Red Hat OpenStack Platform 16.2

过渡到容器化服务

使用 OpenStack Platform 容器化服务的基本指南

Last Updated: 2024-05-21

Red Hat OpenStack Platform 16.2 过渡到容器化服务

使用 OpenStack Platform 容器化服务的基本指南

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

本指南提供了一些基本信息，用于帮助用户熟悉在容器中运行的 OpenStack Platform 服务。

目录

使开源包含更多

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

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

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

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

第 1 章 简介

旧版 Red Hat OpenStack Platform 使用了通过 Systemd 管理的服务。但是，最新版本的 OpenStack Platform 现在使用容器来运行服务。有些管理员可能不知道容器化 OpenStack Platform 服务如何运作，因此本指南旨在帮助您了解 OpenStack Platform 容器镜像和容器化服务。这包括：

- 如何获取和修改容器镜像
- 如何在 overcloud 中管理容器化服务
- 了解容器与 Systemd 服务的不同

主要目标是帮助您了解容器化 OpenStack Platform 服务，以便从基于 Systemd 的环境转换到基于容器的环境。

1.1. 容器化服务和 KOLLA

每个主 Red Hat OpenStack Platform (RHOSP)服务都在容器中运行。这提供了一种将每个服务保留在独立于主机的隔离命名空间中的各个服务的方法。这有以下影响：

- 在部署过程中，RHOSP 从红帽客户门户网站拉取并运行容器镜像。
- **podman** 命令运行管理功能，如启动和停止服务。
- 要升级容器，您必须拉取新的容器镜像，并将现有容器替换为更新的版本。

Red Hat OpenStack Platform 使用通过 **Kolla** 工具组构建和管理的一组容器。

第 2 章 获取和修改容器镜像

容器化 overcloud 需要访问具有所需容器镜像的 registry。本章介绍了如何准备 registry 和 undercloud 和 overcloud 配置，以便将容器镜像用于 Red Hat OpenStack Platform。

2.1. 准备容器镜像

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

注意

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

流程

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

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

此命令包括以下附加选项：

- **--local-push-destination**，在 undercloud 上设置 registry 作为存储容器镜像的位置。这意味着 director 从 Red Hat Container Catalog 拉取必要的镜像并将其推送到 undercloud 上的 registry 中。director 将该 registry 用作容器镜像源。如果直接从 Red Hat Container Catalog 拉取镜像，请忽略这个选项。
- **--output-env-file** 是环境文件名称。此文件的内容包括用于准备您的容器镜像的参数。在本例中，文件的名称是 **containers-prepare-parameter.yaml**。

注意

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

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

2.2. 容器镜像准备参数

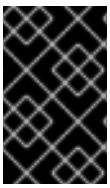
用于准备您的容器的默认文件 (**containers-prepare-parameter.yaml**) 包含 **ContainerImagePrepare** heat 参数。此参数定义一个用于准备一系列镜像的策略列表：

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

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

参数	描述
excludes	用于从策略中排除镜像名称的正则表达式列表。
includes	用于在策略中包含的正则表达式列表。至少一个镜像名称必须与现有镜像匹配。如果指定了 **includes**，将忽略所有 **excludes**。
modify_append_tag	要附加到目标镜像标签的字符串。例如，如果使用标签 16.2.3-5.161 拉取镜像并将 **modify_append_tag** 设置为 **-hotfix**，director 会将最终镜像标记为 16.2.3-5.161-hotfix。
modify_only_with_labels	过滤想要修改的镜像的镜像标签字典。如果镜像与定义的标签匹配，则 director 将该镜像包括在修改过程中。
modify_role	在上传期间但在将镜像推送到目标 registry 之前运行的 ansible 角色名称字符串。
modify_vars	要传递给 **modify_role** 的变量的字典。
push_destination	定义用于在上传过程中要将镜像推送到的 registry 的命名空间。 ● 如果设为 **true**，则使用主机名将 **push_destination** 设置为 undercloud registry 命名空间，这是建议的方法。 ● 如果设置为 **false**，则不会推送到本地 registry，节点直接从源拉取镜像。 ● 如果设置为自定义值，则 director 将镜像推送到外部本地 registry。 当直接从 Red Hat Container Catalog 拉取镜像时，如果在生产环境中将此参数设置为 **false**，则所有 overcloud 节点都将通过外部连接同时从 Red Hat Container Catalog 拉取镜像，这可能会导致出现带宽问题。仅使用 **false** 直接从托管容器镜像的 Red Hat Satellite Server 拉取。 如果 **push_destination** 参数设置为 **false** 或未定义，且远程 registry 需要身份验证，请将 **ContainerImageRegistryLogin** 参数设置为 **true**，并使用 **ContainerImageRegistryCredentials** 参数包含凭据。
pull_source	从中拉取原始容器镜像的源 registry 。

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



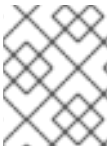
重要

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

set 参数接受一组 **key: value** 定义：

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

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



注意

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

2.3. 容器镜像标记准则

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

version

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

release

对应于版本流中特定容器镜像版本的发行版本。

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

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

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

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

```
parameter_defaults:
```

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

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

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

- 在设置 **tag** 时，director 始终会在安装和更新期间下载 **tag** 中设置的版本的最新容器镜像 **release**。
- 如果没有设置 **tag**，则 director 会结合使用 **tag_from_label** 的值和最新的主要版本。

