Red Hat OpenStack Platform 17.1

自定义 Red Hat OpenStack Platform 部署

根据您的环境和要求自定义 Red Hat OpenStack Platform 核心部署。

Last Updated: 2024-05-23

Red Hat OpenStack Platform 17.1 自定义 Red Hat OpenStack Platform 部署

根据您的环境和要求自定义 Red Hat OpenStack Platform 核心部署。

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

有关如何根据环境和要求自定义 Red Hat OpenStack Platform 基本部署以及如何使用 Ansible 和编排服务的指导。

目录

使开源包含更多

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

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

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

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

第 1 章 规划自定义 UNDERCLOUD 功能

在 undercloud 上配置并安装 director 之前，您可以计划在 undercloud 中包含自定义功能。

1.1. 字符编码配置

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

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

1.2. 使用代理运行 UNDERCLOUD 时的注意事项

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

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

系统范围的代理配置

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

http_proxy
: 用于标准 HTTP 请求的代理。

https_proxy
: 用于 HTTP 请求的代理。

no_proxy
: 从代理通信中排除的、以逗号分隔的域列表。

系统范围的代理方法有以下限制：

- 由于 **pam_env** PAM 模块中的固定大小缓冲，**no_proxy** 的最大长度为 1024 个字符。

dnf 代理配置

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

proxy
: 代理服务器的 URL。

proxy_username
: 用于连接到代理服务器的用户名。

proxy_password
: 用于连接到代理服务器的密码。

proxy_auth_method
: 代理服务器使用的身份验证方法。

有关这些选项的更多信息，请运行 **man dnf.conf**。

dnf 代理方法有以下限制：

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

Red Hat Subscription Manager 代理

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

proxy_hostname
用于代理的主机。

proxy_scheme
在将代理写入软件仓库定义时，代理的方案。

proxy_port
代理的端口。

proxy_username
用于连接到代理服务器的用户名。

proxy_password
用于连接到代理服务器的密码。

no_proxy
要从代理通信中排除的、以逗号分隔的特定主机的主机名后缀列表。

有关这些选项的更多信息，请运行 **man rhsm.conf**。

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

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

透明代理

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

第 2 章 可组合服务和自定义角色

overcloud 通常由预定义角色（如 Controller 节点、Compute 节点和不同的存储节点类型）中的节点组成。这些默认角色各自包含 director 节点上的核心 heat 模板集合中定义的一组服务。但是，您也可以创建包含特定服务集的自定义角色。

您可以使用此灵活性在不同的角色上创建不同的服务组合。本章介绍了自定义角色、可组合服务和使用它们的方法的架构。

2.1. 支持的角色架构

使用自定义角色和可组合服务时可以使用以下架构：

默认架构

使用默认的 **roles_data** 文件。所有控制器服务都包含在一个 Controller 角色中。

自定义可组合服务

创建自己的角色，并使用它们生成自定义 **roles_data** 文件。请注意，只测试并验证了有限的可组合服务组合，红帽无法支持所有可组合服务组合。

2.2. 检查 ROLES_DATA 文件

roles_data 文件包含 director 部署到节点上的角色的 YAML 格式列表。每个角色都包含组成该角色的所有服务的定义。使用以下示例片断以了解 **roles_data** 语法：

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

核心 heat 模板集合包括一个默认的 **roles_data** 文件，位于 **/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**。默认文件包含以下角色类型的定义：

- **Controller**
- **Compute**
- **BlockStorage**
- **ObjectStorage**
- **Ceph 存储**.

openstack overcloud deploy 命令在部署过程中包含默认的 **roles_data.yaml** 文件。但是，您可以使用 **-r** 参数使用自定义 **roles_data** 文件覆盖此文件：

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-custom.yaml
```

2.3. 创建 ROLES_DATA 文件

虽然您可以手动创建自定义 **roles_data** 文件，但您也可以使用单独的角色模板自动生成该文件。director 提供 **openstack overcloud role generate** 命令来加入多个预定义角色，并自动生成自定义 **roles_data** 文件。

流程

1. 列出默认角色模板：

```
$ openstack overcloud role list
BlockStorage
CephStorage
Compute
ComputeHCI
ComputeOvsDpdk
Controller
...
```

2. 查看角色定义：

```
$ openstack overcloud role show Compute
```

3. 生成包含 **Controller**、**Compute** 和 **Networker** 角色的自定义 **roles_data.yaml** 文件：

```
$ openstack overcloud roles \
 generate -o <custom_role_file> \
 Controller Compute Networker
```

- 将 **<custom_role_file>** 替换为要生成的新角色文件的名称和位置，如 **/home/stack/templates/roles_data.yaml**。

注意

Controller 和 **Networker** 角色包含相同的网络代理。这意味着网络服务从 **Controller** 角色扩展到 **Networker** 角色，overcloud 会平衡 **Controller** 和 **Networker** 节点之间网络服务的负载。

要使此 **Networker** 角色独立，您可以创建自己的自定义 **Controller** 角色，以及您需要的任何其他角色。这可让您从您自己的自定义角色生成 **roles_data.yaml** 文件。

4. 将 **roles** 目录从核心 heat 模板集合复制到 **stack** 用户的主目录：

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
```

5. 添加或修改此目录中的自定义角色文件。使用带有任何角色子命令的 **--roles-path** 选项，将这个目录用作自定义角色的源：

```
$ openstack overcloud role \
 generate -o my_roles_data.yaml \
 --roles-path /home/stack/templates/roles \
 Controller Compute Networker
```

此命令从 **~/roles** 目录中的单个角色生成单个 **my_roles_data.yaml** 文件。

注意

默认角色集合还包含 **ControllerOpenstack** 角色，该角色不包括 **Networker**, **Messaging**, 和 **Database** 角色的服务。您可以将 **ControllerOpenstack** 与独立的 **Networker**, **Messaging**, 和 **Database** 角色结合使用。

2.4. 支持的自定义角色

下表包含有关可用自定义角色的信息。您可以在 **/usr/share/openstack-tripleo-heat-templates/roles** 目录中找到自定义角色模板。

角色	描述	File
BlockStorage	OpenStack Block Storage (cinder)节点。	**BlockStorage.yaml**
CephAll	完整的单机 Ceph Storage 节点。包括 OSD、MON、对象网关(RGW)、对象操作(MDS)、管理器(MGR)和 RBD 镜像功能。	**CephAll.yaml**
CephFile	独立横向扩展 Ceph Storage 文件角色。包括 OSD 和对象操作(MDS)。	**CephFile.yaml**
CephObject	独立横向扩展 Ceph Storage 对象角色。包括 OSD 和对象网关(RGW)。	**CephObject.yaml**
CephStorage	Ceph Storage OSD 节点角色。	**CephStorage.yaml**
ComputeAlt	备用 Compute 节点角色.	**ComputeAlt.yaml**
ComputeDVR	DVR 启用的 Compute 节点角色。	**ComputeDVR.yaml**
ComputeHCI	具有超融合基础架构的计算节点。包括计算和 Ceph OSD 服务。	**ComputeHCI.yaml**
ComputeInstanceHA	Compute 实例 HA 节点角色.与 **environments/compute-instanceha.yaml 的环境文件** 一起使用。	**ComputeInstanceHA.yaml**
ComputeLiquidio	带有 Cavium Liquidio 智能 NIC 的计算节点.	**ComputeLiquidio.yaml**
ComputeOvsDpdkRT	计算 OVS DPDK RealTime 角色。	**ComputeOvsDpdkRT.yaml**

角色	描述	File
ComputeOvsDpdk	计算 OVS DPDK 角色。	**ComputeOvsDpdk.yaml**
ComputeRealTime	为实时行为优化的计算角色。使用此角色时，必须根据实时计算节点的硬件设置 **overcloud-realtime-compute** 镜像以及角色特定参数 **IsolCpusList**、**NovaComputeCpuDedicatedSet** 和 **NovaComputeCpuSharedSet**。	**ComputeRealTime.yaml**
ComputeSriovRT	Compute SR-IOV RealTime 角色。	**ComputeSriovRT.yaml**
ComputeSriov	计算 SR-IOV 角色。	**ComputeSriov.yaml**
Compute	标准 Compute 节点角色。	**Compute.yaml**
ControllerAllNovaStandalone	不包含数据库、消息传递、网络和 OpenStack Compute (nova)控制组件的控制器角色。使用 **Database**, **Messaging**, **Networker**, 和**Novacontrol** 角色的组合。	**ControllerAllNovaStandalone.yaml**
ControllerNoCeph	载入核心 Controller 服务的控制器角色，但没有 Ceph Storage (MON) 组件。此角色处理数据库、消息传递和网络功能，但不处理任何 Ceph 存储功能。	**ControllerNoCeph.yaml**
ControllerNovaStandalone	不包含 OpenStack Compute (nova)控制组件的控制器角色。与 **Novacontrol** 角色结合使用。	**ControllerNovaStandalone.yaml**
ControllerOpenstack	不包含数据库、消息传递和网络组件的控制器角色。与 **Database**, **Messaging**, 和 **Networker** 角色结合使用。	**ControllerOpenstack.yaml**
ControllerStorageNfs	加载所有核心服务的控制器角色，并使用 Ceph NFS。此角色处理数据库、消息传递和网络功能。	**ControllerStorageNfs.yaml**
Controller	加载所有核心服务的控制器角色。此角色处理数据库、消息传递和网络功能。	**Controller.yaml**
ControllerSriov (ML2/OVN)	与普通的 Controller 角色相同，但部署了 OVN 元数据代理。	**ControllerSriov.yaml**
数据库	独立数据库角色。使用 Pacemaker 将数据库作为 Galera 集群进行管理。	**Database.yaml**
HciCephAll	具有超融合基础架构和所有 Ceph 存储服务的计算节点。包括 OSD、MON、对象网关(RGW)、对象操作(MDS)、管理器(MGR)和 RBD 镜像功能。	**HciCephAll.yaml**

角色	描述	File
HciCephFile	具有超融合基础架构和 Ceph Storage 文件服务的计算节点。包括 OSD 和对象操作(MDS)。	**HciCephFile.yaml**
HciCephMon	具有超融合基础架构和 Ceph Storage 块服务的计算节点。包括 OSD、MON 和 Manager。	**HciCephMon.yaml**
HciCephObject	具有超融合基础架构和 Ceph 存储对象服务的计算节点。包括 OSD 和对象网关(RGW)。	**HciCephObject.yaml**
IronicConductor	Ironic 行为节点角色.	**IronicConductor.yaml**
消息传递	独立消息传递角色.使用 Pacemaker 管理的 RabbitMQ。	**Messaging.yaml**
Networker	独立网络角色。自行运行 OpenStack 网络(neutron)代理。如果您的部署使用 ML2/OVN 机制驱动程序，请参阅 使用 ML2/OVN 部署自定义角色 中的其他步骤。	**Networker.yaml**
NetworkerSriov	与普通的 Networker 角色相同，但部署了 OVN 元数据代理。请参阅 使用 ML2/OVN 部署自定义角色 中的其他步骤。	**NetworkerSriov.yaml**
Novacontrol	独立 **nova-control** 角色，以自行运行 OpenStack Compute (nova)控制代理。	**Novacontrol.yaml**
ObjectStorage	Swift Object Storage 节点角色。	**ObjectStorage.yaml**
Telemetry	包含所有指标和警报服务的遥测角色。	**Telemetry.yaml**

2.5. 检查角色参数

每个角色包含以下参数：

name

（必需） 角色的名称，这是没有空格或特殊字符的纯文本名称。检查所选名称不会导致与其他资源冲突。例如，使用 **Networker** 作为名称而不是 **Network**。

description

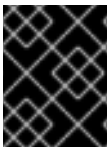
（可选） 角色的纯文本描述。

tags

（可选） 定义角色属性的标签的 YAML 列表。使用此参数定义主角色，带有 **controller** 和 **primary** 标签：

```
- name: Controller
  ...
  tags:
```

```
    - primary
    - controller
...
```



重要

如果您没有标记主要角色，则您定义的第一个角色将成为主要角色。确保此角色是 Controller 角色。

networks

要在角色上配置的网络的 YAML 列表或字典。如果使用 YAML 列表，请列出每个可组合网络：

```
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
```

如果您使用一个字典，请将每个网络映射到可组合网络中的一个特定的**子网**。

```
networks:
  External:
    subnet: external_subnet
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
```

默认网络包括 **External**, **InternalApi**, **Storage**, **StorageMgmt**, **Tenant**, 和 **Management**。

CountDefault

（可选） 定义您要为此角色部署的默认节点数。

HostnameFormatDefault

（可选） 定义角色的默认主机名格式。默认命名惯例使用以下格式：

```
[STACK NAME]-[ROLE NAME]-[NODE ID]
```

例如，默认 Controller 节点被命名：

```
overcloud-controller-0
overcloud-controller-1
overcloud-controller-2
...
```

disable_constraints

（可选） 定义在使用 director 部署时是否禁用 OpenStack Compute (nova)和 OpenStack Image Storage (glance)约束。当您使用预置备节点部署 overcloud 时，请使用此参数。如需更多信息，请参阅 *Director 安装和使用指南中的* 使用预置备节点配置 基本 overcloud。

update_serial

（可选） 定义在 OpenStack 更新选项期间同时更新的节点数量。在默认的 **roles_data.yaml** 文件中：

- 对于 Controller、Object Storage 和 Ceph Storage 节点，默认为 **1**。
- Compute 和 Block Storage 节点的默认值为 **25**。

如果从自定义角色中省略此参数，则默认为 **1**。

ServicesDefault

（可选） 定义要在节点上包含的默认服务列表。更多信息请参阅 第 2.10 节 “检查可组合服务架构”。

您可以使用这些参数创建新角色，并定义角色中包含的服务。

openstack overcloud deploy 命令将 **roles_data** 文件中的参数集成到一些基于 Jinja2 的模板中。例如，在某些点上，**overcloud.j2.yaml** heat 模板会迭代 **roles_data.yaml** 中的角色列表，并创建特定于每个对应角色的参数和资源。

例如，以下代码片段包含 **overcloud.j2.yaml** heat 模板中每个角色的资源定义：

```
  {{role.name}}:
    type: OS::Heat::ResourceGroup
    depends_on: Networks
    properties:
      count: {get_param: {{role.name}}Count}
      removal_policies: {get_param: {{role.name}}RemovalPolicies}
      resource_def:
        type: OS::TripleO::{{role.name}}
        properties:
          CloudDomain: {get_param: CloudDomain}
          ServiceNetMap: {get_attr: [ServiceNetMap, service_net_map]}
          EndpointMap: {get_attr: [EndpointMap, endpoint_map]}
...
```

此代码片段演示了基于 Jinja2 的模板如何融合了 **{{role.name}}** 变量，用于将每个角色的名称定义为 **OS::Heat::ResourceGroup** 资源。这会依次使用 **roles_data** 文件中的每个 **name** 参数来命名各个对应的 **OS::Heat::ResourceGroup** 资源。

2.6. 创建新角色

您可以根据部署的要求，使用可组合服务架构将角色分配给裸机节点。例如，您可能希望创建新的 **Horizon** 角色来仅托管 OpenStack 控制面板(**horizon**)。

流程

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

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

3. 将 **roles** 目录从核心 heat 模板集合复制到 **stack** 用户的主目录：

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
```

4. 在 **home/stack/templates/roles** 中创建一个名为 **Horizon.yaml** 的新文件。

5. 在 **Horizon.yaml** 中添加以下配置，以创建一个包含基本和核心 OpenStack Dashboard 服务的新 **Horizon** 角色：

```
- name: Horizon (1)
  CountDefault: 1 (2)
  HostnameFormatDefault: '%stackname%-horizon-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

(1) 将 **name** 参数设置为自定义角色的名称。自定义角色名称的最大长度为 47 个字符。

(2) 将 **CountDefault** 参数设置为 **1**，以便默认 overcloud 始终包含 **Horizon** 节点。

6. 可选：如果要在现有 overcloud 中扩展服务，请在 **Controller** 角色上保留现有服务。如果要创建新 overcloud，并且希望 OpenStack 控制面板保留在独立角色上，请从 **Controller** 角色定义中删除 OpenStack 仪表板组件：

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    ...
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    # - OS::TripleO::Services::Horizon          # Remove this service
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    ...
```

7. 生成一个名为 **roles_data_horizon.yaml** 的新角色数据文件，其中包含 **Controller**、**Compute** 和 **Horizon** 角色：

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

8. 可选：编辑 **overcloud-baremetal-deploy.yaml** 节点定义文件以配置 Horizon 节点的放置：

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  ...
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  ...
- name: Horizon
  count: 1
  instances:
  - hostname: overcloud-horizon-0
    name: node07
```

2.7. 指南和限制

请注意可组合角色架构的以下准则和限制。

对于不由 Pacemaker 管理的服务：

- 您可以将服务分配给独立的自定义角色。
- 您可以在初始部署后创建额外的自定义角色，并部署它们以扩展现有服务。

对于由 Pacemaker 管理的服务：

- 您可以将 Pacemaker 管理的服务分配给独立的自定义角色。
- Pacemaker 具有 16 个节点限制。如果您将 Pacemaker 服务 (**OS::TripleO::Services::Pacemaker**)分配给 16 个节点，则后续节点必须使用 Pacemaker 远程服务(**OS::TripleO::Services::PacemakerRemote**)。您不能在同一角色上具有 Pacemaker 服务和 Pacemaker 远程服务。
- 不要在不包含 Pacemaker 管理的服务的角色中包含 Pacemaker 服务 (**OS::TripleO::Services::Pacemaker**)。
- 您无法扩展或缩减包含 **OS::TripleO::Services::Pacemaker** 或 **OS::TripleO::Services::PacemakerRemote** 服务的自定义角色。

常规限制：

- 您无法在主版本升级过程中更改自定义角色和可组合服务。
- 在部署 overcloud 后，您无法修改任何角色的服务列表。在 Overcloud 部署后修改服务列表可能会导致部署错误，并在节点上保留孤立的服务。

2.8. 容器化服务架构

director 将核心 OpenStack Platform 服务作为容器安装到 overcloud 上。容器化服务的模板位于 **/usr/share/openstack-tripleo-heat-templates/deployment/** 中。

您必须在角色中为使用容器化服务的所有节点启用 **OS::TripleO::Services::Podman** 服务。为自定义角色配置创建 **roles_data.yaml** 文件时，包括 **OS::TripleO::Services::Podman** 服务以及基本可组合服务。例如，Ironic **Condu** ctor 角色使用以下角色定义：

```
- name: IronicConductor
  description: |
    Ironic Conductor node role
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-ironic-%index%'
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicPxe
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
```

2.9. 容器化服务参数

每个容器化服务模板包含一个 **outputs** 部分，用于定义传递给 OpenStack Orchestration (heat)服务的数据集。除了标准可组合服务参数外，模板还包含一组特定于容器配置的参数。

puppet_config

配置服务时要传递给 Puppet 的数据。在初始 overcloud 部署步骤中，director 创建一组用于配置该服务的容器，然后再运行实际的容器化服务。此参数包括以下子参数：

- **config_volume** - 存储配置的挂载卷。
- **puppet_tags** - 在配置期间传递到 Puppet 的标签。OpenStack 使用这些标签将 Puppet 运行限制为特定服务的配置资源。例如，OpenStack Identity (keystone)容器化服务使用 **keystone_config** 标签来确保一切都只需要在配置容器上运行 **keystone_config** Puppet 资源。
- **step_config** - 传递给 Puppet 的配置数据。这通常继承自引用的可组合服务。
- **config_image** - 用于配置服务的容器镜像。

kolla_config

一组特定于容器的数据，用于定义配置文件位置、目录权限，以及在容器中运行的命令以启动服务。

docker_config

在该服务的配置容器中运行的任务。所有任务都分为以下步骤，以帮助 director 执行暂存部署：

- **第 1 步** - 负载均衡器配置
- **第 2 步** - 核心服务(Database、Redis)
- **第 3 步** - OpenStack Platform 服务初始配置
- **第 4 步** - 常规 OpenStack Platform 服务配置
- **第 5 步** - 服务激活

host_prep_tasks

为裸机节点准备任务以容纳容器化服务。

2.10. 检查可组合服务架构

核心 heat 模板集合包含两组可组合服务模板：

- **deployment** 包含关键 OpenStack 服务的模板。
- **puppet/services** 包含用于配置可组合服务的传统模板。在某些情况下，可组合服务使用此目录中的模板来实现兼容性。在大多数情况下，可组合服务使用 **部署** 目录中的模板。

每个模板都包含标识其用途的描述。例如，**deployment/time/ntp-baremetal-puppet.yaml** 服务模板包含以下描述：

```
description: >
  NTP service deployment using puppet, this YAML file
  creates the interface between the HOT template
  and the puppet manifest that actually installs
  and configure NTP.
```

这些服务模板注册为特定于 Red Hat OpenStack Platform 部署的资源。这意味着，您可以使用 **overcloud-resource-registry-puppet.j2.yaml** 文件中定义的唯一 heat 资源命名空间调用每个资源。所有服务都使用 **OS::TripleO::Services** 命名空间作为其资源类型。

有些资源直接使用基础可组合服务模板：

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: deployment/time/ntp-baremetal-puppet.yaml
  ...
```

但是，核心服务需要容器并使用容器化服务模板。例如，**keystone** 容器化服务使用以下资源：

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: deployment/keystone/keystone-container-puppet.yaml
  ...
```

这些容器化模板通常引用其他模板，以包括依赖项。例如，**deployment/keystone/keystone-container-puppet.yaml** 模板将基础模板的输出存储在 **ContainersCommon** 资源中：

```
resources:
  ContainersCommon:
    type: ../containers-common.yaml
```

然后容器化模板可以包含 **containers-common.yaml** 模板中的功能和数据。

overcloud.j2.yaml heat 模板包含基于 Jinja2 的代码部分，用于在 **roles_data.yaml** 文件中为每个自定义角色定义服务列表：

```
{{role.name}}Services:
  description: A list of service resources (configured in the heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}
```

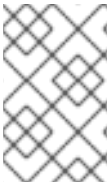
对于默认角色，这将创建以下服务列表参数：**ControllerServices**,**ComputeServices**,**BlockStorageServices**,**ObjectStorageServices**, 和 **CephStorageServices**。

您可以在 **roles_data.yaml** 文件中为每个自定义角色定义默认服务。例如，默认的 Controller 角色包含以下内容：

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
```

```
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

然后，这些服务被定义为 **ControllerServices** 参数的默认列表。



注意

您还可以使用环境文件覆盖服务参数的默认列表。例如，您可以在环境文件中将 **ControllerServices** 定义为 **parameter_default**，以覆盖 **roles_data.yaml** 文件中的服务列表。

2.11. 从角色中添加和删除服务

添加或删除服务的基本方法涉及为节点角色创建默认服务列表的副本，然后添加或删除服务。例如，您可能想要从 Controller 节点中删除 OpenStack Orchestration (heat)。

流程

1. 为 **默认角色** 目录创建自定义副本：

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

2. 编辑 **~/roles/Controller.yaml** 文件，并修改 **ServicesDefault** 参数的服务列表。滚动到 OpenStack 编排服务并删除它们：

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::HeatApi            # Remove this service
    - OS::TripleO::Services::HeatApiCfn         # Remove this service
    - OS::TripleO::Services::HeatApiCloudwatch  # Remove this service
    - OS::TripleO::Services::HeatEngine         # Remove this service
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::NeutronDhcpAgent
```

3. 生成新的 **roles_data** 文件：

```
$ openstack overcloud roles generate -o roles_data-no_heat.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

4. 运行 **openstack overcloud deploy** 命令时包含此新 **roles_data** 文件：

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-no_heat.yaml
```

此命令部署 overcloud，而不在 Controller 节点上安装 OpenStack 编配服务。

注意

您还可以使用自定义环境文件禁用 **roles_data** 文件中的服务。将服务重定向到 disable 到 **OS::Heat::None** 资源。例如：

```
resource_registry:
  OS::TripleO::Services::HeatApi: OS::Heat::None
  OS::TripleO::Services::HeatApiCfn: OS::Heat::None
  OS::TripleO::Services::HeatApiCloudwatch: OS::Heat::None
  OS::TripleO::Services::HeatEngine: OS::Heat::None
```

2.12. 启用禁用的服务

一些服务默认为禁用。这些服务在 **overcloud-resource-registry-puppet.j2.yaml** 文件中注册为 null 操作(**OS::Heat::None**)。例如，块存储备份服务(**cinder-backup**)被禁用：

