Red Hat OpenStack Platform 13

Transitioning to Containerized Services

A basic guide to working with OpenStack Platform containerized services

Last Updated: 2023-01-31

Red Hat OpenStack Platform 13 Transitioning to Containerized Services

A basic guide to working with OpenStack Platform containerized services

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide provides some basic information to help users get accustomed working with OpenStack Platform services running in containers.

Table of Contents

CHAPTER 1. INTRODUCTION

Past versions of Red Hat OpenStack Platform used services managed with Systemd. However, more recent version of OpenStack Platform now use containers to run services. Some administrators might not have a good understanding of how containerized OpenStack Platform services operate, and so this guide aims to help you understand OpenStack Platform container images and containerized services. This includes:

- How to obtain and modify container images
- How to manage containerized services in the overcloud
- Understanding how containers differ from Systemd services

The main goal is to help you gain enough knowledge of containerized OpenStack Platform services to transition from a Systemd-based environment to a container-based environment.

1.1. CONTAINERIZED SERVICES AND KOLLA

Each of the main Red Hat OpenStack Platform (RHOSP) services run in containers. This provides a method to keep each service within its own isolated namespace separated from the host. This has the following effects:

- During deployment, RHOSP pulls and runs container images from the Red Hat Customer Portal.
- The **podman** command operates management functions, like starting and stopping services.
- To upgrade containers, you must pull new container images and replace the existing containers with newer versions.

Red Hat OpenStack Platform uses a set of containers built and managed with the **Kolla** toolset.

CHAPTER 2. OBTAINING AND MODIFYING CONTAINER IMAGES

A containerized overcloud requires access to a registry with the required container images. This chapter provides information on how to prepare the registry and your overcloud configuration to use container images for Red Hat OpenStack Platform.

This guide provides several use cases to configure your overcloud to use a registry. Before attempting one of these use cases, it is recommended to familiarize yourself with how to use the image preparation command. See Section 2.2, "Container image preparation command usage" for more information.

2.1. REGISTRY METHODS

Red Hat OpenStack Platform supports the following registry types:

Remote Registry

The overcloud pulls container images directly from **registry.redhat.io**. This method is the easiest for generating the initial configuration. However, each overcloud node pulls each image directly from the Red Hat Container Catalog, which can cause network congestion and slower deployment. In addition, all overcloud nodes require internet access to the Red Hat Container Catalog.

Local Registry

The undercloud uses the **docker-distribution** service to act as a registry. This allows the director to synchronize the images from **registry.redhat.io** and push them to the **docker-distribution** registry. When creating the overcloud, the overcloud pulls the container images from the undercloud's **docker-distribution** registry. This method allows you to store a registry internally, which can speed up the deployment and decrease network congestion. However, the undercloud only acts as a basic registry and provides limited life cycle management for container images.

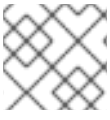
NOTE

The **docker-distribution** service acts separately from **docker**. **docker** is used to pull and push images to the **docker-distribution** registry and does not serve the images to the overcloud. The overcloud pulls the images from the **docker-distribution** registry.

Satellite Server

Manage the complete application life cycle of your container images and publish them through a Red Hat Satellite 6 server. The overcloud pulls the images from the Satellite server. This method provides an enterprise grade solution to store, manage, and deploy Red Hat OpenStack Platform containers.

Select a method from the list and continue configuring your registry details.



NOTE

When building for a multi-architecture cloud, the local registry option is not supported.

2.2. CONTAINER IMAGE PREPARATION COMMAND USAGE

This section provides an overview on how to use the **openstack overcloud container image prepare** command, including conceptual information on the command's various options.

Generating a Container Image Environment File for the Overcloud

One of the main uses of the **openstack overcloud container image prepare** command is to create an environment file that contains a list of images the overcloud uses. You include this file with your overcloud deployment commands, such as **openstack overcloud deploy**. The **openstack overcloud container image prepare** command uses the following options for this function:

--output-env-file

Defines the resulting environment file name.

