# Red Hat OpenStack Platform 13

## 高级 Overcloud 自定义

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

Last Updated: 2023-04-20

# Red Hat OpenStack Platform 13 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

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 摘要

本指南说明了如何使用 Red Hat OpenStack Platform Director 为 Red Hat OpenStack Platform 企业环境配置某些高级功能。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

# 第 1 章 简介

Red Hat OpenStack Platform director 提供了一组工具来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 Overcloud 的准备和配置。但是，正确的生产级别 Overcloud 可能需要额外的配置，包括：

- 将 Overcloud 集成到现有的网络基础架构中的基本网络配置。
- 网络流量隔离在单独的 VLAN 上，用于某些 OpenStack 网络流量类型。
- 用于保护公共端点上的通信的 SSL 配置
- 存储选项，如 NFS、iSCSI、红帽 Ceph 存储以及多个第三方存储设备。
- 将节点注册到 Red Hat Content Delivery Network 或您的内部 Red Hat Satellite 5 或 6 服务器。
- 各种系统级别选项。
- 各种 OpenStack 服务选项。

本指南提供了通过 director 增强 Overcloud 的说明。此时，director 已注册节点，并配置了用于 Overcloud 创建所需的服务。现在，您可以使用本指南中的方法自定义 Overcloud。

**注意**

本指南中的示例是配置 Overcloud 的可选步骤。只有提供 Overcloud 的额外功能才需要这些步骤。仅使用适用于您环境需求的步骤。

# 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，以便您可以在 Red Hat OpenStack Platform director 的上下文了解这些模板的结构和格式。

## 2.1. HEAT 模板

Red Hat OpenStack Platform (RHOSP) director 使用 Heat 编配模板(HOT)作为其 overcloud 部署计划的模板格式。HOT 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 heat 创建的资源集合，以及资源的配置。*资源是 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。*

***注意***

*要使 RHOSP 使用 heat 模板文件作为自定义模板资源，文件扩展必须是* ***.yaml*** *或* ***.template****。*

*Heat 模板有三个主要部分：*

***参数***

*这些设置传递到 heat，以自定义堆栈。您还可以使用 heat 参数自定义默认值。这些设置在模板的* ***parameter*** *部分中定义。*

***Resources***

*这些是作为堆栈一部分创建和配置的具体对象。Red Hat OpenStack Platform (RHOSP) 包含跨越所有组件的一组核心资源。它们在模板的* ***resources*** *部分中定义。*

***输出***

*在创建堆栈后，这些值从 heat 传递。您可以通过 heat API 或客户端工具访问这些值。它们在模板的* ***output*** *部分中定义。*

*以下是基本 heat 模板的示例：*

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

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

resources:
  my_instance:
    type: OS::Nova::Server
```

```
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

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

*此模板使用 **资源类型：OS::Nova::Server** 创建名为 **my_instance** 的实例，其具有特定类别、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，它为模板创建一个堆栈，并为资源模板创建一组子堆栈。这会创建一个堆栈的层次结构，该堆栈从您通过模板定义的主堆栈中分离。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

*环境文件是特殊的模板，可为您的 heat 模板提供自定义。这包括三个关键部分：*

***资源 Registry***

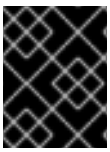
*本节定义链接到其他 heat 模板的自定义资源名称。这提供了一种方法，可以创建在核心资源集合中不存在的自定义资源。它们在环境文件的 **resource_registry** 部分中定义。*

***参数***

*这些是适用于顶级模板参数的通用设置。例如，如果您有一个部署嵌套堆栈（如资源 registry 映射）的模板，这些参数仅适用于顶级模板，而不是嵌套资源的模板。参数在环境文件的 **parameters** 部分中定义。*

***参数默认值***

*这些参数为所有模板中的参数修改默认值。例如，如果您有一个部署嵌套堆栈的 heat 模板，如资源 registry 映射，则参数默认为所有模板。参数默认值在环境文件的 **parameter_defaults** 部分中定义。*



***重要***

*为 overcloud 创建自定义环境文件时，请使用 **parameter_defaults** 而不是 **参数**。这样的参数将应用到 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 heat 模板 **my_template.yaml** 创建堆栈时，可能会包含环境文件 **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，为这个资源类型提供实现，可覆盖任何内置文件。您可以在* ***my_template.yaml*** *文件中包含* ***OS::Nova::Server::MyServer*** *资源。*

***MyIP*** *将参数应用于使用此环境文件部署的主要 heat 模板。在本例中，它只适用于* ***my_template.yaml*** *中的参数。*

***NetworkName*** *适用于主 heat 模板* ***my_template.yaml*** *和与包含主模板的资源关联的模板，如* ***OS::Nova::Server::MyServer*** *资源及其* ***myserver.yaml*** *模板。*

***注意***

*要使 RHOSP 使用 heat 模板文件作为自定义模板资源，文件扩展必须是* ***.yaml*** *或* ***.template****。*

## 2.3. 核心 OVERCLOUD HEAT 模板

*director 包含 Overcloud 的核心 heat 模板集合。此集合保存在* ***/usr/share/openstack-tripleo-heat-templates*** *中。*

*这个模板集合中的主要文件和目录是：*

***overcloud.j2.yaml***

*这是创建 overcloud 环境的主要模板文件。此文件使用 Jinja2 语法来迭代模板中的某些部分，以创建自定义角色。在 overcloud 部署过程中将呈现 Jinja2 格式到 YAML 中。*

***overcloud-resource-registry-puppet.j2.yaml***

*这是创建 overcloud 环境的主要环境文件。它为存储在 overcloud 镜像上的 Puppet 模块提供了一组配置。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 语法来迭代模板中的某些部分，以创建自定义角色。在 overcloud 部署过程中将呈现 Jinja2 格式到 YAML 中。*

***roles_data.yaml***

*这是一个文件，它在 overcloud 中定义角色并将服务映射到各个角色。*

***network_data.yaml***

*这是在 overcloud 中定义网络的文件，以及子网、分配池和 VIP 状态等属性。默认* ***network_data*** *文件包含默认网络：External、In Internal Api、Storage、Storage Management、Tenant 和 Management。您可以创建自定义* ***network_data*** *文件，并使用* ***-n*** *选项将其添加到* ***openstack overcloud deploy*** *命令中。*

***plan-environment.yaml***

*这是定义 overcloud 计划元数据的文件。这包括要使用的计划名称、要使用的主要模板，以及应用到 overcloud 的环境文件。*

***capabilities-map.yaml***

*这是 overcloud 计划的环境文件映射。使用此文件描述并在 director Web UI 上启用环境文件。在 overcloud 计划中的* ***环境*** *目录中检测到的自定义环境文件，但在* ***capabilities-map.yaml*** *中没有定义，在 web UI* ***中指定 Deployment Configuration > Overall Settings 的 Other*** *子选项卡中列出。*

***environments***

*包含可用于创建 overcloud 的额外 heat 环境文件。这些环境文件为生成的 Red Hat OpenStack Platform (RHOSP)环境启用额外的功能。例如，目录包含一个环境文件，用于启用 Cinder NetApp 后端存储(****cinder-netapp-config.yaml****)。在此目录中检测到的任何环境文件（在* ***capabilities-map.yaml*** *文件中未定义）都会在* ***2 的其它 子选项卡中列出。在 director 的 Web UI 中指定 Deployment Configuration > Overall Settings*** *中。*

***network***

这是一组有助于创建隔离的网络和端口的 heat 模板。

**puppet**

这些是主要由 Puppet 配置驱动的模板。**overcloud-resource-registry-puppet.j2.yaml** 环境文件使用此目录中的文件来驱动每个节点上的 Puppet 配置应用。

**Puppet/服务**

这是包含可组合服务架构中所有服务的 heat 模板的目录。

**extraconfig**

这些是启用额外功能的模板。

**firstboot**

提供 director 在最初创建节点时使用 **的第一个_boot** 脚本示例。

## 2.4. 计划环境元数据

计划环境元数据文件允许您定义 overcloud 计划的元数据。此信息用于导入和导出 overcloud 计划，并在您的计划创建 overcloud 期间使用。

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 计划和容器的名称。

**模板**

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

**environments**

定义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路径。

**parameter_defaults**

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

**密码**

用于 overcloud 密码的一组参数。这个功能的方式与标准环境文件中的 **parameter_defaults** 部分相同。通常，director 使用随机生成的密码自动填充这个部分。

**workflow_parameters**

允许您将一组参数提供给 OpenStack Workflow (mistral) 命名空间。您可以使用它来计算和自动生成某些 overcloud 参数。

以下是计划环境文件的语法示例：

```
version: 1.0
name: myovercloud
description: 'My Overcloud Plan'
template: overcloud.yaml
environments:
- path: overcloud-resource-registry-puppet.yaml
- path: environments/docker.yaml
- path: environments/docker-ha.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** 命令包含计划环境文件。例如：*

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

*您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. 功能映射

*capabilities map 提供规划及其依赖项中的环境文件映射。使用此文件通过 director 的 Web UI 描述和启用环境文件。在 overcloud 计划中检测到的自定义环境文件，但没有列在 **capabilities-map.yaml** 中，它们列在 **其他** 子选项卡中的 2 **子选项卡中。指定 web UI 上的 Deployment Configuration > Overall Settings**。*

*默认文件位于 **/usr/share/openstack-tripleo-heat-templates/capabilities-map.yaml**。*

*以下是功能映射的语法示例：*

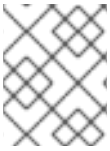
```
topics: 1
  - title: My Parent Section
    description: This contains a main section for different environment files
    environment_groups: 2
      - name: my-environment-group
        title: My Environment Group
        description: A list of environment files grouped together
        environments: 3
          - file: environment_file_1.yaml
            title: Environment File 1
            description: Enables environment file 1
            requires: 4
              - dependent_environment_file.yaml
          - file: environment_file_2.yaml
            title: Environment File 2
            description: Enables environment file 2
            requires: 5
              - dependent_environment_file.yaml
          - file: dependent_environment_file.yaml
            title: Dependent Environment File
            description: Enables the dependent environment file
```

1. **topics** 参数包含 UI 部署配置中的章节列表。每个主题都显示为一个环境选项屏幕，其中包含多个环境组，它们使用 **environment_groups** 参数进行定义。每个主题都可以有纯文本 **标题** 和 **描述**。

2. **environment_groups** 参数列出了 UI 部署配置中的环境文件分组。例如，在存储主题上，您可能有一个用于 Ceph 相关的环境文件的环境组。每个环境组都有纯文本 **标题** 和 **描述**。

3. **environment** 参数显示属于环境组的所有环境文件。**file** 参数是环境文件的位置。每个环境条目都可以具有纯文本 **标题** 和 **描述**。

4 5. **requires** 参数是环境文件的依赖项列表。在本例中，**environment_file_1.yaml** 和 **environment_file_2.yaml** 都要求您启用 **dependent_environment_file.yaml**。



**注意**

Red Hat OpenStack Platform 使用此文件向 director UI 添加对功能的访问。建议不要修改此文件，因为新版本的 Red Hat OpenStack Platform 可能会覆盖这个文件。

## 2.6. 在 OVERCLOUD 创建中包含环境文件

部署命令(**openstack overcloud deploy**)使用 **-e** 选项包括环境文件来自定义 Overcloud。您可以根据需要纳入多个环境文件。但是，环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源更为优先。例如，您可能有两个环境文件：

**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'
```

然后，使用包括的两个环境文件进行部署：

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

在本例中，两个环境文件都包含通用资源类型(**OS::TripleO::NodeExtraConfigPost**)和通用参数(**TimeZone**)。**openstack overcloud deploy** 命令通过以下进程运行：

1. 根据 **--template** 选项，从核心 Heat 模板集合中加载默认配置。
2. 从 **environment-file-1.yaml** 应用配置，这将覆盖默认配置中的任何常见设置。
3. 应用 **environment-file-2.yaml** 的配置，它会覆盖默认配置和 **environment-file-1.yaml** 中的任何常见设置。

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**，因为 **environment-file-2.yaml**。
- **timezone** 参数设为 **Hongkong**，因为 **environment-file-2.yaml**。
- **RabbitFDLimit** 参数根据 **environment-file-1.yaml** 被设置为 **65536**。**environment-file-2.yaml** 不会更改这个值。

这提供了一种从多个环境文件冲突地定义自定义配置到 Overcloud 的方法。

## 2.7. 使用自定义核心 HEAT 模板

在创建 overcloud 时，director 使用位于 **/usr/share/openstack-tripleo-heat-templates** 中的一组核心 Heat 模板。如果要自定义此核心模板集合，请使用 Git 工作流来跟踪更改和合并更新。使用以下 git 进程来帮助管理自定义模板集合：

### 初始化自定义模板集合

使用以下步骤创建包含 Heat 模板集合的初始 Git 存储库：

1. 将模板集合复制到 **stack** 用户目录。这个示例将集合复制到 **~/templates** 目录：

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

2. 进入自定义模板目录并初始化 Git 存储库：

```
$ cd openstack-tripleo-heat-templates
$ git init .
```

3. 配置您的 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。

4. 为初始提交暂存所有模板：

```
$ git add *
```

5. 创建初始提交：

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

这会创建一个包含最新核心模板集合的初始 **master** 分支。使用此分支作为自定义分支的基础，并将新模板版本合并到此分支。

### 创建自定义分支和提交更改

使用自定义分支将您的更改存储到核心模板集合中。使用以下步骤创建 **my-customizations** 分支并向其添加自定义：

1. 创建 **my-customizations** 分支并切换到它：

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

2. 编辑自定义分支中的文件。

3. 在 git 中暂存更改：

```
$ git add [edited files]
```

4. 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 rebase **my-customizations** off **master**，这会导致 git 把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，并在将来的模板更新时重新播放它们。

### 更新自定义模板集合：

更新 undercloud 时，**openstack-tripleo-heat-templates** 软件包也会更新。当发生这种情况时，请按照以下步骤更新自定义模板集合：

1. 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)
```

2. 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
```

3. 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/* .
```

4. 为初始提交添加所有模板：

```
$ git add *
```

5. 为软件包更新创建提交：

```
$ git commit -m "Updates for $PACKAGE"
```

6. 将分支合并到 master 中。如果使用 Git 管理系统（如 GitLab）则使用管理工作流。如果在本地使用 git，请切换到 **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 rebase **my-customization** 分支。

### 重新调整自定义分支

使用以下步骤更新 **my-customization** 分支：

1. 进入 **my-customizations** 分支：

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

2. 将分支重基为 **master** ：

```
$ git rebase master
```

这会更新 **my-customizations** 分支，并重播向此分支发出的自定义提交。

如果 git 在 rebase 期间报告任何冲突，请使用以下步骤：

1. 检查哪些文件包含冲突：

```
$ git status
```

2. 解决标识的模板文件冲突。
3. 添加解析的文件

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

4. 继续 rebase ：

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

### 部署自定义模板

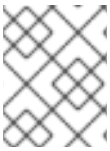
使用以下步骤部署自定义模板集合：

1. 确保已切换到 **my-customization** 分支：

```
git checkout my-customizations
```

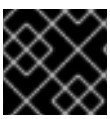
2. 使用 **--templates** 选项运行 **openstack overcloud deploy** 命令，以指定您的本地模板目录：

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



**注意**

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



**重要**

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

## 2.8. JINJA2 渲染

**/usr/share/openstack-tripleo-heat-templates** 中的核心 Heat 模板包含多个以 **j2.yaml** 扩展名结尾的文件。这些文件包含 Jinja2 模板语法，director 会把这些文件呈现给其以 **.yaml** 结尾的静态 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。

支持 Jinja2 的 Heat 模板使用 Jinja2 语法为迭代值创建参数和资源。例如，**overcloud.j2.yaml** 文件包含以下代码片段：

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

当 director 呈现 Jinja2 语法时，director 会迭代 **roles_data.yaml** 文件中定义的角色，并使用角色的名称填充 **{{role.name}}Count** 参数。默认 **roles_data.yaml** 文件包含五个角色，并生成以下示例中的以下参数：

- ***ControllerCount***
- ***ComputeCount***
- ***BlockStorageCount***
- ***ObjectStorageCount***
- ***CephStorageCount***

参数渲染版本示例如下：

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

director 只会在核心 Heat 模板的 目录中呈现 Jinja2enabled 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**)。director 将 **network-isolation.j2.yaml** 文件呈现到 **network-isolation.yaml** 中。在运行 **openstack overcloud deploy** 命令时，使用 **-e** 选项包括 rendered **network-isolation.yaml** 文件的名称。

```
$ openstack ovecloud 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 在自定义核心模板中显示 **network-isolation.j2.yaml** 文件，并呈现到 **network-isolation.yaml** 中。在运行 **openstack overcloud deploy** 命令时，使用 **-e** 选项包括 rendered **network-isolation.yaml** 文件的名称。*

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

***使用案例 3：增加使用量***

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

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

*此 director 使用自定义核心模板位置(**--templates /home/stack/tripleo-heat-templates**)。但是，所选的 **network-isolation.j2.yaml** 不在自定义核心模板中，因此不会 呈现给 **network-isolation.yaml**。这会导致部署失败。*

# 第 3 章 参数

director 模板集合中每个 Heat 模板都包含一个 **parameters** 部分。本节定义特定于特定 overcloud 服务的所有参数。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
- **Puppet/services/*.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** 命令包含环境文件。

接下来的几个部分包含演示如何为 **puppet/services** 目录中的服务配置特定参数的示例。

## 3.1. 示例 1：配置时区

用于设置时区的 Heat 模板(**puppet/services/time/timezone.yaml**)包含 **TimeZone** 参数。如果将 **TimeZone** 参数留空，则 overcloud 会将时间设置为 **UTC** 作为默认值。

要获取时区列表，请运行 **timedatectl list-timezones** 命令。以下示例命令检索亚洲的时区：

```
$ sudo timedatectl list-timezones|grep "Asia"
```

在识别时区后，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：启用网络分布式虚拟路由(DVR)

OpenStack Networking (neutron) API 的 Heat 模板(**puppet/services/neutron-api.yaml**)包含一个参数，用于启用和禁用分布式虚拟路由(DVR)。参数的默认值为 **false**。但是，您可以在环境文件中使用以下内容启用它：

```
parameter_defaults:
  NeutronEnableDVR: true
```

## 3.3. 示例 3：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。**puppet/services/rabbitmq.yaml** Heat 模板允许您将 **RabbitFDLimit** 参数设置为您需要的限制。将以下内容添加到环境文件。

```
parameter_defaults:
  RabbitFDLimit: 65536
```

## 3.4. 示例 4：启用和禁用参数

在某些情况下，您可能需要首先在部署期间设置参数，然后为将来的部署操作禁用 参数，如更新或扩展操作。例如，要在 overcloud 创建过程中包括自定义 RPM，您需要包括以下内容：

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

如果您需要从将来的部署中禁用这个参数，则它不足以移除该参数。相反，您要将参数设置为空值：

```
parameter_defaults:
  DeployArtifactURLs: []
```

这样可确保不再为后续部署操作设置该参数。

## 3.5. 示例 5：基于角色的参数

使用 **[ROLE]Parameters** 参数，将 **[ROLE]** 替换为可组合角色，为特定角色设置参数。

例如，director 在 Controller 和 Compute 节点上配置 **logrotate**。要为 Controller 和 Compute 节点设置不同的 **logrotate** 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 logrotate 参数：

```
parameter_defaults:
  ControllerParameters:
    LogrotateMaxsize: 10M
    LogrotatePurgeAfterDays: 30
  ComputeParameters:
    LogrotateMaxsize: 20M
    LogrotatePurgeAfterDays: 15
```

## 3.6. 识别要修改的参数

Red Hat OpenStack Platform director 为配置提供了多个参数。在某些情况下，您可能难以识别要配置的特定选项以及对应的 director 参数。如果您需要通过 director 配置的选项，请使用以下工作流来识别并将选项映射到特定的 overcloud 参数：

1. 确定您要配置的选项。记录使用 选项的服务。
2. 选中对应的 Puppet 模块，以用于此选项。Red Hat OpenStack Platform 的 Puppet 模块位于 director 节点上的 **/etc/puppet/modules** 下。每个模块对应于特定的服务。例如，**keystone** 模块对应于 OpenStack Identity (keystone)。
   - 如果 Puppet 模块包含控制所选选项的变量，请转到下一步。
   - 如果 Puppet 模块不包含控制所选选项的变量，则此选项没有 hieradata。若有可能，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。**puppet/services/*** 中的模板通常对应于同一服务的 Puppet 模块。例如，**puppet/services/keystone.yaml** 模板为 **keystone** 模块提供 hieradata。
   - 如果 Heat 模板为 Puppet 变量设置了 hieradata，该模板也应披露 director 型参数进行修改。

- 如果 Heat 模板没有为 Puppet 变量设置 hieradata，请使用配置 hook 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：为角色自定义 Hieradata"。

**重要**

不要定义同一自定义 hieradata 哈希的多个实例。同一自定义 hieradata 的多个实例可能会导致 Puppet 运行期间出现冲突，并导致为配置选项设置了意外值。