```
OS::TripleO::Services::CinderBackup: OS::Heat::None
```

若要启用此服务，可包含一个环境文件，它将资源链接到 **puppet/services** 目录中的相应 heat 模板。有些服务在 **environment** 目录中有预定义的环境文件。例如，块存储备份服务使用 **environments/cinder-backup.yaml** 文件，该文件包含以下条目：

流程

1. 在环境文件中添加一个条目，它将 **CinderBackup** 服务链接到包含 **cinder-backup** 配置的 heat 模板：

   ```
   resource_registry:
     OS::TripleO::Services::CinderBackup: ../podman/services/pacemaker/cinder-backup.yaml
   ...
   ```

   此条目覆盖默认的 null 操作资源并启用服务。

2. 在运行 **openstack overcloud deploy** 命令时包含此环境文件：

   ```
   $ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
   templates/environments/cinder-backup.yaml
   ```

第 3 章 使用验证框架

Red Hat OpenStack Platform (RHOSP)包含一个验证框架，可用于验证 undercloud 和 overcloud 的要求和功能。该框架包括两种验证类型：

- 基于 Ansible 的手动验证，您可以通过 **验证** 命令集执行。
- 自动动态验证，在部署过程中执行。

您必须了解您要运行的验证，并跳过与环境无关的验证。例如，部署前验证包括 TLS-everywhere 的测试。如果您不打算为 TLS-everywhere 配置您的环境，则此测试会失败。在 **validation run** 命令中使用 **--validation** 选项来根据您的环境来进一步调整验证。

3.1. 基于 ANSIBLE 的验证

在安装 Red Hat OpenStack Platform (RHOSP) director 的过程中，director 还会从 **openstack-tripleo-validations** 软件包安装一组 playbook。每个 playbook 包含对某些系统要求的测试，以及定义何时运行测试的一系列组：

no-op

运行 *no-op*（无操作）任务的验证，以验证工作流功能正常。这些验证在 undercloud 和 overcloud 上运行。

Prep

检查 undercloud 节点的硬件配置的验证。在运行 **openstack undercloud install** 命令前运行这些验证。

openshift-on-openstack

检查环境是否满足可在 OpenStack 上部署 OpenShift 的要求的验证。

pre-introspection

使用 Ironic 检查程序进行节点内省前要运行的验证。

pre-deployment

在 **openstack overcloud deploy** 命令前要运行的验证。

post-deployment

在 overcloud 部署完成后要运行的验证。

pre-update

在更新前验证 RHOSP 部署的验证。

post-update

在更新后验证 RHOSP 部署的验证。

pre-upgrade

在升级前验证 RHOSP 部署的验证。

升级后

升级后验证 RHOSP 部署的验证。

3.2. 更改验证配置文件

验证配置文件是一个 .ini 文件，您可以编辑该文件来控制验证执行的每个方面以及远程计算机之间的通信。

您可以使用以下方法之一更改默认配置值：

- 编辑默认的 /etc/validations.cfg 文件。
- 复制默认的 **/etc/validations.cfg** 文件，编辑副本，并通过 CLI 使用 **--config** 参数提供副本。如果您创建了您自己的配置文件副本，请使用 **--config** 在每次执行时将 CLI 指向此文件。

默认情况下，验证配置文件的位置为 **/etc/validation.cfg**。

重要

确保正确编辑配置文件或您的验证可能会失败，例如：

- 未检测的验证
- 写入不同位置的回调
- 错误解析的日志

先决条件

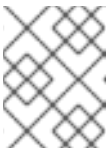
- 您已了解如何验证您的环境。

流程

1. 可选：复制验证配置文件以进行编辑：
   a. 将 **/etc/validation.cfg** 复制到您的主目录中。
   b. 对新配置文件进行所需的编辑。
2. 运行验证命令：

```
$ validation run --config <configuration-file>
```

   - 将 <configuration-file> 替换为您要使用的配置文件的文件路径。



注意

当您运行验证时，输出中的 **Reasons** 列仅限于 79 个字符。要查看验证结果已满，请查看验证日志文件。

3.3. 列出验证

运行 **验证列表** 命令，以列出不同类型的可用验证。

流程

1. source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 运行 **验证列表** 命令：
   - 要列出所有验证，请在没有任何选项的情况下运行命令：

```
$ validation list
```

- 要列出组中的验证，请使用 **--group** 选项运行该命令：

```
$ validation list --group prep
```

注意

如需完整的选项列表，请运行 **验证列表 --help**。

3.4. 运行验证

要运行验证或验证组，请使用 **validation run** 命令。要查看完整的选项列表，请使用 **validation run --help** 命令。

注意

当您运行验证时，输出中的 **Reasons** 列仅限于 79 个字符。要查看验证结果已满，请查看验证日志文件。

流程

1. Source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 验证名为 **tripleo-ansible-inventory.yaml** 的静态清单文件。

```
$ validation run --group pre-introspection -i tripleo-ansible-inventory.yaml
```

注意

您可以在用于独立或 undercloud 部署的 **~/tripleo-deploy/<stack&** gt; 目录中或 overcloud 部署的 **~/overcloud-deploy/<stack&** gt; 目录中找到清单文件。

3. 输入 **验证运行命令** ：

- 要运行单个验证，请输入带 **--validation-name** 选项的命令和验证的名称。例如，要检查每个节点的内存要求，请输入 **--validation check-ram**：

```
$ validation run --validation check-ram
```

要运行多个特定的验证，请使用 **--validation** 选项以及您要运行的验证的逗号分隔列表。有关查看可用验证列表的更多信息，请参阅 列出验证。

- 要运行组中的所有验证，请输入带 **--group** 选项的命令：

```
$ validation run --group prep
```

要查看特定验证的详细输出，请根据报告中特定验证的 UUID 运行验证 历史记录 **get --full** 命令：

```
-
```

```
$ validation history get --full <UUID>
```

3.5. 创建验证

您可以使用 **validation init** 命令创建验证。执行 命令会导致新验证的基本模板。您可以编辑新的验证角色以满足您的要求。

重要

红帽不支持用户创建的验证。

先决条件

- 您已了解如何验证您的环境。
- 您可以访问运行命令的目录。

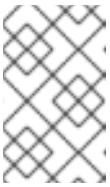
流程

1. 创建验证：

```
$ validation init <my-new-validation>
```

- 将 **<my-new-validation** > 替换为新验证的名称。
  这个命令的执行会导致创建以下目录和子目录：

```
/home/stack/community-validations
├── library
├── lookup_plugins
├── playbooks
└── roles
```



注意

如果您看到错误消息 "The Community Validations 被默认禁用，请确保验证配置文件中将 **enable_community_validations** 参数设置为 **True**。此文件的默认名称和位置为 **/etc/validation.cfg**。

2. 编辑角色以满足您的要求。

其他资源

- 第 3.2 节 “更改验证配置文件”.

3.6. 查看验证历史记录

director 在运行一个验证或一组验证后保存每个验证的结果。使用 **validation history list** 命令查看验证后的结果。

先决条件

- 您已运行了一个验证或一组验证。

流程

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

```
$ source ~/stackrc
```

3. 您可以查看所有验证列表或最新验证：
   - 查看所有验证的列表：

```
$ validation history list
```

   - 使用 **--validation** 选项查看特定验证类型的历史记录：

```
$ validation history get --validation <validation-type>
```

     - 将 *<validation-type >* 替换为验证类型，如 ntp。
4. 查看特定验证 UUID 的日志：

```
$ validation show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

其他资源

- assembly_using-the-validation-framework[使用验证框架]

3.7. 验证框架日志格式

在运行一个验证或一组验证后，director 将每个验证的 JSON 格式日志保存在 **/var/logs/validations** 目录中。您可以手动查看文件，或使用 **验证历史记录 get --full** 命令显示特定验证 UUID 的日志。

每个验证日志文件都遵循特定的格式：

- **<UUID>_<Name>_<Time>**

  UUID
  验证的 Ansible UUID。

  名称
  验证的 Ansible 名称。

  Time
  运行验证时的开始日期和时间。

每个验证日志包含三个主要部分：

- plays
- stats
- validation_output

plays

plays 部分包含有关 director 在验证过程中执行的任务的信息：

play

一个 play 就是一组任务。每个 **play** 部分包含关于这一特定组任务的信息，包括开始和结束时间、持续时间、play 的主机组，以及验证 ID 和路径。

tasks

director 为执行验证运行的个别 Ansible 任务。每个 **tasks** 部分包含一个 **hosts** 部分，其中包含了在每个单独主机上执行的操作以及执行操作的结果。**tasks** 部分还包含一个 **task** 部分，其中包含了任务的持续时间。

stats

stats 部分包含每个主机上所有任务结果的基本摘要，如成功和失败的任务。

validation_output

如果任何任务在验证过程中失败或导致警告消息，则 **validation_output** 会包含该失败或警告的输出。

3.8. 验证框架日志输出格式

验证框架的默认行为是以 JSON 格式保存验证日志。您可以使用 **ANSIBLE_STDOUT_CALLBACK** 环境变量更改日志输出。

要更改验证输出日志格式，请运行验证并包含 **--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** 选项：

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- 将 **<callback>** 替换为 Ansible 输出回调。要查看标准 Ansible 输出回调列表，请运行以下命令：

```
$ ansible-doc -t callback -l
```

验证框架包括以下额外回调：

validation_json

框架将 JSON 格式的验证结果保存为 **/var/logs/validations** 中的日志文件。这是验证框架的默认回调。

validation_stdout

框架在屏幕上显示 JSON 格式的验证结果。

http_json

框架将 JSON 格式的验证结果发送到外部日志记录服务器。您还必须包含此回调的额外环境变量：

HTTP_JSON_SERVER

外部服务器的 URL。

HTTP_JSON_PORT

外部服务器的 API 入口点的端口。8989 中的默认端口。

使用额外的 **--extra-env-vars** 选项设置这些环境变量：

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
  --extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
  --extra-env-vars HTTP_JSON_PORT=8989 \
```

```
--validation check-ram
```

重要

在使用 **http_json** 回调前，必须将 **http_json** 添加到 **ansible.cfg** 文件中的 **callback_whitelist** 参数中：

```
callback_whitelist = http_json
```

3.9. 动态验证

Red Hat OpenStack Platform (RHOSP)在可组合服务的模板中包括动态验证。动态验证在 overcloud 部署过程的关键步骤中验证服务操作状态。

在部署过程中自动运行动态验证。有些动态验证也使用 **openstack-tripleo-validations** 软件包中的角色。

第 4 章 其他内省操作

在某些情况下，您可能想要在标准 overcloud 部署工作流外执行内省。例如，在替换现有未使用节点上的硬件后，您可能希望内省新节点或刷新内省数据。

4.1. 执行单个节点内省

要在可用节点上执行单个内省，请将节点设置为管理模式并执行内省。

流程

1. 将所有节点设置为 **manageable** 状态：

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. 执行内省：

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

内省完成后，节点切换为 **available** 状态。

4.2. 在初始内省后执行节点内省操作

因为 **--provide** 选项的原因，所有节点在初始内省后都进入 **available** 状态。要在初始内省后在所有节点上执行内省，请将节点设置为管理模式并执行内省。

流程

1. 将所有节点设置为 **manageable** 状态

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do
openstack baremetal node manage $node ; done
```

2. 运行批量内省命令：

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

内省完成后，所有节点都会变为 **available** 状态。

4.3. 执行网络内省以查看接口信息

网络内省会从网络交换机获取链路层发现协议 (LLDP) 数据。以下命令可显示某个节点上所有接口的某个 LLDP 信息子集，或显示某个节点和接口的全部信息。这对故障排除非常有用。director 默认会启用 LLDP 数据收集。

流程

1. 要获取节点上的接口列表，请运行以下命令：

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

例如：

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-
79f8b86c7cd9
+-----------+-------------------+----------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs   | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+----------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                  | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                  | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]             | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+----------------------+-------------------+----------------+
```

2. 要查看接口数据和交换机端口信息，请运行以下命令：

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID]
[INTERFACE]
```

例如：

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-
907d-79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------
----------------------------------------------+
| Field                                | Value
|
+--------------------------------------+------------------------------------------------------------------------------
----------------------------------------------+
| interface                            | p2p1
|
| mac                                  | 00:0a:f7:79:93:18
|
| node_ident                           | c89397b7-a326-41a0-907d-79f8b86c7cd9
|
| switch_capabilities_enabled          | [u'Bridge', u'Router']
|
| switch_capabilities_support          | [u'Bridge', u'Router']
|
| switch_chassis_id                    | 64:64:9b:31:12:00
|
| switch_port_autonegotiation_enabled  | True
|
| switch_port_autonegotiation_support  | True
|
| switch_port_description              | ge-0/0/2.0
|
| switch_port_id                       | 507
|
| switch_port_link_aggregation_enabled | False
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type                 | Unknown
```

```
|
| switch_port_mtu                    | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids                 | [101]
|
| switch_port_vlans                    | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities           | None
|
| switch_system_name                   | rhos-compute-node-sw1
|
+--------------------------------------+-----------------------------------------------------------------------
-----------------------------------------------+
```

4.4. 获取硬件内省详细信息

裸机服务 hardware-inspection-extras 功能默认启用，您可以使用它来检索 overcloud 配置的硬件详情。有关 **undercloud.conf** 文件中的 **inspection_extras** 参数的更多信息，请参阅 Director 配置参数。

例如，**numa_topology** 收集程序就是硬件检查额外功能的一部分，包括每个 NUMA 节点的以下信息：

- RAM（单位为 KB）
- 物理 CPU 内核数和同级线程数
- 和 NUMA 节点关联的 NIC

流程

- 要获得以上列出的信息，请使用裸机节点的 UUID 替换 <UUID> 来完成以下命令：

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

以下示例显示获取的裸机节点 NUMA 信息：

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
```

```
{
  "cpu": 2,
  "thread_siblings": [
    10,
    26
  ],
  "numa_node": 1
},
{
  "cpu": 0,
  "thread_siblings": [
    0,
    16
  ],
  "numa_node": 0
},
{
  "cpu": 5,
  "thread_siblings": [
    13,
    29
  ],
  "numa_node": 1
},
{
  "cpu": 7,
  "thread_siblings": [
    15,
    31
  ],
  "numa_node": 1
},
{
  "cpu": 7,
  "thread_siblings": [
    7,
    23
  ],
  "numa_node": 0
},
{
  "cpu": 1,
  "thread_siblings": [
    9,
    25
  ],
  "numa_node": 1
},
{
  "cpu": 6,
  "thread_siblings": [
    6,
    22
  ],
  "numa_node": 0
},
```

```
{
  "cpu": 3,
  "thread_siblings": [
    11,
    27
  ],
  "numa_node": 1
},
{
  "cpu": 5,
  "thread_siblings": [
    5,
    21
  ],
  "numa_node": 0
},
{
  "cpu": 4,
  "thread_siblings": [
    12,
    28
  ],
  "numa_node": 1
},
{
  "cpu": 4,
  "thread_siblings": [
    4,
    20
  ],
  "numa_node": 0
},
{
  "cpu": 0,
  "thread_siblings": [
    8,
    24
  ],
  "numa_node": 1
},
{
  "cpu": 6,
  "thread_siblings": [
    14,
    30
  ],
  "numa_node": 1
},
{
  "cpu": 3,
  "thread_siblings": [
    3,
    19
  ],
  "numa_node": 0
},
```

```
    {
      "cpu": 2,
      "thread_siblings": [
        2,
        18
      ],
      "numa_node": 0
    }
  ],
  "ram": [
    {
      "size_kb": 66980172,
      "numa_node": 0
    },
    {
      "size_kb": 67108864,
      "numa_node": 1
    }
  ],
  "nics": [
    {
      "name": "ens3f1",
      "numa_node": 1
    },
    {
      "name": "ens3f0",
      "numa_node": 1
    },
    {
      "name": "ens2f0",
      "numa_node": 0
    },
    {
      "name": "ens2f1",
      "numa_node": 0
    },
    {
      "name": "ens1f1",
      "numa_node": 0
    },
    {
      "name": "ens1f0",
      "numa_node": 0
    },
    {
      "name": "eno4",
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
```

```
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

第 5 章 自动发现裸机节点

您可以使用 auto-discovery 来注册 overcloud 节点并生成它们的元数据，而无需创建 **instackenv.json** 文件。这种改进可有助于缩短收集节点信息所需时间。例如，如果您使用 auto-discovery，则不核对 IPMI IP 地址，然后创建 **instackenv.json**。

5.1. 启用自动发现

启用并配置裸机自动发现，以便在使用 PXE 引导时自动发现和导入加入置备网络的节点。

流程

1. 在 **undercloud.conf** 文件中启用裸机自动发现：

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

   - **enable_node_discovery** - 启用之后，任何使用 PXE 来引导内省虚拟内存盘的节点都在 Bare Metal 服务 (ironic) 中自动注册。
   - **discovery_default_driver** - 设置用于已发现节点的驱动程序。例如，**ipmi**。

2. 将您的 IPMI 凭证添加到 ironic：

   a. 将您的 IPMI 凭证添加到名为 **ipmi-credentials.json** 的文件。请替换本例中的 **SampleUsername**、**RedactedSecurePassword** 和 **bmc_address** 值，以适应您的环境：

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  }
]
```

3. 将 IPMI 凭证文件导入 ironic：

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

5.2. 测试自动发现

PXE 引导连接到置备网络的节点，以测试裸机自动发现功能。

流程

1. 启动所需节点。

2. 运行 **openstack baremetal node list** 命令。应该看到新节点以 **enrolled** 状态列出：

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. 为各个节点设置资源类：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. 为各个节点配置内核和 ramdisk：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. 将所有节点设置为 available：

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node provide $NODE ; done
```

5.3. 使用规则发现不同供应商的硬件

如果拥有异构硬件环境，您可以使用内省规则来分配凭证和远程管理凭证。例如，您可能需要单独的发现规则来处理使用 DRAC 的 Dell 节点。

流程

1. 创建名为 **dell-drac-rules.json** 并包含以下内容的文件：

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true},
      {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
       "value": "Dell Inc."}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
```

```
            "value": "SampleUsername"},
           {"action": "set-attribute", "path": "driver_info/ipmi_password",
            "value": "RedactedSecurePassword"},
           {"action": "set-attribute", "path": "driver_info/ipmi_address",
            "value": "{data[inventory][bmc_address]}"}
       ]
   },
   {
       "description": "Set the vendor driver for Dell hardware",
       "conditions": [
           {"op": "eq", "field": "data://auto_discovered", "value": true},
           {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
            "value": "Dell Inc."}
       ],
       "actions": [
           {"action": "set-attribute", "path": "driver", "value": "idrac"},
           {"action": "set-attribute", "path": "driver_info/drac_username",
            "value": "SampleUsername"},
           {"action": "set-attribute", "path": "driver_info/drac_password",
            "value": "RedactedSecurePassword"},
           {"action": "set-attribute", "path": "driver_info/drac_address",
            "value": "{data[inventory][bmc_address]}"}
       ]
   }
]
```

- 请替换此例中的用户名和密码值以适合您的环境：

2. 将规则导入 ironic：

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

第 6 章 配置自动配置集标记

内省操作会执行一系列的基准数据测试，director 将保存这些测试数据。director 保存这些测试中的数据。可使用多种方式创建使用此数据的策略集：

- 策略可以识别性能不佳或不稳定的节点，并隔离这些节点，使其在 overcloud 中使用。
- 这些策略可定义是否将节点自动标记到特定配置集。

6.1. 策略文件语法

策略文件使用 JSON 格式，它包括了一组规则。每个规则都定义一个 description、一个 condition 和一个 action。**description** 是规则的纯文本描述，**condition** 使用键值模式定义一个评估，**action** 是条件的执行。

Description

description 是规则的纯文本描述。

例如：

```
"description": "A new rule for my node tagging policy"
```

Conditions

condition 就是使用以下键-值来定义评估：

field

定义要评估的字段：

- **memory_mb** - 节点的内存大小 (MB)。
- **cpus** - 节点 CPU 的总线程数。
- **cpu_arch** - 节点 CPU 的架构。
- **local_gb** - 节点根磁盘的总存储空间。

op

指定测试所使用的操作。这包括如下属性：

- **eq** - 等于
- **ne** - 不等于
- **lt** - 少于
- **gt** - 多于
- **le** - 少于或等于
- **ge** - 多于或等于
- **in-net** - 检查一个 IP 地址是否在指定的网络中
- **matches** - 需要完全和提供的正则表达式相匹配

- **contains** - 需要一个包括和提供的正则表达式相匹配的值；
- **is-empty** - 检查该**字段**是否为空

invert
: 一个布尔值，用来指定是否对检查结果进行反向处理。

multiple
: 在存在多个结果的情况下，定义使用的测试。此参数包括如下属性：

  - **any** - 只需要任何一个结果匹配
  - **all** - 需要所有结果都匹配
  - **first** - 需要第一个结果匹配

value
: 测试中的值。如果项和操作结果为这个值，则条件返回为一个"true"的结果。否则，条件返回 false 的结果。

例如：

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

如果条件为 **true**，策略将执行一个操作。此操作使用 **action** 密钥和其他密钥，具体取决于 **action** 的值：

- **fail** - 使内省失败。需要一个 **message** 参数来包括失败的信息。
- **set-attribute** - 在一个 ironic 节点上设置一个属性。需要一个 **path** 项，它是到一个 ironic 属性（如 **/driver_info/ipmi_address**）的路径，以及一个 **value** 值。
- **set-capability** - 在一个 ironic 节点上设置一个能力。需要 **name** 和 **value** 字段，它们是新能力的名称和值。这将替换这个功能的现有值。例如，使用它来定义节点配置集。
- **extend-attribute** - 与 **set-attribute** 相似，只是在存在相同能力时把这个值附加到当前的值后面。如果同时使用了 **unique** 参数，则在相同值已存在时不进行任何操作。

例如：

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

6.2. 策略文件示例

以下是一个包含内省规则的 JSON 文件示例（**rules.json**）：

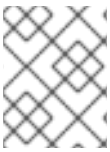
```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
```

```
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

这个示例包括 3 个规则：

- 如果内存低于 4096 MiB，内省失败。如果您想将某些节点排除在云之外，则可应用这些类型的规则。
- 硬盘容量大于或等于 1 TiB 的节点会被无条件地分配 swift-storage 配置集。
- 硬盘容量在 1 TiB 和 40 GiB 间的节点可以作为 Compute 节点或 Controller 节点。您可以分配两个能力（**compute_profile** 和 **control_profile**），使 **openstack overcloud profiles match** 命令稍后可以作出最终选择。要使此过程成功，必须删除现有配置集能力，否则现有配置集能力具有优先级。

配置集匹配规则不更改任何其他节点。



注意

使用内省规则分配**配置集**总会覆盖存在的值。但是，对于已经具有配置集能力的节点，会忽略 **[PROFILE]_profile** 能力。

6.3. 将策略文件导入到 DIRECTOR

要应用您在策略 JSON 文件中定义的策略规则，您必须将策略文件导入到 director。

流程

1. 将策略文件导入 director：

```
$ openstack baremetal introspection rule import <policy_file>
```

   - 将 **<policy_file** > 替换为您的策略规则文件的名称，例如 **rules.json**。

2. 运行内省进程：

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

3. 检索策略应用到的节点的 UUID：

```
$ openstack baremetal node list
```

4. 确认节点已被分配了策略规则文件中定义的配置集：

```
$ openstack baremetal node show <node_uuid>
```

5. 如果您在内省规则中出现错误，则删除所有规则：

```
$ openstack baremetal introspection rule purge
```

第 7 章 自定义容器镜像

Red Hat OpenStack Platform (RHOSP)服务在容器中运行，因此要部署必须获取容器镜像的 RHOSP 服务。您可以生成和自定义为 RHOSP 部署准备容器镜像的环境文件。

7.1. 为 DIRECTOR 安装准备容器镜像

红帽支持使用以下方式为您的 overcloud 管理容器镜像：

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

undercloud 安装需要一个环境文件来确定从何处获取容器镜像以及如何存储它们。准备 director 安装时，您可以生成默认的容器镜像准备文件。您可以自定义默认容器镜像准备文件。

7.1.1. 容器镜像准备参数

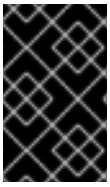
用于准备您的容器的默认文件 (**containers-prepare-parameter.yaml**) 包含 **ContainerImagePrepare** heat 参数。此参数定义一个用于准备一系列镜像的策略列表：

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

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

参数	描述
excludes	用于从策略中排除镜像名称的正则表达式列表。
includes	用于在策略中包含的正则表达式列表。至少一个镜像名称必须与现有镜像匹配。如果指定了 **includes**，将忽略所有 **excludes**。
modify_append_tag	要附加到目标镜像标签的字符串。例如，如果您使用标签 17.1.0-5.161 拉取镜像并将 **modify_append_tag** 设置为 **-hotfix**，director 会将最终镜像标记为 17.1.0-5.161-hotfix。
modify_only_with_labels	过滤想要修改的镜像的镜像标签字典。如果镜像与定义的标签匹配，则 director 将该镜像包括在修改过程中。

参数	描述
modify_role	在上传期间但在将镜像推送到目标 registry 之前运行的 ansible 角色名称字符串。
modify_vars	要传递给 **modify_role** 的变量的字典。
push_destination	定义用于在上传过程中要将镜像推送到的 registry 的命名空间。 ● 如果设为 **true**，则使用主机名将 **push_destination** 设置为 undercloud registry 命名空间，这是建议的方法。 ● 如果设置为 **false**，则不会推送到本地 registry，节点直接从源拉取镜像。 ● 如果设置为自定义值，则 director 将镜像推送到外部本地 registry。 当直接从 Red Hat Container Catalog 拉取镜像时，如果在生产环境中将此参数设置为 **false**，则所有 overcloud 节点都将通过外部连接同时从 Red Hat Container Catalog 拉取镜像，这可能会导致出现带宽问题。仅使用 **false** 直接从托管容器镜像的 Red Hat Satellite Server 拉取。 如果 **push_destination** 参数设置为 **false** 或未定义，且远程 registry 需要身份验证，请将 **ContainerImageRegistryLogin** 参数设置为 **true**，并使用 **ContainerImageRegistryCredentials** 参数包含凭据。
pull_source	从中拉取原始容器镜像的源 registry 。
set	用于定义从何处获取初始镜像的 **key: value** 定义的字典。
tag_from_label	使用指定容器镜像元数据标签的值为每个镜像创建标签，并拉取该标记的镜像。例如，如果设置了 **tag_from_label: {version}-{release}**，则 director 会使用 **version** 和 **release** 标签来构造新标签。对于一个容器，**version** 可能会设置为 17.1.0，**release** 可能会设置为 **5.161**，这会导致标签 17.1.0-5.161。仅当未在 **set** 字典中定义 **tag** 时，director 才使用此参数。



重要

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

set 参数接受一组 **key: value** 定义：

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

注意

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

7.1.2. 容器镜像标记准则

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

version

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

release

对应于版本流中特定容器镜像版本的发行版本。

例如，如果 Red Hat OpenStack Platform 的最新版本为 17.1.0，容器镜像的发行版本为 **5.161**，则生成的容器镜像标签为 17.1.0-5.161。

Red Hat Container Registry 还使用一组主要和次要 **version** 标签，链接到该容器镜像版本的最新发行版本。例如，17.1 和 17.1.0 链接到 17.1.0 容器流中的最新 **版本**。如果出现 17.1 的新次要版本，17.1 标签链接到 **新次要发行版本** 流的最新发行版本，而 17.1.0 标签则继续链接到 17.1.0 流中 **的最新发行版本**。

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

- **tag** 的默认值是您的 OpenStack Platform 版本的主要版本。对于这个版本，它是 17.1。这始终对应于最新的次要版本和发行版本。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1
      ...
```