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

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

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

- **tag_from_label** 的默认值为 **{version}-{release}**，对应于每个容器镜像的版本和发行版本元数据标签。例如，如果容器镜像的 **version** 设置为 16.2.3，**release** 设置为 5.161，生成的容器镜像标签为 16.2.3-5.161。
- **tag** 参数始终优先于 **tag_from_label** 参数。要使用 **tag_from_label**，在容器准备配置中省略 **tag** 参数。
- **tag** 和 **tag_from_label** 之间的一个关键区别是：director 仅基于主要或次要版本标签使用 **tag** 拉取镜像，Red Hat Container Registry 将这些标签链接到版本流中的最新镜像发行版本，而 director 使用 **tag_from_label** 对每个容器镜像执行元数据检查，以便 director 生成标签并拉取对应的镜像。

2.4. 从私有 REGISTRY 获取容器镜像

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

ContainerImageRegistryCredentials

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

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

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

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

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

重要

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

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

ContainerImageRegistryLogin

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

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

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

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

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

2.5. 分层镜像准备条目

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

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

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

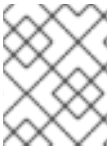
```
    tag:16.2
- push_destination: true
  includes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp-rhel8
    tag: 16.2.1-hotfix
```

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

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

2.6. 准备期间修改镜像

可在准备镜像期间修改镜像，然后立即使用修改的镜像部署 overcloud。



注意

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

修改镜像的情况包括：

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

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

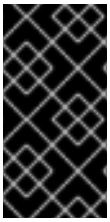
Ansible 角色 **tripleo-modify-image** 与所需的角色接口相一致，并提供修改用例必需的行为。使用 **ContainerImagePrepare** 参数中与修改相关的键控制修改：

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

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

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

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



重要

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

2.7. 更新容器镜像的现有软件包

注意

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

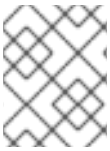
步骤

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

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

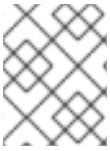
2.8. 将额外的 RPM 文件安装到容器镜像中

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



注意

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



注意

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

步骤

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

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

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

2.9. 通过自定义 DOCKERFILE 修改容器镜像

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

注意

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

步骤

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

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

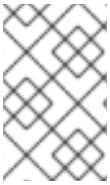
2. 以下示例显示了 **/home/stack/nova-custom/Dockerfile** 文件。运行任何 **USER** 根指令后，必须切换回原始镜像默认用户：

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

USER "root"

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

USER "nova"
```

2.10. 为容器镜像准备 SATELLITE 服务器

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

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



注意

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

步骤

1. 创建所有容器镜像的列表：

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{ .Name }}" | sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-dashboard-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/openshift" | grep ose-prometheus
```

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

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.6
registry.redhat.io/openshift4/ose-prometheus:v4.6
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.6
```

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

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

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

   该定制产品将会包含您的镜像。

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

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

5. 添加 Ceph Storage 4 容器镜像：

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```

注意

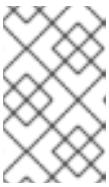
如果要安装 Ceph 仪表板，请在 **hammer repository create** 命令中包含 **--name rhceph-4-dashboard-rhel8** ：

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-dashboard-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-dashboard-rhel8
```

6. 同步容器镜像：

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

等待 Satellite 服务器完成同步。



注意

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

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

8. 检查 **base** 镜像的可用标签：

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

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

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

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

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

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

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

    - **namespace** - Satellite 服务器上 registry 的 URL 和端口。Red Hat Satellite 上的默认 registry 端口为 443。

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

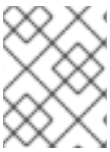
      - 如果您使用了内容视图，则前缀的结构为 **[组织]-[环境]-[内容视图]-[产品]-**。例如：**acme-production-myosp16-osp_containers-**。

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

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

以下示例环境文件包含特定于 Satellite 的参数：

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:443
      neutron_driver: null
      tag: '16.2'
      ...
```



注意

要使用存储在 Red Hat Satellite Server 上的特定容器镜像版本，请将 **标签** 键值对设置为 **set** 字典中的特定版本。例如，要使用 16.2.2 镜像流，请在 **set** 字典中设置 **tag: 16.2.2**。

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

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

第 3 章 使用容器安装 UNDERCLOUD

本章介绍了如何创建基于容器的 undercloud 并进行更新的信息。

3.1. 配置 DIRECTOR

director 安装过程需要 **undercloud.conf** 配置文件中的某些设置，director 从 **stack** 用户的主目录中读取该文件。完成以下步骤，复制默认模板作为基础进行配置。

步骤

1. 将默认模板复制到 **stack** 用户的主目录：