### 工作流示例

您可能要更改 OpenStack Identity (keystone) 的通知格式。使用工作流，您可以：

1. 识别要配置的 OpenStack 参数(**notification_format**)。
2. 在 **keystone** Puppet 模块中搜索 **notification_format** 设置。例如：

```
$ grep notification_format /etc/puppet/modules/keystone/manifests/*
```

在本例中，**keystone** 模块使用 **keystone::notification_format** 变量管理此选项。

3. 为此变量搜索 **keystone** 服务模板。例如：

```
$ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-templates/puppet/services/keystone.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 Parameter | Puppet Hieradata | OpenStack Identity (keystone) 选项 |
|---|---|---|
| **KeystoneNotificationFormat** | **keystone::notification_format** | **notification_format** |

这意味着，在 overcloud 环境文件中设置 **KeystoneNotificationFormat**，将在 overcloud 的配置期间在 **keystone.conf** 文件中设置 **notification_format** 选项。

# 第 4 章 配置 HOOK

配置 hook 提供了一种将您自己的配置功能注入 Overcloud 部署过程的方法。这包括在主 Overcloud 服务配置和 hook （包括基于 Puppet 的配置）前后注入自定义配置的 hook。

## 4.1. 首次启动：自定义第一个引导配置

director 提供了一种机制，可在创建 Overcloud 的初始节点上执行配置。director 通过 **cloud-init** 达到此目的，您可以使用 **OS::TripleO::NodeUserData** 资源类型进行调用。

在本例中，您将使用所有节点上的自定义 IP 地址更新名称服务器。您必须首先创建一个基本的 heat 模板 (**/home/stack/templates/nameserver.yaml**)，该脚本将运行一个脚本，将每个节点的 **resolv.conf** 附加到特定名称服务器。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}
```

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

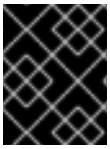
```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

要添加第一次引导配置，请在首先创建 Overcloud 时将环境文件添加到堆栈中，以及其他环境文件。例如：

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

**-e** 将环境文件应用到 Overcloud 堆栈。

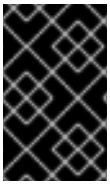
这会在所有节点首次创建并首次引导时将配置添加到所有节点。这些模板的后续包含不会运行这些脚本，如更新 Overcloud 堆栈。



**重要**

您只能将 **OS::TripleO::NodeUserData** 注册到一个 heat 模板。后续使用会覆盖要使用的 heat 模板。

## 4.2. PRE-CONFIGURATION：自定义特定 OVERCLOUD 角色



**重要**

本文档的早期版本使用 **OS::TripleO::Tasks::*PreConfig** 资源来为每个角色提供预配置 hook。director 的 Heat 模板集合需要专用于使用这些 hook，这意味着您不应该将它们用于自定义用途。反之，请使用下面概述的 **OS::TripleO::*ExtraConfigPre** hook。

Overcloud 使用 Puppet 作为 OpenStack 组件的核心配置。director 提供了一组 hook，用于在第一次引导完成并开始核心配置前为特定节点角色提供自定义配置。这些 hook 包括：

**OS::TripleO::ControllerExtraConfigPre**

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

**OS::TripleO::ComputeExtraConfigPre**

在 Puppet 核心配置之前，应用到 Compute 节点的额外配置。

**OS::TripleO::CephStorageExtraConfigPre**

在核心 Puppet 配置之前，应用到 Ceph Storage 节点的额外配置。

**OS::TripleO::ObjectStorageExtraConfigPre**

在 Puppet 核心配置之前，应用到 Object Storage 节点的额外配置。

**OS::TripleO::BlockStorageExtraConfigPre**

在 Puppet 核心配置之前，应用到块存储节点的附加配置。

**OS::TripleO::[ROLE]ExtraConfigPre**

在 Puppet 核心配置之前，应用到自定义节点的额外配置。用可组合角色名称替换 **[ROLE]**。

在本例中，您首先创建一个基本的 heat 模板(**/home/stack/templates/nameserver.yaml**)，它将运行一个脚本来写入具有变量名称服务器的节点的 **resolv.conf**。

```
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，**Resources** 部分包含以下内容：*

**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 **nameserver_ip** 参数中存储的值。*

**CustomExtraDeploymentPre**

*这会执行软件配置，这是来自 **CustomExtraConfigPre** 资源的软件配置。注意以下几点：*

- *配置参数引用 **CustomExtraConfigPre** 资源，因此 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，是此 **hook** 模板中的强制要求。*
- ***actions** 参数定义何时应用配置。在这种情况下，我们仅在创建或更新 **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，它存储来自父模板中的 **DeployIdentifier**。此参数为每个部署更新的资源提供时间戳。这可确保在后续的 **overcloud** 更新中资源恢复。*

***接下来，创建一个环境文件(/home/stack/templates/pre_config.yaml)，将 heat 模板注册到基于角色的资源类型。例如，要仅应用到 Controller 节点，请使用 ControllerExtraConfigPre hook：***

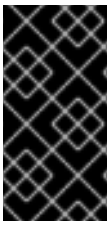
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

要应用配置，请在创建或更新 Overcloud 时将环境文件添加到堆栈中，以及其他环境文件。例如：

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

这会在核心配置开始初始 Overcloud 创建或后续更新时，将配置应用到所有 Controller 节点。



重要

您只能为每个 hook 只注册每个资源的一个 Heat 模板。后续使用会覆盖要使用的 Heat 模板。

### 4.3. PRE-CONFIGURATION：自定义所有 OVERCLOUD 角色

Overcloud 使用 Puppet 作为 OpenStack 组件的核心配置。director 提供了一个 hook，用于在第一次引导完成后以及启动核心配置前配置所有节点类型：

**OS::TripleO::NodeExtraConfig**

在核心 Puppet 配置之前，将额外的配置应用到所有节点角色。

在本例中，您首先创建一个基本的 heat 模板(/home/stack/templates/nameserver.yaml)，它将运行一个脚本，将每个节点的 resolv.conf 附加到变量名称服务器。

```
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，Resources 部分包含以下内容：**

**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 **nameserver_ip** 参数中存储的值。

**CustomExtraDeploymentPre**

这会执行软件配置，这是来自 **CustomExtraConfigPre** 资源的软件配置。注意以下几点：

- 配置参数引用 **CustomExtraConfigPre** 资源，因此 **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，是此 **hook** 模板中的强制要求。
-

actions 参数定义何时应用配置。在这种情况下，我们仅在创建或更新 Overcloud 时应用配置。可能的操作包括 CREATE、UPDATE、DELETE、SUSPEND 和 RESUME。

- input_values 参数包含名为 deploy_identifier 的子参数，用于存储父模板中的 DeployIdentifier。此参数为每个部署更新的资源提供时间戳。这可确保在后续的 overcloud 更新中资源恢复。

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

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

要应用配置，请在创建或更新 Overcloud 时将环境文件添加到堆栈中，以及其他环境文件。例如：

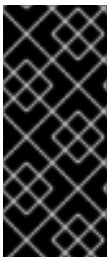
```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

这会在内核配置开始初始 Overcloud 创建或后续更新时，将配置应用到所有节点。

重要

您只能将 OS::TripleO::NodeExtraConfig 注册到一个 Heat 模板。后续使用会覆盖要使用的 Heat 模板。

### 4.4. POST-CONFIGURATION：自定义所有 OVERCLOUD 角色



重要

本文档的早期版本使用 OS::TripleO::Tasks::*PostConfig 资源来为每个角色提供安装后 hook。director 的 Heat 模板集合需要专用于使用这些 hook，这意味着您不应该将它们用于自定义用途。反之，使用下面概述的 OS::TripleO::NodeExtraConfigPost hook。

当您完成 Overcloud 的创建但希望在初始创建或后续更新时，可以为所有角色添加额外的配置，会出现这种情况。在这种情况下，您可以使用以下后配置 hook：

**OS::TripleO::NodeExtraConfigPost**

在核心 Puppet 配置后，应用到所有节点角色的额外配置。

在本例中，您首先创建一个基本的 heat 模板(/home/stack/templates/nameserver.yaml)，它将运行一个脚本，将每个节点的 resolv.conf 附加到变量名称服务器。

```
description: >
  Extra hostname configuration

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

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，Resources 部分包含以下内容：

**CustomExtraConfig**

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 nameserver_ip 参数中存储的值。

***CustomExtraDeployments***

*这会执行软件配置，这是来自 CustomExtraConfig 资源的软件配置。注意以下几点：*

- *配置参数引用 CustomExtraConfig 资源，以便 Heat 了解要应用的配置。*
- *servers 参数检索 Overcloud 节点的映射。此参数由父模板提供，是此 hook 模板中的强制要求。*
- *actions 参数定义何时应用配置。在这种情况下，我们仅在创建 Overcloud 时应用配置。可能的操作包括 CREATE、UPDATE、DELETE、SUSPEND 和 RESUME。*
- *input_values 包含一个名为 deploy_identifier 的参数，它存储来自父模板中的 DeployIdentifier。此参数为每个部署更新的资源提供时间戳。这可确保在后续的 overcloud 更新中资源恢复。*

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

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

*要应用配置，请在创建或更新 Overcloud 时将环境文件添加到堆栈中，以及其他环境文件。例如：*

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

*这会在初始 Overcloud 创建或后续更新上完成核心配置后，将配置应用到所有节点。*

重要

您只能将 OS::TripleO::NodeExtraConfigPost 注册到一个 Heat 模板。后续使用会覆盖要使用的 Heat 模板。

### 4.5. PUPPET：为角色自定义 HIERADATA

Heat 模板集合包含一组参数，用于将额外配置传递给某些节点类型。这些参数将配置保存为节点的 Puppet 配置的 hieradata。这些参数：

**ControllerExtraConfig**

添加至所有 Controller 节点的配置。

**ComputeExtraConfig**

添加至所有 Compute 节点的配置。

**BlockStorageExtraConfig**

添加至所有块存储节点的配置。

**ObjectStorageExtraConfig**

添加至所有 Object Storage 节点的配置

**CephStorageExtraConfig**

配置以添加到所有 Ceph Storage 节点

**[ROLE]ExtraConfig**

要添加到可组合角色的配置。用可组合角色名称替换 [ROLE]。

**ExtraConfig**

要添加到所有节点的配置。

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 主机保留的内存增加到 1024 MB，并将 VNC 键映射设置为日语：

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
```

```
    nova::compute::vnc_keymap: ja
```

在运行 **openstack overcloud deploy** 时包括此环境文件。

**重要**

您只能定义一个参数一次。后续使用会覆盖之前的值。

### 4.6. PUPPET：为单个节点自定义 HIERADATA

您可以使用 **Heat** 模板集合为各个节点设置 **Puppet hieradata**。要做到这一点，您需要获取保存为节点的内省数据一部分的系统 **UUID**：

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 | jq
.extra.system.product.uuid
```

这会输出一个系统 **UUID**。例如：

```
"F5055C6C-477F-47FB-AFE5-95C6928C407F"
```

在定义节点特定 **hieradata** 的环境中使用此系统 **UUID**，并将 **per_node.yaml** 模板注册到预配置 **hook** 中。例如：

```
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" ]}}'
```

在运行 **openstack overcloud deploy** 时包括此环境文件。

**per_node.yaml** 模板在与每个系统 **UUID** 对应的节点上生成一组 **heiradata** 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。在上例中，**per_node.yaml** 模板在所有 **Compute** 节点上运行（根据 **OS::TripleO::ComputeExtraConfigPre hook**），但只有有系统 **UUID F5055C6C-477F-47FB-AFE5-95C6928C407F** 接收 **hieradata** 的 **Compute** 节点。

这提供了一种根据特定要求定制每个节点的方法。

如需有关 NodeDataLookup 的更多信息，请参阅使用容器化 Red Hat Ceph 部署 Overcloud 指南中的配置 Ceph Storage 集群设置。

### 4.7. PUPPET：应用自定义清单

在某些情况下，您可能需要为 Overcloud 节点安装和配置一些附加组件。您可以通过自定义 Puppet 清单来实现此目的，此清单在主配置完成后应用到节点。作为基本示例，您可能想要将 motd 安装到每个节点。完成的过程是首先创建启动 Puppet 配置的 Heat 模板 (/home/stack/templates/custom_puppet_config.yaml)。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json

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

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

这包括模板中的 /home/stack/templates/motd.pp，并将其传递给节点以进行配置。motd.pp 文件本身包含用于安装和配置 motd 的 Puppet 类。

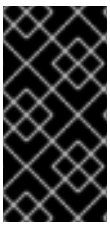
接下来，创建一个环境文件(/home/stack/templates/puppet_post_config.yaml)，将您的 heat 模板注册为 OS::TripleO::NodeExtraConfigPost: 资源类型。

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

最后，在创建或更新 Overcloud 堆栈时，包含此环境文件以及其他环境文件：

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

这会将 **motd.ppp** 的配置应用到 **Overcloud** 中的所有节点。



**重要**

不要定义同一自定义 **hieradata** 哈希的多个实例。同一自定义 **hieradata** 的多个实例可能会导致 **Puppet** 运行期间出现冲突，并导致为配置选项设置了意外值。

# 第 5 章 OVERCLOUD 注册

Overcloud 提供了在红帽 Content Delivery Network、Red Hat Satellite Server 5 或 Red Hat Satellite Server 6 中注册节点的方法。

## 5.1. 使用环境文件注册 OVERCLOUD

从 Heat 模板集合中复制注册文件：

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

编辑 ~/templates/rhel-registration/environment-rhel-registration.yaml 文件，并更改适用于您的注册方法和详情的参数值。

常规参数

**rhel_reg_method**

选择注册方法。门户、卫星 或禁用。

**rhel_reg_type**

要注册的单元类型。留空来注册系统

**rhel_reg_auto_attach**

自动为此系统附加兼容订阅。设置为 true 来启用。要禁用此功能，请将此参数留空。

**rhel_reg_service_level**

用于自动附加的服务级别。

**rhel_reg_release**

使用这个参数为自动附加设置发行版本。留空，以使用 Red Hat Subscription Manager 中的默认值。

**rhel_reg_pool_id**

要使用的订阅池 ID。如果没有自动附加订阅，请使用此项。要找到此 ID，请运行 sudo subscription-manager list --available --all --matches="*OpenStack*"，并使用生成的 池 ID 值。

**rhel_reg_sat_url**

要注册 Overcloud 节点的 Satellite 服务器的基本 URL。此参数使用卫星服务器 HTTP URL 而不是 HTTPS URL。例如，使用 http://satellite.example.com 而不使用 https://satellite.example.com。Overcloud 创建过程使用此 URL 来决定您正在使用 Red Hat Satellite Server 5 还是 Red Hat Satellite Server 6。在使用 Red Hat Satellite Server 6 时，Overcloud 获取 katello-ca-consumer-latest.noarch.rpm 文件，使用 subscription-manager 注册，并安装 katello-agent。在使用 Red Hat Satellite Server 5 时，Overcloud 获取 RHN-ORG-TRUSTED-SSL-CERT 文件并使用 rhnreg_ks 注册。

rhel_reg_server_url

要使用的订阅服务的主机名。默认值是 客户门户网站 Subscription Management, subscription.rhn.redhat.com。如果没有使用这个选项，则会在客户门户网站订阅管理中注册该系统。订阅服务器 URL 使用 https://hostname:port/prefix 的形式。

rhel_reg_base_url

您要用来接收更新的内容交付服务器的主机名。默认值为 https://cdn.redhat.com。由于卫星 6 托管自己的内容，因此该 URL 必须用于使用卫星 6 注册的系统。内容的基本 URL 使用 https://hostname:port/prefix 的形式。

rhel_reg_org

要用于注册的组织。要找到此 ID，请从 undercloud 节点运行 sudo subscription-manager orgs。在提示时输入您的红帽凭证，并使用生成的 密钥值。

rhel_reg_environment

要在所选机构中使用的环境。

rhel_reg_repos

要启用的、以逗号分隔的软件仓库列表。

rhel_reg_activation_key

要用于注册的激活码。当使用激活码进行注册时，还必须指定您要用于注册的机构。

rhel_reg_user; rhel_reg_password

用于注册的用户名和密码。如果可能，请使用激活码注册。

rhel_reg_machine_name

用于注册的机器名称。如果要使用节点的主机名，请留空。

rhel_reg_force

设置为 true 以强制注册选项，例如在重新注册节点时。

**rhel_reg_sat_repo**

包含 Red Hat Satellite 6 服务器管理工具的存储库，如 katello-agent。确保存储库名称对应于您的卫星服务器版本，并且存储库在卫星服务器上同步。例如，rhel-7-server-satellite-tools-6.2-rpms 对应于 Red Hat Satellite 6.2。

升级参数

**UpdateOnRHELRegistration**

如果设置为 True，在注册完成后触发 overcloud 软件包的更新。默认设置为 False。

HTTP 代理参数

**rhel_reg_http_proxy_host**

HTTP 代理的主机名。例如： proxy.example.com。

**rhel_reg_http_proxy_port**

HTTP 代理通信的端口。例如： 8080。

**rhel_reg_http_proxy_username**

用于访问 HTTP 代理的用户名。

**rhel_reg_http_proxy_password**

用于访问 HTTP 代理的密码。

重要

如果使用代理服务器，请确保所有 overcloud 节点都具有指向 rhel_reg_http_proxy_host 参数中定义的主机的路由。如果没有路由到此主机，subscription-manager 将超时并导致部署失败。

部署命令(openstack overcloud deploy)使用 -e 选项添加环境文件。添加 ~/templates/rhel-registration/environment-rhel-registration.yaml 和 ~/templates/rhel-registration/rhel-registration-resource-registry.yaml。例如：

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

重要

注册设置为 OS::TripleO::NodeExtraConfig Heat 资源。这意味着您只能使用此资源进行注册。如需更多信息，请参阅 第 4.2 节 “pre-Configuration：自定义特定 Overcloud 角色”。

### 5.2. 示例 1：注册到客户门户网站

以下使用 my-openstack 激活密钥将 overcloud 节点注册到红帽客户门户，并订阅池 1a85f9223e3d5e43013e3d6e8ff506fd。

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1234567"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_repos: "rhel-7-server-rpms,rhel-7-server-extras-rpms,rhel-7-server-rh-common-rpms,rhel-ha-for-rhel-7-server-rpms,rhel-7-server-openstack-13-rpms,rhel-7-server-rhceph-3-osd-rpms,rhel-7-server-rhceph-3-mon-rpms,rhel-7-server-rhceph-3-tools-rpms"
  rhel_reg_method: "portal"
  rhel_reg_sat_repo: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_sat_url: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

### 5.3. 示例 2：注册到 RED HAT SATELLITE 6 服务器

以下将 overcloud 节点注册到位于 sat6.example.com 的 Red Hat Satellite 6 服务器，并使用 my-openstack 激活密钥订阅池 1a85f9223e3d5e43013e3d6e8ff506fd。在这种情况下，激活密钥也提供要启用的存储库。

```
parameter_defaults:
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_method: "satellite"
```

```
  rhel_reg_sat_url: "http://sat6.example.com"
  rhel_reg_sat_repo: "rhel-7-server-satellite-tools-6.2-rpms"
  rhel_reg_repos: ""
  rhel_reg_auto_attach: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

### 5.4. 示例 3：注册到 RED HAT SATELLITE 5 SERVER

以下将 overcloud 节点注册到位于 sat5.example.com 的 Red Hat Satellite 5 服务器，使用 my-openstack 激活密钥，并自动附加订阅。在这种情况下，激活密钥也提供要启用的存储库。

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_method: "satellite"
  rhel_reg_sat_url: "http://sat5.example.com"
  rhel_reg_repos: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_pool_id: ""
  rhel_reg_release: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_sat_repo: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

### 5.5. 示例 4：通过 HTTP 代理注册

以下示例参数为您的所需注册方法设置 HTTP 代理设置：

```
parameter_defaults:
  ...
  rhel_reg_http_proxy_host: "proxy.example.com"
  rhel_reg_http_proxy_port: "8080"
  rhel_reg_http_proxy_username: "proxyuser"
  rhel_reg_http_proxy_password: "p@55w0rd!"
  ...
```

### 5.6. 高级注册方法

在某些情况下，您可能想要将不同的角色注册到不同的订阅类型。例如，您可能只将 Controller 节点订阅到 OpenStack Platform 订阅，并将 Ceph Storage 节点订阅到 Ceph Storage 订阅。本节提供了一些高级注册方法，以帮助为不同的角色分配单独的订阅。