- 要更改为 OpenStack Platform 容器镜像的特定次要版本，请将标签设置为次要版本。例如，若要更改为 17.1.2，请将 **tag** 设置为 17.1.2。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1.2
      ...
```

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

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

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
```

```
    ...
    # tag: 17.1
    ...
  tag_from_label: '{version}-{release}'
```

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

```
"Labels": {
  "release": "5.161",
  "version": "17.1.0",
  ...
}
```

- **tag_from_label** 的默认值为 **{version}-{release}**，对应于每个容器镜像的版本和发行版本元数据标签。例如，如果容器镜像为 **版本** 设置了 17.1.0，并且为 **发行版本** 设置了 5.161，则生成的容器镜像标签为 17.1.0-5.161。

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

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

7.1.3. 排除 Ceph Storage 容器镜像

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

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

流程

1. 编辑 **containers-prepare-parameter.yaml** 文件并添加 **ceph_images: false** 参数。以下是该文件的示例，参数粗体显示：

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: {version}-{release}
    set:
      name_prefix: rhosp17-openstack-
      name_suffix: ''
      tag: 17.1_20231214.1
      rhel_containers: false
      neutron_driver: ovn
      ceph_images: false
    push_destination: true
```

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

3. 创建新的容器镜像文件，以用于 overcloud 部署：

sudo openstack tripleo container image prepare -e containers-prepare-parameter.yaml --output-env-file <new_container_images_file>

- 将 **<new_container_images_file>** 替换为包含新参数的输出文件。

4. 将新容器镜像文件添加到 overcloud 部署环境文件列表中。

7.1.4. 准备期间修改镜像

可在准备镜像期间修改镜像，然后立即使用修改的镜像部署 overcloud。

注意

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

修改镜像的情况包括：

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

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

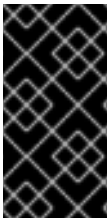
Ansible 角色 **tripleo-modify-image** 与所需的角色接口相一致，并提供修改用例必需的行为。使用 **ContainerImagePrepare** 参数中与修改相关的键控制修改：

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

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

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

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

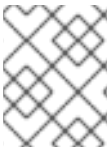


重要

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

7.1.5. 更新容器镜像的现有软件包

您可以为 Red Hat OpenStack Platform (RHOSP)容器更新容器镜像中的现有软件包。



注意

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

流程

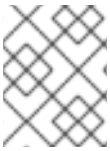
1. 下载 RPM 软件包以安装到容器镜像上。
2. 编辑 **containers-prepare-parameter.yaml** 文件，以更新容器镜像上的所有软件包：

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

3. 保存 **containers-prepare-parameter.yaml** 文件。
4. 运行 **openstack overcloud deploy** 命令时，包含 **containers-prepare-parameter.yaml** 文件。

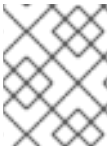
7.1.6. 将额外的 RPM 文件安装到容器镜像中

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



注意

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



注意

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

流程

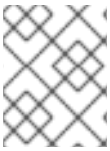
- 以下示例 **ContainerImagePrepare** 条目仅在 **nova-compute** 镜像上安装一些热修复软件包：

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
```

```
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

7.1.7. 通过自定义 Dockerfile 修改容器镜像

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



注意

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

步骤

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

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

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

```
FROM registry.redhat.io/rhosp-rhel9/openstack-nova-compute:latest

USER "root"

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

USER "nova"
```

7.1.8. 为容器镜像准备 Satellite 服务器

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

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

注意

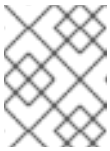
此过程需要身份验证凭据以从 **registry.redhat.io** 访问容器镜像。红帽建议创建一个 registry 服务帐户并使用这些凭据访问 **registry.redhat.io** 内容，而不使用您的个人用户凭据。有关更多信息，请参阅"红帽容器 registry 身份验证"。

步骤

1. 创建所有容器镜像的列表：

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel9" --format="{{ .Name }}" |
sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep
<ceph_dashboard_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep <ceph_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/openshift4" | grep ose-prometheus
```

- 将 **<ceph_dashboard_image_file>** 替换为部署使用的 Red Hat Ceph Storage 版本的镜像文件的名称：
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**
- 将 **<ceph_image_file>** 替换为部署使用的 Red Hat Ceph Storage 版本的镜像文件的名称：
  - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-rhel9**



注意

openstack-ovn-bgp-agent 镜像位于 **registry.redhat.io/rhosp-rhel9/openstack-ovn-bgp-agent-rhel9:17.1**。

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

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.12
registry.redhat.io/openshift4/ose-prometheus:v4.12
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.12
```

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

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

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

该定制产品将会包含您的镜像。

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

```
$ while read IMAGE; do \
```

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

5. 添加 Ceph Storage 容器镜像：

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

- 将 **<ceph_image_file** > 替换为部署使用的 Red Hat Ceph Storage 版本的镜像文件的名称：
  - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-rhel9**

注意

如果要安装 Ceph 仪表板，请在 **hammer repository create** 命令中包含 **--name <ceph_dashboard_image_name** > ：

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

- 将 **<ceph_dashboard_image_file** > 替换为部署使用的 Red Hat Ceph Storage 版本的镜像文件的名称：
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**

6. 同步容器镜像：

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

等待 Satellite 服务器完成同步。

注意

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

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

8. 检查 **base** 镜像的可用标签：

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

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

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

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

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

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

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

- **namespace** - Satellite 服务器上 registry 的 URL。

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

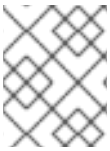
  - 如果您使用了内容视图，则前缀的结构为 **[组织]-[环境]-[内容视图]-[产品]-**。例如：**acme-production-myosp17-osp_containers-**。

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

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

以下示例环境文件包含特定于 Satellite 的参数：

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp17_1-osp_containers-rhceph-6
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp17_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:5000
      neutron_driver: null
      tag: '17.1'
      ...
```



注意

要使用存储在 Red Hat Satellite Server 上的特定容器镜像版本，请将 **标签** 键值对 **设置为集合** 字典中的特定版本。例如，要使用 17.1.2 镜像流，请在 **set** 字典中设置 **tag: 17.1.2**。

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

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

7.1.9. 部署供应商插件

要将一些第三方硬件用作块存储后端，您必须部署供应商插件。以下示例演示了如何部署厂商插件以使用 Dell EMC 硬件作为块存储后端。

流程

1. 为您的 overcloud 创建新的容器镜像文件：

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

2. 编辑 **containers-prepare-parameter-dellemc.yaml** 文件。

3. 在主 Red Hat OpenStack Platform 容器镜像的策略中添加一个 **exclude** 参数。使用此参数排除厂商容器镜像将替换的容器镜像。在示例中，容器镜像是 **cinder-volume** 镜像：

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

```
    set:
      namespace: registry.redhat.io/rhosp-rhel9
      name_prefix: openstack-
      name_suffix: ''
      tag: 17.1
      ...
    tag_from_label: "{version}-{release}"
```

4. 在 **ContainerImagePrepare** 参数中添加新策略，其中包含厂商插件的替代容器镜像：

```
parameter_defaults:
  ContainerImagePrepare:
    ...
    - push_destination: true
      includes:
        - cinder-volume
      set:
        namespace: registry.connect.redhat.com/dellemc
        name_prefix: openstack-
        name_suffix: -dellemc-rhosp16
        tag: 16.2-2
        ...
```

5. 将 registry.connect.redhat.com registry 的身份验证详情添加到 **ContainerImageRegistryCredentials** 参数中：

```
parameter_defaults:
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      [service account username]: [service account password]
    registry.connect.redhat.com:
      [service account username]: [service account password]
```

6. 保存 **containers-prepare-parameter-dellemc.yaml** 文件。

7. 包含 **containers-prepare-parameter-dellemc.yaml** 文件以及任何部署命令，如 **openstack overcloud deploy**：

```
$ openstack overcloud deploy --templates
    ...
    -e containers-prepare-parameter-dellemc.yaml
    ...
```

当 director 部署 overcloud 时，overcloud 将使用厂商容器镜像，而不是标准容器镜像。

重要

containers-prepare-parameter-dellemc.yaml 文件取代了 overcloud 部署中的标准 **containers-prepare-parameter.yaml** 文件。不要在 overcloud 部署中包含标准 **containers-prepare-parameter.yaml** 文件。为您的 undercloud 安装和更新保留标准 **containers-prepare-parameter.yaml** 文件。

7.2. 执行高级容器镜像管理

默认容器镜像配置适合大多数环境。在某些情况下，您的容器镜像配置可能需要一些自定义，如版本固定。

7.2.1. 为 undercloud 固定容器镜像

在某些情况下，您可能需要 undercloud 的一组特定容器镜像版本。在这种情况下，您必须将镜像固定到特定的版本。要固定镜像，您必须生成和修改容器配置文件，然后将 undercloud 角色数据和容器配置文件结合，以生成包含服务到容器镜像映射的环境文件。在 **undercloud.conf** 文件的 **custom_env_files** 参数中包含此环境文件。

步骤

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 使用 **--output-env-file** 选项运行 **openstack tripleo container image prepare default** 命令，生成包含默认镜像配置的文件：

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

3. 根据您的环境要求，修改 **undercloud-container-image-prepare.yaml** 文件。
   a. 移除 **tag:** 参数，以便 director 可以使用 **tag_from_label:** 参数。director 使用此参数来标识每个容器镜像的最新版本，拉取每个镜像，并在 director 中的容器 registry 上标记每个镜像。
   b. 移除 undercloud 的 Ceph 标签。
   c. 确保 **neutron_driver:** 参数为空。不要将此参数设置为 **OVN**，因为 undercloud 不支持 OVN。
   d. 包含容器镜像 registry 凭据：

```
ContainerImageRegistryCredentials:
  registry.redhat.io:
    myser: 'p@55w0rd!'
```

注意

您无法将容器镜像推送到新 undercloud 上的 undercloud registry，因为 **image-serve** registry 尚未安装。您必须将 **push_destination** 值设置为 **false**，或使用自定义值直接从源拉取镜像。如需更多信息，请参阅 容器镜像准备参数。

4. 生成新的容器镜像配置文件，该文件结合使用 undercloud 角色文件和自定义 **undercloud-container-image-prepare.yaml** 文件：

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

undercloud-container-images.yaml 文件是一个环境文件，包含服务参数到容器镜像的映射。例如，OpenStack Identity (keystone) 使用 **ContainerKeystoneImage** 参数来定义其容器镜像：

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
```

请注意，容器镜像标签与 **{version}-{release}** 格式匹配。

5. 将 **undercloud-container-images.yaml** 文件包含在 **undercloud.conf** 文件的 **custom_env_files** 参数中。在运行 undercloud 安装时，undercloud 服务使用来自此文件的固定容器镜像映射。

7.2.2. 为 overcloud 固定容器镜像

在某些情况下，您可能需要 overcloud 的一组特定容器镜像版本。在这种情况下，您必须将镜像固定到特定的版本。若要固定镜像，您必须创建 **containers-prepare-parameter.yaml** 文件，使用此文件将容器镜像拉取到 undercloud registry，并生成包含固定镜像列表的环境文件。

例如，**containers-prepare-parameter.yaml** 文件可能包含以下内容：

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel9
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

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

ContainerImagePrepare 参数包含单个规则 **set**。此规则 **set** 不得包含 **tag** 参数，且必须依赖 **tag_from_label** 参数来标识每个容器镜像的最新版本和发行版本。director 使用此规则 **set** 来标识每个容器镜像的最新版本，拉取每个镜像，并在 director 中的容器 registry 上标记每个镜像。

步骤

1. 运行 **openstack tripleo container image prepare** 命令，该命令从 **containers-prepare-parameter.yaml** 文件中定义的源中拉取所有镜像。包含 **--output-env-file** 以指定将包含固定容器镜像列表的输出文件：

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

overcloud-images.yaml 文件是一个环境文件，包含服务参数到容器镜像映射。例如，OpenStack Identity (keystone) 使用 **ContainerKeystoneImage** 参数来定义其容器镜像：

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
```

请注意，容器镜像标签与 **{version}-{release}** 格式匹配。

2. 在运行 **openstack overcloud deploy** 命令时，以特定顺序将 **containers-prepare-parameter.yaml** 和 **overcloud-images.yaml** 文件包含在环境文件集合中：

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/containers-prepare-parameter.yaml \
    -e /home/stack/overcloud-images.yaml \
    ...
```

overcloud 服务使用 **overcloud-images.yaml** 文件中列出的固定镜像。

第 8 章 为 RED HAT OPENSTACK PLATFORM 环境自定义网络

您可以为 Red Hat OpenStack Platform (RHOSP)环境自定义 undercloud 和 overcloud 物理网络。

8.1. 自定义 UNDERCLOUD 网络

您可以自定义 undercloud 网络配置，以使用特定的网络功能安装 undercloud。如果您有 IPv6 节点和基础架构，您还可以将 undercloud 和 provisioning 网络配置为使用 IPv6 而不是 IPv4。

8.1.1. 配置 undercloud 网络接口

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

步骤

1. 登录 undercloud 主机。
2. 创建新文件 **undercloud-os-net-config.yaml**，并包含所需的网络配置。
   在 **addresses** 部分中，包含 **local_ip**，如 **172.20.0.1/26**。如果在 undercloud 中启用了 TLS，还必须包含 **undercloud_public_host**，如 **172.20.0.2/32** 和 **undercloud_admin_host**，如 **172.20.0.3/32**。

   下面是一个示例：

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - type: interface
    name: nic2
```

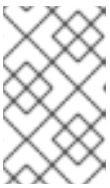
   要为特定接口创建网络绑定，请使用以下示例：

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
```

```
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
      name: nic2
    - type: interface
      name: nic3
```

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

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



注意

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

4. 安装 undercloud。

验证

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

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - type: interface
    name: nic2
```

8.1.2. 为使用 IPv6 的裸机置备配置 undercloud

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

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

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

先决条件

- undercloud 上的 IPv6 地址。如需更多信息，请参阅 *Overcloud 的 IPv6 网络*指南中的在 undercloud 上配置 IPv6 地址。

流程

1. 打开 **undercloud.conf** 文件。
2. 将 IPv6 地址模式指定为 stateless 或 stateful ：

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

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

注意

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

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

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

4. 为裸机节点设置默认网络接口：

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

5. 指定 undercloud 是否应该在 provisioning 网络中创建一个路由器：

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

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

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

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

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

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

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

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

8. 为置备节点配置 DHCP 分配范围：

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

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

9. 可选：配置将流量转发到外部网络的网关：

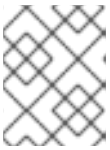
```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

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

10. 将 DHCP 范围配置为在检查过程中使用：

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

- 将 < **ipv6_address_inspection_** start> 替换为在检查过程中要使用的网络范围的 IPv6 地址。
- 将 < **ipv6_address_inspection_** end> 替换为在检查过程中要使用的网络范围末尾的 IPv6 地址。



注意

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

11. 为子网配置 IPv6 名称服务器：

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

- 将 **<ipv6_dns>** 替换为特定于子网的 DNS 名称服务器。

12. 使用 **virt-customize** 工具修改 overcloud 镜像，以禁用 **cloud-init** 网络配置。有关更多信息，请参阅红帽知识库解决方案 使用 virt-customize 修改 Red Hat Linux OpenStack Platform Overcloud 镜像。

8.2. 自定义 OVERCLOUD 网络

您可以为 overcloud 自定义物理网络的配置。例如，您可以使用 Jinja2 ansible 格式的 NIC 模板文件为网络接口控制器(NIC)创建配置文件，即 **j2**。

8.2.1. 定义自定义网络接口模板

您可以创建一组自定义网络接口模板，为 overcloud 环境中的每个节点定义 NIC 布局。overcloud 核心模板集合包含一组用于不同用例的默认 NIC 布局。您可以使用带有 **.j2.yaml** 扩展的 Jinja2 格式文件来创建自定义 NIC 模板。director 在部署期间将 Jinja2 文件转换为 YAML 格式。

然后，您可以将 **overcloud-baremetal-deploy.yaml** 节点定义文件中的 **network_config** 属性设置为自定义 NIC 模板，以便为特定节点置备网络。有关更多信息，请参阅为 overcloud 置备裸机节点。

8.2.1.1. 创建自定义 NIC 模板

创建一个 NIC 模板，为 overcloud 环境中的每个节点自定义 NIC 布局。

流程

1. 将您需要的示例网络配置模板从 **/usr/share/ansible/roles/tripleo_network_config/templates/** 复制到环境文件目录中：

```
$ cp /usr/share/ansible/roles/tripleo_network_config/templates/<sample_NIC_template>
/home/stack/templates/<NIC_template>
```

   - 将 **<sample_NIC_template** > 替换为您要复制的示例 NIC 模板的名称，如 **single_nic_vlans/single_nic_vlans.j2**。
   - 将 **<NIC_template** > 替换为自定义 NIC 模板文件的名称，如 **single_nic_vlans.j2**。

2. 更新自定义 NIC 模板中的网络配置，以匹配 overcloud 网络环境的要求。有关可用于配置 NIC 模板的属性的详情，请参考 网络接口配置选项。有关 NIC 模板示例，请参阅自定义 网络接口示例。

3. 创建或更新现有环境文件以启用自定义 NIC 配置模板：

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
  CephStorageNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans_storage.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
```

4. 如果您的 overcloud 使用默认的内部负载均衡，请在环境文件中添加以下配置来为 Redis 和 OVNDB 分配可预测的虚拟 IP：

```
parameter_defaults:
  RedisVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
  OVNDBsVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
```

   - 将 **<vip_address** > 替换为分配池范围之外的 IP 地址。

8.2.1.2. 网络接口配置选项

使用下表来了解用于配置网络接口的可用选项。

interface

定义一个网络接口。网络接口名称使用实际接口名称(**eth0**、**eth1**、 **enp0s25**)或一组编号的接口(**nic1**、**nic** 2、**nic3**)。 **当使用编号接口（如 nic1 和 nic2 ）时，角色中的主机网络接口不必完全相同，而不是命名的接口，如 eth0 和 eno2。例如，一个主机可能有一个接口 em1 和 em2，而另一个主机有 eno1 和 eno2，但您可以将两个主机的 NIC 指代为 nic1 和 nic2。**

编号的接口的顺序与命名的网络接口类型的顺序对应：

- **ethX 接口，如 eth 0、eth1 等。它们通常位于板上接口。**
- **enoX 接口，如 eno0、eno1 等。它们通常位于板上接口。**

- **enX 接口，按字母数字排序，如 enp3s 0、enp3s1、 ens3 等。这些通常是附加组件接口。**

编号的 NIC 方案仅包含实时接口，例如，如果接口已将电缆附加到交换机。如果您有一些具有四个接口的主机以及一些接口，请使用 nic1 到 nic4，并且每个主机上仅附加四个电缆。

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

表 8.1. 接口选项

选项	默认	描述
name		接口的名称。网络接口名称使用实际接口名称(**eth0、eth1、enp0s25**)或一组编号的接口(**nic1**、**nic** 2、**nic3)。**
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给接口的 IP 地址列表。
Routes		分配给接口的路由列表。如需更多信息，请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
primary	False	将接口定义为主接口。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	None	要传递给 DHCP 客户端的参数。
dns_servers	None	您要用于接口的 DNS 服务器列表。
ethtool_opts		将这个选项设置为 **"rx-flow-hash udp4 sdfn"**，以便在某些 NIC 上使用 VXLAN 时提高吞吐量。

vlan

定义 VLAN。使用从 parameters 部分传递的 VLAN ID 和子网。

例如：

```
- type: vlan
  device: nic{{ loop.index + 1 }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask:
    {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

表 8.2. VLAN 选项

选项	默认	描述
vlan_id		VLAN ID。
device		用于附加 VLAN 的父设备。当 VLAN 不是 OVS 网桥的成员时，请使用此参数。例如，使用此参数将 VLAN 附加到绑定接口设备。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给 VLAN 的 IP 地址列表。
Routes		分配给 VLAN 的路由列表。更多信息请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
primary	False	将 VLAN 定义为主接口。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	无	要传递给 DHCP 客户端的参数。
dns_servers	无	用于 VLAN 的 DNS 服务器列表。

ovs_bond

在 Open vSwitch 中定义将两个或多个 接口 接合在一起的绑定。这有助于冗余并增加带宽。

例如：

```
members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
      primary: true
    - type: interface
      name: nic3
      mtu: {{ min_viable_mtu }}
```

表 8.3. ovs_bond options

选项	默认	描述
name		绑定的名称。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给绑定的 IP 地址列表。
Routes		分配给绑定的路由列表。更多信息请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
primary	False	将接口定义为主接口。
成员		要在绑定中使用的一系列接口对象。
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。

选项	默认	描述
defroute	True	使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时应用。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	无	要传递给 DHCP 客户端的参数。
dns_servers	无	要用于绑定的 DNS 服务器列表。

ovs_bridge

在 Open vSwitch 中定义一个网桥，将多个 interface, ovs_bond, 和 vlan 对象连接在一起。

网络接口类型 ovs_bridge 取 参数名称。

注意

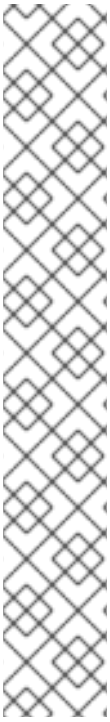
如果您有多个网桥，则必须使用除接受默认名称 bridge_name 以外的不同网桥名称。如果不使用不同的名称，则在聚合阶段，则两个网络绑定将放置在同一网桥上。

如果您要为外部 tripleo 网络定义 OVS 网桥，则保留 bridge_name 和 interface_name 值作为部署框架，会自动将这些值替换为外部网桥名称和外部接口名称。

例如：

```
- type: ovs_bridge
  name: br-bond
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bound_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
```

```
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu }}
```



注意

OVS 网桥连接到 Networking 服务(neutron)服务器来获取配置数据。如果 OpenStack 控制流量（通常是 Control Plane 和 Internal API 网络）放置在 OVS 网桥上，那么每当您升级 OVS 时，与 neutron 服务器的连接将会丢失，或者 OVS 网桥由 admin 用户或进程重启。这会导致停机。如果在这些情况下无法接受停机时间，您必须将 Control 组网络放在单独的接口或绑定中，而不是 OVS 网桥：

- **当您将 Internal API 网络放在 provisioning 接口和 OVS 网桥到第二个接口上时，您可以达到最小设置。**
- **要实现绑定，至少需要两个绑定（我们的网络接口）。将控制组放在 Linux 绑定(Linux 网桥)上。如果交换机不支持 LACP 回退到单一接口进行 PXE 引导，那么这个解决方案至少需要 5 个 NIC。**

表 8.4. ovs_bridge options

选项	默认	描述
name		网桥的名称。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给网桥的 IP 地址列表。
Routes		分配给网桥的路由列表。更多信息请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
成员		要在网桥中使用的一系列接口、VLAN 和绑定对象。
ovs_options		在创建网桥时传递给 OVS 的一组选项。
ovs_extra		一组选项，用于在网桥的网络配置文件中设置为 OVS_EXTRA 参数。

选项	默认	描述
defroute	True	使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时应用。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	无	要传递给 DHCP 客户端的参数。
dns_servers	无	要用于网桥的 DNS 服务器列表。

linux_bond

定义将两个或多个 接口 接合在一起的 Linux 绑定。这有助于冗余并增加带宽。确保在 bonding_options 参数中包含基于内核的绑定选项。

例如：

```
- type: linux_bond
  name: bond1
  mtu: {{ min_viable_mtu }}
  bonding_options: "mode=802.3ad lacp_rate=fast updelay=1000 miimon=100
xmit_hash_policy=layer3+4"
  members:
    type: interface
    name: ens1f0
    mtu: {{ min_viable_mtu }}
    primary: true
  type: interface
    name: ens1f1
    mtu: {{ min_viable_mtu }}
```

表 8.5. linux_bond options

选项	默认	描述
name		绑定的名称。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给绑定的 IP 地址列表。

选项	默认	描述
Routes		分配给绑定的路由列表。请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
primary	False	将接口定义为主接口。
成员		要在绑定中使用的一系列接口对象。
bonding_options		创建绑定时的一组选项。
defroute	True	使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时应用。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	无	要传递给 DHCP 客户端的参数。
dns_servers	无	要用于绑定的 DNS 服务器列表。

linux_bridge

定义一个 Linux 网桥，它将多个 interface, linux_bond, 和 vlan 对象连接在一起。外部网桥也为参数使用两个特殊值：

- **bridge_name，它被外部网桥名称替代。**
- **interface_name，它被外部接口替换。**

例如：