```
[stack@director ~]$ cp \
  /usr/share/python-tripleoclient/undercloud.conf.sample \
  ~/undercloud.conf
```

2. 编辑 **undercloud.conf** 文件。这个文件包含用于配置 undercloud 的设置。如果忽略或注释掉某个参数，undercloud 安装将使用默认值。

3.2. DIRECTOR 配置参数

以下列表包含用于配置 **undercloud.conf** 文件的参数的相关信息。将所有参数保留在相关部分内以避免出错。

重要

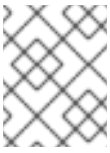
您至少必须将 **container_images_file** 参数设置为包含容器镜像配置的环境文件。如果没有将此参数正确设置为适当的文件，则 director 无法从 **ContainerImagePrepare** 参数获取容器镜像规则集，也无法从 **ContainerImageRegistryCredentials** 参数获取容器 registry 身份验证详情。

默认值

以下参数会在 **undercloud.conf** 文件的 **[DEFAULT]** 部分中进行定义：

additional_architectures

overcloud 支持的附加（内核）架构的列表。目前，除了默认的 **x86_64** 架构外，overcloud 还支持 **ppc64le** 架构。



注意

当启用对 ppc64le 的支持时，还必须将 **ipxe_enabled** 设置为 **False**。有关使用多个 CPU 架构配置 undercloud 的更多信息，请参阅 配置多个 CPU 架构 overcloud。

certificate_generation_ca

为所请求证书签名的 CA 的 **certmonger** 别名。仅在设置了 **generate_service_certificate** 参数的情况下使用此选项。如果您选择 **local** CA，certmonger 会将本地 CA 证书提取到 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** 并将该证书添加到信任链中。

clean_nodes

确定是否在部署之间和内省之后擦除硬盘。

cleanup

删除临时文件。把它设置为 **False** 以保留部署期间使用的临时文件。如果出现错误，临时文件可帮助您调试部署。

container_cli

用于容器管理的 CLI 工具。将此参数设置为 **podman**。Red Hat Enterprise Linux 8.4 仅支持 **podman**。

container_healthcheck_disabled

禁用容器化服务运行状况检查。红帽建议您启用运行状况检查，并将此选项设置为 **false**。

container_images_file

含有容器镜像信息的 Heat 环境文件。此文件可能包含以下条目：

- 所有需要的容器镜像的参数
- **ContainerImagePrepare** 参数（用于推动必要的镜像准备）。通常，含有此参数的文件被命名为 **containers-prepare-parameter.yaml**。

container_insecure_registries

供 **podman** 使用的不安全 registry 列表。如果您想从其他来源（如私有容器 registry）拉取镜像，则使用此参数。在大多数情况下，如果在 Satellite 中注册了 undercloud，**podman** 就有从 Red Hat Container Catalog 或 Satellite Server 拉取容器镜像的证书。

container_registry_mirror

配置的 **podman** 使用的可选 **registry-mirror**。

custom_env_files

要添加到 undercloud 安装中的其他环境文件。

deployment_user

安装 undercloud 的用户。如果此参数保留为不设置，则使用当前的默认用户 **stack**。

discovery_default_driver

为自动注册的节点设置默认驱动程序。需要启用 **enable_node_discovery** 参数，且必须在 **enabled_drivers** 列表中包含驱动程序。

enable_ironic; enable_ironic_inspector; enable_mistral; enable_nova; enable_tempest; enable_validations; enable_zaqar

定义要为 director 启用的核心服务。保留这些参数设为 **true**。

enable_node_discovery

自动注册通过 PXE 引导内省虚拟内存盘 (ramdisk) 的所有未知节点。新节点使用 **fake** 作为默认驱动程序，但您可以设置 **discovery_default_driver** 覆盖它。您也可以使用内省规则为新注册的节点指定驱动程序信息。

enable_novajoin

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

enable_routed_networks

定义是否支持路由的 control plane 网络。

enable_swift_encryption

定义是否启用 Swift 加密。

enable_telemetry

定义是否在 undercloud 中安装 OpenStack Telemetry 服务（gnocchi、aodh、panko）。如果您想自动安装和配置 telemetry 服务，则将 **enable_telemetry** 参数设为 **true**。默认值为 **false**，即在 undercloud 中禁用 telemetry。如果使用要利用指标数据的其他产品，如 Red Hat CloudForms，则需

要这个参数。

警告

每个组件都不支持 RBAC。Alarming 服务(aodh)和 Gnocchi 不考虑安全 RBAC 规则。

enabled_hardware_types

要为 undercloud 启用的硬件类型的列表。

generate_service_certificate

定义 undercloud 安装期间是否生成 SSL/TLS 证书，此证书用于 **undercloud_service_certificate** 参数。undercloud 安装会保存生成的证书 **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem**。**certificate_generation_ca** 参数中定义的 CA 将为此证书签名。

heat_container_image

要使用的 heat 容器镜像的 URL。请保留不设置。

heat_native

使用 **heat-all** 运行基于主机的 undercloud 配置。请保留为 **true**。

hieradata_override

在 director 上配置 Puppet hieradata 的 **hieradata** 覆盖文件的路径，为 **undercloud.conf** 参数外的服务提供自定义配置。如果设置此参数，undercloud 安装会将此文件复制到 **/etc/puppet/hieradata** 目录并将其设为层次结构中的第一个文件。有关使用此功能的更多信息，请参阅在 undercloud 上配置 hieradata。

inspection_extras

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

inspection_interface

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

inspection_runbench

在节点内省过程中运行一组基准测试。将此参数设为 **true** 以启用基准测试。如果您需要在检查注册节点的硬件时执行基准数据分析操作，则需要使用这个参数。

ipa_otp

定义在 IPA 服务器注册 undercloud 节点使用的一次性密码。启用 **enable_novajoin** 之后需要提供此密码。

ipv6_address_mode

undercloud 置备网络的 IPv6 地址配置模式。以下列表包含这个参数的可能值：

- dhcpv6-stateless - 使用路由器公告 (RA) 的地址配置以及使用 DHCPv6 的可选信息。
- DHCPv6-stateful - 地址配置和使用 DHCPv6 的可选信息。

ipxe_enabled

定义使用 iPXE 还是标准的 PXE。默认为 **true**，其启用 iPXE。将此参数设置为 **false** 以使用标准 PXE。对于 PowerPC 部署，或混合了 PowerPC 和 x86 的部署，请将此值设置为 **false**。

local_interface

指定 director Provisioning NIC 的接口。这也是该 director 用于 DHCP 和 PXE 引导服务的设备。把这个项的值改为您选择的设备。使用 **ip addr** 命令可以查看连接了哪些设备。以下是一个 **ip addr** 命令的结果输出示例：

```
2: em0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic em0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