The following snippet is an example of this file's contents:

```
parameter_defaults:
  DockerAodhApiImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
  DockerAodhConfigImage: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
...
```

The environment file also contains the **DockerInsecureRegistryAddress** parameter set to the IP address and port of the undercloud registry. This parameter configures overcloud nodes to access images from the undercloud registry without SSL/TLS certification.

Generating a Container Image List for Import Methods

If you aim to import the OpenStack Platform container images to a different registry source, you can generate a list of images. The syntax of list is primarily used to import container images to the container registry on the undercloud, but you can modify the format of this list to suit other import methods, such as Red Hat Satellite 6.

The **openstack overcloud container image prepare** command uses the following options for this function:

--output-images-file

Defines the resulting file name for the import list.

The following is an example of this file's contents:

```
container_images:
- imagename: registry.redhat.io/rhosp13/openstack-aodh-api:13.0-34
- imagename: registry.redhat.io/rhosp13/openstack-aodh-evaluator:13.0-34
...
```

Setting the Namespace for Container Images

Both the **--output-env-file** and **--output-images-file** options require a namespace to generate the resulting image locations. The **openstack overcloud container image prepare** command uses the following options to set the source location of the container images to pull:

--namespace

Defines the namespace for the container images. This is usually a hostname or IP address with a directory.

--prefix

Defines the prefix to add before the image names.

As a result, the director generates the image names using the following format:

- **[NAMESPACE]/[PREFIX][IMAGE NAME]**

Setting Container Image Tags

Use the **--tag** and **--tag-from-label** options together to set the tag for each container images.

--tag

Sets the specific tag for all images from the source. If you only use this option, director pulls all container images using this tag. However, if you use this option in combination with **--tag-from-label**, director uses the **--tag** as a source image to identify a specific version tag based on labels. The **--tag** option is set to **latest** by default.

--tag-from-label

Use the value of specified container image labels to discover and pull the versioned tag for every image. Director inspects each container image tagged with the value that you set for **--tag**, then uses the container image labels to construct a new tag, which director pulls from the registry. For example, if you set **--tag-from-label {version}-{release}**, director uses the **version** and **release** labels to construct a new tag. For one container, **version** might be set to **13.0** and **release** might be set to **34**, which results in the tag **13.0-34**.

IMPORTANT

The Red Hat Container Registry uses a specific version format to tag all Red Hat OpenStack Platform container images. This version format is **{version}-{release}**, which each container image stores as labels in the container metadata. This version format helps facilitate updates from one **{release}** to the next. For this reason, you must always use the **--tag-from-label {version}-{release}** when running the **openstack overcloud container image prepare** command. Do not only use **--tag** on its own to to pull container images. For example, using **--tag latest** by itself causes problems when performing updates because director requires a change in tag to update a container image.

2.3. CONTAINER IMAGES FOR ADDITIONAL SERVICES

The director only prepares container images for core OpenStack Platform Services. Some additional features use services that require additional container images. You enable these services with environment files. The **openstack overcloud container image prepare** command uses the following option to include environment files and their respective container images:

-e

Include environment files to enable additional container images.

The following table provides a sample list of additional services that use container images and their respective environment file locations within the **/usr/share/openstack-tripleo-heat-templates** directory.

Service	Environment File
Ceph Storage	**environments/ceph-ansible/ceph-ansible.yaml**
Collectd	**environments/services-docker/collectd.yaml**
Congress	**environments/services-docker/congress.yaml**
Fluentd	**environments/services-docker/fluentd.yaml**

Service	Environment File
OpenStack Bare Metal (ironic)	**environments/services-docker/ironic.yaml**
OpenStack Data Processing (sahara)	**environments/services-docker/sahara.yaml**
OpenStack EC2-API	**environments/services-docker/ec2-api.yaml**
OpenStack Key Manager (barbican)	**environments/services-docker/barbican.yaml**
OpenStack Load Balancing-as-a-Service (octavia)	**environments/services-docker/octavia.yaml**
OpenStack Shared File System Storage (manila)	**environments/manila-{backend-name}-config.yaml** NOTE: See OpenStack Shared File System (manila)for more information.
Open Virtual Network (OVN)	**environments/services-docker/neutron-ovn-dvr-ha.yaml**
Sensu	**environments/services-docker/sensu-client.yaml**

The next few sections provide examples of including additional services.