#### 配置 Hook

种方法是编写特定于角色的脚本，并通过特定于角色的 hook 来包括它们。例如，可以将以下代码片段添加到 OS::TripleO::ControllerExtraConfigPre 资源的模板中，这样可以确保仅 Controller 节点接收这些订阅详情。

```
ControllerRegistrationConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config: |
      #!/bin/sh
      sudo subscription-manager register --org 1234567 \
        --activationkey "my-openstack"
      sudo subscription-manager attach --pool 1a85f9223e3d5e43013e3d6e8ff506fd
      sudo subscription-manager repos --enable rhel-7-server-rpms \
        --enable rhel-7-server-extras-rpms \
        --enable rhel-7-server-rh-common-rpms \
        --enable rhel-ha-for-rhel-7-server-rpms \
        --enable rhel-7-server-openstack-13-rpms \
        --enable rhel-7-server-rhceph-3-mon-rpms

ControllerRegistrationDeployment:
  type: OS::Heat::SoftwareDeployment
  properties:
    server: {get_param: server}
    config: {get_resource: ControllerRegistrationConfig}
    actions: ['CREATE','UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

该脚本使用一组 subscription-manager 命令来注册系统，附加订阅并启用所需的存储库。

有关 hook 的更多信息，请参阅 第 4 章 配置 Hook。

基于 Ansible 的配置

您可以使用 director 的动态清单脚本在特定角色上执行基于 Ansible 的注册。例如，您可以使用以下 play 注册 Controller 节点：

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-mon-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        activationkey: my-openstack
        org_id: 1234567
        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： - 使用激活密钥注册节点 - 禁用任何启用自动启用的存储库 - 仅启用与 Controller 节点相关的存储库。存储库使用 repos 变量列出。

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

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

该命令执行以下操作： - 运行动态清单脚本来获取主机及其组列表 - 将 playbook 任务应用到 playbook 的 hosts 参数中定义的组内的节点，本例中为 Controller 组。

*有关 overcloud 上运行 Ansible 自动化的更多信息，请参阅 Director 安装和使用 指南中的 "运行 Ansible Automation "。*

重要

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

作为 第 5 章 overcloud 注册 的 rhel 注册方法 的替代选择，director 可以使用基于 Ansible 的方法将 overcloud 节点注册到红帽客户门户或 Red Hat Satellite 6 服务器。此方法依赖于 overcloud 中启用基于 Ansible 的配置(config-download)。

## 6.1. RED HAT SUBSCRIPTION MANAGER (RHSM)可组合服务

rhsm 可组合服务提供了一种通过 Ansible 注册 overcloud 节点的方法。默认 roles_data 文件中的每个角色都包含一个 OS::TripleO::Services::Rhsm 资源，它默认为禁用。要启用该服务，您可以将资源注册到 rhsm 可组合服务文件中。例如：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
```

rhsm 可组合服务接受一个 RhsmVars 参数，它允许您定义与注册相关的多个子参数。例如：

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    rhsm_org_id: "1234567"
```

您还可以将 RhsmVars 参数与特定于角色的参数（如 ControllerParameters）结合使用，在为不同的节点类型启用特定的软件仓库时提供灵活性。

下一节是可用于 RhsmVars 参数的子参数列表，可用于 rhsm 可组合服务。

### *6.2. RHSMVARS SUB-PARAMETERS*

| rhsm | Description |
|---|---|
| **rhsm_method** | 选择注册方法。**门户、卫星 或禁用**。 |
| **rhsm_org_id** | 用于注册的组织。要找到此 ID，请从 undercloud 节点运行 **sudo subscription-manager orgs**。在提示时输入您的红帽凭证，并使用生成的 **密钥值**。 |
| **rhsm_pool_ids** | 要使用的订阅池 ID。如果没有自动附加订阅，请使用此项。要找到此 ID，请运行 **sudo subscription-manager list --available --all --matches="*OpenStack*"**，并使用生成的 **池 ID** 值。 |
| **rhsm_activation_key** | 用于注册的激活码。 |
| **rhsm_autosubscribe** | 自动为此系统附加兼容订阅。设置为 **true** 来启用。 |
| **rhsm_satellite_url** | 注册 Overcloud 节点的 Satellite 服务器的基本 URL。 |
| **rhsm_repos** | 要启用的软件仓库列表。 |
| **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_rhsm_proxy_hostname** | HTTP 代理的主机名。例如： **proxy.example.com**。 |
| **rhsm_rhsm_proxy_port** | HTTP 代理通信的端口。例如： **8080**。 |
| **rhsm_rhsm_proxy_user** | 用于访问 HTTP 代理的用户名。 |
| **rhsm_rhsm_proxy_password** | 用于访问 HTTP 代理的密码。 |

*现在，您已经了解了 rhsm 可组合服务的工作方式以及如何进行配置，您可以使用以下步骤配置您自己的注册详情。*

### *6.3. 使用 RHSM 可组合服务注册 OVERCLOUD*

*使用以下步骤创建可启用和配置 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. 创建环境文件(templates/rhsm.yml)以存储配置。

2. 在环境文件中包含您的配置。例如：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
```

**resource_registry 将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源关联，每个角色都可用。**

**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

**您还可以为特定 overcloud 角色提供注册详情。下一节提供了此示例。**

### 6.4. 将 RHSM 可组合服务应用到不同的角色

**您可以以每个角色为基础应用 rhsm 可组合服务。例如，您可以将一组配置应用到 Controller 节点，并将不同的配置集合应用到 Compute 节点。**

**流程**

1. *创建环境文件(templates/rhsm.yml)以存储配置。*

2. *在环境文件中包含您的配置。例如：*

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-7-server-rpms
        - rhel-7-server-extras-rpms
        - rhel-7-server-rh-common-rpms
        - rhel-ha-for-rhel-7-server-rpms
        - rhel-7-server-openstack-13-rpms
        - rhel-7-server-rhceph-3-osd-rpms
        - rhel-7-server-rhceph-3-mon-rpms
        - rhel-7-server-rhceph-3-tools-rpms
      rhsm_activation_key: "my-openstack"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
      rhsm_method: "portal"
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-7-server-rpms
        - rhel-7-server-extras-rpms
        - rhel-7-server-rh-common-rpms
        - rhel-ha-for-rhel-7-server-rpms
        - rhel-7-server-openstack-13-rpms
        - rhel-7-server-rhceph-3-tools-rpms
      rhsm_activation_key: "my-openstack"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
      rhsm_method: "portal"
```

*resource_registry 将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源关联，每个角色都可用。*

*ControllerParameters 和 computeParameters 使用自己的 RhsmVars 参数将订阅详情传递给其对应的角色。*

3. *保存环境文件*

### 6.5. 将 OVERCLOUD 注册到红帽卫星服务器

创建一个环境文件，它将启用和配置 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，以存储配置。

2. 在环境文件中包含您的配置。例如：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.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: 7.9
```

resource_registry 将 rhsm 可组合服务与 OS::TripleO::Services::Rhsm 资源关联，每个角色都可用。

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。

这些过程在 overcloud 中启用并配置 rhsm。但是，如果您使用 第 5 章 overcloud 注册 中的 rhel-registration 方法，则必须禁用它切换到基于 Ansible 的方法。使用以下步骤从 rhel-registration 方法切换到基于 Ansible 的方法。

### 6.6. 切换到 RHSM 可组合服务

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

此流程演示了如何从 rhel-registration 方法切换到 rhsm 可组合服务。

流程

1. 在以后的部署操作中排除 rhel-registration 环境文件。在大多数情况下，这是以下文件：
   - rhel-registration/environment-rhel-registration.yaml
   - rhel-registration/rhel-registration-resource-registry.yaml
2. 为将来的部署操作添加 rhsm 可组合服务参数的环境文件。

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

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

改为：

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.yaml
```

为了帮助将您的详细信息从 rhel-registration 方法转换为 rhsm 方法，请使用下表来映射您的参数及其值。

### 6.7. RHEL-REGRATION 到 RHSM 映射

| rhel-registration | rhsm / RhsmVars |
|---|---|
| rhel_reg_method | rhsm_method |
| rhel_reg_org | rhsm_org_id |
| rhel_reg_pool_id | rhsm_pool_ids |

| rhel-registration | rhsm / RhsmVars |
|---|---|
| rhel_reg_activation_key | rhsm_activation_key |
| rhel_reg_auto_attach | rhsm_autosubscribe |
| rhel_reg_sat_url | rhsm_satellite_url |
| rhel_reg_repos | rhsm_repos |
| rhel_reg_user | rhsm_username |
| rhel_reg_password | rhsm_password |
| rhel_reg_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 |

*现在，您已为 rhsm 服务配置了环境文件，您可以将它包含在下一个 overcloud 部署操作中。*

### *6.8. 使用 RHSM 可组合服务部署 OVERCLOUD*

*此过程演示了如何将 rhsm 配置应用到 overcloud。*

**流程**

1. *在运行 openstack overcloud deploy 命令时，包括 config-download 选项和环境文件以及 rhsm.yml 环境文件：*

```
openstack overcloud deploy \
  <other cli args> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/config-download-environment.yaml \
  --config-download \
  -e ~/templates/rhsm.yaml
```

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

2. 等待 overcloud 部署完成。

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

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

# 第 7 章 可组合服务和自定义角色

Overcloud 通常由预定义角色的节点组成，如 Controller 节点、计算节点和不同的存储节点类型。这些默认角色各自包含 director 节点上的 Heat 模板集合中定义的一组服务。但是，核心 Heat 模板的架构提供了以下功能的方法：

- 创建自定义角色
- 从每个角色添加或删除服务

这样，可以在不同的角色上创建不同的服务组合。本章将探讨自定义角色的架构、可组合服务以及使用它们的方法。

## 7.1. 支持的角色架构

在使用自定义角色和可组合服务时，可以使用以下架构：

**架构 1 - 默认架构**

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

**架构 2 - 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。请参阅 第 7.2.3 节 “支持的自定义角色”。

**架构 3 - 自定义可组合服务**

创建自己的 角色 并使用它们来生成自定义 roles_data 文件。请注意，只有有限的可组合服务组合已被测试并验证，红帽无法支持所有可组合的服务组合。

## 7.2. 角色

### 7.2.1. 检查 roles_data 文件

Overcloud 创建流程利用 roles_data 文件定义其角色。roles_data 文件包含角色的 YAML 格式列表。以下是 roles_data 语法的缩写示例：

```
- name: Controller
```

```
  description: |
    Controller role that has all the controler 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 模板集合包含位于 /usr/share/openstack-tripleo-heat-templates/ roles_data.yaml 的默认 roles_data 文件。默认文件定义了以下角色类型：**

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

**openstack overcloud deploy 命令在部署期间包含此文件。您可以使用 -r 参数通过自定义 roles_data 文件覆盖此文件。例如：**

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

### 7.2.2. 创建 roles_data 文件

**虽然您可以手动创建自定义 roles_data 文件，但也可以使用单个角色模板自动生成文件。director 提供了几个命令来管理角色模板，并自动生成自定义 roles_data 文件。**

要列出默认角色模板，请使用 **openstack overcloud role list** 命令：

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

要查看角色的 **YAML** 定义，请使用 **openstack overcloud role show** 命令：

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

要生成自定义 **roles_data** 文件，请使用 **openstack overcloud roles generate** 命令将多个预定义角色加入到一个文件中。例如，以下命令将控制器、**Compute** 和 **Networker** 角色加入到一个文件中：

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

**-o** 定义要创建的文件的名称。

这会创建自定义 **roles_data** 文件。但是，上例使用 **Controller** 和 **Networker** 角色，这些角色都包含相同的网络代理。这意味着网络服务从控制器扩展到 **Networker** 角色。**overcloud** 在 **Controller** 和 **Networker** 节点之间平衡网络服务的负载。

要使这个 **Networker** 角色独立，您可以创建自己的自定义 **Controller** 角色，以及所需的任何其他角色。这样，您可以从自己的自定义角色轻松生成 **roles_data** 文件。

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 ~/.
```

在此目录中添加或修改自定义角色文件。将 **--roles-path** 选项用于上述任何角色子命令中的 **--roles-path** 选项，以将此目录用作自定义角色的源。例如：

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

*这会从 ~/roles 目录中的各个角色生成单个 my_roles_data.yaml 文件。*

**注意**

*默认角色集合还包含 ControllerOpenStack 角色，该角色不包括 网络者、消息传递和数据库 角色的服务。您可以将 ControllerOpenStack 与独立 网络器、消息传递和数据库 角色结合使用。*

### 7.2.3. 支持的自定义角色

*下表描述了 /usr/share/openstack-tripleo-heat-templates/roles 中提供的所有受支持角色。*

| 角色 | Description | 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** | 单机横向扩展 Ceph 存储文件角色.包括 OSD 和对象操作(MDS)。 | **CephFile.yaml** |
| **CephObject** | 独立横向扩展 Ceph 存储对象角色。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 Compute 节点角色。 | **ComputeDVR.yaml** |
| **ComputeHCI** | 带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** | 计算实例 HA 节点角色.搭配 **environments/compute-instanceha.yaml** 的环境文件 使用。 | **ComputeInstanceHA.yaml** |

| 角色 | Description | File |
| --- | --- | --- |
| **ComputeLiquidio** | 带有 Cavium Liquidio 智能 NIC 的计算节点. | **ComputeLiquidio.yaml** |
| **ComputeOvsDpdkRT** | 计算 OVS DPDK RealTime 角色。 | **ComputeOvsDpdkRT.yaml** |
| **ComputeOvsDpdk** | 计算 OVS DPDK 角色。 | **ComputeOvsDpdk.yaml** |
| **ComputePPC64LE** | ppc64le 服务器的计算角色。 | **ComputePPC64LE.yaml** |
| **ComputeRealTime** | 针对实时行为优化的计算角色。使用此角色时，强制使用 **overcloud-realtime-compute** 镜像，以及特定角色参数 **IsolCpusList** 和 **NovaVcpuPinSet** 相应地设置为实时计算节点的硬件。 | **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** | 没有包含数据库、消息传递、网络和 OpenStack Compute (nova)控制组件的控制器角色。使用，与数据库、**Messaging**、**Networker** 和 **Novacontrol** 角色结合使用。 | **ControllerAllNovaStandalone.yaml** |
| **ControllerNoCeph** | 加载核心 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** | 没有包含 OpenStack Compute (nova)控制组件的控制器角色。使用 和 **Novacontrol** 角色。 | **ControllerNovaStandalone.yaml** |
| **ControllerOpenstack** | 没有包含数据库、消息传递和网络组件的控制器角色。使用，与数据库、**Messaging** 和 **Networker** 角色结合使用。 | **ControllerOpenstack.yaml** |

| 角色 | Description | File |
| --- | --- | --- |
| **ControllerStorageNfs** | Controller 角色加载了所有核心服务并使用 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** |
| **数据库** | 独立数据库角色.使用 Pacemaker 作为 Galera 集群管理的数据库。 | **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** |
| **HciCephFile** | 具有超融合基础架构和 Ceph Storage 文件服务的计算节点。包括 OSD 和对象操作(MDS)。 | **HciCephFile.yaml** |
| **HciCephMon** | 具有超融合基础架构和 Ceph Storage 块存储服务的计算节点。包括 OSD、MON 和 Manager。 | **HciCephMon.yaml** |
| **HciCephObject** | 具有超融合基础架构和 Ceph Storage 对象存储服务的计算节点。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。如果您的部署使用了 ML2/OVN 机制驱动程序，请参阅使用 ML2/OVN 部署自定义角色 中的附加步骤。 | **Networker.yaml** |
| **Novacontrol** | 独立 **nova-control** 角色，用来自行运行 OpenStack Compute (nova)控制代理。 | **novacontrol.yaml** |
| **ObjectStorage** | Swift 对象存储节点角色. | **ObjectStorage.yaml** |
| **Telemetry** | 具有所有指标和警报服务的 Telemetry 角色。 | **Telemetry.yaml** |

### *7.2.4. 检查角色参数*

*每个角色使用以下参数：*

***name***

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

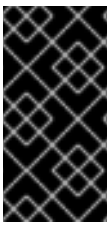
***description***

*（可选） 角色纯文本描述。*

***tags***

*（可选） o 定义角色属性的标签的 YAML 列表。使用此参数将 控制器和 主标签一起定义 主要角色 ：*

```
- name: Controller
  ...
  tags:
    - primary
    - controller
  ...
```



*重要*

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

*网络*

*要在角色上配置的网络 YAML 列表：*

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

*默认网络包括 外部、InternalApi、存储、StorageMgmt、Tenant，以及管理。*

***CountDefault***

*（可选） 定义要为此角色部署的默认节点数量。*

***HostnameFormatDefault***

（可选） 定义角色的默认主机名格式。默认命名规则使用以下格式：

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

例如，默认的 Controller 节点被命名：

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

**disable_constraints**

（可选） 定义在使用 director 部署时是否禁用 OpenStack Compute (nova)和 OpenStack Image Storage (glance)约束。使用预置备节点部署 overcloud 时使用。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 "。

**disable_upgrade_deployment**

（可选） 定义是否为特定角色禁用升级。这提供了一种升级角色中各个节点的方法，并确保服务的可用性。例如， Compute 和 Swift Storage 角色使用此参数。

**update_serial**

（可选） 定义在 OpenStack 更新选项中同时要更新的节点数量。在默认的 roles_data.yaml 文件中：

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

如果从自定义角色省略此参数，则默认为 1。

**ServicesDefault**

（可选） 定义要在节点上包括的服务的默认列表。如需更多信息，请参阅 第 7.3.2 节 “检查可组合的服务架构”。

这些参数提供了创建新角色以及定义要包含哪些服务的方法。

*openstack overcloud deploy 命令将参数从 roles_data 文件集成到一些基于 Jinja2 的模板。例如，在某些点上，overcloud.j2.yaml Heat 模板会迭代 roles_data.yaml 中的角色列表，并创建特定于各个角色的参数和资源。*

*overcloud.j2.yaml Heat 模板中每个角色的资源定义显示为以下代码片段：*

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

*此片段显示基于 Jinja2 的模板如何融合 {{role.name}} 变量，将每个角色的名称定义为 OS::Heat::ResourceGroup 资源。反过来，使用 roles_data 文件中的每个 name 参数来命名每个对应的 OS::Heat::ResourceGroup 资源。*

### *7.2.5. 创建新角色*

*在本例中，旨在创建一个新的 Horizon 角色，以仅托管 OpenStack 仪表板(horizon)。在这种情况下，您可以创建一个自定义 角色 目录，其中包含新角色信息。*

*创建 默认角色 目录的自定义副本：*

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

*创建一个名为 ~/roles/Horizon.yaml 的新文件，再创建一个名为 base 和 core OpenStack Dashboard 服务的新 Horizon 角色。例如：*

```
- 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::Fluentd
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

**最好将 CountDefault 设置为 1，以便默认的 Overcloud 始终包含 Horizon 节点。**

**如果在现有 overcloud 中扩展服务，请将现有服务保留在 Controller 角色上。如果创建新 overcloud，并且希望 OpenStack 控制面板保留在独立角色中，请从 Controller 角色定义中删除 OpenStack 控制面板组件：**

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

**使用角色目录作为源生成新的 roles_data 文件：**

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

**您可能需要为此角色定义一个新类别，以便可以标记特定的节点。在本例中，使用以下命令来创建 horizon 类别：**

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

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property "capabilities:boot_option"="local" --
property "capabilities:profile"="horizon" horizon
$ openstack flavor set --property resources:VCPU=0 --property resources:MEMORY_MB=0 --
property resources:DISK_GB=0 --property resources:CUSTOM_BAREMETAL=1 horizon
```

使用以下命令将节点标记到新类型中：

```
$ openstack baremetal node set --property capabilities='profile:horizon,boot_option:local' 58c3d07e-
24f2-48a7-bbb6-6843f0e8ee13
```

使用以下环境文件片段定义 **Horizon** 节点数和类别：

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

在运行 **openstack overcloud deploy** 命令时，包含新的 **roles_data** 文件和环境文件。例如：

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

部署完成后，这会创建一个由一个 **Controller** 节点、一个 **Compute** 节点和一个 **Networker** 节点组成的三节点 **Overcloud**。要查看 **Overcloud** 的节点列表，请运行以下命令：

```
$ openstack server list
```

### 7.3. 可组合的服务

#### 7.3.1. 指南和限制

请注意，以下可组合节点架构的指南和限制。

对于不由 **Pacemaker** 管理的服务：

- 您可以将服务分配给独立自定义角色。
- 您可以在初始部署后创建额外的自定义角色，并将它们部署以扩展现有服务。

对于由 Pacemaker 管理的服务：

- 您可以将 Pacemaker 管理的服务分配给独立的自定义角色。
- Pacemaker 有 16 个节点的限制。如果将 Pacemaker 服务 (OS::TripleO::Services::Pacemaker)分配给 16 个节点，则后续任何节点都必须使用 Pacemaker 远程服务(OS::TripleO::Services::PacemakerRemote)。您不能在同一角色中使用 Pacemaker 服务和 Pacemaker 远程服务。
- 在不含 Pacemaker 管理的服务的角色上，不要将 Pacemaker 服务 (OS::TripleO::Services::Pacemaker)包含。
- 您无法扩展或缩减包含 OS::TripleO::Services::Pacemaker 或 OS::TripleO::Services::PacemakerRemote 服务的自定义角色。

常规限制：

- 您不能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离开节点上的孤立服务。

### 7.3.2. 检查可组合的服务架构

核心 heat 模板集合包含两组可组合服务模板：

- Puppet/服务 包含用于配置可组合服务的基础模板。
- Docker/服务 包含关键 OpenStack 平台服务的容器化模板。这些模板作为对某些基础模板的增强并引用回基础模板。

每个模板都包含一个标识其用途的描述。例如，ntp.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 RHOSP 部署的资源。这意味着，您可以使用 overcloud-resource-registry-puppet.j2.yaml 文件中定义的唯一 heat 资源命名空间调用每个资源。所有服务都将 OS::TripleO::Services 命名空间用作其资源类型。

有些资源直接使用基本可组合服务模板：

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: puppet/services/time/ntp.yaml
  ...
```