在这个例子中，External NIC 使用 **em0**，Provisioning NIC 使用 **em1**（当前没有被配置）。在这种情况下，将 **local_interface** 设置为 **em1**。配置脚本会把这个接口附加到一个自定义的网桥（由 **inspection_interface** 参数定义）上。

local_ip

为 director Provisioning NIC 定义的 IP 地址。这也是 director 用于 DHCP 和 PXE 引导服务的 IP 地址。除非 Provisioning 网络需要使用其他子网（如该 IP 地址与环境中的现有 IP 地址或子网冲突）保留默认值 **192.168.24.1/24**。
对于 IPv6，本地 IP 地址前缀长度必须是 **/64**，以支持有状态和无状态连接。

local_mtu

要用于 **local_interface** 的最大传输单元 (MTU)。对于 undercloud 不要超过 1500。

local_subnet

要用于 PXE 引导和 DHCP 接口的本地子网。**local_ip** 地址应该属于这个子网。默认值为 **ctlplane-subnet**。

net_config_override

网络配置覆盖模板的路径。如果设置此参数，undercloud 将使用 JSON 或 YAML 格式的模板以使用 **os-net-config** 配置网络，并忽略 **undercloud.conf** 中设置的网络参数。当您要配置绑定或向接口添加一个选项时，请使用此参数。有关自定义 undercloud 网络接口的更多信息，请参阅配置 undercloud 网络接口。

networks_file

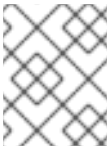
覆盖用于 **heat** 的网络文件。

output_dir

输出状态目录、处理的 heat 模板和 Ansible 部署文件。

overcloud_domain_name

要在部署 overcloud 时使用的 DNS 域名。



注意

配置 overcloud 时，必须将 **CloudDomain** 参数设置为匹配的值。配置 overcloud 时，在环境文件中设置此参数。

roles_file

要用来覆盖用于 undercloud 安装的默认角色文件的角色文件。强烈建议您将此参数保留为不设置，以便 director 安装使用默认的角色文件。

scheduler_max_attempts

调度程序尝试部署实例的次数上限。此值必须大于或等于您期望一次部署的裸机节点数，以避免调度时的潜在争用情形。

service_principal

使用该证书的服务的 Kerberos 主体。仅在您的 CA 需要 Kerberos 主体（如在 FreeIPA 中）时使用此参数。

subnets

用于置备和内省的路由网络子网的列表。默认值仅包括 **ctlplane-subnet** 子网。如需更多信息，请参阅 子网。

templates

要覆盖的 heat 模板文件。

undercloud_admin_host

通过 SSL/TLS 为 director Admin API 端点定义的 IP 地址或主机名。director 配置将 IP 地址作为路由的 IP 地址附加到 director 软件网桥，其使用 **/32** 子网掩码。
如果 **undercloud_admin_host** 不在与 **local_ip** 相同的 IP 网络中，您必须将 **ControlVirtualInterface** 参数设置为您希望 undercloud 上的 admin API 侦听的接口。默认情况下，admin API 侦听 **br-ctlplane** 接口。在自定义环境文件中设置 **ControlVirtualInterface** 参数，并通过配置 **custom_env_files** 参数在 **undercloud.conf** 文件中包含自定义环境文件。

有关自定义 undercloud 网络接口的详情，请参考配置 undercloud 网络接口。

undercloud_debug

把 undercloud 服务的日志级别设置为 **DEBUG**。将此值设置为 **true** 以启用 **DEBUG** 日志级别。

undercloud_enable_selinux

在部署期间启用或禁用 SELinux。除非调试问题，否则强烈建议保留此值设为 **true**。

undercloud_hostname

定义 undercloud 的完全限定主机名。如果设置，undercloud 安装将配置所有系统主机名设置。如果保留未设置，undercloud 将使用当前的主机名，但您必须相应地配置所有主机名设置。

undercloud_log_file

用于存储 undercloud 安装和升级日志的日志文件的路径。默认情况下，日志文件是主目录中的 **install-undercloud.log**。例如，**/home/stack/install-undercloud.log**。

undercloud_nameservers

用于 undercloud 主机名解析的 DNS 名称服务器列表。

undercloud_ntp_servers

帮助同步 undercloud 日期和时间的网络时间协议服务器列表。

undercloud_public_host

通过 SSL/TLS 为 director Public API 端点定义的 IP 地址或主机名。director 配置将 IP 地址作为路由的 IP 地址附加到 director 软件网桥，其使用 **/32** 子网掩码。
如果 **undercloud_public_host** 不在与 **local_ip** 相同的 IP 网络中，您必须将 **PublicVirtualInterface** 参数设置为您希望 undercloud 上公共 API 侦听的公共接口。默认情况下，公共 API 侦听 **br-ctlplane** 接口。在自定义环境文件中设置 **PublicVirtualInterface** 参数，并通过配置 **custom_env_files** 参数在 **undercloud.conf** 文件中包含自定义环境文件。

有关自定义 undercloud 网络接口的详情，请参考配置 undercloud 网络接口。

undercloud_service_certificate

用于 OpenStack SSL/TLS 通信的证书的位置和文件名。理想的情况是从一个信任的证书认证机构获得这个证书。否则，生成自己的自签名证书。

undercloud_timezone

undercloud 的主机时区。如果未指定时区，director 将使用现有时区配置。

undercloud_update_packages

定义是否在安装 undercloud 期间更新软件包。

子网

每个置备子网在 **undercloud.conf** 文件中都有一个对应的同名部分。例如，要创建称为 **ctlplane-subnet** 的子网，在 **undercloud.conf** 文件中使用以下示例：

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

您可以根据自身环境所需来指定相应数量的置备网络。

重要

director 在创建子网后无法更改子网的 IP 地址。

cidr

director 用来管理 overcloud 实例的网络。这是 undercloud **neutron** 服务管理的 Provisioning 网络。保留其默认值 **192.168.24.0/24**，除非您需要 Provisioning 网络使用其他子网。

masquerade

定义是否伪装 **cidr** 中定义的用于外部访问的网络。这为 Provisioning 网络提供了网络地址转换 (NAT)，使 Provisioning 网络能够通过 director 进行外部访问。

