它。

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**。

ipxe_deploy

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" 。

store_events

定义是否在 Undercloud 的 Ceilometer 中保存事件信息。

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。

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。
- 一个 Overcloud 内核、ramdisk 和完整镜像 - 写到节点硬盘中的一个基本的 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-
9.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-
9.0.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。部署的过程以及 Overcloud 都需要这些镜像。另外，这个脚本还会在 director 的 PXE 服务器上安装内省（interospection）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
```

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

Overcloud 节点需要一个名称解析服务器来提供 DNS 解析主机名。对于一个没有网络分离的标准 Overcloud，这个服务器在 Undercloud 的 **neutron** 子网中定义。使用以下命令设置名称解析服务器：

```
$ neutron subnet-list
$ neutron subnet-update [subnet-uuid] --dns-nameserver [nameserver-ip]
```

查看子网来验证名称解析服务器：

```
$ neutron subnet-show [subnet-uuid]
+-------------------+-----------------------------------------------
-+
| Field             | Value
|
+-------------------+-----------------------------------------------
-+
| ...               |
```

```
|
| dns_nameservers   | 8.8.8.8
|
| ...               |
|
+-------------------+---------------------------------------------
-+
```

重要

如果需要把服务网络流量分离到不同的网络中，Overcloud 节点会使用网络环境模板中的 **DnsServer** 参数。这包括在 第 6.2 节 “分离网络” 中的高级配置情景中。

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 "Back Up and Restore the Director Undercloud"。

4.10. 完成 UNDERCLOUD 的配置

到此已完成了 Undercloud 配置。在下一章中会介绍基本的 Overcloud 配置，包括注册节点、内省它们、把它们标记为不同的节点角色。

第 5 章 配置基本 OVERCLOUD

本章介绍了对一个企业级的 OpenStack Platform 环境进行基本配置的步骤。具有基本配置的 Overcloud 不包括定制的功能，但是您可以为基本配置的 Overcloud 添加高级配置选项，或根据您的具体情况对它进行定制。相关信息包括在 第 6 章 *为 Overcloud 配置高级的定制环境* 中。

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

- 第 4 章 *安装 Undercloud* 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 第 2.4 节 “Overcloud 的配置要求”。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 192.0.2.0/24 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	192.0.2.1	aa:aa:aa:aa:aa:aa	不需要
Controller	DHCP	bb:bb:bb:bb:bb:bb	192.0.2.205
Compute	DHCP	cc:cc:cc:cc:cc:cc	192.0.2.206

- 所有其它网络类型使用 Provisioning 网络作为 OpenStack 服务。但是，您也可以为其它网络通讯类型创建额外的网络。相关信息，请参阅 第 6.2 节 “分离网络”。

5.1. 为 OVERCLOUD 注册节点

director 需要一个节点定义模板。这个文件（**instackenv.json**）是一个 JSON 格式的文件，它包括了环境中的节点的硬件信息和电源管理信息。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":"192.0.2.205"
        },
        {
            "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":"192.0.2.206"
        }
    ]
}
```

这个模板使用以下属性：

mac

节点上的网络接口的 MAC 地址列表。对于每个系统的 Provisioning NIC，只使用 MAC 地址。

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。

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 baremetal import --json ~/instackenv.json
```

这会导入模板，并把模板中的每个节点注册到 director。

为所有节点分配内核和 ramdisk 镜像：

```
$ openstack baremetal configure boot
```

现在，节点已在 director 中注册并被配置。使用以下命令可以在 CLI 中查看这些节点的列表：

```
$ ironic 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 C, *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 baremetal introspection bulk start
```

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。

或者，在每个节点上独立进行一个内省操作。把节点设置为管理模式，执行内省操作，然后把节点移出维护模式：

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [NODE UUID] provide
```

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（tag）来把它们标记为特定的配置集。这些配置集标签会把节点和 flavor 相匹配，从而使 flavor 被分配到一个部署角色。在 Undercloud 的安装过程中，会创建默认的配置集 flavor：**compute**、**control**、**swift-storage**、**ceph-storage** 和 **block-storage**，在多数环境中，都可以在不经过修改的情况下使用它们。

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 C, *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：

```
$ ironic node-update 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
```

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。

重要

director 当前还不支持 UEFI 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
```

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）。

在这个示例中，使用磁盘的序列号来指定 root 设备来部署 Overcloud 镜像。

首先，收集 director 通过内省获得的每个节点的硬件信息。这些信息保存在 the OpenStack Object Storage 服务器（swift）中。把这些信息下载到一个新目录中：

```
$ mkdir swift-data
$ cd swift-data
$ export IRONIC_DISCOVERD_PASSWORD=`sudo grep admin_password
/etc/ironic-inspector/inspector.conf | awk '! /^#/ {print $NF}' | awk -
F'=' '{print $2}'`
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do swift
-U service:ironic -K $IRONIC_DISCOVERD_PASSWORD download ironic-
inspector inspector_data-$node; done
```

这会从内省中下载每个 **inspector_data** 项中的信息。所有项都会使用节点的 UUID 作为项的名称的一部分：

```
$ ls -1
inspector_data-15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
inspector_data-46b90a4d-769b-4b26-bb93-50eaefcdb3f4
inspector_data-662376ed-faa8-409c-b8ef-212f9754c9c7
inspector_data-6fc70fe4-92ea-457b-9713-eed499eda206
inspector_data-9238a73a-ec8b-4976-9409-3fcff9a8dca3
inspector_data-9cbfe693-8d55-47c2-a9d5-10e059a14e07
inspector_data-ad31b32d-e607-4495-815c-2b55ee04cdb1
inspector_data-d376f613-bc3e-4c4b-ad21-847c4ec850f8
```

检查每个节点的磁盘信息。以下信息显示了每个节点的 ID 和磁盘信息：

```
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do echo
"NODE: $node" ; cat inspector_data-$node | jq '.inventory.disks' ; echo
"-----" ; done
```

例如，一个节点的数据可能会显示 3 个磁盘：

```
NODE: 46b90a4d-769b-4b26-bb93-50eaefcdb3f4
[
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sda",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000001",
    "serial": "WD-000000000001"
```

```
    },
    {
      "size": 1000215724032,
      "vendor": "ATA",
      "name": "/dev/sdb",
      "model": "WDC WD1002F9YZ",
      "wwn": "0x0000000000000002",
      "serial": "WD-000000000002"
    },
    {
      "size": 1000215724032,
      "vendor": "ATA",
      "name": "/dev/sdc",
      "model": "WDC WD1002F9YZ",
      "wwn": "0x0000000000000003",
      "serial": "WD-000000000003"
    },
]
```

在这个例子中，磁盘 2 被设置为 root 设备（序列号为 **WD-000000000002**）。这需要在节点定义中修改 **root_device** 参数：

```
$ ironic node-update 97e3f7b3-5629-473e-a187-2193ebe0b5c7 add
properties/root_device='{"serial": "WD-000000000002"}'
```

这将帮助 director 区分特定的磁盘来作为 root 磁盘。当开始创建 Overcloud 时，director 会部署这个节点，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. 完成基本配置

这包括了对 Overcloud 进行基本配置的步骤。现在可以：

- 使用高级配置步骤对环境进行定制。如需了解更多相关信息，请参阅 第 6 章 *为 Overcloud 配置高级的定制环境*。
- 或部署一个基本的 Overcloud。如需了解更多相关信息，请参阅 第 7 章 *创建 Overcloud*。

重要

基本的 Overcloud 会使用本地 LVM 存储作为块存储，这并不是一个被支持的配置。我们推荐使用一个外部的存储环境作为块存储。例如，第 6.7 节 “配置 NFS 存储” 介绍了配置一个 NFS 共享作为块存储的方法。

第 6 章 为 OVERCLOUD 配置高级的定制环境

本章是 第 5 章 *配置基本 Overcloud* 一章的延续。到目前为止，director 已注册了节点并为 Overcloud 的创建配置了所需的服务。现在，您可以使用本章介绍的方法对 Overcloud 进行定制。

注意

本章中所举的例子是配置 Overcloud 的可选步骤。这些步骤只有在 Overcloud 需要额外功能时才需要进行。请只执行适用于您的环境的步骤。

6.1. 了解 HEAT 模板

本章中的自定义配置使用 Heat 模板和环境变量来定义 Overcloud 的特定功能，如网络分离（network isolation）和网络接口配置。本节对 Heat 模板进行了一个基本的介绍，从而使您可以对 Red Hat OpenStack Platform director 中使用的模板结构和格式有所了解。

6.1.1. Heat 模板

director 使用 Heat Orchestration Templates（HOT）作为模板格式来组成 Overcloud 的部署计划。HOT 格式的模板通常使用 YAML 格式。一个模板的目的是创建一个*栈（stack）*，栈中包括了一组资源集合，Heat 会创建并配置每个资源。资源就是 OpenStack 中的对象（object），它包括计算资源、网络配置、安全组、扩展规则和自定义资源。

一个 Heat 模板有以下 3 个主要项：

- **Parameters（参数）** - 一组传递给 heat 的参数，可以被用来自定义一个栈，并设置在没有传递值时相关参数所使用的默认值。这些参数在模板的 **`parameters`** 项中定义。
- **Resources（资源）** - 一组作为栈的一部分需要创建和配置的对象。OpenStack 包括一组分布在所有组件中的资源，它们在模板的 **`resources`** 项中定义。
- **Output（输出）** - 一组在栈创建后从 heat 传递的值。您可以通过 heat API 或客户端工具程序来访问这些值。它们在模板的 **`output`** 项中定义。

以下是一个基本 heat 模板的示例：

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance
```

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

output:
  instance_name:
    description: Get the instance's name
    value: { get_attr: [ my_instance, name ] }
```

这个模板使用资源类型 **type: OS::Nova::Server** 创建一个名为 **my_instance** 的实例，它具有特定的 flavor、镜像和关键字。这个栈会返回 **instance_name** 的值（**My Cirros Instance**）。

当 Heat 处理一个模板时会为模板创建一个栈，并为资源模板创建一组子栈。这就形成了一个分级结构的栈，它的最高级就是使用您的模板定义的主栈。使用以下命令可以查看栈的分级结构：

```
$ heat stack-list --show-nested
```

6.1.2. 环境文件

环境文件就是一个特殊类型的模板，它为 Heat 模板提供了自定义的功能。这个文件包括三个主要部分：

- **Resource Registry（资源注册表）** - 它设置了自定义资源名，并连接到其它 heat 模板。这提供了一个创建没有存在于核心资源集合中的自定义资源的方法。它在环境文件的 **resource_registry** 项中设置。
- **Parameters（参数）** - 应用到高级别模板参数中的常规设置。例如，您有一个模板用来部署嵌套的栈，如资源注册表映射，这些参数只需要应用于高级别的模板，而不需要在嵌套的栈中进行应用。参数在环境文件中的 **parameters** 部分进行定义。
- **Parameter Defaults（参数默认值）** - 这些参数会修改所有模板中的参数默认值。例如，您有一个模板用来部署嵌套的栈，如资源注册表映射，参数的默认值会应用到所有模板中。包括高级别的模板以及所有嵌套的资源。参数的默认值在环境文件的 **parameter_defaults** 项中定义。

重要

在为 Overcloud 创建自定义环境文件时，推荐使用 **parameter_defaults** 而不是 **parameters**，这样可以使参数应用到 Overcloud 中的所有模板中。

以下是一个基本环境文件的示例：

```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml

parameter_defaults:
  NetworkName: my_network

parameters:
```

```
MyIP: 192.168.0.1
```

例如，当通过一个特定 heat 模板（**my_template.yaml**）创建一个栈时，可以包括这个环境文件（**my_env.yaml**）。**my_env.yaml** 文件会创建一个名为 **OS::Nova::Server::MyServer** 的新资源类型。**myserver.yaml** 文件是一个 Heat 模板文件，它为这个资源类型提供了一个实施来覆盖内建的设置。您可以在 **my_template.yaml** 文件中包括 **OS::Nova::Server::MyServer** 资源。

MyIP 只会对和这个环境文件一起部署的主 Heat 模板应用一个参数。在这个示例中，它只应用于 **my_template.yaml** 中的参数。

NetworkName 会对主 Heat 模板（在这个示例中是 **my_template.yaml**），以及与包括在主模板中的资源相关联的模板进行应用（在这个示例中，资源是 **OS::Nova::Server::MyServer**，模板是 **myserver.yaml**）。

6.1.3. 核心 Overcloud Heat 模板

director 为 Overcloud 包括了一个核心 Heat 模板集合，它被保存在 **/usr/share/openstack-tripleo-heat-templates** 中。

这个集合中包括了许多 heat 模板和环境文件。其中需要特别说明的主文件和目录是：

- **overcloud.yaml** - 这是创建 Overcloud 环境所使用的主要模板。
- **overcloud-resource-registry-puppet.yaml** - 这是创建 Overcloud 环境所使用的主要环境文件。它为 Puppet 模块提供了一组存储在 Overcloud 镜像中的配置。当 director 为每个节点写入 Overcloud 镜像后，Heat 将使用在环境文件中注册的资源来为每个节点进行配置。
- **environments** - 一个包括了可以应用到 Overcloud 部署中的环境文件示例的目录。

6.2. 分离网络

director 提供了配置分离 Overcloud 网络的方法。这意味着，Overcloud 环境把不同类型的网络数据分离到不同的网络，从而可以把特定的网络数据分配给特定的网络接口或接口绑定。在配置完分离的网络后，director 会配置 OpenStack 服务来使用分离的网络。如果没有配置分离的网络，所有服务都会在 Provisioning 网络中运行。

在这个示例中，所有的服务都使用独立的网络：

- Network 1 - Provisioning
- Network 2 - Internal API
- Network 3 - Tenant Network
- Network 4 - Storage
- Network 5 - Storage Management
- Network 6 - Management
- Network 7 - External 和 Floating IP（在创建 Overcloud 后被映射）

在这个示例中，每个 Overcloud 节点使用两个网络接口组成网络绑定来处理标记 VLAN（tagged VLAN）中的网络。这个绑定使用以下网络设置：

表 6.1. 网络子网和 VLAN 分配

网络类型	子网	VLAN
Internal API	172.16.0.0/24	201
Tenant	172.17.0.0/24	202
Storage	172.18.0.0/24	203
Storage Management	172.19.0.0/24	204
Management	172.20.0.0/24	205
External / Floating IP	10.1.1.0/24	100

如需了解更多网络配置示例，请参阅 第 3.2 节 “规划网络”。

6.2.1. 创建自定义接口模板

Overcloud 网络配置需要一组网络接口模板。您可以对这些模板进行定制来基于角色对节点进行配置。这些模板是 YAML 格式的标准 Heat 模板（请参阅 第 6.1.1 节 “Heat 模板”）。director 包括了一组模板示例以供参考：

- **/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans** - 这个目录中包括了基于角色的、带有 VLAN 配置的单独 NIC 的模板。
- **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans** - 这个目录中包括了基于角色的、绑定 NIC 配置的模板。
- **/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics** - 这个目录包括了多 NIC 配置的模板，其中的每个角色都使用一个 NIC。
- **/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-linux-bridge-vlans** - 这个目录中包括了基于角色的、带有 VLAN 配置的单独 NIC 的模板，其中的 VLAN 使用 Linux 网桥而不是使用 Open vSwitch 网桥。

在这个示例中，使用默认绑定的 NIC 配置作为一个基础。复制 **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans**。

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/network/config/bond-with-vlans ~/templates/nic-configs
```

这会创建一组本地的 heat 模板，它们为每个角色定义了一个绑定的网络接口配置。每个模板都包括标准的 **parameters**、**resources** 和 **output** 项。在这个示例中，我们只编辑 **resources** 项，每个 **resources** 以以下内容开始：

```
resources:
OsNetConfigImpl:
  type: OS::Heat::StructuredConfig
  properties:
    group: os-apply-config
    config:
      os_net_config:
        network_config:
```

这会创建一个对 **os-apply-config** 命令和 **os-net-config** 子命令的请求来为一个节点配置网络属性。**network_config** 项中包括了自定义的接口配置，这些配置以类型的形式进行组织，它们包括：

interface

定义一个单独网络接口。这个配置指定了每个接口需要使用实际的接口名（"eth0"、"eth1"、"enp0s25"）还是使用接口编号（"nic1"、"nic2"、"nic3"）。

```
            - type: interface
              name: nic2
```

vlan

定义一个 VLAN。使用从 **parameters** 项中传递来的 VLAN ID 和子网。

```
            - type: vlan
              vlan_id: {get_param: ExternalNetworkVlanID}
              addresses:
                - ip_netmask: {get_param: ExternalIpSubnet}
```

ovs_bond

定义 Open vSwitch 中的绑定。一个绑定会把两个 **interfaces** 组合在一起来起到冗余和增加带宽的目的。

```
            - type: ovs_bond
              name: bond1
              members:
              - type: interface
                name: nic2
              - type: interface
                name: nic3
```

ovs_bridge

在 Open vSwitch 中定义网桥。网桥把多个 **interface**、**ovs_bond** 和 **vlan** 对象连接在一起。

```
            - type: ovs_bridge
              name: {get_input: bridge_name}
              members:
                - type: ovs_bond
                  name: bond1
                  members:
                    - type: interface
```

```
                  name: nic2
                  primary: true
                - type: interface
                  name: nic3
            - type: vlan
              device: bond1
              vlan_id: {get_param: ExternalNetworkVlanID}
              addresses:
                - ip_netmask: {get_param: ExternalIpSubnet}
```

linux_bond

定义一个把两个或多个 **interfaces** 连接到一起的 Linux 绑定。这可以起到冗余和增加带宽的效果。请确定在 **bonding_options** 参数中包括了基于内核的绑定选项。如需了解更多与 Linux 绑定选项相关的信息，请参阅 Red Hat Enterprise Linux 7 Networking Guide 中的 4.5.1. Bonding Module Directives。

```
              - type: linux_bond
                name: bond1
                members:
                - type: interface
                  name: nic2
                - type: interface
                  name: nic3
                bonding_options: "mode=802.3ad"
```

linux_bridge

定义一个 Linux 网桥，它把多个 **interface**、**linux_bond** 和 **vlan** 项连接到一起。

```
              - type: linux_bridge
                name: bridge1
                addresses:
                  - ip_netmask:
                      list_join:
                        - '/'
                        - - {get_param: ControlPlaneIp}
                          - {get_param: ControlPlaneSubnetCidr}
                members:
                  - type: interface
                    name: nic1
                    primary: true
              - type: vlan
                vlan_id: {get_param: ExternalNetworkVlanID}
                device: bridge1
                addresses:
                  - ip_netmask: {get_param: ExternalIpSubnet}
                routes:
                  - ip_netmask: 0.0.0.0/0
                    default: true
                    next_hop: {get_param:
ExternalInterfaceDefaultRoute}
```

如需了解更详细的信息，请参阅 附录 E, *网络接口参数*。

在这个示例中，我们使用默认的绑定节点配置。例如，**/home/stack/templates/nic-configs/controller.yaml** 模板使用以下 **network_config** 设置：

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            - type: interface
              name: nic1
              use_dhcp: false
              addresses:
                - ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
            - type: ovs_bridge
              name: {get_input: bridge_name}
              dns_servers: {get_param: DnsServers}
              members:
                - type: ovs_bond
                  name: bond1
                  ovs_options: {get_param: BondInterfaceOvsOptions}
                  members:
                    - type: interface
                      name: nic2
                      primary: true
                    - type: interface
                      name: nic3
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: ExternalNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: ExternalIpSubnet}
                  routes:
                    - default: true
                      next_hop: {get_param:
ExternalInterfaceDefaultRoute}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: InternalApiIpSubnet}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: StorageNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: StorageIpSubnet}
                - type: vlan
```

```
                device: bond1
                vlan_id: {get_param: StorageMgmtNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: StorageMgmtIpSubnet}
              - type: vlan
                device: bond1
                vlan_id: {get_param: TenantNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: TenantIpSubnet}
              - type: vlan
                device: bond1
                vlan_id: {get_param: ManagementNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: ManagementIpSubnet}
```

注意

Management 网络的配置内容在网络接口 Heat 模板中被注释掉。取消注释这些内容可以启用 Management 网络。

这个模板定义了一个网桥（通常是名为 **br-ex** 的外部网桥），并创建了一个由两个编号的接口（**nic2** 和 **nic3**）组成的一个名为 **bond1** 的绑定接口。这个网络还包括了一组加标签的 VLAN（tagged VLAN）设备，并使用 **bond1** 作为父设备。这个模板还包括了一个接口，它被用来连接回 director（**nic1**）。

附录 F, *网络接口模板示例* 中包括了更多网络接口模板示例。

请注意，许多参数使用了 **get_param** 功能。您可以在一个针对于您的网络所创建的一个环境文件中定义它们。

重要

未使用的接口可能会导致不需要的默认路由和网络循环。例如，您的模板可能会包括一个网络接口（**nic4**），它不使用任何为 OpenStack 服务分配的 IP，但使用 DHCP 或默认的路由。为了避免网络冲突，从 **ovs_bridge** 设备中删除所有使用的接口，并禁用 DHCP 和默认路由设置：