但是核心服务需要容器，如使用容器化服务模板。例如，keystone 容器化服务使用以下方法：

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: docker/services/keystone.yaml
  ...
```

这些容器化模板通常引用到基础模板，以包括 Puppet 配置。例如，docker/services/keystone.yaml 模板将基本模板的输出存储在 KeystoneBase 参数中：

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
```

然后，容器化模板可以包含基础模板中的功能和数据。

overcloud.j2.yaml heat 模板包含 Jinja2 的代码部分，用于在 roles_data.yaml 文件中为每个自定义角色定义一个服务列表：

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

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

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

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

这些服务随后被定义为 **ControllerServices** 参数的默认列表。

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

### 7.3.3. 从角色添加和删除服务

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，您可能会从 **Controller** 节点中删除 **OpenStack Orchestration (heat)**。在这种情况下，创建 **默认角色** 目录的自定义副本：

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

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

```
    - 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
```

**生成新的 roles_data 文件。例如：**

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

**在运行 openstack overcloud deploy 命令时包含这个新的 roles_data 文件。例如：**

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

**这会部署一个没有安装在 Controller 节点上的 OpenStack Orchestration 服务的 Overcloud。**

**注意**

**您还可以使用自定义环境文件在 roles_data 文件中禁用服务。重定向服务，以禁用 OS::Heat::None 资源。例如：**

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

### 7.3.4. 启用 Disabled Services

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

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

**若要启用此服务，可包含一个环境文件，用于将资源链接到 puppet/services 目录中对应的 Heat 模板。有些服务在 环境 目录中具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，该文件包含以下内容：**

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

这会覆盖默认的 null 操作资源并启用该服务。在运行 openstack overcloud deploy 命令时包括此环境文件。

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

**提示**

有关如何启用禁用的服务的另一个示例，请参阅 OpenStack 数据处理 指南中的 安装。本节介绍如何在 overcloud 上启用 OpenStack 数据处理服务(sahara)。

### 7.3.5. 使用 No Services 创建通用节点

Red Hat OpenStack Platform 提供了在不配置任何 OpenStack 服务的情况下创建通用 Red Hat Enterprise Linux 7 节点的功能。当您需要托管 Red Hat OpenStack Platform 核心环境外的软件时，这非常有用。例如，OpenStack 平台提供与 Kibana 和 Sensu 等监控工具的集成（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 7 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 7 镜像。这意味着该节点已安装一些 Red Hat OpenStack Platform 软件，但没有启用或配置。

创建通用节点需要没有 ServicesDefault 列表的新角色：

```
- name: Generic
```

在您的自定义 roles_data 文件中包括角色(roles_data_with_generic.yaml)。确保保留现有的 Controller 和 Compute 角色。

您还可以包括一个环境文件(generic-node-params.yaml)来指定所需通用 Red Hat Enterprise Linux 7 节点的数量以及要置备的节点时的类别。例如：

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

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

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

*这会部署一个具有一个 **Controller** 节点、一个 **Compute** 节点和一个通用 **Red Hat Enterprise Linux 7** 节点的三节点环境。*

# 第 8 章 容器化服务

director 将核心 OpenStack Platform 服务安装为 overcloud 上的容器。本节提供了一些有关容器化服务如何工作的背景信息。

## 8.1. 容器化服务架构

director 将核心 OpenStack Platform 服务安装为 overcloud 上的容器。容器化服务的模板位于 /usr/share/openstack-tripleo-heat-templates/docker/services/。这些模板引用对应的可组合服务模板。例如，OpenStack Identity (keystone)容器化服务模板(docker/services/keystone.yaml)包括以下资源：

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
  properties:
    EndpointMap: {get_param: EndpointMap}
    ServiceData: {get_param: ServiceData}
    ServiceNetMap: {get_param: ServiceNetMap}
    DefaultPasswords: {get_param: DefaultPasswords}
    RoleName: {get_param: RoleName}
    RoleParameters: {get_param: RoleParameters}
```

这个类型 指的是对应的 OpenStack Identity (keystone)可组合服务，并从该模板中提取 输出 数据。容器化服务将这些数据与其自己的特定容器数据合并。

使用容器化服务的所有节点都必须启用 OS::TripleO::Services::Docker 服务。为自定义角色配置创建 roles_data.yaml 文件时，请将带有基本可组合服务的 OS::TripleO::Services::Docker 服务作为容器化服务包括在内。例如，Keystone 角色使用以下角色定义：

```
- name: Keystone
  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::Fluentd
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Keystone
```

### 8.2. 容器化服务参数

每个容器化服务模板包含一个 outputs 部分，它定义了传递给 director 的 OpenStack Orchestration (heat)服务的数据集合。除了标准可组合服务参数外，模板还包含一组特定于容器配置的参数。第 7.2.4 节 “检查角色参数”

**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，用于在实际的容器化服务运行之前配置该服务。此参数包括以下子参数：+

- config_volume - 存储配置的已挂载 docker 卷。
- puppet_tags - 在配置期间传递给 Puppet 的标签。这些标签在 OpenStack Platform 中使用，可以将 Puppet 运行限制为特定的服务配置资源。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 资源在配置容器上运行。
- 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**

组特定于容器的数据，用于定义配置文件位置、目录权限以及要在容器上运行的 命令来启动服务。

**docker_config**

在服务配置容器中运行的任务。所有任务都分组为步骤，以帮助 director 执行暂存的部署。这些步骤为：+

- 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 - 通用 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。

### 8.3. 修改 OPENSTACK PLATFORM 容器

红帽通过 Red Hat Container Catalog (registry.redhat.io)提供了一组预构建的容器镜像。可以修改这些镜像并向其中添加其他层。这对于向容器添加认证第三方驱动程序的 RPM 非常有用。

**注意**

为确保继续支持修改后的 OpenStack Platform 容器镜像，请确保生成的镜像符合 "Red Hat Container Support Policy"。

本例演示了如何自定义最新的 openstack-keystone 镜像。但是，这些说明也可以应用到其他镜像：

1. 拉取您要修改的镜像。例如，对于 openstack-keystone 镜像：

```
$ sudo docker pull registry.redhat.io/rhosp13/openstack-keystone:latest
```

2. 检查原始镜像上的默认用户。例如，对于 openstack-keystone 镜像：

```
$ sudo docker run -it registry.redhat.io/rhosp13/openstack-keystone:latest whoami
root
```

**注意**

openstack-keystone 镜像使用 root 作为默认用户。其他镜像使用特定用户。例如，openstack-glance-api 将 glance 用于默认用户。

3.

创建 Dockerfile 以在现有容器镜像上构建额外层。以下示例从 Container Catalog 拉取最新的 OpenStack Identity (keystone)镜像，并将自定义 RPM 文件安装到镜像中：

```
FROM registry.redhat.io/rhosp13/openstack-keystone
MAINTAINER Acme
LABEL name="rhosp13/openstack-keystone-acme" vendor="Acme" version="2.1"
release="1"

# switch to root and install a custom RPM, etc.
USER root
COPY custom.rpm /tmp
RUN rpm -ivh /tmp/custom.rpm

# switch the container back to the default user
USER root
```

4. 构建并标记新镜像。例如，使用保存在 /home/stack/keystone 目录中的本地 Dockerfile 进行构建，并将其标记为 undercloud 的本地 registry：

```
$ docker build /home/stack/keystone -t "192.168.24.1:8787/rhosp13/openstack-keystone-
acme:rev1"
```

5. 将生成的镜像推送到 undercloud 的本地 registry：

```
$ docker push 192.168.24.1:8787/rhosp13/openstack-keystone-acme:rev1
```

6. 编辑 overcloud 容器镜像环境文件（通常为 overcloud_images.yaml）并更改适当的参数，以使用自定义容器镜像。

警告

容器目录发布容器镜像，其中包含内置于其中的完整软件堆栈。当容器目录发布包含更新和安全修复的容器镜像时，您现有的自定义容器将 不包括 这些更新，且需要使用从 Catalog 中的新镜像版本重建。

### 8.4. 部署供应商插件

要将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以使用 Dell EMC 硬件作为块存储后端。

1. *登录到 **registry.connect.redhat.com** 目录：*

```
$ docker login registry.connect.redhat.com
```

2. *下载插件：*

```
$ docker pull registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

3. *使用与 **OpenStack** 部署相关的 **undercloud IP** 地址，标记并将镜像推送到本地 **undercloud registry**：*

```
$ docker tag registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13

$ docker push 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

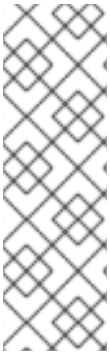
4. *使用包含以下参数的额外环境文件部署 **overcloud**：*

```
parameter_defaults:
  DockerCinderVolumeImage: 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-
rhosp13
```

# 第 9 章 基本网络隔离

本章介绍了如何使用标准网络隔离配置 overcloud。这包括：

- 用于启用网络隔离的渲染环境文件(/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml)。
- 一个复制的环境文件来配置网络默认值(/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml)。
- 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应您自己的网络。
- 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。
- 启用 NIC 的环境文件。本例使用位于 环境 目录中的默认 文件。
- 自定义您的网络参数的任何其他环境文件。



**注意**

运行 openstack overcloud netenv validate 命令，以验证您的 network-environment.yaml 文件的语法。此命令还会验证用于计算、控制器、存储和可组合角色网络文件的独立 nic-config 文件。使用 -f 或 --file 选项指定您要验证的文件：

```
$ openstack overcloud netenv validate -f ~/templates/network-environment.yaml
```

本章中的以下内容演示了如何定义各个方面：

## 9.1. 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。要使用隔离的网络，overcloud 包含可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为每个

网络定义所有端口和 VIP。在呈现时，它会使用完整资源 registry 在同一位置生成一个 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
```

此文件的第一部分具有 OS::TripleO::Network::* 资源的资源 registry 声明。默认情况下，这些资源使用 OS::Heat::None 资源类型，这不会创建任何网络。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每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，各自在网络子集中都有 IP。

overcloud 网络的其他功能，如 第 10 章 自定义可组合网络 和 第 11 章 自定义网络接口模板 依赖于此网络隔离环境文件。因此，您需要使用部署命令包含所生成的文件的名称。例如：

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

## 9.2. 修改隔离的网络配置

复制默认 network_data.yaml 文件，并修改副本来配置默认隔离网络。

流程

1. 复制默认 network_data 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，并修改参数以符合您的网络要求。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'}]
```

为每个网络编辑以下内容：

- VLAN 定义用于此网络的 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。大多用来定义外部网络的默认路由，但在需要时可用于其他网络。

使用 -n 选项，将自定义 network_data 文件与您的部署包含。如果没有 -n 选项，部署命令将使用默认网络详细信息。

## 9.3. 网络接口模板

overcloud 网络配置需要一组网络接口模板。这些模板是采用 YAML 格式的标准 heat 模板。每个角色都需要一个 NIC 模板，以便 director 能够正确配置该角色内的每个节点。

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**

要在模板中包括的网络参数。

资源

取参数中定义的 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 模板利用 Jinja2 语法来帮助呈现模板。例如，以下片段来自 single-nic-vlans 配置中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，这只为 Storage、Internal API 和 Tenant 网络呈现网络信息：

```
- 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 11 章 自定义网络接口模板 探索如何将默认 Jinja2 的模板呈现为标准 YAML 版本，您可以用作自定义基础。*

### 9.4. 默认网络接口模板

*director 包含 /usr/share/openstack-tripleo-heat-templates/network/config/ 中的模板，以适应最常见的网络场景。下表概述了每个 NIC 模板集以及您必须用来启用模板所需的相应环境文件。*

**注意**

*启用 NIC 模板的每个环境文件都使用后缀 .j2.yaml。这是未发送的 Jinja2 版本。确保在部署中只包含 .yaml 后缀的 rendered 文件名，该名称只使用 .yaml 后缀。*

| NIC 目录 | Description | 环境文件 |
|---|---|---|
| **single-nic-vlans** | 带有附加到默认 Open vSwitch 网桥的 control plane 和 VLAN 的单个 NIC (**nic1**)。 | **environments/net-single-nic-with-vlans.j2.yaml** |
| **single-nic-linux-bridge-vlans** | 带有附加到默认 Linux 网桥的 control plane 和 VLAN 的单个 NIC (**nic1**)。 | **environments/net-single-nic-linux-bridge-with-vlans** |
| **bond-with-vlans** | 附加至 **nic1** 的 control plane。默认带有绑定 NIC 配置的 Open vSwitch 网桥(**nic2** 和 **nic3)**和附加 VLAN。 | **environments/net-bond-with-vlans.yaml** |

| NIC 目录 | Description | 环境文件 |
| --- | --- | --- |
| **multiple-nics** | 附加至 **nic1** 的 control plane。为 **network_data** 文件中定义的每个网络分配每个后续 NIC。默认情况下，这是 **nic2** 的 Storage Management （从 nic3 到 **nic3**，内部 API 为 **nic4**，对 **br-tenant** 网桥上的租户到 **nic5** ），以及默认 Open vSwitch 网桥上的 **nic6** External to nic6。 | **environments/net-multiple-nics.yaml** |

***注意***

*使用没有外部网络的环境文件（如 **net-bond-with-vlans-no-external.yaml**）并使用 **IPv6** （如 **net-bond-with-vlans-v6.yaml**）。它们可用于向后兼容，且无法与可组合网络运行。*

*每个默认 **NIC** 模板集都包含 **role.role.j2.yaml** 模板。此文件使用 **Jinja2** 为每一个可组合角色呈现其他文件。例如，如果 **overcloud** 使用 **Compute**、**Controller** 和 **Ceph Storage** 角色，则部署会根据 **role.role.j2.yaml** 呈现新的模板，例如*

- ***compute.yaml***
- ***controller.yaml***
- ***ceph-storage.yaml。***

### *9.5. 启用基本网络隔离*

*此流程演示了如何使用其中一个默认 **NIC** 模板启用基本网络隔离。在本例中，这是带有 **VLAN** 模板的单个 **NIC (single-nic-vlans)**。*

***流程***

1. *在运行 **openstack overcloud deploy** 命令时，请确保包含以下内容的环境文件名称：*

- *自定义 network_data 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*
- *任何与您配置相关的额外环境文件。*

*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 10 章 自定义可组合网络

本章介绍了 第 9 章 基本网络隔离 中介绍的概念和步骤，并演示如何使用额外的可组合网络配置 overcloud。这包括：

- 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，可在默认值外创建额外网络。
- 个自定义 roles_data 文件，用于将自定义网络分配给角色。
- 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。
- 启用 NIC 的环境文件。本例使用位于 环境 目录中的默认 文件。
- 自定义您的网络参数的任何其他环境文件。这个示例使用环境文件来自定义 OpenStack 服务映射到可组合网络。

**注意**

运行 openstack overcloud netenv validate 命令，以验证您的 network-environment.yaml 文件的语法。此命令还会验证用于计算、控制器、存储和可组合角色网络文件的独立 nic-config 文件。使用 -f 或 --file 选项指定您要验证的文件：

```
$ openstack overcloud netenv validate -f ~/templates/network-environment.yaml
```

本章中的以下内容演示了如何定义各个方面：

## 10.1. 可组合网络

overcloud 默认使用以下预定义网络片段集合：

- Control Plane
- 内部 API
- 存储
- 存储管理
- 租户
- 外部
- 管理（可选）

可组合网络允许您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，可以将它提供给多个角色。

director 支持在部署和更新阶段创建自定义网络。这些额外网络可用于 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)管理要部署的网络列表。您可以使用 -n 选项在部署命令中包含此文件。若无此选项，部署将使用默认文件(/usr/share/openstack-tripleo-heat-templates/network_data.yaml)。

### 10.2. 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，您可以将网络呈现给多个角色。

过程

**流程**

1. 复制默认 **network_data** 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，并为您的新网络添加一个部分。例如：

```
- name: StorageBackup
  vip: true
  name_lower: storage_backup
  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'
```

- 设置网络的人类可读名称。这个参数是唯一强制参数。如果要为可读性设置名称，请使用 **name_lower** 参数，例如，如果要将 **InternalApi** 改为 **internal_api**。不要修改 **name** 参数。

- **VIP：true** 在新网络上创建虚拟 **IP** 地址(VIP)。此 **IP** 用作 **service-to-network** 映射参数中列出的服务的目标 **IP (ServiceNetMap)**。请注意，**VIP** 仅供使用 **Pacemaker** 的角色使用。**overcloud** 的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**、**allocation_pools** 和 **gateway_ip** 设置网络的默认 **IPv4** 子网、**IP** 范围和网关。

使用 **-n** 选项，将自定义 **network_data** 文件与您的部署包含。如果没有 **-n** 选项，部署命令将使用默认网络集合。

### 10.3. 在角色中包含可组合网络

您可以将可组合网络分配给环境中定义的 **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 **custon roles_data** 文件，请将默认复制到您的主目录：

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

2. 编辑自定义 **roles_data** 文件。

3. 滚动到您要添加可组合网络的角色，并将网络名称添加到 网络 列表中。例如，要将网络添加到 **Ceph Storage** 角色，请使用以下代码片段作为指南：

```
- 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。

在运行 **openstack overcloud deploy** 命令时，使用 **-r** 选项包括 **roles_data** 文件。如果没有 **-r** 选项，部署命令将使用默认角色集合以及对应的网络。

### 10.4. 为可组合网络分配 OPENSTACK 服务

每个 **OpenStack** 服务都会分配到资源注册表中的镜像默认网络类型。这些服务被绑定到在网络类型分配的网络中的 **IP** 地址。尽管 **OpenStack** 服务在这些网络之间划分，但实际物理网络的数量可能与网络环境文件中所定义的不同。您可以通过在环境文件中定义新的网络映射，将 **OpenStack** 服务重新分配给不同的网络类型，如 **/home/stack/templates/service-reassignments.yaml**。**ServiceNetMap** 参数决定要用于每个服务的网络类型。

例如，您可以通过修改突出显示的部分将 **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** 将这些服务放在 **Storage** 备份网络中，而不是存储管理网络。这意味着您只需要为 **Storage Backup** 网络定义一组 **parameter_defaults**，而不是存储管理网络。

**director** 将您的自定义 **ServiceNetMap** 参数定义合并到从 **ServiceNetMapDefaults** 获取的预定义默认值列表中，并覆盖默认值。然后，**director** 返回完整的列表，包括自定义到 **ServiceNetMap**，用于为

各种服务配置网络分配。

服务映射只适用于在 network_data 文件中对使用 Pacemaker 的节点使用 vip: true 的网络。overcloud 的负载平衡器将来自 VIP 的流量重定向到特定的服务端点。

注意

完整的默认服务列表可在 /usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml 中的 ServiceNetMapDefaults 参数中找到。

## 10.5. 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，使用 Single NIC with VLAN 模板 (net-single-nic-with-vlans)。

流程

1. 在运行 openstack overcloud deploy 命令时，请确保包括：

- 自定义 network_data 文件。
- 带有 network-to-role 分配的自定义 roles_data 文件。
- 默认网络隔离配置的呈现文件名。
- 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。
- 与网络相关的任何其他环境文件，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 \
  ...
```

这将在 overcloud 中的节点之间部署可组合网络，包括其他自定义网络。

重要

请记住，如果介绍新的自定义网络，如管理网络，您必须再次显示模板。只需将网络名称添加到 roles_data.yaml 文件即可。

### 10.6. 重命名默认网络

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

- InternalApi
- 外部
- 存储
- StorageMgmt
- 租户

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

**流程**

1. 在 **network_data.yaml** 文件中，为您要重命名的每个网络在 **name_lower** 参数中输入新名称：

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

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

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

# 第 11 章 自定义网络接口模板

本章阐述了 第 9 章 基本网络隔离 中介绍的概念和步骤。本章的目的是演示如何创建一组自定义网络接口模板以适合您的环境中的节点。这包括：

- 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。在这种情况下，您将为自定义模板呈现默认基础。
- 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)来引用您的自定义接口模板。
- 自定义您的网络参数的任何其他环境文件。
- 如果使用自定义网络，则自定义 network_data 文件。
- 如果创建额外的或自定义可组合网络，则自定义 network_data 文件和自定义 roles_data 文件。

**注意**

运行 openstack overcloud netenv validate 命令，以验证您的 network-environment.yaml 文件的语法。此命令还会验证用于计算、控制器、存储和可组合角色网络文件的独立 nic-config 文件。使用 -f 或 --file 选项指定您要验证的文件：

```
$ openstack overcloud netenv validate -f ~/templates/network-environment.yaml
```