注意

director 配置还使用相关 **sysctl** 内核参数自动启用 IP 转发。

dhcp_start; dhcp_end

overcloud 节点 DHCP 分配范围的开始值和终止值。确保此范围包含足够的 IP 地址，以分配给您的节点。如果没有为子网指定，director 通过删除为 **local_ip**、**gateway**、**undercloud_admin_host**、undercloud_public_host、**undercloud_public_host** 以及 **inspection_iprange** 参数的值来确定分配池。
您可以通过指定启动和结束地址对列表，为 undercloud control plane 子网配置非持续分配池。另外，您可以使用 **dhcp_exclude** 选项排除 IP 地址范围中的 IP 地址。例如，以下配置同时创建分配池 **172.20.0.100-172.20.0.150** 和 **172.20.0.200-172.20.0.250** ：

选项 1

```
dhcp_start = 172.20.0.100,172.20.0.200
dhcp_end = 172.20.0.150,172.20.0.250
```

选项 2

```
dhcp_start = 172.20.0.100
dhcp_end = 172.20.0.250
dhcp_exclude = 172.20.0.151-172.20.0.199
```

dhcp_exclude

DHCP 分配范围中排除的 IP 地址。例如，以下配置排除 IP 地址 **172.20.0.105** 和 IP 地址范围 **172.20.0.210-172.20.0.219** ：

```
dhcp_exclude = 172.20.0.105,172.20.0.210-172.20.0.219
```

dns_nameservers

特定于子网的 DNS 名称服务器。如果没有为子网定义名称服务器，子网将使用 **undercloud_nameservers** 参数中定义的名称服务器。

gateway

overcloud 实例的网关。它是 undercloud 主机，会把网络流量转发到外部网络。保留其默认值 **192.168.24.1**，除非您需要 director 使用其他 IP 地址，或想直接使用外部网关。

host_routes

此网络上 overcloud 实例的 Neutron 管理的子网的主机路由。这也为 undercloud 上的 **local_subnet** 配置主机路由。

inspection_iprange

在检查过程中，此网络上的节点要使用的临时 IP 范围。这个范围不得与 **dhcp_start** 和 **dhcp_ end** 定义的范围重叠，但必须位于同一个 IP 子网中。

修改这些参数的值以符合您的配置。完成后，请保存文件。

3.3. 安装 DIRECTOR

完成以下步骤以安装 director 并执行一些基本安装后任务。

步骤

1. 运行以下命令，以在 undercloud 上安装 director：

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

此命令会启动 director 配置脚本。director 安装附加软件包，根据 **undercloud.conf** 中的配置配置其服务，并启动所有 RHOSP 服务容器。这个脚本会需要一些时间来完成。

此脚本会生成两个文件：

- **undercloud-passwords.conf** - director 服务的所有密码列表。
- **stackrc** - 用来访问 director 命令行工具的一组初始变量。

2. 确认 RHOSP 服务容器正在运行：

```
[stack@director ~]$ sudo podman ps -a --format "{{.Names}} {{.Status}}"
```

以下命令输出显示 RHOSP 服务容器正在运行(**Up**)：

```
memcached Up 3 hours (healthy)
haproxy Up 3 hours
rabbitmq Up 3 hours (healthy)
mysql Up 3 hours (healthy)
iscsid Up 3 hours (healthy)
keystone Up 3 hours (healthy)
keystone_cron Up 3 hours (healthy)
neutron_api Up 3 hours (healthy)
logrotate_crond Up 3 hours (healthy)
neutron_dhcp Up 3 hours (healthy)
neutron_l3_agent Up 3 hours (healthy)
neutron_ovs_agent Up 3 hours (healthy)
ironic_api Up 3 hours (healthy)
ironic_conductor Up 3 hours (healthy)
ironic_neutron_agent Up 3 hours (healthy)
ironic_pxe_tftp Up 3 hours (healthy)
ironic_pxe_http Up 3 hours (unhealthy)
ironic_inspector Up 3 hours (healthy)
ironic_inspector_dnsmasq Up 3 hours (healthy)
neutron-dnsmasq-qdhcp-30d628e6-45e6-499d-8003-28c0bc066487 Up 3 hours
...
```

3. 运行以下命令初始化 **stack** 用户来使用命令行工具：

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

提示现在指示 OpenStack 命令对 undercloud 进行验证并执行；

```
(undercloud) [stack@director ~]$
```

director 的安装已完成。您现在可以使用 director 命令行工具了。

3.4. 对容器化 UNDERCLOUD 执行次要更新

director 提供更新 undercloud 节点上主软件包的命令。使用 director 在 RHOSP 环境当前版本中执行次要更新。

流程

1. 在 undercloud 节点上，以 **stack** 用户身份登录。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 使用 **dnf update** 命令更新 director 主软件包：

```
$ sudo dnf update -y python3-tripleoclient* tripleo-ansible ansible
```

4. 使用 **openstack undercloud upgrade** 命令更新 undercloud 环境：

```
$ openstack undercloud upgrade
```

5. 等待 undercloud 更新过程完成。
6. 重启 undercloud 以更新操作系统的内核和其他系统软件包：

```
$ sudo reboot
```

7. 稍等片刻，直到节点启动。

第 4 章 使用容器部署和更新 OVERCLOUD

本章介绍了如何创建基于容器的 overcloud 并更新它。

4.1. 部署 OVERCLOUD

此流程演示了如何以最低配置部署 overcloud。结果将是基本的双节点 overcloud (1 个 Controller 节点，1 个 Compute 节点)。