```
- type: linux_bridge
    name: bridge_name
    mtu:
      get_attr: [MinViableMtu, value]
    use_dhcp: false
    dns_servers:
```

```
      get_param: DnsServers
    domain:
      get_param: DnsSearchDomains
    addresses:
    - ip_netmask:
        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
    routes:
      list_concat_unique:
        - get_param: ControlPlaneStaticRoutes
```

表 8.6. linux_bridge options

选项	默认	描述
name		网桥的名称。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。
addresses		分配给网桥的 IP 地址列表。
Routes		分配给网桥的路由列表。更多信息请参阅 Routes。
mtu	1500	连接的最大传输单元(MTU)。
成员		要在网桥中使用的一系列接口、VLAN 和绑定对象。
defroute	True	使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时应用。
persist_mapping	False	编写设备别名配置而不是系统名称。
dhclient_args	无	要传递给 DHCP 客户端的参数。
dns_servers	无	要用于网桥的 DNS 服务器列表。

Routes

定义应用到网络接口、VLAN、网桥或绑定的路由列表。

例如：

```
- type: linux_bridge
  name: bridge_name
  ...
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
```

选项	默认	描述
ip_netmask	无	目标网络的 IP 和子网掩码。
default	False	将此路由设置为默认路由。等同于设置 **ip_netmask: 0.0.0.0/0**。
next_hop	无	用于访问目标网络的路由器的 IP 地址。

8.2.1.3. 自定义网络接口示例

以下示例演示了如何自定义网络接口模板。

分隔控制组和 OVS 网桥示例

以下示例 Controller 节点 NIC 模板配置与 OVS 网桥分开的控制组。该模板使用五个网络接口，并为编号的接口分配多个标记的 VLAN 设备。该模板在 nic4 和 nic5 上创建 OVS 网桥。

```
network_config:
- type: interface
  name: nic1
  mtu: {{ ctlplane_mtu }}
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
```

```
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
{% for network in role_networks if not network.startswith('Tenant') %}
- type: vlan
  device: bond_api
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: linux_bond
    name: bond-data
    mtu: {{ min_viable_mtu_dataplane }}
    bonding_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic4
      mtu: {{ min_viable_mtu_dataplane }}
      primary: true
    - type: interface
      name: nic5
      mtu: {{ min_viable_mtu_dataplane }}
{% for network in role_networks if network.startswith('Tenant') %}
  - type: vlan
    device: bond-data
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

多个 NIC 示例

以下示例使用第二个 NIC 连接到具有 DHCP 地址的基础架构网络，以及绑定的另一个 NIC。

```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    mtu: {{ tenant_mtu }}
    use_dhcp: true
    primary: true
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
```

```
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: ovs_bridge
    name: br-bond
    mtu: {{ external_mtu }}
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
    members:
      - type: interface
        name: nic10
        mtu: {{ external_mtu }}
        use_dhcp: false
        primary: true
      - type: vlan
        mtu: {{ external_mtu }}
        vlan_id: {{ external_vlan_id }}
        addresses:
        - ip_netmask: {{ external_ip }}/{{ external_cidr }}
        routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

8.2.1.4. 为预置备节点自定义 NIC 映射

如果使用预置备节点，您可以使用以下方法之一为特定节点指定 os-net-config 映射：

- **在环境文件中配置 NetConfigDataLookup heat 参数，并运行不带 --network-config 的 openstack overcloud node provision 命令。**
- **在节点定义文件 overcloud-baremetal-deploy.yaml 中配置 net_config_data_lookup 属性，并使用 --network-config 运行 openstack overcloud node provision 命令。**

注意

如果没有使用预置备节点，则必须在节点定义文件中配置 NIC 映射。有关配置 net_config_data_lookup 属性的更多信息，请参阅 裸机节点置备属性。

您可以为每个节点上的物理接口分配别名，以预先确定哪些物理 NIC 映射到特定别名，如 nic1 或 nic2，您可以将 MAC 地址映射到指定的别名。您可以使用 MAC 地址或 DMI 关键字映射特定节点，也可以使用 DMI 关键字映射一组节点。以下示例配置三个节点，以及两个节点组，其别名指向物理接口。生成的配置由 os-net-config 应用。在每个节点上，您可以在 /etc/os-net-config/mapping.yaml 文件的 interface_mapping 部分中看到应用的配置。

示例 1：在 os-net-config-mappings.yaml中配置 NetConfigDataLookup 参数

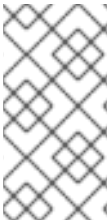
```
NetConfigDataLookup:
  node1: (1)
    nic1: "00:c8:7c:e6:f0:2e"
  node2:
    nic1: "00:18:7d:99:0c:b6"
  node3: (2)
    dmiString: "system-uuid" (3)
    id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
    nic1: em3
  # Dell PowerEdge
  nodegroup1: (4)
    dmiString: "system-product-name"
    id: "PowerEdge R630"
    nic1: em3
    nic2: em1
    nic3: em2
  # Cisco UCS B200-M4"
  nodegroup2:
    dmiString: "system-product-name"
    id: "UCSB-B200-M4"
    nic1: enp7s0
    nic2: enp6s0
```

(1) 将 node1 映射到指定的 MAC 地址，并将 nic1 分配为该节点上的 MAC 地址的别名。

(2) 使用系统 UUID "A8C85861-1B16-4803-8689-AFC62984F8F6"将 node3 映射到节点，并将 nic1 分配为这个节点上的 em3 接口的别名。

(3) dmiString 参数必须设置为有效的字符串关键字。有关有效字符串关键字的列表，请查看 DMIDECODE (8)手册页。

(4) 将 nodegroup1 中的所有节点映射到具有产品名称为 "PowerEdge R630" 的节点，并将 nic1、nic2 和 nic3 分配为该节点上命名接口的别名。



注意

通常，os-net-config 只注册已以 UP 状态连接的接口。但是，如果您使用自定义映射文件的硬编码接口，接口也会注册，即使它处于 DOWN 状态。

示例 2：在 overcloud-baremetal-deploy.yaml - 特定节点中配置 net_config_data_lookup 属性

```
- name: Controller
  count: 3
  defaults:
    network_config:
      net_config_data_lookup:
        node1:
          nic1: "00:c8:7c:e6:f0:2e"
        node2:
          nic1: "00:18:7d:99:0c:b6"
        node3:
          dmiString: "system-uuid"
          id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
          nic1: em3
        # Dell PowerEdge
        nodegroup1:
          dmiString: "system-product-name"
          id: "PowerEdge R630"
          nic1: em3
          nic2: em1
          nic3: em2
        # Cisco UCS B200-M4"
        nodegroup2:
          dmiString: "system-product-name"
          id: "UCSB-B200-M4"
          nic1: enp7s0
          nic2: enp6s0
```

示例 3：在 overcloud-baremetal-deploy.yaml 中配置 net_config_data_lookup 属性 - 角色的所有节点

```
- name: Controller
  count: 3
  defaults:
    network_config:
      template: templates/net_config_bridge.j2
      default_route_network:
      - external
  instances:
```

```
- hostname: overcloud-controller-0
  network_config:
    <name/groupname>:
      nic1: 'XX:XX:XX:XX:XX:XX'
      nic2: 'YY:YY:YY:YY:YY:YY'
      nic3: 'ens1f0'
```

8.2.2. 可组合网络

如果要在不同网络上托管特定的网络流量，可以创建自定义可组合网络。director 提供了一个启用了网络隔离的默认网络拓扑。您可以在 /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml 中找到此配置。

overcloud 默认使用以下预定义网络段集合：

- 内部 API
- 存储
- 存储管理
- 租户
- 外部

您可以使用可组合网络为各种服务添加网络。例如，如果您有一个专用于 NFS 流量的网络，您可以将其提供给多个角色。

director 支持在部署和更新阶段创建自定义网络。您可以将这些额外网络用于裸机节点、系统管理，或为不同的角色创建单独的网络。您还可以使用它们创建多组网络，以便在网络间路由流量的分割部署。

8.2.2.1. 添加可组合网络

使用可组合网络为各种服务添加网络。例如，如果您有一个专用于存储备份流量的网络，您可以将网络提供给多个角色。

流程

1. 列出可用的网络配置文件示例：

```
$ ll /usr/share/openstack-tripleo-heat-templates/network-data-samples/
-rw-r--r--. 1 root root 1554 May 11 23:04 default-network-isolation-ipv6.yaml
-rw-r--r--. 1 root root 1181 May 11 23:04 default-network-isolation.yaml
-rw-r--r--. 1 root root 1126 May 11 23:04 ganesha-ipv6.yaml
-rw-r--r--. 1 root root 1100 May 11 23:04 ganesha.yaml
-rw-r--r--. 1 root root 3556 May 11 23:04 legacy-routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2929 May 11 23:04 legacy-routed-networks.yaml
-rw-r--r--. 1 root root  383 May 11 23:04 management-ipv6.yaml
-rw-r--r--. 1 root root  290 May 11 23:04 management.yaml
-rw-r--r--. 1 root root  136 May 11 23:04 no-networks.yaml
-rw-r--r--. 1 root root 2725 May 11 23:04 routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2033 May 11 23:04 routed-networks.yaml
-rw-r--r--. 1 root root  943 May 11 23:04 vip-data-default-network-isolation.yaml
-rw-r--r--. 1 root root  848 May 11 23:04 vip-data-fixed-ip.yaml
-rw-r--r--. 1 root root 1050 May 11 23:04 vip-data-routed-networks.yaml
```

2. 将您需要的示例网络配置文件从 **/usr/share/openstack-tripleo-heat-templates/network-data-samples** 复制到环境文件目录中：

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml /home/stack/templates/network_data.yaml
```

3. 编辑 **network_data.yaml** 配置文件并为新网络添加一个部分：

```
- name: StorageBackup
  vip: false
  name_lower: storage_backup
  subnets:
    storage_backup_subnet:
      ip_subnet: 172.16.6.0/24
      allocation_pools:
        - start: 172.16.6.4
        - end: 172.16.6.250
      gateway_ip: 172.16.6.1
```

为您的环境配置任何其他网络属性。有关可用于配置网络属性的属性的更多信息，请参阅 **link:https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_with_director/assembly_configuring-overcloud-networking_installing-director-**

on-the-undercloud#ref_network-definition-file-configuration-options_overcloud_networking [网络定义文件配置选项]。

4. 当您添加包含虚拟 IP 的额外可组合网络并希望将一些 API 服务映射到此网络时，请使用 CloudName{network.name} 定义来为 API 端点设置 DNS 名称：

```
CloudName{{network.name}}
```

下面是一个示例：

```
parameter_defaults:
  ...
  CloudNameOcProvisioning: baremetal-vip.example.com
```

5. 将您需要的示例网络 VIP 定义模板从 /usr/share/openstack-tripleo-heat-templates/network-data-samples 复制到环境文件目录中。以下示例将 vip-data-default-network-isolation.yaml 复制到名为 vip_data.yaml 的本地环境文件：

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/vip-data-default-network-isolation.yaml  /home/stack/templates/vip_data.yaml
```

6. 编辑 vip_data.yaml 配置文件。虚拟 IP 数据是虚拟 IP 地址定义列表，各自包含分配 IP 地址的网络的名称：

```
- network: storage_mgmt
  dns_name: overcloud
- network: internal_api
  dns_name: overcloud
- network: storage
  dns_name: overcloud
- network: external
  dns_name: overcloud
  ip_address: <vip_address>
- network: ctlplane
  dns_name: overcloud
```

- 将 <vip_address > 替换为所需的虚拟 IP 地址。

有关您可以在 VIP 定义文件中配置网络 VIP 属性的属性的更多信息，请参阅 Network VIP 属性属性。

7.

复制示例网络配置模板。Jinja2 模板用于定义 NIC 配置模板。如果其中一个示例与您的要求匹配，浏览 /usr/share/ansible/roles/tripleo_network_config/templates/ 目录中提供的示例。如果示例与您的要求不匹配，请复制示例配置文件，并根据您的需要进行修改：

```
$ cp
/usr/share/ansible/roles/tripleo_network_config/templates/single_nic_vlans/single_nic_vlans.j2
/home/stack/templates/
```

8. **编辑 single_nic_vlans.j2 配置文件：**

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    # force the MAC address of the bridge to this interface
    primary: true
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
```

9. **在 overcloud-baremetal-deploy.yaml 配置文件中设置 network_config 模板：**

```
- name: CephStorage
  count: 3
  defaults:
    networks:
    - network: storage
```

```
    - network: storage_mgmt
    - network: storage_backup
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
```

10. **调配 overcloud 网络。此操作会生成一个输出文件，它将在部署 overcloud 时使用环境文件：**

```
(undercloud)$ openstack overcloud network provision --output <deployment_file>
/home/stack/templates/<networks_definition_file>.yaml
```

- **将 <networks_definition_file > 替换为网络定义文件的名称，如 network_data.yaml。**
- **将 <deployment_file > 替换为用于部署命令生成的 heat 环境文件的名称，如 /home/stack/templates/overcloud-networks-deployed.yaml。**

11. **置备网络 VIP 并生成 vip-deployed-environment.yaml 文件。在部署 overcloud 时，您可以使用此文件：**

```
(overcloud)$ openstack overcloud network vip provision  --stack <stack> --output
<deployment_file> /home/stack/templates/vip_data.yaml
```

- **将 & lt;stack > 替换为置备网络 VIP 的堆栈名称。如果未指定，则默认为 overcloud。**
- **将 <deployment_file > 替换为用于部署命令生成的 heat 环境文件的名称，如 /home/stack/templates/overcloud-vip-deployed.yaml。**

8.2.2.2. 在角色中包含可组合网络

您可以将可组合网络分配给您的环境中定义的 overcloud 角色。例如，您可以包含带有 Ceph Storage 节点的自定义 StorageBackup 网络。

流程

1. **如果您还没有自定义 roles_data.yaml 文件，请将默认值复制到您的主目录中：**

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
/home/stack/templates/roles_data.yaml
```

2. 编辑自定义 **roles_data.yaml** 文件。

3. 在网络列表中包含您要添加 网络 的角色的网络名称。例如，要将 **StorageBackup** 网络添加到 **Ceph Storage** 角色，请使用以下示例片断：

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    Storage
      subnet: storage_subnet
    StorageMgmt
      subnet: storage_mgmt_subnet
    StorageBackup
      subnet: storage_backup_subnet
```

4. 将自定义网络添加到对应的角色后，保存文件。

运行 **openstack overcloud deploy** 命令时，使用 **-r** 选项包含自定义 **roles_data.yaml** 文件。如果没有 **-r** 选项，部署命令将使用一组默认的角色，以及它们对应的网络。

8.2.2.3. 将 OpenStack 服务分配给可组合网络

每个 **OpenStack** 服务都分配给资源 **registry** 中的默认网络类型。这些服务绑定到网络类型分配的网络中的 **IP** 地址。虽然 **OpenStack** 服务在这些网络中划分，但实际的物理网络数量可能会因网络环境文件中定义的不同。您可以通过在环境文件中定义新网络映射将 **OpenStack** 服务重新分配给不同的网络类型，如 **/home/stack/templates/service-reassignments.yaml**。**ServiceNetMap** 参数决定您要用于每个服务的网络类型。

例如，您可以通过修改突出显示的部分，将存储管理网络服务重新分配给 **Storage Backup Network**：

```
parameter_defaults:
  ServiceNetMap:
    SwiftStorageNetwork: storage_backup
    CephClusterNetwork: storage_backup
```

将这些参数改为 **storage_backup** 将这些服务放在 **Storage Backup** 网络中，而不是 **Storage Management** 网络。这意味着，您只需要为 **Storage Backup** 网络而不是 **Storage Management** 网络定义一组 **parameter_defaults**。

director 将自定义 ServiceNetMap 参数定义合并到预定义的默认值列表中，它从 ServiceNetMapDefaults 获取并覆盖默认值。director 将完整列表（包括自定义）返回到 ServiceNetMap，后者用于为各种服务配置网络分配。

服务映射适用于使用 Pacemaker 的节点的 network_data.yaml 文件中的 vip: true 网络。overcloud 负载均衡器将来自 VIP 的流量重定向到特定的服务端点。

注意

您可以在 /usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml 文件中的 ServiceNetMapDefaults 参数中找到默认服务的完整列表。

8.2.2.4. 启用自定义可组合网络

使用其中一个默认 NIC 模板启用自定义可组合网络。在本例中，将 Single NIC 与 VLAN 模板一起使用(custom_single_nic_vlans)。

流程

1. 查找 stackrc undercloud 凭证文件：

```
$ source ~/stackrc
```

2. 置备 overcloud 网络：

```
$ openstack overcloud network provision \
  --output overcloud-networks-deployed.yaml \
  custom_network_data.yaml
```

3. 置备网络 VIP：

```
$ openstack overcloud network vip provision \
    --stack overcloud \
    --output overcloud-networks-vips-deployed.yaml \
     custom_vip_data.yaml
```

4. 置备 overcloud 节点：

```
$ openstack overcloud node provision \
   --stack overcloud \
   --output overcloud-baremetal-deployed.yaml \
   overcloud-baremetal-deploy.yaml
```

5. 构建 **openstack overcloud deploy** 命令，按所需顺序指定配置文件和模板，例如：

```
$ openstack overcloud deploy --templates \
  --networks-file network_data_v2.yaml \
  -e overcloud-networks-deployed.yaml \
  -e overcloud-networks-vips-deployed.yaml \
  -e overcloud-baremetal-deployed.yaml
  -e custom-net-single-nic-with-vlans.yaml
```

本例命令可在 **overcloud** 中的节点间部署可组合网络，包括您的额外网络。

8.2.2.5. 重命名默认网络

您可以使用 **network_data.yaml** 文件修改默认网络的用户可见名称：

- **InternalApi**
- **外部**
- **存储**
- **StorageMgmt**
- **租户**

要更改这些名称，请不要修改 **name** 字段。相反，将 **name_lower** 字段更改为网络的新名称，并使用新名称更新 **ServiceNetMap**。

流程

1. 在 **network_data.yaml** 文件中，为每个您要重命名的网络在 **name_lower** 参数中输入新名

称：

```
- name: InternalApi
  name_lower: MyCustomInternalApi
```

2. 在 service_net_map_replace 参数中包含 name_lower 参数的默认值：

```
- name: InternalApi
  name_lower: MyCustomInternalApi
  service_net_map_replace: internal_api
```

8.2.3. 其他 overcloud 网络配置

本章介绍了 第 8.2.1 节 “定义自定义网络接口模板” 中概述的概念和程序，并提供一些额外的信息来帮助配置 overcloud 网络的部分。

8.2.3.1. 配置路由和默认路由

您可以通过以下两种方式之一设置主机的默认路由。如果接口使用 DHCP，并且 DHCP 服务器提供网关地址，则系统将使用该网关的默认路由。否则，您可以在具有静态 IP 的接口上设置默认路由。

虽然 Linux 内核支持多个默认网关，但它仅使用具有最低指标的网关。如果有多个 DHCP 接口，这可能会导致无法预计的默认网关。在这种情况下，建议为使用默认路由的接口以外的接口设置 defroute: false。

例如，您可能希望 DHCP 接口(nic3)作为默认路由。使用以下 YAML 代码片段，禁用另一 DHCP 接口(nic2)上的默认路由：

```
# 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 参数仅适用于通过 DHCP 获取的路由。

要在具有静态 IP 的接口上设置静态路由，请指定到子网的路由。例如，您可以通过 Internal API 网络上的 172.17.0.1 上的网关设置到 10.1.2.0/24 子网的路由：

```
- type: vlan
  device: bond1
  vlan_id: 9
  addresses:
  - ip_netmask: 172.17.0.100/16
  routes:
  - ip_netmask: 10.1.2.0/24
    next_hop: 172.17.0.1
```

8.2.3.2. 配置基于策略的路由

要从 Controller 节点上的不同网络配置无限访问，请配置基于策略的路由。基于策略的路由使用路由表，在具有多个接口的主机上，您可以根据源地址通过特定接口发送流量。您可以将来自不同源的数据包路由到不同的网络，即使目的地是相同的。

例如，您可以将路由配置为根据数据包的源地址将流量发送到内部 API 网络，即使默认路由是 External 网络。您还可以为每个接口定义特定的路由规则。

Red Hat OpenStack Platform 使用 os-net-config 工具为您的 overcloud 节点配置网络属性。os-net-config 工具管理 Controller 节点上的以下网络路由：

- **/etc/iproute2/rt_tables 文件中的路由表**
- **/etc/sysconfig/network-scripts/rule-{ifname} 文件中的 IPv4 规则**
- **/etc/sysconfig/network-scripts/rule6-{ifname} 文件中的 IPv6 规则**
- **/etc/sysconfig/network-scripts/route-{ifname}中的路由表特定路由**

先决条件

- 您已成功安装了 undercloud。如需更多信息，请参阅使用 director *安装和管理 Red Hat OpenStack Platform 指南中的在 undercloud 上安装* director。

流程

1. 从 /home/stack/templates/custom-nics 目录创建自定义 NIC 模板中的 接口 条目，为接口定义路由，并定义与部署相关的规则：

```
network_config:
- type: interface
  name: em1
  use_dhcp: false
  addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
  routes:
    - default: true
      next_hop: {{ external_gateway_ip }}
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
      next_hop: {{ external_gateway_ip }}
      table: 2
      route_options: metric 100
  rules:
    - rule: "iif em1 table 200"
      comment: "Route incoming traffic to em1 with table 200"
    - rule: "from 192.0.2.0/24 table 200"
      comment: "Route all traffic from 192.0.2.0/24 with table 200"
    - rule: "add blackhole from 172.19.40.0/24 table 200"
    - rule: "add unreachable iif em1 from 192.168.1.0/24"
```

2. 在部署命令中包含您的自定义 NIC 配置和网络环境文件，以及与部署相关的任何其他环境文件：

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

验证

- 在 Controller 节点上输入以下命令来验证路由配置是否正常工作：

```
$ cat /etc/iproute2/rt_tables
$ ip route
$ ip rule
```

8.2.3.3. 配置巨型帧

最大传输单元(MTU)设置决定了通过单个以太网帧传输的最大数据量。使用较大的值会降低开销，因为每个帧以标头的形式添加数据。默认值为 1500，使用较高的值需要配置交换机端口来支持巨型帧。大多数交换机支持至少 9000 的 MTU，但很多交换机默认配置为 1500。

VLAN 的 MTU 不能超过物理接口的 MTU。确保在绑定或接口中包含 MTU 值。

存储、存储管理、内部 API 和租户网络都可以从巨型帧中受益。

您可以在 jinja2 模板或 network_data.yaml 文件中更改 mtu 的值。如果您在 network_data.yaml 文件中设置了值，它将在部署过程中呈现。

警告

路由器通常不能跨第 3 层边界转发巨型帧。为避免连接问题，请不要更改 Provisioning 接口、外部接口和任何浮动 IP 接口的默认 MTU。

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: bridge_name
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: vlan
    mtu: 9000 (1)
    vlan_id: {{ storage_vlan_id }}
```

```
    addresses:
    - ip_netmask: {{ storage_ip }}/{{ storage_cidr }}
    routes: {{ [storage_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ storage_mgmt_mtu }} (2)
    vlan_id: {{ storage_mgmt_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_mgmt_ip }}/{{ storage_mgmt_cidr }}
    routes: {{ [storage_mgmt_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}
    routes: {{ [internal_api_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ external_mtu }}
    vlan_id: {{ external_vlan_id }}
    addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr }}
    routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

(1)

在 jinja2 模板中直接更新 MTU 值。

(2)

部署期间，MTU 值从 network_data.yaml 文件获取。

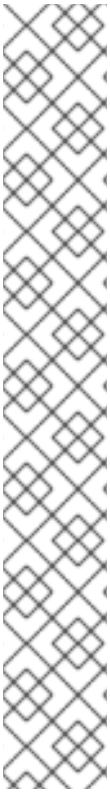
8.2.3.4. 配置 ML2/OVN 北向路径 MTU 发现，以实现巨型帧碎片

如果内部网络上的虚拟机向外部网络发送巨型帧，并且内部网络的最大传输单元(MTU)超过外部网络的 MTU，则北向帧可以轻松地超过外部网络的容量。

ML2/OVS 会自动处理通过数据包问题进行这一处理，ML2/OVN 会自动为 TCP 数据包处理它。

但是，为了确保在使用 ML2/OVN 机制驱动程序的部署中正确处理 oversized northbound UDP 数据包，您需要执行额外的配置步骤。

这些步骤将 ML2/OVN 路由器配置为将 ICMP "fragmentation required" 数据包返回到发送虚拟机，其中发送应用可以将有效负载分成较小的数据包。



注意

在 east/west 流量中，RHOSP ML2/OVN 部署不支持在 east/west 路径上大于最小 MTU 的数据包碎片。例如：

- VM1 位于 Network1 上，MTU 为 1300。
- VM2 位于 Network2 上，MTU 为 1200。
- VM1 和 VM2 之间的指示的 ping 大小为 1171 或小于成功。大小大于 1171 的 ping 会导致数据包丢失的 100%。

由于没有客户的具体要求，红帽还没有计划提供对它的支持。

流程

1. 在 ml2_conf.ini 的 [ovn] 部分中设置以下值：

