Red Hat OpenStack Platform 16.2

高级 Overcloud 自定义

使用 Red Hat OpenStack Platform director 配置高级功能的方法

Last Updated: 2024-05-21

Red Hat OpenStack Platform 16.2 高级 Overcloud 自定义

使用 Red Hat OpenStack Platform director 配置高级功能的方法

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

使用 Red Hat OpenStack Platform director 为 Red Hat OpenStack Platform (RHOSP)企业环境配置特定的高级功能。这包括网络隔离、存储配置、SSL 通信和常规配置方法等功能。

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

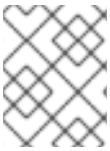
使用 Create Issue 表单对文档提供反馈。JIRA 问题将在 Red Hat OpenStack Platform Jira 项目中创建，您可以在其中跟踪您的反馈进度。

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 OVERCLOUD 配置简介

Red Hat OpenStack Platform (RHOSP) director 提供了一组工具，可用于置备和创建功能齐全的 OpenStack 环境，也称为 overcloud。*Director 安装和使用指南涵盖了* 基本 overcloud 的准备和配置。但是，生产级别 overcloud 可能需要额外的配置：

- 将 overcloud 整合到现有网络基础架构的基本网络配置。
- 独立 VLAN 上的网络流量隔离某些 OpenStack 网络流量类型。
- SSL 配置来保护公共端点的通信
- 存储选项，如 NFS、iSCSI、Red Hat Ceph Storage 以及多个第三方存储设备。
- Red Hat Content Delivery Network 节点注册，或使用您的内部 Red Hat Satellite 5 或 6 服务器注册。
- 各种系统级别选项。
- 各种 OpenStack 服务选项。



注意

本指南中的示例是配置 overcloud 的可选步骤。只有在您想要为 overcloud 提供额外功能时才需要这些步骤。使用适用于您的环境要求的步骤。

第 2 章 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 模板的基本介绍，以便您可以在 Red Hat OpenStack Platform director 的上下文中了解这些模板的结构和格式。

2.1. HEAT 模板

director 使用 Heat 编配模板（过期）作为 overcloud 部署计划的模板格式。过程格式的模板通常以 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 模板有三个主要部分：

parameters

这些设置为 heat，它提供了一种自定义堆栈的方法，以及参数的默认值，而无需传递值。这些设置在**模板的** parameter 部分中定义。

资源

使用 **resources** 部分定义使用此模板部署堆栈时创建的资源，如计算实例、网络和存储卷。Red Hat OpenStack Platform (RHOSP)包含所有组件中的一组核心资源。这些是作为堆栈的一部分创建和配置的特定对象。

输出

使用 **outputs** 部分声明您的云用户在创建堆栈后可以访问的输出参数。您的云用户可以使用这些参数来请求堆栈的详细信息，如部署的实例的 IP 地址，或者作为堆栈的一部分部署的 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 **type: OS::Nova::Server** 创建名为 **my_instance** 的实例，其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，它名为 **My Cirros Instance**。

当 heat 处理模板时，它会从模板创建堆栈，并为资源模板创建一组子堆栈。这会创建一个堆栈层次结构，该层次结构从您使用模板定义的主堆栈中生成。您可以使用以下命令查看堆栈层次结构：

```
$ openstack stack list --nested
```

2.2. 环境文件

环境文件是一种特殊的模板，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。这提供了一种创建在核心资源集合中不存在的自定义资源的方法。

parameters

这些是您应用到顶级模板的参数的常见设置。例如，如果您有一个模板来部署嵌套堆栈，如资源 registry 映射，则参数仅适用于顶级模板，而不适用于嵌套资源的模板。

parameter_defaults

这些参数修改所有模板中的参数的默认值。例如，如果您有一个 heat 模板来部署嵌套堆栈，如资源 registry 映射，则参数默认为适用于所有模板。

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
```

从特定的 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了一个实现，可覆盖任何内置资源。您可以在 **my_template.yaml** 文件中包含 **OS::Nova::Server::MyServer** 资源。

MyIP 仅将参数应用到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 适用于主 heat 模板 **my_template.yaml**，以及与主模板中包含的资源关联的模板，如本例中的 **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，文件扩展名必须是 .yaml 或 .template。

2.3. 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。

此模板集合中的主要文件和目录是：

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 语法迭代模板中的某些部分以创建自定义角色。在 overcloud 部署过程中将 Jinja2 格式呈现为 YAML。

overcloud-resource-registry-puppet.j2.yaml

这是 director 用于创建 overcloud 环境的主要环境文件。它为存储在 overcloud 镜像上的 Puppet 模块提供一组配置。在 director 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 语法迭代模板中的某些部分以创建自定义角色。在 overcloud 部署过程中将 Jinja2 格式呈现为 YAML。

roles_data.yaml

此文件包含 overcloud 中角色的定义，并将服务映射到各个角色。

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

此文件包含您的 overcloud 计划元数据的定义。这包括要使用的计划名称、主模板和要应用到 overcloud 的环境文件。

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驱动每个节点上的 Puppet 配置应用。

environments

此目录包含可用于创建 overcloud 的其他 heat 环境文件。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，以启用 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** 目录中的模板替换了 **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。

firstboot

此目录包含 director 最初创建节点时使用的 **first_boot** 脚本示例。

2.4. 规划环境元数据

您可以在计划环境元数据文件中为您的 overcloud 计划定义元数据。director 在 overcloud 创建过程中应用元数据，并在导入和导出 overcloud 计划时应用元数据。

使用计划环境文件来定义 director 可以通过 OpenStack Workflow (Mistral)服务执行的工作流。计划环境元数据文件包括以下参数：

version

模板的版本。

name

用于存储计划文件的 OpenStack Object Storage (swift)中的 overcloud 计划名称和容器的名称。

模板

要用于 overcloud 部署的核心父模板。这通常是 **overcloud.yaml**，它是 **overcloud.yaml.j2** 模板的呈现版本。

environments

定义您要使用的环境文件列表。使用 **路径** 子参数指定每个环境文件的名称和相对位置。

parameter_defaults

要在 overcloud 中使用的一组参数。这个功能与标准环境文件中的 **parameter_defaults** 部分相同。

密码

要用于 overcloud 密码的一组参数。这个功能与标准环境文件中的 **parameter_defaults** 部分相同。通常，director 会自动使用随机生成的密码填充此部分。

workflow_parameters

使用此参数为 OpenStack Workflow (mistral)命名空间提供一组参数。您可以使用它来计算和自动生成某些 overcloud 参数。

以下片段是计划环境文件语法的示例：

```
version: 1.0
name: myovercloud
description: 'My Overcloud Plan'
template: overcloud.yaml
environments:
- path: overcloud-resource-registry-puppet.yaml
- path: environments/containers-default-parameters.yaml
- path: user-environment.yaml
parameter_defaults:
  ControllerCount: 1
  ComputeCount: 1
```

```
  OvercloudComputeFlavor: compute
  OvercloudControllerFlavor: control
workflow_parameters:
  tripleo.derive_params.v1.derive_parameters:
    num_phy_cores_per_numa_node_for_pmd: 2
```

您可以使用 **-p** 选项通过 **openstack overcloud deploy** 命令包括计划环境元数据文件：

```
(undercloud) $ openstack overcloud deploy --templates \
  -p /my-plan-environment.yaml \
  [OTHER OPTIONS]
```

您还可以使用以下命令查看现有 overcloud 计划的计划元数据：

```
(undercloud) $ openstack object save overcloud plan-environment.yaml --file -
```

2.5. 在 OVERCLOUD 创建中包含环境文件

在部署命令中使用 **-e** 选项包括环境文件。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 命令通过以下过程运行：

1. 从核心 heat 模板集合中加载默认配置。
2. 应用 **environment-file-1.yaml** 中的配置，这将覆盖默认配置中的任何常见设置。
3. 应用 **environment-file-2.yaml** 中的配置，该配置会覆盖默认配置和 **environment-file-1.yaml** 中的所有常用设置。

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** 参数设置为在 environment-file-2.yaml 中定义，如 **environment-file-2.yaml** 中定义的。
- **RabbitFDLimit** 参数设置为 **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，而无需与多个环境文件冲突。

2.6. 使用自定义核心 HEAT 模板

在创建 overcloud 时，director 会使用位于 /usr/share/openstack-tripleo-heat-templates 中的一组核心 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。

d. 为初始提交暂存所有模板：

```
$ git add *
```

e. 创建初始提交：

```
$ git commit -m "Initial creation of custom core heat templates"
```

这将创建一个初始 master 分支，其中包含最新的核心模板集合。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存储在核心模板集合中。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. 在 git 中暂存更改：

```
$ 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，您可以对 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 软件包可能还会接收更新。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更新 **my-customization** 分支。

- 更新 **my-customization** 分支：

a. 进入 **my-customizations** 分支：

```
$ git checkout my-customizations
```

b. 对 **master** 进行分支更新：

```
$ git rebase master
```

这会更新 **my-customizations** 分支，并重播为此分支进行的自定义提交。

- 解决更新过程中发生的任何冲突：

a. 检查哪些文件包含冲突：

```
$ git status
```

b. 解决标识的模板文件冲突。

c. 添加解析的文件：

```
$ git add [resolved files]
```

d. 继续更新：

```
$ git rebase --continue
```

- 部署自定义模板集合：

a. 确保您已切换到 **my-customization** 分支：

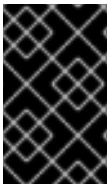
```
git checkout my-customizations
```

b. 使用 **--templates** 选项运行 **openstack overcloud deploy** 命令以指定您的本地模板目录：

```
$ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
```

注意

如果您指定了 **--templates** 选项，**director** 将使用默认模板目录 **(/usr/share/openstack-tripleo-heat-templates)**。



重要

红帽建议使用 **第 4 章 *配置 hook*** 中的方法，而不是修改 **heat** 模板集合。

2.7. 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具有 **.yaml** 扩展的静态 **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。

Jinja2-enabled heat 模板使用 **Jinja2** 语法为迭代值创建参数和资源。例如，**overcloud.j2.yaml** 文件包含以下片断：

```
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，并使用角色名称填充 **{{role.name}}Count** 参数。默认 **roles_data.yaml** 文件包含五个角色，并从示例中生成以下参数：

- **ControllerCount**
- **ComputeCount**
- **BlockStorageCount**

- **ObjectStorageCount**
- **CephStorageCount**

参数的呈现版本示例如下：

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

director 仅从核心 heat 模板的目录中呈现 Jinja2-enabled 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/network-isolation.j2.yaml

director 使用默认核心模板位置(--templates)，并将 network-isolation.j2.yaml 文件呈现到 network-isolation.yaml 中。运行 openstack overcloud deploy 命令时，请使用 -e 选项包含渲染的 network-isolation.yaml 文件的名称。

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

使用案例 2：自定义核心模板

模板目录： /home/stack/tripleo-heat-templates

环境文件： /home/stack/tripleo-heat-templates/environments/network-isolation.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 network-isolation.j2.yaml 文件呈现到 network-isolation.yaml 中。运行 openstack overcloud deploy 命令时，请使用 -e 选项包含渲染的 network-isolation.yaml 文件的名称。

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

使用案例 3：不正确的用法

模板目录： /usr/share/openstack-tripleo-heat-templates/

环境文件： /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml

director 使用默认的核心模板位置(--templates /usr/share/openstack-tripleo-heat-templates)。但是，所选 network-isolation.j2.yaml 不在自定义核心模板中，因此它不会呈现到 network-isolation.yaml 中。这会导致部署失败。

将 Jinja2 语法处理为静态模板

使用 process-templates.py 脚本将 openstack-tripleo-heat-templates 的 Jinja2 语法呈现到一组静态模板中。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 tools 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 模板转换为其渲染的 YAML 版本，并将结果保存到 ~/openstack-tripleo-heat-templates-rendered。

第 3 章 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
- deployment114.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 命令包括环境文件。

3.1. 示例 1：配置时区

用于设置时区(puppet/services/time/timezone.yaml)的 Heat 模板包含一个 TimeZone 参数。如果将 TimeZone 参数留空，则 overcloud 会将 time 设为 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"
```

找到时区后，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'
```

-

3.2. 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
```

3.3. 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，然后为将来的部署操作（如 updates 或 scaling 操作）禁用该参数。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"]
```

要从将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。

3.4. 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，并为每个特定角色设置 sshd 参数：

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

3.5. 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，您可能会遇到识别要配置的特定选项以及对应的 director 参数的难度。如果要使用 director 配置某个选项，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。每个模块都对应于特定的服务。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。部署中的模板通常与 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 变量设置了 hieradata，则模板还应关闭您可以修改的基于 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 4.5 节 “puppet：自定义角色的 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 变量管理此选项。

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

您在 overcloud 环境文件中设置 KeystoneNotificationFormat，然后在 overcloud 配置期间在 keystone.conf 文件中设置 notification_format 选项。

第 4 章 配置 HOOK

使用配置 hook 将您自己的自定义配置功能注入 overcloud 部署过程中。您可以创建 hook 在主 overcloud 服务配置之前和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。

4.1. 第一次引导：自定义第一次引导配置

director 使用 cloud-init 在初始创建 overcloud 后在所有节点上执行配置。您可以使用 NodeUserData 资源类型调用 cloud-init。

OS::TripleO::NodeUserData

应用到所有节点的 cloud-init 配置。

OS::TripleO::Controller::NodeUserData

应用到 Controller 节点的 cloud-init 配置。

OS::TripleO::Compute::NodeUserData

应用到 Compute 节点的 cloud-init 配置。

OS::TripleO::CephStorage::NodeUserData

应用到 Ceph Storage 节点的 cloud-init 配置。

OS::TripleO::ObjectStorage::NodeUserData

应用到 Object Storage 节点的 cloud-init 配置。

OS::TripleO::BlockStorage::NodeUserData

应用到块存储节点的 cloud-init 配置。

OS::TripleO::[ROLE]::NodeUserData

应用到自定义节点的 cloud-init 配置。将 [ROLE] 替换为可组合角色名称。

在本例中，使用所有节点上的自定义 IP 地址更新名称服务器：

流程

1.

创建一个基本的 heat 模板 ~/templates/nameserver.yaml，它将运行脚本，以在每个节点上附加带有特定名称服务器的 resolv.conf 文件。您可以使用 OS::TripleO::MultipartMime 资源类型来发送配置脚本。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

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

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

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

2. 创建一个环境文件 ~/templates/firstboot.yaml，它将 heat 模板注册为 OS::TripleO::NodeUserData 资源类型。

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

3. 要在 overcloud 中添加第一次引导配置，请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/firstboot.yaml \
    ...
```

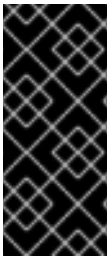
这会在首次创建和第一次引导时将配置添加到所有节点中。后续包含这些模板（如更新 overcloud 堆栈）不会运行这些脚本。

重要

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

4.2. 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，可用于在第一次引导后和核心配置开始前用于为特定节点角色执行自定义配置。这些 hook 包括：



重要

本文档的早期版本使用 OS::TripleO::Tasks:: thePreConfig 资源来基于每个角色提供预配置 hook。heat 模板集合需要专用使用这些 hook，这意味着您不应该使用它们进行自定义。取而代之，请使用此处概述的 OS::TripleO::114ExtraConfigPre 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

在核心 Puppet 配置之前，应用到 Compute 节点的其他配置。

OS::TripleO::CephStorageExtraConfigPre

在核心 Puppet 配置之前，应用到 Ceph Storage 节点的其他配置。

OS::TripleO::ObjectStorageExtraConfigPre

在核心 Puppet 配置之前，应用到 Object Storage 节点的其他配置。

OS::TripleO::BlockStorageExtraConfigPre

在核心 Puppet 配置之前，应用到块存储节点的其他配置。

OS::TripleO::[ROLE]ExtraConfigPre

在核心 Puppet 配置之前，应用到自定义节点的其他配置。将 [ROLE] 替换为可组合角色名称。

在本例中，将 resolv.conf 文件附加到特定角色的所有节点上，并带有一个变量名称服务器：

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','UPDATE']
      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，我们定义了 Bash 脚本，heat 将 _NAMESERVER_IP_ 替换为存储在 nameserver_ip 参数的值。

CustomExtraDeploymentPre

这将执行一个软件配置，这是来自 **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，这是此 **hook** 模板中强制的。
- **actions** 参数定义何时应用配置。在本例中，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** 更新的资源获取。

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，若要仅将配置应用到 **Controller** 节点，请使用 **ControllerExtraConfigPre hook**：

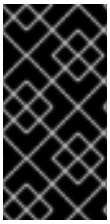
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

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 \
    ...
```

这会在核心配置开始创建或后续更新之前，将配置应用到所有 **Controller** 节点。



重要

您可以为每个 **hook** 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。

4.3. 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**，可用于在第一次引导后配置所有节点类型，并在核心配置开始前：

OS::TripleO::NodeExtraConfig

在核心 **Puppet** 配置之前，应用到所有节点角色的其他配置。

在这个示例中，使用变量名称服务器在每个节点上附加 **resolv.conf** 文件：

流程

1. 创建一个基本的 **heat** 模板 **~/templates/nameserver.yaml**，该脚本使用变量名称服务器附加每个节点的 **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','UPDATE']
      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

此参数定义软件配置。在本例中，您定义一个 Bash 脚本，heat 将 _NAMESERVER_IP_ 替换为存储在 nameserver_ip 参数的值。

CustomExtraDeploymentPre

此参数执行软件配置，这是来自 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，这是此 hook 模板中强制的。**
- **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，它存储父模板的 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 更新的资源获取。**

2. **创建一个环境文件 ~/templates/pre_config.yaml，它将 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：

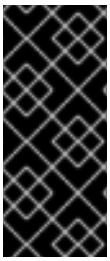
```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

这会在核心配置开始创建或后续更新之前，将配置应用到所有节点。

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。

4.4. 后配置：自定义所有 OVERCLOUD 角色



重要

本文档的早期版本使用 OS::TripleO::Tasks:: thePostConfig 资源来基于每个角色提供后配置 hook。heat 模板集合需要专用使用这些 hook，这意味着您不应该使用它们进行自定义。反之，请使用此处概述的 OS::TripleO::NodeExtraConfigPost hook。

当您完成 overcloud 的创建，但您想要在初始创建时或后续 overcloud 更新时，为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其他配置。

在这个示例中，使用变量名称服务器在每个节点上附加 resolv.conf 文件：

流程

1.