流程

1. Source **stackrc** 文件：

```
$ source ~/stackrc
```

2. 运行 **deploy** 命令，并包含包含 overcloud 镜像位置的文件（通常为 **overcloud_images.yaml**）：

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/overcloud_images.yaml \
  --ntp-server pool.ntp.org
```

3. 等待 overcloud 完成部署。

4.2. 更新 OVERCLOUD

有关更新容器化 overcloud 的详情，请参考 Keeping Red Hat OpenStack Platform Updated 指南。

第 5 章 使用容器化服务

本章提供了一些管理容器的命令示例以及如何对 OpenStack Platform 容器进行故障排除

5.1. 管理容器化服务

Red Hat OpenStack (RHOSP) 平台在 undercloud 和 overcloud 节点上的容器中运行服务。在某些情况下，您可能需要控制主机上的单个服务。本节介绍了可在节点上运行的用于管理容器化服务的一些常见命令。

列出容器和镜像

要列出运行中的容器，请运行以下命令：

```
$ sudo podman ps
```

要在命令输出中包括停止的或失败的容器，将 **--all** 选项添加到命令中：

```
$ sudo podman ps --all
```

要列出容器镜像，请运行以下命令：

```
$ sudo podman images
```

检查容器属性

要查看容器或容器镜像的属性，请使用 **podman inspect** 命令。例如，要检查 **keystone** 容器，请运行以下命令：

```
$ sudo podman inspect keystone
```

使用 Systemd 服务管理容器

早期版本的 OpenStack Platform 使用 Docker 及其守护进程管理容器。在 OpenStack Platform 16 中，Systemd 服务接口管理容器的生命周期。每个容器都是一个服务，您运行 Systemd 命令，为每个容器执行特定操作。

注意

不建议使用 Podman CLI 停止、启动和重启容器，因为 Systemd 会应用重启策略。请使用 Systemd 服务命令。

要检查容器状态，请运行 **systemctl status** 命令：

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
   Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
 Main PID: 29012 (podman)
   CGroup: /system.slice/tripleo_keystone.service
           └─29012 /usr/bin/podman start -a keystone
```

要停止容器，请运行 **systemctl stop** 命令：

```
$ sudo systemctl stop tripleo_keystone
```

要启动容器，请运行 **systemctl start** 命令：

```
$ sudo systemctl start tripleo_keystone
```

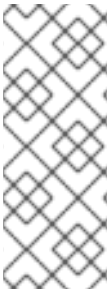
要重启容器，请运行 **systemctl restart** 命令：

```
$ sudo systemctl restart tripleo_keystone
```

由于没有守护进程监控容器状态，Systemd 在以下情况下自动重启大多数容器：

- 清除退出代码或信号，如运行 **podman stop** 命令。
- 取消清除退出代码，如启动后的 podman 容器崩溃。
- 取消清除信号。
- 如果容器启动时间超过 1 分 30 秒，则超时。

有关 Systemd 服务的更多信息，请参阅 **systemd.service** 文档。



注意

在重启容器后，针对其中的服务配置文件所做的所有更改都会恢复。这是因为容器基于 **/var/lib/config-data/puppet-generated/** 中节点的本地文件系统上的文件重新生成服务配置。例如，如果您编辑了 **keystone** 容器中的 **/etc/keystone/keystone.conf**，并重启了该容器，则该容器会使用节点的本地文件系统上的 **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** 来重新生成配置，以覆盖重启之前在该容器中所做的所有更改。

使用 Systemd 计时器监控 podman 容器

Systemd 计时器接口管理容器运行健康检查。每个容器都有一个计时器，它会运行一个服务单员来执行健康检查脚本。

要列出所有 OpenStack Platform 容器计时器，请运行 **systemctl list-timers** 命令并将输出限制为包含 **tripleo** 的行：

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left      Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer       tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:33 UTC  4s left      Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer      tripleo_mistral_engine_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left      Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer            tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left      Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer           tripleo_memcached_healthcheck.service
(...)
```

要检查特定容器计时器的状态，请对运行状况检查服务运行 **systemctl status** 命令：

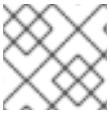
```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
```

```
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

要停止、启动、重启和显示容器计时器的状态，请根据 **.timer** Systemd 资源运行相关 **systemctl** 命令。例如，要检查 **tripleo_keystone_healthcheck.timer** 资源的状态，可运行以下命令：

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

如果健康状况检查服务被禁用，但该服务的计时器存在并启用，则意味着检查当前超时，但将根据计时器运行。您还可以手动启动检查。



注意

podman ps 命令不显示容器运行状态。

检查容器日志

OpenStack Platform 16 引入了一个新的日志记录目录 **/var/log/containers/stdout**，其中包含所有容器的标准输出 (stdout) 以及每个容器合并到一个文件中的标准错误 (stderr)。

Paunch 和 **container-puppet.py** 脚本配置 podman 容器以将其输出推送至 **/var/log/containers/stdout** 目录，这将创建所有日志的集合，甚至是删除的容器，如 **container-puppet-*** 容器。

主机会对此目录进行日志轮转以防止产生巨大的文件及占用太多磁盘空间的问题。

如果替换了容器，新的容器将输出到同一日志文件中，因为 **podman** 会使用容器名而非容器 ID。

您也可以使用 **podman logs** 命令检查容器化服务的日志。例如，要查看 **keystone** 容器的日志，请运行以下命令：

```
$ sudo podman logs keystone
```

访问容器

要进入容器化服务的 shell，请使用 **podman exec** 命令以启动 **/bin/bash**。例如，要进入 **keystone** 容器的 shell，请运行以下命令：

```
$ sudo podman exec -it keystone /bin/bash
```