## 11.1. 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适合特定网络布局的自定

义 NIC 模板。您可能想要将 上的控制服务和数据服务分隔到单独的 NIC。在这种情况下，您可以通过以下方式将服务映射到 NIC 分配：

- NIC1 (Provisioning)：
  - 置备/Control Plane
- NIC2 (Control Group)
  - 内部 API
  - 存储管理
  - 外部（公共 API）
- NIC3 (Data Group)
  - 租户网络(VXLAN 隧道)
  - 租户 VLAN/提供程序 VLAN
  - 存储
  - 外部 VLAN （浮动 IP/SNAT）
- NIC4 （管理）
  - 管理

11.2. 为自定义呈现默认网络接口模板

为简化自定义模板的配置，请呈现默认 NIC 模板的 Jinja2 语法，并使用渲染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 模板转换为其呈现的 YAML 版本，并将结果保存到 ~/openstack-tripleo-heat-templates-rendered 中。

如果使用自定义网络文件或自定义角色文件，您可以使用 -n 和 -r 选项分别包含这些文件。例如：

```
$ ./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. 您可以编辑 custom-nics 中设置的模板，以适应您自己的网络配置。

### 11.3. 网络接口架构

您在 第 11.2 节 “为自定义呈现默认网络接口模板” 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。此部分应当保持不变，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修改一些参数的值。

Resources

resources 部分是主网络接口配置的位置。在大多数情况下，resource 部分是唯一需要编辑 的资源部分。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:
```

这会运行一个脚本(run-os-net-config.sh)，它为 os-net-config 创建配置文件，以用于配置节点上的网络属性。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。根据设备类型，您可以按顺序排列该自定义接口数据。

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 模板的绝对位置。该脚本位于 undercloud 上的 /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh。

## 11.4. 网络接口参考

网络接口配置包括以下参数：

**interface**

定义单个网络接口。该配置使用实际接口名称("eth0"、"eth1"、"enp0s25")或一组数字接口("nic1", "nic3")定义各个接口。

例如：

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

表 11.1. 接口选项

| Option | 默认值 | Description |
| --- | --- | --- |
| name | | Interface 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 | | 分配给接口的 IP 地址列表 |
| Routes | | 分配给接口的路由列表。请参阅 Routes。 |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 主要 | False | 将接口定义为主接口 |
| defroute | True | 使用 DHCP 服务提供的默认路由。仅在启用 **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 | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 用于接口的 DNS 服务器列表 |
| ethtool_opts | | 将这个选项设置为 **"rx-flow-hash udp4 sdfn"**，以便在某些 NIC 上使用 VXLAN 时提高吞吐量。 |

***vlan***

***一个 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 11.2. VLAN 选项***

| Option | 默认值 | Description |
| --- | --- | --- |
| vlan_id | | VLAN ID |
| 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。请参阅 Routes。 |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 主要 | False | 将 VLAN 定义为主接口 |
| defroute | True | 使用 DHCP 服务提供的默认路由。仅在启用 **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 | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 用于 VLAN 的 DNS 服务器列表 |

***ovs_bond***

***在 Open vSwitch 中定义一个绑定，将两个或多个 接口 一起加入。这有助于冗余和增加带宽。***

***例如：***

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

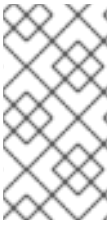
*表 11.3. ovs_bond options*

| Option | 默认值 | Description |
|---|---|---|
| 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。 |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 主要 | False | 将接口定义为主接口 |
| 成员 | | 绑定中使用的一系列接口对象 |
| ovs_options | | 创建绑定时要传递给 OVS 的一组选项 |
| ovs_extra | | 设置为在绑定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。仅在启用 **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 | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 用于绑定的 DNS 服务器列表 |

***ovs_bridge***

***在 Open vSwitch 中定义网桥，它将多个接口、ovs_bond 和 vlan 对象连接在一起。***

***网络接口类型 ovs_bridge 取一个参数 名称。***



**注意**

如果有多个网桥，则必须使用除接受 bridge_name 的默认名称以外的不同网桥名称。如果不使用不同的名称，则在聚合阶段，将两个网络绑定放置在同一网桥中。

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

例如：

```
- type: ovs_bridge
  name: 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 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，则与 Neutron 服务器的连接会在 OVS 升级时丢失，或者由管理员用户或进程重启 OVS 网桥。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，则控制组网络应放在单独的接口或绑定中，而不是在 OVS 网桥上：

- 在置备接口和 OVS 网桥上的 VLAN 上放置内部 API 网络时，可以达到最小设置。
- 如果要绑定，则至少需要两个绑定(four 网络接口)。该控制组应当放在 Linux 绑定(Linux 网桥)上。如果切换不支持 LACP 回退到单个接口进行 PXE 引导，则该解决方案至少需要 5 个 NIC。

*表 11.4. ovs_bridge options*

| Option | 默认值 | Description |
|---|---|---|
| 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。请参阅 Routes。 |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 成员 | | 网桥中使用的一系列接口、VLAN 和绑定对象 |
| ovs_options | | 创建网桥时要传递给 OVS 的一组选项 |
| ovs_extra | | 设置在网桥网络配置文件中作为 OVS_EXTRA 参数的选项 |
| defroute | True | 使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。 |
| persist_mapping | False | 编写设备别名配置，而不是系统名称 |
| dhclient_args | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 网桥使用的 DNS 服务器列表 |

***linux_bond***

***定义一个将两个或者多个 接口 接合在一起的 Linux 绑定。这有助于冗余和增加带宽。确保在 bonding_options 参数中包含基于内核的绑定选项。有关 Linux 绑定选项的详情请参考 7.7.1。Red Hat Enterprise Linux 7 网络指南中的绑定模块指令。***

***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。这样可确保绑定将 MAC 地址用于 nic2。***

***表 11.5. linux_bond options***

| Option | 默认值 | Description |
|---|---|---|
| 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。 |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 主要 | False | 将接口定义为主接口。 |
| 成员 | | 绑定中使用的一系列接口对象 |
| bonding_options | | 创建绑定时一组选项。有关 Linux 绑定选项的详情请参考 7.7.1。Red Hat Enterprise Linux 7 网络指南中的绑定模块指令。 |
| defroute | True | 使用 DHCP 服务提供的默认路由。仅在启用 **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 | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 用于绑定的 DNS 服务器列表 |

***linux_bridge***

*定义 **Linux 网桥，它将多个接口、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，它被替换为外部接口。***

*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 11.6. linux_bridge options***

| Option | 默认值 | Description |
|---|---|---|
| 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。请参阅 Routes。 |

| Option | 默认值 | Description |
| --- | --- | --- |
| mtu | 1500 | 连接的最大传输单元(MTU) |
| 成员 | | 网桥中使用的一系列接口、VLAN 和绑定对象 |
| defroute | True | 使用 DHCP 服务提供的默认路由。仅在启用 **use_dhcp** 或 **use_dhcpv6** 时才适用。 |
| persist_mapping | False | 编写设备别名配置，而不是系统名称 |
| dhclient_args | None | 传递给 DHCP 客户端的参数 |
| dns_servers | None | 网桥使用的 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
      default: true
      next_hop:
        get_param: EC2MetadataIp
```

| Option | 默认值 | Description |
| --- | --- | --- |
| ip_netmask | None | 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 | None | 用于访问目的地网络的路由器的 IP 地址。 |

### *11.5. 网络接口布局示例*

**以下示例 Controller 节点 NIC 模板片断如何配置自定义网络场景，使其保持控制组与 OVS 网桥独立：**

```
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:

              # NIC 1 - Provisioning
              - type: interface
                name: nic1
                use_dhcp: false
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop:
                    get_param: EC2MetadataIp

              # NIC 2 - Control Group
              - type: interface
                name: nic2
                use_dhcp: false
              - type: vlan
                device: nic2
                vlan_id:
                  get_param: InternalApiNetworkVlanID
                addresses:
                - ip_netmask:
                    get_param: InternalApiIpSubnet
              - type: vlan
                device: nic2
                vlan_id:
                  get_param: StorageMgmtNetworkVlanID
                addresses:
                - ip_netmask:
                    get_param: StorageMgmtIpSubnet
              - type: vlan
                device: nic2
                vlan_id:
                  get_param: ExternalNetworkVlanID
                addresses:
```

```
      - ip_netmask:
          get_param: ExternalIpSubnet
        routes:
        - default: true
          next_hop:
            get_param: ExternalInterfaceDefaultRoute

    # NIC 3 - Data Group
    - type: ovs_bridge
      name: bridge_name
      dns_servers:
        get_param: DnsServers
      members:
      - type: interface
        name: nic3
        primary: true
      - type: vlan
        vlan_id:
          get_param: StorageNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageIpSubnet
      - type: vlan
        vlan_id:
          get_param: TenantNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: TenantIpSubnet

      # NIC 4 - Management
      - type: interface
        name: nic4
        use_dhcp: false
        addresses:
        - ip_netmask: {get_param: ManagementIpSubnet}
        routes:
        - default: true
          next_hop: {get_param: ManagementInterfaceDefaultRoute}
```

***此模板使用四个网络接口，并将多个标记的 VLAN 设备分配给编号的接口 nic1 到 nic4。在 nic3 上，它创建托管 Storage 和租户网络的 OVS 网桥。因此，它会创建以下布局：***

- ***NIC1 (Provisioning)：***
  - ***置备/Control Plane***
- ***NIC2 (Control Group)***

- 内部 API
- 存储管理
- 外部（公共 API）

- NIC3 (Data Group)
  - 租户网络(VXLAN 隧道)
  - 租户 VLAN/提供程序 VLAN
  - 存储
  - 外部 VLAN （浮动 IP/SNAT）

- NIC4 （管理）
  - 管理

### 11.6. 自定义网络的网络接口注意事项

当您使用可组合网络时，process-templates.py 脚本会呈现静态模板，使其包含您在 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（包括自定义可组合网络）的静态文件。
- 每个角色的每个静态文件都包含正确的网络定义。

每个静态文件都需要自定义网络的所有参数定义，即使角色上未使用了网络。检查以确保渲染的模板包含这些参数。例如，如果只将 StorageBackup 网络添加到 Ceph 节点，则所有角色的 NIC 配置模板中的 parameter 部分还包括此定义：

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

如果需要，您还可以包含 VLAN ID 和/或网关 IP 的参数定义：

```
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 配置模板才会在模板的 network_config 部分中使用 StorageBackup 参数。

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

### 11.7. 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，它描述 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 部分包含对各个节点角色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] 是该特定角色对应的网络接口模板。例如：*

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

*parameter_defaults 部分包含定义每种网络类型的网络选项的参数列表。*

### *11.8. 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 |
|---|---|---|
| **ControlPlaneDefaultRoute** | Control Plane 上的路由器 IP 地址，它默认用作 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，则设置为 undercloud IP。 | 字符串 |
| **ControlPlaneSubnetCidr** | Control Plane 上使用 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（尽管始终是一个数字） |
| ***NetCidr** | 特定网络的完整网络和 CIDR 子网掩码。默认值自动设置为 **network_data** 文件中的网络的 **ip_subnet** 设置。例如：**InternalApiNetCidr: 172.16.0.0/24** | 字符串 |
| ***AllocationPools** | "特定网络的 IP 分配范围。默认值自动设置为 **network_data** 文件中的 networks **allocation_pools** 设置。例如：**InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]** | hash |

| 参数 | 描述 | 类型 |
| --- | --- | --- |
| ***NetworkVlanID** | 特定网络上的节点的 VLAN ID。默认会自动设置为 **network_data** 文件中的网络 **vlan** 设置。例如：**InternalApiNetworkVlanID: 201** | number |
| ***InterfaceDefaultRoute** | 特定网络的路由器地址，您可以使用这些地址作为角色的默认路由，或用于路由到其他网络。默认值自动设置为 **network_data** 文件中的网络 **gateway_ip** 设置。例如：**InternalApiInterfaceDefaultRoute: 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 |
| **EC2MetadataIp** | 用于置备 overcloud 节点的元数据服务器的 IP 地址。设置为 Control Plane 上 undercloud 的 IP 地址。 | 字符串 |
| **BondInterfaceOvsOptions** | 绑定接口的选项。例如：**BondInterfaceOvsOptions: "bond_mode=balance-slb"** | 字符串 |
| **NeutronExternalNetworkBridge** | 用于 OpenStack Networking (neutron)的外部网桥名称旧值。默认情况下，这个值为空，允许在 **NeutronBridgeMappings** 中定义多个物理网桥。这通常不应该被覆盖。 | 字符串 |
| **NeutronFlatNetworks** | 定义要在 neutron 插件中配置的扁平网络。默认为 "datacentre"，以允许外部网络创建。例如：**NeutronFlatNetworks: "datacentre"** | 字符串 |
| **NeutronBridgeMappings** | 要使用的物理网桥映射逻辑。默认为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，您指的是逻辑名称。例如，**NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"** | 字符串 |

| 参数 | 描述 | 类型 |
| --- | --- | --- |
| **NeutronPublicInterface** | 定义在不使用网络隔离时，为网络节点桥接到 **br-ex** 的接口。通常，除了具有两个网络的小型部署中，不使用。例如：**NeutronPublicInterface: "eth0"** | 字符串 |
| **NeutronNetworkType** | T OpenStack Networking (neutron)的租户网络类型。要指定多个值，请使用逗号分隔列表。指定的第一个类型在所有可用网络都用尽前，将使用下一个类型。例如：**NeutronNetworkType: "vxlan"** | 字符串 |
| **NeutronTunnelTypes** | neutron 租户网络的隧道类型。要指定多个值，使用以逗号分开的字符串。例如：*NeutronTunnelTypes: 'gre,vxlan'* | 字符串/用逗号分开的列表 |
| **NeutronTunnelIdRanges** | 用于租户网络分配的 GRE 隧道 ID 范围。例如：**NeutronTunnelIdRanges "1:1000"** | 字符串 |
| **NeutronVniRanges** | 用于租户网络分配的 VXLAN VNI ID 范围。例如：**NeutronVniRanges: "1:1000"** | 字符串 |
| **NeutronEnableTunnelling** | 定义是启用还是彻底禁用所有隧道网络。除非您确保绝不会想要创建调整的网络，否则 将启用此项。默认值为 enabled。 | 布尔值 |
| **NeutronNetworkVLANRanges** | ML2 和 Open vSwitch VLAN 映射范围以提供支持。默认为允许 **datacentre** 物理网络中的任何 VLAN。要指定多个值，请使用逗号分隔列表。例如：**NeutronNetworkVLANRanges: "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。默认为"openvswitch"。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/用逗号分开的列表 |

### 11.9. 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，您可以使用该文件来启用 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
  # The IP address of the EC2 metadata server. Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.2.1
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  NeutronExternalNetworkBridge: """
```

### 11.10. 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 部署命令中包括所有相关的网络环境文件。

**流程**

1. 在运行 **openstack overcloud deploy** 命令时，请确保包括：

   - 自定义 **network_data** 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。
   - 包含自定义 NIC 模板资源引用的自定义环境网络配置。

- 任何与您配置相关的额外环境文件。

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，然后是 **network-environment.yaml** 文件。后续的 **custom-network-configuration.yaml** 覆盖了来自前两个文件中的 **OS::TripleO::[ROLE]::Net::SoftwareConfig** 资源。

- 如果使用可组合网络，则使用以下命令包含 **network_data** 和 **roles_data** 文件。

# 第 12 章 额外网络配置

本章介绍了 第 11 章 自定义网络接口模板 中介绍的概念和步骤，并提供了一些附加信息来帮助配置 overcloud 网络的部分。

## 12.1. 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 连接到基础架构网络所需的修改，并使用 DHCP 地址连接到基础架构网络，并使用第三和第四个 NIC 作为绑定：

```
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的接口名称(eth0、eth 1、enp0s25)或一组编号的接口(nic1、 nic2、nic3)。当使用编号接口(nic1、nic 2 等)而不是指定接口(eth 0、eno2 等)时，角色内主机的网络接口不必完全相同。例如，一个主机可能具有 em1 和 em2 接口，而另一个主机具有 eno1 和 eno2，但您可以将两个主机的 NIC 指代为 nic1 和 nic2。

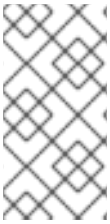
数字接口的顺序与命名网络接口类型的顺序对应：

- ethX 接口，如 eth 0、eth1 等。它们通常是板载的接口。
- enoX 接口，如 eno 0、eno1 等。它们通常是板载的接口。

- *enX 接口，按数字顺序排序，如 enp3s 0、enp3s1、 ens3 等。它们通常是附加组件接口。*

*numbered NIC 方案只考虑在线接口，例如，如果它们附加了交换机的电缆。如果您有一个有四个接口以及一些带有六个接口的主机，您应该使用 nic1 到 nic4，并且仅在每台主机上插入四个电缆。*

*您可以将物理接口硬编码到特定的别名。这样，您可以预先确定哪个物理 NIC 将映射为 nic1 或 nic2 等。您还可以将 MAC 地址映射到指定的别名。*



*注意*

*通常，OS-net-config 将仅注册已处于 UP 状态的接口。但是，如果您使用自定义映射文件执行硬代码接口，那么即使接口处于 DOWN 状态，也会注册该接口。*

*使用环境文件将接口映射到别名。在本例中，每个节点都有 nic1 和 nic2 预定义条目。*

*注意*

*如果要使用 NetConfigDataLookup 配置，在 NodeUserData 资源 registry 中还必须包含 os-net-config-mappings.yaml 文件。*

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack/tripleo-heat-templates/firstboot/os-net-config-
mappings.yaml
parameter_defaults:
  NetConfigDataLookup:
    node1:
      nic1: "em1"
      nic2: "em2"
    node2:
      nic1: "00:50:56:2F:9F:2E"
      nic2: "em2"
```

*然后，生成的配置会被 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。*

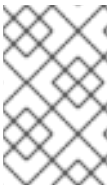
### *12.2. 配置路由和默认路由*

*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，且 DHCP 服务器提供网关地址，系统为该网关使用默认路由。否则，您可以使用静态 IP 在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那一个。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 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 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
```

### 12.3. 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更少，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。确保在绑定和/或接口上包括 MTU 值。*

*存储、存储管理、内部 API 和租户网络都受益于巨型帧。在测试中，项目的网络吞吐量在将巨型帧与 VXLAN 隧道结合使用时具有显著改进。*

*注意*

*建议 Provisioning 接口、外部接口和任何浮动 IP 接口保留在 1500 的默认 MTU 中。否则可能会发生连接问题。这是因为路由器通常无法跨第 3 层边界转发巨型帧。*

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

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

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

### *12.4. 在中继接口上配置原生 VLAN*

如果中继接口或绑定在原生 VLAN 上有一个网络，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，则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:
      - ip_netmask: 0.0.0.0/0
        next_hop:
          get_param: ExternalInterfaceDefaultRoute
    members:
      - type: ovs_bond
        name: bond1
        ovs_options:
          get_param: BondInterfaceOvsOptions
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

**注意**

将地址（以及可能路由）语句移动到网桥时，从网桥中删除对应的 VLAN 接口。对所有适用的角色进行更改。外部网络仅依赖于控制器，因此只有控制器模板需要更改。另一方面，存储网络会附加到所有角色，因此如果存储网络位于默认 VLAN 上，则所有角色都需要修改。

### 12.5. 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，并在虚拟网络中提供网络地址转换(NAT)。有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: table full, drop 数据包）。您可以增加连接跟踪(conntrack)的限值，并避免这些类型的错误。您可以为 RHOSP 部署中的一个或多个角色增加 conntrack 限制。

前提条件

- *成功安装 RHOSP undercloud。*

*流程*

1. *以 stack 用户身份登录 undercloud 主机。*

2. *提供 undercloud 凭据文件：*

```
$ source ~/stackrc
```

3. *创建自定义 YAML 环境文件。*

*Example*

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

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

*Example*

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

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

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

```
parameter_defaults:
  <role>Parameters:
    ExtraSysctlSettings:
```

```
    net.nf_conntrack_max:
      value: <simultaneous_connections>
```

- 将 <role > 替换为角色的名称。

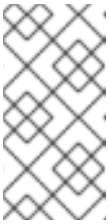
  例如，使用 ControllerParameters 为 Controller 角色设置 conntrack 限值，或者为 Compute 角色设置 conntrack 限值。

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

  **Example**

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

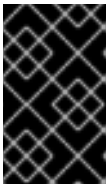
```
parameter_defaults:
  ControllerParameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: 500000
```



**注意**

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

5. 运行部署命令，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更为优先。

**Example**

```
$ 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 13 章 网络接口绑定

本章定义了在自定义网络配置中使用的一些绑定选项。

## 13.1. 网络接口绑定和链路聚合控制协议(LACP)

您可以将多个物理 NIC 捆绑到一起组成一个逻辑频道，称为绑定。您可以配置绑定来为高可用性系统或增加吞吐量提供冗余。

重要

不支持在绑定接口中使用单根 I/O 虚拟化(SR-IOV)。

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

绑定可与可选链路聚合控制协议(LACP)一起使用。LACP 是一个协商协议，可为负载平衡和容错创建动态绑定。

红帽建议使用 Linux 内核绑定(bond type: linux_bond) over OvS 内核绑定(bond type: ovs_bond)。用户模式绑定(bond type：ovs_dpdk_bond)必须与用户模式桥接(type: ovs_user_bridge)一同使用，而不是内核模式网桥(type: ovs_bridge)。但是，不要在同一节点上组合 ovs_bridge 和 ovs_user_bridge。