创建一个基本的 heat 模板 ~/templates/nameserver.yaml，该脚本使用变量名称服务器附加每个节点的 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','UPDATE']
      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，您定义一个 Bash 脚本，heat 将 _NAMESERVER_IP_ 替换为存储在 nameserver_ip 参数的值。

CustomExtraDeployments

这将执行一个软件配置，这是来自 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，这是此 hook 模板中强制的。
- actions 参数定义何时应用配置。在本例中，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 更新的资源获取。

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。

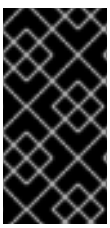
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

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 \
    ...
```

这会在内核配置完成初始 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。

4.5. PUPPET：自定义角色的 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 Puppet 配置的 hieradata：

ControllerExtraConfig

添加至所有 Controller 节点的配置。

ComputeExtraConfig

配置为添加到所有 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 Object Storage 节点的配置。

CephStorageExtraConfig

添加至所有 Ceph Storage 节点的配置。

[ROLE]ExtraConfig

要添加到可组合角色的配置。将 [ROLE] 替换为可组合角色名称。

ExtraConfig

要添加到所有节点的配置。

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 keymap 设置为日语，请使用 ComputeExtraConfig 参数中的以下条目：

```
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。

4.6. PUPPET：为单个节点自定义 HIERADATA

您可以使用 heat 模板集合为单个节点设置 Puppet hieradata：

流程

1. 从节点的内省数据中识别系统 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 模板注册到预先配置 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与每个系统 UUID 对应的 hieradata 文件，并包含您定义的 hieradata。如果没有定义 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。

您可以使用此机制来根据特定要求对每个节点进行定制。

有关 NodeDataLookup 的更多信息，请参阅 *Deploying an overcloud with containerized Red Hat Ceph* 指南中的 Altering the disk layout in Ceph Storage nodes。

4.7. 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。您可以使用在主配置完成后应用到节点的自定义 Puppet 清单来达到此目的。例如，您可能需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，用于启动 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}
```

本例在模板中包含 /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，它将您的 heat 模板注册为

OS::TripleO::NodeExtraConfigPost: 资源类型。

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

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。

第 5 章 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或 Red Hat Satellite Server。

如果您使用以前的 Red Hat OpenStack Platform 版本中的 rhel-registration 方法，则必须禁用它并切换到基于 Ansible 的方法。如需更多信息，请参阅 切换到 rhsm 可组合服务 和 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 5.9 节 “手动运行基于 Ansible 的注册”。

5.1. 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，默认是禁用的。要启用该服务，请在 rhsm 可组合服务文件中注册该资源：

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

rhsm 可组合服务接受 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-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 8.4
```

您还可以将 RhsmVars 参数与特定于角色的参数结合使用，如 ControllerParameters，以在为不同节点类型启用特定存储库时提供灵活性。

5.2. 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。如需有关可用的 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 orgs**。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="DemoRed Hat OpenStack** 114"，并使用生成的 **池 ID** 值。使用列表格式将多个 ID 传递给此参数。
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。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。如果您使用 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。如果您使用 Satellite 服务器，请将此值更改为 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，请使用激活码注册。
rhsm_password	注册的密码。如果可能，请使用激活码注册。
rhsm_release	用于固定软件仓库的 Red Hat Enterprise Linux 发行版本。对于 Red Hat OpenStack Platform，它被设置为 8.4
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	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** 设为**端口** 时才可以使用 **rhsm_activation_key** 和 **rhsm_repos**。如果 **rhsm_method** 设为 ***satellite***，则只能使用 **rhsm_activation_key** 或 **rhsm_repos**。

5.3. 将 OVERCLOUD 注册到 RHSM 可组合服务

创建启用和配置 **rhsm** 可组合服务的环境文件。**director** 使用此环境文件注册和订阅您的节点。

流程

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

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-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-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: 8.4
```

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

- **RhsmVars** 变量将参数传递给 **Ansible**，以配置您的红帽注册。

3. 保存环境文件。

5.4. 将 RHSM 可组合服务应用到不同的角色

您可以根据角色应用 **rhsm** 可组合服务。例如，您可以将不同的配置集合应用到 **Controller** 节点、计算节点和 **Ceph Storage** 节点。

流程

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

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-8-for-x86_64-baseos-eus-rpms
        - rhel-8-for-x86_64-appstream-eus-rpms
        - rhel-8-for-x86_64-highavailability-eus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - openstack-16.2-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.4
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-eus-rpms
        - rhel-8-for-x86_64-appstream-eus-rpms
        - rhel-8-for-x86_64-highavailability-eus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - openstack-16.2-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.4
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-rpms
        - rhel-8-for-x86_64-appstream-rpms
```

```
    - rhel-8-for-x86_64-highavailability-rpms
    - ansible-2.9-for-rhel-8-x86_64-rpms
    - openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms
  rhsm_username: "myusername"
  rhsm_password: "p@55w0rd!"
  rhsm_org_id: "1234567"
  rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
  rhsm_method: "portal"
  rhsm_release: 8.4
```

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

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。**

5.5. 将 OVERCLOUD 注册到 RED HAT SATELLITE SERVER

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网站。

流程

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

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: 8.4
```

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

RhsmVars 变量将参数传递给 Ansible，以配置您的红帽注册。

3. **保存环境文件。**

5.6. 切换到 RHSM 可组合服务

以前的 rhel-registration 方法运行一个 bash 脚本来处理 overcloud 注册。此方法的脚本和环境文件位于 /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. **将 rhsm 可组合服务参数的环境文件添加到未来的部署操作中。**

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
```

您还可以将 /usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml 环境文件包含在部署中以启用该服务。

5.7. RHEL-REGISTRATION 到 RHSM 映射

为了帮助将您的详情从 rhel-registration 方法转换到 rhsm 方法，请使用下表映射您的参数和值。

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-registration	rhsm / RhsmVars
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

5.8. 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 控制您的 overcloud 节点的注册过程。

流程

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

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

这可启用 overcloud 的 Ansible 配置和基于 Ansible 的注册。

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
```

5.9. 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在部署的 overcloud 上执行手动 Ansible 注册。使用此脚本将节点角色定义为主机组，然后使用 ansible-playbook 针对它们运行 playbook。使用以下示例 playbook 手动注册 Controller 节点。

流程

1. 创建一个 playbook，它使用 redhat_subscription 模块来注册您的节点。例如，以下 playbook 适用于 Controller 节点：

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - openstack-beta-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        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。
  - 禁用任何自动启用的软件仓库。
  - 仅启用与 Controller 节点相关的软件仓库。存储库使用 repos 变量列出。

2. 部署 overcloud 后，您可以运行以下命令，以便 Ansible 对 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 参数中定义的组中节点，本例中为 Controller 组。

第 6 章 可组合服务和自定义角色

overcloud 通常由预定义的角色（如 Controller 节点、计算节点和不同的存储节点类型）的节点组成。每个默认角色都包含 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合的自定义角色。

您可以使用此灵活性在不同的角色上创建不同服务组合。本章探索自定义角色、可组合服务和使用方法的架构。

6.1. 支持的角色架构

使用自定义角色和可组合服务时有以下架构：

默认构架

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。

支持的独立角色

使用 /usr/share/openstack-tripleo-heat-templates/roles 中的预定义文件来生成自定义 roles_data 文件。更多信息请参阅 第 6.4 节 “支持的自定义角色”。

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，测试并验证了有限的可组合服务组合，红帽不支持所有可组合的服务组合。

6.2. 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角色的所有服务的定义。使用以下示例代码片段了解 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**
- **CephStorage.**

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
```

6.3. 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文件。director 提供多个命令来管理角色模板并自动生成自定义 roles_data 文件。**

流程

1. **列出默认角色模板：**

```
$ openstack overcloud roles list
BlockStorage
CephStorage
```

```
Compute
ComputeHCI
ComputeOvsDpdk
Controller
...
```

2. 使用 **openstack overcloud roles show** 命令查看 YAML 格式的角色定义：

```
$ openstack overcloud roles show Compute
```

3. 生成自定义 **roles_data** 文件。使用 **openstack overcloud roles generate** 命令将多个预定义角色加入到一个文件中。例如，运行以下命令生成 **roles_data.yaml** 文件，该文件包含 **Controller, Compute, 和 Networker** 角色：

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

使用 **-o** 选项定义输出文件的名称。

此命令创建自定义 **roles_data** 文件。但是，上例使用 **Controller** 和 **Networker** 角色，它们都使用相同的网络代理。这意味着网络服务从 **Controller** 角色扩展到 **Networker** 角色，**overcloud** 会在 **Controller** 和 **Networker** 节点之间平衡网络服务的负载。

要使此 **Networker** 角色独立，您可以创建自己的自定义角色角色，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** 文件。

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

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

5. 在此目录中添加或修改自定义角色文件。将 **--roles-path** 选项与任何 **role** 子命令一起使用，将这个目录用作自定义角色的源：

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

此命令从 **~/roles** 目录中的单独角色生成单个 **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 Networker, Messaging, and Database 角色结合使用。

6.4. 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 scale-out Ceph Storage 文件角色。包括 OSD 和对象操作(MDS)。	**CephFile.yaml**
CephObject	独立 scale-out 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 Instance HA 节点角色。与 **environments/compute-instanceha.yaml 的环境文件** 一起使用。	**ComputeInstanceHA.yaml**
ComputeLiquidio	具有 Cavium Liquidio 智能 NIC 的计算节点。	**ComputeLiquidio.yaml**
ComputeOvsDpdkRT	计算 OVS DPDK RealTime 角色。	**ComputeOvsDpdkRT.yaml**
ComputeOvsDpdk	计算 OVS DPDK 角色。	**ComputeOvsDpdk.yaml**

角色	描述	File
ComputePPC64LE	ppc64le 服务器的 compute 角色。	**ComputePPC64LE.yaml**
ComputeRealTime	为实时行为进行了优化的 compute 角色。使用此角色时，必须 **overcloud-realtime-compute** 镜像可用，并且角色特定的参数 **IsolCpusList**、**NovaComputeCpuDedicatedSet** 和 **NovaComputeCpuSharedSet** 根据实时计算节点的硬件设置。	**ComputeRealTime.yaml**
ComputeSriovRT	计算 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	载入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 Storage 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 存储块存储服务的计算节点。包括 OSD、MON 和管理器。	**HciCephMon.yaml**
HciCephObject	具有超融合基础架构和 Ceph Storage 对象服务的计算节点。包括 OSD 和对象网关(RGW)。	**HciCephObject.yaml**
IronicConductor	ironic Conductor 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，请参阅网络指南中的 使用 ML2/OVN 部署自定义 角色中的其他步骤。	**Networker.yaml**
NetworkerSriov	与正常的 Networker 角色相同，但部署了 OVN 元数据代理。请参阅网络指南中的 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 Telemetry 角色。	**Telemetry.yaml**

6.5. 检查角色参数

每个角色都包含以下参数：

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

如果没有标记主角色，您定义的第一个角色将变为主角色。确保此角色是 **Controller** 角色。

网络

您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。默认命名约定使用以下格式：

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

例如，默认 Controller 节点被命名为：

```
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。有关更多信息，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 OpenStack 更新选项期间要同时更新的节点数量。在默认的 roles_data.yaml 文件中：

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

如果您从自定义角色中省略此参数，则默认为 1。

ServicesDefault

（可选） 定义要包含在节点上的默认服务列表。更多信息请参阅 第 6.8 节 “检查可组合服务架构”。

您可以使用这些参数创建新角色，并定义要包含在角色中的服务。

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，在某些时候，overcloud.j2.yaml heat 模板会迭代 roles_data.yaml 中的角色列表，并创建特定于每个相应角色的参数和资源。

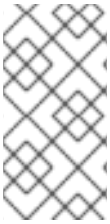
例如，以下片段包含 overcloud.j2.yaml heat 模板中各个角色的资源定义：

```
  {{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]}
...
```

此片段演示基于 Jinja2 的模板如何纳入 {{role.name}} 变量，以将每个角色的名称定义为 OS::114::ResourceGroup 资源。这依次使用 roles_data 文件中的每个 name 参数来命名每个对应的 OS::114::ResourceGroup 资源。

6.6. 创建新角色

您可以使用可组合服务架构根据部署的要求创建新角色。例如，您可能希望仅创建一个新的 Horizon 角色来仅托管 OpenStack 控制面板(horizon)。



注意

角色名称必须以字母或数字结尾，并且仅包含字母、数字和连字符。角色名称中不得使用下划线。

流程

1. **创建默认 roles 目录的自定义副本：**

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

2. **创建名为 ~/roles/114.yaml 的新文件，并创建一个包含基本和核心 OpenStack 仪表板服务的**

新 Horizon 角色：

```
- name: Horizon
  CountDefault: 1
  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
```

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

3. **可选：如果要扩展现有 overcloud 中的服务，请在 Controller 角色中保留现有服务。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 Dashboard 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
    ...
```

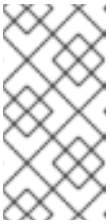
4. 使用 ~/roles 目录作为源生成新的 roles_data-horizon.yaml 文件：

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

5. 为此角色定义一个新类别，以便您可以标记特定的节点。在本例中，使用以下命令创建一个 horizon 类别：

   a. 创建 horizon 类别：

   ```
   (undercloud)$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 horizon
   ```

   注意

   这些属性不用于调度实例，但计算调度程序将使用磁盘大小来确定根分区大小。

   b. 使用自定义资源类为 Dashboard 服务(horizon)标记您要指定的每个裸机节点：

   ```
   (undercloud)$ openstack baremetal node set --resource-class baremetal.HORIZON
   <NODE>
   ```

   将 <NODE> 替换为裸机节点的 ID。

   c. 将 horizon 类别与自定义资源类关联：

   ```
   (undercloud)$ openstack flavor set --property
   resources:CUSTOM_BAREMETAL_HORIZON=1 horizon
   ```

   要确定与裸机节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，用下划线替换 punctuation，并使用 CUSTOM_ 前缀。

   注意

   类别只能请求一个裸机资源类实例。

d. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

```
(undercloud)$ openstack flavor set --property resources:VCPU=0 --property
resources:MEMORY_MB=0 --property resources:DISK_GB=0 horizon
```

6. 使用以下环境文件片断定义 **Horizon** 节点数和类型：

```
parameter_defaults:
  OvercloudHorizonFlavor: horizon
  HorizonCount: 1
```

7. 在 **openstack overcloud deploy** 命令中包含新的 **roles_data-horizon.yaml** 文件和环境文件，以及与部署相关的任何其他环境文件：

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-horizon.yaml -e
~/templates/node-count-flavor.yaml
```

此配置会创建一个三节点 **overcloud**，它由一个 **Controller** 节点、一个 **Compute** 节点和一个 **Networker** 节点组成。要查看 **overcloud** 中的节点列表，请运行以下命令：

```
$ openstack server list
```

6.7. 指南和限制

请注意可组合角色架构的以下准则和限制：

对于不是由 **Pacemaker** 管理的服务：

- 您可以为独立自定义角色分配服务。
- 您可以在初始部署后创建额外的自定义角色，并进行部署以扩展现有服务。

对于由 **Pacemaker** 管理的服务：

- 您可以为独立自定义角色分配 Pacemaker 管理的服务。

- Pacemaker 具有 16 个节点限制。如果将 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)。

- 您不能扩展或缩减包含 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，并将孤立的服务留在节点上。

6.8. 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。

每个模板包含一个标识其目的的描述。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。所有服务将 **OS::TripleO::Services** 命名空间用于其资源类型。

有些资源直接使用基本可组合服务模板：

```
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含依赖项。例如，**deployment/keystone/keystone-container-puppet.yaml** 模板将基本模板的输出存储在 **ContainersCommon** 资源中：

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

然后，容器化模板可以包含来自 **containers-common.yaml** 模板的功能和数据。

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

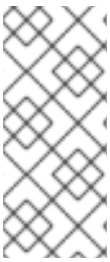
```
{{role.name}}Services:
  description: A list of service resources (configured in the heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}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

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

您可以在 **roles_data.yaml** 文件中定义每个自定义角色的默认服务。例如，默认 **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 **services** 列表。

6.9. 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. 创建默认 **roles** 目录的自定义副本：

```
$ 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
```

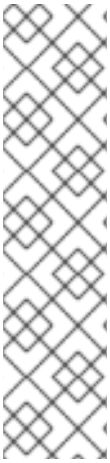
3. 生成新的 roles_data 文件：

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

4. 在运行 openstack overcloud deploy 命令时包括此新的 roles_data 文件：

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

此命令在没有 Controller 节点上安装的 OpenStack 编排服务的情况下部署 overcloud。



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 OS::114::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
```

6.10. 启用禁用的服务

一些服务会被默认禁用。这些服务在 overcloud-resource-registry-puppet.j2.yaml 文件中作为 null 操作注册。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，请包含一个环境文件，该文件将资源链接到 puppet/services 目录中对应的 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 操作资源并启用该服务。

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
```

6.11. 创建没有服务的通用节点

您可以创建通用 Red Hat Enterprise Linux 8.4 节点，而无需配置任何 OpenStack 服务。当您需要托管 Red Hat OpenStack Platform (RHOSP)环境外的软件时，这非常有用。例如，RHOSP 提供与 Kibana 和 Sensu 等监控工具集成。如需更多信息，请参阅 *监控工具配置指南*。虽然红帽不提供对监控工具本身的支持，但 director 可以创建通用 Red Hat Enterprise Linux 8.4 节点来托管这些工具。

注意

通用节点仍然使用基本 overcloud-full 镜像，而不是基本 Red Hat Enterprise Linux 8 镜像。这意味着节点安装了一些 Red Hat OpenStack Platform 软件，但没有启用或配置。

流程

1. 在自定义 roles_data.yaml 文件中创建一个没有包括 ServicesDefault 列表的通用角色：

```
- name: Generic
- name: Controller
  CountDefault: 1
```

```
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

确保保留现有的 Controller 和 Compute 角色。

2. **创建一个环境文件 generic-node-params.yaml，以指定在选择要置备的节点时所需的通用 Red Hat Enterprise Linux 8 节点数量以及类型：**

```
parameter_defaults:
  OvercloudGenericFlavor: baremetal
  GenericCount: 1
```

3. **在运行 openstack overcloud deploy 命令时，同时包含角色文件和环境文件：**

```
$ openstack overcloud deploy --templates \
-r ~/templates/roles_data_with_generic.yaml \
-e ~/templates/generic-node-params.yaml
```

此配置部署一个具有一个 Controller 节点、一个 Compute 节点和一个通用 Red Hat Enterprise Linux 8 节点的三节点环境。

第 7 章 容器化服务

director 将核心 OpenStack Platform 服务作为容器安装在 overcloud 上。本节提供有关容器化服务如何工作的一些背景信息。

7.1. 容器化服务架构

director 将核心 OpenStack Platform 服务作为容器安装在 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，请包含 OS::TripleO::Services::Podman 服务以及基本可组合服务。例如，IronicConductor 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
```

7.2. 容器化服务参数

每个容器化服务模板包含一个 output 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（请参见 第 6.5 节 “检查角色参数”）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 会创建一组容器，用于在实际容器化服务运行前配置该服务。这个参数包括以下子参数：

- config_volume - 存储配置的挂载卷。
- puppet_tags - 在配置期间传递给 Puppet 的标签。OpenStack 使用这些标签将 Puppet 运行限制到特定服务的配置资源。例如，OpenStack Identity (keystone)容器化服务使用 keystone_config 标签来确保所有需要 keystone_config Puppet 资源都在配置容器中运行。
- 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 - 用于配置该服务的容器镜像。

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该服务。

docker_config

在服务的配置容器中运行的任务。所有任务都分组到以下步骤中，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 服务的初始配置
- 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。

7.3. 准备容器镜像

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

注意

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

流程

1. 以 stack 用户身份登录 undercloud 主机。
2. 生成默认的容器镜像准备文件：

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

此命令包括以下附加选项：

-

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

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

注意

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

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

7.4. 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：

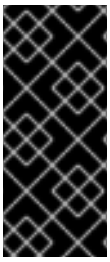
```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

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**。

参数	描述
modify_append_tag	要附加到目标镜像标签的字符串。例如，如果使用标签 16.2.3-5.161 拉取镜像并将 **modify_append_tag** 设置为 **-hotfix**，director 会将最终镜像标记为 16.2.3-5.161-hotfix。
modify_only_with_labels	过滤想要修改的镜像的镜像标签字典。如果镜像与定义的标签匹配，则 director 将该镜像包括在修改过程中。
modify_role	在上传期间但在将镜像推送到目标 registry 之前运行的 ansible 角色名称字符串。
modify_vars	要传递给 **modify_role** 的变量的字典。
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。

参数	描述
tag_from_label	使用指定容器镜像元数据标签的值为每个镜像创建标签，并拉取该标记的镜像。例如，如果设置了 **tag_from_label: {version}-{release}**，则 director 会使用 **version** 和 **release** 标签来构造新标签。对于一个容器，**version** 可能会设置为 16.2.3，**release** 可能会设置为 **5.161**，这会产生标签 16.2.3-5.161。仅当未在 **set** 字典中定义 **tag** 时，director 才使用此参数。



重要

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

set 参数接受一组 key: value 定义：

键	描述
ceph_image	Ceph Storage 容器镜像的名称。
ceph_namespace	Ceph Storage 容器镜像的命名空间。
ceph_tag	Ceph Storage 容器镜像的标签。
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	Ceph Storage Alert Manager 容器镜像的名称、命名空间和标签。
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	Ceph Storage Grafana 容器镜像的名称、命名空间和标签。
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	Ceph Storage Node Exporter 容器镜像的名称、命名空间和标签。

键	描述
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。

7.5. 容器镜像标记准则

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

version

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

release

对应于版本流中特定容器镜像版本的发行版本。

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

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

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

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

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

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

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

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

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

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

- tag_from_label 参数根据其从 Red Hat Container Registry 中检查到的最新容器镜像发行版

本的标签元数据生成标签。例如，特定容器的标签可能会使用以下 version 和 release 元数据：

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

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

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

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

7.6. 从私有 REGISTRY 获取容器镜像

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

ContainerImageRegistryCredentials

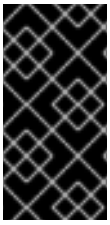
有些容器镜像 registry 需要进行身份验证才能访问镜像。在这种情况下，请使用您的 containers-prepare-parameter.yaml 环境文件中的 ContainerImageRegistryCredentials 参数。ContainerImageRegistryCredentials 参数使用一组基于私有 registry URL 的键。每个私有 registry URL 使用其自己的键和值对定义用户名（键）和密码（值）。这提供了一种为多个私有 registry 指定凭据的方法。

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

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

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

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



重要

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

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

ContainerImageRegistryLogin

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

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

```
parameter_defaults:
  ContainerImagePrepare:
```

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

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

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

7.7. 分层镜像准备条目

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

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

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

```
- nova-api
set:
  namespace: registry.redhat.io/rhosp-rhel8
  tag: 16.2.1-hotfix
```

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

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

7.8. 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。

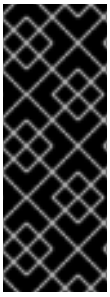
Ansible 角色 tripleo-modify-image 与所需的角色接口相一致，并提供修改用例必需的行为。使用 ContainerImagePrepare 参数中与修改相关的键控制修改：

- 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：

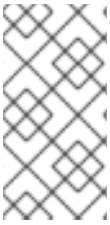
```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```



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。

7.9. 更新容器镜像的现有软件包



注意

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

步骤

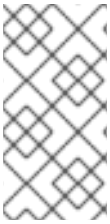
- 以下示例 ContainerImagePrepare 条目使用 undercloud 主机的 dnf 软件仓库配置在容器镜像的所有软件包中更新：