```
- type: interface
  name: nic4
  use_dhcp: false
  defroute: false
```

6.2.2. 创建一个网络环境文件

网络环境文件是一个 Heat 环境文件，它描述了 Overcloud 的网络环境，并指向在前一节中提到的网络接口配置模板。您可以在定义网络 IP 地址范围的同时还定义子网和 VLAN。然后根据本地环境对这些值进行定制。

为了方便用户，director 包括了一组环境文件示例。每个环境文件对应于 **/usr/share/openstack-tripleo-heat-templates/network/config/** 中的示例网络接口文件：

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml** - 在 **single-nic-vlans** 网络接口目录中的、带有 VLAN 配置的单一 NIC 的环境文件。同时，还包括了用来禁用 External 网络的环境文件（**net-single-nic-with-vlans-no-external.yaml**）和用来启用 IPv6 的环境文件（**net-single-nic-with-vlans-v6.yaml**）。

- **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml** - 在 **bond-with-vlans** 网络接口目录中的、绑定的 NIC 配置的环境文件。同时，还包括了用来禁用 External 网络的环境文件（**net-bond-with-vlans-no-external.yaml**）和用来启用 IPv6 的环境文件（**net-bond-with-vlans-v6.yaml**）。

- **/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml** - 在 **multiple-nics** 网络接口目录中的、多 NIC 配置的环境文件。同时，还包括用来启用 IPv6 的环境文件（**net-multiple-nics-v6.yaml**）。

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-linux-bridge-with-vlans.yaml** - 带有使用 Linux 网桥而不是 Open vSwitch 网桥的单一 NIC 配置的环境变量，这个 NIC 配置所在的目录是 **single-nic-linux-bridge-vlans**。

这里使用了一个经过修改的 **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml** 文件版本。把这个文件复制到 stack 用户的 **templates** 目录中。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml /home/stack/templates/network-environment.yaml
```

修改环境文件使它包括与您的环境相关的参数，特别是：

- **resource_registry** 的部分应该包括到每个角色的自定义网络接口模板的链接。相关信息，请参阅 第 6.1.2 节 “环境文件”。

- **parameter_defaults** 项应该包括一组参数，它们被用来定义每个网络类型的网络选项。如需获得这些参数的详细信息，请参阅 附录 G, *网络环境选项*。

例如：

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig:
/home/stack/templates/nic-configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/ceph-storage.yaml

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ManagementNetCidr: 172.20.0.0/24
  ExternalNetCidr: 10.1.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end':
```

```
'172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end':
'172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end':
'172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end':
'172.19.0.200'}]
  ManagementAllocationPools: [{'start': '172.20.0.10', 'end':
'172.20.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '10.1.1.10', 'end': '10.1.1.50'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.1.1.1
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.2.254
  # The IP address of the EC2 metadata server. Generally the IP of the
Undercloud
  EC2MetadataIp: 192.0.2.1
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ManagementNetworkVlanID: 205
  ExternalNetworkVlanID: 100
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: "''"
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=balance-slb"
```

它为每个网络定义了选项。所有网络都使用单独的 VLAN，而子网被用来为主机和虚拟 IP 分配 IP 地址。在上面的示例中，Internal API 网络的分配池从 172.16.0.10 开始，直到 172.16.0.200（使用 VLAN 201）。静态 IP 和虚拟 IP 的分配范围从 172.16.0.10 开始，直到 172.16.0.200（使用 VLAN 201）。

External 网络用来运行 Horizon dashboard 和 Public API。如果使用 External 网络进行云管理，以及对浮动 IP 的管理，需要确保有足够的空间容纳一个 IP 池来为虚拟机提供 IP 地址。在这个示例中，为 External 网络分配的 IP 地址范围是从 10.1.1.10 到 10.1.1.50，而从 10.1.1.51 开始的未使用的 IP 地址可以作为浮动 IP 地址使用。或者，把 Floating IP 网络放置到一个独立的 VLAN 中，并在创建 Overcloud 后进行配置来使用它。

`BondInterfaceOvsOptions` 提供了使用 **`nic2`** 和 **`nic3`** 组成绑定接口的方法。如需了解更多与绑定选项相关的信息，请参阅 附录 H, *Open vSwitch 绑定选项*。

重要

由于资源可用性的问题，在创建 Overcloud 后改变网络配置可能会出现配置问题。例如，一个用户在网络分离模板中修改了一个网络的子网范围，因为这个资源可能已在使用，重新配置操作会失败。

6.2.3. 把 OpenStack 服务分配到分离的网络

每个 OpenStack 服务都会被分配到资源注册表中的一个默认网络类型。这些服务然后会和网络类型

所分配的网络中的一个 IP 地址相绑定。虽然 OpenStack 服务在这些网络中被分开，实际的物理网络数量可能会和网络环境文件中所定义的不同。您可以通过在网络环境文件（**/home/stack/templates/network-environment.yaml**）中定义一个新的网络映射来把 OpenStack 服务重新分配给不同的网络类型。**ServiceNetMap** 参数决定了每个服务所使用的网络类型。

例如，您可以通过修改 **ServiceNetMap** 把 Storage Management 网络服务重新分配给 Storage Network：

```
parameter_defaults:
  ...
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: internal_api
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage          # Change from 'storage_mgmt'
    SwiftProxyNetwork: storage
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage        # Change from 'storage_mgmt'
    CephPublicNetwork: storage
    # Define which network will be used for hostname resolution
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
    ...
```

把这些参数改为 **storage** 会把这些服务放置到 Storage 网络而不是 Storage Management 网络。这意味着，您只需要为 Storage 网络定义一组 **parameter_defaults**，而不是 Storage Management 网络。

6.2.4. 选择要部署的网络

在一般情况下，环境文件中的 **resource_registry** 项中的设置不需要修改。如果只需要其中列出的一部分网络，可以对网络列表进行修改。

注意

在指定自定义网络和端口时，不要在部署命令中包括 **environments/network-isolation.yaml**，而是在网络环境文件中指定所有的网络和端口。

为了使用分离的网络，服务器需要有每个网络上的 IP。您可以在 Undercloud 中使用 neutron 来管理分离网络中的 IP 地址，这需要为每个网络启动 neutron 端口创建。您可以在环境文件中覆盖资源注册表中的设置。

首先是可以被部署的网络和端口的完整列表：

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/storage_mgmt.yaml
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml
  OS::TripleO::Network::Management: /usr/share/openstack-tripleo-heat-
templates/network/management.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::ManagementVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort:
```

```
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt.yaml
  OS::TripleO::CephStorage::Ports::ManagementPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/management.yaml

  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt.yaml
  OS::TripleO::SwiftStorage::Ports::ManagementPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/management.yaml

  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt.yaml
  OS::TripleO::BlockStorage::Ports::ManagementPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/management.yaml
```

这个文件的第一部分包括了 **OS::TripleO::Network::*** 资源的资源注册信息。在默认情况下，这些资源指向一个 **noop.yaml** 文件，它不会创建任何网络。通过把这些资源指向相关的 YAML 文件，就可以启用对这些网络的创建。

接下来的几个部分会为每个角色中的节点创建 IP 地址。控制器节点（controller node）有每个网络上的 IP，而计算节点（compute node）和存储节点（storage node）具有网络中相应子网的 IP。

要在没有预配置网络的情况下进行部署，为角色禁用网络定义，以及相关的端口定义。例如，所有到 **storage_mgmt.yaml** 的指代都需要替换为指代到 **noop.yaml**：

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/noop.yaml
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
```

```
  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

parameter_defaults:
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: ctlplane # Admin connection for Undercloud
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage # Changed from storage_mgmt
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage # Changed from storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
```

```
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

使用 **noop.yaml** 将不会创建任何网络或端口，因此，Storage Management 网络上的服务会被默认位于 Provisioning 网络中。通过 **ServiceNetMap**，可以把 Storage Management 服务移到另外一个网络中（如 Storage network）。

6.3. 控制节点的位置

默认情况下，director 为每个角色随机选择节点，通常基于它们的配置集标签（tag）。但是，director 也提供了指定特定节点的功能。它可以实现：

- 分配特定节点 ID，如 **controller-0**、**controller-1** 等
- 设置自定义的主机名
- 设置特定的 IP 地址
- 设置特定的虚拟 IP 地址

6.3.1. 分配特定的节点 ID

这个过程把节点 ID 分配给特定节点。节点 ID 的示例包括 **controller-0**、**controller-1**、**novacompute-0**、**novacompute-1** 等等。

第一步是，分配 ID 作为每个节点的能力，从而使 Nova 调度程序可以在部署时进行匹配。例如：

```
ironic node-update <id> replace
properties/capabilities='node:controller-0,boot_option:local'
```

这会把能力 **node:controller-0** 分配给节点。使用连续的索引值来为所有节点进行分配（以 0 开始）。确定对于一个特定角色（Controller、Compute 以及每个存储角色）的所有节点都以同样形式进行标记（tag），否则 Nova 调度程序将无法正确匹配能力。

下一步是，创建一个 Heat 环境文件（例如，**scheduler_hints_env.yaml**），它使用调度程序的 hint 来为每个节点匹配能力。例如：

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
```

要使用这些调度程序 hint，在进行 Overcloud 创建时在 **overcloud deploy command** 中包括 **scheduler_hints_env.yaml** 环境文件。

按照同样的方法，使用以下参数设置每个节点：

- **ControllerSchedulerHints** 用于 Controller 节点。
- **NovaComputeSchedulerHints** 用于 Compute 节点。
- **BlockStorageSchedulerHints** 用于 Block Storage 节点。
- **ObjectStorageSchedulerHints** 用于 Object Storage 节点。

- **CephStorageSchedulerHints** 用于 Ceph Storage 节点。

> **注意**
>
> 节点位置的设置会比配置集匹配有更高优先级。为了避免调度失败，在部署时使用 **baremetal** flavor，而不要使用针对于配置集匹配的 flavor（如 **compute**、**control** 等）。例如：
>
> ```
> $ openstack overcloud deploy ... --control-flavor baremetal --
> compute-flavor baremetal ...
> ```

6.3.2. 自定义主机名

通过使用 第 6.3.1 节 "分配特定的节点 ID" 中介绍的节点 ID 配置，director 可以为每个节点分配一个特定的自定义主机名。如把系统的主机名设置为 **rack2-row12** 来表示它所在的位置。

为了自定义节点主机名，在环境文件中（如 第 6.3.1 节 "分配特定的节点 ID" 中的 **scheduler_hints_env.yaml** 文件）使用 **HostnameMap** 参数。例如：

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  NovaComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-compute-0: overcloud-compute-prod-abc-0
```

在 **parameter_defaults** 的部分中定义 **HostnameMap**，使用 **HostnameFormat** 参数设置 head 定义的原始主机名的映射信息（如 **overcloud-controller-0**），第二个值是那个节点的自定义主机名（如 **overcloud-controller-prod-123-0**）。

通过使用这个方法以及节点 ID 的放置功能，每个节点将会有一个自定义主机名。

6.3.3. 分配可预测的 IP

为了可以对环境进行更好的控制，director 可以在每个网络中为 Overcloud 节点分配特定的 IP。使用核心 Heat 模板集合中的 **environments/ips-from-pool-all.yaml** 环境文件，把这个文件复制到 **stack** 用户的 **templates** 目录中。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/ips-from-
pool-all.yaml ~/templates/.
```

ips-from-pool-all.yaml 文件包括两个主要部分。

第一部分是一组 **resource_registry** 用来覆盖默认设置。它们用来通知 director 在一个节点类型的指定端口上使用一个特定的 IP。修改每个资源来使用到代表它们的模板的绝对路径。例如：

```
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external_from_pool.yaml
```

```
  OS::TripleO::Controller::Ports::InternalApiPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api_from_pool.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_from_pool.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_from_pool.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant_from_pool.yaml
```

默认的设置是，所有节点类型上的所有网络都使用预先配置的 IP。如果需要设置特定的网络或节点类型使用默认的 IP 分配设置，在环境文件中把与那个节点类型或网络相关的 **resource_registry** 项删除。

第二个部分是 parameter_defaults，它代表了实际分配的 IP 地址。每个节点类型都有一个相关的参数：

- **ControllerIPs** 代表 Controller 节点。
- **NovaComputeIPs** 代表 Compute 节点。
- **CephStorageIPs** 代表 Ceph Storage 节点。
- **BlockStorageIPs** 代表 Block Storage 节点。
- **SwiftStorageIPs** 代表 Object Storage 节点。

每个参数就是一个网络名称到地址列表的映射信息。每个网络类型最少具有的地址数量不能少于它们中的节点数量。director 会顺序分配地址。每个类型的第一个节点会被分配相关列表中的第一个地址，第二个节点被分配相关列表中的第二个地址，以此类推。

例如，如果一个 Overcloud 将要包括三个 Ceph Storage 节点，CephStorageIPs 参数的设置会和以下类似：

```
CephStorageIPs:
  storage:
  - 172.16.1.100
  - 172.16.1.101
  - 172.16.1.102
  storage_mgmt:
  - 172.16.3.100
  - 172.16.3.101
  - 172.16.3.102
```

第一个 Ceph Storage 节点会接收两个地址：172.16.1.100 和 172.16.3.100。第二个节点接收 172.16.1.101 和 172.16.3.101，第三个节点接收 172.16.1.102 和 172.16.3.102。相同原则适用于其它节点类型。

确定所选的 IP 地址位于环境文件中定义的每个网络的地址分配池之外（请参阅 第 6.2.2 节 “创建一个网络环境文件”）。例如，确定 **internal_api** 分配的地址位于 **InternalApiAllocationPools** 的范围之外，这会避免与每个网络所选 VIP 的冲突。同样，确定分配的 IP 地址与为外部负载平衡环境定义的 VIP 配置没有冲突（请参阅 第 6.5 节 “配置外部负载平衡”）。

要在部署过程中应用这个配置，在 **openstack overcloud deploy** 命令中包括环境文件。如果使用网络隔离（请参阅 第 6.2 节 “分离网络”），在这个文件的后面包括 **network-isolation.yaml** 文件。例如：

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/ips-from-pool-all.yaml [OTHER OPTIONS]
```

6.3.4. 分配可预测的虚拟 IP

除了为每个节点定义可预测的 IP 地址，director 还可以为集群的服务提供相似的可预测的虚拟 IP（VIP）。要实现这个功能，对 第 6.2.2 节 “创建一个网络环境文件” 中的网络环境文件进行如下修改：

1. 在 **resource_registry** 项中添加一组 **OS::TripleO::Network::Ports** 资源定义：

   ```
   resource_registry:
     OS::TripleO::Network::Ports::NetVipMap: /usr/share/openstack-
   tripleo-heat-templates/network/ports/net_vip_map_external.yaml
     OS::TripleO::Network::Ports::ExternalVipPort:
   /usr/share/openstack-tripleo-heat-
   templates/network/ports/noop.yaml
     OS::TripleO::Network::Ports::InternalApiVipPort:
   /usr/share/openstack-tripleo-heat-
   templates/network/ports/noop.yaml
     OS::TripleO::Network::Ports::StorageVipPort:
   /usr/share/openstack-tripleo-heat-
   templates/network/ports/noop.yaml
     OS::TripleO::Network::Ports::StorageMgmtVipPort:
   /usr/share/openstack-tripleo-heat-
   templates/network/ports/noop.yaml
     OS::TripleO::Network::Ports::RedisVipPort:
   /usr/share/openstack-tripleo-heat-
   templates/network/ports/from_service.yaml
   ```

   这会覆盖主资源注册表中的 VIP 端口定义。

2. 在 **parameter_defaults** 项中添加 VIP 参数：

   ```
   parameter_defaults:
     ...
     # Predictable VIPs
     ControlPlaneIP: 192.168.0.230
     ExternalNetworkVip: 10.1.1.190
     InternalApiNetworkVip: 172.16.0.30
     StorageNetworkVip: 172.18.0.30
     StorageMgmtNetworkVip: 172.19.0.40
     ServiceVips:
       redis: 172.16.0.31
   ```

重要

如果使用这个功能，需要为所有网络分配一个 VIP。

从它们相关的分配池范围内选择这些 IP。例如，从 **InternalApiAllocationPools** 范围中选 **InternalApiNetworkVip**。一个例外是 **ControlPlaneIP**，您需要从 **undercloud.conf** 文件所定义的分配范围之外进行选择。

6.4. 配置容器化的 COMPUTE 节点

director 提供了一个选项来把 OpenStack 的容器化项目（kolla）的服务集成到 Overcloud 的 Compute 节点上。这包括使用 Red Hat Enterprise Linux Atomic Host 作为基本操作系统和独立的容器来运行不同 OpenStack 服务的 Compute 节点。

重要

容器化 Compute 节点现在还是一个技术预览（Technology Preview ）。技术预览将不被 Red Hat Subscription Service Level Agreements（SLAs）所完全支持，也不能保证它的所有功能都可以正常运行。Technology Preview 功能并不是为当前的生产环境所提供的，但用户可以通过这些功能来尽早接触将来会被使用的新产品技术，同时可以反馈您的意见来完善产品的开发。如需了解更多与对技术预览的支持相关的信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

director 的核心 Heat 模板集合包括环境文件来帮助配置容器化的 Compute 节点。这些文件包括：

- **docker.yaml** - 配置容器化 Compute 节点的主要环境文件。
- **docker-network.yaml** - 没有网络隔离的容器化 Compute 节点网络的环境文件。
- **docker-network-isolation.yaml** - 使用网络隔离的容器化 Compute 节点的环境文件。

6.4.1. 容器化 Compute 环境文件（docker.yaml）

docker.yaml 是包括容器化 Compute 节点配置的主环境文件。它包括了 **resource_registry** 中的项：

```
resource_registry:
  OS::TripleO::ComputePostDeployment: ../docker/compute-post.yaml
  OS::TripleO::NodeUserData:
../docker/firstboot/install_docker_agents.yaml
```

OS::TripleO::NodeUserData

在第一次引导时，提供一个使用自定义配置的 Heat 模板。在这种情况下，它会在第一次引导时在 Compute 节点上安装 **openstack-heat-docker-agents** 容器。这个容器提供了一组初始脚本来配置容器化 Compute 节点，以及 Heat hook 来和 director 进行通讯。

OS::TripleO::ComputePostDeployment

提供一组 Compute 节点的后配置资源的 Heat 模板。这包括了一个软件配置资源，它为 Puppet 提供了一组 **tags**：

```
ComputePuppetConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: puppet
    options:
```

```
          enable_hiera: True
          enable_facter: False
          tags:
package,file,concat,file_line,nova_config,neutron_config,neutron_
agent_ovs,neutron_plugin_ml2
        inputs:
        - name: tripleo::packages::enable_install
          type: Boolean
          default: True
        outputs:
        - name: result
        config:
          get_file: ../puppet/manifests/overcloud_compute.pp
```

这些 tag 定义了要传递给 **openstack-heat-docker-agents** 容器的 Puppet 模板。

docker.yaml 文件包括了一个名为 **NovaImage** 的 **parameter**，它会在配置 Compute 节点时使用一个不同的镜像（**atomic-image**）替换标准的 **overcloud-full** 镜像。第 6.4.2 节 “上传 Atomic Host 镜像” 介绍了上传这个新镜像的方法。

docker.yaml 文件还包括了一个 **parameter_defaults** 部分，它定义了 Docker 的注册表以及 Compute 节点服务要使用的镜像。您可以修改这个部分来使用本地的注册表，而不使用默认的 registry.access.redhat.com。如需了解配置一个本地注册表的信息，请参阅 第 6.4.3 节 “使用本地注册表”。

6.4.2. 上传 Atomic Host 镜像

director 需要把一个 Red Hat Enterprise Linux 7 Atomic Host 的云镜像（Cloud Image）导入到它的镜像存储中（**atomic-image**）。这是因为，在 Overcloud 创建的 provisioning 阶段，Compute 节点需要这个镜像作为基础 OS。