在控制和存储网络上，红帽建议使用带有 VLAN 和 LACP 的 Linux 绑定，因为 OVS 绑定增加了在 OVS 或 neutron 代理重启时出现 control plane 中断的可能性。Linux 绑定/LACP/VLAN 配置在不出现 OVS 中断的情况下提供 NIC 管理。

以下是一个使用 VLAN 的 Linux 绑定配置示例。

```
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
```

***以下示例显示了将 Linux 绑定插入到 OVS 网桥中***

```
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}
```

***以下示例显示了 OVS 用户空间网桥：***

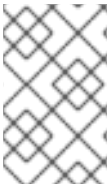
```
params:
        $network_config:
```

```
    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}
      #ovs_extra:
      #- set interface dpdk0 mtu_request=$MTU
      #- set interface dpdk1 mtu_request=$MTU
      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
```

### *13.2. OPEN VSWITCH 绑定选项*

***overcloud 通过 Open vSwitch (OVS)提供网络。使用下表来了解 OVS 内核和 OVS-DPDK 对绑定接口的支持兼容性。OVS/OVS-DPDK balance-tcp 模式仅作为技术预览提供。***



***注意***

*这个支持需要 **Open vSwitch 2.9** 或更高版本。*

| OVS 绑定模式 | Application（应用程序） | 备注 | 兼容 LACP 选项 |
|---|---|---|---|
| active-backup | 高可用性（主动-被动） | | 主动、被动或关闭 |
| balance-slb | 增加吞吐量（主动） | • 性能受每个数据包的额外解析影响。<br>• vhost-user 锁定争用的可能性。 | 主动、被动或关闭 |

| | | | |
|---|---|---|---|
| balance-tcp （仅技术预览） | 不建议(active-active) | • 取消 L4 哈希所需的性能影响。<br>• 与 balance-slb 一样，性能会受到每个数据包的额外解析影响，并可能对 vhost-user 锁定争用造成潜在的影响。<br>• 必须启用 LACP。 | 主动或被动 |

***您可以使用 BondInterfaceOvsOptions 参数在网络环境文件中配置绑定接口，如下例所示：***

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

### *13.3. LINUX 绑定选项*

***您可以在网络接口模板中使用 LACP 和 Linux 绑定：***

```
    - 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"
```

- ***模式 - 启用 LACP。***
- ***lacp_rate - 定义是否每 1 秒发送 LACP 数据包，还是每 30 秒发送一次。***
- ***updelay - 定义接口必须在用于流量前必须激活的最短时间（这有助于缓解端口中断）。***
- ***miimon - 用于使用驱动程序的 MIIMON 功能监控端口状态的时间间隔（毫秒）。***

*有关 Linux 绑定选项的详情请参考 7.7.1。绑定模块指令 （在 Red Hat Enterprise Linux 7 网络指南》中。*

### 13.4. OVS 绑定选项

*下表根据硬件提供了这些选项和一些替代方案的一些解释。*

*表 13.1. 绑定选项*

| | |
|---|---|
| **bond_mode=balance-slb** | 根据源 MAC 地址和输出 VLAN 平衡流，在流量特征发生变化时定期重新平衡。与 **balance-slb** 绑定可以实现有限的负载均衡形式，无需远程交换机知识或协作。SLB 将每个源 MAC 和 VLAN 对分配给链接，并通过该链接传输来自该 MAC 和 VLAN 的所有数据包。这个模式使用基于源 MAC 地址和 VLAN 号码的简单哈希算法，在流量特征改变时定期重新平衡。这个模式与 Linux 绑定驱动程序使用的模式 2 绑定类似。这个模式也可用于提供负载均衡，即使交换机没有配置为使用 LACP。 |
| **bond_mode=active-backup** | 这个模式提供活跃的/standby 故障转移，其中待机 NIC 会在活跃连接失败时恢复网络操作。物理交换机仅显示一个 MAC 地址。这个模式不需要任何特殊的交换机支持或配置，当链接连接到独立的交换机时可以正常工作。这个模式不提供负载均衡。 |
| **lacp=[active\|passive\|off]** | 控制链路聚合控制协议(LACP)行为。只有某些交换机支持 LACP。如果您的交换机不支持 LACP，请使用 **bond_mode=balance-slb** 或 **bond_mode=active-backup**。 |
| **other-config:lacp-fallback-ab=true** | 将 LACP 行为设置为将 bond_mode=active-backup 设置为回退。 |
| **other_config:lacp-time=[fast\|slow]** | 将 LACP heartbeat 设置为 1 秒(fast)或 30 秒（低）。默认值为 slow。 |
| **other_config:bond-detect-mode=[miimon\|carrier]** | 将链路检测设置为使用 miimon heartbeats (miimon)或监控载波（错误）。默认值为载体。 |
| **other_config:bond-miimon-interval=100** | 如果使用 miimon，以毫秒为单位设置 heartbeat 间隔。 |
| **bond_updelay=1000** | 需要激活链接的毫秒数才能防止出现问题。 |
| **other_config:bond-rebalance-interval=10000** | 在绑定成员之间重新平衡流之间毫秒。将此值设置为 0，以禁用绑定成员间的重新平衡流量。 |

# 第 14 章 控制节点放置

director 的默认行为是为每个角色随机选择节点，这通常基于其配置集标签。但是，director 提供了定义特定节点放置的功能。这是以下几个有用的方法：

- 分配特定的节点 ID，如 controller-0、controller-1 等
- 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**

手动设置可预测 IP 地址、虚拟 IP 地址和端口，可以减轻分配池的需求。但是，建议为每个网络保留分配池，以便轻松扩展新节点。确保静态定义的 IP 地址不在分配池之外。有关设置分配池的更多信息，请参阅 第 11.7 节 “自定义网络环境文件”。

## 14.1. 分配特定节点 ID

此流程为特定节点分配节点 ID。节点 ID 示例包括 controller-0、controller-1、compute-0、compute-1 等。

第一步是将 ID 分配为部署时与计算调度程序匹配的每个节点功能。例如：

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

这会将功能 node:controller-0 分配给节点。为所有节点使用唯一连续索引（从 0 开始）重复此模式。确保给定角色(Controller、Compute 或每个存储角色)的所有节点都相同，否则计算调度程序将不会正确匹配功能。

下一步是使用调度程序提示来匹配每个节点的功能，创建 Heat 环境文件（如 scheduler_hints_env.yaml）。例如：

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

要使用这些调度程序提示，请在 Overcloud 创建过程中包括 ' scheduler_hints_env.yaml' 环境文件及 overcloud deploy 命令。

每个角色都可以通过这些参数进行相同的方法：

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

注意

节点放置优先于配置集匹配。为避免调度失败，请使用默认的 baremetal 类别进行部署，而不是为配置文件匹配的类别(计算、控制 等)。例如：

```
$ openstack overcloud deploy ... --control-flavor baremetal --compute-flavor baremetal
...
```

## 14.2. 分配自定义主机名

与 第 14.1 节 “分配特定节点 ID” 中的节点 ID 配置相结合，director 还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在的位置（例如 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

不要在节点部署后重命名节点。在部署后重命名节点会导致实例管理问题。

要自定义节点主机名，请使用环境文件中的 HostnameMap 参数，如来自 第 14.1 节 “分配特定节点 ID” 的 ' scheduler_hints_env.yaml' 文件。例如：

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

在 parameter_defaults 部分中定义 HostnameMap，并将每个 map 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（如 overcloud-controller-prod-123-0）。

将此方法与节点 ID 放置结合使用可确保每个节点具有自定义主机名。

### 14.3. 分配可预测 IP

为进一步控制生成的环境，director 还可在每个网络上分配具有特定 IP 的 Overcloud 节点。在核心 Heat 模板集合中，使用 environments/ips-from-pool-all.yaml 环境文件。

将这个文件复制到 stack 用户的 templates 目录中。

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

ips-from-pool-all.yaml 文件中有两个主要部分。

第一个是覆盖默认值的 resource_registry 引用集合。它们告诉 director 对节点类型的给定端口使用特定 IP。修改每个资源，以使用其对应模板的绝对路径。例如：

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

默认配置将所有节点上的所有网络设置为使用预先分配的 IP。若要允许特定网络或节点类型使用默认 IP 分配，只需从环境文件中删除与该节点类型或网络相关的 resource_registry 条目。

第二个部分是 parameter_defaults，其中分配了实际的 IP 地址。每个节点类型都有一个关联的参数：

- Controller 节点的 ControllerIP。
- Compute 节点的 ComputeIP。
- Ceph Storage 节点的 CephStorageIP。
- BlockStorageIPs 用于块存储节点。
- 对象存储 节点的 SwiftStorageIP。
- [ROLE]IP 用于自定义角色。用角色名称替换 [ROLE]。

每个参数都是一个到地址列表的网络名称映射。每种网络类型必须至少包含与该网络上节点相同的地址。director 会按顺序分配地址。每种类型的第一个节点接收各自列表中的第一个地址，第二个节点接收各个各个列表中的第二个地址，依此类推。

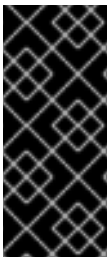
例如，如果 Overcloud 将包含三个 Ceph Storage 节点，CephStorageIPs 参数可能类似如下：

```
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。第二个地址接收 172.16.1.101 和 172.16.3.101，第三个则接收 172.16.1.102 和 172.16.3.102。相同的模式也适用于其他节点类型。

确保所选的 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（请参阅 第 11.7 节 “自定义网络环境文件”）。例如，确保 internal_api 分配在 InternalApiAllocationPools 范围之外。这可避免与所选 IP 相冲突。同样，请确保 IP 分配不会与 VIP 配置冲突，可以是标准的可预测 VIP 放置（请参阅 第 14.4 节 “分配可预测虚拟 IP”）或外部负载均衡（请参阅 第 23.2 节 “配置外部负载平衡”）。



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，即使删除节点也是如此。要指定列表中给定条目不再被使用，请将 IP 值替换为 DELETED 或 UNUSED 等值。不应从 IP 列表中删除条目，只应更改或添加。

要在部署期间应用此配置，请使用 openstack overcloud deploy 命令包含 ips-from-pool-all.yaml 环境文件。

重要

如果使用网络隔离，请在 network-isolation.yaml 文件后包含 ips-from-pool-all.yaml 文件。

例如：

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

### 14.4. 分配可预测虚拟 IP

除了为每个节点定义可预测的 IP 地址外，director 还提供类似为集群服务定义可预测虚拟 IP (VIP)的功能。要做到这一点，从 第 11.7 节 “自定义网络环境文件” 编辑网络环境文件，并在 parameter_defaults 部分中添加 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'}]
```

从对应的分配池范围之外，选择这些 IP。例如，为 InternalApiVirtualFixedIPs 选择一个 IP 地址，该地址没有在 InternalApiAllocationPools 范围内。

此步骤只适用于使用默认内部负载均衡配置的 overcloud。如果使用外部负载均衡器分配 VIP，请使用 Overcloud 指南中的专用外部负载平衡 中的步骤。

# 第 15 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其服务。这意味着 overcloud 配置需要额外的环境文件来为其公共 API 端点启用 SSL/TLS。下面的章节演示了如何配置 SSL/TLS 证书，并将它作为 overcloud 创建的一部分包含在内。

**注意**

这个过程只支持公共 API 端点启用 SSL/TLS。内部和 Admin API 仍未加密。

此过程需要网络隔离来定义公共 API 的端点。

## 15.1. 初始化签名主机

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

/etc/pki/CA/index.txt 文件存储所有签名证书的记录。检查是否存在此文件。如果不存在，请创建一个空文件：

```
$ sudo touch /etc/pki/CA/index.txt
```

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

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

## 15.2. 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，您可能需要使用自己的证书颁发机构。例如，您可能想获得仅限内部的证书颁发机构。

例如，生成密钥和证书对以充当证书颁发机构：

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

openssl req 命令会要求输入认证机构的详细信息。输入这些详情。

这会创建一个名为 ca.crt.pem 的证书颁发机构文件。

### 15.3. 将证书颁发机构添加到客户端

对于旨在使用 SSL/TLS 通信的任何外部客户端，请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。复制到客户端后，在客户端上运行以下命令将其添加到证书颁发机构信任捆绑包中：

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

例如，undercloud 需要证书颁发机构文件的副本，以便它在创建期间与 overcloud 端点通信。

### 15.4. 创建 SSL/TLS 密钥

运行以下命令生成 SSL/TLS 密钥(server.key.pem)，我们在不同时间点上生成 undercloud 或 overcloud 证书：

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

### 15.5. 创建 SSL/TLS 证书签名请求

下一个流程为 overcloud 创建证书签名请求。复制默认的 OpenSSL 配置文件以进行自定义。

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

编辑自定义 openssl.cnf 文件并设置用于 overcloud 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 = 10.0.0.1
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 = 10.0.0.1
DNS.1 = 10.0.0.1
DNS.2 = myovercloud.example.com
```

**将 commonName_default 设置为以下内容之一：**

- **如果使用 IP 通过 SSL/TLS 访问，请将虚拟 IP 用于公共 API。使用环境文件中的 PublicVirtualFixedIPs 参数设置此 VIP。更多信息请参阅 第 14.4 节 “分配可预测虚拟 IP”。如果您不使用可预测的 VIP，director 从 ExternalAllocationPools 参数中定义的范围分配第一个 IP 地址。**

- **如果使用完全限定域名通过 SSL/TLS 访问，则改为使用域名。**

**在 alt_names 部分中，包含与 IP 条目相同的公共 API IP 地址以及 DNS 条目。如果也使用 DNS，请在同一部分中包含服务器的主机名作为 DNS 条目。有关 openssl.cnf 的更多信息，请运行 man openssl.cnf。**

**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
```

**确保为 -key 选项包括您在 第 15.4 节 “创建 SSL/TLS 密钥” 中创建的 SSL/TLS 密钥。**

**使用 server.csr.pem 文件在下一节中创建 SSL/TLS 证书。**

### 15.6. 创建 SSL/TLS 证书

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：

- 指定 v3 扩展的配置文件。将此作为 -config 选项包含。
- 来自 第 15.5 节 “创建 SSL/TLS 证书签名请求” 的证书签名请求来生成证书并通过证书颁发机构进行签名。将此作为 -in 选项包含。
- 您在 第 15.2 节 “创建证书颁发机构” 中创建的证书颁发机构，为证书签名。将此作为 -cert 选项包含。
- 您在 第 15.2 节 “创建证书颁发机构” 中创建的证书颁发机构私钥。将此包括为 -keyfile 选项。

这会导致名为 server.crt.pem 的证书。将此证书与来自 第 15.4 节 “创建 SSL/TLS 密钥” 的 SSL/TLS 密钥一起使用，以启用 SSL/TLS。

### 15.7. 启用 SSL/TLS

从 Heat 模板集合中复制 enable-tls.yaml 环境文件：

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

编辑此文件并对这些参数进行以下更改：

**SSLCertificate**

将证书文件的内容(server.crt.pem)复制到 SSLCertificate 参数。例如：

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

**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都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。例如：

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO9hkJZnGP6qb6wtYUoy1bVP7
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```

重要

私钥内容需要对所有新行的缩进级别相同。

**OS::TripleO::NodeTLSData**

将 OS::TripleO::NodeTLSData 的资源 路径改为绝对路径：

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

### 15.8. 注入 ROOT 证书

如果证书 signer 没有包括在 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。从 heat 模板集合中复制 inject-trust-anchor.yaml 环境文件：

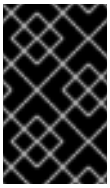
```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/inject-trust-anchor.yaml
~/templates/.
```

编辑此文件并对这些参数进行以下更改：

**SSLRootCertificate**

将 root 证书颁发机构文件(ca.crt.pem)的内容复制到 SSLRootCertificate 参数。例如：

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



重要

证书颁发机构内容为所有新行都需要相同的缩进级别。

**OS::TripleO::NodeTLSCAData**

将 OS::TripleO::NodeTLSCAData 的资源 路径改为绝对路径：

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

如果要注入多个 CA，您可以使用 inject-trust-anchor-hiera.yaml 环境文件。例如，您可以同时注入 undercloud 和 overcloud 的 CA：

```
parameter_defaults:
  CAMap:
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        ... cert content ...
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        ... cert content ...
        -----END CERTIFICATE-----
```

### 15.9. 配置 DNS 端点

**如果使用 DNS 主机名通过 SSL/TLS 访问 overcloud，则需要将 custom-domain.yaml 文件复制到 /home/stack/templates 中。您可以在 /usr/share/tripleo-heat-templates/environments/predictable-placement/ 中找到这个文件。**

1. **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：**

**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。**

```
# title: Custom Domain Name
# description: |
#   This environment contains the parameters that need to be set in order to
#   use a custom domain name and have all of the various FQDNs reflect it.
parameter_defaults:
  # The DNS domain used for the hosts. This must match the overcloud_domain_name
configured on the undercloud.
  # Type: string
  CloudDomain: localdomain

  # The DNS name of this cloud. E.g. ci-overcloud.tripleo.org
  # Type: string
  CloudName: overcloud.localdomain

  # The DNS name of this cloud's provisioning network endpoint. E.g. 'ci-
overcloud.ctlplane.tripleo.org'.
  # Type: string
  CloudNameCtlplane: overcloud.ctlplane.localdomain

  # The DNS name of this cloud's internal_api endpoint. E.g. 'ci-
overcloud.internalapi.tripleo.org'.
  # Type: string
```

```
  CloudNameInternal: overcloud.internalapi.localdomain

  # The DNS name of this cloud's storage endpoint. E.g. 'ci-overcloud.storage.tripleo.org'.
  # Type: string
  CloudNameStorage: overcloud.storage.localdomain

  # The DNS name of this cloud's storage_mgmt endpoint. E.g. 'ci-
overcloud.storagemgmt.tripleo.org'.
  # Type: string
  CloudNameStorageManagement: overcloud.storagemgmt.localdomain
```

2. **在新的或现有环境文件中，添加用于参数默认值的 DNS 服务器列表：**

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

### 15.10. 在 OVERCLOUD 创建过程中添加环境文件

**部署命令(openstack overcloud deploy)使用 -e 选项添加环境文件。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(cloudname.yaml)**
- **注入 root 证书颁发机构(inject-trust-anchor.yaml的环境文件)**
- **设置公共端点映射的环境文件：**
  - **如果使用 DNS 名称访问公共端点，请使用 /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml**
  - **如果使用 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**

**例如：**

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

### 15.11. 更新 SSL/TLS 证书

如果需要在以后更新证书：

- 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.yaml** 文件并更新 **SSLRootCertificate** 参数。

新证书内容就位后，重新运行您的部署命令。例如：

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

# 第 16 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 端点中启用 SSL/TLS。由于所需的证书数量，director 与 Red Hat Identity Management (IdM)服务器集成以充当证书颁发机构和管理 overcloud 证书。此过程涉及使用 novajoin 将 overcloud 节点注册到 IdM 服务器。

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

## 16.1. 将 UNDERCLOUD 添加到 CA

在部署 overcloud 之前，您必须将 undercloud 添加到证书颁发机构(CA)中：

1. 在 undercloud 节点上，安装 python-novajoin 软件包：

```
$ sudo yum install python-novajoin
```

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

```
$ sudo /usr/libexec/novajoin-ipa-setup \
    --principal admin \
    --password <IdM admin password> \
    --server <IdM server hostname> \
    --realm <overcloud cloud domain (in upper case)> \
    --domain <overcloud cloud domain> \
    --hostname <undercloud hostname> \
    --precreate
```

在以下部分中，您将使用生成的一次性密码(OTP)注册 undercloud。

## 16.2. 将 UNDERCLOUD 添加到 IDM

此流程使用 IdM 注册 undercloud，并配置 novajoin。在 undercloud.conf 中配置以下设置（在 [DEFAULT] 部分中）：

1. 默认情况下，禁用 novajoin 服务。启用它：

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

2. 您需要设置一个一次性密码(OTP)，以使用 IdM 注册 undercloud 节点：

```
ipa_otp = <otp>
```

3. 确保 neutron 的 DHCP 服务器提供 overcloud 的域名与 IdM 域（您小写的 kerberos 域）匹配：

```
overcloud_domain_name = <domain>
```

4. 为 undercloud 设置适当的主机名：

```
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，您需要查看 novajoin 连接超时值。在 undercloud.conf 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（以秒为单位），例如 5：

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

7. 保存 undercloud.conf 文件。

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

```
$ openstack undercloud install
```

*验证*

1. *检查 **keytab** 文件是否有 **undercloud** 的密钥条目：*

```
[root@undercloud-0 ~]# klist -kt
Keytab name: FILE:/etc/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL


[root@undercloud-0 ~]# klist -kt /etc/novajoin/krb5.keytab
Keytab name: FILE:/etc/novajoin/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
```

2. ***使用主机原则测试系统 /etc/krb.keytab 文件：***