```
ContainerImagePrepare:
- push_destination: true
  ...
```

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

7.10. 将额外的 RPM 文件安装到容器镜像中

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



注意

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

注意

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

步骤

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

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

7.11. 通过自定义 DOCKERFILE 修改容器镜像

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

注意

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

步骤

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

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

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

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

USER "root"

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

USER "nova"
```

7.12. 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供应商插件，将 Dell EMC 硬件用作块存储后端。

有关支持的后端设备和驱动程序的更多信息，请参阅存储指南中的第三方 *存储提供商*。

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-rhel8
        name_prefix: openstack-
        name_suffix: ''
        tag: 16.2
        ...
      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. **使用任何部署命令包括 containers-prepare-parameter-dellemc.yaml 文件，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。

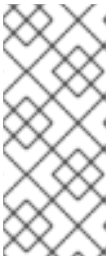
重要

containers-prepare-parameter-dellemc.yaml 文件替换了 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 undercloud 安装和更新保留标准 containers-prepare-parameter.yaml 文件。

第 8 章 基本网络隔离

配置 overcloud 以使用隔离网络，以便可以隔离托管特定类型的网络流量。Red Hat OpenStack Platform (RHOSP)包括一组环境文件，可用于配置此网络隔离。您可能还需要额外的环境文件来进一步自定义网络参数：

- 可用于启用网络隔离的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml)。



注意

在使用 director 部署 RHOSP 前，文件 network-isolation.yaml 和 network-environment.yaml 仅采用 Jinja2 格式，且具有 .j2.yaml 扩展。director 在部署过程中将这些文件呈现到 .yaml 版本。

- 可用于配置网络默认值的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml)。
- 可用于定义网络设置的 network_data 文件，如 IP 范围、子网和虚拟 IP。本例演示了如何创建默认副本并编辑它以适合您的网络。
- 用于为每个节点定义 NIC 布局的模板。overcloud 核心模板集合包含一组用于不同用例的默认值。
- 可用于启用 NIC 的环境文件。本例使用位于 environments 目录中的默认文件。

8.1. 网络隔离

overcloud 默认为 provisioning 网络分配服务。但是，director 可以将 overcloud 网络流量划分为隔离的网络。若要使用隔离的网络，overcloud 包含一个启用此功能的环境文件。核心 heat 模板中的 environments/network-isolation.j2.yaml 文件是一个 Jinja2 文件，该文件在可组合网络文件中定义每个网络的所有端口和 VIP。呈现后，它会在带有完整资源 registry 的同一位置生成 network-isolation.yaml 文件：

```
resource_registry:
  # networks as defined in network_data.yaml
  OS::TripleO::Network::Storage: ../network/storage.yaml
  OS::TripleO::Network::StorageMgmt: ../network/storage_mgmt.yaml
```

```
  OS::TripleO::Network::InternalApi: ../network/internal_api.yaml
  OS::TripleO::Network::Tenant: ../network/tenant.yaml
  OS::TripleO::Network::External: ../network/external.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::StorageVipPort: ../network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: ../network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: ../network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::ExternalVipPort: ../network/ports/external.yaml
  OS::TripleO::Network::Ports::RedisVipPort: ../network/ports/vip.yaml

  # Port assignments by role, edit role definition to assign networks to roles.
  # Port assignments for the Controller
  OS::TripleO::Controller::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: ../network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::TenantPort: ../network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ExternalPort: ../network/ports/external.yaml

  # Port assignments for the Compute
  OS::TripleO::Compute::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::Compute::Ports::InternalApiPort: ../network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::TenantPort: ../network/ports/tenant.yaml

  # Port assignments for the CephStorage
  OS::TripleO::CephStorage::Ports::StoragePort: ../network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
```

此文件的第一个部分具有 OS::TripleO::Network::114 资源的资源 registry 声明。默认情况下，这些资源使用 OS::114::None 资源类型，它不创建任何网络。通过将这些资源重定向到每个网络的 YAML 文件，您可以启用创建这些网络。

接下来的几个部分为各个角色中的节点创建 IP 地址。控制器节点在每个网络上都有 IP。计算和存储节点各自在网络的子集上具有 IP。

overcloud 网络的其他功能，如 第 9 章 *自定义可组合网络* 和 第 10 章 *自定义网络接口模板* 依赖于 network-isolation.yaml 环境文件。因此，您必须在部署命令中包含呈现的环境文件：

```
$ openstack overcloud deploy --templates \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  ...
```

8.2. 修改隔离的网络配置

复制默认的 network_data.yaml 文件并修改副本来配置默认的隔离网络。

流程

1. **复制默认的 network_data.yaml 文件：**

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

2. **编辑 network_data.yaml 文件的本地副本，并修改参数以满足您的网络要求。例如，内部 API 网络包含以下默认网络详情：**

```
- name: InternalApi
  name_lower: internal_api
  vip: true
  vlan: 201
  ip_subnet: '172.16.2.0/24'
  allocation_pools: [{'start': '172.16.2.4', 'end': '172.16.2.250'}]
```

编辑每个网络的以下值：

- **vlan 定义这个网络要使用的 VLAN ID。**
- **ip_subnet 和 ip_allocation_pools 为网络设置默认子网和 IP 范围。**
- **网关 设置网络的网关。使用这个值为外部网络或其他网络定义默认路由（如果需要）。**

使用 -n 选项将自定义 network_data.yaml 文件包含在部署中。如果没有 -n 选项，部署命令使用默认的网络详情。

8.3. 网络接口模板

overcloud 网络配置需要一组网络接口模板。这些模板采用 YAML 格式的标准 heat 模板。每个角色都需要一个 NIC 模板，以便 director 可以正确地配置该角色中的每个节点。

所有 NIC 模板都包含与标准 heat 模板相同的部分：

heat_template_version

要使用的语法版本。

description

模板的字符串描述。

parameters

模板中包含的网络参数。

资源

使用参数 中定义的参数，并将其应用到网络配置脚本。

输出

呈现用于配置的最终脚本。

/usr/share/openstack-tripleo-heat-templates/network/config 中的默认 NIC 模板使用 Jinja2 语法呈现模板。例如，single-nic-vlans 配置中的以下片段为每个网络呈现一组 VLAN：

```
{%- for network in networks if network.enabled|default(true) and network.name in role.networks %}
- type: vlan
  vlan_id:
    get_param: {{network.name}}NetworkVlanID
  addresses:
  - ip_netmask:
      get_param: {{network.name}}IpSubnet
{%- if network.name in role.default_route_networks %}
```

对于默认 Compute 节点，这只呈现存储、内部 API 和租户网络的网络信息：

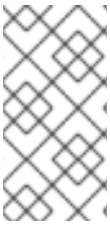
```
- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
- type: vlan
```

```
vlan_id:
  get_param: TenantNetworkVlanID
device: bridge_name
addresses:
- ip_netmask:
    get_param: TenantIpSubnet
```

第 10 章 *自定义网络接口模板* 了解如何将基于 Jinja2 的默认模板呈现到标准 YAML 版本，您可以将它们用作自定义的基础。

8.4. 默认网络接口模板

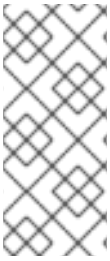
director 包含 /usr/share/openstack-tripleo-heat-templates/network/config/ 中的模板，以适应大多数常见的网络场景。下表概述了每个 NIC 模板集以及必须用来启用模板的对应环境文件。



注意

启用 NIC 模板的每个环境文件都使用后缀 .j2.yaml。这是未渲染的 Jinja2 版本。确保在部署过程中包含呈现的文件名，它使用 .yaml 后缀。

NIC 目录	描述	环境文件
single-nic-vlans	单个 NIC (**nic1**)，带有附加到默认的 Open vSwitch 网桥的 control plane 和 VLAN。	**environments/net-single-nic-with-vlans.j2.yaml**
single-nic-linux-bridge-vlans	单个 NIC (**nic1**)，带有附加到默认 Linux 网桥的 control plane 和 VLAN。	**environments/net-single-nic-linux-bridge-with-vlans**
bond-with-vlans	附加到 **nic1** 的 control plane。默认 Open vSwitch 网桥带有绑定的 NIC 配置（**nic2** 和 **nic3**）并附加了 VLAN。	**environments/net-bond-with-vlans.yaml**
multiple-nics	附加到 **nic1** 的 control plane。将每个后续 NIC 分配给 **network_data.yaml** 文件中定义的每个网络。默认情况下，Storage 到 **nic2**, Storage Management 到 **nic3**, Internal API 到 **nic4**, Tenant 到 **br-tenant** 网桥上的 **nic5**，External 到默认 Open vSwitch 网桥上的 **nic6**。	**environments/net-multiple-nics.yaml**



注意

存在用于在没有外部网络的情况下部署 overcloud 的环境文件，如 net-bond-with-vlans-no-external.yaml，以及用于 IPv6 部署，如 net-bond-with-vlans-v6.yaml。这些是向后兼容的，且不适用于可组合网络。

每个默认 NIC 模板集都包含一个 role.role.j2.yaml 模板。此文件使用 Jinja2 为每个可组合角色呈现其他文件。例如，如果您的 overcloud 使用 Compute、Controller 和 Ceph Storage 角色，部署会根据 role.role.j2.yaml 呈现新模板，如以下模板：

- compute.yaml
- controller.yaml
- ceph-storage.yaml.

8.5. 启用基本网络隔离

director 包括可用于启用基本网络隔离的模板。这些文件位于 /usr/share/openstack-tripleo-heat-templates/environments 目录中。例如，您可以使用模板在带有基本网络隔离的 VLAN 的单一 NIC 上部署 overcloud。在这种情况下，使用 net-single-nic-with-vlans 模板。

流程

1. 运行 openstack overcloud deploy 命令时，请确保包含以下呈现的环境文件：
   - 自定义 network_data.yaml 文件。
   - 默认网络隔离文件的呈现文件名。
   - 默认网络环境文件的呈现文件名。
   - 默认网络接口配置文件的呈现文件名。

- **与配置相关的任何其他环境文件。**

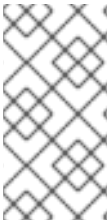
例如：

```
$ openstack overcloud deploy --templates \
    ...
    -n /home/stack/network_data.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
    ...
```

第 9 章 自定义可组合网络

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。要使用额外的可组合网络配置 overcloud，您必须配置以下文件和模板：

- 启用网络隔离的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml)。
- 用于配置网络默认值的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml)。
- 自定义 network_data 文件，用于在默认值之外创建额外网络。
- 为角色分配自定义网络的自定义 roles_data 文件。
- 为各个节点定义 NIC 布局的模板。overcloud 核心模板集合包含一组用于不同用例的默认值。
- 启用 NIC 的环境文件。本例使用位于 environments 目录中的默认文件。
- 用于自定义网络参数的任何其他环境文件。本例使用环境文件来自定义到可组合网络的 OpenStack 服务映射。



注意

上一个列表中的一些文件是 Jinja2 格式文件，且具有 .j2.yaml 扩展名。director 在部署过程中将这些文件呈现到 .yaml 版本。

9.1. 可组合网络

overcloud 默认使用以下一组预定义的网络片段：

- Control Plane

- 内部 API
- 存储
- 存储管理
- tenant
- 外部
- 管理（可选）

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，您可以将其呈现到多个角色。

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 ironic 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个网络集合，以用于在网络间路由流量的分割部署。

单个数据文件(network_data.yaml)管理您要部署的网络列表。使用 -n 选项将此文件包含在部署命令中。如果没有这个选项，部署使用默认的 /usr/share/openstack-tripleo-heat-templates/network_data.yaml 文件。

9.2. 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，您可以将网络呈现到多个角色。

流程

1. 复制默认的 network_data.yaml 文件：

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

2. 编辑 network_data.yaml 文件的本地副本，并为新网络添加一个部分：

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 21
  vip: true
  ip_subnet: '172.21.1.0/24'
  allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
  gateway_ip: '172.21.1.1'
```

您可以在 network_data.yaml 文件中使用以下参数：

name

设置网络的人类可读名称。这个参数是唯一强制参数。您还可以使用 name_lower 来规范化名称以实现可读性。例如，将 InternalApi 更改为 internal_api。

name_lower

设置名称的小写版本，director 映射到分配给 roles_data.yaml 文件中角色的相应网络。

vlan

设置要用于此网络的 VLAN。

vip: true

在新网络上创建虚拟 IP 地址(VIP)。此 IP 用作 service-to-network 映射参数(ServiceNetMap)中列出的服务的目标 IP。请注意，VIP 仅由使用 Pacemaker 的角色使用。overcloud 负载均衡服务将来自这些 IP 的流量重定向到对应的服务端点。

ip_subnet

以 CIDR 格式设置默认 IPv4 子网。

allocation_pools

为 IPv4 子网设置 IP 范围

gateway_ip

设置网络的网关。

Routes

向网络添加额外的路由。使用包含每个额外路由的 JSON 列表。每个列表项目包含一个

字典值映射。使用以下示例语法：

```
routes: [{'destination':'10.0.0.0/16', 'nexthop':'10.0.2.254'}]
```

subnets

创建属于此网络中的其他路由子网。此参数接受一个字典值，其中包含路由子网的小写名称作为键，以及 vlan、ip_subnet、allocation_pools，以及 gateway_ip 参数作为映射到子网的值。以下示例演示了这个布局：

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 200
  vip: true
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '171.21.0.4', 'end': '172.21.0.250'}]
  gateway_ip: '172.21.0.1'
  subnets:
    storage_backup_leaf1:
      vlan: 201
      ip_subnet: '172.21.1.0/24'
      allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
      gateway_ip: '172.19.1.254'
```

此映射在 spine leaf 部署中很常见。如需更多信息，请参阅 *Spine Leaf Networking* 指南。

3. 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
```

4. 使用 -n 选项，在部署命令中包含自定义 network_data.yaml 文件。如果没有 -n 选项，部署命令会使用默认的网络集合。

5. 如果您需要可预测的虚拟 IP 地址(VIP)，请将自定义网络的 VirtualFixedIPs 参数添加到 heat

环境文件的 parameter_defaults 部分，例如 my_network_vips.yaml ：

```
<% my_customer_network %>VirtualFixedIPs:  [{'ip_address':'<% ipaddres %>'}]
```

下面是一个示例：

```
parameter_defaults:
  ...
  # Predictable VIPs
  StorageBackuptVirtualFixedIPs:  [{'ip_address':'172.21.1.9'}]
```

6. 使用 -e 选项，将 heat 环境文件 my_network_vips.yaml 包含在部署命令中。

其他资源

- 分配可预测的虚拟 IP
- 环境文件
- *命令行界面参考中的 overcloud 部署*

9.3. 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，您可以使用 Ceph Storage 节点包含自定义 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/.
```

2. 编辑自定义 roles_data.yaml 文件。

3. 将网络名称包含在您要添加 网络 的角色的网络列表中。例如，要将 StorageBackup 网络添加到 Ceph Storage 角色中，请使用以下示例片断：

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    - Storage
    - StorageMgmt
    - StorageBackup
```

4. 将自定义网络添加到其各自角色后，保存文件。

运行 **openstack overcloud deploy** 命令时，请使用 **-r** 选项包含自定义 **roles_data.yaml** 文件。如果没有 **-r** 选项，部署命令使用默认角色集及其相应分配的网络。

9.4. 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** 服务被分为这些网络，但实际物理网络的数量可能与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来将 **OpenStack** 服务重新分配给不同的网络类型，例如 **/home/stack/templates/service-reassignments.yaml**。**ServiceNetMap** 参数决定您要用于各个服务的网络类型。

例如，您可以通过修改高亮的部分将 **Storage Management** 网络服务重新分配给 **Storage Backup Network**：

```
parameter_defaults:
  ServiceNetMap:
    SwiftMgmtNetwork: storage_backup
    CephClusterNetwork: storage_backup
```

将这些参数改为 **storage_backup** 会将这些服务放在 **Storage Backup** 网络中，而不是 **Storage Management** 网络。这意味着，您必须仅为 **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，它将从 **ServiceNetMapDefaults** 获取并覆盖默认值。**director** 将完整的列表（包括自定义）返回到 **ServiceNetMap**，用于为各种服务配置网络分配。

服务映射适用于使用 **Pacemaker** 的 **network_data.yaml** 文件中使用 **vip: true** 的网络。**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** 文件的 **ServiceNetMapDefaults** 参数中找到默认服务的完整列表。

9.5. 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** 模板搭配使用 **(net-single-nic-with-vlans)**。

流程

1. 运行 **openstack overcloud deploy** 命令时，请确保包含以下文件：

- 自定义 **network_data.yaml** 文件。
- 带有 **network-to-role** 分配的自定义 **roles_data.yaml** 文件。
- 默认网络隔离的呈现文件名。
- 默认网络环境文件的呈现文件名。
- 默认网络接口配置的呈现文件名。
- 与网络相关的额外环境文件，如服务重新分配。

例如：

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -r /home/stack/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
```

```
-e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
-e /home/stack/templates/service-reassignments.yaml \
...
```

本例命令在 overcloud 中的节点之间部署可组合网络，包括您的额外网络。

重要

请记住，如果您要引入新的自定义网络，如管理网络，则必须再次呈现模板。只需将网络名称添加到 **roles_data.yaml** 文件中不足。

9.6. 重命名默认网络

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

- **InternalApi**
- **外部**
- **存储**
- **StorageMgmt**
- **tenant**

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，为您要重命名的每个网络的 **name_lower** 参数输入新名称：

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

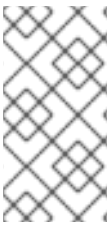
2. **在 service_net_map_replace 参数中包含 name_lower 参数的默认值：**

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

第 10 章 自定义网络接口模板

配置 第 8 章 *基本网络隔离* 后，您可以创建一组自定义网络接口模板来适合环境中的节点。例如，您可以包含以下文件：

- 启用网络隔离的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml)。
- 用于配置网络默认值的环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml)。
- 为各个节点定义 NIC 布局的模板。overcloud 核心模板集合包含一组用于不同用例的默认值。要创建自定义 NIC 模板，请呈现默认 Jinja2 模板作为自定义模板的基础。
- 启用 NIC 的自定义环境文件。本例使用自定义环境文件(/home/stack/templates/custom-network-configuration.yaml)来引用您的自定义模板。
- 用于自定义网络参数的任何其他环境文件。
- 如果您自定义网络，则自定义 network_data.yaml 文件。
- 如果您创建额外的或自定义可组合网络，则自定义 network_data.yaml 文件和自定义 roles_data.yaml 文件。



注意

上一个列表中的一些文件是 Jinja2 格式文件，且具有 .j2.yaml 扩展名。director 在部署过程中将这些文件呈现到 .yaml 版本。

10.1. 自定义网络架构

默认 NIC 模板可能不适用于特定的网络配置。例如，您可能想要创建自己的自定义 NIC 模板来适合特定的网络布局。您可能想要将控制服务和数据服务隔离到单独的 NIC。在这种情况下，您可以使用以下方法将服务映射到 NIC 分配：

- NIC1 (Provisioning)
  - Provisioning/Control Plane
- NIC2 （控制组）
  - 内部 API
  - 存储管理
  - 外部（公共 API）
- NIC3 （数据组）
  - 租户网络(VXLAN 隧道)
  - 租户 VLAN/提供程序 VLAN
  - 存储
  - 外部 VLAN （利用 IP/SNAT）
- NIC4 （管理）
  - 管理

10.2. 呈现用于自定义的默认网络接口模板

要简化自定义模板的配置，请呈现默认 NIC 模板的 Jinja2 语法，并使用呈现的模板作为自定义配置的基础。

流程

1. 使用 **process-templates.py** 脚本呈现 **openstack-tripleo-heat-templates** 集合的副本：

```
$ cd /usr/share/openstack-tripleo-heat-templates
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

这会将所有 **Jinja2** 模板转换为其渲染的 **YAML** 版本，并将结果保存到 **~/openstack-tripleo-heat-templates-rendered**。

如果使用自定义网络文件或自定义角色文件，您可以分别使用 **-n** 和 **-r** 选项包括这些文件：

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered -n
/home/stack/network_data.yaml -r /home/stack/roles_data.yaml
```

2. 复制多个 **NIC** 示例：

```
$ cp -r ~/openstack-tripleo-heat-templates-rendered/network/config/multiple-nics/
~/templates/custom-nics/
```

3. 编辑 **custom-nics** 中设置的模板，以适应您自己的网络配置。

10.3. 网络接口架构

您在 第 10.2 节 “呈现用于自定义的默认网络接口模板” 中呈现的自定义 **NIC** 模板包含 **parameters** 和 **resources** 部分。

参数

parameters 部分包含网络接口的所有网络配置参数。这包括子网范围和 **VLAN ID** 等信息。本节应保持不变，因为 **heat** 模板从其父模板中继承值。但是，您可以使用网络环境文件来修改某些参数的值。

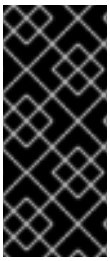
Resources

resources 部分是主网络接口配置发生的位置。在大多数情况下，**resource** 部分是唯一需要修改的。每个 **resources** 部分都以以下标头开始：