从 Red Hat Enterprise Linux 7 Atomic Host 产品页（https://access.redhat.com/downloads/content/271/ver=/rhel---7/7.2.2-2/x86_64/product-software）中下载 **Cloud Image**，把它保存到 **stack** 用户的家目录下的 **images** 子目录中。

当镜像下载完成后，使用 **stack** 用户把镜像导入到 director。

```
$ glance image-create --name atomic-image --file ~/images/rhel-atomic-
cloud-7.2-12.x86_64.qcow2 --disk-format qcow2 --container-format bare
```

这会导入这个镜像，以及其它 Overcloud 镜像。

```
$ glance image-list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 27b5bad7-f8b2-4dd8-9f69-32dfe84644cf | atomic-image           |
| 08c116c6-8913-427b-b5b0-b55c18a01888 | bm-deploy-kernel       |
| aec4c104-0146-437b-a10b-8ebc351067b9 | bm-deploy-ramdisk      |
| 9012ce83-4c63-4cd7-a976-0c972be747cd | overcloud-full         |
| 376e95df-c1c1-4f2a-b5f3-93f639eb9972 | overcloud-full-initrd  |
| 0b5773eb-4c64-4086-9298-7f28606b68af | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

6.4.3. 使用本地注册表

默认的设置是使用红帽的容器注册表来进行镜像下载。但是，为了节省带宽，也可以在 Overcloud 的创建过程中使用一个本地的注册表。

您可以选择使用一个存在的本地注册表，或安装一个新的本地注册表。要安装一个新的注册表，请参阅 Getting Started with Containers 文档中的 Chapter 2. Get Started with Docker Formatted Container Images。

把所需的所有镜像导入到注册表中：

```
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
compute:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-neutron-
openvswitch-agent:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
db-server:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-heat-docker-
agents:latest
```

在获得镜像后，把它们标记到适当的注册表主机：

```
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
compute:latest localhost:8787/registry.access.redhat.com/openstack-
nova-compute:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
localhost:8787/registry.access.redhat.com/openstack-data:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest localhost:8787/registry.access.redhat.com/openstack-
nova-libvirt:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-neutron-
openvswitch-agent:latest
localhost:8787/registry.access.redhat.com/openstack-neutron-
openvswitch-agent:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
```

```
db-server:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-heat-docker-
agents:latest localhost:8787/registry.access.redhat.com/openstack-heat-
docker-agents:latest
```

把它们推到注则表：

```
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-compute:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
data:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-libvirt:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
neutron-openvswitch-agent:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
heat-docker-agents:latest
```

在 **templates** 子目录中创建一个主 **docker.yaml** 环境文件：

```
$ cp /usr/share/openstack-tripleo-heat-
templates/environments/docker.yaml ~/templates/.
```

编辑这个文件，修改 **resource_registry** 来使用绝对路径：

```
resource_registry:
  OS::TripleO::ComputePostDeployment: /usr/share/openstack-tripleo-
heat-templates/docker/compute-post.yaml
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/docker/firstboot/install_docker_agents.yaml
```

把 **parameter_defaults** 中的 **DockerNamespace** 设置为您的注册表的 URL。另外，还需要把 **DockerNamespaceIsRegistry** 设置为 **true**。例如：

```
parameter_defaults:
  DockerNamespace: registry.example.com:8787/registry.access.redhat.com
  DockerNamespaceIsRegistry: true
```

现在，本地的注册表包括了所需的 docker 镜像，容器化的 Compute 现在被设置为使用这个注册表。

6.4.4. 在 Overcloud 部署中包括环境文件

在运行 Overcloud 创建命令时，在 **openstack overcloud deploy** 命令中包括容器化 Compute 节点的主环境文件（**docker.yaml**）和网络环境文件（**docker-network.yaml**）。例如：

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/docker.yaml -e
/usr/share/openstack-tripleo-heat-templates/environments/docker-
network.yaml [OTHER OPTIONS] ...
```

容器化的 Compute 节点也可以在一个网络分离的 Overcloud 环境中正常工作。这也需要主环境文件和网络分离文件（**docker-network-isolation.yaml**）。在 第 6.2 节 “分离网络” 介绍的网络分离文件前添加这些文件。例如：

```
openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/docker.yaml -e /usr/share/openstack-
tripleo-heat-templates/environments/docker-network-isolation.yaml -e
/usr/share/openstack-tripleo-heat-templates/environments/net-single-
nic-with-vlans.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml [OTHER OPTIONS] ...
```

director 创建了一个带有容器化 Compute 节点的 Overcloud。

6.5. 配置外部负载平衡

一个 Overcloud 会使用多个 Controller 来组成一个高可用性集群，从而确保 OpenStack 服务获得最大的操作性能。另外，集群还可以实现对 OpenStack 服务访问的负载平衡（在各个 Controller 节点间平衡分配访问操作，减少每个节点的服务器负载）。这个功能也可以通过使用一个外部的负载平衡系统来实现。例如，一个机构可以选择使用自己的、基于硬件的负载平衡系统来在 Controller 节点间分配访问流量。

如需了解更多配置外部负载平衡系统的信息，请参阅 External Load Balancing for the Overcloud。

6.6. 配置 IPV6 网络

默认情况下，Overcloud 使用 IPv4 来配置服务端点（endpoint）。但是，Overcloud 同样支持 IPv6 端点，这一点对支持 IPv6 基础架构的机构会非常有用。director 包括了一组环境文件来帮助创建基于 IPv6 的 Overcloud。

如需了解更多在 Overcloud 中配置 IPv6 的信息，请参阅 IPv6 Networking for the Overcloud。

6.7. 配置 NFS 存储

本节介绍了配置 Overcloud 来使用 NFS 共享的方法。整个安装和配置的过程基于对核心 Heat 模板中的一个环境文件的修改。

在 **/usr/share/openstack-tripleo-heat-templates/environments/** 中，核心 heat 模板集合包括了一组环境文件。这些环境文件可以帮助对由 director 创建的 Overcloud 所支持的特定文件进行定制配置。其中，包括一个用来对存储进行配置的环境文件（**/usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml**）。把这个文件复制到 **stack** 用户的模板目录中。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-
environment.yaml ~/templates/.
```

环境文件包括了一些参数，它们可以帮助配置 Openstack 的块和镜像存储、cinder 和 glance 的不同存储选项。在这个示例中，您把 Overcloud 配置为使用一个 NFS 共享。修改以下参数：

CinderEnableIscsiBackend

启用 iSCSI 后端。把它设置为 **`false`**。

CinderEnableRbdBackend

启用 Ceph 存储后台。把它设置为 **`false`**。

CinderEnableNfsBackend

启动 NFS 后端。把它设置为 **`true`**。

NovaEnableRbdBackend

为 Nova 临时存储（ephemeral storage）启用 Ceph 存储。把它设置为 **`false`**。

GlanceBackend

定义 Glance 的后端。把它设为 **`file`** 来为镜像使用基于文件的存储。Overcloud 会为 Glance 在一个挂载的 NFS 共享中存储这些文件。

CinderNfsMountOptions

卷存储的 NFS 挂载选项。

CinderNfsServers

为卷共享挂载的 NFS 共享。例如，192.168.122.1:/export/cinder。

GlanceFilePcmkManage

为镜像存储启用 Pacemaker 来管理共享。如果被禁用，Overcloud 会把镜像存储在 Controller 节点的文件系统中。把它设置为 **`true`**。

GlanceFilePcmkFstype

定义 Pacemaker 用来进行镜像存储的文件系统类型。把它设为 **`nfs`**。

GlanceFilePcmkDevice

挂载的、用于镜像存储的 NFS 共享。例如：192.168.122.1:/export/glance。

GlanceFilePcmkOptions

用于镜像存储的 NFS 挂载选项。

环境文件的选项应该和以下类似：

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: 'file'

  CinderNfsMountOptions: 'rw,sync'
  CinderNfsServers: '192.0.2.230:/cinder'

  GlanceFilePcmkManage: true
  GlanceFilePcmkFstype: 'nfs'
  GlanceFilePcmkDevice: '192.0.2.230:/glance'
  GlanceFilePcmkOptions:
```

```
'rw,sync,context=system_u:object_r:glance_var_lib_t:s0'
```

重要

在 **GlanceFilePcmkOptions** 参数中包括 **context=system_u:object_r:glance_var_lib_t:s0** 允许 glance 访问 **/var/lib** 目录。如果没有这个 SELinux 设置，glance 将无法写挂载点。

这些参数被集成在一起作为 heat 模板集合的一部分。这样设置它们会创建两个 cinder 和 glance 使用的 NFS 挂载点。

保存这个文件。

6.8. 配置 CEPH 存储

director 提供了两个主要的方法把 Red Hat Ceph Storage 集成到 Overcloud。

创建一个带有自己的 Ceph Storage Cluster 的 Overcloud

director 在创建 Overcloud 时可以创建一个 Ceph Storage Cluster。director 会创建一组使用 Ceph OSD 来存储数据的 Ceph Storage 节点。另外，director 还会在 Overcloud 的 Controller 节点上安装 Ceph Monitor 服务。这意味着，如果创建了一个带有 3 个高可用性 controller 节点的 Overcloud，Ceph Monitor 服务也会成为高可用性服务。

把一个已存在的 Ceph Storage 集成到 Overcloud

如果您已有一个 Ceph Storage Cluster，则可以在部署 Overcloud 的时候把它集成到 Overcloud。这意味着，您可以管理和扩展 Overcloud 配置以外的集群。

如需了解更多相关信息，请参阅 Red Hat Ceph Storage for the Overcloud 中的相关内容。

6.9. 配置第三方存储

director 包括了一些环境文件来帮助配置第三方存储供应商。这包括：

Dell Storage Center

部署一个单一的 Dell Storage Center 后台作为 Block Storage（cinder）服务。

环境文件位于 **/usr/share/openstack-tripleo-heat-templates/environments/cinder-eqlx-config.yaml**。

如需了解更详细的配置信息，请参阅 Dell Storage Center Back End Guide。

Dell EqualLogic

部署一个单独的 Dell EqualLogic 后台作为 Block Storage（cinder）服务。

环境文件位于 **/usr/share/openstack-tripleo-heat-templates/environments/cinder-eqlx-config.yaml**。

如需了解更详细的配置信息，请参阅 Dell EqualLogic Back End Guide。

NetApp Block Storage

部署一个 NetApp storage appliance 作为 Block Storage（cinder）服务的后端。

环境文件位于 **/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml/cinder-netapp-config.yaml**。

如需了解详细的配置信息，请参阅 NetApp Block Storage Back End Guide。

6.10. 在 OVERCLOUD 中启用 SSL/TLS

默认情况下，Overcloud 使用未加密的端点（endpoints）提供相关的服务。因此，Overcloud 的配置需要一个额外的环境文件来为它的 Public API 端点启用 SSL/TLS。

注意

这个过程只为 Public API 端点启用 SSL/TLS。Internal API 和 Admin API 仍然没有加密。

这个过程需要网络分离来为 Public API 定义端点。如需了解与网络分离相关的信息，请参阅 第 6.2 节 "分离网络"。

请确认已有一个私人密钥并创建了证书认证机构（CA）。如需了解更多与创建 SSL/TLS 密钥和证书认证机构文件的信息，请参阅 附录 A, *SSL/TLS 证书配置*。

6.10.1. 启用 SSL/TLS

从 Heat 模板集合中复制 **enable-tls.yaml** 环境文件：

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/environments/enable-tls.yaml ~/templates/.
```

编辑这个文件，对以下参数进行修改：

SSL证书

把证书文件的内容复制到 **SSLCertificate** 参数中。例如：

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----

MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

重要

证书授权内容中的所有新行都需要有相同的行缩进。

SSL 密钥

把私人密钥的内容复制到 **SSLKey** 参数。例如：

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO9hkJZnGP6qb6wtYUoy1bVP7
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```

重要

私人密钥的内容中的所有新行都需要有相同的行缩进。

EndpointMap

EndpointMap 包括了使用 HTTPS 和 HTTP 的服务的映射信息。如果 SSL 使用 DNS，不要修改这个部分的默认设置。但是，如果使用一个 IP 地址作为 SSL 证书的常规名（请参阅 *附录 A, SSL/TLS 证书配置*），使用 **IP_ADDRESS** 替换所有 **CLOUDNAME** 实例。运行以下命令：

```
$ sed -i 's/CLOUDNAME/IP_ADDRESS/' ~/templates/enable-tls.yaml
```

重要

不要使用实际的值替换 **IP_ADDRESS** 和 **CLOUDNAME**，Heat 会在 Overcloud 创建的过程中替换这些变量。

OS::TripleO::NodeTLSData

把 **OS::TripleO::NodeTLSData:** 的资源路径改为一个绝对路径：

```
resource_registry:
  OS::TripleO::NodeTLSData: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/tls/tls-cert-inject.yaml
```

6.10.2. 注入一个 Root 证书

如果证书的签发者不在 Overcloud 镜像中的默认的 trust store 中，则需要把证书"注入"到 Overcloud 镜像中。从 heat 模板集合中复制 **inject-trust-anchor.yaml** 环境文件：

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/environments/inject-trust-anchor.yaml ~/templates/.
```

编辑这个文件，对以下参数进行修改：

SSLRootCertificate

把 root 证书授权文件的内容复制到 **SSLRootCertificate** 参数。例如：

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

重要

证书授权内容中的所有新行都需要有相同的行缩进。

OS::TripleO::NodeTLSCAData

把 **OS::TripleO::NodeTLSCAData:** 的资源路径改为一个绝对路径：

```
resource_registry:
  OS::TripleO::NodeTLSCAData: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/tls/ca-inject.yaml
```

6.10.3. 配置 DNS 端点

如果使用 DNS 主机名通过 SSL/TLS 来访问 Overcloud，创建一个新环境文件（**~/templates/cloudname.yaml**）来定义 Overcloud 端点的主机名。使用以下参数：

CloudName

Overcloud 端点的 DNS 主机名。

DnsServers

使用的 DNS 服务器列表。配置的 DNS 服务器需要包括一个配置的 **CloudName** 的项，它需要和 Public API 的 IP 地址相匹配。

以下是这个文件的一个示例：

```
parameter_defaults:
  CloudName: overcloud.example.com
  DnsServers: ["10.0.0.1"]
```

6.10.4. 在 Overcloud 创建期间添加环境文件

在 第 7 章 *创建 Overcloud* 中介绍的部署命令（**openstack overcloud deploy**）中使用 **-e** 选项来添加环境文件。使用以下顺序在这个部分添加环境文件：

- 启用 SSL/TLS 的环境文件（**enable-tls.yaml**）
- 设置 DNS 主机名的环境文件（**cloudname.yaml**）
- 注入 root 证书授权的环境文件（**inject-trust-anchor.yaml**）

例如：

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/enable-tls.yaml -e ~/templates/cloudname.yaml -e
~/templates/inject-trust-anchor.yaml
```

6.11. 配置基础参数

Overcloud 的 Heat 模板包括了一组基础参数，您可以在运行 **openstack overcloud deploy** 命令前配置它们。把这些基础参数包括在一个环境文件的 **parameter_defaults** 项中，并在 **openstack overcloud deploy** 命令中使用这个环境文件。

如需了解 Overcloud 基础参数的完整列表，请参阅 附录 D, *基础参数*。

示例 1：配置时区

在环境文件中把时区设置为 **Japan**：

```
parameter_defaults:
  TimeZone: 'Japan'
```

示例 2：禁用第 3 层高可用性（L3HA）

如果需要为 OpenStack Networking 禁用第 3 层高可用性功能（L3HA），把以下内容添加到环境文件中：

```
parameter_defaults:
  NeutronL3HA: False
```

示例 3：配置 Telemetry Dispatcher

OpenStack Telemetry（**ceilometer**）服务包括了一个时间序列数据存储的新组件（**gnocchi**）。现在，可以在 Red Hat OpenStack Platform 中把默认的 Ceilometer dispatcher 切换为使用新组件，而不是使用标准的数据库。这个设置是通过 **CeilometerMeterDispatcher** 实现的，您可以把它设置为以下之一：

- **database** - 为 Ceilometer dispatcher 使用标准数据库。这是默认选项。
- **gnocchi** - 为 Ceilometer dispatcher 使用新的时间序列数据库。

要切换为使用时间序列数据库，在环境文件中添加以下内容：

```
parameter_defaults:
  CeilometerMeterDispatcher: gnocchi
```

示例 4：配置 RabbitMQ File Descriptor Limit

在特定配置中，您可能需要为 RabbitMQ 服务器提高文件描述符（file descriptor）的数量限制。根据需要，为 **RabbitFDLimit** 参数设置适当的值：

```
parameter_defaults:
  RabbitFDLimit: 65536
```

6.12. 注册 OVERCLOUD

Overcloud 提供了把节点注册到 Red Hat Content Delivery Network、Red Hat Satellite 5 server 或 Red Hat Satellite 6 server 的方法。您可以选择使用环境变量实现它，也可以使用命令行来实现。

6.12.1. 方法 1：命令行

部署命令（**openstack overcloud deploy**）使用一组选项来定义您的注册详情。第 7.1 节 “设置 Overcloud 参数” 中的表格包括了这些选项以及它们的描述信息。使用 第 7 章 *创建 Overcloud* 中的部署命令时包括这些选项，例如：

```
# openstack overcloud deploy --templates --rhel-reg --reg-method
satellite --reg-sat-url http://example.satellite.com  --reg-org MyOrg -
-reg-activation-key MyKey --reg-force [...]
```

6.12.2. 方法 2 - 环境文件

从 Heat 模板集合中复制注册文件：

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/extraconfig/pre_deploy/rhel-registration ~/templates/.
```

编辑 **~/templates/rhel-registration/environment-rhel-registration.yaml**，根据您具体的注册情况和方法，修改以下值：

rhel_reg_method

选择注册方法。可以是 **portal**、**satellite** 或 **disable**。

rhel_reg_type

注册的单元类型。如果注册为一个 **system**，把它设为空

rhel_reg_auto_attach

为系统自动附加兼容的订阅。如需启用这个功能，把它设置为 **true**。

rhel_reg_service_level

自动附加所使用的服务级别。

rhel_reg_release

使用这个参数来为自动附加设置发行版本。如果设置为空，则使用从 Red Hat Subscription Manager 获得的默认值。

rhel_reg_pool_id

使用的订阅池 ID。在没有使用自动附加功能时使用这个参数。

rhel_reg_sat_url

注册 Overcloud 节点的 Satellite 服务器的基本 URL。这个参数需要使用 Satellite 的 HTTP URL 而不是 HTTPS URL。例如，http://satellite.example.com，而不是 https://satellite.example.com。Overcloud 的创建过程会使用这个 URL 来决定服务器是 Red Hat Satellite 5 还是 Red Hat Satellite 6。如果是 Red Hat Satellite 6 服务器，Overcloud 会获

得 **katello-ca-consumer-latest.noarch.rpm** 文件，使用 **subscription-manager** 进行注册，并安装 **katello-agent**。如果是一个 Red Hat Satellite 5 服务器，Overcloud 会获得 **RHN-ORG-TRUSTED-SSL-CERT** 文件，并使用 **rhnreg_ks** 进行注册。

rhel_reg_server_url

订阅服务使用的主机名。默认是 Customer Portal Subscription Management（subscription.rhn.redhat.com）。如果这个选项没有被使用，系统会使用 Customer Portal Subscription Management 进行注册。订阅服务器 URL 的格式是 https://hostname:port/prefix。

rhel_reg_base_url

获得系统更新的内容服务器的主机名，它的默认值是 https://cdn.redhat.com。因为 Satellite 6 主机有它自己的内容，注册到 Satellite 6 的系统需要在这个参数中指定 URL。基本 URL 的格式是 https://hostname:port/prefix。

rhel_reg_org

注册的机构

rhel_reg_environment

在所选机构内使用的环境

rhel_reg_repos

以逗号分隔的软件仓库列表。如需了解更详细的信息，请参阅 第 2.5 节 “软件仓库的要求”。

rhel_reg_activation_key

注册使用的激活码。

rhel_reg_user; rhel_reg_password

注册的用户名和密码。如果可能，使用激活码进行注册。

rhel_reg_machine_name

机器名。如果使用节点的主机名，把它设为空。

rhel_reg_force

把它设置为 **true** 来强制使用您的注册选项。例如，重新注册节点。

rhel_reg_sat_repo

包括 Red Hat Satellite 6 的管理工具程序（如 **katello-agent**）的仓库。例如，**rhel-7-server-satellite-tools-6.1-rpms**。