Ceph Storage

If deploying a Red Hat Ceph Storage cluster with your overcloud, you need to include the **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** environment file. This file enables the composable containerized services in your overcloud and the director needs to know these services are enabled to prepare their images.

In addition to this environment file, you also need to define the Ceph Storage container location, which is different from the OpenStack Platform services. Use the **--set** option to set the following parameters specific to Ceph Storage:

--set ceph_namespace

Defines the namespace for the Ceph Storage container image. This functions similar to the **--namespace** option.

--set ceph_image

Defines the name of the Ceph Storage container image. Usually,this is **rhceph-3-rhel7**.

--set ceph_tag

Defines the tag to use for the Ceph Storage container image. This functions similar to the **--tag** option. When **--tag-from-label** is specified, the versioned tag is discovered starting from this tag.

The following snippet is an example that includes Ceph Storage in your container image files:

```
$ openstack overcloud container image prepare \
  ...
```

```
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml \
--set ceph_namespace=registry.redhat.io/rhceph \
--set ceph_image=rhceph-3-rhel7 \
--tag-from-label {version}-{release} \
...
```

OpenStack Bare Metal (ironic)

If deploying OpenStack Bare Metal (ironic) in your overcloud, you need to include the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml** environment file so the director can prepare the images. The following snippet is an example on how to include this environment file:

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml \
  ...
```

OpenStack Data Processing (sahara)

If deploying OpenStack Data Processing (sahara) in your overcloud, you need to include the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml** environment file so the director can prepare the images. The following snippet is an example on how to include this environment file:

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/sahara.yaml \
  ...
```

OpenStack Neutron SR-IOV

If deploying OpenStack Neutron SR-IOV in your overcloud, include the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml** environment file so the director can prepare the images. The default Controller and Compute roles do not support the SR-IOV service, so you must also use the **-r** option to include a custom roles file that contains SR-IOV services. The following snippet is an example on how to include this environment file:

```
$ openstack overcloud container image prepare \
  ...
  -r ~/custom_roles_data.yaml
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-sriov.yaml \
  ...
```

OpenStack Load Balancing-as-a-Service (octavia)

If deploying OpenStack Load Balancing-as-a-Service in your overcloud, include the **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml** environment file so the director can prepare the images. The following snippet is an example on how to include this environment file:

```
$ openstack overcloud container image prepare \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml
```

```
\
  ...
```

OpenStack Shared File System (manila)

Using the format **manila-{backend-name}-config.yaml**, you can choose a supported back end to deploy the Shared File System with that back end. Shared File System service containers can be prepared by including any of the following environment files:

```
environments/manila-isilon-config.yaml
environments/manila-netapp-config.yaml
environments/manila-vmax-config.yaml
environments/manila-cephfsnative-config.yaml
environments/manila-cephfsganesha-config.yaml
environments/manila-unity-config.yaml
environments/manila-vnx-config.yaml
```

For more information about customizing and deploying environment files, see the following resources:

- Deploying the updated environment in *CephFS via NFS Back End Guide for the Shared File System Service*
- Deploy the Shared File System Service with NetApp Back Ends in *NetApp Back End Guide for the Shared File System Service*
- Deploy the Shared File System Service with a CephFS Back End in *CephFS Back End Guide for the Shared File System Service*