-

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
```

此片段运行脚本(run-os-net-config.sh)，它为 os-net-config 创建配置文件，以用于在节点上配置网络属性。network_config 部分包含发送到 run-os-net-config.sh 脚本的自定义网络接口数据。您可以根据设备类型按序列排列此自定义接口数据。



重要

如果创建自定义 NIC 模板，您必须将 run-os-net-config.sh 脚本位置设置为每个 NIC 模板的绝对路径。该脚本位于 undercloud 上的 /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh。

10.4. 网络接口参考

网络接口配置包含以下参数：

interface

定义一个网络接口。配置使用实际接口名称("eth0", "eth1", "enp0s25")或一组数字接口 ("nic1", "nic2", "nic3") 定义各个接口：

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

表 10.1. 接口选项

选项	默认	描述
name		接口的名称。
use_dhcp	False	使用 DHCP 获取 IP 地址。
use_dhcpv6	False	使用 DHCP 获取 v6 IP 地址。

选项	默认	描述
addresses		分配给接口的 IP 地址列表。
Routes		分配给接口的路由列表。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。
defroute	True	使用 DHCP 服务提供的默认路由。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。
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"**，以便在特定 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
  vlan_id:{get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
```

表 10.2. VLAN 选项

选项	默认	描述
vlan_id		VLAN ID。

选项	默认	描述
device		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	定义 VLAN 作为主接口。
defroute	True	使用 DHCP 服务提供的默认路由。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。
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 VLAN 的 DNS 服务器列表。

ovs_bond

在 Open vSwitch 中定义绑定，将两个或多个 接口 接合在一起。这有助于实现冗余性并增加带宽。

例如：

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

```
      name: nic2
    - type: interface
      name: nic3
```

表 10.3. 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		创建绑定时传递给 OVS 的一组选项。
ovs_extra		在绑定的网络配置文件中设置为 OVS_EXTRA 参数的一组选项。
defroute	True	使用 DHCP 服务提供的默认路由。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。
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** 使用 参数名称。

注意

如果您有多个网桥，则必须使用除接受 **bridge_name** 的默认名称以外的不同网桥名称。如果您不使用不同的名称，那么在聚合阶段，则会将两个网络绑定放在同一网桥上。

如果您要为外部 **tripleo** 网络定义 **OVS** 网桥，则分别保留 **bridge_name** 和 **interface_name** 值，作为部署框架，则分别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: bridge_name
  addresses:
  - ip_netmask:
      list_join:
      - /
      - - {get_param: ControlPlaneIp}
        - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: interface_name
- type: vlan
  device: bridge_name
  vlan_id:
    {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask:
        {get_param: ExternalIpSubnet}
```

注意

OVS 网桥连接到网络服务(neutron)服务器，以获取配置数据。如果 OpenStack 控制流量（通常是 Control Plane 和 Internal API 网络）放置在 OVS 网桥上，那么当您升级 OVS 时，与 neutron 服务器的连接都会丢失，或者 OVS 网桥由 admin 用户或进程重启。这会导致一些停机时间。如果在这些情况下无法接受停机时间，您必须将控制组网络放在单独的接口或绑定中，而不是在 OVS 网桥上：

- **当您将内部 API 网络放在调配接口上的 VLAN 和 OVS 网桥到第二个接口上时，您可以达到最小的设置。**
- **要实现绑定，您需要至少有两个绑定（四个网络接口）。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 10.4. 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		创建网桥时传递给 OVS 的一组选项。
ovs_extra		在网桥的网络配置文件中，设置为 OVS_EXTRA 参数的一组选项。
defroute	True	使用 DHCP 服务提供的默认路由。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。

选项	默认	描述
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	用于网桥的 DNS 服务器列表。

linux_bond

定义将两个或者多个 接口 接合在一起的 Linux 绑定。这有助于实现冗余性并增加带宽。确保您在 bonding_options 参数中包含基于内核的绑定选项。

例如：

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

请注意，nic2 使用 primary: true 来确保绑定使用 nic2 的 MAC 地址。

表 10.5. 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。
Routes		分配给绑定的路由列表。请参阅 Routes。

选项	默认	描述
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。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。
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。外部网桥也对参数使用两个特殊值：

- **bridge_name，它替换为外部网桥名称。**
- **interface_name，它替换为外部接口。**

例如：

```
- type: linux_bridge
  name: bridge_name
  addresses:
    - ip_netmask:
        list_join:
          - /
          - - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: interface_name
```

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

表 10.6. 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。只有在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。
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	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: interface
      name: nic2
```

```
...
routes:
  - ip_netmask: 10.1.2.0/24
    gateway_ip: 10.1.2.1
```

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	用于访问目的地网络的路由器的 IP 地址。

10.5. 网络接口布局示例

以下 Controller 节点 NIC 模板片段演示了如何配置自定义网络场景，使控制组与 OVS 网桥分开：

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
              - type: interface
                name: nic1
                mtu:
                 get_param: ControlPlaneMtu
                use_dhcp: false
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                  list_concat_unique:
                    - get_param: ControlPlaneStaticRoutes
              - type: ovs_bridge
                name: bridge_name
                dns_servers:
                  get_param: DnsServers
                domain:
                  get_param: DnsSearchDomains
```

```
    members:
    - type: ovs_bond
      name: bond1
      mtu:
        get_attr: [MinViableMtu, value]
      ovs_options:
        get_param: BondInterfaceOvsOptions
      members:
        - type: interface
          name: nic2
          mtu:
            get_attr: [MinViableMtu, value]
          primary: true
        - type: interface
          name: nic3
          mtu:
            get_attr: [MinViableMtu, value]
    - type: vlan
      mtu:
        get_param: StorageMtu
      vlan_id:
        get_param: StorageNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: StorageIpSubnet
      routes:
        list_concat_unique:
          - get_param: StorageInterfaceRoutes
    - type: vlan
      mtu:
        get_param: StorageMgmtMtu
      vlan_id:
        get_param: StorageMgmtNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: StorageMgmtIpSubnet
      routes:
        list_concat_unique:
          - get_param: StorageMgmtInterfaceRoutes
    - type: vlan
      mtu:
       get_param: InternalApiMtu
      vlan_id:
        get_param: InternalApiNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: InternalApiIpSubnet
      routes:
        list_concat_unique:
          - get_param: InternalApiInterfaceRoutes
     - type: vlan
       mtu:
         get_param: TenantMtu
       vlan_id:
         get_param: TenantNetworkVlanID
       addresses:
```

```
          - ip_netmask:
              get_param: TenantIpSubnet
            routes:
              list_concat_unique:
                - get_param: TenantInterfaceRoutes
          - type: vlan
            mtu:
              get_param: ExternalMtu
            vlan_id:
              get_param: ExternalNetworkVlanID
            addresses:
            - ip_netmask:
                get_param: ExternalIpSubnet
            routes:
              list_concat_unique:
                - get_param: ExternalInterfaceRoutes
                - - default: true
                    next_hop:
                      get_param: ExternalInterfaceDefaultRoute
```

此模板使用三个网络接口，并将多个标记的 VLAN 设备分配给编号的接口 nic1 到 nic3。在 nic2 和 nic3 上，此模板创建托管存储、租户和外部网络的 OVS 网桥。因此，它会创建以下布局：

- **NIC1 (Provisioning)**
  - **Provisioning/Control Plane**
- **NIC2 和 NIC3 （管理）**
  - **内部 API**
  - **存储**
  - **存储管理**
  - **租户网络(VXLAN 隧道)**
  - **租户 VLAN/提供程序 VLAN**

- 外部（公共 API）
- 外部 VLAN （利用 IP/SNAT）

10.6. 自定义网络的网络接口模板注意事项

当您使用可组合网络时，process-templates.py 脚本会使静态模板包含在 network_data.yaml 和 roles_data.yaml 文件中定义的网络和角色。确保您的渲染的 NIC 模板包含以下项目：

- 每个角色的静态文件，包括自定义可组合网络。
- 每个角色的静态文件中的正确网络定义。

每个静态文件都需要任何自定义网络的所有参数定义，即使网络没有在角色中使用。确保呈现的模板包含这些参数。例如，如果您只向 Ceph 节点添加 StorageBackup 网络，还必须将该定义包含在所有角色的 NIC 配置模板中的 parameter 部分：

```
parameters:
  ...
  StorageBackupIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ...
```

如果需要，您还可以包括 VLAN ID 和/或网关 IP 的 parameters 定义：

```
parameters:
  ...
  StorageBackupNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  StorageBackupDefaultRoute:
   description: The default route of the storage backup network.
   type: string
  ...
```

自定义网络的 IpSubnet 参数会出现在每个角色的参数定义中。但是，由于 Ceph 角色可能是唯一使用 StorageBackup 网络的角色，因此只有 Ceph 角色的 NIC 配置模板会使用模板的 network_config 部分

中的 StorageBackup 参数。

```
$network_config:
  network_config:
  - type: interface
    name: nic1
    use_dhcp: false
    addresses:
    - ip_netmask:
        get_param: StorageBackupIpSubnet
```

10.7. 自定义网络环境文件

自定义网络环境文件（本例中为 /home/stack/templates/custom-network-configuration.yaml）是一个 heat 环境文件，用于描述 overcloud 网络环境并指向自定义网络接口配置模板。您可以为网络定义子网和 VLAN，以及 IP 地址范围。然后，您可以为本地环境自定义这些值。

resource_registry 部分包含对每个节点角色的自定义网络接口模板的引用。每个注册的资源都使用以下格式：

- OS::TripleO::[ROLE]::Net::SoftwareConfig: [FILE]

[ROLE] 是角色名称，[FILE] 是该特定角色的对应网络接口模板。例如：

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/custom-nics/controller.yaml
```

parameter_defaults 部分包含一个参数列表，用于定义每种网络类型的网络选项。

10.8. 网络环境参数

下表是网络环境文件的 parameter_defaults 部分中可以使用的参数列表，以覆盖 NIC 模板中的默认参数值。

参数	描述	类型

参数	描述	类型
ControlPlaneDefaultRoute	Control Plane 上路由器的 IP 地址，用作 Controller 节点以外的角色的默认路由。如果使用 IP masquerade 而不是路由器，则将此值设置为 undercloud IP。	字符串
ControlPlaneSubnetCidr	Control Plane 上使用的 IP 网络的 CIDR 子网掩码。如果 Control Plane 网络使用 192.168.24.0/24，则 CIDR 为 **24**。	字符串（虽然始终是一个数字）
***NetCidr**	特定网络的完整网络和 CIDR 子网掩码。默认会自动设置为 **network_data.yaml** 文件中的 network **ip_subnet** 设置。例如，Internal **ApiNetCidr: 172.16.0.0/24**。	字符串
evinceAllocationPools	特定网络的 IP 分配范围。默认会自动设置为 **network_data.yaml** 文件中的网络 **allocation_pools** 设置。For example, **InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]**.	hash
***NetworkVlanID**	特定网络中节点的 VLAN ID。默认自动设置为 **network_data.yaml** 文件中的 network **vlan** 设置。例如，Internal **ApiNetworkVlanID: 201**。	number
***InterfaceDefaultRoute**	特定网络的路由器地址，可用作角色的默认路由或路由到其他网络的默认路由。默认会自动设置为 **network_data.yaml** 文件中的网络 **gateway_ip** 设置。例如，Internal **ApiInterfaceDefaultRoute: 172.16.0.1**。	字符串
DnsServers	添加到 resolv.conf 的 DNS 服务器列表。通常允许最多 2 个服务器。	以逗号分隔的列表
BondInterfaceOvsOptions	绑定接口的选项。例如，**BondInterfaceOvsOptions: "bond_mode=balance-slb"**。	字符串

参数	描述	类型
NeutronExternalNetworkBridge	用于 OpenStack Networking (neutron)的外部网桥名称 legacy 值。默认情况下，这个值为空，这意味着您可以在 **NeutronBridgeMappings** 中定义多个物理网桥。在正常情况下，请勿覆盖这个值。	字符串
NeutronFlatNetworks	定义要在 neutron 插件中配置的扁平网络。默认值为 **datacentre**，允许外部网络创建。例如，**NeutronFlatNetworks: "datacentre** "。	字符串
NeutronBridgeMappings	要使用的逻辑网桥映射。默认值将主机上的外部网桥(**br-ex**)映射到物理名称(**datacentre**)。在创建 OpenStack Networking (neutron) 提供商网络或浮动 IP 网络时，请参考逻辑名称。例如，**NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant** "。	字符串
NeutronPublicInterface	在不使用网络隔离时，为网络节点定义您要桥接到 **br-ex** 的接口。通常在只有一个网络的小型部署中不使用。例如：**NeutronPublicInterface: "eth0"。**	字符串
NeutronNetworkType	OpenStack Networking(neutron) 的租户网络类型。要指定多个值，请使用逗号分隔的列表。在所有可用网络用尽前，系统会使用您指定的第一种类型，然后会使用下一个类型。例如，Neutron **NetworkType: "vxlan** "。请注意，ML2/OVN 机制驱动程序不支持 vxlan，这是默认的 ML2 机制驱动程序。	字符串

参数	描述	类型
NeutronTunnelTypes	neutron 租户网络的隧道类型。要指定多个值，请使用逗号分隔的字符串。例如，NeutronT **unnelTypes: 'gre,vxlan'**。请注意，ML2/OVN 机制驱动程序不支持 vxlan，这是默认的 ML2 机制驱动程序。	字符串/以逗号分隔的列表
NeutronTunnelIdRanges	要用于租户网络分配的 GRE 隧道 ID 范围。例如，NeutronT **unnelIdRanges "1:1000"。**	字符串
NeutronVniRanges	要用于租户网络分配的 VXLAN VNI ID 范围。例如，**NeutronVniRanges: "1:1000"。**	字符串
NeutronEnableTunnelling	定义是否启用或禁用所有隧道网络。除非您不想在以后创建隧道网络，否则否则请保留此项。默认值为 **true**。	布尔值
NeutronNetworkVLANRanges	您要支持的 ML2 和 Open vSwitch VLAN 映射范围。默认为允许 **datacentre** 物理网络中的任何 VLAN。要指定多个值，请使用逗号分隔的列表。例如，Neutron **NetworkVLANRanges: "datacentre:1:1000,tenant:100:299,tenant:310:399"**。	字符串
NeutronMechanismDrivers	neutron 租户网络的机制驱动程序。默认值为 **ovn**。要指定多个值，请使用逗号分隔的字符串。例如，**NeutronMechanismDrivers: 'openvswitch,l2population'**.	字符串/以逗号分隔的列表

10.9. 自定义网络环境文件示例

以下片段是一个环境文件的示例，可用于启用 NIC 模板并设置自定义参数。

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig:
```

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

parameter_defaults:
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.2.254
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  NeutronExternalNetworkBridge: """
```

10.10. 使用自定义 NIC 启用网络隔离

要使用网络隔离和自定义 NIC 模板部署 overcloud，请在 overcloud 部署命令中包含所有相关网络环境文件。

流程

1. **运行 openstack overcloud deploy 命令时，包括以下文件：**

   - **自定义 network_data.yaml 文件。**
   - **默认网络隔离的呈现文件名。**
   - **默认网络环境文件的呈现文件名。**
   - **包含对自定义 NIC 模板的资源引用的自定义环境网络配置。**
   - **与配置相关的任何其他环境文件。**

例如：

```
$ openstack overcloud deploy --templates \
    ...
    -n /home/stack/network_data.yaml \
```

```
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
-e /home/stack/templates/custom-network-configuration.yaml \
...
```

- 首先包含 **network-isolation.yaml** 文件，然后包含 **network-environment.yaml** 文件。后续的 **custom-network-configuration.yaml** 覆盖了前两个文件中的 **OS::TripleO::[ROLE]::Net::SoftwareConfig** 资源。

- 如果使用可组合网络，请使用这个命令包括 **network_data.yaml** 和 **roles_data.yaml** 文件。

第 11 章 额外网络配置

本章介绍了 第 10 章 *自定义网络接口模板* 中介绍的概念和程序，并提供一些额外的信息，以帮助配置 overcloud 网络的一部分。

11.1. 配置自定义接口

单个接口可能需要修改。以下示例显示了使用第二个 NIC 连接到具有 DHCP 地址的基础架构网络所需的修改，并为绑定使用第三个和第四个 NIC：

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

网络接口模板使用实际接口名称（eth0, eth1, enp0s25）或一组编号的接口（nic1, nic2, nic3）。当使用编号接口（nic1, nic2 等）而不是命名接口（eth0, eno2 等）时，角色中的主机的网络接口不必完全相同。例如，一个主机可能具有接口 em1 和 em2，而另一个主机有 eno1 和 eno2，但您可以将两个主机的 NIC 称为 nic1 和 nic2。

编号的接口的顺序对应于命名网络接口类型的顺序：

- ethX 接口，如 eth 0、eth1 等。这些通常是载入接口。
- enoX 接口，如 eno 0、eno1 等。这些通常是载入接口。

- enX 接口，数字排序，如 enp3s 0、enp3s1、 ens3 等等。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，如果接口附加到交换机上，则接口有电缆。如果您的主机有四个接口，而有些主机有六个接口，请使用 nic1 到 nic4，并为每个主机上仅附加四个电缆。

您可以为特定节点配置 os-net-config 映射，并将别名分配给每个节点上的物理接口，以预确定哪个物理 NIC 映射到特定的别名，如 nic1 或 nic2。您还可以将 MAC 地址映射到指定的别名。您可以将接口映射到环境文件中的别名。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。以下示例将三个节点和两个带有别名的节点组配置为物理接口。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 部分看到应用的配置。

os-net-config-mappings.yaml示例

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-templates/firstboot/os-net-config-
mappings.yaml
parameter_defaults:
  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，并分配 nic1 作为此节点上 MAC 地址的别名。

2

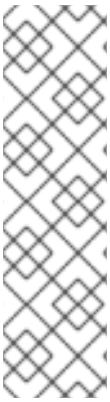
使用系统 UUID "A8C85861-1B16-4803-8689-AFC62984F8F6" 将 node3 映射到节点，并将 nic1 指定为此节点上的 em3 接口的别名。

3

dmiString 参数必须设置为一个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产品名称 "PowerEdge R630" 的节点，并分配 nic1、nic2 和 nic3 作为这些节点上命名接口的别名。



注意

- 如果要使用 NetConfigDataLookup 配置，还必须在 NodeUserData 资源 registry 中包含 os-net-config-mappings.yaml 文件。
- 通常，OS-net-config 仅注册已经以 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。

11.2. 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，DHCP 服务器提供网关地址，系统将使用该网关的默认路由。否则，您可以在带有静态 IP 的接口上设置默认路由。

虽然 Linux 内核支持多个默认网关，但它只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)是默认路由。使用以下 YAML 代码片段禁用另一个 DHCP 接口上的默认路由(nic2)：

```
# 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。

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

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

11.3. 配置基于策略的路由

在 Controller 节点上，若要配置来自不同网络的无限访问，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，即使目的地相同。

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，即使默认路由用于外部网络。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 undercloud。如需更多信息，请参阅 *Director 安装和使用指南中的安装 director*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/16.2/html-single/director_installation_and_usage/index#installing-the-undercloud
- 您已从 openstack-tripleo-heat-templates 目录中呈现了默认的 .j2 网络接口模板。更多信息请参阅 第 10.2 节 “呈现用于自定义的默认网络接口模板”。

流程

1. 从 ~/templates/custom-nics 目录在自定义 NIC 模板中创建 route_table 和 interface 条目，为接口定义路由，并定义与您的部署相关的规则：

```
$network_config:
  network_config:

  - type: route_table
    name: custom
    table_id: 200

  - type: interface
    name: em1
    use_dhcp: false
    addresses:
      - ip_netmask: 192.0.2.1/24

    routes:
      - ip_netmask: 10.1.3.0/24
        next_hop: 192.0.2.5
        route_options: "metric 10"
        table: 200
    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. 将 **run-os-net-config.sh** 脚本位置设置为您创建的每个自定义 **NIC** 模板的绝对路径。该脚本位于 **undercloud** 上的 **/usr/share/openstack-tripleo-heat-templates/network/scripts/** 目录中：

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-
config.sh
```

3. 在部署命令中包含自定义 **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 ~/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

验证

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

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

11.4. 配置巨型帧

最大传输单元(**MTU**)设置决定了通过单一以太网帧传输的最大数据量。使用较大的值会导致开销较少，因为每个帧以标头的形式添加数据。默认值为 **1500**，使用更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**。确保您在绑定或接口中包含 **MTU** 值。

存储、存储管理、内部 **API** 和租户网络都受益于巨型帧。

警告

路由器通常无法跨第 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。

```
- type: ovs_bond
  name: bond1
  mtu:
    get_param: [MaxViableMtu, value]
  ovs_options:
    get_param: BondInterfaceOvsOptions
  members:
    - type: interface
      name: nic2
      mtu: 9000
      primary: true
    - type: interface
      name: nic3
      mtu: 9000

# The external interface should stay at default
- type: vlan
  device: bond1
  vlan_id:
    get_param: ExternalNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
  routes:
    list_concat_unique
      - get_param: ExternalInterfaceRoutes
      - - default: true
          next_hop:
            get_param: ExternalInterfaceDefaultRoute