在 第 7 章 *创建 Overcloud* 中介绍的部署命令（**openstack overcloud deploy**）中使用 **-e** 选项来添加环境文件。例如，添加 **~/templates/rhel-registration/environment-rhel-registration.yaml** 和 **~/templates/rhel-registration/rhel-registration-resource-registry.yaml**：

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/rhel-registration/environment-rhel-
registration.yaml -e /home/stack/templates/rhel-registration/rhel-
registration-resource-registry.yaml
```

重要

把注册方法设置为 **OS::TripleO::NodeExtraConfig** Heat 资源。这意味着，您只能使用这个资源进行注册。如需了解更多信息，请参阅 第 6.14 节 "自定义 Overcloud 的预配置"。

6.13. 自定义第一次引导的配置

director 提供了一个在初始创建 Overcloud 时在所有节点上进行配置操作的机制。director 使用 **cloud-init**，您可以使用 **OS::TripleO::NodeUserData** 资源类型调用它。

在这个示例中，您需要在所有节点上更新域名解析服务器来使用一个自定义的 IP 地址。首先，创建一个基本的 heat 模板（**/home/stack/templates/nameserver.yaml**），它运行一个脚本来为每个节点的 **resolv.conf** 添加一个特定的名称解析服务器（nameserver）。使用 **OS::TripleO::MultipartMime** 资源类型来发送配置脚本。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

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

  nameserver_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
        echo "nameserver 192.168.1.1" >> /etc/resolv.conf

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

接下来，创建一个环境文件（**/home/stack/templates/firstboot.yaml**），它把您的 heat 模板注册为 **OS::TripleO::NodeUserData** 资源类型。

```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

为了添加首次引导时的配置，在首次创建 Overcloud 时把环境文件添加到栈中。例如：

```
$ openstack overcloud deploy --templates -e
/home/stack/templates/firstboot.yaml
```

其中的 **-e** 把环境文件添加到 Overcloud 栈。

在所有节点首次创建并首次引导时，这些配置会被添加到所有节点上。其后包括这些模板的操作（如更新 Overcloud 栈）将不再运行这些脚本。

重要

您可以只把 **`OS::TripleO::NodeUserData`** 注册到一个 heat 模板。随后的使用会覆盖 heat 模板。

6.14. 自定义 OVERCLOUD 的预配置

Overcloud 使用 Puppet 进行 OpenStack 组件的核心配置。director 提供了一组在第一次引导完成后，核心配置开始前，提供自定义配置的资源。这些资源包括：

OS::TripleO::ControllerExtraConfigPre

在核心 Puppet 配置前，应用到 Controller 节点上的额外配置。

OS::TripleO::ComputeExtraConfigPre

在核心 Puppet 配置前，应用到 Controller 节点上的额外配置。

OS::TripleO::CephStorageExtraConfigPre

在核心 Puppet 配置前，应用到 CephStorage 节点上的额外配置。

OS::TripleO::NodeExtraConfig

在核心 Puppet 配置前，应用到所有节点角色上的额外配置。

在这个示例中，首先创建一个基本的 heat 模板（**`/home/stack/templates/nameserver.yaml`**），它运行一个脚本来为每个节点的 **`resolv.conf`** 添加一个不同的名称解析服务器（nameserver）。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string

resources:
  ExtraPreConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
```

```
            _NAMESERVER_IP_: {get_param: nameserver_ip}
  ExtraPreDeployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config: {get_resource: ExtraPreConfig}
      server: {get_param: server}
      actions: ['CREATE','UPDATE']

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on
changes
    value: {get_attr: [ExtraPreDeployment, deploy_stdout]}
```

重要

server 参数是应用配置的服务器列表，它由父模板提供。这个参数在所有预配置模板中都是必需的。

接下来，创建一个环境文件（**/home/stack/templates/pre_config.yaml**），它会把您的 heat 模板注册为 **OS::TripleO::NodeExtraConfig** 资源类型。

```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml
  parameter_defaults:
  nameserver_ip: 192.168.1.1
```

为了应用配置，在创建或更新 Overcloud 时把环境文件加入到栈。例如：

```
$ openstack overcloud deploy --templates -e
/home/stack/templates/pre_config.yaml
```

这会在初始创建的主配置开始前，或以后的更新过程的主配置开始前，在所有节点中应用配置。

重要

您可以只把这些资源注册到一个 Heat 模板。以后的使用会覆盖 heat 模板来使用每个资源。

6.15. OVERCLOUD 创建后的自定义配置

在创建完 Overcloud 后，您可能需要在初始创建时，或以后对 Overcloud 进行更新时添加以下额外的配置。在这种情况下，您可以使用 **OS::TripleO::NodeExtraConfigPost** 资源来应用使用标准的 **OS::Heat::SoftwareConfig** 类型的配置。这会在主 Overcloud 配置完成后应用额外的配置。

在这个示例中，首先创建一个基本的 heat 模板（**/home/stack/templates/nameserver.yaml**），它运行一个脚本来为每个节点的 **resolv.conf** 添加一个不同的名称解析服务器（nameserver）。

```
heat_template_version: 2014-10-16
```

```
description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string

resources:
  ExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  ExtraDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
      servers:  {get_param: servers}
      config: {get_resource: ExtraConfig}
      actions: ['CREATE','UPDATE']
```

重要

servers 参数是应用配置的服务器列表，它由父模板提供。这个参数在所有 **OS::TripleO::NodeExtraConfigPost** 模板中都是必需的。

接下来，创建一个环境文件（**/home/stack/templates/post_config.yaml**），它会把我们的 Heat 模板注册为 **OS::TripleO::NodeExtraConfigPost:** 资源类型。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost:
/home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

为了应用配置，在创建或更新 Overcloud 时把环境文件加入到栈。例如：

```
$ openstack overcloud deploy --templates -e
/home/stack/templates/post_config.yaml
```

这会在初始创建的主配置完成后，或以后的更新过程的主配置完成后，在所有节点中应用配置。

重要

您可以只把 **`OS::TripleO::NodeExtraConfigPost`** 注册到一个 heat 模板。随后的使用会覆盖 heat 模板。

6.16. 自定义 PUPPET 配置数据

Heat 模板集合包括一组参数来把额外的配置传递到特定的节点类型。这些参数把相关的配置保存为 hieradata 来作为节点的 Puppet 配置。这些参数包括：

ExtraConfig
: 添加到所有节点的配置

controllerExtraConfig
: 添加到所有 Controller 节点的配置。

NovaComputeExtraConfig
: 添加到所有 Compute 节点的配置。

BlockStorageExtraConfig
: 添加到所有 Block Storage 节点的配置。

ObjectStorageExtraConfig
: 添加到所有 Object Storage 节点的配置。

CephStorageExtraConfig
: 添加到所有 Ceph Storage 节点的配置。

为了把额外的配置添加到部署后的配置过程中，创建一个在 **`parameter_defaults`** 的部分中包括这些参数的环境文件。例如，把 Compute 主机的保留内存增加到 1024 MB，把 VNC 的键盘输入设置为日语：

```
parameter_defaults:
  NovaComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

在运行 **`openstack overcloud deploy`** 时包括这个环境文件。

重要

您只能定义每个参数一次。以后的使用会覆盖以前的值。

6.17. 应用自定义 PUPPET 配置

在特定情况下，您可能会需要在 Overcloud 节点上安装并配置一些额外的组件。您可以通过在主配置完成后，在节点上应用一个自定义 Puppet manifest 来达到这个目的。作为一个基本的例子，您可以在每个节点上安装 **motd**。这会首先创建一个 Heat 模板（**/home/stack/templates/custom_puppet_config.yaml**）来启动 Puppet 配置。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: motd.pp}
      group: puppet
      options:
        enable_hiera: True
        enable_facter: False

  ExtraPuppetDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
```

这会在模板内包括 **/home/stack/templates/motd.pp**，并把它传递给节点进行配置。**motd.pp** 文件本身包括了我们的 Puppet 类来进行安装和配置 **motd**。

接下来，创建一个环境文件（**/home/stack/templates/puppet_post_config.yaml**），它会把我们的 Heat 模板注册为 **OS::TripleO::NodeExtraConfigPost:** 资源类型。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost:
/home/stack/templates/custom_puppet_config.yaml
```

最后，在创建或更新 Overcloud 栈时包括这个环境文件：

```
$ openstack overcloud deploy --templates -e
/home/stack/templates/puppet_post_config.yaml
```

这会把 **motd.pp** 中的配置应用到 Overcloud 的所有节点上。

6.18. 使用定制的核心 HEAT 模板

在创建 Overcloud 时，director 会使用一组核心的 heat 模板。您可以把这些标准的 heat 模板复制到一个本地目录中，使用这些模板来创建您自己的 Overcloud。

把 **/usr/share/openstack-tripleo-heat-templates** 中的 heat 模板复制到 **stack** 用户的模板目录中：

```
$ cp -r /usr/share/openstack-tripleo-heat-templates ~/templates/my-overcloud
```

这会创建 Overcloud Heat 模板的一个克隆。在运行 **openstack overcloud deploy** 时，我们使用了 **--templates** 选项来指定本地的模板目录。请参阅 第 7 章 *创建 Overcloud*。

注意

如果没有为 **--templates** 选项设置值，director 会使用默认的模板目录（**/usr/share/openstack-tripleo-heat-templates**）。

重要

红帽会在后续的发行版本中提供对 heat 模板的更新。使用一个经过修改过的模板集合会造成您的定制版本和位于 **/usr/share/openstack-tripleo-heat-templates** 中的原始版本的不同。红帽推荐使用以下章节中介绍的方法来进行配置，而不是直接修改 heat 模板集合：

- 第 6.14 节 “自定义 Overcloud 的预配置”
- 第 6.15 节 “Overcloud 创建后的自定义配置”
- 第 6.16 节 “自定义 Puppet 配置数据”
- 第 6.17 节 “应用自定义 Puppet 配置”

在创建 heat 模板集合的一个副本时，您需要使用一个版本控制工具（如 **git**）来记录对模板的改变。

第 7 章 创建 OVERCLOUD

创建 OpenStack 环境的最后一个阶段是运行 **`openstack overcloud deploy`** 命令进行创建。在运行这个命令前，您需要已经对关键的选项，以及如何包括自定义环境文件有所了解。本章将讨论 **`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。

7.1. 设置 OVERCLOUD 参数

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

表 7.1. 部署参数

参数	描述	示例
--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。	~/templates/my-overcloud
--stack STACK	创建或更新的栈的名称	overcloud
-t [TIMEOUT], --timeout [TIMEOUT]	部署超时时间（分钟）	240
--control-scale [CONTROL_SCALE]	扩展的 Controller 节点数量	3
--compute-scale [COMPUTE_SCALE]	扩展的 Compute 节点数量	3
--ceph-storage-scale [CEPH_STORAGE_SCALE]	扩展的 Ceph 节点数量	3

--block-storage-scale [BLOCK_STORAGE_SCALE]	扩展的 Cinder 节点数量	3
--swift-storage-scale [SWIFT_STORAGE_SCALE]	扩展的 Swift 节点数量	3
--control-flavor [CONTROL_FLAVOR]	Controller 节点使用的 flavor	control
--compute-flavor [COMPUTE_FLAVOR]	Compute 节点使用的 flavor	compute
--ceph-storage-flavor [CEPH_STORAGE_FLAVOR]	Ceph 节点使用的 flavor	ceph-storage
--block-storage-flavor [BLOCK_STORAGE_FLAVOR]	Cinder 节点使用的 flavor	cinder-storage
--swift-storage-flavor [SWIFT_STORAGE_FLAVOR]	Swift 存储节点使用的 flavor	swift-storage
--neutron-flat-networks [NEUTRON_FLAT_NETWORKS]	（已过时）定义在 neutron 插件中配置的平面网络（flat nework）。默认是 "datacentre"，允许外部网络创建	datacentre
--neutron-physical-bridge [NEUTRON_PHYSICAL_BRIDGE]	（已过时）在每个 hypervisor 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	br-ex
--neutron-bridge-mappings [NEUTRON_BRIDGE_MAPPINGS]	（已过时）使用的物理网桥映射逻辑。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（floating network）	datacentre:br-ex

--neutron-public-interface [NEUTRON_PUBLIC_INTERFACE]	（已过时）定义网络节点的 br-ex 中的网桥接口	nic1, eth0
--neutron-network-type [NEUTRON_NETWORK_TYPE]	（已过时）neutron 的租户网络类型	gre 或 vxlan
--neutron-tunnel-types [NEUTRON_TUNNEL_TYPES]	（已过时）Neutron 租户网络的隧道（tunnel）类型。使用逗号分隔的字符串可以指定多个值	*vxlan gre,vxlan*
--neutron-tunnel-id-ranges [NEUTRON_TUNNEL_ID_RANGES]	（已过时）可以用来进行租户网络分配的 GRE tunnel ID 的范围	1:1000
--neutron-vni-ranges [NEUTRON_VNI_RANGES]	（已过时）可以用来进行租户网络分配的 VXLAN VNI ID 范围	1:1000
--neutron-disable-tunneling	（已过时）禁用 tunneling 功能来在 Neutron 中使用 VLAN 分段网络或平面网络	
--neutron-network-vlan-ranges [NEUTRON_NETWORK_VLAN_RANGES]	（已过时）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。	datacentre:1:1000
--neutron-mechanism-drivers [NEUTRON_MECHANISM_DRIVERS]	（已过时）neutron 租户网络的驱动。默认值是 "openvswitch"。使用逗号分隔的字符串可以指定多个值	*openvswitch,l2population*
--libvirt-type [LIBVIRT_TYPE]	hypervisor 使用的虚拟类型	kvm,qemu
--ntp-server [NTP_SERVER]	用来同步时间的 NTP 服务器	pool.ntp.org
--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域扩展。	

--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。	ocuser
-e [EXTRA HEAT TEMPLATE], --extra-template [EXTRA HEAT TEMPLATE]	传递给 Overcloud 部署的额外环境文件。这个参数可以指定多次。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，一个参数出现在一个环境文件中，当这个环境文件的后续环境文件中又出现了这个参数，则后续文件中的参数设置会覆盖前面文件中的设置。	-e ~/templates/my-config.yaml
--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。	--environment-directory ~/templates
--validation-errors-fatal	Overcloud 的创建过程会进行一个部署前的检查。当设置了这个选项时，如果部署前的检查出现任何错误，整个操作会退出。我们推荐使用这个参数，因为任何错误都有可能造成您的部署失败。	
--validation-warnings-fatal	Overcloud 的创建过程会进行一个部署前的检查。当设置了这个选项时，如果部署前的检查出现任何非关键性的警告，整个操作会退出。	
--dry-run	对 Overcloud 进行验证检查，而不是实际创建 Overcloud。	
--force-postconfig	强制进行 Overcloud 部署后的配置。	--force-postconfig
--answers-file ANSWERS_FILE	到带有选项和参数的 YAML 文件的路径。	--answers-file ~/answers.yaml

--rhel-reg	把 Overcloud 节点注册到客户门户网站或 Satellite 6	
--reg-method	overcloud 节点使用的注册方法	如果使用 Red Hat Satellite 6 或 Red Hat Satellite 5，设置为 **`satellite`**；如果使用客户门户网站（Customer Portal），设置为 **`portal`**
--reg-org [REG_ORG]	注册的机构	
--reg-force	强制注册系统（即使已经注册过）	
--reg-sat-url [REG_SAT_URL]	注册 Overcloud 节点的 Satellite 服务器的基本 URL。这个参数需要使用 Satellite 的 HTTP URL 而不是 HTTPS URL。例如，http://satellite.example.com，而不是 https://satellite.example.com。Overcloud 的创建过程会使用这个 URL 来决定服务器是 Red Hat Satellite 5 还是 Red Hat Satellite 6。如果是 Red Hat Satellite 6 服务器，Overcloud 会获得 **`katello-ca-consumer-latest.noarch.rpm`** 文件，使用 **`subscription-manager`** 进行注册，并安装 **`katello-agent`**。如果是一个 Red Hat Satellite 5 服务器，Overcloud 会获得 **`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	

注意

运行以下命令获得选项的完整列表：

```
$ openstack help overcloud deploy
```

7.2. 在 OVERCLOUD 创建中包括环境文件

使用 **-e** 指定一个用来定制 Overcloud 的环境文件。您可以根据需要添加多个环境文件，但是，环境文件的顺序非常重要，后面的环境文件中定义的参数和资源会覆盖以前环境文件中定义的相同参数和资源。以下是环境文件顺序的一个示例：

- 所有网络分离文件，包括 heat 模板集合中的初始文件 - **environments/network-isolation.yaml**，然后是自定义的 NIC 配置文件。如需了解更多与网络分离相关的信息，请参阅 第 6.2 节 “分离网络”。
- 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 的栈定义的一部分。

同样，使用 **--environment-directory** 选项添加的目录中包括的环境文件也会成为 Overcloud 的栈定义的一部分。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。例如：

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

director 需要这些环境文件来进行重新部署，以及使用部署后的功能（请参阅 第 8 章 *创建 Overcloud 后执行的任务*）。没有正确包括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，手工修改的配置会被 director 的配置覆盖。

重要

保存原始的部署命令以便以后使用和修改。例如，在名为 **deploy-overcloud.sh** 的脚本文件中保存您的部署命令：

```
#!/bin/bash
openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -t 150 \
  --control-scale 3 \
  --compute-scale 3 \
  --ceph-storage-scale 3 \
  --swift-storage-scale 0 \
  --block-storage-scale 0 \
  --compute-flavor compute \
  --control-flavor control \
  --ceph-storage-flavor ceph-storage \
  --swift-storage-flavor swift-storage \
  --block-storage-flavor block-storage \
  --ntp-server pool.ntp.org \
  --neutron-network-type vxlan \
  --neutron-tunnel-types vxlan \
  --libvirt-type qemu
```

这保存了 Overcloud 部署命令的参数和环境文件以供以后使用（如修改 Overcloud 或对 Overcloud 进行扩展）。您可以根据 Overcloud 的具体情况修改并重新运行这个脚本。

7.3. OVERCLOUD 创建示例

以下是一个启动 Overcloud 创建过程的示例，它包括了自定义的环境文件：

```
$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --control-scale 3 \
  --compute-scale 3 \
  --ceph-storage-scale 3 \
  --control-flavor control \
  --compute-flavor compute \
  --ceph-storage-flavor ceph-storage \
  --ntp-server pool.ntp.org \
  --neutron-network-type vxlan \
  --neutron-tunnel-types vxlan \
```

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 /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml** - **-e** 选项为 Overcloud 部署添加了一个额外的环境文件。在这里，它是一个初始化网络分离配置的环境文件。
- **-e ~/templates/network-environment.yaml** - **-e** 选项为 Overcloud 部署添加了一个环境文件。在这里，它是在 第 6.2.2 节 “创建一个网络环境文件” 中创建的网络环境文件。
- **-e ~/templates/storage-environment.yaml** - **-e** 选项为 Overcloud 部署添加了一个额外的环境文件。在这里，它是一个用来初始存储配置的自定义环境文件。
- **--control-scale 3** - 把 Controller 节点扩展到 3 个。
- **--compute-scale 3** - 把 Compute 节点扩展到 3 个。
- **--ceph-storage-scale 3** - 把 Ceph Storage 节点扩展到 3 个。
- **--control-flavor control** - 为 Controller 节点使用一个特定的 flavor。
- **--compute-flavor compute** - 为 Compute 节点使用一个特定的 flavor。
- **--ceph-storage-flavor ceph-storage** - 为 Ceph Storage 节点使用一个特定的 flavor。
- **--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。
- **--neutron-network-type vxlan** - 在 Overcloud 中使用虚拟可扩展 LAN（Virtual Extensible LAN，简称 VXLAN）作为 neutron 网络。
- **--neutron-network-type vxlan** - 在 Overcloud 中使用虚拟可扩展 LAN（Virtual Extensible LAN，简称 VXLAN）作为 neutron 通道（tunneling）。

7.4. 监控 OVERCLOUD 的创建过程

Overcloud 创建过程开始，director 会部署您的节点。这个过程需要一些时间完成。要查看 Overcloud 创建的状态，在另外一个终端窗口中以 **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 命令会显示创建 Overcloud 的当前状态。