2.4. USING THE RED HAT REGISTRY AS A REMOTE REGISTRY SOURCE

Red Hat hosts the overcloud container images on **registry.redhat.io**. Pulling the images from a remote registry is the simplest method because the registry is already configured and all you require is the URL and namespace of the image that you want to pull. However, during overcloud creation, the overcloud nodes all pull images from the remote repository, which can congest your external connection. As a result, this method is not recommended for production environments. For production environments, use one of the following methods instead:

- Setup a local registry
- Host the images on Red Hat Satellite 6

Procedure

1. To pull the images directly from **registry.redhat.io** in your overcloud deployment, an environment file is required to specify the image parameters. Run the following command to generate the container image environment file:

```
(undercloud) $ sudo openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --prefix=openstack- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

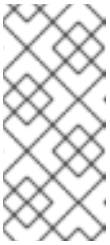
   - Use the **-e** option to include any environment files for optional services.

- Use the **-r** option to include a custom roles file.
- If using Ceph Storage, include the additional parameters to define the Ceph Storage container image location: **--set ceph_namespace**, **--set ceph_image**, **--set ceph_tag**.

2. Modify the **overcloud_images.yaml** file and include the following parameters to authenticate with **registry.redhat.io** during deployment:

```
ContainerImageRegistryLogin: true
ContainerImageRegistryCredentials:
  registry.redhat.io:
    <USERNAME>: <PASSWORD>
```

- Replace **<USERNAME>** and **<PASSWORD>** with your credentials for **registry.redhat.io**. The **overcloud_images.yaml** file contains the image locations on the undercloud. Include this file with your deployment.



NOTE

Before you run the **openstack overcloud deploy** command, you must log in to the remote registry:

```
(undercloud) $ sudo docker login registry.redhat.io
```

The registry configuration is ready.

2.5. USING THE UNDERCLOUD AS A LOCAL REGISTRY

You can configure a local registry on the undercloud to store overcloud container images.

You can use director to pull each image from the **registry.redhat.io** and push each image to the **docker-distribution** registry that runs on the undercloud. When you use director to create the overcloud, during the overcloud creation process, the nodes pull the relevant images from the undercloud **docker-distribution** registry.

This keeps network traffic for container images within your internal network, which does not congest your external network connection and can speed the deployment process.

Procedure

1. Find the address of the local undercloud registry. The address uses the following pattern:

```
<REGISTRY_IP_ADDRESS>:8787
```

Use the IP address of your undercloud, which you previously set with the **local_ip** parameter in your **undercloud.conf** file. For the commands below, the address is assumed to be **192.168.24.1:8787**.

2. Log in to **registry.redhat.io**:

```
(undercloud) $ docker login registry.redhat.io --username $RH_USER --password
$RH_PASSWD
```

3. Create a template to upload the images to the local registry, and the environment file to refer to those images:

```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=registry.redhat.io/rhosp13 \
  --push-destination=192.168.24.1:8787 \
  --prefix=openstack- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml \
  --output-images-file /home/stack/local_registry_images.yaml
```

   - Use the **-e** option to include any environment files for optional services.
   - Use the **-r** option to include a custom roles file.
   - If using Ceph Storage, include the additional parameters to define the Ceph Storage container image location: **--set ceph_namespace**, **--set ceph_image**, **--set ceph_tag**.

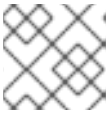
4. Verify that the following two files have been created:

   - **local_registry_images.yaml**, which contains container image information from the remote source. Use this file to pull the images from the Red Hat Container Registry (**registry.redhat.io**) to the undercloud.
   - **overcloud_images.yaml**, which contains the eventual image locations on the undercloud. You include this file with your deployment.

5. Pull the container images from the remote registry and push them to the undercloud registry:

```
(undercloud) $ openstack overcloud container image upload \
  --config-file  /home/stack/local_registry_images.yaml \
  --verbose
```

   Pulling the required images might take some time depending on the speed of your network and your undercloud disk.

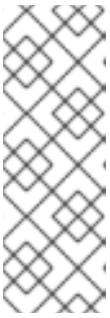


NOTE

The container images consume approximately 10 GB of disk space.

6. The images are now stored on the undercloud's **docker-distribution** registry. To view the list of images on the undercloud's **docker-distribution** registry, run the following command:

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog | jq .repositories[]
```