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

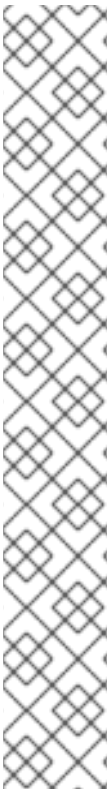
11.5. 配置 ML2/OVN 北向路径 MTU 发现，以实现巨型帧碎片

如果内部网络中的虚拟机将巨型帧发送到外部网络，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 会自动处理这种过度化数据包问题，ML2/OVN 会自动处理 TCP 数据包。

但是，为了确保正确处理使用 ML2/OVN 机制驱动程序的部署中的无线 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"fragment required"数据包返回到发送虚拟机，其中发送的应用程序会将有效负载分成较小的数据包。



注意

在 east/west 流量中，RHOSP ML2/OVN 部署不支持碎片超过 east/west 路径上最小 MTU 的数据包。例如：

- VM1 位于 Network1 上，其 MTU 为 1300。
- VM2 位于 Network2 上，其 MTU 为 1200。
- 在 VM1 和 VM2 之间以方向形式进行 ping，大小为 1171 或更少成功。大于 1171 的 ping 会导致数据包丢失百分比。

  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。

先决条件

- 带有 kernel-4.18.0-193.20.1.el8_2 或更高版本的RHEL 8.2.0.4 或更高版本。

流程

1. 检查内核版本。

```
ovs-appctl -t ovs-vswitchd dpif/show-dp-features br-int
```

2. 如果输出包含 **Check pkt length action: No**，或者输出中没有 检查 **pkt length** 操作 字符串，请升级到 **RHEL 8.2.0.4** 或更高版本，或者不会将巨型帧发送到具有较小的 **MTU** 的外部网络。

3. 如果输出包含 **Check pkt length action: Yes**，请在 **ml2_conf.ini** 的 **[ovn]** 部分中设置以下值：

```
ovn_emit_need_to_frag = True
```

11.6. 在中继接口上配置原生 VLAN

如果中继的接口或绑定在原生 **VLAN** 上有网络，则 **IP** 地址会直接分配给网桥，且没有 **VLAN** 接口。

例如，如果外部网络位于原生 **VLAN** 上，绑定的配置类似如下：

```
network_config:
  - type: ovs_bridge
    name: bridge_name
    dns_servers:
      get_param: DnsServers
    addresses:
      - ip_netmask:
          get_param: ExternalIpSubnet
    routes:
      list_concat_unique:
        - get_param: ExternalInterfaceRoutes
        - - default: true
            next_hop:
              get_param: ExternalInterfaceDefaultRoute
    members:
      - type: ovs_bond
        name: bond1
        ovs_options:
          get_param: BondInterfaceOvsOptions
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

注意

当您将地址或路由语句移到网桥时，从网桥中删除对应的 VLAN 接口。对所有适用的角色进行更改。外部网络仅存在于控制器上，因此只有控制器模板需要更改。存储网络附加到所有角色，因此如果存储网络位于默认 VLAN 上，则所有角色都需要修改。

11.7. 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的防火墙，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，并导致错误，如 nf_conntrack: 表 full，丢弃数据包。您可以提高连接跟踪(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。

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. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 创建自定义 YAML 环境文件。

示例

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

4. 您的环境文件必须包含 keywords 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一个新值。

示例

在本例中，您可以在 RHOSP 部署中的所有主机上设置 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。**

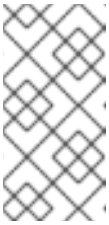
  例如，使用 ControllerParameters 为 Controller 角色设置 conntrack 限制，或 ComputeParameters 为 Compute 角色设置 conntrack 限制。

- **将 < simultaneous_connections > 替换为您要同时允许的连接数量。**

示例

在本例中，您只能为 RHOSP 部署中的 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：429 496739）。

5. **运行部署命令，包括核心 heat 模板、环境文件和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具有优先权。

示例

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

其他资源

- 环境文件
- 在 overcloud 创建中包含环境文件

第 12 章 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。

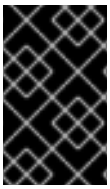
12.1. OVERCLOUD 节点的网络接口绑定

您可以将多个物理 NIC 捆绑在一起，以组成一个名为 bond 的单一逻辑频道。您可以配置绑定来为高可用性系统提供冗余功能，或提高吞吐量。

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

表 12.1. 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**。

12.2. 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：

```
...
        params:
          $network_config:
            network_config:
            - type: ovs_user_bridge
              name: br-ex
              use_dhcp: false
              members:
              - type: ovs_dpdk_bond
                name: dpdkbond0
```

```
            mtu: 2140
            ovs_options: {get_param: BondInterfaceOvsOptions}
            rx_queue:
              get_param: NumDpdkInterfaceRxQueues
            members:
            - type: ovs_dpdk_port
              name: dpdk0
              mtu: 2140
              members:
              - type: interface
                name: p1p1
            - type: ovs_dpdk_port
              name: dpdk1
              mtu: 2140
              members:
              - type: interface
                name: p1p2
```

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：

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

12.3. 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，并在流量模式发生变化时定期重新平衡。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。网络服务(neutron)将每个源 MAC 和 VLAN 对分配到一个链接，并通过该链接从该 MAC 和 VLAN 传输所有数据包。使用基于源 MAC 地址和 VLAN 号的简单哈希算法，并在流量模式更改时定期重新平衡。balance-slb 模式与 Linux 绑定驱动程序使用的 2 绑定模式类似。您可以使用此模式来提供负载平衡，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，网络服务会使一个 NIC 处于待机状态。当活跃连接失败时，待机 NIC 会恢复网络操作。物理交换机只显示一个 MAC 地址。这个模式不需要交换机配置，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平衡。

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，则将 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)。默认为载体。

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），以防止出现问题。

other_config:bond-rebalance-interval=10000

设置在绑定成员之间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。

12.4. 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 是一种协商协议，它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 绑定接口的兼容性和 LACP 选项。

重要

OVS/OVS-DPDK balance-tcp 模式仅作为技术预览提供。

重要

在控制和存储网络中，红帽建议您将 Linux 绑定与 VLAN 和 LACP 搭配使用，因为 OVS 绑定会造成 control plane 中断的可能性，这可能会在 OVS 或 neutron 代理重启进行更新、热修复和其他事件时发生。Linux bond/LACP/VLAN 配置提供 NIC 管理，而无需 OVS 中断。

表 12.2. 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**	**Active**,**passive**, 或 **off**	
增加了吞吐量（主动-主动）	**balance-slb**	**Active**,**passive**, 或 **off**	• 性能受到每个数据包的额外解析的影响。 • vhost-user 锁定争用的可能性可能会有可能。
	balance-tcp	**主动** 或 **被动**	• 仅**限技术预览**。不建议在生产环境中使用。 • L4 哈希所需的 Recirculation 会对性能产生影响。 • 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 锁定争用的可能性较高。 • 必须启用 LACP。

12.5. 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 linux 绑定来绑定两个接口：

```
...
        params:
          $network_config:
            network_config:
```

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

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 8 配置和管理网络指南中的 "根据绑定模式启用流交换配置 "。**

lacp_rate

定义 LACP 数据包是否每 1 秒发送一次，或每 30 秒发送一次。

updelay

定义接口在用于流量之前必须激活的最短时间。这个最小配置有助于缓解端口冻结中断。

miimon

使用驱动程序的 MIIMON 功能监控端口状态的时间间隔（毫秒）。

使用以下额外示例配置您自己的 Linux 绑定指南：

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

```
....
        params:
          $network_config:
            network_config:
            - type: linux_bond
              name: bond_api
              bonding_options: "mode=active-backup"
              use_dhcp: false
              dns_servers:
                get_param: DnsServers
              members:
              - type: interface
                name: nic3
                primary: true
              - type: interface
```

```
        name: nic4

    - type: vlan
      vlan_id:
        get_param: InternalApiNetworkVlanID
      device: bond_api
      addresses:
      - ip_netmask:
          get_param: InternalApiIpSubnet
```

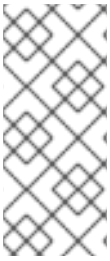
- **OVS 网桥上的 Linux 绑定。绑定设置为 802.3ad LACP 模式，它带有一个 VLAN：**

```
...
        params:
          $network_config:
            network_config:
          -  type: ovs_bridge
             name: br-tenant
             use_dhcp: false
             mtu: 9000
             members:
               - type: linux_bond
                 name: bond_tenant
                 bonding_options: "mode=802.3ad updelay=1000 miimon=100"
                 use_dhcp: false
                 dns_servers:
                   get_param: DnsServers
                 members:
                 - type: interface
                   name: p1p1
                   primary: true
                 - type: interface
                   name: p1p2
               - type: vlan
                 device: bond_tenant
                 vlan_id: {get_param: TenantNetworkVlanID}
                 addresses:
                   -
                     ip_netmask: {get_param: TenantIpSubnet}
```

第 13 章 控制节点放置

默认情况下，director 会随机选择各个角色的节点，通常根据节点的 profile 标签。但是，您还可以定义特定的节点放置。这在以下情况中很有用：

- 分配特定节点 ID，如 controller-0、controller-1
- 分配自定义主机名
- 分配特定的 IP 地址
- 分配特定的虚拟 IP 地址



注意

为网络手动设置可预测的 IP 地址、虚拟 IP 地址和端口，从而降低分配池需求。但是，建议为每个网络保留分配池，以简化扩展新节点。确保任何静态定义的 IP 地址都位于分配池之外。

13.1. 分配特定节点 ID

您可以将节点 ID 分配给特定的节点，例如 controller-0、controller-1、compute-0 和 compute-1。

流程

1. 将 ID 分配为计算调度程序在部署时匹配的每个节点功能：

```
openstack baremetal node set --property capabilities='node:controller-0,boot_option:local'
<id>
```

此命令将能力 node:controller-0 分配给节点。使用唯一连续索引重复此模式，从 0 开始用于所有节点。确保给定角色的所有节点(Controller、Compute 或每个存储角色)都以相同的方式标记，或者计算调度程序无法正确匹配该功能。

2. 创建一个 heat 环境文件（如 scheduler_hints_env.yaml），该文件使用调度程序提示来匹

配每个节点的功能：

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

使用以下参数为其他角色类型配置调度程序提示：

- Controller 节点的 ControllerSchedulerHints。
- Compute 节点的 ComputeSchedulerHints。
- Block Storage 节点的 BlockStorageSchedulerHints。
- Object Storage 节点的 ObjectStorageSchedulerHints。
- CephStorageSchedulerHints 用于 Ceph Storage 节点。
- [ROLE]SchedulerHints 用于自定义角色。将 [ROLE] 替换为角色名称。

3. 在 overcloud deploy 命令中包含 scheduler_hints_env.yaml 环境文件。

注意

节点放置优先于配置集匹配。为避免调度失败，请使用默认 baremetal 类型进行部署，而不是为配置集匹配设计的类别(计算、控制)：在环境文件中将对应的 flavor 参数设置为 baremetal：

```
parameter_defaults:
  OvercloudControllerFlavor: baremetal
  OvercloudComputeFlavor: baremetal
```

13.2. 分配自定义主机名

与 第 13.1 节 “分配特定节点 ID” 中的节点 ID 配置相结合，director 也可以为每个节点分配特定的自定

义主机名。当您需要定义系统所处的位置（如 rack2-row12）、匹配清单标识符或其他需要自定义主机名的情况时，这非常有用。

重要

部署之后不要重命名节点。在部署后重命名节点会导致实例管理问题。

流程

- 在环境文件中使用 HostnameMap 参数，如 第 13.1 节 “分配特定节点 ID” 中的 scheduler_hints_env.yaml 文件：

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

在 parameter_defaults 部分中定义 HostnameMap，并将每个映射设置为 heat 使用 HostnameFormat 参数（如 overcloud-controller-0）定义的原始主机名，第二个值是该节点所需的自定义主机名(overcloud-controller-prod-123-0)。

使用此方法与节点 ID 放置结合使用，以确保每个节点具有自定义主机名。

13.3. 分配可预测的 IP

为了进一步控制生成的环境，director 可以使用每个网络上的特定 IP 地址分配 overcloud 节点。

流程

1. 创建环境文件以定义预测的 IP 地址：

```
$ touch ~/templates/predictive_ips.yaml
```

2. 在 ~/templates/predictive_ips.yaml 文件中创建一个 parameter_defaults 部分，并使用以

下语法为每个网络上的每个节点定义预先 IP 寻址：

```
parameter_defaults:
  <role_name>IPs:
    <network>:
    - <IP_address>
    <network>:
    - <IP_address>
```

每个节点角色都有唯一的参数。将 <role_name>IPs 替换为相关参数：

- Controller 节点的 ControllerIPs。
- Compute 节点的 ComputeIPs。
- Ceph Storage 节点的 CephStorageIPs。
- BlockStorageIPs 用于块存储节点。
- Object Storage 节点的 Swift StorageIPs。
- [ROLE]IPs 用于自定义角色。将 [ROLE] 替换为角色名称。

每个参数都是网络名称到地址列表的映射。每种网络类型必须至少拥有多个地址，因为该网络上的节点将至少有多地址。director 按顺序分配地址。每种类型的第一个节点接收每个对应列表的第一个地址，第二个节点会在每个各自列表上接收第二个地址，以此类推。

例如，如果要在 overcloud 中部署三个带有预测 IP 地址的 Ceph Storage 节点，请使用以下示例语法：

```
parameter_defaults:
  CephStorageIPs:
    storage:
    - 172.16.1.100
    - 172.16.1.101
    - 172.16.1.102
    storage_mgmt:
```

```
    - 172.16.3.100
    - 172.16.3.101
    - 172.16.3.102
```

第一个 Ceph Storage 节点接收两个地址： 172.16.1.100 和 172.16.3.100。第二个接收 172.16.1.101 和 172.16.3.101，第三个接收 172.16.1.102 和 172.16.3.102。相同的模式适用于其他节点类型。

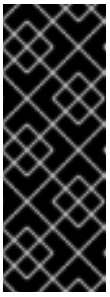
要在 control plane 上配置可预测的 IP 地址，请将 /usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-ctlplane.yaml 文件复制到 stack 用户的 templates 目录中：

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

使用以下参数配置新的 ips-from-pool-ctlplane.yaml 文件。您可以将 control plane IP 地址声明与其他网络的 IP 地址声明合并，且只使用一个文件为所有角色中的所有网络声明 IP 地址。您还可以将可预测的 IP 地址用于 spine/leaf。每个节点必须具有正确的子网的 IP 地址。

```
parameter_defaults:
  ControllerIPs:
    ctlplane:
    - 192.168.24.10
    - 192.168.24.11
    - 192.168.24.12
    internal_api:
    - 172.16.1.20
    - 172.16.1.21
    - 172.16.1.22
    external:
    - 10.0.0.40
    - 10.0.0.57
    - 10.0.0.104
  ComputeLeaf1IPs:
    ctlplane:
    - 192.168.25.100
    - 192.168.25.101
    internal_api:
    - 172.16.2.100
    - 172.16.2.101
  ComputeLeaf2IPs:
    ctlplane:
    - 192.168.26.100
    - 192.168.26.101
    internal_api:
    - 172.16.3.100
    - 172.16.3.101
```

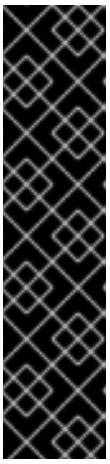
确保您选择的 IP 地址不在网络环境文件中定义的每个网络的分配池之外。例如，确保 internal_api 分配不在 InternalApiAllocationPools 范围之外，以避免与自动选择的任何 IP 冲突。另外，请确保 IP 分配与 VIP 配置不冲突，适用于标准可预测的 VIP 放置（请参阅 第 13.4 节 “分配可预测的虚拟 IP”）或外部负载均衡（请参阅 第 21.4 节 “配置外部负载均衡”）。



重要

如果删除了 overcloud 节点，请不要删除 IP 列表中的条目。IP 列表基于底层 heat 索引，即使删除了节点，此索引不会改变。要指示列表中的给定条目不再使用，请将 IP 值替换为 DELETED 或 UNUSED 等值。不应从 IP 列表中删除条目，只更改或添加。

3. 要在部署期间应用此配置，请使用 openstack overcloud deploy 命令包括 predictive_ips.yaml 环境文件。



重要

如果使用网络隔离，请在 network-isolation.yaml 文件后包括 predictive_ips.yaml 文件：

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

13.4. 分配可预测的虚拟 IP

除了为每个节点定义可预测的 IP 地址外，您还可以为集群服务定义可预测的虚拟 IP (VIP)。

流程

- 编辑网络环境文件并在 parameter_defaults 部分添加 VIP 参数：

```
parameter_defaults:
  ...
  # Predictable VIPs
  ControlFixedIPs: [{'ip_address':'192.168.201.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.0.9'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.1.1.9'}]
```

```
StorageVirtualFixedIPs: [{'ip_address':'172.18.0.9'}]
StorageMgmtVirtualFixedIPs: [{'ip_address':'172.19.0.9'}]
RedisVirtualFixedIPs: [{'ip_address':'172.16.0.8'}]
OVNDBsVirtualFixedIPs: [{'ip_address':'172.16.0.7'}]
```

从对应的分配池范围之外选择这些 IP。例如，为 InternalApiVirtualFixedIPs 选择一个不在 InternalApiAllocationPools 范围内的 IP 地址。

注意

此步骤仅适用于使用默认内部负载平衡配置的 overcloud。如果要将 VIP 分配给外部负载均衡器，请使用 Overcloud 的专用外部负载均衡 中的步骤。

第 14 章 在 OVERCLOUD 公共端点中启用 SSL/TLS

默认情况下，overcloud 将未加密的端点用于 overcloud 服务。要在 overcloud 中启用 SSL/TLS，红帽建议您使用证书颁发机构(CA)解决方案。

当您使用证书颁发机构(CA)解决方案时，您有生产就绪解决方案，如证书续订、证书撤销列表(CRL)和行业可接受的加密。有关使用 Red Hat Identity Manager (IdM)作为 CA 的详情，请参考在 Ansible 实施 TLS。

您可以使用以下手动过程只为公共 API 端点启用 SSL/TLS，内部和管理员 API 保持未加密的。如果不使用 CA，还必须手动更新 SSL/TLS 证书。如需更多信息，请参阅 手动更新 SSL/TLS 证书。

先决条件

- 网络隔离，以定义公共 API 的端点。
- openssl-perl 软件包已安装。
- 您有一个 SSL/TLS 证书。如需更多信息，请参阅 配置自定义 SSL/TLS 证书。

14.1. 初始化签名主机

签名主机是使用证书颁发机构生成并签名新证书的主机。如果您从未在所选签名主机上创建 SSL 证书，您可能需要初始化该主机，让它能够为新证书签名。

流程

1. /etc/pki/CA/index.txt 文件包含所有签名证书的记录。请确定文件系统路径和 index.txt 文件已存在：

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. /etc/pki/CA/serial 文件标识下一个序列号，以用于下一个要签名的证书。检查是否存在此文件。如果此文件不存在，则使用新启动值创建新文件：

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

14.2. 创建证书颁发机构

一般情况下，您需要使用一个外部的证书认证机构来签发您的 SSL/TLS 证书。在某些情况下，您可能想使用自己的证书颁发机构。例如，您可能想拥有仅限内部使用的证书颁发机构。

流程

1. 生成密钥和证书对以充当证书颁发机构：

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

2. openssl req 命令会要求输入认证机构的详细信息。根据提示输入相关信息。这些命令创建一个称为 ca.crt.pem 的证书颁发机构文件。

3. 将证书位置设置为 enable-tls.yaml 文件中的 PublicTLSCAFile 参数的值。当您将证书位置设置为 PublicTLSCAFile 参数的值时，您可以确保 CA 证书路径添加到 clouds.yaml 身份验证文件中。

```
parameter_defaults:
  PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

14.3. 将此证书颁发机构添加到客户端

对于旨在使用 SSL/TLS 通信的所有外部客户端，将证书颁发机构文件复制到需要访问 Red Hat OpenStack Platform 环境的每个客户端。

流程

1. 将证书颁发机构复制到客户端系统：

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

2. 将证书颁发机构文件复制到每个客户端后，在每个客户端上运行以下命令，将证书添加到证书颁发机构信任捆绑包中：

```
$ sudo update-ca-trust extract
```

14.4. 创建 SSL/TLS 密钥

在 OpenStack 环境中启用 SSL/TLS 需要一个 SSL/TLS 密钥来生成证书。

流程

1. 运行以下命令以生成 SSL/TLS 密钥 (server.key.pem)：

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

14.5. 创建 SSL/TLS 证书签名请求

完成以下步骤以创建证书签名请求。

流程

1. 复制默认 OpenSSL 配置文件：

```
$ cp /etc/pki/tls/openssl.cnf .
```

2. 编辑新的 openssl.cnf 文件并配置要用于 director 的 SSL 参数。一个要修改的参数类型的示例包括：

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
```

```
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

将 commonName_default 设置为以下条目之一：

- **如果使用 IP 地址通过 SSL/TLS 访问 director，则使用 undercloud.conf 文件中的 undercloud_public_host 参数。**
- **如果使用完全限定域名通过 SSL/TLS 访问 director，则使用此域名。**