7.5. 访问 OVERCLOUD

director 产生一个脚本来配置和帮助验证 Overcloud 和 director 主机间的通讯。director 把这个文件保存为 **stack** 用户家目录的 **overcloudrc** 文件。运行以下命令来使用这个文件：

```
$ source ~/overcloudrc
```

这会加载从 director 主机的 CLI 访问 Overcloud 所需的环境变量。如要返回 director 主机的原始环境，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这

个用户的 SSH 访问权限。要通过 SSH 访问一个节点，找到相关节点的 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@192.0.2.23
```

7.6. 完成 OVERCLOUD 的创建

到此，基本的 Overcloud 创建过程已完成。如需了解更多创建后的功能，请参阅 第 8 章 *创建 Overcloud 后执行的任务*。

第 8 章 创建 OVERCLOUD 后执行的任务

本章介绍了在创建 Overcloud 后需要执行的任务。

8.1. 创建 OVERCLOUD TENANT 网络

Overcloud 需要为实例提供一个 Tenant 网络。source **overcloud** 并在 Neutron 中创建一个初始的 Tenant 网络。例如：

```
$ source ~/overcloudrc
$ neutron net-create default
$ neutron subnet-create --name default --gateway 172.20.1.1 default
172.20.0.0/16
```

这会创建一个名为 **default** 的基本 Neutron 网络。Overcloud 会使用一个内部的 DHCP 来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：

```
$ neutron net-list
+-----------------------+-------------+---------------------------
------------------------+
| id                    | name        | subnets
|
+-----------------------+-------------+---------------------------
------------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-
1c6f8f4f332f 172.20.0.0/16 |
+-----------------------+-------------+---------------------------
------------------------+
```

8.2. 创建 OVERCLOUD EXTERNAL 网络

在此之前，已配置了节点接口来使用 External 网络（第 6.2 节 “分离网络”），但您仍需要在 Overcloud 中创建这个网络，从而可以为实例分配浮动 IP 地址。

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
$ neutron net-create nova --router:external --provider:network_type
flat --provider:physical_network datacentre
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 10.1.1.0/24
```

在这个示例中，创建了一个名为 **nova** 的网络。Overcloud 需要使用这个特定的名称来实现浮动 IP 地址池。另外，它对验证测试也非常重要（第 8.5 节 “验证 Overcloud”）。

这个命令还会把网络映射到 **datacentre** 物理网络。作为一个默认的设置，**datacentre** 会被映射到 **br-ex** 网桥。如果在创建 Overcloud 时没有使用经过定制的 neutron 设置，这个选项就需要使用默认的设置。

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
$ neutron net-create nova --router:external --provider:network_type
vlan --provider:physical_network datacentre --provider:segmentation_id
104
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 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：

```
$ neutron net-list
+-----------------------+-------------+----------------------------
-----------------------+
| id                    | name        | subnets
|
+-----------------------+-------------+----------------------------
-----------------------+
| d474fe1f-222d-4e32... | nova        | 01c5f621-1e0f-4b9d-9c30-
7dc59592a52f 10.1.1.0/24  |
+-----------------------+-------------+----------------------------
-----------------------+
```

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：

```
$ neutron net-create ext-net --router:external --
provider:physical_network floating --provider:network_type vlan --
provider:segmentation_id 105
$ neutron subnet-create --name ext-subnet --enable_dhcp=False --
allocation-pool start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 ext-
net 10.1.2.0/24
```

8.4. 创建 OVERCLOUD 供应商网络

供应商（provider）网络就是一个被物理地附加到位于部署的 Overcloud 以外的网络。它可以是一个已存在的基础架构网络，或是一个通过路由而不是浮动 IP 来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：

```
$ neutron net-create --provider:physical_network datacentre --
provider:network_type vlan --provider:segmentation_id 201 --shared
provider_network
```

这个命令会创建一个共享的网络。另外，也可以通过不使用 --shared 来创建一个租户网络，这个网络将只对特定的租户有效。如果把一个供应商网络标记为外部，则只有操作员可以在这个网络上创建端口。

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：

```
$ neutron subnet-create --name provider-subnet --enable_dhcp=True --
allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway
10.9.101.254 provider_network 10.9.101.0/24
```

8.5. 验证 OVERCLOUD

Overcloud 使用 Tempest 进行一组集成测试。以下展示了如何使用 Tempest 来验证 Overcloud。如果从 Undercloud 运行这个测试，需要确保 Undercloud 主机可以访问 Overcloud 的内部 API 网络。例如，在 Undercloud 主机上添加一个临时的 VLAN（ID: 201）来访问内部 API 网络，使用 172.16.0.201/24 地址：

```
$ 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
```

在运行 Tempest 前，检查 **heat_stack_owner** 角色存在于 Overcloud：

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

如果角色不存在，则创建它：

```
$ keystone role-create --name heat_stack_owner
```

安装 Tempest toolset：

```
$ sudo yum install openstack-tempest
```

在 **stack** 用户的家目录中设置一个 **tempest** 目录，并复制一个本地版本的 Tempest 套件：

```
$ mkdir ~/tempest
$ cd ~/tempest
$ /usr/share/openstack-tempest-10.0.0/tools/configure-tempest-directory
```

这会创建一个本地版本的 Tempest 工具集。

在 Overcloud 创建过程完成后，director 会创建一个名为 **~/tempest-deployer-input.conf** 的文件。这个文件会提供一组和您的 Overcloud 相关的 Tempest 配置选项。运行以下命令来使用这个文件来配置 Tempest：

```
$ tools/config_tempest.py --deployer-input ~/tempest-deployer-
input.conf --debug --create identity.uri $OS_AUTH_URL
identity.admin_password $OS_PASSWORD --network-id d474fe1f-222d-4e32-
9242-cd1fefe9c14b
```

$OS_AUTH_URL 和 **$OS_PASSWORD** 这两个环境变量使用 **overcloudrc** 中设置的值。**--network-id** 是在 第 8.2 节 “创建 Overcloud External 网络” 中创建的外部网络的 UUID。

重要

这个配置脚本从 Tempest 测试中下载 Cirros 镜像。请确保这个目录可以直接访问互联网或通过代理服务器访问互联网。如果命令行操作需要通过代理进行，需要设置 **http_proxy** 环境变量。

使用以下命令运行整个 Tempest 测试：

```
$ tools/run-tests.sh
```

注意

完整运行整个 Tempest 测试可能会需要很长时间。您可以使用 **'.*smoke'** 选项来只运行其中一部分的测试。

```
$ tools/run-tests.sh '.*smoke'
```

每个测试会针对 Overcloud 运行，它们的输出会显示每个测试以及它们的结果。在相同的目录中会产生一个包括更多测试信息的文件 **tempest.log**。例如，输出结果可能会显示以下失败的测试：

```
{2}
tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_sp
ecify_keypair [18.305114s] ... FAILED
```

这会包括一个相关的、包括更多信息的日志条目。在日志中搜索使用冒号分隔的测试命名空间的最后两个部分。在这个示例中，搜索 **ServersTestJSON:test_create_specify_keypair**：

```
$ grep "ServersTestJSON:test_create_specify_keypair" tempest.log -A 4
2016-03-17 14:49:31.123 10999 INFO tempest_lib.common.rest_client [req-
a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request
(ServersTestJSON:test_create_specify_keypair): 500 POST
http://192.168.201.69:8774/v2/2f8bef15b284456ba58d7b149935cbc8/os-
keypairs 4.331s
2016-03-17 14:49:31.123 10999 DEBUG tempest_lib.common.rest_client
[req-a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request - Headers:
{'Content-Type': 'application/json', 'Accept': 'application/json', 'X-
Auth-Token': '<omitted>'}
        Body: {"keypair": {"name": "tempest-key-722237471"}}
    Response - Headers: {'status': '500', 'content-length': '128', 'x-
compute-request-id': 'req-a7a29a52-0a52-4232-9b57-c4f953280e2c',
'connection': 'close', 'date': 'Thu, 17 Mar 2016 04:49:31 GMT',
'content-type': 'application/json; charset=UTF-8'}
        Body: {"computeFault": {"message": "The server has either erred
or is incapable of performing the requested operation.", "code": 500}}
_log_request_full /usr/lib/python2.7/site-
packages/tempest_lib/common/rest_client.py:414
```

注意

使用 **`-A 4`** 选项会显示接下来的 4 行内容，它们通常是请求的 header 和 body，以及返回的 header 和 body。

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=192.0.2.205 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=192.0.2.206 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=192.0.2.207 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，您可能会需要修改 Overcloud 来添加一些功能，或改变它的运行方式。要修改 Overcloud，在您的定制环境文件和 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 第 7 章 *创建 Overcloud* 创建了一个 Overcloud，您应该重新运行以下命令：

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/network-environment.yaml -e ~/templates/storage-
environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-
scale 3 --control-flavor control --compute-flavor compute --ceph-
storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-
network-type vxlan --neutron-tunnel-types vxlan
```

director 会在 heat 中检查 **overcloud** 栈，然后根据环境文件和 heat 模板更新栈中的每个项。它不会重新创建 Overcloud，而是更改已存在的 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 /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/network-environment.yaml -e ~/templates/storage-
environment.yaml -e ~/templates/new-environment.yaml --control-scale 3
--compute-scale 3 --ceph-storage-scale 3 --control-flavor control --
compute-flavor compute --ceph-storage-flavor ceph-storage --ntp-server
pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

这包括从环境文件中读入到栈中的新参数和资源。

重要

一般情况下，不要手工修改 Overcloud 的配置，因为 director 可能会在以后覆盖这些所做的修改。

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。

```
$ nova image-create instance_name image_name
$ glance image-download image_name --file exported_vm.qcow2
```

把导入的镜像上传到 Overcloud，然后启动一个新实例。

```
$ glance image-create --name imported_image --file exported_vm.qcow2 --
disk-format qcow2 --container-format bare
$ nova boot --poll --key-name default --flavor m1.demo --image
imported_image --nic net-id=net_id imported
```

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 Compute 节点上执行维护操作。为了避免下线时间，按照以下方法把 Compute 节点上的虚拟机迁移到 Overcloud 中的另外一个 Compute 节点上。

所有 Compute 节点都需要一个共享的 SSH 密钥，从而使每个主机的 **nova** 用户都可以在迁移的过程中访问这些节点。使用以下步骤在每个 Compute 节点上设置一个 SSH 密钥对。

创建一个 SSH 密钥：

```
$ ssh-keygen -t rsa -f nova_id_rsa
```

把 SSH 密钥复制到每个 Compute 节点上的 **nova** 用户的家目录中。

以 **nova** 用户登录到每个 Compute 节点，运行以下命令来设置密钥：

```
NOVA_SSH=/var/lib/nova/.ssh
mkdir ${NOVA_SSH}

cp nova_id_rsa ${NOVA_SSH}/id_rsa
chmod 600 ${NOVA_SSH}/id_rsa
cp nova_id_rsa.pub ${NOVA_SSH}/id_rsa.pub
cp nova_id_rsa.pub ${NOVA_SSH}/authorized_keys

chown -R nova.nova ${NOVA_SSH}

# enable login for nova user on compute hosts:
usermod -s /bin/bash nova

# add ssh keys of overcloud nodes into known hosts:
ssh-keyscan -t rsa `os-apply-config --key hosts --type raw --key-
default '' | awk '{print $1}'` >> /etc/ssh/ssh_known_hosts
```

在 director 上，source **overcloudrc**，并获得当前的 nova 服务列表：

```
$ source ~/stack/overcloudrc
$ nova service-list
```

在要迁移的节点上禁用 **nova-compute** 服务。

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

这会防止新的虚拟机在它上面运行。

启动从节点上迁移实例的操作：

```
$ nova host-servers-migrate [hostname]
```

使用以下命令可以查看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]
```

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：

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

8.10. 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 包括了一组策略来防止这个问题的出现。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。为了可以删除 Overcloud，把策略恢复为原来的值。

8.11. 删除 OVERCLOUD

在需要的时候，Overcloud 整个可以被删除。

删除一个存在的 Overcloud：

```
$ heat stack-delete overcloud
```

确认删除 Overcloud：

```
$ heat stack-list
```

删除操作会需要几分钟完成。

在删除操作完成后，按照标准的步骤重新安装您的 Overcloud。

第 9 章 扩展 OVERCLOUD

在某些情况下，您可以需要在创建 Overcloud 后添加或删除节点。例如，可能需要为 Overcloud 添加 Compute 节点。在这些情况下，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 中最少有一个 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，需要保证最少有 10 GB 的可用磁盘空间。这些磁盘空间被用来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":"192.0.2.207"
        },
        {
            "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":"192.0.2.208"
        }
    ]
}
```

如需了解与这些参数相关的信息，请参阅 第 5.1 节 “为 Overcloud 注册节点”。

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：

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [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：

```
$ ironic node-update [NODE UUID] add
properties/capabilities='profile:compute,boot_option:local'
```

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：

```
$ glance 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：

```
$ ironic node-update [NODE UUID] add
driver_info/deploy_kernel='09b40e3d-0382-4925-a356-3a4b4f36b514'
$ ironic node-update [NODE UUID] add
driver_info/deploy_ramdisk='765a46af-4417-4592-91e5-a300ead3faf6'
```

扩展 Overcloud 需要重新运行 **openstack overcloud deploy**（使用新的节点数量）。例如，扩展到 5 个 Compute 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

确认包括了初始 Overcloud 创建中的所有环境文件和选项。这包括和非 Compute 节点相同的扩展参数。

9.2. 删除 COMPUTE 节点

在某些情况下，您可能需要从 Overcloud 中删除节点。例如，需要替换一个有问题的 Compute 节点。

重要

在从 Overcloud 中删除一个 Compute 节点前，把这个节点上的负载迁移到其它 Compute 节点上。请参阅 第 8.9 节 “从一个 Overcloud Compute 节点中迁移虚拟机”。

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

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

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，找到 Overcloud 栈的 UUID：

```
$ heat stack-list
```

找到要被删除的节点的 UUID：

```
$ nova 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

如果您在创建 Overcloud 时传递了额外的环境变量，使用 **-e** 或 **--environment-file** 选项再次传递它们来避免对 Overcloud 的不必要的改变。

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
$ nova service-delete [service-id]
$ source ~/stack/stackrc
```

删除节点的 Open vSwitch 代理：

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

现在，可以安全地把节点从 Overcloud 中删除，并用于其它目的。

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 第 8.9 节 “从一个 Overcloud Compute 节点中迁移虚拟机”。
- 从 Overcloud 中删除 Compute 节点。详细信息，请参阅 第 9.2 节 “删除 Compute 节点”。
- 添加新 Compute 节点扩展 Overcloud。详细信息，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 Controller 节点。这个过程包括运行 **openstack overcloud deploy** 命令来

更新需要替换 controller 节点的 Overcloud。请注意，这个过程不是完全自动的，在 Overcloud 栈更新过程中，**openstack overcloud deploy** 命令会在某个阶段报告一个错误，Overcloud 栈更新的过程会随之停止。这时，需要一些人工的干预后，**openstack overcloud deploy** 进程才会继续。

9.4.1. 预检查

在替换一个 Overcloud Controller 节点前，需要对 Red Hat OpenStack Platform 环境进行检查。这可以避免在替换 Controller 节点的过程中出现问题。在 Undercloud 中执行以下的预检查操作来决定替换 Controller 节点是否安全。

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

   ```
   $ source stackrc
   $ heat stack-list --show-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-discoverd.service
   openstack-ironic-discoverd-dnsmasq.service
   $ sudo cp /var/lib/ironic-discoverd/inspector.sqlite
   /home/stack/backup
   $ sudo systemctl start openstack-ironic-api.service openstack-
   ironic-conductor.service openstack-ironic-discoverd.service
   openstack-ironic-discoverd-dnsmasq.service
   ```

3. Undercloud 需要保证最少有 10 GB 的可用磁盘空间。这些磁盘空间被用来保存镜像转换和缓存。

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 集群中的每个节点的以下参数：

   - **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
   $ nova 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 ~]$ nova 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 |
+--------------------------------------+------------------------+
```

在这个示例中，您需要删除 **overcloud-controller-1** 节点，并使用 **overcloud-controller-3** 替换它。首先，把节点设置为维护模式，从而使 director 不再关心故障节点。把 **nova list** 中的实例 ID 与 **ironic node-list** 中的节点 ID 相关联

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

使用 **control** 配置集标记新节点。

```
[stack@director ~]$ ironic node-update 75b25e9a-948d-424a-9b3b-
f0ef70a6eacf add
properties/capabilities='profile:control,boot_option:local'
```

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']}]
```

重要

如果替换索引为 0 的节点，在进行替换前，编辑 heat 模板，修改 bootstrap 节点的索引以及节点验证索引。创建 director 的 Heat 模板集合的一个副本（请参阅 第 6.18 节 “使用定制的核心 Heat 模板”），在 **overcloud.yaml** 文件中运行以下命令：

```
$ sudo sed -i "s/resource\.0/resource.1/g" ~/templates/my-overcloud/overcloud.yaml
```

这会修改以下资源的节点索引：

```
ControllerBootstrapNodeConfig:
  type: OS::TripleO::BootstrapNode::SoftwareConfig
    properties:
      bootstrap_nodeid: {get_attr: [Controller, resource.0.hostname]}
      bootstrap_nodeid_ip: {get_attr: [Controller, resource.0.ip_address]}
```

以及：

```
AllNodesValidationConfig:
  type: OS::TripleO::AllNodes::Validation
  properties:
    PingTestIps:
      list_join:
      - ' '
      - - {get_attr: [Controller, resource.0.external_ip_address]}
        - {get_attr: [Controller, resource.0.internal_api_ip_address]}
        - {get_attr: [Controller, resource.0.storage_ip_address]}
        - {get_attr: [Controller, resource.0.storage_mgmt_ip_address]}
        - {get_attr: [Controller, resource.0.tenant_ip_address]}
```

在指定节点索引后，重新部署 Overcloud，包括 **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

如果您在创建 Overcloud 时使用了额外的环境文件或选项，现在则需要再次使用它们来避免对 Overcloud 的不必要的改变。

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 ~]$ heat stack-list --show-nested
```

9.4.3. 手工干预

在 **ControllerNodesPostDeployment** 阶段，Overcloud 栈更新会在 **ControllerLoadBalancerDeployment_Step1** 的步骤中出现因为 **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，在这个阶段需要一些手工干预。请进行以下步骤：

1. 获得 Controller 节点的 IP 地址列表。例如：

```
[stack@director ~]$ nova 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. 为集群添加新节点：

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

9. 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。

10. 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。

11. 启动新的 Controller 节点：

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

12. 在新节点上启动 keystone 服务。从其它节点上把 **/etc/keystone** 目录复制到 director 主机：

```
[heat-admin@overcloud-controller-0 ~]$ sudo -i
[root@overcloud-controller-0 ~]$ scp -r /etc/keystone
stack@192.168.0.1:~/.
```

登录到新的 Controller 节点。从新节点中删除 **/etc/keystone** 目录，从 director 主机上复制 **keystone** 文件：

```
[heat-admin@overcloud-controller-3 ~]$ sudo -i
[root@overcloud-controller-3 ~]$ rm -rf /etc/keystone
[root@overcloud-controller-3 ~]$ scp -r
stack@192.168.0.1:~/keystone /etc/.
[root@overcloud-controller-3 ~]$ chown -R keystone: /etc/keystone
[root@overcloud-controller-3 ~]$ chown root
/etc/keystone/logging.conf
/etc/keystone/default_catalog.templates
```

编辑 **/etc/keystone/keystone.conf**，把 **admin_bind_host** 和 **public_bind_host** 参数设置为新 Controller 节点的 IP 地址。要找到这些 IP 地址，使用 **ip addr** 命令，在以下网络中查找 IP 地址：

- **admin_bind_host** - Provisioning 网络
- **public_bind_host** - Internal API 网络

注意

如果使用一个定制的 **ServiceNetMap** 参数部署 Overcloud，这些网络可能会不同。

例如，如果 Provisioning 网络使用 **192.168.0.0/24** 子网，Internal API 使用 **172.17.0.0/24** 子网，使用以下命令在这些网络中找出节点的 IP 地址：

```
[root@overcloud-controller-3 ~]$ ip addr | grep
"192\.168\.0\..*/24"
[root@overcloud-controller-3 ~]$ ip addr | grep
"172\.17\.0\..*/24"
```

13. 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
```

14. 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 overcloud-controller-3
```