```
ovn_emit_need_to_frag = True
```

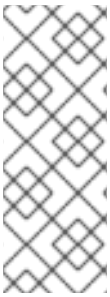
8.2.3.5. 在中继接口上配置原生 VLAN

如果中继接口或绑定在原生 VLAN 上有一个网络，IP 地址会直接分配给网桥，且没有 VLAN 接口。

以下示例配置了一个绑定接口，其中 External 网络位于原生 VLAN 中：

```
network_config:
- type: ovs_bridge
  name: br-ex
  addresses:
  - ip_netmask: {{ external_ip }}/{{ external_cidr }}
  routes: {{ external_host_routes }}
  members:
  - type: ovs_bond
    name: bond1
```

```
ovs_options: {{ bond_interface_ovs_options }}
members:
- type: interface
  name: nic3
  primary: true
- type: interface
  name: nic4
```



注意

当您将 address 或 route 语句移到网桥时，从网桥中删除对应的 VLAN 接口。对所有适用的角色进行更改。External 网络仅在控制器上，因此只有控制器模板需要更改。Storage 网络附加到所有角色，因此如果 Storage 网络位于默认的 VLAN 中，则所有角色都需要修改。

8.2.3.6. 增加 netfilter 跟踪的最大连接数

Red Hat OpenStack Platform (RHOSP)网络服务(neutron)使用 netfilter 连接跟踪来构建有状态防火墙，并在虚拟网络上提供网络地址转换(NAT)。在某些情况下，可能会导致内核空间达到最大连接限制，并产生错误，如 nf_conntrack: table full, discard packet。您可以增加连接跟踪(conntrack)的限制，并避免这些类型的错误。您可以在 RHOSP 部署中增加一个或多个角色的 conntrack 限制，或跨所有节点增加 conntrack 限制。

先决条件

- **成功安装 RHOSP undercloud。**

流程

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

```
$ source ~/stackrc
```

3. **创建自定义 YAML 环境文件。**

示例

```
$ vi /home/stack/templates/custom-environment.yaml
```

4. **您的环境文件必须包含关键字 parameter_defaults 和 ExtraSysctlSettings。为 netfilter 可在变量 net.nf_conntrack_max 中跟踪的最大连接数输入新值。**

示例

在本例中，您可以在 RHOSP 部署中的所有主机中设置 conntrack 限制：

```
parameter_defaults:
  ExtraSysctlSettings:
    net.nf_conntrack_max:
      value: 500000
```

使用 & lt;role>Parameter 参数为特定角色设置 conntrack 限制：

```
parameter_defaults:
  <role>Parameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: <simultaneous_connections>
```

- **将 <role > 替换为角色的名称。**

  例如，使用 ControllerParameters 为 Controller 角色设置 conntrack 限制，或 ComputeParameters 为 Compute 角色设置 conntrack 限制。

- **将 < simultaneous_connections > 替换为您要允许的同时连接数量。**

示例

在本例中，您只能在 RHOSP 部署中为 Controller 角色设置 conntrack 限制：

```
parameter_defaults:
  ControllerParameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: 500000
```

注意

net.nf_conntrack_max 的默认值为 500000 连接。最大值为：4294967295。

5. 运行部署命令，并包含核心 heat 模板、环境文件以及新的自定义环境文件。

重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源更为优先。

示例

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/custom-environment.yaml
```

其他资源

- 环境文件
- 在 overcloud 创建中包含环境文件

8.2.4. 网络接口绑定

您可以在自定义网络配置中使用各种绑定选项。

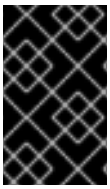
8.2.4.1. overcloud 节点的网络接口绑定

您可以将多个物理 NIC 捆绑在一起组成一个逻辑频道，称为绑定。您可以配置绑定，为高可用性系统提供冗余性或增加吞吐量。

Red Hat OpenStack Platform 支持 Open vSwitch (OVS)内核绑定、OVS-DPDK 绑定和 Linux 内核绑定。

表 8.7. 支持的接口绑定类型

绑定类型	类型值	允许的网桥类型	允许的成员
OVS 内核绑定	**ovs_bond**	**ovs_bridge**	**interface**
OVS-DPDK 绑定	**ovs_dpdk_bond**	**ovs_user_bridge**	**ovs_dpdk_port**
Linux 内核绑定	**linux_bond**	**ovs_bridge** 或 **linux_bridge**	**interface**



重要

不要在同一节点上组合 **ovs_bridge** 和 **ovs_user_bridge**。

8.2.4.2. 创建 Open vSwitch (OVS)绑定

您可以在网络接口模板中创建 OVS 绑定。例如，您可以创建一个绑定作为 OVS 用户空间网桥的一部分：

```
- type: ovs_user_bridge
  name: br-dpdk0
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: {{ num_dpdk_interface_rx_queues }}
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      members:
      - type: interface
        name: nic4
    - type: ovs_dpdk_port
      name: dpdk1
      members:
      - type: interface
        name: nic5
```

在本例中，您可以从两个 DPDK 端口创建绑定。

ovs_options 参数包含绑定选项。您可以使用 **BondInterfaceOvsOptions** 参数在网络环境文件中配置绑定选项：

```
environment_parameters:
  BondInterfaceOvsOptions: "bond_mode=active_backup"
```

8.2.4.3. Open vSwitch (OVS)绑定选项

您可以使用 NIC 模板文件中的 ovs_options heat 参数设置各种 Open vSwitch (OVS)绑定选项。

bond_mode=balance-slb

源负载平衡(slb)根据源 MAC 地址和输出 VLAN 平衡流，并在流量模式更改时定期重新平衡。当您使用 balance-slb 绑定选项配置绑定时，远程交换机不需要配置。Networking 服务(neutron)将每个源 MAC 和 VLAN 对分配给链接，并通过该链接传输来自该 MAC 和 VLAN 的所有数据包。使用基于源 MAC 地址和 VLAN 号的简单哈希算法，随着流量模式的变化，定期重新平衡。balance-slb 模式与 Linux 绑定驱动程序使用的模式 2 绑定类似。您可以使用此模式提供负载平衡，即使交换机没有配置为使用 LACP。

bond_mode=active-backup

当您使用 active-backup 绑定模式配置绑定时，网络服务会将一个 NIC 保留为待机。当活跃连接失败时，待机 NIC 会恢复网络操作。物理交换机仅显示一个 MAC 地址。这个模式不需要切换配置，当链接连接到单独的交换机时，可以正常工作。这个模式不提供负载均衡。

lacp=[active | passive | off]

控制链路聚合控制协议(LACP)行为。只有某些交换机支持 LACP。如果您的交换机不支持 LACP，请使用 bond_mode=balance-slb 或 bond_mode=active-backup。

other-config:lacp-fallback-ab=true

如果 LACP 失败，将 active-backup 设置为绑定模式。

other_config:lacp-time=[fast | slow]

将 LACP heartbeat 设置为 1 秒 (fast) 或 30 秒 （slow）。默认设置会较慢。

other_config:bond-detect-mode=[miimon | carrier]

将链路检测设置为使用 miimon heartbeats (miimon)或监控载体(carrier)。默认值为 carrier。

other_config:bond-miimon-interval=100

如果使用 miimon，请设置心跳间隔（毫秒）。

bond_updelay=1000

设置链接必须激活的时间间隔（毫秒），以防止流动。

other_config:bond-rebalance-interval=10000

设置在绑定成员之间流重新平衡的时间间隔（毫秒）。将此值设置为 0，以禁用绑定成员之间的流重新平衡。

8.2.4.4. 使用带有 Open vSwitch (OVS)绑定模式的链路聚合控制协议(LACP)

您可以将绑定与可选的链路聚合控制协议(LACP)一起使用。LACP 是一个协商协议，为负载平衡和容错创建动态绑定。

使用下表了解 OVS 内核和 OVS-DPDK 绑定接口与 LACP 选项的兼容性。

重要

在控制和存储网络上，红帽建议您在 VLAN 和 LACP 中使用 Linux 绑定，因为 OVS 绑定可能会在 OVS 或 neutron 代理重启更新、热修复和其他事件时发生 control plane 中断。Linux bond/LACP/VLAN 配置提供 NIC 管理，而无需 OVS 中断。

表 8.8. OVS 内核和 OVS-DPDK 绑定模式的 LACP 选项

目标	OVS 绑定模式	兼容 LACP 选项	备注
高可用性（主动-被动）	**active-backup**	主动、被动 或关闭	
增加吞吐量(active-active)	**balance-slb**	主动、被动 或关闭	• 性能受每个数据包的额外解析的影响。 • vhost-user 锁定争用的可能性。

	balance-tcp	**主动 或 被动**	• 与 balance-slb 一样，性能会受到每个数据包的额外解析的影响，并且 vhost-user 锁定争用存在潜在的。 • 必须配置并启用 LACP。 • 设置 **lb-output-action=true**。例如： `ovs-vsctl set port <bond port> other_config:lb-output-action=true`

8.2.4.5. 创建 Linux 绑定

您可以在网络接口模板中创建 Linux 绑定。例如，您可以创建一个绑定两个接口的 Linux 绑定：

```
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
```

bonding_options 参数为 Linux 绑定设置特定的绑定选项。

模式

设置绑定模式，示例中是 802.3ad 或 LACP 模式。有关 Linux 绑定模式的更多信息，请参阅 Red Hat Enterprise Linux 9 配置和管理网络指南中的 "上游交换配置取决于绑定模式 "。

lacp_rate

定义是否每 1 秒发送 LACP 数据包，或者每 30 秒发送一次。

updelay

定义接口在用于流量前必须处于活跃状态的最短时间。这个最小配置有助于缓解端口流中断。

miimon

用于使用驱动程序的 MIIMON 功能监控端口状态的时间间隔（毫秒）。

使用以下附加示例来配置您自己的 Linux 绑定：

- 使用一个 VLAN 将 Linux 绑定设置为 active-backup 模式：

```
....

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: "mode=active-backup"
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask:
        {{ internal_api_ip }}/{{ internal_api_cidr }}
      routes:
        {{ internal_api_host_routes }}
```

- OVS 网桥上的 Linux 绑定。绑定设置为 802.3ad LACP 模式，它带有一个 VLAN：

```
- type: linux_bond
  name: bond_tenant
  mtu: {{ min_viable_mtu_ctlplane }}
  bonding_options: "mode=802.3ad updelay=1000 miimon=100"
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameserver }}
  domain: {{ dns_search_domains }}
  members:
    - type: interface
      name: p1p1
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: interface
      name: p1p2
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: vlan
      mtu: {{ tenant_mtu }}
      vlan_id: {{ tenant_vlan_id }}
      addresses:
        - ip_netmask:
            {{ tenant_ip }}/{{ tenant_cidr }}
      routes:
        {{ tenant_host_routes }}
```

重要

您必须设置 min_viable_mtu_ctlplane，然后才能使用它。将 /usr/share/ansible/roles/tripleo_network_config/templates/2_linux_bonds_vlans.j2 复制到您的 templates 目录中，并根据您的需要进行修改。如需更多信息，请参阅 可组合网络，并参考与网络配置模板相关的步骤。

8.2.5. 更新网络配置文件的格式

Red Hat OpenStack Platform (RHOSP) 17.0 中更改了网络配置 yaml 文件格式。网络配置文件 network_data.yaml 的结构已更改，NIC 模板文件格式已从 yaml 文件格式改为 Jinja2 ansible 格式 j2。

您可以使用以下转换工具将当前部署中的现有网络配置文件转换为 RHOSP 17+ 格式：

- **convert_v1_net_data.py**
- **convert_heat_nic_config_to_ansible_j2.py**

您还可以手动转换现有的 NIC 模板文件。

您需要转换的文件包括：

- network_data.yaml
- 控制器 NIC 模板
- 计算 NIC 模板
- 任何其它自定义网络文件

8.2.5.1. 更新网络配置文件的格式

Red Hat OpenStack Platform (RHOSP) 17.0 中更改了网络配置 yaml 文件的格式。您可以使用 convert_v1_net_data.py 转换工具，将当前部署中的现有网络配置文件转换为 RHOSP 17+ 格式。

流程

1. 下载转换工具：
   - /usr/share/openstack-tripleo-heat-templates/tools/convert_v1_net_data.py
2. 将 RHOSP 16+ 网络配置文件转换为 RHOSP 17+ 格式：

```
$ python3 convert_v1_net_data.py <network_config>.yaml
```

   - 将 < network_config > 替换为您要转换的现有配置文件的名称，如 network_data.yaml。

8.2.5.2. 自动将 NIC 模板转换为 Jinja2 Ansible 格式

在 Red Hat OpenStack Platform (RHOSP) 17.0 中，NIC 模板文件格式已从 yaml 文件格式改为 Jinja2 Ansible 格式 j2。

您可以使用 convert_heat_nic_config_to_ansible_j2.py 转换工具，将当前部署中的现有 NIC 模板文件转换为 Jinja2 格式。

您还可以手动转换现有的 NIC 模板文件。如需更多信息，请参阅 手动将 NIC 模板转换为 Jinja2 Ansible 格式。

您需要转换的文件包括：

- 控制器 NIC 模板
- 计算 NIC 模板
- 任何其它自定义网络文件

流程

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

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

3. 将转换工具复制到 undercloud 上的当前目录：

```
$ cp /usr/share/openstack-tripleo-heat-
templates/tools/convert_heat_nic_config_to_ansible_j2.py .
```

4. 将计算和控制器 NIC 模板文件以及任何其他自定义网络文件转换为 Jinja2 Ansible 格式：

```
$ python3 convert_heat_nic_config_to_ansible_j2.py \
 [--stack <overcloud> | --standalone] --networks_file <network_config.yaml> \
 <network_template>.yaml
```

- 将 <overcloud> 替换为 overcloud 堆栈的名称或 UUID。如果未指定 --stack，堆栈默

认为 **overcloud**。

注意

您只能在 RHOSP 16 部署中使用 --stack 选项，因为它需要在 undercloud 节点上运行编排服务(heat)。从 RHOSP 17 开始，RHOSP 部署使用临时 heat，它会在容器中运行编排服务。如果编排服务不可用，或者没有堆栈，则使用 --standalone 选项而不是 --stack。

- **将 <network_config.yaml > 替换为描述网络部署的配置文件的名称，如 network_data.yaml。**

- **将 < network_template > 替换为您要转换的网络配置文件的名称。**

重复此命令，直到您转换了所有自定义网络配置文件。convert_heat_nic_config_to_ansible_j2.py 脚本为您传递给它的每个 yaml 文件生成一个 .j2 文件，以进行转换。

5. **检查每个生成的 .j2 文件，以确保配置正确并完成您的环境，并手动处理工具生成的注释，以突出显示无法转换配置的位置。有关手动将 NIC 配置转换为 Jinja2 格式的更多信息，请参阅 Heat 参数到 Ansible 变量映射。**

6. **配置 network-environment.yaml 文件中的 *NetworkConfigTemplate 参数，使其指向生成的 .j2 文件：**

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

7. **从您的 network-environment.yaml 文件中删除旧网络配置文件的 resource_registry 映射：**

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

8.2.5.3. 手动将 NIC 模板转换为 Jinja2 Ansible 格式

在 Red Hat OpenStack Platform (RHOSP) 17.0 中，NIC 模板文件格式已从 yaml 文件格式改为 Jinja2 Ansible 格式 j2。

您可以手动转换现有的 NIC 模板文件。

您还可以使用 convert_heat_nic_config_to_ansible_j2.py 转换工具，将当前部署中的现有 NIC 模板文件转换为 Jinja2 格式。如需更多信息，请参阅自动将 NIC 模板转换为 Jinja2 ansible 格式。

您需要转换的文件包括：

- 控制器 NIC 模板
- 计算 NIC 模板
- 任何其它自定义网络文件

流程

1. 创建 Jinja2 模板。您可以通过从 undercloud 节点上的 /usr/share/ansible/roles/tripleo_network_config/templates/ 目录中复制示例模板来创建新模板。
2. 将 heat 内部函数替换为 Jinja2 过滤器。例如，使用以下过滤器来计算 min_viable_mtu ：

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
```

3. 使用 Ansible 变量为您的部署配置网络属性。您可以手动配置每个网络，或通过迭代 role_networks 来配置每个网络：
   - 要手动配置每个网络，请将每个 get_param 函数替换为对应的 Ansible 变量。例如，如果您当前的部署使用 get_param: InternalApiNetworkVlanID 配置 vlan_id，然后将以下

配置添加到您的模板中：

```
vlan_id: {{ internal_api_vlan_id }}
```

表 8.9. 从 heat 参数映射到 Ansible vars的网络属性示例

yaml 文件格式	Jinja2 ansible 格式，j2
- type: vlan device: nic2 vlan_id: get_param: InternalApiNetworkVlanID addresses: - ip_netmask: get_param: InternalApiIpSubnet	- type: vlan device: nic2 vlan_id: {{ internal_api_vlan_id }} addresses: - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}

- **要编程地配置每个网络，请将 Jinja2 for-loop 结构添加到您的模板，该模板使用 role_networks 通过其角色名称检索可用的网络。**

  示例

```
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{%- endfor %}
```

有关 heat 参数与 Ansible 变量等效的映射的完整列表，请参阅 Heat 参数到 Ansible 变量映射。

4. **配置 network-environment.yaml 文件中的 *NetworkConfigTemplate 参数，使其指向生成的 .j2 文件：**

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

5. **从您的 network-environment.yaml 文件中删除旧网络配置文件的 resource_registry 映射：**

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
```

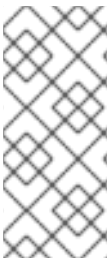
8.2.5.4. Heat 参数到 Ansible 变量映射

NIC 模板文件格式已从 yaml 文件格式改为 Jinja2 ansible 格式 j2，在 Red Hat OpenStack Platform (RHOSP) 17.x 中。

要手动将现有 NIC 模板文件转换为 Jinja2 ansible 格式，您可以将 heat 参数映射到 Ansible 变量，以配置部署中预置备节点的网络属性。如果您运行 openstack overcloud node provision 而无需指定 --network-config 可选参数，您还可以将 heat 参数映射到 Ansible 变量。

例如，如果您当前的部署使用 get_param: InternalApiNetworkVlanID 配置 vlan_id，则将其替换为新 Jinja2 模板中的以下配置：

```
vlan_id: {{ internal_api_vlan_id }}
```



注意

如果您使用 --network-config 可选参数运行 openstack overcloud node provision 来置备节点，则必须使用 overcloud-baremetal-deploy.yaml 中的参数为您的部署配置网络属性。如需更多信息，请参阅 Heat 参数用于置备定义文件映射。

下表列出从 heat vars 到等效的 Ansible 变量的可用映射

表 8.10. 从 heat 参数映射到 Ansible vars

Heat 参数	Ansible 变量
BondInterfaceOvsOptions	**{{ bond_interface_ovs_options }}**

Heat 参数	Ansible 变量
ControlPlaneIp	**{{ ctlplane_ip }}**
ControlPlaneDefaultRoute	**{{ ctlplane_gateway_ip }}**
ControlPlaneMtu	**{{ ctlplane_mtu }}**
ControlPlaneStaticRoutes	**{{ ctlplane_host_routes }}**
ControlPlaneSubnetCidr	**{{ ctlplane_subnet_cidr }}**
DnsSearchDomains	**{{ dns_search_domains }}**
DnsServers	**{{ ctlplane_dns_nameservers }}** **注意** 此 Ansible 变量填充在 **undercloud.conf** 中为 **DEFAULT/undercloud_nameservers** 和 **%SUBNET_SECTION%/dns_nameservers** 配置的 IP 地址。**%SUBNET_SECTION%/dns_nameservers** 的配置会覆盖 **DEFAULT/undercloud_nameservers** 的配置，以便您可以将不同的 DNS 服务器用于 undercloud 和 overcloud，以及不同置备子网中的节点的不同 DNS 服务器。
NumDpdkInterfaceRxQueues	**{{ num_dpdk_interface_rx_queues }}**

配置没有在表中列出的 heat 参数

要配置没有在表中列出的 heat 参数，您必须将参数配置为 {{role.name}}ExtraGroupVars。将参数配置为 {{role.name}}ExtraGroupVars 参数后，您可以在新模板中使用该参数。例如，要配置 StorageSupernet 参数，请在网络配置文件中添加以下配置：

```
parameter_defaults:
  ControllerExtraGroupVars:
    storage_supernet: 172.16.0.0/16
```

然后，您可以将 {{ storage_supernet }} 添加到 Jinja2 模板。

警告

如果将 --network-config 选项与节点置备一起使用，则此过程将无法正常工作。需要自定义 vars 的用户不应使用 --network-config 选项。相反，在创建 Heat 堆栈后，将节点网络配置应用到 config-download ansible 运行。

将 Ansible 变量语法转换为以编程方式配置每个网络

当您使用 Jinja2 for-loop 结构，通过迭代 role_networks 通过迭代 role_networks 来检索可用的网络，您需要检索每个网络角色的小写名称来添加到每个属性的前面。使用以下结构将 Ansible 变量 从上表转换为所需语法：

{{ lookup('vars', networks_lower[network] ~ '_<property>') }}

- **将 <property > 替换为您要设置的属性，如 ip,vlan_id, 或 mtu。**

例如，要动态填充每个 NetworkVlanID 的值，请将 {{ <network_name>_vlan_id }} 替换为以下配置：

```
{{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}`
```

8.2.5.5. 用于置备定义文件映射的 Heat 参数

如果您使用 --network-config 可选参数运行 openstack overcloud node provision 命令置备节点，则必须使用节点定义文件 overcloud-baremetal-deploy.yaml 中的参数为您的部署配置网络属性。

如果您的部署使用预置备节点，您可以将 heat 参数映射到 Ansible 变量，以配置网络属性。如果您运行 openstack overcloud node provision 而无需指定 --network-config 可选参数，您还可以将 heat 参数映射到 Ansible 变量。有关使用 Ansible 变量配置网络属性的更多信息，请参阅 Heat 参数到 Ansible 变量映射。

下表列出了从 heat 参数到节点定义文件 overcloud-baremetal-deploy.yaml 中等同的 network_config 属性的可用映射。

表 8.11. 从 heat 参数映射到节点定义文件 overcloud-baremetal-deploy.yaml

Heat 参数	network_config 属性
BondInterfaceOvsOptions	bond_interface_ovs_options
DnsSearchDomains	dns_search_domains
NetConfigDataLookup	net_config_data_lookup
NeutronPhysicalBridge	physical_bridge_name
NeutronPublicInterface	public_interface_name
NumDpdkInterfaceRxQueues	num_dpdk_interface_rx_queues
{{role.name}}NetworkConfigUpdate	network_config_update

下表列出了从 heat 参数到网络定义文件 network_data.yaml 中等效的属性的可用映射。

表 8.12. 从 heat 参数映射到网络定义文件 network_data.yaml

Heat 参数	IPv4 network_data.yaml property	IPv6 network_data.yaml property
<network_name>IpSubnet	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.1.0/24	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64
<network_name>NetworkVlanID	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>
<network_name>Mtu	- name: <network_name> mtu:	- name: <network_name> mtu:

Heat 参数	IPv4 **network_data.yaml** property	IPv6 **network_data.yaml** property
<network_name>InterfaceDefaultRoute	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.16.0/24 gateway_ip: 172.16.16.1	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64 gateway_ipv6: 2001:db8:a::1
<network_name>InterfaceRoutes	- name: <network_name> subnets: subnet01: ... routes: - destination: 172.18.0.0/24 nexthop: 172.18.1.254	- name: <network_name> subnets: subnet01: ... routes_ipv6: - destination: 2001:db8:b::/64 nexthop: 2001:db8:a::1

8.2.5.6. 对网络数据模式的更改

在 Red Hat OpenStack Platform (RHOSP) 17 中更新了网络数据模式。RHOSP 16 及更早版本中使用的网络数据模式之间的主要区别，以及 RHOSP 17 及之后的版本中使用的网络数据模式，如下所示：

- 基础子网已移到 子网 映射中。这与非路由和路由部署的配置一致，如 spine-leaf 网络。
- enabled 选项不再用于忽略禁用的网络。相反，您必须从配置文件中删除禁用的网络。
- 随着使用它的 heat 资源已被删除，不再需要 compat_name 选项。
- 以下参数不再在网络级别有效：ip_subnet,gateway_ip,allocation_pools,routes,ipv6_subnet,gateway_ipv6,ipv6_allocation_pools, 和 routes_ipv6。这些参数仍然在子网级别使用。
- 引入了一个新的参数 physical_network，用于在 metalsmith 中创建 ironic 端口。
-

新的参数 network_type 和 segmentation_id 替换 {{network.name}}NetValueSpecs，用于将网络类型设置为 vlan。

- RHOSP 17 中弃用了以下参数：
  - {{network.name}}NetCidr
  - {{network.name}}SubnetName
  - {{network.name}}Network
  - {{network.name}}AllocationPools
  - {{network.name}}Routes
  - {{network.name}}SubnetCidr_{{subnet}}
  - {{network.name}}AllocationPools_{{subnet}}
  - {{network.name}}Routes_{{subnet}}

第 9 章 使用 ANSIBLE 配置和管理 RED HAT OPENSTACK PLATFORM

您可以使用 Ansible 配置和管理 overcloud，以及管理容器。

9.1. 基于 ANSIBLE 的 OVERCLOUD 注册

director 使用基于 Ansible 的方法将 overcloud 节点注册到红帽客户门户网站或 Red Hat Satellite Server。

i:f 您使用了之前 Red Hat OpenStack Platform 版本中的 rhel-registration 方法，您必须禁用它并切换到基于 Ansible 的方法。如需更多信息，请参阅 第 9.1.6 节 “切换到 rhsm 可组合服务” 和 第 9.1.7 节 “rhel-registration 到 rhsm 映射”。