要以根用户身份进入 **keystone** 容器的 shell，请运行以下命令：

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

要退出容器，请运行以下命令：

```
# exit
```

5.2. 容器化服务故障排除

如果容器化服务在 overcloud 部署期间或之后失败，请使用以下建议来确定失败的根本原因：

注意

在运行这些命令前，请检查您是否已登录到 overcloud 节点，而不是在 undercloud 上运行这些命令。

检查容器日志

每个容器都会保留其主进程的标准输出内容。此输出充当日志，以帮助确定容器运行期间实际发生什么。例如，要查看 **keystone** 容器的日志，请使用以下命令：

```
$ sudo podman logs keystone
```

在大多数情况下，此日志提供了容器故障的原因。

检查容器

在某些情况下，您可能需要验证容器的相关信息。例如，请使用以下命令来查看 **keystone** 容器的相关数据：

```
$ sudo podman inspect keystone
```

这提供了一个 JSON 对象，其中包含低级配置数据。您可以通过管道将这些输出内容传递给 **jq** 命令，以对特定数据进行解析。例如，要查看 **keystone** 容器的加载情况，请运行以下命令：

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

您还可以使用 **--format** 选项将数据解析到一行中，这在针对一组容器数据运行命令时非常有用。例如，要重建用于运行 **keystone** 容器的选项，请使用包含 **--format** 选项的以下 **inspect** 命令：

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' keystone
```

注意

--format 选项会按照 Go 语法来创建查询。

将这些选项和 **podman run** 命令一起使用以重新创建容器用于故障排除目的：

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
```

```
$ sudo podman run --rm $OPTIONS /bin/bash
```

在容器中运行命令

在某些情况下，您可能需要通过特定的 Bash 命令从容器中获取信息。在此情况下，使用以下 **podman** 命令以在运行中容器内执行命令。例如，要在 **keystone** 容器中运行命令：

```
$ sudo podman exec -ti keystone <COMMAND>
```

注意

-ti 选项会通过交互式伪终端来运行命令。

将 &**lt;COMMAND&** gt; 替换为您的所需命令。例如，每个容器都有一个健康检查脚本，用于验证服务的连接状况。您可以使用以下命令为 **keystone** 运行这个健康检查脚本：

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

要访问容器的 shell，请运行 **podman exec**，并将 **/bin/bash** 用作命令：

```
$ sudo podman exec -ti keystone /bin/bash
```

导出容器

当容器出现故障时，您可能需要调查文件中包含的所有内容。在这种情况下，您可以将容器的整个文件系统导出为 **tar** 归档。例如，要导出 **keystone** 容器的文件系统，请运行以下命令：

```
$ sudo podman export keystone -o keystone.tar
```

这个命令会创建 **keystone.tar** 归档，以供您提取和研究。

第 6 章 SYSTEMD 服务与容器化服务进行比较

本章提供了一些参考材料，用于显示容器化服务与 Systemd 服务不同。

6.1. SYSTEMD 服务和容器化服务

下表显示了基于 Systemd 的服务和由 Systemd 服务控制的 **podman** 容器之间的关联。

组件	systemd 服务	容器
OpenStack Image Storage (glance)	**tripleo_glance_api.service**	**glance_api**
HAProxy	**tripleo_haproxy.service**	**hapoxy**
OpenStack Orchestration (heat)	**tripleo_heat_api.service** **tripleo_heat_api_cfn.service** **tripleo_heat_api_cron.service** **tripleo_heat_engine.service**	**heat_api** **heat_api_cfn** **heat_api_cron** **heat_engine**
OpenStack Bare Metal (ironic)	**tripleo_ironic_api.service** **tripleo_ironic_conductor.service** **tripleo_ironic_inspector.service** **tripleo_ironic_inspector_dnsmasq.service** **tripleo_ironic_neutron_agent.service** **tripleo_ironic_pxe_http.service** **tripleo_ironic_pxe_tftp.service** **tripleo_iscsid.service**	**ironic_api** **ironic_conductor** **ironic_inspector** **ironic_inspector_dnsmasq** **ironic_neutron_agent** **ironic_pxe_http** **ironic_pxe_tftp** **iscsid**
Keepalived	**tripleo_keepalived.service**	**keepalived**
OpenStack Identity (keystone)	**tripleo_keystone.service** **tripleo_keystone_cron.service**	**Keystone** **keystone_cron**
logrotate	**tripleo_logrotate_crond.service**	**logrotate_crond**
Memcached	**tripleo_memcached.service**	**memcached**

组件	systemd 服务	容器
OpenStack Workflow (mistral)	**tripleo_mistral_api.service** **tripleo_mistral_engine.service** **tripleo_mistral_event_engine.service** **tripleo_mistral_executor.service**	**mistral_api** **mistral_engine** **mistral_event_engine** **mistral_executor**
MySQL	**tripleo_mysql.service**	**mysql**
OpenStack Networking (neutron)	**tripleo_neutron_api.service** **tripleo_neutron_dhcp.service** **tripleo_neutron_l3_agent.service** **tripleo_neutron_ovs_agent.service**	**neutron_api** **neutron_dhcp** **neutron_l3_agent** **neutron_ovs_agent**
OpenStack Compute (nova)	**tripleo_nova_api.service** **tripleo_nova_api_cron.service** **tripleo_nova_compute.service** **tripleo_nova_conductor.service** **tripleo_nova_metadata.service** **tripleo_nova_placement.service** **tripleo_nova_scheduler.service**	**nova_api** **nova_api_cron** **nova_compute** **nova_conductor** **nova_metadata** **nova_placement** **nova_scheduler**
RabbitMQ	**tripleo_rabbitmq.service**	**rabbitmq**
OpenStack Object Storage (swift)	**tripleo_swift_account_reaper.service** **tripleo_swift_account_server.service** **tripleo_swift_container_server.service** **tripleo_swift_container_updater.service** **tripleo_swift_object_expirer.service** **tripleo_swift_object_server.service** **tripleo_swift_object_updater.service** **tripleo_swift_proxy.service** **tripleo_swift_rsync.service**	**swift_account_reaper** **swift_account_server** **swift_container_server** **swift_container_updater** **swift_object_expirer** **swift_object_server** **swift_object_updater** **swift_proxy** **swift_rsync**