编辑 alt_names 部分，使其包含以下条目：

- **IP - 客户端用于通过 SSL 访问 director 的 IP 地址列表。**
- **DNS - 客户端用于通过 SSL 访问 director 的域名列表。其中也包含公共 API IP 地址作为在 alt_names 部分末尾的 DNS 条目。**

注意

有关 openssl.cnf 的更多信息，请运行 man openssl.cnf 命令。

3. **运行以下命令以生成证书签名请求 (server.csr.pem)：**

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

确保使用 -key 选项包括 OpenStack SSL/TLS 密钥。

此命令生成 server.csr.pem 文件，这是证书签名请求。使用此文件创建 OpenStack SSL/TLS 证书。

14.6. 创建 SSL/TLS 证书

要为 OpenStack 环境生成 SSL/TLS 证书，必须存在以下文件：

openssl.cnf

指定 v3 扩展的自定义配置文件。

server.csr.pem

生成证书并使用 CA 对证书进行签名的证书签名请求。

ca.crt.pem

对证书进行签名的证书颁发机构。

ca.key.pem

证书颁发机构私钥。

流程

1. 如果尚未存在，请创建 newcerts 目录：

```
sudo mkdir -p /etc/pki/CA/newcerts
```

2. 运行以下命令，为 undercloud 或 overcloud 创建证书：

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

这个命令使用以下选项：

-config

使用一个自定义配置文件，它是带有 v3 扩展的 openssl.cnf 文件。

-extensions v3_req

已启用的 v3 扩展。

-days

定义证书到期前的天数。

-in'

证书签名请求。

-out

生成的签名证书。

-cert

证书颁发机构文件。

-keyfile

证书颁发机构私钥。

此命令创建名为 server.crt.pem 的新证书。将此证书与 OpenStack SSL/TLS 密钥一起使用

14.7. 启用 SSL/TLS

要在 overcloud 中启用 SSL/TLS，您必须创建一个环境文件，其中包含 SSL/TLS 认证和私钥的参数。

流程

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

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

2. 编辑此文件并对这些参数进行以下更改：

SSLCertificate

将证书文件(server.crt.pem)的内容复制到 SSLCertificate 参数中：

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGS
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQ
    -----END CERTIFICATE-----
```

重要

证书内容为所有新行需要相同的缩进级别。

SSLIntermediateCertificate

如果您有中间证书，请将中间证书的内容复制到 SSLIntermediateCertificate 参数中：

```
parameter_defaults:
  SSLIntermediateCertificate: |
    -----BEGIN CERTIFICATE-----
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvpBCwUAMFgxCzAJB
    ...
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQE
    -----END CERTIFICATE-----
```

重要

证书内容为所有新行需要相同的缩进级别。

SSLKey

将私钥(server.key.pem)的内容复制到 SSLKey 参数中：

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0IVO
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4X
    -----END RSA PRIVATE KEY-----
```

重要

私钥内容为所有新行需要相同的缩进级别。

14.8. 注入 ROOT 证书

如果证书签名者不在 overcloud 镜像的默认信任存储中，您必须将证书颁发机构注入 overcloud 镜像。

流程

1. **从 heat 模板集合中复制 inject-trust-anchor-hiera.yaml 环境文件：**

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

编辑此文件并对这些参数进行以下更改：

CAMap

列出要注入 overcloud 的每个证书颁发机构内容(CA)。overcloud 要求用于为 undercloud 和 overcloud 为证书签名的 CA 文件。将 root 证书颁发机构文件的内容(ca.crt.pem)复制到条目中。例如，您的 CAMap 参数可能类似如下：

```
parameter_defaults:
  CAMap:
    ...
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCS
        BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBw
        UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBA
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDBzCCAe+gAwIBAgIJAIc75A7FD++DMA0GCS
        BAMMD3d3dy5leGFtcGxlLmNvbTAeFw0xOTAxMz
        Um54yGCARyp3LpkxvyfMXX1DokpS1uKi7s6CkF
        -----END CERTIFICATE-----
```

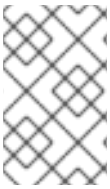
重要

证书颁发机构内容在所有新行中都需要相同的缩进级别。

您还可以使用 CAMap 参数注入其他 CA。

14.9. 配置 DNS 端点

如果您使用 DNS 主机名通过 SSL/TLS 访问 overcloud，请将 /usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml 文件复制到 /home/stack/templates 目录中。



注意

如果初始部署中不包含此环境文件，则无法使用 TLS-everywhere 架构重新部署。

为所有字段配置主机和域名，如果需要，为自定义网络添加参数：

CloudDomain

主机的 DNS 域。

CloudName

overcloud 端点的 DNS 主机名。

CloudNameCtlplane

provisioning 网络端点的 DNS 名称。

CloudNameInternal

内部 API 端点的 DNS 名称。

CloudNameStorage

存储端点的 DNS 名称。

CloudNameStorageManagement

存储管理端点的 DNS 名称。

DnsServers

要使用的 DNS 服务器列表。配置的 DNS 服务器必须包含与公共 API 的 IP 地址匹配的 CloudName 的条目。

流程

- 在参数默认值下添加要在新的或现有环境文件下使用的 DNS 服务器列表：

```
parameter_defaults:
  DnsServers: ["10.0.0.254"]
  ....
```

提示

您可以使用 CloudName{network.name} 定义，在使用虚拟 IP 的可组合网络上为 API 端点设置 DNS 名称。

如需更多信息，请参阅 添加可组合网络。

14.10. 在创建 OVERCLOUD 期间添加环境文件

将 -e 选项与部署命令 openstack overcloud deploy 一起使用，以在部署过程中包含环境文件。按照以下顺序添加本节中的环境文件：

- 启用 SSL/TLS 的环境文件(enable-tls.yaml)
- 设置 DNS 主机名的环境文件(custom-domain.yaml)
- 注入 root 证书颁发机构的环境文件(inject-trust-anchor-hiera.yaml)
- 设置公共端点映射的环境文件：
  -

如果您使用 DNS 名称来访问公共端点，请使用 /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml

- 如果您使用 IP 地址访问公共端点，请使用 /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-ip.yaml

流程

- 使用以下部署命令片断作为如何包含 SSL/TLS 环境文件的示例：

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

14.11. 手动更新 SSL/TLS 证书

如果您使用自己的 SSL/TLS 证书，这些证书不会在任何地方(TLS-e)进程中自动生成，请完成以下步骤。

流程

1. 使用以下内容编辑 heat 模板：
   - 编辑 enable-tls.yaml 文件并更新 SSLCertificate、SSLKey 和 SSLIntermediateCertificate 参数。
   - 如果您的证书颁发机构已更改，请编辑 inject-trust-anchor-hiera.yaml 文件并更新 CAMap 参数。
2. 重新运行部署命令：

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
```

```
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

3. **在 director 上，为每个 Controller 运行以下命令：**

```
ssh heat-admin@<controller> sudo podman \
restart $(podman ps --format="{{.Names}}" | grep -w -E 'haproxy(-bundle-.*-[0-9]+)?')
```

第 15 章 使用身份管理在内部和公共端点中启用 SSL/TLS

您可以在某些 overcloud 端点上启用 SSL/TLS。由于所需的证书数量，director 与红帽身份管理(IdM)服务器集成，以充当证书颁发机构并管理 overcloud 证书。

要在 OpenStack 组件中检查 TLS 支持的状态，请参阅 TLS 启用状态列表。

15.1. OPENSTACK 的身份管理(IDM)服务器建议

红帽提供了以下信息以帮助您集成 IdM 服务器和 OpenStack 环境。

有关为 IdM 安装准备 Red Hat Enterprise Linux 的详情，请参考 安装身份管理。

运行 ipa-server-install 命令来安装和配置 IdM。您可以使用命令参数跳过交互式提示。使用以下建议，以便您的 IdM 服务器可以与您的 Red Hat OpenStack Platform 环境集成：

表 15.1. 参数建议

选项	建议
--admin-password	请注意您提供的值。配置 Red Hat OpenStack Platform 以用于 IdM 时，您将需要此密码。
--ip-address	请注意您提供的值。undercloud 和 overcloud 节点需要网络访问此 ip 地址。
--setup-dns	使用这个选项在 IdM 服务器上安装集成的 DNS 服务。undercloud 和 overcloud 节点使用 IdM 服务器进行域名解析。
--auto-forwarders	使用这个选项使用 **/etc/resolv.conf** 中的地址作为 DNS 转发器。
--auto-reverse	使用这个选项解析 IdM 服务器 IP 地址的反向记录和区域。如果反向记录或区域无法解析，IdM 会创建反向区域。这简化了 IdM 部署。
--ntp-server, **--ntp-pool**	您可以使用这两个选项或其中一个选项来配置 NTP 源。IdM 服务器和 OpenStack 环境必须具有正确的和同步时间。

您必须打开 IdM 所需的防火墙端口，以启用与 Red Hat OpenStack Platform 节点的通信。如需更多信息，请参阅打开 IdM 所需的端口。

其他资源

- 配置和管理身份管理
- Red Hat Identity Management 文档

15.2. 使用 ANSIBLE 实现 TLS-E

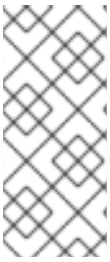
您可以使用新的 tripleo-ipa 方法在 overcloud 端点上启用 SSL/TLS，在任何位置称为 TLS (TLS-e)。由于所需的证书数量，Red Hat OpenStack Platform 与 Red Hat Identity Management (IdM)集成。当您使用 tripleo-ipa 配置 TLS-e 时，IdM 是证书颁发机构。

先决条件

确保 undercloud 的所有配置步骤（如创建 stack 用户）已完成。如需了解更多详细信息，请参阅 Director 安装和使用 以了解更多详细信息

流程

使用以下步骤在 Red Hat OpenStack Platform 的新安装或您要使用 TLS-e 配置的现有部署中实施 TLS-e。如果在预置备节点中使用 TLS-e 部署 Red Hat OpenStack Platform，则必须使用此方法。



注意

如果您要为现有环境实施 TLS-e，则需要运行命令，如 openstack undercloud install 和 openstack overcloud deploy。这些过程是幂等的，仅调整现有的部署配置，以匹配更新的模板和配置文件。

1. 配置 /etc/resolv.conf 文件：

在 /etc/resolv.conf 中设置 undercloud 上的适当的搜索域和名称服务器。例如，如果部署域是 example.com，并且 FreeIPA 服务器的域是 bigcorp.com，那么将以下行添加到 /etc/resolv.conf 中：

```
search example.com bigcorp.com
nameserver $IDM_SERVER_IP_ADDR
```

-

2. **安装所需的软件：**

```
sudo dnf install -y python3-ipalib python3-ipaclient krb5-devel
```

3. **使用特定于您的环境的值导出环境变量：**

```
export IPA_DOMAIN=bigcorp.com
export IPA_REALM=BIGCORP.COM
export IPA_ADMIN_USER=$IPA_USER (1)
export IPA_ADMIN_PASSWORD=$IPA_PASSWORD (2)
export IPA_SERVER_HOSTNAME=ipa.bigcorp.com
export UNDERCLOUD_FQDN=undercloud.example.com (3)
export USER=stack
export CLOUD_DOMAIN=example.com
```

(1) (2) **IdM 用户凭证是一个管理用户，它可以添加新主机和服务。**

(3) **UNDERCLOUD_FQDN 参数的值与 /etc/hosts 中的第一个主机名到 IP 地址映射匹配。**

4. **在 undercloud 上运行 undercloud-ipa-install.yaml ansible playbook：**

```
ansible-playbook \
--ssh-extra-args "-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" \
/usr/share/ansible/tripleo-playbooks/undercloud-ipa-install.yaml
```

5. **在 undercloud.conf 中添加以下参数**

```
undercloud_nameservers = $IDM_SERVER_IP_ADDR
overcloud_domain_name = example.com
```

6. **[可选] 如果您的 IPA 域与您的 IPA 域不匹配，请设置 certmonger_krb_realm 参数的值：**

   a. **在 /home/stack/hiera_override.yaml 中设置 certmonger_krb_realm 的值：**

   -

```
parameter_defaults:
  certmonger_krb_realm = EXAMPLE.COMPANY.COM
```

b. **将 undercloud.conf 中的 custom_env_files 参数的值设置为 /home/stack/hiera_override.yaml ：**

```
custom_env_files = /home/stack/hiera_override.yaml
```

7. **部署 undercloud：**

```
openstack undercloud install
```

验证

通过完成以下步骤验证 undercloud 是否已正确注册：

1. **列出 IdM 中的主机：**

```
$ kinit admin
$ ipa host-find
```

2. **确认 undercloud 上存在 /etc/novajoin/krb5.keytab。**

```
ls /etc/novajoin/krb5.keytab
```

注意

novajoin 目录名称仅用于传统的命名目的。

在 overcloud 上配置 TLS-e

当您随处使用 TLS 部署 overcloud 时，来自 Undercloud 和 Overcloud 的 IP 地址将自动注册到 IdM。

1. **在部署 overcloud 之前，创建一个 YAML 文件 tls-parameters.yaml，其内容类似如下。您选择的值将特定于您的环境：**

```
parameter_defaults:
```

```
  DnsSearchDomains: ["example.com"]
  DnsServers: ["192.168.1.13"]
  CloudDomain: example.com
  CloudName: overcloud.example.com
  CloudNameInternal: overcloud.internalapi.example.com
  CloudNameStorage: overcloud.storage.example.com
  CloudNameStorageManagement: overcloud.storagemgmt.example.com
  CloudNameCtlplane: overcloud.ctlplane.example.com
  IdMServer: freeipa-0.redhat.local
  IdMDomain: redhat.local
  IdMInstallClientPackages: False

resource_registry:
    OS::TripleO::Services::IpaClient: /usr/share/openstack-tripleo-heat-
templates/deployment/ipa/ipaservices-baremetal-ansible.yaml
```

- **OS::TripleO::Services::IpaClient 参数显示的值会覆盖 enable-internal-tls.yaml 文件中的默认设置。您必须确保 openstack overcloud deploy 命令中的 tls-parameters.yaml 文件遵循 enable-internal-tls.yaml。**

2. **部署 overcloud。您需要在部署命令中包含 tls-parameters.yaml ：**

```
DEFAULT_TEMPLATES=/usr/share/openstack-tripleo-heat-templates/
CUSTOM_TEMPLATES=/home/stack/templates

openstack overcloud deploy \
-e ${DEFAULT_TEMPLATES}/environments/ssl/tls-everywhere-endpoints-dns.yaml \
-e ${DEFAULT_TEMPLATES}/environments/services/haproxy-public-tls-certmonger.yaml \
-e ${DEFAULT_TEMPLATES}/environments/ssl/enable-internal-tls.yaml \
-e ${CUSTOM_TEMPLATES}/tls-parameters.yaml \
...
```

3. **通过查询 keystone 获取端点列表来确认每个端点正在使用 HTTPS ：**

```
openstack endpoint list
```

15.3. 使用 NOVAJOIN 在 RED HAT IDENTITY MANAGER (IDM)中注册节点

novajoin 是作为部署过程的一部分，用来在 Red Hat Identity Manager (IdM)中注册节点的默认工具。红帽建议通过默认的 novajoin 解决方案设置新的基于 ansible 的 tripleo-ipa 解决方案，以使用 TLS 配置 undercloud 和 overcloud。如需更多信息，请参阅使用 Ansible 实施 TLS。

您必须执行注册过程，然后才能执行 IdM 集成的其余部分。注册过程包括以下步骤：

1. 将 undercloud 节点添加到证书颁发机构(CA)

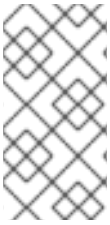
2. 将 undercloud 节点添加到 IdM

3. 可选：将 IdM 服务器设置为 overcloud 的 DNS 服务器

4. 准备环境文件并部署 overcloud

5. 在 IdM 和 RHOSP 中测试 overcloud 注册

6. 可选：在 IdM 中为 novajoin 添加 DNS 条目



注意

带有 novajoin 的 IdM 注册目前仅适用于 undercloud 和 overcloud 节点。在以后的版本中，支持用于 overcloud 实例的 novajoin 集成。

15.4. 将 UNDERCLOUD 节点添加到证书颁发机构中

在部署 overcloud 之前，请通过在 undercloud 节点上安装 python3-novajoin 软件包并运行 novajoin-ipa-setup 脚本，将 undercloud 添加到证书颁发机构(CA)中。

流程

1. 在 undercloud 节点上安装 python3-novajoin 软件包：

```
$ sudo dnf install python3-novajoin
```

2. 在 undercloud 节点上，运行 novajoin-ipa-setup 脚本，并调整值以符合您的部署：

```
$ sudo /usr/libexec/novajoin-ipa-setup \
  --principal admin \
  --password <IdM admin password> \
  --server <IdM server hostname> \
  --realm <realm> \
```

```
  --domain <overcloud cloud domain> \
  --hostname <undercloud hostname> \
  --precreate
```

使用生成的一次性密码(OTP)来注册 undercloud。

15.5. 将 UNDERCLOUD 节点添加到 RED HAT IDENTITY MANAGER (IDM)

将 undercloud 节点添加到证书颁发机构(CA)后，将 undercloud 注册到 IdM 并配置 novajoin。在 undercloud.conf 文件的 [DEFAULT] 部分中配置以下设置：

流程

1. 启用 novajoin 服务：

```
[DEFAULT]
enable_novajoin = true
```

2. 设置一次性密码(OTP)，以便您可以使用 IdM 注册 undercloud 节点：

```
ipa_otp = <otp>
```

3. 将 overcloud 的域名设置为由 neutron 的 DHCP 服务器提供：

```
overcloud_domain_name = <domain>
```

4. 为 undercloud 设置主机名：

```
undercloud_hostname = <undercloud FQDN>
```

5. 将 IdM 设置为 undercloud 的名称服务器：

```
undercloud_nameservers = <IdM IP>
```

6. 对于较大的环境，请查看 novajoin 连接超时值。在 undercloud.conf 文件中，添加对名为 undercloud-timeout.yaml 的新文件的引用：

```
hieradata_override = /home/stack/undercloud-timeout.yaml
```

在 **undercloud-timeout.yaml** 中添加以下选项。您可以指定超时值，例如 **5** ：

```
nova::api::vendordata_dynamic_connect_timeout: <timeout value>
nova::api::vendordata_dynamic_read_timeout: <timeout value>
```

7. 可选： 如果您希望本地 **openSSL** 证书颁发机构为 **director** 中的公共端点生成 **SSL** 证书，请将 **generate_service_certificate** 参数设置为 **true** ：

```
generate_service_certificate = true
```

8. 保存 **undercloud.conf** 文件。

9. 运行 **undercloud** 部署命令，将更改应用到现有的 **undercloud**：

```
$ openstack undercloud install
```

验证

通过完成以下步骤验证 **undercloud** 是否已正确注册：

1. 列出 **IdM** 中的主机：

```
$ kinit admin
$ ipa host-find
```

2. 确认 **undercloud** 上存在 **/etc/novajoin/krb5.keytab**。

```
ls /etc/novajoin/krb5.keytab
```

15.6. 将 RED HAT IDENTITY MANAGER (IDM)设置为 OVERCLOUD 的 DNS 服务器

要启用 **IdM** 环境的自动检测并方便注册，请将 **IdM** 设置为您的 **DNS** 服务器。这个过程是可选的，但推荐使用。

流程

1. 连接到 **undercloud**：

```
$ source ~/stackrc
```

2. 将 **control plane** 子网配置为使用 **IdM** 作为 **DNS** 名称服务器：

```
$ openstack subnet set ctlplane-subnet --dns-nameserver  <idm_server_address>
```

3. 在环境文件中设置 **DnsServers** 参数以使用您的 **IdM** 服务器：

```
parameter_defaults:
  DnsServers: ["<idm_server_address>"]
```

此参数通常在自定义 **network-environment.yaml** 文件中定义。

15.7. 准备环境文件并使用 NOVAJOIN 注册部署 OVERCLOUD

要使用 **IdM** 集成部署 **overcloud**，您可以创建和编辑环境文件，将 **overcloud** 配置为根据您在 **overcloud** 中定义的域使用自定义域参数 **CloudDomain** 和 **CloudName**。然后，您可以使用所有环境文件以及部署所需的任何其他环境文件来部署 **overcloud**。

流程

1. 创建 **/usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** 环境文件的副本：

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/predictable-
placement/custom-domain.yaml \
  /home/stack/templates/custom-domain.yaml
```

2. 编辑 **/home/stack/templates/custom-domain.yaml** 环境文件，并设置 **CloudDomain** 和 **CloudName bang** 值以适合您的部署：

```
parameter_defaults:
  CloudDomain: lab.local
  CloudName: overcloud.lab.local
  CloudNameInternal: overcloud.internalapi.lab.local
```

```
CloudNameStorage: overcloud.storage.lab.local
CloudNameStorageManagement: overcloud.storagemgmt.lab.local
CloudNameCtlplane: overcloud.ctlplane.lab.local
```

3. 选择适合您的环境的 **TLS** 的实施：

- 使用 **enable-tls.yaml** 环境文件来保护带有自定义证书的外部端点：

  a. 将 **/usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml** 复制到 **/home/stack/templates**。

  b. 修改 **/home/stack/enable-tls.yaml** 环境文件，使其包含您的自定义证书和密钥。

  c. 在部署中包含以下环境文件，以保护内部和外部端点：

  - **enable-internal-tls.yaml**
  - **tls-every-endpoints-dns.yaml**
  - **custom-domain.yaml**
  - **enable-tls.yaml**

```
openstack overcloud deploy \
  --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
  -e /home/stack/templates/enable-tls.yaml
```

- 使用 **haproxy-public-tls-certmonger.yaml** 环境文件来保护使用 **IdM** 发布的证书的外部端点。对于此实现，您必须为 **novajoin** 使用的 **VIP** 端点创建 **DNS** 条目：

  a. 您必须为 **novajoin** 使用的 **VIP** 端点创建 **DNS** 条目。识别位于

'/home/stack/templates 的自定义 network-environment.yaml 文件中的 overcloud 网络：

```
parameter_defaults:
  ControlPlaneDefaultRoute: 192.168.24.1
  ExternalAllocationPools:
  - end: 10.0.0.149
    start: 10.0.0.101
  InternalApiAllocationPools:
  - end: 172.17.1.149
    start: 172.17.1.10
  StorageAllocationPools:
  - end: 172.17.3.149
    start: 172.17.3.10
  StorageMgmtAllocationPools:
  - end: 172.17.4.149
    start: 172.17.4.10
```

b.

在 heat 模板中为每个 overcloud 网络创建虚拟 IP 地址列表，例如 /home/stack/public_vip.yaml。

```
parameter_defaults:
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

c.