15. 等待 Keystone 服务在所有节点上都已启动。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep
keystone -A1
Clone Set: openstack-keystone-clone [openstack-keystone]
Started: [ 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
openstack-keystone-clone 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
```

手工配置已完成。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

如果您在创建 Overcloud 时使用了额外的环境文件或选项，现在则需要再次使用它们来避免对 Overcloud 的不必要的改变。但是请注意，**remove-controller.yaml** 文件不再需要。

9.4.4. 完成配置 Overcloud 服务

在 Overcloud 栈更新完成后，还需要一些最终的配置。登录到一个 Controller 节点，使用 Pacemaker 刷新所有停止的服务：

```
[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

Source **overcloudrc** 文件从而可以和 Overcloud 进行交互。检查您的路由器来确定 Overcloud 环境中的 L3 代理正确地运行了路由器。在这个示例中，使用名为 **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

被删除节点的 Compute 服务仍然存在于 Overcloud 中，它需要被删除。Source **overcloudrc** 文件从而可以和 Overcloud 进行交互。检查被删除节点的 compute 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 Storage 节点的方法。相关信息可以通过 Red Hat Ceph Storage for the Overcloud 获得。

9.6. 替换 OBJECT 存储节点

要在 Object 存储集群中替换节点，您需要：

- 更新带有新 Object 存储节点的 Overcloud，防止 Director 创建 ring 文件。
- 使用 **swift-ring-builder** 对节点手工添加或删除节点。

以下介绍了在保证集群正常的情况下，如何替换节点的方法。在这个示例中，有一个带有 2 个节点的 Object 存储集群。我们需要添加一个额外的节点，然后替换有问题的节点。

首先，创建一个包括以下内容的环境文件（名为 **~/templates/swift-ring-prevent.yaml**）：

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 3
```

SwiftRingBuild 和 **RingBuild** 参数分别定义了 Overcloud 是否自动为 Object 存储和 Controller 节点构建 ring 文件。**ObjectStorageCount** 定义了在环境中有多少个 Object 存储节点。在这里，我们把节点数从 2 个扩展为 3 个。

在 **openstack overcloud deploy** 命令中包括 **swift-ring-prevent.yaml** 文件，以及 Overcloud 中的其它环境文件：

```
$ openstack overcloud deploy --templates [ENVIRONMENT_FILES] -e swift-ring-prevent.yaml
```

注意

把这个文件添加到环境文件的最后，从而使它的参数可以覆盖前面的环境文件参数。

在部署完成后，Overcloud 就包括了一个额外的 Object 存储节点。但是，这个节点的存储目录还没有创建，用于节点对象存储的 ring 文件也没有构建。这意味着，您需要手工创建存储目录，并手工构建 ring 文件。

注意

使用以下步骤在 Controller 节点上构建 ring 文件。

登录到新节点并创建存储目录：

```
$ sudo mkdir -p /srv/node/d1
$ sudo chown -R swift:swift /srv/node/d1
```

注意

您还可以在这个目录中挂载一个外部存储设备。

把存在的 ring 文件复制到节点。以 **heat-admin** 用户身份登录到一个 Controller 节点，然后切换到超级用户（superuser）。例如，对于一个 IP 地址为 192.168.201.24 的 Controller 节点：

```
$ ssh heat-admin@192.168.201.24
$ sudo -i
```

把 **/etc/swift/*.builder** 文件从 Controller 节点复制到新的 Object 存储节点的 **/etc/swift/** 目录中。如果需要，把文件放到 director 主机上：

```
[root@overcloud-controller-0 ~]# scp /etc/swift/*.builder stack@192.1.2.1:~/.
```

然后，把文件放到新节点上：

```
[stack@director ~]$ scp ~/*.builder heat-admin@192.1.2.24:~/.
```

以 **heat-admin** 用户的身份登录到新的 Object 存储节点上，然后切换到超级用户。例如，对于 IP 地址为 192.168.201.29 的 Object 存储节点：

```
$ ssh heat-admin@192.168.201.29
$ sudo -i
```

把文件复制到 **/etc/swift** 目录：

```
# cp /home/heat-admin/*.builder /etc/swift/.
```

把新的 Object 存储节点添加到帐号、容器和对象 ring 中。为新节点运行以下命令：

```
# swift-ring-builder /etc/swift/account.builder add zX-IP:6002/d1
weight
# swift-ring-builder /etc/swift/container.builder add zX-IP:6001/d1
weight
# swift-ring-builder /etc/swift/object.builder add zX-IP:6000/d1 weight
```

在这些命令中替换以下值：

zX

使用代表特定区的一个整数替换 X（例如，Zone 1 的值为 1）。

IP

帐号、容器和对象服务要监听的 IP 地址。这应该和每个存储节点的 IP 地址相匹配，特别是和 **/etc/swift/object-server.conf**、**/etc/swift/account-server.conf** 和 **/etc/swift/container-server.conf** 中的 **DEFAULT** 部分中的 **bind_ip** 的值相匹配。

weight

表示一个设备和其它设备相比较的相对权重。它的值通常是 **100**。

注意

针对 rings 文件运行 **swift-ring-builder** 命令来检查当前节点存在的值：

```
# swift-ring-builder /etc/swift/account.builder
```

删除您需要从账户、容器和对象 ring 中替换的节点。为每个节点运行以下命令：

```
# swift-ring-builder /etc/swift/account.builder remove IP
# swift-ring-builder /etc/swift/container.builder remove IP
# swift-ring-builder /etc/swift/object.builder remove IP
```

使用节点的 IP 地址替换 **IP**。

在所有节点间重新分布分区：

```
# swift-ring-builder /etc/swift/account.builder rebalance
# swift-ring-builder /etc/swift/container.builder rebalance
# swift-ring-builder /etc/swift/object.builder rebalance
```

把 **/etc/swift/** 的所有者设置改为 **root** 用户和 **swift** 组：

```
# chown -R root:swift /etc/swift
```

重启 **openstack-swift-proxy** 服务：

```
# systemctl restart openstack-swift-proxy.service
```

到此为止，ring 文件（*.ring.gz 和 *.builder）应该已在新节点上被更新：

```
/etc/swift/account.builder
/etc/swift/account.ring.gz
/etc/swift/container.builder
/etc/swift/container.ring.gz
/etc/swift/object.builder
/etc/swift/object.ring.gz
```

把这些文件复制到 Controller 节点和存在的 Object Storage 节点（除去要删除的节点）的 **/etc/swift/** 目录中。如果需要，把文件放置到 director 主机：

```
[root@overcloud-objectstorage-2 swift]# scp *.builder
stack@192.1.2.1:~/
[root@overcloud-objectstorage-2 swift]# scp *.ring.gz
stack@192.1.2.1:~/
```

把这个文件复制到每个节点的 **/etc/swift/** 中。

在每个节点中，把 **/etc/swift/** 的所有者设置改为 **root** 用户和 **swift** 组：

```
# chown -R root:swift /etc/swift
```

新节点被添加并作为 ring 的一部分。在把旧节点从 ring 中删除前，请确认新节点已完成了一个完整的数据复制过程。

为了从 ring 中删除旧节点，减少 **ObjectStorageCount** 的值。在这个示例中，我们把它从 3 减为 2：

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 2
```

创建一个环境文件（**remove-object-node.yaml**）来指定并删除旧的 Object 存储节点。在这个示例中，我们删除 **overcloud-objectstorage-1**：

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

在部署命令中包括这两个环境文件：

```
$ openstack overcloud deploy --templates -e swift-ring-prevent.yaml -e
remove-object-node.yaml ...
```

director 从 Overcloud 中删除 Object 存储节点，并更新 Overcloud 中的其它节点来使删除生效。

第 10 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。

10.1. 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]
```

10.2. 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

在重新引导一个节点前，**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
```

访问内省 Ramdisk

内省 ramdisk 使用一个动态的登录元素。这意味着，在进行内省故障排除时您可以提供一个临时密码或一个 SSH 密钥来访问节点。使用以下方法来设置对 ramdisk 的访问：

1. 为 **openssl passwd -1** 命令提供一个临时密码来产生一个 MD5 哈希。例如：

   ```
   $ openssl passwd -1 mytestpassword
   $1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
   ```

2. 编辑 **/httpboot/inspector.ipxe** 文件，找到以 **kernel** 开始的行并添加 **rootpwd** 参数和 MD5 哈希。例如：

   ```
   kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-
   callback-url=http://192.168.0.1:5050/v1/continue ipa-inspection-
   collectors=default,extra-hardware,logs
   systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1
   ipa-inspection-benchmarks=cpu,mem,disk
   rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/"
   ```

   或者，使用 **sshkey** 参数和您的公共 SSH 密钥。

   注意

   rootpwd 和 **sshkey** 参数都需要使用引号。

3. 开始内省过程，通过 **arp** 命令或 DHCP 日志找到 IP 地址：

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

4. 使用临时密码或 SSH 密钥，以 root 用户身份进行 SSH 连接。

   ```
   $ ssh root@192.0.2.105
   ```

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*
```

10.3. 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 的实施在以上 3 个层面中出现问题，使用 OpenStack 客户端和服务日志文件来诊断相关的错误。

10.3.1. 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。

10.3.2. 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。

10.3.3. 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@192.0.2.14
```

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 的整个节点实施过程失败的问题。在这种情况下，一个 Overcloud 角色类型的 **OS::Heat::ResourceGroup** 会出错。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 10.5 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。这些信息可被用于进行故障诊断和故障排除。技术支持和开发人员也可以通过 SOS 获得有用的信息。SOS 在 Undercloud 和 Overcloud 中都有用。运行以下命令安装 **sos** 软件包：

```
$ sudo yum install sos
```

产生一个报告

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

Controller 节点部署后的过程包括 6 个主要的部署步骤。它包括：

表 10.1. Controller 节点配置步骤

步骤	描述
ControllerLoadBalancerDeployment_Step1	初始负载均衡软件配置，包括 Pacemaker、RabbitMQ、Memcached、Redis 和 Galera。
ControllerServicesBaseDeployment_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。
ControllerRingbuilderDeployment_Step3	OpenStack Object Storage (**swift**) 的初始 ring 构建。
ControllerOvercloudServicesDeployment_Step4	配置所有的 OpenStack Platform 服务（**nova**、**neutron**、**cinder**、**sahara**、**ceilometer**、**heat**、**horizon**、**aodh**、**gnocchi**）。

ControllerOvercloudServicesDeployment_Step5	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。
ControllerOvercloudServicesDeployment_Step6	Overcloud 的最终配置。

10.4. 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** 命令扫描活跃的 IP 地址范围。这个示例会扫描 192.0.2.0/24 这个范围，使用 Provisioning 网络的 IP 子网值（CIDR 格式）替换它：

```
# nmap -sn 192.0.2.0/24
```

检查 **nmap** 扫描的结果输出：

例如，您可以看到 Undercloud 的 IP 地址，以及在这个子网中的任何其它主机。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 192.0.2.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.0.2.1
Host is up (0.00057s latency).
Nmap scan report for 192.0.2.2
Host is up (0.00048s latency).
Nmap scan report for 192.0.2.3
Host is up (0.00045s latency).
Nmap scan report for 192.0.2.5
Host is up (0.00040s latency).
Nmap scan report for 192.0.2.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

10.5. 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
   ```

10.6. 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。本节介绍了对这些可能出现的问题进行故障排除的方法。

10.6.1. 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 10.3.3 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。

10.6.2. Controller 服务失败

Overcloud Controller 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 Controller 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节点进行故障诊断，以便 Overcloud 可以以最大能力运行。

在高可用性集群中，Controller 节点使用 Pacemaker 管理资源和服务。Pacemaker Configuration System（**pcs**）是一个用来管理 Pacemaker 集群的工具程序。在集群的 Controller 节点上运行这个命令来执行配置和监控操作。在一个高可用性集群中，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。

10.6.3. 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 第 8.9 节 “从一个 Overcloud Compute 节点中迁移虚拟机”。

10.6.4. Ceph Storage 服务故障

如果 Red Hat Ceph Storage 集群出现故障，参阅 Red Hat Ceph Storage Configuration Guide 中的 Part 10. Logging and Debugging。

10.7. 对 UNDERCLOUD 进行性能微调

本节中提供的建议针对于提高 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和其它内部数据结构的内存池的大小（以字节为单位）。它的默认值是 8M，对于 Undercloud，理想的值是 20M。

innodb_buffer_pool_size

缓冲池（数据库用来缓存表和索引数据的内存区）的大小（以字节为单位）。它的默认值是 128M，对于 Undercloud，理想的值是 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。为了手工设置 worker 的数量，编辑 **/etc/heat/heat.conf** 文件，把 **num_engine_workers** 参数的值设置为所需的 worker 的数量（理想值是 4），然后重启 heat 引擎：

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

10.8. UNDERCLOUD 和 OVERCLOUD 的重要日志

在出现问题时，使用以下日志查找 Undercloud 和 Overcloud 的信息。

表 10.2. Undercloud 和 Overcloud 的重要日志

信息	Undercloud 或 Overcloud	日志位置
一般的 director 服务	Undercloud	**/var/log/nova/*** **/var/log/heat/*** **/var/log/ironic/***

内省（Introspection）	Undercloud	**/var/log/ironic/*** **/var/log/ironic-inspector/***
Provisioning	Undercloud	**/var/log/ironic/***
Cloud-Init 日志	Overcloud	**/var/log/cloud-init.log**
Overcloud 配置（最后一次 Puppet 运行的概述）	Overcloud	**/var/lib/puppet/state/last_run_summary.yaml**
Overcloud 配置（最后一次 Puppet 运行的报告）	Overcloud	**/var/lib/puppet/state/last_run_report.yaml**
Overcloud 配置（所有 Puppet 报告）	Overcloud	**/var/lib/puppet/reports/overcloud-*/***
一般的 Overcloud 服务	Overcloud	**/var/log/ceilometer/*** **/var/log/ceph/*** **/var/log/cinder/*** **/var/log/glance/*** **/var/log/heat/*** **/var/log/horizon/*** **/var/log/httpd/*** **/var/log/keystone/*** **/var/log/libvirt/*** **/var/log/neutron/*** **/var/log/nova/*** **/var/log/openvswitch/*** **/var/log/rabbitmq/*** **/var/log/redis/*** **/var/log/swift/***

高可用性日志	Overcloud	**`/var/log/pacemaker.log`**

附录 A. SSL/TLS 证书配置

作为 第 4.6 节 “配置 director” 或 第 6.10 节 “在 Overcloud 中启用 SSL/TLS” 所介绍的操作的一个可选部分，您可以设置在 Undercloud 或 Overcloud 上使用 SSL/TLS 进行通讯。但是，如果使用一个自签发的证书，则需要一些特定的配置来使用这个证书。

A.1. 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.2. 把证书认证机构添加到客户端中

任何需要使用 SSL/TLS 与 Undercloud 或 Overcloud 进行通讯的客户端，把这个证书认证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来把这个证书认证机构文件加入到信任的认证机构中：

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

A.3. 创建一个 SSL/TLS 密钥

运行以下命令产生 SSL/TLS 密钥（**server.key.pem**）。我们可以在不同地方使用它来产生自己的 Undercloud 或 Overcloud 证书：

```
$ openssl genrsa -out server.key.pem 2048
```

A.4. 创建一个 SSL/TLS 证书签发请求

下一步会为 Undercloud 或 Overcloud 创建一个证书签发请求。

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 = 192.168.0.1
DNS.2 = instack.localdomain
DNS.3 = vip.localdomain
```

把 **commonName_default** 设置为 Public API 的 IP 地址，或 FQDN：

- 对于 Undercloud，使用 **undercloud.conf** 中的 **undercloud_public_vip** 参数。如果这个 IP 地址使用了一个 FQDN，则使用这个 FQDN。
- 对于 Overcloud，使用 Public API 的 IP 地址（您的网络分离环境文件中的 **ExternalAllocationPools** 参数的第一个地址）。如果这个 IP 地址使用了一个 FQDN，则使用这个 FQDN。

在 **alt_names** 部分包括相同的 Public API IP 地址作为 IP 项。如果还使用 DNS，在相同的部分包括服务器的主机名作为 DNS 项。如需了解更多与 **openssl.cnf** 相关的信息，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.3 节 “创建一个 SSL/TLS 密钥” 中创建的 SSL/TLS 密钥。

重要

openssl req 命令会要求输入证书的一些详细信息，包括 Common Name。把 Common Name 设置为 Undercloud 或 Overcloud 的（取决于您要创建哪个证书） Public API 的 IP 地址。**openssl.cnf** 文件会使用这个 IP 地址作为默认值。

使用 **server.csr.pem** 文件创建 SSL/TLS 证书。

A.5. 创建 SSL/TLS 证书

以下命令会为 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
```

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.4 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。
- 在 第 A.1 节 “创建一个证书认证机构（CA）” 中创建的证书认证机构，它可以签发证书。把它作为 `-cert ` 选项。

这会产生一个名为 **server.crt.pem** 的证书。使用这个证书以及在 第 A.3 节 “创建一个 SSL/TLS 密钥” 中产生的 SSL/TLS 密钥来在 Undercloud 或 Overcloud 中启用 SSL/TLS。

A.6. 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.1 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的列表中，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” 中的介绍继续安装 Undercloud。

A.7. 在 OVERCLOUD 中使用证书

在 第 6.10 节 “在 Overcloud 中启用 SSL/TLS” 中介绍的 **enable-tls.yaml** 环境文件中包括您的证书文件的内容。Overcloud 的部署操作会从 **enable-tls.yaml** 中获得参数，并自动把它们集成到 Overcloud 中的每个节点上。

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 第 5.1 节 “为 Overcloud 注册节点” 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。

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 私人密钥的内容。私人密钥的内容必须在一行中（其中的换行需要以 **\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。当节点的状态由 **deploying** 变为 **deploy wait-callback** 后，手工启动这个节点。
- 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来决定部署过程是否已完成。当部署完成后，节点状态会变为 **active**。然后，手工重启所有 Overcloud 节点。

附录 C. 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：

- 策略可以找出并分离 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。

C.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 C.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"
  }
]
```