组件	systemd 服务	容器
OpenStack Messaging (zaqar)	**tripleo_zaqar.service** **tripleo_zaqar_websocket.service**	**zaqar** **zaqar_websocket**
aodh	**tripleo_aodh_api.service** **tripleo_aodh_evaluator.service** **tripleo_aodh_api_cron.service** **tripleo_aodh_listener.service** **tripleo_aodh_notifier.service**	**aodh_api** **aodh_listener** **aodh_evaluator** **aodh_api_cron** **aodh_notifier**
gnocchi	**tripleo_gnocchi_api.service** **tripleo_gnocchi_metricd.service** **tripleo_gnocchi_statsd.service**	**gnocchi_api** **gnocchi_metricd** **gnocchi_statsd**
opendoi	**tripleo_ceilometer_agent_central.service** **tripleo_ceilometer_agent_compute.service** **tripleo_ceilometer_agent_notification.service**	**ceilometer_agent_central** **ceilometer_agent_compute** **ceilometer_agent_notification**

6.2. SYSTEMD 日志位置和容器化日志位置

下表显示了基于 Systemd 的 OpenStack 日志及其用于容器的等效日志。所有基于容器的日志位置都位于物理主机上，并挂载到容器中。

OpenStack 服务	systemd 服务日志	容器日志
aodh	**/var/log/aodh/**	**/var/log/containers/aodh/** **/var/log/containers/httpd/aodh-api/**
opendoi	**/var/log/ceilometer/**	**/var/log/containers/ceilometer/**
cinder	**/var/log/cinder/**	**/var/log/containers/cinder/** **/var/log/containers/httpd/cinder-api/**
Glance	**/var/log/glance/**	**/var/log/containers/glance/**

OpenStack 服务	systemd 服务日志	容器日志
gnocchi	**/var/log/gnocchi/**	**/var/log/containers/gnocchi/** **/var/log/containers/httpd/gnocchi-api/**
heat	**/var/log/heat/**	**/var/log/containers/heat/** **/var/log/containers/httpd/heat-api/** **/var/log/containers/httpd/heat-api-cfn/**
Horizon	**/var/log/horizon/**	**/var/log/containers/horizon/** **/var/log/containers/httpd/horizon/**
Keystone	**/var/log/keystone/**	**/var/log/containers/keystone** **/var/log/containers/httpd/keystone/**
数据库	**/var/log/mariadb/** **/var/log/mongodb/** **/var/log/mysqld.log**	**/var/log/containers/mysql/**
neutron	**/var/log/neutron/**	**/var/log/containers/neutron/** **/var/log/containers/httpd/neutron-api/**
nova	**/var/log/nova/**	**/var/log/containers/nova/** **/var/log/containers/httpd/nova-api/** **/var/log/containers/httpd/placement/**
panko		**/var/log/containers/panko/** **/var/log/containers/httpd/panko-api/**
rabbitmq	**/var/log/rabbitmq/**	**/var/log/containers/rabbitmq/**
redis	**/var/log/redis/**	**/var/log/containers/redis/**

OpenStack 服务	systemd 服务日志	容器日志
swift	**/var/log/swift/**	**/var/log/containers/swift/**

6.3. SYSTEMD 配置与容器化配置

下表显示了基于 Systemd 的 OpenStack 配置及其用于容器的等效配置。所有基于容器的配置位置都位于物理主机上，并挂载到容器中，并将（通过 **kolla**）合并到每个对应容器中的配置中。

OpenStack 服务	systemd 服务配置	容器配置
aodh	**/etc/aodh/**	**/var/lib/config-data/puppet-generated/aodh/**
opendoi	**/etc/ceilometer/**	**/var/lib/config-data/puppet-generated/ceilometer/etc/ceilometer/**
cinder	**/etc/cinder/**	**/var/lib/config-data/puppet-generated/cinder/etc/cinder/**
Glance	**/etc/glance/**	**/var/lib/config-data/puppet-generated/glance_api/etc/glance/**
gnocchi	**/etc/gnocchi/**	**/var/lib/config-data/puppet-generated/gnocchi/etc/gnocchi/**
hapoxy	**/etc/haproxy/**	**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/**
heat	**/etc/heat/**	**/var/lib/config-data/puppet-generated/heat/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api_cfn/etc/heat/**
Horizon	**/etc/openstack-dashboard/**	**/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/**
Keystone	**/etc/keystone/**	**/var/lib/config-data/puppet-generated/keystone/etc/keystone/**

OpenStack 服务	systemd 服务配置	容器配置
数据库	**/etc/my.cnf.d/** **/etc/my.cnf**	**/var/lib/config-data/puppet-generated/mysql/etc/my.cnf.d/**
neutron	**/etc/neutron/**	**/var/lib/config-data/puppet-generated/neutron/etc/neutron/**
nova	**/etc/nova/**	**/var/lib/config-data/puppet-generated/nova/etc/nova/** **/var/lib/config-data/puppet-generated/etc/placement/**
panko		**/var/lib/config-data/puppet-generated/panko/etc/panko**
rabbitmq	**/etc/rabbitmq/**	**/var/lib/config-data/puppet-generated/rabbitmq/etc/rabbitmq/**
redis	**/etc/redis/** **/etc/redis.conf**	**/var/lib/config-data/puppet-generated/redis/etc/redis/** **/var/lib/config-data/puppet-generated/redis/etc/redis.conf**
swift	**/etc/swift/**	**/var/lib/config-data/puppet-generated/swift/etc/swift/** **/var/lib/config-data/puppet-generated/swift_ringbuilder/etc/swift/**