根据需要，为每个 VIP 和区在 IdM 中添加 DNS 条目：

```
ipa dnsrecord-add lab.local overcloud --a-rec 10.0.0.101
ipa dnszone-add ctlplane.lab.local
ipa dnsrecord-add ctlplane.lab.local overcloud --a-rec 192.168.24.101
ipa dnszone-add internalapi.lab.local
ipa dnsrecord-add internalapi.lab.local overcloud --a-rec 172.17.1.101
ipa dnszone-add storage.lab.local
ipa dnsrecord-add storage.lab.local overcloud --a-rec 172.17.3.101
ipa dnszone-add storagemgmt.lab.local
ipa dnsrecord-add storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

d.

在部署中包含以下环境文件，以保护内部和外部端点：

- **enable-internal-tls.yaml**

- **tls-everywhere-endpoints-dns.yaml**
- **haproxy-public-tls-certmonger.yaml**
- **custom-domain.yaml**
- **public_vip.yaml**

```
openstack overcloud deploy \
  --templates \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-
internal-tls.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-
everywhere-endpoints-dns.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-
public-tls-certmonger.yaml \
   -e /home/stack/templates/custom-domain.yaml \
   -e /home/stack/templates/public-vip.yaml
```

注意

您不能使用 novajoin 在预先存在的部署(TLS-e)上随处实施 TLS。

其他资源

- **使用 Ansible 实现 TLS-e**

第 16 章 配置镜像导入方法和共享暂存区域

OpenStack Image 服务(glance)的默认设置由安装 Red Hat OpenStack Platform 时使用的 heat 模板决定。镜像服务 heat 模板为 deployment/glance/glance-api-container-puppet.yaml。

您可以使用以下方法导入镜像：

web-download

使用 web-download 方法从 URL 导入镜像。

glance-direct

使用 glance-direct 方法从本地卷导入镜像。

16.1. 创建并部署 GLANCE-SETTINGS.YAML 文件

使用自定义环境文件配置导入参数。这些参数覆盖核心 heat 模板集合中存在的默认值。示例环境内容包含过期镜像导入的参数。

```
parameter_defaults:
  # Configure NFS backend
  GlanceBackend: file
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.168.122.1:/export/glance

  # Enable glance-direct import method
  GlanceEnabledImportMethods: glance-direct,web-download

  # Configure NFS staging area (required for glance-direct import method)
  GlanceStagingNfsShare: 192.168.122.1:/export/glance-staging
```

GlanceBackend、GlanceNfsEnabled 和 GlanceNfsShare 参数在 *高级 Overcloud 自定义指南*中的存储配置部分中定义。

使用两个新参数来导入镜像导入，以定义导入方法和共享 NFS 暂存区域。

GlanceEnabledImportMethods

定义可用的导入方法、web-download （默认）和 glance-direct。只有在您希望启用除 web-download 外的其他方法时才需要此参数。

GlanceStagingNfsShare

配置 glance-direct 导入方法使用的 NFS staging 区域。此空间可以在高可用性集群配置中的节点间共享。如果要使用此参数，还必须将 GlanceNfsEnabled 参数设置为 true。

流程

1. 创建一个新文件，如 glance-settings.yaml。使用示例中的语法填充此文件。

2. 在 openstack overcloud deploy 命令中包含 glance-settings.yaml 文件，以及与部署相关的任何其他环境文件：

```
$ openstack overcloud deploy --templates -e glance-settings.yaml
```

有关使用环境文件的更多信息，请参阅*高级 Overcloud 自定义指南*中的 Overcloud 中包含的环境文件部分。

16.2. 控制镜像 WEB-IMPORT 源

您可以通过在可选的 glance-image-import.conf 文件中添加 URI 块列表和允许列表来限制 web-import 镜像下载的源。

您可以在三个级别上允许或阻止镜像源 URI：

- scheme (allowed_schemes, disallowed_schemes)
- host (allowed_hosts, disallowed_hosts)
- port (allowed_ports, disallowed_ports)

如果您在任何级别上同时指定了允许列表和 blocklist，则允许列表将被忽略。

镜像服务(glance)应用以下决策逻辑来验证镜像源 URI：

1. 检查方案。

   a. 缺少方案： reject

   b. 如果存在允许列表，且方案没有存在于允许列表中： reject。否则，跳过 C 并继续 2。

   c. 如果有一个 blocklist，且方案存在于 blocklist 中： reject。

2. 检查主机名。

   a. 缺少主机名： reject

   b. 如果存在允许列表，且允许列表中没有主机名： reject。否则，跳过 C 并继续 3。

   c. 如果有一个 blocklist，并且 blocklist 中存在主机名： reject。

3. 如果 URI 中有一个端口，则会检查端口。

   a. 如果存在允许列表，并且允许列表中没有该端口： reject。否则，跳过 B 并继续 4。

   b. 如果有一个 blocklist，并且 blocklist 中存在的端口： reject。

4. URI 被接受为有效。

如果您允许一个方案，可以通过将其添加到允许列表，或者不将其添加到黑名单中，允许任何使用该方案的默认端口的 URI （不允许在 URI 中包含端口）。如果 URI 中包含了一个端口，则 URI 会根据默认的决策逻辑进行验证。

16.3. 镜像导入示例

例如，FTP 的默认端口是 21。因为 *ftp* 是一个允许列表，所以允许 URL ftp://example.org/some/resource，但由于 21 不在端口允许列表中，到同一资源的 URL ftp://example.org:21/some/resource 会被拒绝。

```
allowed_schemes = [http,https,ftp]
disallowed_schemes = []
allowed_hosts = []
disallowed_hosts = []
allowed_ports = [80,443]
disallowed_ports = []
```

16.4. 默认镜像导入块列表和允许列表设置

glance-image-import.conf 文件是一个可选文件，包含以下默认选项：

- allowed_schemes - [*http*, *https*]
- disallowed_schemes - empty list
- allowed_hosts - 空列表
- disallowed_hosts - empty list
- allowed_ports - [80, 443]
- disallowed_ports - empty list

如果使用默认值，则最终用户只能使用 http 或 https 方案访问 URI。用户可以指定的唯一端口是 80 和 443。用户不必指定端口，但如果这样做，它必须是 80 或 443。

您可以在镜像服务源代码树的 etc/ 子目录中找到 glance-image-import.conf 文件。确保您正在查找 Red Hat OpenStack Platform 发行版本的正确分支。

16.5. 在镜像导入中注入元数据来控制虚拟机启动的位置

最终用户可以将镜像上传到镜像服务，并使用这些镜像启动虚拟机。这些用户提供的（非管理员）镜像必须在特定的一组计算节点上启动。实例分配给计算节点由镜像元数据属性控制。

镜像属性注入插件在导入过程中将元数据属性注入镜像。通过编辑 glance-image-import.conf 文件的 [image_import_opts] 和 [inject_metadata_properties] 部分来指定属性。

要启用镜像属性注入插件，请在 [image_import_opts] 部分添加以下行：

```
[image_import_opts]
image_import_plugins = [inject_image_metadata]
```

要将元数据注入由特定用户提供的镜像注入，请设置 ignore_user_roles 参数。例如，使用以下配置将 property1 的值和 property2 的两个值注入任何非 admin 用户下载的镜像中。

```
[DEFAULT]
[image_conversion]
[image_import_opts]
image_import_plugins = [inject_image_metadata]
[import_filtering_opts]
[inject_metadata_properties]
ignore_user_roles = admin
inject = PROPERTY1:value,PROPERTY2:value;another value
```

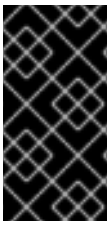
参数 ignore_user_roles 是插件忽略的 Identity 服务(keystone)角色的逗号分隔列表。这意味着，如果使镜像导入调用的用户具有任何这些角色，则插件不会将任何属性注入镜像中。

参数 注入 是一个以逗号分隔的属性和值列表，它们注入到导入的镜像的镜像记录中。每个属性和值都必须加引号 (':')。

您可以在镜像服务源代码树的 etc/ 子目录中找到 glance-image-import.conf 文件。确保您正在查找 Red Hat OpenStack Platform 发行版本的正确分支。

第 17 章 存储配置

本章概述了可用于为您的 overcloud 配置存储选项的几种方法。



重要

overcloud 将本地临时存储和逻辑卷管理器(LVM)存储用于默认存储选项。在生产环境中支持本地临时存储，但不支持 LVM 存储。

17.1. 配置 NFS 存储

您可以将 overcloud 配置为使用共享 NFS 存储。

17.1.1. 支持的配置和限制

支持的 NFS 存储

- 红帽建议您使用经认证的存储后端和驱动程序。红帽不推荐使用来自通用 NFS 后端的 NFS 存储，因为它的功能与认证的存储后端和驱动程序相比有限制。例如，通用 NFS 后端不支持卷加密和卷多附加等功能。有关支持的驱动程序的详情，请查看 红帽生态系统目录。

- 对于 Block Storage (cinder)和 Compute (nova)服务，您必须使用 NFS 版本 4.0 或更高版本。Red Hat OpenStack Platform (RHOSP)不支持早期版本的 NFS。

不支持的 NFS 配置

- RHOSP 不支持 NetApp 功能 NAS 安全，因为它会干扰正常的卷操作。director 默认禁用该功能。因此，不要编辑以下 heat 参数，用于控制 NFS 后端还是 NetApp NFS Block Storage 后端是否支持 NAS 安全：

  - CinderNetappNasSecureFileOperations
  - CinderNetappNasSecureFilePermissions
  - CinderNasSecureFileOperations

- CinderNasSecureFilePermissions

使用 NFS 共享时的限制

- 当后端是 NFS 共享时，具有交换磁盘的实例无法调整大小或重新构建。

17.1.2. 配置 NFS 存储

您可以将 overcloud 配置为使用共享 NFS 存储。

流程

1. 创建一个环境文件来配置 NFS 存储，如 nfs_storage.yaml。

2. 在新环境文件中添加以下参数来配置 NFS 存储：

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableNfsBackend: true
  GlanceBackend: file
  CinderNfsServers: 192.0.2.230:/cinder
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
```

注意

不要配置 CinderNfsMountOptions 和 GlanceNfsOptions 参数，因为它们的默认值启用适合大多数 Red Hat OpenStack Platform (RHOSP)环境的 NFS 挂载选项。您可以在 environments/storage/glance-nfs.yaml 文件中看到 GlanceNfsOptions 参数的值。如果您在配置多个服务以共享相同的 NFS 服务器时遇到问题，请联系红帽支持。

3. 使用其他环境文件将 NFS 存储环境文件添加到堆栈中，并部署 overcloud：

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

17.1.3. 配置外部 NFS 共享以进行转换

当块存储服务(cinder)在 overcloud Controller 节点上执行镜像格式转换，且空间有限时，大型镜像服务(glance)镜像转换可能会导致完全使用节点 root 磁盘空间。您可以使用外部 NFS 共享进行转换，以防止完全填充节点上的空间。

有两个 director heat 参数控制外部 NFS 共享配置：

- CinderImageConversionNfsShare
- CinderImageConversionNfsOptions

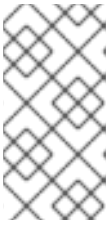
流程

1. 以 stack 用户身份登录 undercloud，再提供 stackrc 凭证文件。

```
$ source ~/stackrc
```

2. 在新的或现有与存储相关的环境文件中，添加有关外部 NFS 共享的信息。

```
parameter_defaults:
  CinderImageConversionNfsShare: 192.168.10.1:/convert
```



注意

控制 NFS 挂载选项的 CinderImageConversionNfsOptions 参数的默认值足以满足大多数环境中。

3. 在 openstack overcloud deploy 命令中包含新配置的环境文件，以及与您的环境相关的任何其他环境文件。

```
$ openstack overcloud deploy \
--templates \
…
-e <existing_overcloud_environment_files> \
-e <new_environment_file> \
…
```

- 使用作为现有部署一部分的环境文件列表替

换。<existing_overcloud_environment_files>

- 将 <new_environment_file > 替换为包含 NFS 共享配置的新的或编辑的环境文件。

17.2. 配置 CEPH STORAGE

使用以下方法之一将 Red Hat Ceph Storage 集成到 Red Hat OpenStack Platform overcloud 中。

使用自己的 Ceph Storage 集群创建 overcloud

您可以在 overcloud 上创建 Ceph 存储集群。director 会创建一组使用 Ceph OSD 存储数据的 Ceph Storage 节点。director 还会在 overcloud Controller 节点上安装 Ceph Monitor 服务。这意味着，如果组织创建具有三个高可用性 Controller 节点的 overcloud，Ceph 监控器也会成为高可用性服务。有关更多信息，请参阅使用容器化 Red Hat Ceph 部署 Overcloud。

将现有 Ceph Storage 集群集成到 overcloud 中

如果您有一个现有的 Ceph Storage 集群，您可以在部署过程中将此集群集成到 Red Hat OpenStack Platform overcloud 中。这意味着您可以在 overcloud 配置外管理和扩展集群。有关更多信息，请参阅将 Overcloud 与现有 Red Hat Ceph 集群集成。

17.3. 使用外部对象存储集群

您可以通过在 Controller 节点上禁用默认对象存储服务部署来重复使用外部 OpenStack Object Storage (swift)集群。这会禁用 Object Storage 的代理和存储服务，并配置 haproxy 和 OpenStack 标识(keystone)，以使用给定的外部对象存储端点。

注意

您必须手动管理外部 Object Storage (swift)集群上的用户帐户。

先决条件

- 您需要外部 Object Storage 集群的端点 IP 地址，以及来自外部 Object Storage proxy-server.conf 文件的 authtoken 密码。您可以使用 openstack endpoint list 命令查找此信息。

流程

1. 使用以下内容创建名为 swift-external-params.yaml 的新文件：

- 将 **EXTERNAL.IP:PORT** 替换为外部代理的 **IP** 地址和端口，
- 将 **AUTHTOKEN** 替换为 **SwiftPassword** 行上外部代理的 **authtoken** 密码。

```
parameter_defaults:
  ExternalSwiftPublicUrl: 'https://EXTERNAL.IP:PORT/v1/AUTH_%(tenant_id)s'
  ExternalSwiftInternalUrl: 'http://192.168.24.9:8080/v1/AUTH_%(tenant_id)s'
  ExternalSwiftAdminUrl: 'http://192.168.24.9:8080'
  ExternalSwiftUserTenant: 'service'
  SwiftPassword: AUTHTOKEN
```

2. 将此文件保存为 **swift-external-params.yaml**。
3. 使用以下外部对象存储服务环境文件以及与部署相关的任何其他环境文件，部署 **overcloud**：

```
openstack overcloud deploy --templates \
-e [your environment files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml \
-e swift-external-params.yaml
```

17.4. 配置 CEPH 对象存储以使用外部 CEPH 对象网关

Red Hat OpenStack Platform (RHOSP) Director 支持将外部 **Ceph** 对象网关(RGW)配置为对象存储服务。要使用外部 **RGW** 服务进行身份验证，您必须配置 **RGW**，以验证用户及其在 **Identity** 服务 **(keystone)**中的角色。

有关如何配置外部 **Ceph** 对象网关的更多信息，请参阅 ***Using Keystone with the Ceph Object Gateway Guide*** 中的 **Configuring the Ceph Object Gateway to use Keystone authentication**。

流程

1. 将以下 **parameter_defaults** 添加到自定义环境文件，如 **swift-external-params.yaml**，并调整值以符合您的部署：

```
parameter_defaults:
  ExternalSwiftPublicUrl: 'http://<Public RGW endpoint or
loadbalancer>:8080/swift/v1/AUTH_%(project_id)s'
  ExternalSwiftInternalUrl: 'http://<Internal RGW endpoint>:8080/swift/v1/AUTH_%
(project_id)s'
```

```
ExternalSwiftAdminUrl: 'http://<Admin RGW endpoint>:8080/swift/v1/AUTH_%(project_id)s'
ExternalSwiftUserTenant: 'service'
SwiftPassword: 'choose_a_random_password'
```

注意

示例代码片段包含与您在环境中使用的值不同的参数值：

- **远程 RGW 实例侦听 8080 的默认端口。端口可能根据配置外部 RGW 的不同而有所不同。**
- **overcloud 中创建的 swift 用户使用 SwiftPassword 参数定义的密码。您必须将外部 RGW 实例配置为使用同一密码通过 rgw_keystone_admin_password 与 Identity 服务进行身份验证。**

2. **将以下代码添加到 Ceph 配置文件，将 RGW 配置为使用 Identity 服务。替换变量值以适合您的环境：**

```
rgw_keystone_api_version = 3
rgw_keystone_url = http://<public Keystone endpoint>:5000/
rgw_keystone_accepted_roles = member, Member, admin
rgw_keystone_accepted_admin_roles = ResellerAdmin, swiftoperator
rgw_keystone_admin_domain = default
rgw_keystone_admin_project = service
rgw_keystone_admin_user = swift
rgw_keystone_admin_password =
<password_as_defined_in_the_environment_parameters>
rgw_keystone_implicit_tenants = true
rgw_keystone_revocation_interval = 0
rgw_s3_auth_use_keystone = true
rgw_swift_versioning_enabled = true
rgw_swift_account_in_url = true
```

注意

director 默认在 Identity 服务中创建以下角色和用户：

- **rgw_keystone_accepted_admin_roles: ResellerAdmin, swiftoperator**
- **rgw_keystone_admin_domain: default**
- **rgw_keystone_admin_project: service**
- **rgw_keystone_admin_user: swift**

3. **使用与部署相关的任何其他环境文件，使用额外的环境文件部署 overcloud：**

```
openstack overcloud deploy --templates \
-e <your_environment_files>
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml
-e swift-external-params.yaml
```

验证

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

```
$ source ~/stackrc
```

3. **验证 Identity 服务(keystone)中是否存在端点：**

```
$ openstack endpoint list --service object-store

+---------+-----------+-------+-------+---------+-----------+---------------+
| ID | Region    | Service Name | Service Type | Enabled | Interface | URL |
+---------+-----------+-------+-------+---------+-----------+---------------+
| 233b7ea32aaf40c1ad782c696128aa0e | regionOne | swift | object-store | True    | admin     |
http://192.168.24.3:8080/v1/AUTH_%(project_id)s |
```

```
| 4ccde35ac76444d7bb82c5816a97abd8 | regionOne | swift | object-store | True    | public    |
https://192.168.24.2:13808/v1/AUTH_%(project_id)s |
| b4ff283f445348639864f560aa2b2b41 | regionOne | swift | object-store | True    | internal  |
http://192.168.24.3:8080/v1/AUTH_%(project_id)s |
+---------+-----------+-------+-------+---------+-----------+---------------+
```

4. **创建测试容器：**

```
$ openstack container create <testcontainer>
+---------------+--------------+----------------------------------+
| account | container | x-trans-id |
+---------------+--------------+----------------------------------+
| AUTH_2852da3cf2fc490081114c434d1fc157 | testcontainer | tx6f5253e710a2449b8ef7e-
005f2d29e8 |
+---------------+--------------+----------------------------------+
```

5. **创建配置文件以确认您可以将数据上传到容器：**

```
$ openstack object create testcontainer undercloud.conf
+----------------+--------------+----------------------------------+
| object          | container     | etag                             |
+----------------+--------------+----------------------------------+
| undercloud.conf | testcontainer | 09fcffe126cac1dbac7b89b8fd7a3e4b |
+----------------+--------------+----------------------------------+
```