C.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。

C.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
```

C.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 D. 基础参数

以下是配置 Overcloud 的基础参数列表。这些参数在 director 的核心 Heat 模板集合的 **`overcloud.yaml`** 文件中的 **`parameters`** 项中定义。

AdminPassword

> **类型：** 字符串
>
> OpenStack Identity 管理帐号的密码。

AdminToken

> **类型：** 字符串
>
> OpenStack Identity 用户身份认证的 secret。

AodhPassword

> **类型：** 字符串
>
> OpenStack Telemetry Alarming 服务的密码。

BlockStorageCount

> **I类型：**数字
>
> Overcloud 中的 Block Storage 节点数。

BlockStorageExtraConfig

> **类型：** json
>
> 注入到集群中的 Block Storage 配置。

BlockStorageHostnameFormat

> **类型：** 字符串
>
> Block Storage 节点主机名的格式。

BlockStorageImage

> **类型：** 字符串
>
> 用于设置 Block Storag 节点的镜像。

BlockStorageRemovalPolicies

> **类型：** json
>
> 在进行需要删除特定资源的更新时，要从 **`BlockStorageResourceGroup`** 中删除的资源列表。

BlockStorageSchedulerHints

> **类型：** json
>
> 传递给 OpenStack Compute 的可选 scheduler hint。

CeilometerBackend

类型： 字符串

OpenStack Telemetry 后端类型。选择 **mongodb** 或 **mysql**。

CeilometerComputeAgent

类型： 字符串

指示 Compute 代理是否存在，并期望相应地配置 **nova-compute** 。

CeilometerMeterDispatcher

类型： 字符串

OpenStack Telemetry（**ceilometer**）服务包括了一个时间序列数据存储的新组件（**gnocchi**）。现在，可以在 Red Hat OpenStack Platform 中把默认的 Ceilometer dispatcher 切换为使用新组件，而不是使用标准的数据库。这个设置是通过 **CeilometerMeterDispatcher** 实现的，您可以把它设置为以下之一：

- **database** - 为 Ceilometer dispatcher 使用标准数据库。这是默认选项。
- **gnocchi** - 为 Ceilometer dispatcher 使用新的时间序列数据库。

CeilometerMeteringSecret

类型： 字符串

OpenStack Telemetry 服务间共享的秘密。

CeilometerPassword

类型： 字符串

OpenStack Telemetry 服务账户的密码。

CephAdminKey

类型： 字符串

Ceph admin 客户的密钥。可以使用 **ceph-authtool --gen-print-key** 创建。

CephClientKey

类型： 字符串

Ceph 客户密钥。可以使用 **ceph-authtool --gen-print-key** 创建。当前只用于外部的 Ceph 部署来创建 OpenStack 的用户密钥环（keyring）。

CephClusterFSID

类型： 字符串

Ceph 集群的 FSID。需要是一个 UUID。

CephExternalMonHost

类型： 字符串

外部管理的 Ceph Monitors 主机 IP 列表。只用于外部 Ceph 部署。

CephMonKey

类型： 字符串

Ceph Monitors 密钥。可以使用 **`ceph-authtool --gen-print-key`** 创建。

CephStorageCount

l类型：数字

Overcloud 中的 Ceph Storage 节点数量。

CephStorageExtraConfig

类型： json

注入到集群中的 Ceph Storage 配置。

CephStorageHostnameFormat

类型： 字符串

Ceph Storage 节点主机名的格式。

CephStorageImage

类型： 字符串

用于设置 Ceph Storag 节点的镜像。

CephStorageRemovalPolicies

类型： json

在进行需要删除特定资源的更新时，要从 **`CephStorageResourceGroup`** 中删除的资源列表。

CephStorageSchedulerHints

类型： json

传递给 OpenStack Compute 的可选 scheduler hint。

CinderEnableIscsiBackend

类型： 布尔值

是否为 Block Storage 启用 iSCSI 后端。

CinderEnableNfsBackend

类型： 布尔值

是否为 Block Storage 启用 NFS 后端。

CinderEnableRbdBackend

类型： 布尔值

是否为 Block Storage 启用 Ceph Storage 后端。

CinderISCSIHelper

类型： 字符串

与 Block Storage 一起使用的 iSCSI helper。

CinderLVMLoopDeviceSize

l类型：数字

Block Storage LVM 驱动使用的环回文件的大小。

CinderNfsMountOptions

类型： 字符串

Block Storage NFS 后端使用的 NFS 挂载选项。当 **`CinderEnableNfsBackend`** 为 true 时有效。

CinderNfsServers

类型：逗号分隔的列表

Block Storage NFS 后端使用的 NFS 服务器。当 **`CinderEnableNfsBackend`** 为 true 时有效。

CinderPassword

类型： 字符串

Block Storage 服务帐号的密码。

CloudDomain

类型： 字符串

主机的 DNS 域。这需要和 Undercloud 网络中的 **`dhcp_domain`** 配置相匹配。它的默认值是 **`localdomain`**。

CloudName

类型： 字符串

云的 DNS 名。例如：**`ci-overcloud.tripleo.org`**。

ComputeCount

l类型：数字

Overcloud 中的 Compute 节点数量。

ComputeHostnameFormat

类型： 字符串

Compute 节点主机名的格式。

ComputeRemovalPolicies

类型： json

在进行需要删除特定资源的更新时，要从 **`ComputeResourceGroup`** 中删除的资源列表。

ControlFixedIPs

类型： json

Controller 节点的固定 IP 地址。

ControlVirtualInterface

类型： 字符串

虚拟 IP 分配的接口。

ControllerCount

l类型：数字

Overcloud 中的 Controller 节点数量。

ControllerEnableCephStorage

类型： 布尔值

是否在 Controller 上部署 Ceph Storage (OSD)。

ControllerEnableSwiftStorage

类型： 布尔值

是否在 Controller 上启用 Object Storage。

controllerExtraConfig

类型： json

注入到集群中的 Controller 配置。

ControllerHostnameFormat

类型： 字符串

Controller 节点主机名的格式。

controllerImage

类型： 字符串

用于设置 Block Storag 节点的镜像。

ControllerRemovalPolicies

类型： json

在进行需要删除特定资源的更新时，要从 **`ControllerResourceGroup`** 中删除的资源列表。

ControllerSchedulerHints

类型： json

传递给 OpenStack Compute 的可选 scheduler hint。

CorosyncIPv6

类型： 布尔值

在 Corosync 中启用 IPv6。

Debug

类型： 字符串

把它设为 **`true`** 以在所有服务中启用故障排除功能。

DeployIdentifier

类型： 字符串

设置为一个唯一的值来重新运行对 Overcloud 的 **`stack-update`** 进行配置的操作。

EnableFencing

类型： 布尔值

指定是否在 Pacemaker 中启用隔离功能。

EnableGalera

类型： 布尔值

指定是否使用 Galera，而不是使用一般的 MariaDB。

ExtraConfig

类型： json

注入到集群中的额外配置。任何针对于角色的 **`ExtraConfig`**（如**`controllerExtraConfig`**）都会优先于标准的 **`ExtraConfig`**。

FencingConfig

类型： json

Pacemaker 的隔离设置。JSON 应该有以下结构：

```
{
  "devices": [
    {
      "agent": "AGENT_NAME",
      "host_mac": "HOST_MAC_ADDRESS",
      "params": {"PARAM_NAME": "PARAM_VALUE"}
    }
  ]
}
```

例如：

```
{
  "devices": [
    {
      "agent": "fence_xvm",
      "host_mac": "52:54:00:aa:bb:cc",
      "params": {
        "multicast_address": "225.0.0.12",
        "port": "baremetal_0",
        "manage_fw": true,
        "manage_key_file": true,
        "key_file": "/etc/fence_xvm.key",
        "key_file_password": "abcdef"
```

```
            }
        }
    ]
}
```

GlanceBackend

类型： 字符串

OpenStack Image 后端使用的短名。它需要是 **`swift`**、**`rbd`** 或 **`file`** 之一。

GlanceLogFile

类型： 字符串

用于记录 OpenStack Image 日志信息的文件的路径。

GlanceNotifierStrategy

类型： 字符串

OpenStack Image 通知队列使用的策略。默认为 **noop**。

GlancePassword

类型： 字符串

OpenStack Image 所使用的 OpenStack Image 服务帐号的密码。

GnocchiBackend

类型： 字符串

Gnocchi 后端使用的短名。它需要是 **`swift`**、**`rbd`** 或 **`file`** 之一。

GnocchiIndexerBackend

类型： 字符串

Gnocchi indexer 后端使用的短名。

GnocchiPassword

类型： 字符串

Gnocchi 服务账户的密码。

HAProxySyslogAddress

类型： 字符串

HAProxy 把它的日志发送到的 Syslog 地址。

HeatPassword

类型： 字符串

Heat 服务账户的密码。

HeatStackDomainAdminPassword

类型： 字符串

heat_stack_domain_admin 用户的密码。

HorizonAllowedHosts

类型：逗号分隔的列表

允许连接到 OpenStack Dashboard 的 IP/主机名列表。

HypervisorNeutronPhysicalBridge

类型： 字符串

在每个 hypervisor 上创建的 OVS 网桥。它的默认值是 **br-ex**，这和 control plane 节点相同（因为我们对 Open vSwitch 代理有一个统一配置）。通常情况下，不需要修改这个值。

HypervisorNeutronPublicInterface

类型： 字符串

指定把哪个接口添加到 **HypervisorNeutronPhysicalBridge**。

ImageUpdatePolicy

类型： 字符串

指定在构建实例时使用什么策略。**REBUILD** 为重新构建，**REBUILD_PRESERVE_EPHEMERAL** 则保留 **/mnt**。

InstanceNameTemplate

类型： 字符串

用来产生实例名的模板字符串。

InternalApiVirtualFixedIPs

类型： json

控制对 InternalApiVirtualInterface 端口的 IP 分配。例如：

```
[{'ip_address':'1.2.3.4'}]
```

KeyName

类型： 字符串

一个存在的 OpenStack Compute 密钥对，使用它来允许对实例进行 SSH 访问。

KeystoneCACertificate

类型： 字符串

OpenStack Identity 自签发的证书认证机构证书。

KeystoneNotificationDriver

类型：逗号分隔的列表

OpenStack Identity 使用的、以逗号分隔的通知驱动。

KeystoneNotificationFormat

类型： 字符串

OpenStack Identity 通知的格式。

KeystoneSSLCertificate

类型： 字符串

用于验证令牌有效性的 OpenStack Identity 证书。

KeystoneSSLCertificateKey

类型： 字符串

签发令牌的 OpenStack Identity 密钥。

KeystoneSigningCertificate

类型： 字符串

用于验证令牌有效性的 OpenStack Identity 证书。

KeystoneSigningKey

类型： 字符串

签发令牌的 OpenStack Identity 密钥。

ManageFirewall

类型： 布尔值

指定是否管理防火墙规则。

MemcachedIPv6

类型： 布尔值

在 Memcached 中启用 IPv6。

MongoDbIPv6

类型： 布尔值

如果 MongoDB VIP 是 IPv6，启用 IPv6。

MongoDbNoJournal

类型： 布尔值

指定是否禁用 MongoDb 日志功能（journaling）。

MysqlInnodbBufferPoolSize

l类型：数字

缓冲池的大小（以 MB 为单位）。如果把它设为 0，则被认为是“没有任何值”，这会使用低级别默认值。

MysqlMaxConnections

l类型：数字

配置 MySQL **`max_connections`** 的设置。

NeutronAgentExtensions

类型：逗号分隔的列表

为 OpenStack Networking 代理启用的、以逗号分隔的扩展列表。

NeutronAgentMode

类型： 字符串

在 Controller 主机上的 **`neutron-l3-agent`** 代理模式。

NeutronAllowL3AgentFailover

类型： 字符串

允许自动进行 L3 代理故障转移。

NeutronBridgeMappings

类型：逗号分隔的列表

使用的 OVS 逻辑到物理网桥的映射。默认为把主机上的外部网桥（**`br-ex`**）映射到一个物理名（*datacentre*）， 它可以用来创建供应商网络（我们使用它作为默认的浮动网络）。如需改变这个设置，使用不同的安装后网络脚本，或确保保留 *datacentre* 作为映射网络名。

NeutronComputeAgentMode

类型： 字符串

在 Compute 节点上的 **`neutron-l3-agent`** 代理模式。

NeutronControlPlaneID

类型： 字符串

`ctlplane` 网络的 Neutron ID 或名称。

NeutronCorePlugin

类型： 字符串

OpenStack Networking 的核心插件。这个值应该是从 neutron.core_plugins 命名空间加载的进入点。

NeutronDVR

类型： 字符串

是否配置 OpenStack Networking Distributed Virtual Router

NeutronDhcpAgentsPerNetwork

I类型：数字

每个网络可以调度的 OpenStack Networking dhcp 代理的数量

NeutronDnsmasqOptions

类型： 字符串

`neutron-dhcp-agent` 的 Dnsmasq 选项。默认值会强制把 MTU 设置为 **`NeutronTenantMtu`** 的值。

NeutronEnableIsolatedMetadata

类型： 字符串

如果为 true，DHCP 为 VM 提供元数据路由。

NeutronEnableL2Pop

类型： 字符串

在 OpenStack Networking 代理中启用或禁用 L2 population 功能。

NeutronEnableTunnelling

类型： 字符串

指定是否在 OpenStack Networking 中启用隧道（ tunneling）。

NeutronExternalNetworkBridge

类型： 字符串

用于外部网络流量的网桥名。

NeutronFlatNetworks

类型：逗号分隔的列表

定义在 OpenStack Networking 插件中配置的平面网络列表。默认是 *datacentre*，允许外部网络创建。

NeutronL3HA

类型： 字符串

如果需要为 OpenStack Networking 禁用第 3 层高可用性功能（L3HA），在环境文件中把它设置为 **`false`**。

NeutronMechanismDrivers

类型：逗号分隔的列表

OpenStack Networking 租户网络的 mechanism 驱动。

NeutronMetadataProxySharedSecret

类型： 字符串

共享秘密以防止欺诈

NeutronNetworkType

类型：逗号分隔的列表

OpenStack Networking 的租户网络类型。

NeutronNetworkVLANRanges

类型：逗号分隔的列表

支持的 OpenStack Networking ML2 和 OpenVSwitch vlan 的映射范围。请参阅 OpenStack Networking 的文档来获得允许的值。它的默认设置是在 *datacentre* 网络网络中允许任何 VLAN（请参阅 **`NeutronBridgeMappings`**）。

NeutronPassword

类型： 字符串

OpenStack Networking 代理使用的 OpenStack Networking 服务帐号密码。

NeutronPluginExtensions

类型：逗号分隔的列表

为 OpenStack Networking 插件启用的、以逗号分隔的扩展列表。

NeutronPublicInterface

类型： 字符串

指定哪个接口为网络节点桥接到 **`br-ex`**。

NeutronPublicInterfaceDefaultRoute

类型： 字符串

`NeutronPublicInterface` 的自定义默认路由。

NeutronPublicInterfaceIP

类型： 字符串

分配给 **`NeutronPublicInterface`** 的自定义 IP 地址。

NeutronPublicInterfaceRawDevice

类型： 字符串

如果设置，公共接口将会是一个 VLAN，它使用这个设备作为原设备。

NeutronPublicInterfaceTag

类型： 字符串

创建一个公共 VLAN 的 VLAN tag。这个 tag 会被用来为每个 control plane 节点创建到外围网桥的访问端口，并为这个端口分配从 OpenStack Networking 公共网络返回的 IP 地址。

NeutronServicePlugins

类型：逗号分隔的列表

以逗号分隔的、从 neutron.service_plugins 命名空间加载的服务插件进入点。

NeutronTenantMtu

类型： 字符串

租户网络的默认 MTU。对于 VXLAN/GRE tunneling，它的值应最少比物理网络的 MTU 小 50 个字节。这个值会被用于在需要以太网设备中设置 MTU。这个值将会用于构建 **`NeutronDnsmasqOptions`**，从而决定通过 DHCP 分配给 VM 的 MTU。

NeutronTunnelIdRanges

类型：逗号分隔的列表

以逗号分隔的 **`<tun_min>:<tun_max>`** 元组列表，它代表了可用于租户网络分配的 GRE 隧道 ID 的范围。

NeutronTunnelTypes

类型：逗号分隔的列表

OpenStack Networking 租户网络的隧道（tunnel）类型。

NeutronTypeDrivers

类型：逗号分隔的列表

以逗号分隔的、加载的网络类型驱动进入点列表。

NeutronVniRanges

类型：逗号分隔的列表

以逗号分隔的 **`<vni_min>:<vni_max>`** 元组列表，它代表了可用于租户网络分配的 VXLAN VNI ID 的范围。

NovaComputeDriver

类型： 字符串

用来管理实例的 OpenStack Compute 驱动。默认为 **`libvirt.LibvirtDriver`** 驱动。

NovaComputeExtraConfig

类型： json

注入到集群中的 Compute 节点配置。

NovaComputeLibvirtType

类型： 字符串

定义使用的 Libvirt 类型。默认为 **`kvm`**。

NovaComputeLibvirtVifDriver

类型： 字符串

网络的 Libvirt VIF 驱动配置。

NovaComputeSchedulerHints

类型： json

传递给 OpenStack Compute 的可选 scheduler hint。

NovaEnableRbdBackend

类型： 布尔值

是否为 Nova 启用 Ceph 后端。

NovaIPv6

类型： 布尔值

在 Nova 中启用 IPv6。

NovaImage

类型： 字符串

配置 Compute 节点所使用的镜像。

NovaOVSBridge

类型： 字符串

Open vSwitch 使用的集成网桥的名称。

NovaPassword

类型： 字符串

OpenStack Compute 服务账户的密码。

NovaSecurityGroupAPI

类型： 字符串

安全 API 类的完全类名称。

NtpServer

类型：逗号分隔的列表

用逗号隔开的 NTP 服务器列表

ObjectStorageCount

I类型：数字

Overcloud 中的 Object Storage 节点数量。

ObjectStorageExtraConfig

类型： json

注入到集群中的 ObjectStorage 配置。

ObjectStorageHostnameFormat

类型： 字符串

Object Storage 节点主机名的格式。

ObjectStorageRemovalPolicies

类型： json

在进行需要删除特定资源的更新时，要从 **`ObjectStorageResourceGroup`** 中删除的资源列表。

ObjectStorageSchedulerHints

类型： json

传递给 OpenStack Compute 的可选 scheduler hint。

OvercloudBlockStorageFlavor

类型： 字符串

在部署时请求的 Block Storage 节点的 flavor。

OvercloudCephStorageFlavor

类型： 字符串

在部署时请求的 Ceph Storage 节点的 flavor。

OvercloudComputeFlavor

类型： 字符串

在部署过程中 Compute 节点请求的 flavor。

OvercloudControlFlavor

类型： 字符串

在部署过程中 Controller 节点请求的 flavor。

OvercloudSwiftStorageFlavor

类型： 字符串

在部署时请求的 Object Storage 节点的 flavor。

PublicVirtualFixedIPs

类型： json

控制对 **`PublicVirtualInterface`** 端口的 IP 分配。例如：

```
[{'ip_address':'1.2.3.4'}]
```

PublicVirtualInterface

类型： 字符串

指定面向公众的虚拟 IP 要被分配的接口。如果使用 VLAN，它应该是 **`int_public`**。

PurgeFirewallRules

类型： 布尔值

指定在设置新的防火墙规则前，是否要删除旧的规则。

RabbitClientPort

l类型：数字

设置 RabbitMQ 订阅端口。

RabbitClientUseSSL

类型： 字符串

RabbitMQ 客户端订阅参数用来指定到 RabbitMQ 主机的 SSL 连接。

RabbitCookieSalt

类型： 字符串

RabbitMQ cookie 的 salt。如果修改这个值，会强制随机产生改变的 RabbitMQ cookie。

RabbitFDLimit

类型： 字符串

配置 RabbitMQ file descriptor 的限制。

RabbitIPv6

类型： 布尔值

在 RabbitMQ 中启用 IPv6。

RabbitPassword

类型： 字符串

RabbitMQ 的密码。

RabbitUserName

类型： 字符串

RabbitMQ 的用户名

RedisPassword

类型： 字符串

Redis 的密码

SaharaPassword

类型： 字符串

OpenStack Clustering 服务帐号的秘密。

ServerMetadata

类型： json

在 Overcloud 中创建节点时传递给 OpenStack Compute 的额外属性或元数据。

ServiceNetMap

类型： json

服务名到网络名的映射信息。通常在自已注册表中的 **`parameter_defaults`** 中设置。

SnmpdReadonlyUserName

类型： 字符串

在所有 Overcloud 节点上运行的、带有只读权限的 SNMPd 的用户名。

SnmpdReadonlyUserPassword

类型： 字符串

在所有 Overcloud 节点上运行的、带有只读权限的 SNMPd 的密码。

StorageMgmtVirtualFixedIPs

类型： json

控制对 StorageMgmgVirtualInterface 端口的 IP 分配。例如：

-

```
[{'ip_address':'1.2.3.4'}]
```

StorageVirtualFixedIPs

类型： json

控制对 **`StorageVirtualInterface`** 端口的 IP 分配。例如：

```
[{'ip_address':'1.2.3.4'}]
```

SwiftHashSuffix

类型： 字符串

在使用哈希决定 ring 中的映射时，用来作为 salt 使用的随机字符串。

SwiftMinPartHours

l类型：数字

在进行一个重平衡（rebalance）后，在删除 ring 中的一个分区前需要等待的最少时间（以小时为单位）。

SwiftMountCheck

类型： 布尔值

Object Storage account/container/object-server.conf 中的 **`mount_check`** 值。

SwiftPartPower

l类型：数字

在创建 Object Storage ring 时使用的分区 power。

SwiftPassword

类型： 字符串

Object Storage proxy 所使用的 Object Storage 服务帐号的密码。

SwiftReplicas

l类型：数字

在 Object Storage ring 中使用的副本数量。

SwiftStorageImage

类型： 字符串

用于设置 Object Storage 节点的镜像。

TimeZone

类型： 字符串

设置 Overcloud 部署的时区。如果 **`TimeZone`** 参数为空，Overcloud 会被默认设置为 **`UTC`** 时间。Director 会识别在时区数据库 */usr/share/zoneinfo/* 中定义的标准时区名。例如，如果您需要把时区设置为 **`Japan`**，您可以查看 of */usr/share/zoneinfo* 文件来找到适当的值：

```
$ ls /usr/share/zoneinfo/
Africa      Asia        Canada   Cuba   EST      GB       GMT-0
HST      iso3166.tab  Kwajalein  MST      NZ-CHAT   posix
right      Turkey     UTC       Zulu
America     Atlantic   CET      EET    EST5EDT  GB-Eire  GMT+0
Iceland  Israel       Libya      MST7MDT  Pacific   posixrules
ROC        UCT        WET
Antarctica  Australia  Chile    Egypt  Etc      GMT
Greenwich  Indian   Jamaica      MET        Navajo   Poland
PRC         ROK        Universal  W-SU
Arctic      Brazil     CST6CDT  Eire   Europe   GMT0
Hongkong   Iran     Japan        Mexico     NZ       Portugal
PST8PDT     Singapore  US         zone.tab
```

以上的输出列表包括了时区文件，以及包括额外时区文件的目录。例如，在上面的列表中，**Japan** 是一个单独的时区文件，而 **Africa** 是一个目录，它包括了其它额外的时区文件：