除了基于 director 的注册方法外，您还可以在部署后手动注册。如需更多信息，请参阅 第 9.1.9 节 “手动运行基于 Ansible 的注册”。

9.1.1. Red Hat Subscription Manager (RHSM)可组合服务

您可以使用 rhsm 可组合服务通过 Ansible 注册 overcloud 节点。默认 roles_data 文件中的每个角色都包含一个 OS::TripleO::Services::Rhsm 资源，默认情况下是禁用的。要启用该服务，将资源注册到 rhsm 可组合服务文件中：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

rhsm able 服务接受 RhsmVars 参数，您可以使用它来定义与注册相关的多个子参数：

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 9.2
```

您还可以将 RhsmVars 参数与特定于角色的参数结合使用，如 ControllerParameters，在为不同的节

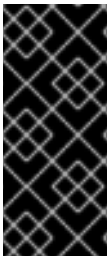
点类型启用特定的存储库时提供灵活性。

RhsmVars sub-parameters

在配置 rhsm 可组合服务时，使用以下子参数作为 RhsmVars 参数的一部分。有关可用的 Ansible 参数的更多信息，请参阅 角色文档。

rhsm	描述
rhsm_method	选择注册方法。门户、satellite 或禁用。
rhsm_org_id	要用于注册的组织。要找到此 ID，请从 undercloud 节点运行 sudo subscription-manager 组织。在提示符处输入您的红帽凭证，并使用生成的 Key 值。有关您的机构 ID 的更多信息，请参阅了解红帽订阅管理机构 ID。
rhsm_pool_ids	要使用的订阅池 ID。如果您不想自动附加订阅，请使用此参数。要找到此 ID，请从 undercloud 节点运行 sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"，并使用生成的 Pool ID 值。
rhsm_activation_key	要用于注册的激活码。
rhsm_autosubscribe	使用此参数自动将兼容订阅附加到这个系统中。将值设为 true 以启用此功能。
rhsm_baseurl	获取内容的基本 URL。默认 URL 是 Red Hat Content Delivery Network。如果使用 Satellite 服务器，请将此值更改为 Satellite 服务器内容存储库的基本 URL。
rhsm_server_hostname	用于注册的订阅管理服务的主机名。默认为 Red Hat Subscription Management 主机名。如果使用 Satellite 服务器，请将此值改为您的 Satellite 服务器主机名。
rhsm_repos	要启用的存储库列表。
rhsm_username	注册的用户名。如果可能，使用激活码进行注册。
rhsm_password	注册密码。如果可能，使用激活码进行注册。
rhsm_release	用于固定存储库的 Red Hat Enterprise Linux 版本。对于 Red Hat OpenStack Platform，这设置为 9.2
rhsm_rhsm_proxy_hostname	HTTP 代理的主机名。例如： proxy.example.com。
rhsm_rhsm_proxy_port	HTTP 代理通信的端口。例如：8080。

rhsm	描述
rhsm_rhsm_proxy_user	用于访问 HTTP 代理的用户名。
rhsm_rhsm_proxy_password	用于访问 HTTP 代理的密码。



重要

只有在 rhsm_method 设置为 portal 时，才能一起使用 rhsm_activation_key 和 rhsm_repos。如果将 rhsm_method 设置为 *satellite*，则只能使用 rhsm_activation_key 或 rhsm_repos。

9.1.2. RhsmVars sub-parameters

在配置 rhsm 可组合服务时，使用以下子参数作为 RhsmVars 参数的一部分。有关可用的 Ansible 参数的更多信息，请参阅 角色文档。

rhsm	描述
rhsm_method	选择注册方法。门户、**satellite** 或 **禁用**。
rhsm_org_id	要用于注册的组织。要找到此 ID，请从 undercloud 节点运行 **sudo subscription-manager** 组织。在提示符处输入您的红帽凭证，并使用生成的 **Key** 值。有关您的机构 ID 的更多信息，请参阅了解红帽订阅管理机构 ID。
rhsm_pool_ids	要使用的订阅池 ID。如果您不想自动附加订阅，请使用此参数。要找到此 ID，请从 undercloud 节点运行 **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"**，并使用生成的 **Pool ID** 值。
rhsm_activation_key	要用于注册的激活码。
rhsm_autosubscribe	使用此参数自动将兼容订阅附加到这个系统中。将值设为 **true** 以启用此功能。
rhsm_baseurl	获取内容的基本 URL。默认 URL 是 Red Hat Content Delivery Network。如果使用 Satellite 服务器，请将此值更改为 Satellite 服务器内容存储库的基本 URL。
rhsm_server_hostname	用于注册的订阅管理服务的主机名。默认为 Red Hat Subscription Management 主机名。如果使用 Satellite 服务器，请将此值改为您的 Satellite 服务器主机名。
rhsm_repos	要启用的存储库列表。

rhsm	描述
rhsm_username	注册的用户名。如果可能，使用激活码进行注册。
rhsm_password	注册密码。如果可能，使用激活码进行注册。
rhsm_release	用于固定存储库的 Red Hat Enterprise Linux 版本。对于 Red Hat OpenStack Platform，这设置为 9.2
rhsm_rhsm_proxy_hostname	HTTP 代理的主机名。例如： **proxy.example.com**。
rhsm_rhsm_proxy_port	HTTP 代理通信的端口。例如：**8080**。
rhsm_rhsm_proxy_user	用于访问 HTTP 代理的用户名。
rhsm_rhsm_proxy_password	用于访问 HTTP 代理的密码。

重要

只有在 rhsm_method 设置为 portal 时，才能一起使用 rhsm_activation_key 和 rhsm_repos。如果将 rhsm_method 设置为 *satellite*，则只能使用 rhsm_activation_key 或 rhsm_repos。

9.1.3. 使用 rhsm 可组合服务注册 overcloud

创建一个启用和配置 rhsm 可组合服务的环境文件。director 使用这个环境文件来注册和订阅您的节点。

流程

1. 创建名为 **templates/rhsm.yml** 的环境文件，以存储配置。

2. 在环境文件中包含您的配置。例如：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
```

```
    - rhel-9-for-x86_64-appstream-eus-rpms
    - rhel-9-for-x86_64-highavailability-eus-rpms
    …
  rhsm_username: "myusername"
  rhsm_password: "p@55w0rd!"
  rhsm_org_id: "1234567"
  rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhsm_method: "portal"
  rhsm_release: 9.2
```

- **resource_registry 部分将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源相关联，该资源可在每个角色上可用。**

- **RhsmVars 变量向 Ansible 传递参数来配置红帽注册。**

3. **要基于每个角色应用 rhsm 可组合服务，请在环境文件中包含您的配置。例如，您可以将不同的配置应用到 Controller 节点、Compute 节点和 Ceph Storage 节点：**

```
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  CephStorageParameters:
```

```
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-rpms
      - rhel-9-for-x86_64-appstream-rpms
      - rhel-9-for-x86_64-highavailability-rpms
      - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

ControllerParameters、ComputeParameters 和 CephStorageParameters 参数各自使用单独的 RhsmVars 参数，将订阅详情传递给对应的角色。

注意

在 CephStorageParameters 参数中设置 RhsmVars 参数，以使用特定于 Ceph Storage 的 Red Hat Ceph Storage 订阅和存储库。确保 rhsm_repos 参数包含标准 Red Hat Enterprise Linux 软件仓库，而不是 Controller 和 Compute 节点所需的延长更新支持(EUS)存储库。

4. **保存环境文件。**

9.1.4. 将 rhsm 可组合服务应用到不同的角色

您可以基于每个角色应用 rhsm 可组合服务。例如，您可以将不同的配置应用到 Controller 节点、Compute 节点和 Ceph Storage 节点。

流程

1. **创建名为 templates/rhsm.yml 的环境文件，以存储配置。**

2. **在环境文件中包含您的配置。例如：**

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
```

```
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
      - rhceph-6-tools-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
    rhsm_method: "portal"
    rhsm_release: 9.2
ComputeParameters:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - rhceph-6-tools-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
    rhsm_method: "portal"
    rhsm_release: 9.2
CephStorageParameters:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-rpms
      - rhel-9-for-x86_64-appstream-rpms
      - rhel-9-for-x86_64-highavailability-rpms
      - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

resource_registry 将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源相关联，该资源可在每个角色上可用。

ControllerParameters、ComputeParameters 和 CephStorageParameters 参数各自使用单独的 RhsmVars 参数，将订阅详情传递给对应的角色。

注意

在 CephStorageParameters 参数中设置 RhsmVars 参数，以使用特定于 Ceph Storage 的 Red Hat Ceph Storage 订阅和存储库。确保 rhsm_repos 参数包含标准 Red Hat Enterprise Linux 软件仓库，而不是 Controller 和 Compute 节点所需的延长更新支持(EUS)存储库。

3. 保存环境文件。

9.1.5. 将 overcloud 注册到 Red Hat Satellite 服务器

创建一个环境文件，启用并配置 rhsm 可组合服务，以将节点注册到 Red Hat Satellite，而不是红帽客户门户。

流程

1. 创建名为 templates/rhsm.yml 的环境文件，以存储配置。

2. 在环境文件中包含您的配置。例如：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 9.2
```

resource_registry 将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源相关联，该资源可在每个角色上可用。

RhsmVars 变量向 Ansible 传递参数来配置红帽注册。

3. 保存环境文件。

9.1.6. 切换到 rhsm 可组合服务

前面的 rhel-registration 方法运行 bash 脚本来处理 overcloud 注册。此方法的脚本和环境文件位于位于 /usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/ 的核心 heat 模板集合中。

完成以下步骤，从 rhel-registration 方法切换到 rhsm 可组合服务。

流程

1. 从将来的部署操作中排除 rhel-registration 环境文件。在大多数情况下，排除以下文件：
   - rhel-registration/environment-rhel-registration.yaml
   - rhel-registration/rhel-registration-resource-registry.yaml
2. 如果您使用自定义 roles_data 文件，请确保 roles_data 文件中的每个角色都包含 OS::TripleO::Services::Rhsm 可组合服务。例如：

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  ...
  ServicesDefault:
    ...
    - OS::TripleO::Services::Rhsm
    ...
```

3. 为 rhsm 可组合服务参数添加环境文件，以备将来部署操作。

此方法将 rhel-registration 参数替换为 rhsm 服务参数，并更改启用该服务的 heat 资源：

```
resource_registry:
  OS::TripleO::NodeExtraConfig: rhel-registration.yaml
```

至：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

您还可以在部署中包含 **/usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml** 环境文件以启用该服务。

9.1.7. rhel-registration 到 rhsm 映射

要帮助将您的详情从 **rhel-registration** 方法转换为 **rhsm** 方法，请使用下表来映射您的参数和值。

rhel-registration	rhsm / RhsmVars
rhel_reg_method	rhsm_method
rhel_reg_org	rhsm_org_id
rhel_reg_pool_id	rhsm_pool_ids
rhel_reg_activation_key	rhsm_activation_key
rhel_reg_auto_attach	rhsm_autosubscribe
rhel_reg_sat_url	rhsm_satellite_url
rhel_reg_repos	rhsm_repos
rhel_reg_user	rhsm_username
rhel_reg_password	rhsm_password
rhel_reg_release	rhsm_release
rhel_reg_http_proxy_host	rhsm_rhsm_proxy_hostname
rhel_reg_http_proxy_port	rhsm_rhsm_proxy_port
rhel_reg_http_proxy_username	rhsm_rhsm_proxy_user
rhel_reg_http_proxy_password	rhsm_rhsm_proxy_password

9.1.8. 使用 rhsm 可组合服务部署 overcloud

使用 rhsm 可组合服务部署 overcloud，以便 Ansible 控制 overcloud 节点的注册过程。

流程

1. 使用 openstack overcloud deploy 命令包含 rhsm.yml 环境文件：

```
openstack overcloud deploy \
  <other cli args> \
  -e ~/templates/rhsm.yaml
```

这将启用 overcloud 和基于 Ansible 的注册的 Ansible 配置。

2. 等待 overcloud 部署完成。

3. 检查 overcloud 节点上的订阅详情。例如，登录到 Controller 节点并运行以下命令：

```
$ sudo subscription-manager status
$ sudo subscription-manager list --consumed
```

9.1.9. 手动运行基于 Ansible 的注册

您可以使用 director 节点上的动态清单脚本对部署的 overcloud 执行基于 Ansible 的手动注册。使用此脚本将节点角色定义为主机组，然后使用 ansible-playbook 对它们运行 playbook。使用以下示例 playbook 来手动注册 Controller 节点。

流程

1. 创建一个使用 redhat_subscription 模块注册节点的 playbook。例如，以下 playbook 适用于 Controller 节点：

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
```

```
    - rhel-9-for-x86_64-highavailability-eus-rpms
    - openstack-17.1-for-rhel-9-x86_64-rpms
    - fast-datapath-for-rhel-9-x86_64-rpms
tasks:
  - name: Register system
    redhat_subscription:
      username: myusername
      password: p@55w0rd!
      org_id: 1234567
      release: 9.2
      pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
  - name: Disable all repos
    command: "subscription-manager repos --disable *"
  - name: Enable Controller node repos
    command: "subscription-manager repos --enable {{ item }}"
    with_items: "{{ repos }}"
```

- **此 play 包含三个任务：**
  - **注册节点。**
  - **禁用任何启用了自动的仓库。**
  - 仅启用与 **Controller** 节点相关的存储库。存储库通过 **repos** 变量列出。

2. **部署 overcloud 后，您可以运行以下命令，以便 Ansible 对您的 overcloud 执行 playbook (ansible-osp-registration.yml)：**

```
$ ansible-playbook -i /usr/bin/tripleo-ansible-inventory ansible-osp-registration.yml
```

这个命令执行以下操作：

- 运行动态清单脚本以获取主机及其组的列表。
- **将 playbook 任务应用到 playbook 的 hosts 参数中定义的组中的节点，本例中为 Controller 组。**

9.2. 使用 ANSIBLE 配置 OVERCLOUD

Ansible 是应用 overcloud 配置的主要方法。本章介绍了有关如何与 overcloud Ansible 配置进行交互的信息。

尽管 director 会自动生成 Ansible playbook，您最好熟悉 Ansible 语法。有关使用 Ansible 的更多信息，请参见 https://docs.ansible.com/。

注意

Ansible 还使用了角色这一概念，但这有别于 OpenStack Platform director 中的角色。Ansible 角色形成了 playbook 的可重复使用的组件，而 director 角色包含 OpenStack 服务到节点类型的映射。

9.2.1. 基于 Ansible 的 overcloud 配置 (config-download)

config-download 功能是 director 用于配置 overcloud 的方法。director 将 config-download 与 OpenStack Orchestration (heat)一起使用来生成软件配置，并将配置应用到每个 overcloud 节点。尽管 heat 从 SoftwareDeployment 资源创建所有部署数据以执行 overcloud 安装和配置，但 heat 并不应用任何配置。Heat 仅通过 heat API 提供配置数据。

作为结果，在运行 openstack overcloud deploy 命令时后会发生以下操作：

- director 根据 openstack-tripleo-heat-templates 创建新的部署计划，并通过包含环境文件和参数来自定义该计划。
- director 使用 heat 来解释部署计划，并创建 overcloud 堆栈和所有下级资源。这包括使用 OpenStack Bare Metal 服务 (ironic) 置备节点。
- Heat 也会从部署计划创建软件配置。director 从这一软件配置中编译 Ansible playbook。
- director 在 overcloud 节点上生成一个临时用户 (tripleo-admin)，专门用于进行 Ansible SSH 访问。
- director 下载 heat 软件配置，并使用 heat 输出生成一系列 Ansible playbook。
- director 通过 ansible-playbook 将 Ansible playbook 应用到 overcloud 节点。

9.2.2. config-download 工作目录

ansible-playbook 命令创建 Ansible 项目目录，默认名称 ~/config-download/overcloud。此项目目录从 heat 存储下载的软件配置。它包括您需要运行 ansible-playbook 来配置 overcloud 的所有与 Ansible 相关的文件。

目录的内容包括：

- tripleo-ansible-inventory.yaml - 包含所有 overcloud 节点的 hosts 和 vars 的 Ansible 清单文件。
- ansible.log - 最近运行的 ansible-playbook 中的日志文件。
- ansible.cfg - 运行 ansible-playbook 时使用的配置文件。
- ansible-playbook-command.sh - 用于重新运行 ansible-playbook 的可执行文件脚本。
- SSH_PRIVATE_KEY - 用于访问 overcloud 节点的私有 ssh 密钥。

1. 重现 ansible-playbook

创建项目目录后，运行 ansible-playbook-command.sh 命令以重现部署。

```
$ ./ansible-playbook-command.sh
```

您可以使用额外的参数运行脚本，如 check mode --check、限制 hosts --limit，以及覆盖变量 -e。

```
$ ./ansible-playbook-command.sh --check
```

9.2.3. 检查 config-download 日志

在 config-download 过程中，Ansible 在 undercloud 上的 /home/stack 目录中创建一个名为 ansible.log 的日志文件。

流程

1. 使用 **less** 命令查看日志：

```
$ less ~/ansible.log
```

9.2.4. 对工作目录执行 Git 操作

config-download 工作目录是本地 **Git** 软件仓库。每次运行部署操作时，**director** 会向工作目录添加一个包含相关更改的 **Git** 提交。可以执行 **Git** 操作以查看不同阶段部署的配置，并将此配置与不同部署进行比较。

请注意工作目录的限制。例如，如果您使用 **Git** 恢复到早期版本的 **config-download** 工作目录，则此操作仅影响工作目录中的配置。不会影响以下配置：

- **overcloud** 数据架构： 应用上一版本的工作目录软件配置不会撤消数据迁移和架构更改。
- **overcloud** 的硬件布局：恢复到之前的软件配置不会撤消与 **overcloud** 硬件相关的更改，如扩展或缩减。
- **heat** 堆栈：恢复到以前版本的工作目录不会影响 **heat** 堆栈中存储的配置。**heat** 堆栈创建软件配置的新版本，并应用到 **overcloud**。要对 **overcloud** 进行永久更改，在重新运行 **openstack overcloud deploy** 之前修改应用到 **overcloud** 堆栈的环境文件。

完成以下步骤，比较 **config-download** 工作目录的不同提交项。

流程

1. 更改到 **overcloud** 的 **config-download** 工作目录，通常称为 **overcloud** ：

```
$ cd ~/config-download/overcloud
```

2. 运行 **git log** 命令，以列出工作目录中的提交。您也可以通过格式化日志输出来显示日期：

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
```

```
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

默认情况下，最近的提交显示在前面。

3. 针对两个提交哈希运行 **git diff**，以查看部署之间的所有变化：

```
$ git diff a7e9063 dfb9d12
```

9.2.5. 使用 config-download 的部署方法

在 overcloud 部署上下文中有四种使用 config-download 的主要方法：

标准部署

运行 **openstack overcloud deploy** 命令，以在置备阶段后自动运行配置阶段。这是运行 **openstack overcloud deploy** 命令时的默认方法。

分离置备和配置

使用特定选项运行 **openstack overcloud deploy** 命令，以分离置备和配置阶段。

部署后运行 ansible-playbook-command.sh 脚本

使用结合或分离的置备和配置阶段运行 **openstack overcloud deploy** 命令，然后运行 **config-download** 工作目录中提供的 **ansible-playbook-command.sh** 脚本，以重新应用配置阶段。

置备节点，手动创建 config-download，并运行 Ansible

使用特定选项运行 **openstack overcloud deploy** 命令，以置备节点，然后使用 **deploy_steps_playbook.yaml playbook** 运行 **ansible-playbook** 命令。

9.2.6. 在标准部署上运行 config-download

执行 **config-download** 的默认方法是运行 **openstack overcloud deploy** 命令。此方法适合大多数环境。

先决条件

- 成功安装 **undercloud**。

- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

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

```
$ source ~/stackrc
```

3. 运行部署命令。包括 overcloud 所需的任何环境文件：

```
$ openstack overcloud deploy \
 --templates \
 -e environment-file1.yaml \
 -e environment-file2.yaml \
 ...
```

4. 等待部署过程完成。

在部署过程中，director 在 ~/config-download/overcloud 工作目录中生成 config-download 文件。在部署过程完成后，查看工作目录中的 Ansible playbook，以查看为配置 overcloud 而执行的任务 director。

9.2.7. 使用分离的置备和配置运行 config-download

openstack overcloud deploy 命令运行基于 heat 的置备过程，然后运行 config-download 配置过程。您还可以运行该部署命令来单独执行每个过程。使用此方法将 overcloud 节点置备为不同的过程，以便您可以在运行 overcloud 配置过程之前在节点上执行任何手动预配置任务。

先决条件

- 成功安装 undercloud。

- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

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

```
$ source ~/stackrc
```

3. 使用 --stack-only 选项运行部署命令。包括 overcloud 所需的任何环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. 等待置备过程完成。
5. 为 tripleo-admin 用户启用从 undercloud 到 overcloud 的 SSH 访问。config-download 进程使用 tripleo-admin 用户来执行基于 Ansible 的配置：

```
$ openstack overcloud admin authorize
```

6. 在节点上执行任何手动预配置任务。如果使用 Ansible 进行配置，请使用 tripleo-admin 用户来访问节点。
7. 使用 --config-download-only 选项运行部署命令。包括 overcloud 所需的环境文件：

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
```

```
-e environment-file2.yaml \
...
--config-download-only
```

8. 等待配置过程完成。

在配置阶段，director 在 ~/config-download/overcloud 工作目录中生成 config-download 文件。在部署过程完成后，查看工作目录中的 Ansible playbook，以查看为配置 overcloud 而执行的任务 director。

9.2.8. 使用 ansible-playbook-command.sh 脚本运行 config-download

当您使用标准方法或单独的置备和配置过程部署 overcloud 时，director 在 ~/config-download/overcloud 中生成工作目录。此目录包含再次运行配置过程所需的 playbook 和脚本。

先决条件

- 使用以下方法之一部署的 overcloud：
  - 结合置备和配置过程的标准方法。
  - 分隔置备和配置过程。

流程

1. 以 stack 用户身份登录 undercloud 主机。
2. 运行 ansible-playbook-command.sh 脚本。

可以将额外的 Ansible 参数传递给该脚本，再将其原封不动地传递给 ansible-playbook 命令。这样便可利用 Ansible 功能，如检查模式(--check)、限制主机(--limit)或覆盖变量(-e)。例如：

```
$ ./ansible-playbook-command.sh --limit Controller
```

警告

当 --limit 用于大规模部署时，只有执行中包含的主机才会添加到节点的 SSH known_hosts 文件中。因此，一些操作（如实时迁移）可能无法在 known_hosts 文件中没有工作。

注意

要确保 /etc/hosts 文件在所有节点上是最新的，请以 stack 用户身份运行以下命令：

```
(undercloud)$ cd /home/stack/overcloud-deploy/overcloud/config-download/overcloud
(undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible allovercloud \
  -i /home/stack/overcloud-deploy/overcloud/tripleo-ansible-inventory.yaml \
  -m include_role \
  -a name=tripleo_hosts_entries \
  -e @global_vars.yaml
```

3. **等待配置过程完成。**

其他信息

- **这个工作目录中包含一个名为 deploy_steps_playbook.yaml 的 playbook，用于管理 overcloud 配置任务。要查看此 playbook，请运行以下命令：**

```
$ less deploy_steps_playbook.yaml
```

这个 playbook 会使用工作目录中所含的各种任务文件。某些任务文件是所有 OpenStack 平台角色通用的，某些任务文件则特定于某些 OpenStack 平台角色和服务器。

- **这个工作目录中还包含与您在 overcloud 的 roles_data 文件中定义的各个角色相对应的子目录。例如：**

```
$ ls Controller/
```

每个 OpenStack 平台角色目录中还包含相应角色类型的各个服务器的子目录。这些目录采用可组合角色主机名格式：

```
$ ls Controller/overcloud-controller-0
```

- deploy_steps_playbook.yaml 中的 Ansible 任务已标记。要查看完整的标记列表，在 ansible-playbook 中使用 CLI 选项 --list-tags：

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags
deploy_steps_playbook.yaml
```

然后，在 ansible-playbook-command.sh 脚本中通过 --tags、--skip-tags 或 --start-at-task 来应用已标记的配置：

```
$ ./ansible-playbook-command.sh --tags overcloud
```

4. 对 overcloud 运行 config-download playbook 时，可能会收到有关每个主机的 SSH 指纹的消息。要避免这些消息，请在运行 ansible-playbook-command.sh 脚本时包含 --ssh-common-args="-o StrictHostKeyChecking=no"：

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

9.2.9. 使用手动创建的 playbook 运行 config-download

您可以在标准工作流之外创建自己的 config-download 文件。例如，您可以使用 --stack-only 选项运行 openstack overcloud deploy 命令以置备节点，然后单独手动应用 Ansible 配置。

先决条件

- 成功安装 undercloud。
- overcloud 节点已准备好进行部署。
- 与特定 overcloud 自定义相关的 heat 环境文件。

步骤

1. 以 stack 用户身份登录 undercloud 主机。

2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 使用 --stack-only 选项运行部署命令。包括 overcloud 所需的环境文件：

```
$ openstack overcloud deploy \
 --templates \
 -e environment-file1.yaml \
 -e environment-file2.yaml \
 ...
 --stack-only
```