6. **删除 test 容器：**

```
$ openstack container delete -r <testcontainer>
```

17.5. 为镜像服务配置 CINDER 后端

使用 GlanceBackend 参数设置镜像服务用来存储镜像的后端。

重要

您可以为项目创建的默认最大卷数量为 10。

流程

1. **要将 cinder 配置为镜像服务后端，请将以下行添加到环境文件中：**

```
parameter_defaults:
  GlanceBackend: cinder
```

2. 如果启用了 cinder 后端，则默认设置以下参数和值：

```
cinder_store_auth_address = http://172.17.1.19:5000/v3
cinder_store_project_name = service
cinder_store_user_name = glance
cinder_store_password = ****secret****
```

3. 要使用自定义用户名或 cinder_store_ 参数的任何自定义值，请将 ExtraConfig 参数添加到 parameter_defaults 中，并包含您的自定义值：

```
ExtraConfig:
    glance::config::api_config:
      glance_store/cinder_store_auth_address:
        value: "%{hiera('glance::api::authtoken::auth_url')}/v3"
      glance_store/cinder_store_user_name:
        value: <user-name>
      glance_store/cinder_store_password:
        value: "%{hiera('glance::api::authtoken::password')}"
      glance_store/cinder_store_project_name:
        value: "%{hiera('glance::api::authtoken::project_name')}"
```

17.6. 配置附加到一个实例的存储设备的最大数量

默认情况下，您可以将无限数量的存储设备附加到单个实例。要限制最大设备数量，请在计算环境文件中添加 max_disk_devices_to_attach 参数。使用以下示例将 max_disk_devices_to_attach 的值改为 "30"：

```
parameter_defaults:
  ComputeExtraConfig:
        nova::config::nova_config:
          compute/max_disk_devices_to_attach:
            value: '30'
```

指南和注意事项

- 实例支持的存储磁盘数量取决于磁盘使用的总线。例如，IDE 磁盘总线限制为 4 个附加的设备。

- 如果在带有活跃实例的 Compute 节点上更改 max_disk_devices_to_attach，如果最大数低于已附加到实例的设备数，则可能会导致重新构建失败。例如，如果实例 A 附加了 26 个设备，

并且将 max_disk_devices_to_attach 改为 20，则重建实例 A 的请求将失败。

- 在冷迁移过程中，只有您要迁移的实例的源强制配置的最大存储设备数量。移动前不会检查目的地。这意味着，如果 Compute 节点 A 具有 26 个附加磁盘设备，而 Compute 节点 B 则配置最多 20 个附加磁盘设备，则有 26 个附加设备的实例从 Compute 节点 A 到 Compute 节点 B 成功。但是，后续在 Compute 节点 B 中重建实例的请求会失败，因为已附加了 26 个设备超过配置的最大 20 个。

- 在 shelved 卸载实例上不会强制配置的最大值，因为它们没有 Compute 节点。

- 将大量磁盘设备附加到实例可能会降低实例上的性能。根据您的环境可以支持的界限，调整最大数量。

- 具有机器类型 Q35 的实例可以附加最多 500 个磁盘设备。

17.7. 提高镜像服务缓存的可扩展性

使用 glance-api 缓存机制将镜像副本存储在镜像服务(glance) API 服务器上，并自动检索它们以提高可扩展性。使用镜像服务缓存时，glance-api 可以在多个主机上运行。这意味着，它不需要多次从后端存储检索同一镜像。镜像服务缓存不会影响任何镜像服务操作。

使用 Red Hat OpenStack Platform director (tripleo) heat 模板配置镜像服务缓存：

流程

1. 在环境文件中，将 GlanceCacheEnabled 参数的值设置为 true，这会在 glance-api.conf heat 模板中自动将 flavor 值设置为 keystone+cachemanagement ：

```
parameter_defaults:
  GlanceCacheEnabled: true
```

2. 在重新部署 overcloud 时，将环境文件包含在 openstack overcloud deploy 命令中。

3. 可选：在重新部署 overcloud 时，将 glance_cache_pruner 变为替代频率。以下示例显示了 5 分钟的频率：

```
parameter_defaults:
  ControllerExtraConfig:
    glance::cache::pruner::minute: '*/5'
```

根据您的需要调整频率，以避免文件系统的完整场景。选择替代频率时包括以下元素：

- 要在环境中缓存的文件大小。
- 可用文件系统空间量。
- 环境缓存镜像的频率。

17.8. 配置第三方存储

以下环境文件存在于核心 heat 模板集合 /usr/share/openstack-tripleo-heat-templates 中。

Dell EMC Storage Center

为 Block Storage (cinder)服务部署单个 Dell EMC Storage Center 后端。

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

Dell EMC PS 系列

为 Block Storage (cinder)服务部署单个 Dell EMC PS 系列后端。

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

NetApp Block Storage

将 NetApp 存储设备部署为 Block Storage (cinder)服务的后端。

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

第 18 章 安全增强

以下小节提供了一些强化 overcloud 安全性的建议。

18.1. 使用安全 ROOT 用户访问

overcloud 镜像自动包含 root 用户的强化安全性。例如，每个部署的 overcloud 节点都会自动禁用到 root 用户的直接 SSH 访问。您仍然可以访问 overcloud 节点上的 root 用户。

流程

1. 以 stack 用户身份登录 undercloud 节点。
2. 每个 overcloud 节点都有一个 heat-admin 用户帐户。此用户帐户包含 undercloud 公共 SSH 密钥，它提供了 SSH 访问，而无需从 undercloud 到 overcloud 节点的密码。在 undercloud 节点上，以 heat-admin 用户身份通过 SSH 登录 overcloud 节点。
3. 通过 sudo -i 切换到 root 用户。

18.2. 管理 OVERCLOUD 防火墙

每个核心 OpenStack Platform 服务都包含其相应的可组合服务模板中的防火墙规则。这会自动为每个 overcloud 节点创建一组默认的防火墙规则。

overcloud heat 模板包含一组参数，它们可以帮助提供额外的防火墙管理：

ManageFirewall

定义是否自动管理防火墙规则。将此参数设置为 true，以允许 Puppet 在每个节点上自动配置防火墙。如果要手动管理防火墙，设置为 false。默认值是 true。

PurgeFirewallRules

定义是否在配置新 Linux 防火墙规则前清除默认的 Linux 防火墙规则。默认值为 false。

如果将 ManageFirewall 参数设置为 true，您可以在部署时创建额外的防火墙规则。使用 overcloud 的环境文件中的配置 hook 设置 tripleo::firewall::firewall_rules hieradata （请参阅 第 4.5 节

“puppet：自定义角色的 hieradata”）。这个 hieradata 是一个哈希，其中包含防火墙规则名称及其对应参数作为键，它们都是可选的：

port

与规则关联的端口。

dport

与规则关联的目标端口。

sport

与规则关联的源端口。

Proto

与规则关联的协议。默认为 tcp。

action

与规则关联的操作策略。默认为 接受。

jump

要跳到的链。如果存在，它将覆盖 操作。

state

与规则关联的状态数组。默认为 ['NEW']。

source

与规则关联的源 IP 地址。

iniface

与规则关联的网络接口。

链

与规则关联的链。默认为 INPUT。

目的地

与规则关联的目标 CIDR。

以下示例演示了防火墙规则格式的语法：

```
ExtraConfig:
  tripleo::firewall::firewall_rules:
    '300 allow custom application 1':
      port: 999
      proto: udp
      action: accept
    '301 allow custom application 2':
      port: 8081
      proto: tcp
      action: accept
```

这将通过 ExtraConfig 将两个额外的防火墙规则应用到所有节点。

注意

每个规则名称都会成为对应 iptables 规则的注释。每个规则名称都以三位前缀开头，以帮助 Puppet 遵循最终 iptables 文件中定义的所有规则。默认 Red Hat OpenStack Platform 规则使用 000 到 200 范围内的前缀。

18.3. 更改简单网络管理协议(SNMP)字符串

director 为您的 overcloud 提供默认的只读 SNMP 配置。建议更改 SNMP 字符串，以减少未授权用户了解您的网络设备的风险。

注意

使用字符串参数配置 ExtraConfig 接口时，您必须使用以下语法来确保 heat 和 Hiera 不将字符串解释为布尔值："'<VALUE> '"。

在 overcloud 的环境文件中使用 ExtraConfig hook 设置以下 hieradata：

SNMP 传统访问控制设置

snmp::ro_community

IPv4 只读 SNMP 社区字符串。默认值为 public。

snmp::ro_community6

IPv6 只读 SNMP 社区字符串。默认值为 public。

snmp::ro_network

允许 RO 查询 守护进程的网络。这个值可以是字符串或数组。默认值为 127.0.0.1。

snmp::ro_network6

允许 RO 查询 IPv6 守护进程的网络。这个值可以是字符串或数组。默认值为 ::1/128。

tripleo::profile::base::snmp::snmpd_config

要作为安全 valve 添加到 *snmpd.conf* 文件中的行数组。默认值为 []。有关所有可用选项，请参阅 SNMP 配置文件网页。

例如：

```
parameter_defaults:
  ExtraConfig:
    snmp::ro_community: mysecurestring
    snmp::ro_community6: myv6securestring
```

这会更改所有节点上的只读 SNMP 社区字符串。

SNMP 基于视图的访问控制设置(VACM)

snmp::com2sec

VACM com2sec 映射的数组。必须提供 SECNAME、SOURCE 和 COMMUNITY。

snmp::com2sec6

VACM com2sec6 映射的数组。必须提供 SECNAME、SOURCE 和 COMMUNITY。

例如：

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: ["notConfigUser default mysecurestring"]
    snmp::com2sec6: ["notConfigUser default myv6securestring"]
```

这会更改所有节点上的只读 SNMP 社区字符串。

如需更多信息，请参阅 snmpd.conf 手册页。

18.4. 更改 HAPROXY 的 SSL/TLS 密码和规则

如果在 overcloud 中启用了 SSL/TLS，请考虑强化与 HAProxy 配置一起使用的 SSL/TLS 密码和规则。通过强化 SSL/TLS 密码，您可以帮助避免 SSL/TLS 漏洞，如 POODLE 漏洞。

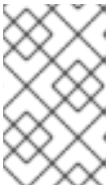
1. 创建名为 tls-ciphers.yaml 的 heat 模板环境文件：

```
touch ~/templates/tls-ciphers.yaml
```

2. 使用环境文件中的 ExtraConfig hook 将值应用到 tripleo::haproxy::ssl_cipher_suite 和 tripleo::haproxy::ssl_options hieradata：

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: 'DHE-RSA-AES128-CCM:DHE-RSA-AES256-
CCM:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES128-CCM:ECDHE-ECDSA-AES256-CCM:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-CHACHA20-POLY1305'

    tripleo::haproxy::ssl_options: 'no-sslv3 no-tls-tickets'
```



注意

cipher 集合是一个连续行。

3. 在部署 overcloud 时，使用 overcloud deploy 命令包括 tls-ciphers.yaml 环境文件：

```
openstack overcloud deploy --templates \
...
-e /home/stack/templates/tls-ciphers.yaml
...
```

18.5. 使用 OPEN VSWITCH 防火墙

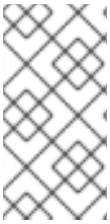
您可以将安全组配置为使用 Red Hat OpenStack Platform director 中的 Open vSwitch (OVS)防火墙驱动程序。使用 NeutronOVSFirewallDriver 参数指定您要使用的防火墙驱动程序：

- iptables_hybrid - 配置网络服务(neutron)以使用基于 iptables/hybrid 的实现。
- openvswitch - 配置网络服务以使用基于 OVS 防火墙流的驱动程序。

openvswitch 防火墙驱动程序包括更高的性能，并减少用于将客户机连接到项目网络的接口和网桥数量。

重要

多播流量由 Open vSwitch (OVS)防火墙驱动程序与 iptables 防火墙驱动程序不同。默认情况下，VRRP 流量被拒绝，您必须在安全组规则中启用 VRRP 流量来访问端点。使用 OVS 时，所有端口共享相同的 OpenFlow 上下文，并且多播流量无法为每个端口单独处理。由于安全组不适用于所有端口（例如，路由器上的端口），所以 OVS 使用 IANA 操作，并将多播流量转发到 RFC 4541 指定的所有端口。



注意

iptables_hybrid 选项与 OVS-DPDK 不兼容。openvswitch 选项与 OVS 硬件卸载不兼容。

在 network-environment.yaml 文件中配置 NeutronOVSFirewallDriver 参数：

```
NeutronOVSFirewallDriver: openvswitch
```

- NeutronOVSFirewallDriver ：配置您要在实施安全组时使用的防火墙驱动程序的名称。可能的值取决于您的系统配置。一些示例包括 noop、openvswitch 和 iptables_hybrid。空字符串的默认值会产生支持的配置。

第 19 章 配置网络插件

director 包括在配置第三方网络插件时可以使用的环境文件：

19.1. FUJITSU CONVERGED FABRIC (C-FABRIC)

您可以使用位于 /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml 的环境文件来启用 Fujitsu Converged Fabric (C-Fabric)插件。

流程

1. 将环境文件复制到您的 模板 子目录中：

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml
/home/stack/templates/
```

2. 编辑 resource_registry 以使用绝对路径：

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuCfab: /usr/share/openstack-tripleo-heat-
templates/puppet/services/neutron-plugin-ml2-fujitsu-cfab.yaml
```

3. 查看 /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml 中的 parameter_defaults ：

- NeutronFujitsuCfabAddress - C-Fabric 的 telnet IP 地址（字符串）
- NeutronFujitsuCfabUserName - 要使用的 C-Fabric 用户名。（字符串）
- NeutronFujitsuCfabPassword - C-Fabric 用户帐户的密码。（字符串）
- NeutronFujitsuCfabPhysicalNetworks - <physical_network>:<vfab_id& gt; tuples 列表，用于指定 physical_network 名称及其对应的 vfab ID。(comma_ separated_list)
- NeutronFujitsuCfabSharePprofile - 确定是否在使用相同的 VLAN ID 的 neutron 端口间共享 C-Fabric pprofile。（布尔值）

- **NeutronFujitsuCfabPprofilePrefix - pprofile 名称的前缀字符串。（字符串）**
- **NeutronFujitsuCfabSaveConfig - 确定是否保存配置。（布尔值）**

4. **要将模板应用到部署，请在 openstack overcloud deploy 命令中包含环境文件：**

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS] ...
```

19.2. FUJITSU FOS 交换机

您可以使用位于 /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml 的环境文件来启用 Fujitsu FOS Switch 插件。

流程

1. **将环境文件复制到您的 模板 子目录中：**

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml /home/stack/templates/
```

2. **编辑 resource_registry 以使用绝对路径：**

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuFossw: /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-plugin-ml2-fujitsu-fossw.yaml
```

3. **查看 /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml 中的 parameter_defaults：**

- **NeutronFujitsuFosswIps - 所有 FOS 交换机的 IP 地址。(comma_ separated_list)**
- **NeutronFujitsuFosswUserName - 要使用的 FOS 用户名。（字符串）**

- **NeutronFujitsuFosswPassword - FOS 用户帐户的密码。（字符串）**
- **NeutronFujitsuFosswPort - 用于 SSH 连接的端口号。（数字）**
- **NeutronFujitsuFosswTimeout - SSH 连接的超时时间。（数字）**
- **NeutronFujitsuFosswUdpDestPort - FOS 交换机上的 VXLAN UDP 目标的端口号。（数字）**
- **NeutronFujitsuFosswOvsdbVlanidRangeMin - 用于绑定 VNI 和物理端口的范围中的最小 VLAN ID。（数字）**
- **NeutronFujitsuFosswOvsdbPort - FOS 交换机上 OVSDB 服务器的端口号。（数字）**

4. **要将模板应用到部署，请在 openstack overcloud deploy 命令中包含环境文件：**

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

第 20 章 配置身份

director 包括有助于配置 Identity Service (keystone)设置的参数：

20.1. 区域名称

默认情况下，overcloud 区域命名为 regionOne。您可以通过添加 KeystoneRegion 条目来更改该文件。在部署 overcloud 后，您无法修改这个值。

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

第 21 章 其它 OVERCLOUD 配置

使用以下配置在 overcloud 中配置各种功能。

21.1. 调试模式

您可以为 overcloud 中的某些服务启用和禁用 DEBUG 级别日志记录模式。

要为服务配置调试模式，请设置对应的 debug 参数。例如，OpenStack Identity (keystone)使用 KeystoneDebug 参数。

流程

- 在环境文件的 parameter_defaults 部分中设置参数：

```
parameter_defaults:
  KeystoneDebug: True
```

将 KeystoneDebug 参数设置为 True 后，/var/log/containers/keystone/keystone.log 标准 keystone 日志文件会使用 DEBUG 级别日志进行更新。

如需 debug 参数的完整列表，请参阅 *Overcloud 参数指南*中的"Debug 参数 "。

21.2. 在 OVERCLOUD 节点上配置内核

Red Hat OpenStack Platform director 包括在 overcloud 节点上配置内核的参数。

ExtraKernelModules

要载入的内核模块。模块名称列为带有空值的 hash 键：

```
ExtraKernelModules:
  <MODULE_NAME>: {}
```

ExtraKernelPackages

在从 ExtraKernelModules 加载内核模块前要安装的内核相关软件包。软件包名称列为带有空值

的 hash 键。

```
ExtraKernelPackages:
  <PACKAGE_NAME>: {}
```

ExtraSysctlSettings

要应用的 sysctl 设置哈希。使用 value 键设置每个参数的值。

```
ExtraSysctlSettings:
  <KERNEL_PARAMETER>:
    value: <VALUE>
```

本例演示了环境文件中的这些参数语法：

```
parameter_defaults:
  ExtraKernelModules:
    iscsi_target_mod: {}
  ExtraKernelPackages:
    iscsi-initiator-utils: {}
  ExtraSysctlSettings:
    dev.scsi.logging_level:
      value: 1
```

21.3. 配置服务器控制台

overcloud 节点的控制台输出并不总是发送到服务器控制台。如果要在服务器控制台中查看此输出，您必须将 overcloud 配置为使用适合您的硬件控制台。使用以下方法之一执行此配置：

- 修改每个 overcloud 角色的 KernelArgs heat 参数。
- 自定义 director 用来置备 overcloud 节点的 overcloud-full.qcow2 镜像。

前提条件

- 成功安装 undercloud。如需更多信息，请参阅 Director 安装和使用指南。
- overcloud 节点已准备好进行部署。

在部署过程中 使用 heat 修改内核参数

1. **以 stack 用户身份登录 undercloud 主机。**

2. **查找 stackrc 凭证文件：**

```
$ source stackrc
```

3. **使用以下内容创建环境文件 overcloud-console.yaml ：**

```
parameter_defaults:
  <role>Parameters:
    KernelArgs: "console=<console-name>"
```

将 & lt;role > 替换为您要配置的 overcloud 角色的名称，并将 < console-name > 替换为您要使用的控制台 ID。例如，使用以下代码片段将默认角色中的所有 overcloud 节点配置为使用 tty0 ：

```
parameter_defaults:
  ControllerParameters:
    KernelArgs: "console=tty0"
  ComputeParameters:
    KernelArgs: "console=tty0"
  BlockStorageParameters:
    KernelArgs: "console=tty0"
  ObjectStorageParameters:
    KernelArgs: "console=tty0"
  CephStorageParameters:
    KernelArgs: "console=tty0"
```

4. **使用 -e 选项，在部署命令中包含 overcloud-console-tty0.yaml 文件。**

修改 overcloud-full.qcow2 镜像

1. **以 stack 用户身份登录 undercloud 主机。**

2. **查找 stackrc 凭证文件：**

```
$ source stackrc
```

3. 修改 overcloud-full.qcow2 镜像中的内核参数，为您的硬件设置正确的控制台。例如，将控制台设置为 tty0 ：

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'grubby --update-kernel=ALL --args="console=tty0"'
```

4. 将镜像导入 director ：

```
$ openstack overcloud image upload --image-path /home/stack/images/overcloud-full.qcow2
```

5. 部署 overcloud。

验证

1. 从 undercloud 登录 overcloud 节点：

```
$ ssh heat-admin@<IP-address>
```

将 <IP-address > 替换为 overcloud 节点的 IP 地址。

2. 检查 /proc/cmdline 文件的内容，并确保 console= 参数设置为您要使用的控制台值：

```
[heat-admin@controller-0 ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-4.18.0-193.29.1.el8_2.x86_64
root=UUID=0ec3dea5-f293-4729-b676-5d38a611ce81 ro console=tty0
console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet
```

21.4. 配置外部负载均衡

overcloud 将多个 Controller 用作高可用性集群，这样可确保 OpenStack 服务的最大操作性能。另外，集群提供了对 OpenStack 服务的访问的负载均衡，这会均匀将流量分发到 Controller 节点，并减少每个节点的服务器超载。您还可以使用外部负载均衡器来执行此分发。例如，您可以使用自己的基于硬件的负载均衡器来处理到 Controller 节点的流量分布。

有关配置外部负载平衡的更多信息，请参阅 Overcloud 指南的专用外部负载平衡。

21.5. 配置 IPV6 联网

本节检查 overcloud 的网络配置。这包括隔离 OpenStack 服务以使用特定网络流量并使用 IPv6 选项配置 overcloud。