```
$ ls /usr/share/zoneinfo/Africa/
Abidjan      Algiers  Bamako  Bissau       Bujumbura   Ceuta
Dar_es_Salaam  El_Aaiun  Harare        Kampala   Kinshasa
Lome        Lusaka  Maseru     Monrovia  Niamey       Porto-Novo
Tripoli
Accra        Asmara   Bangui  Blantyre     Cairo       Conakry
Djibouti       Freetown  Johannesburg  Khartoum  Lagos
Luanda      Malabo  Mbabane    Nairobi   Nouakchott   Sao_Tome
Tunis
Addis_Ababa  Asmera   Banjul  Brazzaville  Casablanca  Dakar
Douala         Gaborone  Juba          Kigali    Libreville
Lubumbashi  Maputo  Mogadishu  Ndjamena  Ouagadougou  Timbuktu
Windhoek
```

UpdateIdentifier

类型： 字符串

在 **stack-update** 期间设置为以前未使用的值会在所有节点上触发软件包更新。

附录 E. 网络接口参数

下表包括了网络接口类型的 Heat 模板参数。

E.1. 接口选项

选项	默认	描述
name		接口名称
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给接口的 IP 地址序列
routes		分配给接口的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
primary	False	作为主接口的接口
defroute	True	使用这个接口作为默认路由
persist_mapping	False	写入设备别名设置，而不是写入系统名
dhclient_args	无	传递给 DHCP 客户端的参数
dns_servers	无	为这个接口使用的 DNS 服务器列表

E.2. VLAN 选项

选项	默认	描述
vlan_id		VLAN ID
device		附加 VLAN 的 VLAN 父设备。例如，使用这个参数把 VLAN 附加到一个绑定的接口设备。
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给 VLAN 的 IP 地址序列
routes		分配给 VLAN 的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
primary	False	作为主接口的 VLAN
defroute	True	使用这个接口作为默认路由
persist_mapping	False	写入设备别名设置，而不是写入系统名
dhclient_args	无	传递给 DHCP 客户端的参数
dns_servers	无	为 VLAN 使用的 DNS 服务器列表

E.3. OVS 绑定选项

选项	默认	描述
name		绑定名
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给绑定的 IP 地址序列
routes		分配给绑定的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
primary	False	作为主接口的接口
members		在绑定中使用的接口项序列
ovs_options		在创建绑定时传递给 OVS 的一组选项
ovs_extra		在绑定的网络配置文件中设置为 OVS_EXTRA 参数的一组选项
defroute	True	使用这个接口作为默认路由
persist_mapping	False	写入设备别名设置，而不是写入系统名
dhclient_args	无	传递给 DHCP 客户端的参数

dns_servers	无	为绑定使用的 DNS 服务器列表

E.4. OVS 网桥选项

选项	默认	描述
name		网桥名
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给网桥的 IP 地址序列
routes		分配给网桥的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
members		在网桥中使用的一组接口、VLAN 和绑定项序列
ovs_options		在创建网桥时传递给 OVS 的一组选项
ovs_extra		在网桥的网络配置文件中设置为 OVS_EXTRA 参数的一组选项
defroute	True	使用这个接口作为默认路由
persist_mapping	False	写入设备别名设置，而不是写入系统名

dhclient_args	无	传递给 DHCP 客户端的参数
dns_servers	无	为网桥使用的 DNS 服务器列表

E.5. LINUX 绑定选项

选项	默认	描述
name		绑定名
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给绑定的 IP 地址序列
routes		分配给绑定的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
primary	False	作为主接口的接口
members		在绑定中使用的接口项序列
bonding_options		在创建绑定时的一组选项。如需了解更多与 Linux 绑定选项相关的信息，请参阅 Red Hat Enterprise Linux 7 Networking Guide 中的 4.5.1. Bonding Module Directives。

defroute	True	使用这个接口作为默认路由
persist_mapping	False	写入设备别名设置，而不是写入系统名
dhclient_args	无	传递给 DHCP 客户端的参数
dns_servers	无	为绑定使用的 DNS 服务器列表

E.6. LINUX 网桥选项

选项	默认	描述
name		网桥名
use_dhcp	False	使用 DHCP 分配 IP 地址
use_dhcpv6	False	使用 DHCP 分配 IPv6 地址
addresses		分配给网桥的 IP 地址序列
routes		分配给网桥的路由序列
mtu	1500	连接的最大传输单元（maximum transmission unit，简称 MTU）
members		在网桥中使用的一组接口、VLAN 和绑定项序列
defroute	True	使用这个接口作为默认路由

persist_mapping	False	写入设备别名设置，而不是写入系统名
dhclient_args	无	传递给 DHCP 客户端的参数
dns_servers	无	为网桥使用的 DNS 服务器列表

附录 F. 网络接口模板示例

在这个附录中包括了一组 Heat 模板示例，它们展示了网络接口的配置。

F.1. 配置接口

每个接口可能会需要根据实际情况进行修改。例如，如果您需要使用第二个 NIC 连接到一个使用 DHCP 地址的网络，并使用第 3 和第 4 个 NIC 进行网络绑定时，您所需要进行的修改如下：

```
network_config:
  # Add a DHCP infrastructure network to nic2
  -
    type: interface
    name: nic2
    use_dhcp: true
  -
    type: ovs_bridge
    name: br-bond
    members:
      -
        type: ovs_bond
        name: bond1
        ovs_options: {get_param: BondInterfaceOvsOptions}
        members:
          # Modify bond NICs to use nic3 and nic4
          -
            type: interface
            name: nic3
            primary: true
          -
            type: interface
            name: nic4
```

网络接口模板可以使用实际的接口名（"eth0"、"eth1"、"enp0s25"），或使用一组接口编号（"nic1"、"nic2"、"nic3")。当使用接口编号（如 **nic1**、**nic2** 等）而不是使用接口名（如 **eth0**、**eno2** 等）时，在一个角色中的主机网络接口不需要是完全相同的。例如，一个主机可能有接口 **em1** 和 **em2**，而另外一个主机有接口 **eno1** 和 **eno2**，您可以使用 **nic1** 和 **nic2** 来指代这两个主机的接口。

编号的接口顺序与使用名称的网络接口类型相对应：

- **ethX** 接口，如 **eth0**、**eth1** 等。这些通常是板上的内置接口。
- **enoX** 接口，如 **eno0**、**eno1** 等。这些通常是板上的内置接口。
- **enX** 接口，以字母顺序排序，如 **enp3s0**、**enp3s1**、**ens3** 等。这些通常是额外附加的接口。

编号的 NIC 机制只会考虑有效的接口（如有网线连接到交换机）。如果您有一些带有 4 个接口的 主机，另外还有一些带有 6 个接口的主机，则应该使用 **nic1** 到 **nic4**，并只在每个主机中的 4 个接口中连接网线。

F.2. 配置路由和默认路由

主机可以通过两种方法获得默认路由设置。如果接口使用 DHCP，而且 DHCP 服务器提供了一个网关地址，系统会使用那个网关作为默认路由。在其它情况下，您可以使用一个静态 IP 在一个接口上设置一个默认的路由。

虽然 Linux 内核支持多个默认网关，但它只会使用 metric 值最低的那个。当有多个 DHCP 接口时，就可能会造成不可预测的默认网关。在这种情况下，推荐为不被用作默认路由使用的接口设置 **defroute=no**。

例如，您可能需要一个 DHCP 接口（**nic3**）作为默认的路由。使用以下 YAML 禁用另外一个 DHCP 接口（**nic2**）上的路由：

```
# No default route on this DHCP interface
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```

注意

defroute 参数只会应用到通过 DHCP 获得的路由。

如需在一个接口上使用静态 IP 设置一个静态路由，为子网指定一个路由。例如，在 Internal API 网络上把到 10.1.2.0/24 子网的路由设置为使用地址为 172.17.0.1 的网关：

```
    - type: vlan
      device: bond1
      vlan_id: {get_param: InternalApiNetworkVlanID}
      addresses:
      - ip_netmask: {get_param: InternalApiIpSubnet}
      routes:
      - ip_netmask: 10.1.2.0/24
        next_hop: 172.17.0.1
```

F.3. 为浮动 IP 使用原生 VLAN

Neutron 使用一个默认的空字符串作为它的外部网络映射，这会把物理接口映射到 **br-int**，而不是直接使用 **br-ex**。这种模式允许多个 Floating IP 网络使用 VLAN 或多个物理连接。

在网络分离环境文件中的 **parameter_defaults** 部分使用 **NeutronExternalNetworkBridge** 参数：

```
  parameter_defaults:
    # Set to "br-ex" when using floating IPs on the native VLAN
    NeutronExternalNetworkBridge: "''"
```

在一个网桥的原生 VLAN 中只使用一个 Floating IP 网络意味着，可以选择设置 neutron 外部网桥。这会导致网络数据包只通过一个网桥，而不是通过两个网桥，从而使网络数据通过 Floating IP 网络时，少量降低对 CPU 的使用。

下一节包括了放置到原生 VLAN 的 External 网络中的 NIC 配置的改变。如果 External 网络被映射到 **br-ex**，除了可以用于 Horizon 仪表板和 Public API，External 网络还可以被用于 Floating IP。

F.4. 在一个主干接口上使用原生 VLAN

如果一个主干接口或绑定有一个在原生 VLAN 中的网络，IP 地址将会被直接分配到网桥，而不会创建一个 VLAN 接口。

例如，当 External 网络在一个原生 VLAN 中时，一个绑定的配置将会和以下类似：

```
network_config:
  - type: ovs_bridge
    name: {get_input: bridge_name}
    dns_servers: {get_param: DnsServers}
    addresses:
      - ip_netmask: {get_param: ExternalIpSubnet}
    routes:
      - ip_netmask: 0.0.0.0/0
        next_hop: {get_param: ExternalInterfaceDefaultRoute}
    members:
      - type: ovs_bond
        name: bond1
        ovs_options: {get_param: BondInterfaceOvsOptions}
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

注意

当把地址（以及可能的路由）声明移到网桥上时，需要把相应的 VLAN 接口从网桥上删除。这个改变需要在所有相关的节点上进行。External 网络只会位于控制器上，所以只有控制器模板需要被修改。Storage 网络的情况则不同，它可以被附加到所有角色上，因此当 Storage 网络位于默认的 VLAN 中时，所有的角色都需要被修改。

F.5. 配置大型帧

最大传输单元（Maximum Transmission Unit，简称 MTU）的设置决定了一个以太网帧可以传输的最大的数据量。因为每个帧除了数据外都需要一个头数据，所以为 MTU 设置一个较大的值可以减少系统的额外消耗。它的默认值是 1500，如果使用一个更高的数值，则需要把交换端口配置为支持大型帧。多数交换设备都最少支持 MTU 的值为 9000，但一些设备把这个值默认设置为 1500。

VLAN 的 MTU 值不能超过物理接口的 MTU 的值。请确保在绑定和接口中包括了 MTU 的值。

使用大型帧对 Storage 网络、Storage Management 网络、Internal API 网络和 Tenant 网络都有好处。作为测试结果，当和 VXLAN tunnel 一起使用大型帧时，Tenant 网络的吞吐量会提高 300%。

注意

我们推荐 Provisioning 接口、External 接口和所有浮动 IP 接口使用默认的 MTU 值 - 1500。如果使用其它值，可能会出现一些连接的问题。这是因为，路由器通常无法在第 3 层网络边界间转发大型的数据帧。

```
- type: ovs_bond
  name: bond1
  mtu: 9000
  ovs_options: {get_param: BondInterfaceOvsOptions}
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000

# The external interface should stay at default
- type: vlan
  device: bond1
  vlan_id: {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    - ip_netmask: 0.0.0.0/0
      next_hop: {get_param: ExternalInterfaceDefaultRoute}

# MTU 9000 for Internal API, Storage, and Storage Management
- type: vlan
  device: bond1
  mtu: 9000
  vlan_id: {get_param: InternalApiNetworkVlanID}
  addresses:
  - ip_netmask: {get_param: InternalApiIpSubnet}
```

附录 G. 网络环境选项

表 G.1. 网络环境选项

参数	描述	示例
InternalApiNetCidr	Internal API 网络的网络和子网	172.17.0.0/24
StorageNetCidr	Storage 网络的网络和子网	
StorageMgmtNetCidr	Storage Management 网络的网络和子网	
TenantNetCidr	Tenant 网络的网络和子网	
ExternalNetCidr	External 网络的网络和子网	
InternalApiAllocationPools	Internal API 网络的分配池（tuple 格式）	[\{*start*: *172.17.0.10*, *end*: *172.17.0.200*}]
StorageAllocationPools	Storage 网络的分配池（tuple 格式）	
StorageMgmtAllocationPools	Storage Management 网络的分配池（tuple 格式）	
TenantAllocationPools	Tenant 网络的分配池（tuple 格式）	
ExternalAllocationPools	External 网络的分配池（tuple 格式）	
InternalApiNetworkVlanID	Internal API 网络的 VLAN ID	200
StorageNetworkVlanID	Storage 网络的 VLAN ID	

参数	描述	示例
StorageMgmtNetworkVlanID	Storage Management 网络的 VLAN ID	
TenantNetworkVlanID	Tenant 网络的 VLAN ID	
ExternalNetworkVlanID	External 网络的 VLAN ID	
ExternalInterfaceDefaultRoute	External 网络的网关 IP 地址	10.1.2.1
ControlPlaneDefaultRoute	Provisioning 网络的网关路由（或 Undercloud IP）	ControlPlaneDefaultRoute: 192.0.2.254
ControlPlaneSubnetCidr	provisioning 网络的CIDR 子网掩码的长度	ControlPlaneSubnetCidr: 24
EC2MetadataIp	EC2 元数据服务器的 IP 地址。通常情况下是 Undercloud 的 IP。	EC2MetadataIp: 192.0.2.1
DnsServers	为 Overcloud 节点定义 DNS 服务器。最多可以包括两个。	DnsServers: ["8.8.8.8","8.8.4.4"]
BondInterfaceOvsOptions	绑定接口的选项	BondInterfaceOvsOptions:"bond_mode=balance-slb"
NeutronFlatNetworks	定义在 neutron 插件中配置的平面网络（flat nework）。默认是 "datacentre"，允许外部网络创建	NeutronFlatNetworks: "datacentre"
NeutronExternalNetworkBridge	在每个 hypervisor 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	NeutronExternalNetworkBridge: "br-ex"

参数	描述	示例
NeutronBridgeMappings	使用的物理网桥映射逻辑。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（floating network）	NeutronBridgeMappings: "datacentre:br-ex"
NeutronPublicInterface	定义网络节点的 br-ex 中的网桥接口	NeutronPublicInterface: "eth0"
NeutronNetworkType	Neutron 的租户网络类型	NeutronNetworkType: "vxlan"
NeutronTunnelTypes	neutron 租户网络的隧道（tunnel）类型。使用逗号分隔的字符串可以指定多个值	NeutronTunnelTypes: *gre,vxlan*
NeutronTunnelIdRanges	可以用来进行租户网络分配的 GRE tunnel ID 的范围	NeutronTunnelIdRanges "1:1000"
NeutronVniRanges	可以用来进行租户网络分配的 VXLAN VNI ID 范围	NeutronVniRanges: "1:1000"
NeutronEnableTunnelling	启用或禁用隧道（tunneling）功能来在 Neutron 中使用 VLAN 分段网络或平面网络。默认设置是启用。	
NeutronNetworkVLANRanges	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: "datacentre:1:1000"
NeutronMechanismDrivers	neutron 租户网络的驱动。默认值是 "openvswitch"。使用逗号分隔的字符串可以指定多个值	NeutronMechanismDrivers: *openvswitch,l2population*

附录 H. OPEN VSWITCH 绑定选项

Overcloud 通过 Open vSwitch（OVS）提供网络功能，对于绑定的接口，它提供了多个选项。在 第 6.2.2 节 “创建一个网络环境文件” 中，您可以在网络环境文件中使用以下参数配置一个绑定的接口：

```
BondInterfaceOvsOptions:
  "bond_mode=balance-slb"
```

以下表格提供了对这些选项的解释信息，以及根据您的具体硬件而提供的不同设置。

重要

不要在 LACP 中使用基于 OVS 的绑定，因为这个配置有问题，而且不被支持。可以使用 *bond_mode=balance-slb* 作为替代来实现相关的功能。另外，您仍然可以在网络接口模板中使用带有 Linux 绑定的 LACP。例如：

```
        - type: linux_bond
          name: bond1
          members:
          - type: interface
            name: nic2
          - type: interface
            name: nic3
          bonding_options: "mode=802.3ad lacp_rate=[fast|slow]
updelay=1000 miimon=100"
```

- **mode** - 启用 LACP。
- **lacp_rate** - 指定 LACP 数据包是每 1 秒还是每 30 秒发送一次。
- **updelay** - 指定一个接口在可以用来处理网络流量前，它已被激活的最短时间（这个设置有助于防止出现 port flapping outage 的问题）。
- **miimon** - 以毫秒为单位的间隔，它用于使用驱动程序的 MIIMON 功能监视端口状态。

如需了解更多与 Linux 绑定选项相关的信息，请参阅 *Red Hat Enterprise Linux 7 Networking Guide* 中的 4.5.1. Bonding Module Directives。

表 H.1. 绑定选项

bond_mode=balance-slb	负载平衡操作是基于源 MAC 地址和输出的 VLAN 进行的，并在网络数据特性出现变化时进行定期的再平衡。带有 **balance-slb** 的绑定允许在不需要远程交换机支持的情况下进行一定形式的负载平衡操作。SLB 会把每个源 MAC 和 VLAN 对分配到一个链接，并通过这个链接从 MAC 和 VLAN 发送所有数据包。这个模式使用一个简单的基于 MAC 地址和 VLAN 号的哈希算法，并在网络数据特性出现变化时进行定期的再平衡。它与 Linux 绑定驱动所使用的 mode 2 绑定类似。这个模式会在交换机配置了绑定，但却没有配置使用 LACP （静态绑定而不是动态绑定）的情况下使用。
bond_mode=active-backup	这个模式提供了一个 active/standby 方式的故障转移功能 - 当 active 连接出现问题时，standy NIC 会恢复网络操作。这只需要在物理交换机上存在一个 MAC 地址。这种模式不需要任何特殊的交换机支持或配置，当连接到不同交换机时同样可以正常工作。这个模式不支持负载平衡。
lacp=[active|passive|off]	控制 LACP（Link Aggregation Control Protocol）操作。只有特定交换机才支持 LACP。如果您的交换机不支持 LACP，请使用 **bond_mode=balance-slb** 或 **bond_mode=active-backup**。 不要在 LACP 中使用基于 OVS 的绑定，因为这个配置有问题，而且不被支持。可以使用 *bond_mode=balance-slb* 作为替代来实现相关的功能。另外，您仍然可以使用带有 Linux 绑定的 LACP：
other-config:lacp-fallback-ab=true	在交换机上把 LACP 设置为 bond_mode=active-backup 作为一个故障恢复。
other_config:lacp-time=[fast|slow]	把 LACP 的"心跳"设置设为 1 秒（fast）或 30 秒（slow）。默认值是 slow。
other_config:bond-detect-mode=[miimon|carrier]	把连接监测的间隔设置为 miimon heartbeat（miimon）或 monitor carrier（carrier）。默认值是 carrier。

other_config:bond-miimon-interval=100	如果使用 miimon，设置心跳间隔（以毫秒为单位）。
other_config:bond_updelay=1000	为了防止发生网络转移，一个连接必须处于活跃状态的时间（以毫秒为单位）
other_config:bond-rebalance-interval=10000	在绑定设备间再平衡网络数据的时间（以毫米为单位）。设为 0 禁用这个功能。

重要

如果在 Provider 网络中使用 Linux 绑定出现丢数据包问题，或出现性能问题，可以考虑在备用（standby）接口中禁用 LRO（Large Receive Offload）。避免为 OVS 绑定添加 Linux 绑定，这样做可能会出现 port-flapping 和连接丢失的问题。