4. 等待置备过程完成。

5. 为 tripleo-admin 用户启用从 undercloud 到 overcloud 的 SSH 访问。config-download 进程使用 tripleo-admin 用户来执行基于 Ansible 的配置：

```
$ openstack overcloud admin authorize
```

6. 生成 config-download 文件：

```
$ openstack overcloud deploy \
 --stack overcloud --stack-only \
 --config-dir  ~/overcloud-deploy/overcloud/config-download/overcloud/
```

   - --stack 指定 overcloud 的名称。
   - --stack-only 确保命令仅部署 heat 堆栈并跳过任何软件配置。
   - --config-dir 指定 config-download 文件的位置。

7. 切换到包含 config-download 文件的目录：

```
$ cd ~/config-download
```

8. **生成静态清单文件：**

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user tripleo-admin \
  --static-yaml-inventory inventory.yaml
```

- **用您的 overcloud 的名称替换 <overcloud>。**

9. **使用 ~/overcloud-deploy/overcloud/config-download/overcloud 文件和静态清单文件来执行配置。要执行部署 playbook，请运行 ansible-playbook 命令：**

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/overcloud-deploy/overcloud/config-download/overcloud/deploy_steps_playbook.yaml
```

注意

针对 overcloud 运行 config-download/overcloud playbook 时，您可能会收到有关每个主机的 SSH 指纹的消息。要避免这些消息，请在 ansible-playbook 命令中包含 --ssh-common-args="-o StrictHostKeyChecking=no" ：

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

10. **等待配置过程完成。**

11

11. **从基于 ansible 的配置手动生成 overcloudrc 文件：**

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. **手动将部署状态设置为成功：**

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<overcloud>"}'
```

- **用您的 overcloud 的名称替换 <overcloud>。**

注意

~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ 目录包含一个名为 **deploy_steps_playbook.yaml** 的 **playbook**。这个 **playbook** 会使用工作目录中所含的各种任务文件。有些任务文件适用于所有 **Red Hat OpenStack Platform (RHOSP)**角色，有些任务文件特定于特定的 **RHOSP** 角色和服务器。

~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ 目录还包含与您在 **overcloud roles_data** 文件中定义的各个角色对应的子目录。每个 **RHOSP** 角色目录还包含该角色类型的单个服务器的子目录。这些目录使用可组合角色主机名格式，如 **Controller**/**overcloud-controller-0**。

deploy_steps_playbook.yaml 中的 **Ansible** 任务已标记。要查看完整的标记列表，在 **ansible-playbook** 中使用 **CLI** 选项 **--list-tags** ：

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags
deploy_steps_playbook.yaml
```

您可以使用 **--tags**、**--skip-tags** 或 **--start-at-task** 和 **ansible-playbook-command.sh** 脚本应用标记的配置：

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

9.2.10. config-download 的限制

config-download 功能有一些限制：

- 在使用 **ansible-playbook CLI** 参数（如 **--tags**、**--skip-tags** 或 **--start-at-task**）时，请勿随意更改部署配置的运行或应用顺序。利用这些 **CLI** 参数，可以轻松地重新运行先前失败的任务或在初始部署的基础上进行迭代。但是，为了保证部署的一致性，您必须通过 **deploy_steps_playbook.yaml** 依序运行所有任务。
- 您不能将 **--start-at-task** 参数用于在任务名称中使用变量的某些任务。例如，**--start-at-task** 参数不适用于以下 **Ansible** 任务：

```
- name: Run puppet host configuration for step {{ step }}
```

- 如果您的 overcloud 部署包含 director 部署的 Ceph Storage 集群，则在使用 --check 选项时您无法跳过 step1 任务，除非您也跳过 external_deploy_steps 任务。

- 您可以使用 --forks 选项设置并行 Ansible 任务的数量。但是，在 25 个并行任务后 config-download 操作的性能会下降。因此，--forks 选项请勿超过 25。

9.2.11. config-download 顶层文件

以下文件是 config-download 工作目录内的重要顶级文件。

Ansible 配置和执行。

以下文件专用于在 config-download 工作目录中配置和执行 Ansible。

ansible.cfg

运行 ansible-playbook 时所使用的配置文件。

ansible.log

来自最近一次 ansible-playbook 运行的日志文件。

ansible-errors.json

包含任何部署错误的 JSON 结构文件。

ansible-playbook-command.sh

从上次部署操作重新运行 ansible-playbook 命令的可执行脚本。

ssh_private_key

Ansible 用于访问 overcloud 节点的 SSH 私钥。

tripleo-ansible-inventory.yaml

包含所有 overcloud 节点的主机和变量的 Ansible 清单文件。

overcloud-config.tar.gz

工作目录的存档。

Playbook

下列文件是 config-download 工作目录中的 playbook。

deploy_steps_playbook.yaml

主要的部署步骤。此 playbook 为您的 overcloud 执行主要的配置操作。

pre_upgrade_rolling_steps_playbook.yaml

主要升级的升级前步骤

upgrade_steps_playbook.yaml

主要升级步骤。

post_upgrade_steps_playbook.yaml

主要升级的升级后步骤。

update_steps_playbook.yaml

次要更新步骤。

fast_forward_upgrade_playbook.yaml

快进升级任务。仅在要从一个长生命版本的 Red Hat OpenStack Platform 升级到下一个版本时使用此 playbook。

9.2.12. config-download 标签

playbook 使用标记的任务控制其应用到 overcloud 的任务。使用包含 ansible-playbook CLI 参数的标签 --tags 或 --skip-tags 控制要执行的任务。以下列表包含有关默认启用的标签的信息：

facts

fact 收集操作。

common_roles

适用于所有节点的 Ansible 角色。

overcloud

用于 overcloud 部署的所有 play。

pre_deploy_steps

在 deploy_steps 操作之前发生的部署。

host_prep_steps

主机准备步骤。

deploy_steps

部署步骤。

post_deploy_steps

在 deploy_steps 操作后进行的步骤。

external

所有外部部署任务。

external_deploy_steps

仅在 undercloud 中运行的外部部署任务。

9.2.13. config-download 部署步骤

deploy_steps_playbook.yaml playbook 配置 overcloud。此 playbook 应用所有必需的软件配置，以基于 overcloud 部署计划部署完整的 overcloud。

本节包含此 playbook 内使用的不同 Ansible play 的总结。本节中的 play 名称是 playbook 中使用的相同名称，也显示在 ansible-playbook 输出中。本节还包含有关在每个 play 上设置的 Ansible 标签的信息。

从 undercloud 收集事实

为 undercloud 节点收集的 fact。

标签： facts

从 overcloud 收集 fact

为 **overcloud** 节点收集的 **fact**。

标签： **facts**

加载全局变量

从 **global_vars.yaml** 加载所有变量。

标签： **always**

TripleO 服务器的通用角色

将常见 **Ansible** 角色应用于所有 **overcloud** 节点，包括用于安装 **bootstrap** 软件包的 **tripleo-bootstrap** 和用于配置 **ssh** 已知主机的 **tripleo-ssh-known-hosts**。

标签： **common_roles**

Overcloud 部署步骤任务 - 步骤 0

从 **deploy_steps_tasks** 模板接口应用任务。

标签： **overcloud**、**deploy_steps**

服务器部署

应用特定于服务器的 **heat** 部署，以进行网络和基础架构数据等配置。包括 **NetworkDeployment**、**<Role>Deployment**、**<Role>AllNodesDeployment**，等等。

标签： **overcloud**、**pre_deploy_steps**

主机准备步骤

从 **host_prep_steps** 模板接口应用任务。

标签： **overcloud**、**host_prep_steps**

外部部署步骤 [1,2,3,4,5]

应用来自 external_deploy_steps_tasks 模板接口的任务。Ansible 仅针对 undercloud 节点运行这些任务。

标签： external、external_deploy_steps

Overcloud 部署步骤任务 [1,2,3,4,5]

从 deploy_steps_tasks 模板接口应用任务。

标签： overcloud、deploy_steps

Overcloud 通用部署步骤任务 [1,2,3,4,5]

应用每个步骤执行的常见任务，包括 puppet 主机配置 container-puppet.py 和 tripleo-container-manage （容器配置和管理）。

标签： overcloud、deploy_steps

部署后服务器

5 步部署过程后执行的配置应用特定于服务器的 heat 部署。

标签： overcloud、post_deploy_steps

部署之后的外部部署任务

应用来自 external_post_deploy_steps_tasks 模板接口的任务。Ansible 仅针对 undercloud 节点运行这些任务。

标签： external、external_deploy_steps

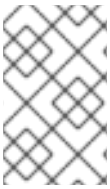
9.3. 使用 ANSIBLE 管理容器

Red Hat OpenStack Platform 17.1 使用 tripleo_container_manage Ansible 角色对容器执行管理操作。您还可以编写自定义 playbook 来执行特定的容器管理操作：

- 收集 heat 生成的容器配置数据。tripleo_container_manage 角色使用此数据来编配容器部署。

- 启动容器。
- 停止容器。
- 更新容器。
- 删除容器。
- 使用特定配置运行容器。

虽然 director 会自动执行容器管理，但您可能想要自定义容器配置，或者在不重新部署 overcloud 的情况下对容器应用热修复。



注意

此角色仅支持 Podman 容器管理。

9.3.1. tripleo-container-manage 角色默认值和变量

以下摘录显示了 tripleo_container_manage Ansible 角色的默认值和变量。

```
# All variables intended for modification should place placed in this file.
tripleo_container_manage_hide_sensitive_logs: '{{ hide_sensitive_logs | default(true)
  }}'
tripleo_container_manage_debug: '{{ ((ansible_verbosity | int) >= 2) | bool }}'
tripleo_container_manage_clean_orphans: true

# All variables within this role should have a prefix of "tripleo_container_manage"
tripleo_container_manage_check_puppet_config: false
tripleo_container_manage_cli: podman
tripleo_container_manage_concurrency: 1
tripleo_container_manage_config: /var/lib/tripleo-config/
tripleo_container_manage_config_id: tripleo
tripleo_container_manage_config_overrides: {}
tripleo_container_manage_config_patterns: '*.json'
# Some containers where Puppet is run, can take up to 10 minutes to finish
# in slow environments.
```

```
tripleo_container_manage_create_retries: 120
# Default delay is 5s so 120 retries makes a timeout of 10 minutes which is
# what we have observed a necessary value for nova and neutron db-sync execs.
tripleo_container_manage_exec_retries: 120
tripleo_container_manage_healthcheck_disabled: false
tripleo_container_manage_log_path: /var/log/containers/stdouts
tripleo_container_manage_systemd_teardown: true
```

9.3.2. tripleo-container-manage molecule 场景

molecule 用于测试 tripleo_container_manage 角色。下面显示了一个 molecule 默认清单：

```
hosts:
  all:
    hosts:
      instance:
        ansible_host: localhost
        ansible_connection: local
        ansible_distribution: centos8
```

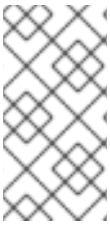
使用方法

Red Hat OpenStack 17.1 仅支持此角色中的 Podman。Docker 支持采用路线图。

Molecule Ansible 角色执行以下任务：

- **收集由 TripleO Heat 模板生成的容器配置数据。此数据被用作数据源。如果容器已经由 Molecule 管理，无论其存在状态，配置数据将根据需要重新配置容器。**
- **管理 systemd 关闭文件。它创建 TripleO Container systemd 服务，在关闭或启动节点时为服务排序是必需的。它还管理 netns-placeholder 服务。**
- **删除不需要或需要重新配置的容器。它使用名为 needs_delete （） 的自定义过滤器，其包含一组规则来确定容器是否需要删除。**
  - **如果容器没有由 tripleo_ansible 管理，则容器不会被删除，或者容器 config_id 与输入 ID 不匹配。**
  - **如果容器没有 config_data，或者容器没有与输入中数据不匹配的 config_data，则会删除容器。请注意，当删除容器时，该角色还会禁用并删除 systemd 服务和 healtchecks。**

- 按照 start_order 容器配置定义的特定顺序创建容器，默认值为 0。
  - 如果容器是 exec，则将运行一个 execs 的专用 playbook，使用 async，以便可以同时运行多个 execs。
  - 否则，会使用 podman_container 来创建容器。如果容器有一个 重启策略，则会配置systemd 服务。如果容器有一个 healthcheck 脚本，则会配置 systemd healthcheck 服务。



注意

tripleo_container_manage_concurrency 参数默认设置为 1，并且放置值高于 2 可能会公开 Podman 锁定的问题。

playbook 示例：

```
- name: Manage step_1 containers using tripleo-ansible
  block:
    - name: "Manage containers for step 1 with tripleo-ansible"
      include_role:
        name: tripleo_container_manage
      vars:
        tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
        tripleo_container_manage_config_id: "tripleo_step1"
```

9.3.3. tripleo_container_manage 角色变量

tripleo_container_manage Ansible 角色包含以下变量：

表 9.1. 角色变量

名称	默认值	描述
tripleo_container_manage_check_puppet_config	false	如果您希望 Ansible 检查 Puppet 容器配置，则使用此变量。Ansible 可使用配置 hash 来识别更新的容器配置。如果容器有一个来自 Puppet 的新配置，请将此变量设置为 **true**，以便 Ansible 能够检测到新配置，并将容器添加到 Ansible 必须重启的容器列表中。

名称	默认值	描述
tripleo_container_manage_cli	podman	使用此变量设置要用于管理容器的命令行界面。**tripleo_container_manage** 角色仅支持 Podman。
tripleo_container_manage_concurrency	1	使用此变量设置要同时管理的容器数量。
tripleo_container_manage_config	/var/lib/tripleo-config/	使用此变量设置容器配置目录的路径。
tripleo_container_manage_config_id	tripleo	使用此变量设置具体配置步骤的 ID。例如，将此值设置为 **tripleo_step2**，以管理部署的第二步的容器。
tripleo_container_manage_config_patterns	*.json	使用此变量设置用于标识容器配置目录中配置文件的 bash 正则表达式。
tripleo_container_manage_debug	false	使用此变量启用或禁用调试模式。如果要运行带有特定一次性配置的容器，以调试模式运行 **tripleo_container_manage** 角色，以输出管理容器生命周期的容器命令，或运行 no-op 容器管理操作进行测试和验证。
tripleo_container_manage_healthcheck_disable	false	使用此变量启用或禁用健康检查。
tripleo_container_manage_log_path	/var/log/containers/stdouts	使用此变量为容器设置 stdout 日志路径。
tripleo_container_manage_systemd_order	false	使用此变量启用或禁用 Ansible 的 systemd 关闭顺序。
tripleo_container_manage_systemd_teardown	true	使用此变量触发已弃用容器的清理。
tripleo_container_manage_config_overrides	{}	使用此变量覆盖任何容器配置。此变量的值来自一个字典，其中每个键都是容器名称和您要覆盖的参数，如容器镜像或用户。此变量不会将自定义覆盖写入 JSON 容器配置文件，并且任何新的容器部署、更新或升级都会恢复到 JSON 配置文件的内容。

名称	默认值	描述
tripleo_container_manage_valid_exit_code	[]	使用此变量检查容器是否返回退出代码。这个值必须是列表，例如 **[0，3]**。

9.3.4. tripleo-container-manage healthchecks

在 Red Hat OpenStack 17.1 之前，容器健康检查是由 systemd 计时器实现的，该计时器将运行 podman exec 来确定给定容器是否健康。现在，它使用 Podman 中的原生健康检查接口，这有助于集成和使用。

要检查容器（如 keystone）是否健康，请运行以下命令：

```
$ sudo podman healthcheck run keystone
```

返回代码应当为 0 和 "healthy "。

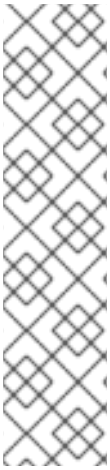
```
"Healthcheck": {
    "Status": "healthy",
    "FailingStreak": 0,
    "Log": [
        {
            "Start": "2020-04-14T18:48:57.272180578Z",
            "End": "2020-04-14T18:48:57.806659104Z",
            "ExitCode": 0,
            "Output": ""
        },
        (...)
    ]
}
```

9.3.5. tripleo-container-manage debug

tripleo_container_manage Ansible 角色允许您对给定容器执行特定的操作。这可用于：

- 使用特定的一次性配置运行容器。
- 输出容器命令以管理容器生命周期。

- 输出 **Ansible** 对容器所做的更改。



注意

要管理单个容器，您需要了解两个方面：

- **overcloud** 部署过程中的步骤是容器部署的。
- 包含容器配置的生成的 **JSON** 文件的名称。

以下是在第 **1** 步中 管理 **HAproxy** 容器的 **playbook** 示例，可覆盖镜像设置：

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo_container_manage
          vars:
            tripleo_container_manage_config_patterns: 'haproxy.json'
            tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
            tripleo_container_manage_config_id: "tripleo_step1"
            tripleo_container_manage_clean_orphans: false
            tripleo_container_manage_config_overrides:
              haproxy:
                image: quay.io/tripleoRed_Hat_OpenStack_Platform-17.1-
Customizing_your_Red_Hat_OpenStack_Platform_deployment.entos9/centos-binary-haproxy:hotfix
```

如果 **Ansible** 以检查模式运行，则不会删除或创建容器，但 **playbook** 结束时会显示一个命令列表来显示 **playbook** 的可能结果。这对于调试非常有用。

```
$ ansible-playbook haproxy.yaml --check
```

添加 **diff** 模式 将显示 **Ansible** 对容器所做的更改。

```
$ ansible-playbook haproxy.yaml --check --diff
```

tripleo_container_manage_clean_orphans 参数是可选的。它可以设置为 false 表示孤立的容器（带有特定 config_id ）不会被删除。它可用于管理单个容器，而不影响具有相同 config_id 的其他正在运行的容器。

tripleo_container_manage_config_overrides 参数是可选的，可用于覆盖特定的容器属性，如 image 或 container 用户。参数使用容器名称和要覆盖的参数创建字典。这些参数必须存在，并在 TripleO Heat 模板中定义容器配置。

请注意，字典不会更新 JSON 文件中的覆盖，以便在执行更新或升级时，容器将与 JSON 文件中的配置重新配置。

第 10 章 使用编排服务(HEAT)配置 OVERCLOUD.

您可以使用编排服务(heat)在 heat 模板和环境文件中创建自定义 overcloud 配置。

10.1. 了解 HEAT 模板

本指南中的自定义配置使用 heat 模板和环境文件来定义 overcloud 的某些方面。本章介绍了 heat 模板，以便您可以了解 Red Hat OpenStack Platform director 中这些模板的结构和格式。

10.1.1. Heat 模板

director 使用 Heat 编配模板(HOT)作为 overcloud 部署计划的模板格式。metastore 格式的模板通常以 YAML 格式表示。模板的目的是定义和创建堆栈，这是 OpenStack Orchestration (heat)创建的资源集合，以及资源的配置。资源是 Red Hat OpenStack Platform (RHOSP)中的对象，可以包含计算资源、网络配置、安全组、扩展规则和自定义资源。

heat 模板包含三个主要部分：

parameters

这些设置是传递给 heat 的设置，它提供了一种自定义堆栈的方法，以及用于不带传递值的参数的任何默认值。这些设置在 模板的 parameter 部分中定义。

资源

使用 resources 部分定义资源，如计算实例、网络和存储卷，您可以在使用此模板部署堆栈时创建这些资源。Red Hat OpenStack Platform (RHOSP)包含一组跨越所有组件的核心资源。这些是要作为堆栈的一部分创建和配置的具体对象。RHOSP 包含一组核心资源，它们跨越所有组件。它们在模板的 resources 部分中定义。

输出

使用 outputs 部分声明云用户可以在堆栈创建后访问的输出参数。您的云用户可以使用这些参数来请求有关堆栈的详细信息，如部署的实例的 IP 地址或作为堆栈一部分部署的 Web 应用程序的 URL。

基本 heat 模板示例：

```
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 ] }
```

此模板使用资源类型 类型：**OS::Nova::Server** 创建名为 **my_instance** 的实例，该实例具有云用户指定的特定类别、镜像和密钥。堆栈可以返回 **instance_name** 的值，名为 **My Cirros Instance**。

当 **heat** 处理模板时，它会为模板创建堆栈，并为资源模板创建一组子堆栈。这会创建您通过模板定义的主堆栈中解掉的堆栈层次结构。您可以使用以下命令查看堆栈层次结构：

```
$ openstack stack list --nested
```

10.1.2. 环境文件

环境文件是特殊的模板，可用于自定义 **heat** 模板。除了核心 **heat** 模板外，您还可以在部署命令中包含环境文件。环境文件包含三个主要部分：

resource_registry

本节定义自定义资源名称，链接到其他 **heat** 模板。这提供了一种创建核心资源集合中不存在的自定义资源的方法。

parameters

这些是应用于顶级模板参数的通用设置。例如，如果您有一个部署嵌套堆栈的模板，如资源 registry 映射，则参数仅适用于顶级模板，而不是应用到嵌套资源的模板。

parameter_defaults

这些参数修改所有模板中参数的默认值。例如，如果您有一个部署嵌套堆栈的 heat 模板，如资源 registry 映射，参数默认为所有模板。

重要

在为 overcloud 创建自定义环境文件时，使用 parameter_defaults 而不是 parameters，以便您的参数应用到 overcloud 的所有堆栈模板。

基本环境文件示例：

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

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

从特定 heat 模板(my_template.yaml)创建堆栈时，可能会包含此环境文件 (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，它为此资源类型提供实施，可覆盖任何内置文件。您可以在 my_template.yaml 文件中包含 OS::Nova::Server::MyServer 资源。

MyIP 仅将参数应用于使用此环境文件部署的主 heat 模板。在本例中，MyIP 仅适用于 my_template.yaml 中的参数。

networkName 应用到主 heat 模板 my_template.yaml，以及与主模板中包含的资源关联的模板，如 OS::Nova::Server::MyServer 资源及其 myserver.yaml 模板。

注意

要使 RHOSP 将 heat 模板文件用作自定义模板资源，文件扩展名必须为 .yaml 或 .template。

10.1.3. 核心 overcloud heat 模板

director 包含 overcloud 的核心 heat 模板集合和环境文件集合。此集合存储在 /usr/share/openstack-tripleo-heat-templates 中。

此模板集合中的主文件和目录有：

overcloud.j2.yaml

这是 director 用于创建 overcloud 环境的主要模板文件。此文件使用 Jinja2 语法来迭代模板中的某些部分，以创建自定义角色。Jinja2 格式在 overcloud 部署期间呈现为 YAML。

overcloud-resource-registry-puppet.j2.yaml

这是 director 用于创建 overcloud 环境的主要环境文件。它为 overcloud 镜像中存储的 Puppet 模块提供一组配置。在将 overcloud 镜像写入每个节点后，heat 会使用此环境文件中注册的资源启动每个节点的 Puppet 配置。此文件使用 Jinja2 语法来迭代模板中的某些部分，以创建自定义角色。Jinja2 格式在 overcloud 部署期间呈现为 YAML。

roles_data.yaml

此文件包含 overcloud 中角色的定义，并将服务映射到每个角色。

network_data.yaml

此文件包含 overcloud 中网络的定义，以及它们的属性，如子网、分配池和 VIP 状态。默认 network_data.yaml 文件包含默认网络： External, Internal Api, Storage, Storage Management, Tenant, 和 Management。您可以创建自定义 network_data.yaml 文件，并使用 -n 选项将其添加到 openstack overcloud deploy 命令中。

plan-environment.yaml

此文件包含 overcloud 计划的元数据定义。这包括要应用到 overcloud 的计划名称、要使用的主模板和环境文件。

capabilities-map.yaml

此文件包含 overcloud 计划的环境文件映射。

部署

此目录包含 heat 模板。overcloud-resource-registry-puppet.j2.yaml 环境文件使用此目录中的文件来驱动每个节点上的 Puppet 配置应用。

environments

此目录包含额外的 heat 环境文件，可用于创建 overcloud。这些环境文件为您的生成的 Red

Hat OpenStack Platform (RHOSP)环境启用额外的功能。例如，该目录包含一个环境文件，用于启用 Cinder NetApp 后端存储(cinder-netapp-config.yaml)。

network

此目录包含一组 heat 模板，可用于创建隔离的网络和端口。

puppet

此目录包含控制 Puppet 配置的模板。overcloud-resource-registry-puppet.j2.yaml 环境文件使用此目录中的文件来驱动每个节点上的 Puppet 配置应用。

puppet/services

此目录包含所有服务配置的传统 heat 模板。部署 目录中的模板替换 puppet/services 目录中的大多数模板。

extraconfig

此目录包含可用于启用额外功能的模板。

10.1.4. 在 overcloud 创建中包含环境文件

在部署命令中包括环境文件，并使用 -e 选项。您可以根据需要纳入多个环境文件。但是，环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。例如，您有两个环境文件，其中包含一个通用资源类型 OS::TripleO::NodeExtraConfigPost，以及一个通用参数 TimeZone ：

environment-file-1.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

parameter_defaults:
  RabbitFDLimit: 65536
  TimeZone: 'Japan'
```

environment-file-2.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-2.yaml
```