```
[root@undercloud-0 ~]# kinit -k
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: host/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:34:30  05/05/2020 10:34:30  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL

[root@undercloud-0 ~]# kdestroy
Other credential caches present, use -A to destroy all
```

3. ***使用 nova 原则测试 novajoin /etc/novajoin/krb.keytab 文件：***

```
[root@undercloud-0 ~]# kinit -kt /etc/novajoin/krb5.keytab 'nova/undercloud-
0.redhat.local@REDHAT.LOCAL'
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: nova/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:39:14  05/05/2020 10:39:14  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL
```

### *16.3. 配置 OVERCLOUD DNS*

***要自动检测 IdM 环境并更容易注册，请考虑使用 IdM 作为您的 DNS 服务器：***

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。*

### 16.4. 将 OVERCLOUD 配置为使用 NOVAJOIN

1. *要启用 IdM 集成，请创建一个 /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，并将 CloudDomain 和 CloudName* 值设置为适合您的部署。例如：*

```
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. *在 overcloud 部署过程中包含以下环境文件：*

- */usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml*

- */usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml*

- */home/stack/templates/custom-domain.yaml*

*例如：*

```
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 \
```

*因此，部署的 overcloud 节点将自动注册到 IdM。*

4. *这仅为内部端点设置 TLS。对于外部端点，您可以使用普通的方法为 /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml 环境文件（必须修改它来添加自定义证书和密钥）。因此，您的 openstack deploy 命令与此类似：*

```
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
```

5. *另外，您还可以使用 IdM 发布公共证书。在这种情况下，您需要使用 /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml 环境文件。例如：*

```
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 /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml
```

# 第 17 章 将您的现有部署转换为使用 TLS

您可以将现有的 overcloud 和 undercloud 端点配置为使用 TLS 加密。此方法使用 novajoin 将部署与红帽身份管理(IdM)集成，允许访问 DNS、Kerberos 和 certmonger。每个 overcloud 节点都使用 certmonger 客户端来检索每个服务的证书。

有关 TLS 的更多信息，请参阅 安全性和强化指南。

## 17.1. 要求

- 对于 Red Hat OpenStack Platform 13，您必须运行版本 z8 或更高版本。
- 您必须有一个现有的 IdM 部署，还必须为 OpenStack 部署提供 DNS 服务。
- 现有部署必须将 FQDN 用于公共端点。默认配置可能使用基于 IP 地址的端点，因此会生成基于 IP 地址的证书；在继续执行这些步骤前，必须将其改为 FQDN。

重要

此过程期间，overcloud 和 undercloud 服务将不可用。

## 17.2. 检查您的端点

默认情况下，您现有的 Red Hat OpenStack Platform 13 overcloud 不会使用 TLS 加密某些端点。例如，以下输出包括使用 http 而不是 https 的 URL；这些 URL 不加密：

```
+----------------------------------+-----------+--------------+--------------+---------+-----------+-----------------------------------------+
| ID                               | Region    | Service Name | Service Type | Enabled | Interface | URL                                     |
+----------------------------------+-----------+--------------+--------------+---------+-----------+-----------------------------------------+
| 0ad11e943e1f4ff988650cfba57b4031 | regionOne | nova         | compute      | True    | internal  | http://172.16.2.17:8774/v2.1            |
| 1413eb9ef38a45b8bee1bee1b0dfe744 | regionOne | swift        | object-store | True    | public    | https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s |
| 1a54f13f212044b0a20468861cd06f85 | regionOne | neutron      | network      | True    | public    | https://overcloud.lab.local:13696       |
| 3477a3a052d2445697bb6642a8c26a91 | regionOne | placement    | placement    | True    | internal  | http://172.16.2.17:8778/placement       |
```

```
| 3f56445c0dd14721ac830d6afb2c2cd4 | regionOne | nova         | compute      | True    | admin     |
http://172.16.2.17:8774/v2.1                      |
| 425b1773a55c4245bcbe3d051772ebba | regionOne | glance       | image        | True    | internal  |
http://172.16.2.17:9292                           |
| 57cf09fa33ed446f8736d4228bdfa881 | regionOne | placement    | placement    | True    | public    |
https://overcloud.lab.local:13778/placement       |
| 58600f3751e54f7e9d0a50ba618e4c54 | regionOne | glance       | image        | True    | public    |
https://overcloud.lab.local:13292                 |
| 5c52f273c3284b068f2dc885c77174ca | regionOne | neutron      | network      | True    | internal  |
http://172.16.2.17:9696                           |
| 8792a4dd8bbb456d9dea4643e57c43dc | regionOne | nova         | compute      | True    | public    |
https://overcloud.lab.local:13774/v2.1            |
| 94bbea97580a4c4b844478aad5a85e84 | regionOne | keystone     | identity     | True    | public    |
https://overcloud.lab.local:13000                 |
| acbf11b5c76d44198af49e3b78ffedcd | regionOne | swift        | object-store | True    | internal  |
http://172.16.1.9:8080/v1/AUTH_%(tenant_id)s      |
| d4a1344f02a74f7ab0a50c5a7c13ca5c | regionOne | keystone     | identity     | True    | internal  |
http://172.16.2.17:5000                           |
| d86c241dc97642419ddc12533447d73d | regionOne | placement    | placement    | True    | admin
| http://172.16.2.17:8778/placement                 |
| de7d6c34533e4298a2752852427a7030 | regionOne | glance       | image        | True    | admin     |
http://172.16.2.17:9292                           |
| e82086062ebd4d4b9e03c7f1544bdd3b | regionOne | swift        | object-store | True    | admin     |
http://172.16.1.9:8080                            |
| f8134cd9746247bca6a06389b563c743 | regionOne | keystone     | identity     | True    | admin     |
http://192.168.24.6:35357                         |
| fe29177bd29545ca8fdc0c777a7cf03f | regionOne | neutron      | network      | True    | admin     |
http://172.16.2.17:9696                           |
+----------------------------------+-----------+--------------+--------------+---------+-----------+-------------------
-------------------------------------+
```

*以下小节解释了如何使用 TLS 加密这些端点。*

### *17.3. 对已知问题应用临时解决方案*

*目前，TLS Everywhere 存在已知问题，因此 overcloud 节点无法注册到 IdM。作为临时解决方案，在运行 overcloud 部署前从所有 overcloud 节点中删除 /etc/ipa/ca.crt/。如需更多信息，请参阅 https://bugzilla.redhat.com/show_bug.cgi?id=1732564。*

*例如，以下脚本是应用临时解决方案的一种方法。您可能需要调整此功能以适合您的部署。*

```
[stack@undercloud-0 ~]$ vi rm-ca.crt-dir.sh
#!/bin/bash

source /home/stack/stackrc
NODES=$(openstack server list -f value -c Networks|sed s/ctlplane=//g)

for NODE in $NODES
do
```

```
    ssh heat-admin@$NODE sudo rm -rf /etc/ipa/ca.crt/
Done

[stack@undercloud-0 ~]$ bash rm-ca.crt-dir.sh
```

### 17.4. 配置端点以使用 TLS

本节介绍如何为现有部署启用 TLS 端点加密，然后如何检查端点是否已正确配置。

在随处启用 TLS 时，有不同的升级路径，具体取决于您的域是如何构成的。这些示例使用示例域名来描述升级路径：

- 重复利用现有的公共端点证书，并在 overcloud 域(lab.local) 的内部 和管理端点(lab.local) 上都启用 TLS。
- 允许 IdM 发布新的公共端点证书，并在 overcloud 域( lab.local )与 IdM 域(lab.local) 的内部 和 admin 端点上都启用 TLS。
- 重复使用现有的公共端点证书，并在其中 overcloud 域(site1.lab.local)的 内部和外部 端点上启用 TLS (lab.local )是 IdM 域的子域(lab.local)。
- 允许 IdM 发布新的公共端点证书，并在其中 overcloud 域(site1.lab.local)是 IdM 域的子域 (lab.local)的 内部和外部 端点上启用 TLS。

本节中的步骤解释了如何使用上述各种组合配置此集成。

#### 17.4.1. 使用与 IdM 相同的域为部署配置 undercloud 集成

此流程描述了如何为使用与 IdM 相同的域的部署配置 undercloud 集成。

Red Hat OpenStack Platform 使用 novajoin 与 Red Hat Identity Management (IdM)集成，然后问题和管理加密证书。在此过程中，您将使用 IdM 注册 undercloud，生成令牌，在 undercloud 配置中启用令牌，然后重新运行 undercloud 和 overcloud 部署脚本。例如：

1. 安装 python-novajoin 以便与 IdM 集成：

```
[stack@undercloud-0 ~]$ sudo yum install python-novajoin
```

2. 运行 **novajoin 配置脚本，并提供您的 IdM 部署的配置详情。例如：**

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

**此输出包括 IdM 的一次性密码(OTP)，这是您的部署的不同值。**

3. **将 undercloud 配置为使用 novajoin，添加一次性密码(OTP)，对 DNS 使用 IdM IP 地址，并描述 overcloud 域。您的部署需要调整此示例：**

```
[stack@undercloud ~]$ vi undercloud.conf
...
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = lab.local
...
```

4. **在 undercloud 中安装 novajoin 服务：**

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

5. **将 overcloud IP 地址添加到 DNS。您需要修改此示例以适合您的部署：**

**注意：检查 overcloud 的 network-environment.yaml，然后在各个网络范围内选择一个 VIP。**

```
[root@ipa ~]$ ipa dnsrecord-add lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add storage.lab.local
[root@ipa ~]# ipa dnsrecord-add storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

6. 为所有端点创建一个 public_vip.yaml 映射：

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  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'}]
```

### 17.4.2. 为使用与 IdM 相同的域的部署配置 overcloud 集成，并保留现有的公共端点证书

1. 确保 openstack overcloud deploy 命令（具有有效设置）中存在以下参数，然后重新运行部署命令：

- ' --NTP-server' - 如果还没有设置，请指定 NTP 服务器以适应您的环境。IdM 服务器应该正在运行 ntp。
- cloud-names.yaml - 包含初始部署命令的 FQDN （而非 IP）。
- enable-tls.yaml - 包含新的 overcloud 证书。例如，请参阅 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml。
- public_vip.yaml - 将端点映射到特定的 ip，以便 dns 可以匹配。
- enable-internal-tls.yaml - 为内部端点启用 TLS。
- tls-everywhere-endpoints-dns.yaml - Configures TLS endpoints using DNS names.您可以查看此文件的内容来检查配置范围。
- HAProxy-internal-tls-certmonger.yaml - certmonger 将管理 haproxy 中的内部证书。
- inject-trust-anchor.yaml - 添加 root 证书颁发机构。只有证书依赖默认尚未被默认使用的通用集合的 CA 链时才需要；例如，当使用自签名时。

例如：

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
\
  -e /home/stack/inject-trust-anchor.yaml
...
```

**注意**

这些环境文件的示例可在以下位置找到：https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl。

### 17.4.3. 为使用与 IdM 相同的域的部署配置 overcloud 集成，并使用 IdM 生成的证书替换现有的公共端点证书

1. 确保 openstack overcloud deploy 命令（具有有效设置）中存在以下参数，然后重新运行部署命令：

- '--NTP-server' - 如果还没有设置，请指定 NTP 服务器以适应您的环境。IdM 服务器应该正在运行 ntp。
- cloud-names.yaml - 包含初始部署命令的 FQDN （而非 IP）。
- enable-tls.yaml - 包含新的 overcloud 证书。例如，请参阅 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml。
- public_vip.yaml - 将端点映射到特定的 ip，以便 dns 可以匹配。
- enable-internal-tls.yaml - 为内部端点启用 TLS。

- *tls-everywhere-endpoints-dns.yaml - Configures TLS endpoints using DNS names.您可以查看此文件的内容来检查配置范围。*

- *HAProxy-public-tls-certmonger.yaml - certmonger 将管理 haproxy 中的内部和公共证书。*

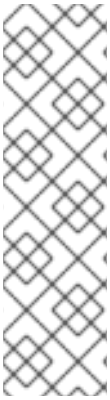
- *inject-trust-anchor.yaml - 添加 root 证书颁发机构。只有证书依赖默认尚未被默认使用的通用集合的 CA 链时才需要；例如，当使用自签名时。*

  *例如：*

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```

*注意*

*这些环境文件的示例可在 https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl 中找到。*



*注意*

*模板 enable-internal-tls.j2.yaml 在 overcloud deploy 命令中被引用为 enable-internal-tls.yaml。*

*另外，enable-tls.yaml 中的旧公共端点证书将被 certmonger 替换为 haproxy-public-tls-certmonger.yaml，但该文件仍必须在升级过程中被引用。*

### *17.4.4. 为使用 IdM 子域的部署配置 undercloud 集成*

*此流程解释了如何为使用 IdM 子域的部署配置 undercloud 集成。*

Red Hat OpenStack Platform 使用 novajoin 与 Red Hat Identity Management (IdM)集成，然后问题和管理加密证书。在此过程中，您将使用 IdM 注册 undercloud，生成令牌，在 undercloud 配置中启用令牌，然后重新运行 undercloud 和 overcloud 部署脚本。例如：

1. 安装 python-novajoin 以便与 IdM 集成：

```
[stack@undercloud-0 ~]$
```

2. 运行 novajoin 配置脚本，并提供您的 IdM 部署的配置详情。例如：

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.site1.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

此输出包括 IdM 的一次性密码(OTP)，这是您的部署的不同值。

3. 将 undercloud 配置为使用 novajoin，并为 DNS 和 NTP 添加 OTP、IdM IP 和 overcloud 域：

```
[stack@undercloud ~]$ vi undercloud.conf
…
[DEFAULT]
undercloud_ntp_servers=X.X.X.X
hieradata_override = /home/stack/hiera_override.yaml
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.site1.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = site1.lab.local
...
```

4. 将 undercloud 配置为使用 novajoin，并为 DNS 添加 OTP、IdM IP 和 overcloud 域：

```
[stack@undercloud-0 ~]$ vi hiera_override.yaml
nova::metadata::novajoin::api::ipa_domain: site1.lab.local
...
```

5. 在 undercloud 中安装 novajoin 服务：

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

6. **将 overcloud IP 地址添加到 DNS。您需要修改此示例以适合您的部署：**

**注意：检查 overcloud 的 network-environment.yaml，然后在各个网络范围内选择一个 VIP。**

```
[root@ipa ~]$ ipa dnsrecord-add site1.lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add site1.ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add site1.internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add site1.storage.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add site1.storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

7. **为每个端点创建一个 public_vip.yaml 映射。例如：**

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  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'}]
```

8. **为每个端点创建 extras.yaml 映射。例如：**

```
parameter_defaults:
  MakeHomeDir: True
  IdMNoNtpSetup: false
  IdMDomain: redhat.local
  DnsSearchDomains: ["site1.redhat.local","redhat.local"]
```

### 17.4.5. 为使用 IdM 子域的部署配置 undercloud 集成，并保留现有的公共端点证书

**此流程解释了如何为使用 IdM 子域的部署配置 undercloud 集成，并仍然保留现有的公共端点证书。**

1. **确保 openstack overcloud deploy 命令（具有有效设置）中存在以下参数，然后重新运行部署命令：**

-

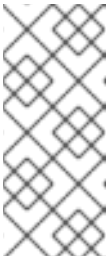
*' --NTP-server' - 如果还没有设置，请指定 NTP 服务器以适应您的环境。IdM 服务器应该正在运行 ntp。*

- *cloud-names.yaml - 包含初始部署命令的 FQDN （而非 IP）。*
- *enable-tls.yaml - 包含新的 overcloud 证书。例如，请参阅 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml。*
- *public_vip.yaml - 包含端点映射到特定的 ip，以便 dns 可以匹配。*
- *'Extras.yaml ' - Contains settings for pam to make home directorys on login, no ntp setup、base IdM 域以及 dns search for resolv.conf。*
- *enable-internal-tls.yaml - 为内部端点启用 TLS。*
- *tls-everywhere-endpoints-dns.yaml - Configures TLS endpoints using DNS names.您可以查看此文件的内容来检查配置范围。*
- *HAProxy-internal-tls-certmonger.yaml - certmonger 将管理 haproxy 中的内部证书。*
- *inject-trust-anchor.yaml - 添加 root 证书颁发机构。只有证书依赖默认尚未被默认使用的通用集合的 CA 链时才需要；例如，当使用自签名时。*

*例如：*

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
```

```
\
  -e /home/stack/inject-trust-anchor.yaml
...
```



**注意**

这些环境文件的示例可在以下位置找到：https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl。

### 17.4.6. 为使用 IdM 子域的部署配置 undercloud 集成，并使用 IdM 生成的证书替换现有的公共端点证书

此流程解释了如何使用 IdM 子域为部署配置 undercloud 集成，以及如何使用 IdM 生成的证书替换现有的公共端点证书。

1. 确保 openstack overcloud deploy 命令（具有有效设置）中存在以下参数，然后重新运行部署命令：

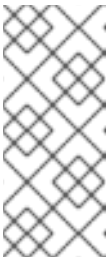
- ' --NTP-server' - 如果还没有设置，请指定 NTP 服务器以适应您的环境。IdM 服务器应该正在运行 ntp。
- cloud-names.yaml - 包含初始部署命令的 FQDN （而非 IP）。
- enable-tls.yaml - 包含新的 overcloud 证书。例如，请参阅 https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml。
- public_vip.yaml - 将端点映射到特定的 ip，以便 dns 可以匹配。
- 'Extras.yaml ' - Contains settings for pam to make home directorys on login, no ntp setup、base IdM 域以及 dns search for resolv.conf。
- enable-internal-tls.yaml - 为内部端点启用 TLS。
- tls-everywhere-endpoints-dns.yaml - Configures TLS endpoints using DNS names.您可以查看此文件的内容来检查配置范围。

- *HAProxy-public-tls-certmonger.yaml - certmonger 将管理 haproxy 中的内部和公共证书。*

- *inject-trust-anchor.yaml - 添加 root 证书颁发机构。只有证书依赖默认尚未被默认使用的通用集合的 CA 链时才需要；例如，当使用自签名时。*

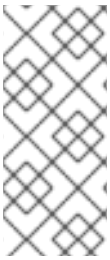
  *例如：*

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```



*注意*

*这些环境文件的示例可在以下位置找到：https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl。*



*注意*

*在本例中，模板 enable-internal-tls.j2.yaml 在 overcloud deploy 命令中被引用为 enable-internal-tls.yaml。另外，enable-tls.yaml 中的旧公共端点证书会使用 haproxy-public-tls-certmonger.yaml 替换 certmonger，但该文件仍必须在升级过程中被引用。*

### *17.5. 检查 TLS 加密*

*overcloud 重新部署完成后，检查是否现在使用 TLS 加密所有端点。在本例中，所有端点都配置为使用 https，表示它们正在使用 TLS 加密：*

```
+----------------------------------+-----------+-------------+---------------+---------+-----------+-----------------
-------------------------------------------+
| ID                               | Region    | Service Name | Service Type   | Enabled | Interface | URL
|
+----------------------------------+-----------+-------------+---------------+---------+-----------+-----------------
```

```
----------------------------------------------+
| 0fee4efdc4ae4310b6a139a25d9c0d9c | regionOne | neutron      | network        | True    | public    |
https://overcloud.lab.local:13696                |
| 220558ab1d2445139952425961a0c89a | regionOne | glance       | image          | True    | public    |
https://overcloud.lab.local:13292                |
| 24d966109ffa419da850da946f19c4ca | regionOne | placement    | placement      | True    | admin     |
https://overcloud.internalapi.lab.local:8778/placement        |
| 27ac9e0d22804ee5bd3cd8c0323db49c | regionOne | nova         | compute        | True    | internal  |
https://overcloud.internalapi.lab.local:8774/v2.1             |
| 31d376853bd241c2ba1a27912fc896c6 | regionOne | swift        | object-store   | True    | admin     |
https://overcloud.storage.lab.local:8080                 |
| 350806234c784332bfb8615e721057e3 | regionOne | nova         | compute        | True    | admin     |
https://overcloud.internalapi.lab.local:8774/v2.1             |
| 49c312f4db6748429d27c60164779302 | regionOne | keystone     | identity       | True    | public    |
https://overcloud.lab.local:13000                |
| 4e535265c35e486e97bb5a8bc77708b6 | regionOne | nova         | compute        | True    | public    |
https://overcloud.lab.local:13774/v2.1                |
| 5e93dd46b45f40fe8d91d3a5d6e847d3 | regionOne | keystone     | identity       | True    | admin     |
https://overcloud.ctlplane.lab.local:35357               |
| 6561984a90c742a988bf3d0acf80d1b6 | regionOne | swift        | object-store   | True    | public    |
https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s       |
| 76b8aad0bdda4313a02e4342e6a19fd6 | regionOne | placement    | placement      | True    | public
| https://overcloud.lab.local:13778/placement               |
| 96b004d5217c4d87a38cb780607bf9fb | regionOne | placement    | placement      | True    | internal  |
https://overcloud.internalapi.lab.local:8778/placement        |
| 98489b4b107f4da596262b712c3fe883 | regionOne | glance       | image          | True    | internal  |
https://overcloud.internalapi.lab.local:9292             |
| bb7ab36f30b14b549178ef06ec74ff84 | regionOne | glance       | image          | True    | admin     |
https://overcloud.internalapi.lab.local:9292             |
| c1547f7bf9a14e9e85eaaaeea26413b7 | regionOne | neutron      | network        | True    | admin     |
https://overcloud.internalapi.lab.local:9696             |
| ca66f499ec544f42838eb78a515d9f1e | regionOne | keystone     | identity       | True    | internal  |
https://overcloud.internalapi.lab.local:5000             |
| df0181358c07431390bc66822176281d | regionOne | swift        | object-store   | True    | internal  |
https://overcloud.storage.lab.local:8080/v1/AUTH_%(tenant_id)s |
| e420350ef856460991c3edbfbae917c1 | regionOne | neutron      | network        | True    | internal  |
https://overcloud.internalapi.lab.local:9696             |
+----------------------------------+-----------+--------------+----------------+---------+-----------+-----------------
----------------------------------------------+
```

# 第 18 章 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。创建一个 debug.yaml 环境文件来存储 debug 参数，并在 parameter_defaults 部分中设置 KeystoneDebug 参数：

```
parameter_defaults:
  KeystoneDebug: True
```

将 KeystoneDebug 参数设置为 True 后，/var/log/containers/keystone/keystone.log 标准 keystone 日志文件使用 DEBUG 级别日志更新。

有关 debug 参数的完整列表，请参阅 Overcloud 参数指南中的 "Debug 参数 "。

# 第 19 章 存储配置

本章概述了为 Overcloud 配置存储选项的方法。

重要

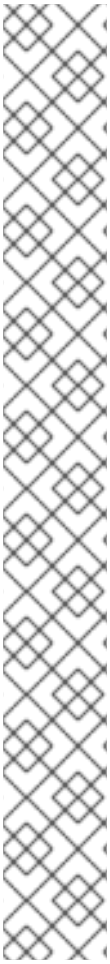
默认情况下，overcloud 使用 OpenStack Compute (nova)提供的本地临时存储，以及由 OpenStack Storage (cinder)提供的 LVM 块存储。但是，企业级 overcloud 不支持这些选项。反之，请使用本章中的一个存储选项。

## 19.1. 配置 NFS 存储

本节论述了如何将 overcloud 配置为使用 NFS 共享。安装和配置过程基于修改核心 heat 模板集合中的现有环境文件。

重要

红帽建议您使用经过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，因为它的功能与认证的存储后端和驱动程序相比受到限制。例如，通用 NFS 后端不支持卷加密和卷 multi-attach 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。

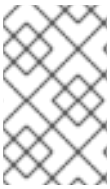


**注意**

有几个 director heat 参数控制 NFS 后端还是 NetApp NFS Block Storage 后端是否支持一个名为 NAS 的 NetApp 功能：

- CinderNetappNasSecureFileOperations
- CinderNetappNasSecureFilePermissions
- CinderNasSecureFileOperations
- CinderNasSecureFilePermissions

红帽不推荐启用此功能，因为它不会影响正常卷操作。director 会默认禁用这个功能，Red Hat OpenStack Platform 不支持它。



**注意**

对于块存储和计算服务，您必须使用 NFS 版本 4.0 或更高版本。

核心 heat 模板集合包含 /usr/share/openstack-tripleo-heat-templates/environments/ 中的一组环境文件。使用这些环境文件，您可以在由 director 创建的 overcloud 中创建自定义配置受支持的功能。这包括配置存储的环境文件。此文件位于 /usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml。

1. 将文件复制到 stack 用户的模板目录中：

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

2. 修改以下参数：

CinderEnableIscsiBackend

*启用 iSCSI 后端。设置为 false。*

**CinderEnableRbdBackend**

*启用 Ceph Storage 后端。设置为 false。*

**CinderEnableNfsBackend**

*启用 NFS 后端。设置为 true。*

**NovaEnableRbdBackend**

*为 Nova 临时存储启用 Ceph Storage。设置为 false。*

**GlanceBackend**

*定义用于 glance 的后端。设置为 file，以将基于文件的存储用于镜像。overcloud 为 glance 将这些文件保存在挂载的 NFS 共享中。*

**CinderNfsMountOptions**

*卷存储的 NFS 挂载选项。*

**CinderNfsServers**

*要为卷存储挂载的 NFS 共享。例如： 192.168.122.1:/export/cinder。*

**GlanceNfsEnabled**

*当 GlanceBackend 设置为 文件 时，GlanceNfsEnabled 将启用通过 NFS 存储在共享位置中的镜像，以便所有 Controller 节点都可以访问镜像。如果禁用，overcloud 会将镜像存储在 Controller 节点的文件系统中。设置为 true。*

**GlanceNfsShare**

*为镜像存储挂载的 NFS 共享。例如： 192.168.122.1:/export/glance。*

**GlanceNfsOptions**

*镜像存储的 NFS 挂载选项。*

*环境文件包含为 Red Hat OpenStack Platform Block Storage (cinder)和 Image (glance)服务配置不同的存储选项的参数。本例演示如何将 overcloud 配置为使用 NFS 共享。*

环境文件中的选项应类似于如下：

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

  CinderNfsMountOptions: rw,sync,context=system_u:object_r:cinder_var_lib_t:s0
  CinderNfsServers: 192.0.2.230:/cinder

  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
  GlanceNfsOptions: rw,sync,context=system_u:object_r:glance_var_lib_t:s0
```

这些参数作为 heat 模板集合的一部分集成。如示例代码所示，在示例代码中所示，为块存储和镜像服务创建两个 NFS 挂载点。

重要

在 GlanceNfsOptions 参数中包含 context=system_u:object_r:glance_var_lib_t:s0 选项，以允许镜像服务访问 /var/lib 目录。如果没有此 SELinux 内容，镜像服务无法写入到挂载点。

3. 在部署 overcloud 时包括 文件。

## 19.2. 配置 CEPH STORAGE

director 提供了两种主要方法，用于将 Red Hat Ceph Storage 集成到 Overcloud 中。

### 使用自己的 Ceph Storage 集群创建 Overcloud

director 能够在创建 Overcloud 期间创建 Ceph Storage 集群。director 会创建一组 Ceph Storage 节点，它使用 Ceph OSD 来存储数据。另外，director 在 Overcloud 的 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控制器节点的 Overcloud，Ceph Monitor 也变成高度可用的服务。有关更多信息，请参阅使用容器化 Red Hat Ceph 部署 Overcloud 指南。

### 将现有的 Ceph 存储集成到 Overcloud 中

如果您已经有一个现有的 Ceph Storage 集群，可以在 Overcloud 部署期间集成。这意味着您可

以在 Overcloud 配置之外管理并扩展集群。有关更多信息，请参阅将 Overcloud 与现有 Red Hat Ceph 集群集成 指南。

### 19.3. 使用外部 OBJECT STORAGE 集群

您可以通过禁用控制器节点上的默认 Object Storage 服务部署来重复使用外部 Object Storage (swift) 集群。这样做会禁用对象存储的代理和存储服务，并将 haproxy 和 keystone 配置为使用给定的外部 Swift 端点。

注意

外部 Object Storage (swift)集群上的用户帐户必须由手动管理。

您需要外部 Object Storage 集群的端点 IP 地址和外部 Object Storage proxy-server.conf 文件中的 authtoken 密码。您可以使用 openstack endpoint list 命令查找此信息。

使用外部 Swift 集群部署 director：

1. 创建包含以下内容的新文件 swift-external-params.yaml ：

   - 使用外部代理的 IP 地址和端口替换 EXTERNAL.IP:PORT。
   - 使用 SwiftPassword 行上的外部代理的 authtoken 密码替换 AUTHTOKEN。

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

2. 将此文件保存为 swift-external-params.yaml。

3. 使用这些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
```

### 19.4. 配置镜像导入方法和 SHARED STAGING 区域

OpenStack Image 服务(glance)的默认设置由安装 OpenStack 时使用的 Heat 模板决定。镜像服务 Heat 模板为 tht/puppet/services/glance-api.yaml。

可互操作镜像导入允许两个镜像导入方法：

- web-download 和
- Glance 直接.

web-download 方法可让您从 URL 导入镜像； glance-direct 方法可让您从本地卷导入镜像。

#### 19.4.1. 创建并部署 glance-settings.yaml 文件

您可以使用环境文件配置导入参数。这些参数覆盖在 Heat 模板中建立的默认值。示例环境内容提供了可互操作镜像导入的参数。

```
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"部分定义。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/13/html/advanced_overcloud_customization/storage_configuration

用于互操作性镜像导入的两个新参数定义了导入方法和共享 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 web-download 以外的其他方法时才需要这一行。

GlanceStagingNfsShare

配置 glance-direct 导入方法使用的 NFS 暂存区域。此空间可以在高可用性群集设置中的节点之间共享。需要将 GlanceNfsEnabled 设置为 true。

配置设置：

1. 创建一个名为 的新文件，如 glance-settings.yaml。此文件的内容应类似于上述示例。

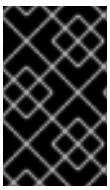
2. 使用 openstack overcloud deploy 命令将该文件添加到 OpenStack 环境中：

```
$ 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 创建环境文件 部分。

### 19.5. 为镜像服务配置 CINDER 后端

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数量为 10。

流程

1. 要将 cinder 配置为镜像服务后端，请在环境文件中添加以下内容：

```
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，或者 cinder_store_ 参数的任何自定义值，请将 ExtraConfig 设置添加到 parameter_defaults 中并传递自定义值：

```
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')}"
```

### 19.6. 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 Compute 环境文件中添加 max_disk_devices_to_attach 参数。以下示例演示了如何将 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 磁盘总线仅限于 4 个附加的设备。
- 在带有活跃实例的 Compute 节点上更改 max_disk_devices_to_attach 时，如果最大数量低于已连接到实例的设备数，则可能会导致重建失败。例如，如果实例 A 关联了 26 个设备，并且您将 max_disk_devices_to_attach 更改为 20，则重建实例 A 的请求将失败。
-

在冷迁移过程中，配置的最大存储设备数只对您要迁移的实例实施。移动之前不会检查目的地。这意味着，如果 Compute 节点 A 具有 26 个附加磁盘设备，并且 Compute 节点 B 配置最多 20 个附加磁盘设备，则带有 26 个连接的实例的冷迁移（从 Compute 节点 A 到 Compute 节点 B）的冷迁移会成功。但是，后续请求在 Compute 节点 B 中重建实例会失败，因为已经附加了 26 个设备，超过配置的最大值 20。

- 在 shelved 已卸载的实例中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的性能。您应该根据您的环境支持的边界调整最大数量。
- 具有机器类型 Q35 的实例可以最多附加 500 个磁盘设备。

## 19.7. 配置第三方存储

director 包括几个环境文件，以帮助配置第三方存储供应商。这包括：

### Dell EMC Storage Center

为 Block Storage (cinder)服务部署单个 Dell EMC Storage Center 后端。

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

有关完整配置信息，请参阅 Dell Storage Center 后端指南。

### Dell EMC PS 系列

为 Block Storage (cinder)服务部署单个 Dell EMC PS 系列后端。

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

有关完整配置信息，请参阅 Dell EMC PS 系列后端指南。

### NetApp Block Storage

*将 NetApp 存储设备部署为 Block Storage (cinder)服务的后端。*

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

*有关完整配置信息，请参阅 NetApp Block Storage 后端指南。*

# 第 20 章 安全增强

以下小节提供了增强 overcloud 安全性的一些建议。

## 20.1. 管理 OVERCLOUD 防火墙

每个核心 OpenStack 平台服务都在其相应的可组合服务模板中包含防火墙规则。这会为每个 overcloud 节点自动创建一组默认防火墙规则。

overcloud Heat 模板包含一组有助于其他防火墙管理的参数：

**ManageFirewall**

定义是否自动管理防火墙规则。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，则设置为 false。默认值是 true。

**PurgeFirewallRules**

定义在配置新防火墙规则前是否清除默认 Linux 防火墙规则。默认值为 false。

如果 ManageFirewall 设为 true，则可以在部署时创建额外的防火墙规则。使用配置 hook 设置 tripleo::firewall::firewall_rules hieradata （请参阅 overcloud 环境文件中的 第 4.5 节 “Puppet：为角色自定义 Hieradata”）。此 hieradata 是一个哈希，包含防火墙规则名称及其对应的参数作为键，它们都是可选的：

**port**

与该规则关联的端口。

**dport**

与该规则关联的目的地端口。

**SPORT**

与该规则关联的源端口。

**proto**

与该规则关联的协议。默认为 tcp。

**action**

与该规则关联的操作策略。默认为 接受。

跳过

要跳至的链。如果存在，它会覆盖 操作。

state

与该规则关联的状态阵列。默认为 ['NEW']。

source

与该规则关联的源 IP 地址。

iniface

与该规则关联的网络接口。

链

与该规则关联的链。默认为 INPUT。

目的地

与该规则关联的目标 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
```

这通过 ExtraConfig 将两个额外的防火墙规则应用于所有节点。

**注意**

每个规则名称都会成为对应 iptables 规则的注释。另请注意，每个规则名称都以三位前缀开头，以帮助 Puppet 在最终 iptables 文件中定义的所有规则。默认的 OpenStack Platform 规则使用 000 到 200 范围中的前缀。

## 20.2. 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 字符串来降低未授权用户了解您的网络设备的风险。

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 公共。

**snmp::ro_community6**

IPv6 只读 SNMP 社区字符串.默认值为 公共。

**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 带有 IPv6 的守护进程的网络。这个值可以是字符串或数组。默认值为 ::1/128。

**tripleo::profile::base::snmp::snmpd_config**

要添加到 snmpd.conf 文件中的行数组，作为安全 valve。默认值为 []。有关所有可用选项，请参阅 SNMP Configuration File 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。

基于 SNMP 视图的访问控制设置(VACM)

**snmp::com2sec**

IPv4 安全名称。

**snmp::com2sec6**

IPv6 安全名称。

例如：

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: mysecurestring
    snmp::com2sec6: 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 man page。

## 20.3. 更改 HAPROXY 的 SSL/TLS CIPHER 和 RULES

如果在 overcloud 中启用了 SSL/TLS （请参阅 第 15 章 在 Overcloud 公共端点中启用 SSL/TLS），您可能想加强与 HAProxy 配置一起使用的 SSL/TLS 密码和规则。这有助于避免 SSL/TLS 漏洞，如 POODLE 漏洞。

在 overcloud 的环境文件中，使用 ExtraConfig hook 设置以下 hieradata：

**tripleo::haproxy::ssl_cipher_suite**

HAProxy 中使用的密码套件。

**tripleo::haproxy::ssl_options**

*HAProxy 中使用的 SSL/TLS 规则。*

*例如，您可能需要使用下列密码和规则：*

- *密码：ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS*

- *规则：no-sslv3 no-tls-tickets*

*使用以下内容创建环境文件：*

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

*注意*

*cipher 集合是一个连续行。*

*在 overcloud 创建中包含此环境文件。*

## 20.4. 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。NeutronOVSFirewallDriver 参数允许您指定要使用的防火墙驱动程序：

- iptables_hybrid - 将 neutron 配置为使用基于 iptables/hybrid 的实施。
- openvswitch - 将 neutron 配置为使用 OVS 防火墙流型驱动程序。

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的处理。使用 iptables 时，VRRP 流量被拒绝，且您必须在 VRRP 流量的安全组规则中启用 VRRP 才能访问端点。使用 OVS 时，所有端口共享相同的 OpenFlow 上下文，并且每个端口无法单独处理多播流量。由于安全组不应用到所有端口（例如，路由器上的端口），OVS 会使用 NORMAL 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。

在 network-environment.yaml 文件中配置 NeutronOVSFirewallDriver 参数：

```
NeutronOVSFirewallDriver: openvswitch
```

- NeutronOVSFirewallDriver ：配置实施安全组时要使用的防火墙驱动程序名称。可能的值可能取决于您的系统配置。示例包括： noop、openvswitch、iptables_hybrid.默认值为空字符串，即 iptables_hybrid。

## 20.5. 使用 SECURE ROOT 用户访问权限

overcloud 镜像自动包含 root 用户的强化安全性。例如，每个部署的 overcloud 节点都会自动禁用对 root 用户的直接 SSH 访问。您仍然可以通过以下方法访问 overcloud 节点上的 root 用户：

1. 登录 undercloud 节点的 stack 用户。

2. 每个 overcloud 节点都有一个 heat-admin 用户帐户。此用户帐户包含 undercloud 的公共 SSH 密钥，它在没有从 undercloud 到 overcloud 节点的密码的情况下提供 SSH 访问。在 undercloud 节点上，使用 heat-admin 用户通过 SSH 登录选定的 overcloud 节点。

3. 使用 sudo -i 切换到 root 用户。

*减少根用户安全性*

有些情况可能需要直接 SSH 访问 root 用户。在这种情况下，您可以减少每个 overcloud 节点的 root 用户的 SSH 限制。

*警告*

此方法仅用于调试目的。不建议在生产环境中使用。

该方法使用第一个引导配置 hook （请参阅 *第 4.1 节 “首次启动：自定义第一个引导配置”*）。将以下内容放在环境文件中：

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/firstboot/userdata_root_password.yaml

parameter_defaults:
  NodeRootPassword: "p@55w0rd!"
```

注意以下几点：

- OS::TripleO::NodeUserData 资源引用在首次引导 cloud-init 阶段期间配置 root 用户的模板。

- NodeRootPassword 参数设置 root 用户的密码。将此参数的值更改为您所需的密码。请注意，环境文件以纯文本字符串形式包含密码，这被视为安全风险。

*在创建 overcloud 时，请使用 openstack overcloud deploy 命令包含此环境文件。*

第 21 章 配置网络插件

director 包括有助于配置第三方网络插件的环境文件：

## 21.1. 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。

1. 将环境文件复制到模板 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>:& lt;vfab_id& gt; tuples，用于指定 physical_network 名称和对应的 vfab ID。(comma_delimited_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 name 的前缀字符串。（字符串）***
- ***NeutronFujitsuCfabSaveConfig - 确定是否保存配置。（布尔值）***

4. ***要将模板应用到您的部署中，请在 openstack overcloud deploy 命令中包含 环境文件。例如：***

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS] ...
```

## *21.2. FUJITSU FOS SWITCH*

***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。***

1. ***将环境文件复制到模板 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_delimited_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 连接的超时周期。（数字）***
- ***NeutronFujitsuFosswUdpDestPort - FOS 交换机上的 VXLAN UDP 目的地的端口号。（数字）***
- ***NeutronFujitsuFosswOvsdbVlanidRangeMin - 用于绑定 VNI 和物理端口范围内的最小 VLAN ID。（数字）***
- ***NeutronFujitsuFosswOvsdbPort - FOS 交换机上 OVSDB 服务器的端口号。（数字）***

4. ***要将模板应用到您的部署中，请在 openstack overcloud deploy 命令中包含 环境文件。例如：***

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

# 第 22 章 配置身份

director 包含有助于配置 Identity Service (keystone)设置的参数：

## 22.1. 区域名称

默认情况下，overcloud 的区域将命名为 regionOne。您可以通过添加 KeystoneRegion 条目来更改您的环境文件。这个设置无法在部署后更改：

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

# 第 23 章 其他配置

## 23.1. 在 OVERCLOUD 节点上配置内核

OpenStack Platform director 包含在 overcloud 节点上配置内核的参数。

**ExtraKernelModules**

要加载的内核模块。模块名称被列为带有空值的 hash 键：

```
ExtraKernelModules:
  <MODULE_NAME>: {}
```

**ExtraKernelPackages**

在从 ExtraKernelModules 加载内核模块前要安装的与内核相关的软件包。软件包名称被列为带有空值的散列键。

```
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>
```

这个示例在环境文件中显示这些参数的语法：

```
parameter_defaults:
  ExtraKernelModules:
    iscsi_target_mod: {}
  ExtraKernelPackages:
    iscsi-initiator-utils: {}
  ExtraSysctlSettings:
    dev.scsi.logging_level:
      value: 1
```

## 23.2. 配置外部负载平衡

Overcloud 使用多个 Controller 作为高可用性集群，确保 OpenStack 服务的最大运行性能。另外，集群为访问 OpenStack 服务提供负载均衡，它会平均分配流量到 Controller 节点，并减少每个节点的服务器过载。也可以使用外部负载均衡器来执行此分发。例如，组织可能会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。

有关配置外部负载均衡的更多信息，请参阅 Overcloud 的专用外部负载平衡 以 获得完整说明。

### 23.3. 配置 IPV6 网络

作为默认情况，Overcloud 使用 Internet Protocol 版本 4 (IPv4)来配置服务端点。但是，Overcloud 还支持互联网协议版本 6 (IPv6)端点，对于支持 IPv6 基础架构的组织有用。director 包括一组环境文件，可帮助创建基于 IPv6 的 Overcloud。

有关在 Overcloud 中配置 IPv6 的详情，请参考 Overcloud 的专用 IPv6 网络 以获得完整说明。