```
parameter_defaults:
  TimeZone: 'Hongkong'
```

在部署命令中包含这两个环境文件：

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

openstack overcloud deploy 命令通过以下进程运行：

1. 从核心 **heat** 模板集合中加载默认配置。
2. 应用 **environment-file-1.yaml** 的配置，它会覆盖默认配置中的任何通用设置。
3. 应用 **environment-file-2.yaml** 的配置，它会覆盖来自默认配置和 **environment-file-1.yaml** 的任何通用设置。

这会对 **overcloud** 的默认配置进行以下更改：

- **OS::TripleO::NodeExtraConfigPost** 资源设置为 **/home/stack/templates/template-2.yaml**，如 **environment-file-2.yaml** 中定义的。
- **timezone** 参数设置为 **Hongkong**，如 **environment-file-2.yaml** 中定义的。
- **RabbitFDLimit** 参数设为 **65536**，如 **environment-file-1.yaml** 中定义的。**environment-file-2.yaml** 不会更改此值。

您可以使用此机制为 **overcloud** 定义自定义配置，而无需多个环境文件中的值冲突。

10.1.5. 使用自定义核心 heat 模板

在创建 overcloud 时，director 使用位于 /usr/share/openstack-tripleo-heat-templates 中的核心 heat 模板。如果要自定义此核心模板集合，请使用以下 Git 工作流来管理您的自定义模板集合：

流程

- 创建包含 heat 模板集合的初始 Git 存储库：

a. 将模板集合复制到 /home/stack/templates 目录中：

```
$ cd ~/templates
$ cp -r /usr/share/openstack-tripleo-heat-templates .
```

b. 进入自定义模板目录并初始化 Git 存储库：

```
$ cd ~/templates/openstack-tripleo-heat-templates
$ git init .
```

c. 配置 Git 用户名和电子邮件地址：

```
$ git config --global user.name "<USER_NAME>"
$ git config --global user.email "<EMAIL_ADDRESS>"
```

- 将 <USER_NAME > 替换为您要使用的用户名。
- 将 <EMAIL_ADDRESS > 替换为您的电子邮件地址。

a. 为初始提交暂存所有模板：

```
$ git add *
```

b. 创建初始提交：

```
$ git commit -m "Initial creation of custom core heat templates"
```

这将创建一个包含最新核心模板集合的初始 master 分支。使用此分支作为自定义分支的基础，并将新模板版本合并到此分支中。

- 使用自定义分支将您的更改保存到核心模板集合。使用以下步骤创建 **my-customizations** 分支并添加自定义：

  a. 创建 **my-customizations** 分支并切换到它：

  ```
  $ git checkout -b my-customizations
  ```

  b. 编辑自定义分支中的文件。

  c. 暂存 **git** 中的更改：

  ```
  $ git add [edited files]
  ```

  d. 将更改提交到自定义分支：

  ```
  $ git commit -m "[Commit message for custom changes]"
  ```

  这会将您的更改作为提交添加到 **my-customizations** 分支。当 **master** 分支更新时，您可以将 **my-customizations** 从 **master** 进行更新，这会导致 **git** 将这些提交添加到更新的模板集合中。这有助于跟踪您的自定义信息，并在将来的模板更新中重新显示它们。

- 更新 **undercloud** 时，**openstack-tripleo-heat-templates** 软件包也可能收到更新。当发生这种情况时，还必须更新自定义模板集合：

  a. 将 **openstack-tripleo-heat-templates** 软件包版本保存为环境变量：

  ```
  $ export PACKAGE=$(rpm -qv openstack-tripleo-heat-templates)
  ```

  b. 进入模板集合目录并为更新的模板创建新分支：

  ```
  $ cd ~/templates/openstack-tripleo-heat-templates
  $ git checkout -b $PACKAGE
  ```

c. 删除分支中的所有文件，并将其替换为新版本：

```
$ git rm -rf *
$ cp -r /usr/share/openstack-tripleo-heat-templates/* .
```

d. 为初始提交添加所有模板：

```
$ git add *
```

e. 为软件包更新创建提交：

```
$ git commit -m "Updates for $PACKAGE"
```

f. 将分支合并到 master 中。如果使用 Git 管理系统（如 GitLab），请使用管理工作流。如果您在本地使用 git，切换到 master 分支并运行 git merge 命令：

```
$ git checkout master
$ git merge $PACKAGE
```

master 分支现在包含核心模板集合的最新版本。现在，您可以从这个更新的集合中重新构建 my-customization 分支。

- 更新 my-customization 分支：

a. 进入 my-customizations 分支：

```
$ git checkout my-customizations
```

b. 将分支更新为 master ：

```
$ git rebase master
```

这将更新 my-customizations 分支，并重播为此分支发出的自定义提交。

- **解决在 rebase 过程中发生的任何冲突：**

  a. **检查哪些文件包含冲突：**

  ```
  $ git status
  ```

  b. **解决确定的模板文件冲突。**

  c. **添加解析的文件：**

  ```
  $ git add [resolved files]
  ```

  d. **继续 rebase：**

  ```
  $ git rebase --continue
  ```

- **部署自定义模板集合：**

  a. **确保您已切换到 my-customization 分支：**

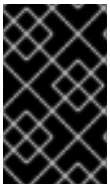
  ```
  git checkout my-customizations
  ```

  b. **使用 --templates 选项运行 openstack overcloud deploy 命令来指定您的本地目录：**

  ```
  $ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
  ```

注意

如果您指定了不带目录的 --templates 选项，director 将使用默认模板目录 (/usr/share/openstack-tripleo-heat-templates)。



重要

红帽建议在 第 10.3 节 “配置 hook” 中使用方法，而不是修改 heat 模板集合。

10.1.6. Jinja2 渲染

/usr/share/openstack-tripleo-heat-templates 中的核心 heat 模板包含多个具有 j2.yaml 文件扩展名的文件。这些文件包含 Jinja2 模板语法，director 会将这些文件呈现成其具有 .yaml 扩展名的静态 heat 模板。例如，主 overcloud.j2.yaml 文件呈现到 overcloud.yaml 中。director 使用生成的 overcloud.yaml 文件。

启用 Jinja2 的 heat 模板使用 Jinja2 语法来创建用于迭代值的参数和资源。例如，overcloud.j2.yaml 文件包含以下代码片段：

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

当 director 呈现 Jinja2 语法时，director 会迭代 roles_data.yaml 文件中定义的角色，并使用角色的名称填充 {{role.name}}Count 参数。默认 roles_data.yaml 文件包含五个角色，并从我们的示例中生成以下参数：

- ControllerCount
- ComputeCount
- BlockStorageCount
- ObjectStorageCount
- CephStorageCount

参数渲染版本示例类似如下：

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

director 仅从核心 heat 模板的 目录内呈现启用了 Jinja2 的模板和环境文件。以下用例演示了呈现 Jinja2 模板的正确方法。

使用案例 1：默认核心模板

模板目录： /usr/share/openstack-tripleo-heat-templates/

环境文件： /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.j2.yaml

director 使用默认核心模板位置(--templates)，并将 enable-internal-tls.j2.yaml 文件呈现到 enable-internal-tls.yaml 中。运行 openstack overcloud deploy 命令时，请使用 -e 选项包括渲染的 enable-internal-tls.yaml 文件的名称。

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

使用案例 2：自定义核心模板

模板目录： /home/stack/tripleo-heat-installer-templates

环境文件： /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

director 使用自定义核心模板位置(--templates /home/stack/tripleo-heat-templates)，director 会将自定义核心模板中的 enable-internal-tls.j2.yaml 文件呈现到 enable-internal-tls.yaml 中。运行 openstack overcloud deploy 命令时，请使用 -e 选项包括渲染的 enable-internal-tls.yaml 文件的名称。

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
  -e /home/stack/tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
  ...
```

使用案例 3：不正确使用

模板目录： /usr/share/openstack-tripleo-heat-templates/

环境文件： /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

director 使用自定义核心模板位置(--templates /home/stack/tripleo-heat-installer-templates)。但是，所选的 enable-internal-tls.j2.yaml 不在自定义核心模板中，因此它不会呈现到 enable-internal-tls.yaml 中。这会导致部署失败。

将 Jinja2 语法处理到静态路由中

使用 process-templates.py 脚本将 openstack-tripleo-heat-templates 的 Jinja2 语法呈现到一组静态路由中。要使用 process-templates.py 脚本呈现 openstack-tripleo-heat-templates 集合的副本，请切换到 openstack-tripleo-heat-templates 目录：

```
$ cd /usr/share/openstack-tripleo-heat-templates
```

运行位于 工具 目录中的 process-templates.py 脚本，以及 -o 选项来定义自定义目录来保存静态副本：

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

这会将所有 Jinja2 模板转换为呈现的 YAML 版本，并将结果保存到 ~/openstack-tripleo-heat-templates-rendered。

10.2. HEAT 参数

director 模板集合中的每个 heat 模板都包含一个 parameters 部分。本节包含特定于特定 overcloud 服务的所有参数的定义。这包括以下内容：

- **overcloud.j2.yaml - 默认基本参数**

- roles_data.yaml - 可组合角色的默认参数
- deploymentAttr.yaml - 特定服务的默认参数

您可以使用以下方法修改这些参数的值：

1. 为您的自定义参数创建一个环境文件。
2. 在环境文件的 parameter_defaults 部分中包含您的自定义参数。
3. 使用 openstack overcloud deploy 命令包含环境文件。

10.2.1. 示例 1：配置时区

用于设置时区的 Heat 模板(puppet/services/time/timezone.yaml)包含 TimeZone 参数。如果将 TimeZone 参数留空，overcloud 会将时间设置为 UTC 作为默认值。

要获取时区列表，请运行 timedatectl list-timezones 命令。以下示例命令检索 Asia 的时区：

```
$ sudo timedatectl list-timezones|grep "Asia"
```

在识别您的时区后，在环境文件中设置 *TimeZone* 参数。以下示例环境文件将 *TimeZone* 的值设置为 ***Asia/Tokyo*** ：

```
parameter_defaults:
  TimeZone: 'Asia/Tokyo'
```

10.2.2. 示例 2：配置 RabbitMQ 文件描述符限制

对于某些配置，您可能需要提高 RabbitMQ 服务器的文件描述符限制。使用 deployment/rabbitmq/rabbitmq-container-puppet.yaml heat 模板在 RabbitFDLimit 参数中设置新限制。在环境文件中添加以下条目：

```
parameter_defaults:
  RabbitFDLimit: 65536
```

10.2.3. 示例 3：启用和禁用参数

您可能需要在部署期间初始设置参数，然后禁用用于将来的部署操作的参数，如更新或扩展操作。例如，要在 overcloud 创建过程中包含自定义 RPM，请在环境文件中包含以下条目：

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

要从未来的部署中禁用此参数，无法删除该参数。相反，您必须将参数设置为空值：

```
parameter_defaults:
  DeployArtifactURLs: []
```

这可确保不再为后续部署操作设置该参数。

10.2.4. 示例 4：基于角色的参数

使用 [ROLE]Parameters 参数，将 [ROLE] 替换为可组合角色，以为特定角色设置参数。

例如，director 在 Controller 和 Compute 节点上配置 sshd。要为 Controller 和 Compute 节点设置不同的 sshd 参数，请创建一个环境文件，其中包含 ControllerParameters 和 ComputeParameters 参数，并为每个特定的角色设置 sshd 参数：

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

10.2.5. 识别您要修改的参数

Red Hat OpenStack Platform director 为配置提供了许多参数。在某些情况下，您可能会遇到识别您要配置的特定选项以及对应的 director 参数的难度。如果有一个选项您要使用 director 配置，请使用以下工作流来识别选项并将其映射到特定的 overcloud 参数：

1. 确定您要配置的选项。记录下使用 选项的服务。

2. 为此选项检查对应的 Puppet 模块。Red Hat OpenStack Platform 的 Puppet 模块位于 director 节点上的 /etc/puppet/modules 下。每个模块对应于特定的服务。例如，keystone 模块对应于 OpenStack Identity (keystone)。

   - 如果 Puppet 模块包含控制所选选项的变量，请继续下一步。
   - 如果 Puppet 模块不包含控制所选选项的变量，则此选项不存在 hieradata。如果可能，您可以在 overcloud 完成部署后手动设置选项。

3. 以 hieradata 的形式检查 Puppet 变量的核心 heat 模板集合。deployment Attr 中的模板通常与同一服务的 Puppet 模块对应。例如，deployment/keystone/keystone-container-puppet.yaml 模板为 keystone 模块提供 hieradata。

   - 如果 heat 模板为 Puppet 变量设置 hieradata，该模板也应披露您可以修改的基于 director 的参数。
   - 如果 heat 模板没有为 Puppet 变量设置 hieradata，请使用配置 hook 来使用环境文件传递 hieradata。有关自定义 hieradata 的更多信息，请参阅 第 10.3.4 节 “puppet：为角色自定义 hieradata”。

流程

1. 要更改 OpenStack Identity (keystone)的通知格式，请使用工作流并完成以下步骤：

   a. 识别您要配置的 OpenStack 参数(notification_format)。

   b. 在 keystone Puppet 模块中搜索 notification_format 设置：

   ```
   $ grep notification_format /etc/puppet/modules/keystone/manifests/*
   ```

   在这种情况下，keystone 模块使用 keystone::notification_format 变量管理这个选项。

   c. 为这个变量搜索 keystone 服务模板：

```
$ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-
templates/deployment/keystone/keystone-container-puppet.yaml
```

输出显示 director 使用 KeystoneNotificationFormat 参数来设置 keystone::notification_format hieradata。

下表显示了最终映射：

director 参数	Puppet hieradata	OpenStack Identity (keystone) 选项
KeystoneNotificationFormat	keystone::notification_format	notification_format

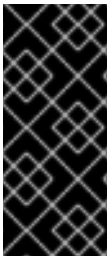
您可以在 overcloud 环境文件中设置 KeystoneNotificationFormat，然后在 overcloud 配置期间设置 keystone.conf 文件中的 notification_format 选项。

10.3. 配置 HOOK

使用配置 hook 将您自己的自定义配置功能注入 overcloud 部署过程。您可以创建 hook，以在主 overcloud 服务配置前后或之后注入自定义配置，以及用于修改和包含基于 Puppet 的配置的 hook。

10.3.1. 预配置：自定义特定的 overcloud 角色

overcloud 使用 Puppet 进行 OpenStack 组件的核心配置。director 提供了一组 hook，可用于在核心配置开始前为特定节点角色执行自定义配置。这些 hook 包括以下配置：



重要

此文档的早期版本使用 OS::TripleO::Tasks::*PreConfig 资源来为每个角色提供预配置 hook。heat 模板集合要求专用使用这些 hook，这意味着您不应该将它们用于自定义用途。反之，使用此处概述的 OS::TripleO::*ExtraConfigPre hook。

OS::TripleO::ControllerExtraConfigPre

在核心 Puppet 配置之前，应用到 Controller 节点的其他配置。

OS::TripleO::ComputeExtraConfigPre

在 Puppet 核心配置之前，应用到 Compute 节点的额外配置。

OS::TripleO::CephStorageExtraConfigPre

在核心 Puppet 配置前，应用到 Ceph Storage 节点的额外配置。

OS::TripleO::ObjectStorageExtraConfigPre

在 Puppet 核心配置之前，应用到对象存储节点的额外配置。

OS::TripleO::BlockStorageExtraConfigPre

在核心 Puppet 配置前，应用到块存储节点的额外配置。

OS::TripleO::[ROLE]ExtraConfigPre

在核心 Puppet 配置之前，应用到自定义节点的额外配置。将 [ROLE] 替换为可组合角色名称。

在本例中，使用变量 nameserver 在特定角色的所有节点上附加 resolv.conf 文件：

流程

1. 创建一个基本的 heat 模板 ~/templates/nameserver.yaml，该脚本将变量名称服务器写入节点的 resolv.conf 文件中：

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

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

resources:
  CustomExtraConfigPre:
    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}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

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

在本例中，resource 部分包含以下参数：

CustomExtraConfigPre

这定义了软件配置。在本例中，我们定义 **Bash** 脚本，**heat** 将 **_NAMESERVER_IP_** 替换为 **nameserver_ip** 参数中存储的值。

CustomExtraDeploymentPre

这会执行软件配置，这是 **CustomExtraConfigPre** 资源的软件配置。注意以下几点：

- **config** 参数会引用 **CustomExtraConfigPre** 资源，以便 **heat** 知道要应用的配置。
- **server** 参数检索 **overcloud** 节点的映射。此参数由父模板提供，并在此 **hook** 模板中强制使用。
- **actions** 参数定义何时应用配置。在这种情况下，您希望在创建 **overcloud** 时应用配置。可能的操作包括 **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** 和 **RESUME**。
- **input_values** 包含一个名为 **deploy_identifier** 的参数，它存储父模板中的 **DeployIdentifier**。此参数为每个部署更新提供资源的时间戳，以确保后续 **overcloud** 更新中的资源恢复。

2. 创建一个环境文件 **~/templates/pre_config.yaml**，将 **heat** 模板注册到基于角色的资源类

型。例如，要将配置应用到 Controller 节点，请使用 ControllerExtraConfigPre hook：

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. 将环境文件添加到堆栈中，以及其他环境文件：

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

这会将配置应用到所有 Controller 节点，然后再在初始 overcloud 创建或后续更新开始。

重要

每个 hook 只能将每个资源注册到一个 heat 模板。后续用法会覆盖要使用的 heat 模板。

10.3.2. 预配置：自定义所有 overcloud 角色

overcloud 使用 Puppet 进行 OpenStack 组件的核心配置。director 提供了一个 hook，可用于在核心配置开始前配置所有节点类型：

OS::TripleO::NodeExtraConfig

在核心 Puppet 配置之前，应用到所有节点角色的额外配置。

在本例中，使用变量 nameserver 在每个节点上附加 resolv.conf 文件：

流程

1. 创建运行脚本的基本 heat 模板 ~/templates/nameserver.yaml，以使用变量 nameserver 附加每个节点的 resolv.conf 文件：

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

```
description: >
  Extra hostname configuration

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

resources:
  CustomExtraConfigPre:
    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}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

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

在本例中，resource 部分包含以下参数：

CustomExtraConfigPre

此参数定义一个软件配置。在本例中，您将定义 Bash 脚本，heat 将 _NAMESERVER_IP_ 替换为 nameserver_ip 参数中存储的值。

CustomExtraDeploymentPre

此参数执行软件配置，这是 CustomExtraConfigPre 资源的软件配置。注意以下几点：

- **config 参数会引用 CustomExtraConfigPre 资源，以便 heat 知道要应用的配**

置。

- server 参数检索 overcloud 节点的映射。此参数由父模板提供，并在此 hook 模板中强制使用。

- actions 参数定义何时应用配置。在这种情况下，您仅在创建 overcloud 时应用配置。可能的操作包括 CREATE、UPDATE、DELETE、SUSPEND 和 RESUME。

- input_values 参数包含一个名为 deploy_identifier 的子参数，它存储了父模板中的 DeployIdentifier。此参数为每个部署更新提供资源的时间戳，以确保后续 overcloud 更新中的资源恢复。

2. 创建一个环境文件 ~/templates/pre_config.yaml，将 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
```

3. 将环境文件添加到堆栈中，以及其他环境文件：

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

这会将配置应用到所有节点，然后再在初始 overcloud 创建或后续更新开始。

重要

您可以将 OS::TripleO::NodeExtraConfig 注册到一个 heat 模板。后续用法会覆盖要使用的 heat 模板。

10.3.3. Post-configuration：自定义所有 overcloud 角色

重要

此文档的早期版本使用 OS::TripleO::Tasks::*PostConfig 资源来为每个角色提供后配置 hook。heat 模板集合要求专用使用这些 hook，这意味着您不应该将它们用于自定义用途。反之，使用此处概述的 OS::TripleO::NodeExtraConfigPost hook。

在完成 overcloud 创建但您希望在初始创建或后续 overcloud 更新时向所有角色添加额外的配置的情况。在这种情况下，使用以下后配置 hook：

OS::TripleO::NodeExtraConfigPost

在核心 Puppet 配置后，应用到所有节点角色的额外配置。

在本例中，使用变量 nameserver 在每个节点上附加 resolv.conf 文件：

流程

1. **创建运行脚本的基本 heat 模板 ~/templates/nameserver.yaml，以使用变量 nameserver 附加每个节点的 resolv.conf 文件：**

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    type: json

resources:
  CustomExtraConfig:
    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}

    CustomExtraDeployments:
      type: OS::Heat::SoftwareDeploymentGroup
      properties:
        servers:  {get_param: servers}
        config: {get_resource: CustomExtraConfig}
        actions: ['CREATE']
        input_values:
          deploy_identifier: {get_param: DeployIdentifier}
```

在本例中，resource 部分包含以下参数：

CustomExtraConfig

这定义了软件配置。在本例中，您将定义 **Bash** 脚本，**heat** 将 **_NAMESERVER_IP_** 替换为 **nameserver_ip** 参数中存储的值。

CustomExtraDeployments

这会执行软件配置，这是 **CustomExtraConfig** 资源的软件配置。注意以下几点：

- **config** 参数会引用 **CustomExtraConfig** 资源，以便 **heat** 知道要应用的配置。
- **servers** 参数检索 **overcloud** 节点的映射。此参数由父模板提供，并在此 **hook** 模板中强制使用。
- **actions** 参数定义何时应用配置。在这种情况下，您希望在创建 **overcloud** 时应用配置。可能的操作包括 **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** 和 **RESUME**。
- **input_values** 包含一个名为 **deploy_identifier** 的参数，它存储父模板中的 **DeployIdentifier**。此参数为每个部署更新提供资源的时间戳，以确保后续 **overcloud** 更新中的资源恢复。

2. 创建一个环境文件 **~/templates/post_config.yaml**，它将 **heat** 模板注册为 **OS::TripleO::NodeExtraConfigPost:** 资源类型。

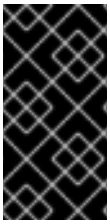
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml
```

```
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. 将环境文件添加到堆栈中，以及其他环境文件：

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

这会在核心配置在初始 overcloud 创建或后续更新后将配置应用到所有节点。



重要

您可以将 OS::TripleO::NodeExtraConfigPost 注册到一个 heat 模板。后续用法会覆盖要使用的 heat 模板。

10.3.4. puppet：为角色自定义 hieradata

heat 模板集合包含一组参数，可用于将额外的配置传递给某些节点类型。这些参数将配置保存为节点上的 Puppet 配置的 hieradata：

ControllerExtraConfig

添加至所有 Controller 节点的配置。

ComputeExtraConfig

添加至所有 Compute 节点的配置。

BlockStorageExtraConfig

添加至所有块存储节点的配置。

ObjectStorageExtraConfig

添加至所有对象存储节点的配置。

CephStorageExtraConfig

添加至所有 Ceph Storage 节点的配置。

[ROLE]ExtraConfig

配置以添加到可组合角色。将 [ROLE] 替换为可组合角色名称。

ExtraConfig

配置以添加到所有节点。

流程

1. 要在部署后配置过程中添加额外的配置，请在 parameter_defaults 部分中创建一个包含这些参数的环境文件。例如，要将 Compute 主机的保留内存增加到 1024 MB，并将 VNC 密钥映射设置为日语，请在 ComputeExtraConfig 参数中使用以下条目：

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

2. 将此环境文件包含在 openstack overcloud deploy 命令中，以及与部署相关的任何其他环境文件。

重要

您只能定义每个参数一次。后续用法会覆盖前面的值。

10.3.5. puppet：为单个节点自定义 hieradata

您可以使用 heat 模板集合为各个节点设置 Puppet hieradata：

流程

1. 从节点的内省数据识别系统 UUID：

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 |
jq .extra.system.product.uuid
```

这个命令返回一个系统 UUID。例如：

```
"f5055c6c-477f-47fb-afe5-95c6928c407f"
```

2. 创建一个环境文件来定义特定于节点的 **hieradata**，并将 **per_node.yaml** 模板注册到预配置 **hook**。在 **NodeDataLookup** 参数中包含您要配置的节点的系统 **UUID**：

```
resource_registry:
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/pre_deploy/per_node.yaml
parameter_defaults:
  NodeDataLookup: '{"f5055c6c-477f-47fb-afe5-95c6928c407f":
{"nova::compute::vcpu_pin_set": [ "2", "3" ]}}'
```

3. 将此环境文件包含在 **openstack overcloud deploy** 命令中，以及与部署相关的任何其他环境文件。

per_node.yaml 模板在节点上生成一组 **hieradata** 文件，它们对应于每个系统 **UUID**，并包含您定义的 **hieradata**。如果未定义 **UUID**，则生成的 **hieradata** 文件为空。在本例中，**per_node.yaml** 模板在所有 **Compute** 节点上运行，由 **OS::TripleO::ComputeExtraConfigPre hook** 定义，但只有具有系统 **UUID f5055c6c-477f-47fb-afe5-95c6928c407f** 接收 **hieradata** 的 **Compute** 节点。

您可以使用此机制根据特定要求定制每个节点。

10.3.6. Puppet：应用自定义清单

在某些情况下，您可能想要在 **overcloud** 节点上安装和配置一些额外的组件。您可以在主配置完成后使用应用到节点的自定义 **Puppet** 清单来实现此目的。作为基本示例，您可能想要在每个节点上安装 **motd**

流程

1. 创建一个 **heat** 模板 **~/templates/custom_puppet_config.yaml**，它将启动 **Puppet** 配置。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    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::SoftwareDeploymentGroup
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
```

本例包括模板中的 /home/stack/templates/motd.pp，并将其传递给配置的节点。motd.pp 文件包含安装和配置 motd 所需的 Puppet 类。

2. **创建一个环境文件 ~templates/puppet_post_config.yaml，将 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
```

3. **将此环境文件包含在 openstack overcloud deploy 命令中，以及与部署相关的任何其他环境文件。**

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

这会将 motd.pp 的配置应用到 overcloud 中的所有节点。