NOTE

The **_catalog** resource by itself displays only 100 images. To display more images, use the **?n=<interger>** query string with the **_catalog** resource to display a larger number of images:

```
(undercloud) $  curl http://192.168.24.1:8787/v2/_catalog?n=150 | jq
.repositories[]
```

To view a list of tags for a specific image, use the **skopeo** command:

```
(undercloud) $ curl -s http://192.168.24.1:8787/v2/rhosp13/openstack-keystone/tags/list | jq
.tags
```

To verify a tagged image, use the **skopeo** command:

```
(undercloud) $ skopeo inspect --tls-verify=false
docker://192.168.24.1:8787/rhosp13/openstack-keystone:13.0-44
```

The registry configuration is ready.

2.6. USING A SATELLITE SERVER AS A REGISTRY

Red Hat Satellite 6 offers registry synchronization capabilities. This provides a method to pull multiple images into a Satellite server and manage them as part of an application life cycle. The Satellite also acts as a registry for other container-enabled systems to use. For more details information on managing container images, see "Managing Container Images" in the *Red Hat Satellite 6 Content Management Guide*.

The examples in this procedure use the **hammer** command line tool for Red Hat Satellite 6 and an example organization called **ACME**. Substitute this organization for your own Satellite 6 organization.

Procedure

1. Create a template to pull images to the local registry:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud container image prepare \
  --namespace=rhosp13 \
  --prefix=openstack- \
  --output-images-file /home/stack/satellite_images
```

   - Use the **-e** option to include any environment files for optional services.
   - Use the **-r** option to include a custom roles file.
   - If using Ceph Storage, include the additional parameters to define the Ceph Storage container image location: **--set ceph_namespace**, **--set ceph_image**, **--set ceph_tag**.

NOTE

This version of the **openstack overcloud container image prepare** command targets the registry on the **registry.redhat.io** to generate an image list. It uses different values than the **openstack overcloud container image prepare** command used in a later step.

2. This creates a file called **satellite_images** with your container image information. You will use this file to synchronize container images to your Satellite 6 server.
3. Remove the YAML-specific information from the **satellite_images** file and convert it into a flat file containing only the list of images. The following **sed** commands accomplish this:

```
(undercloud) $ awk -F ':' '{if (NR!=1) {gsub("[[:space:]]", ""); print $2}}' ~/satellite_images > ~/satellite_images_names
```

This provides a list of images that you pull into the Satellite server.

4. Copy the **satellite_images_names** file to a system that contains the Satellite 6 **hammer** tool. Alternatively, use the instructions in the *Hammer CLI Guide* to install the **hammer** tool to the undercloud.

5. Run the following **hammer** command to create a new product (**OSP13 Containers**) to your Satellite organization:

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP13 Containers"
```

This custom product will contain our images.

6. Add the base container image to the product:

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhosp13/openstack-base \
  --name base
```

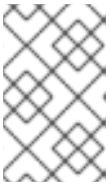
7. Add the overcloud container images from the **satellite_images** file.

```
$ while read IMAGE; do \
  IMAGENAME=$(echo $IMAGE | cut -d"/" -f2 | sed "s/openstack-//g" | sed "s/:.*//g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP13 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE \
  --name $IMAGENAME ; done < satellite_images_names
```

8. Synchronize the container images:

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP13 Containers"
```

Wait for the Satellite server to complete synchronization.



NOTE

Depending on your configuration, **hammer** might ask for your Satellite server username and password. You can configure **hammer** to automatically login using a configuration file. See the "Authentication" section in the *Hammer CLI Guide*.

9. If your Satellite 6 server uses content views, create a new content view version to incorporate the images.

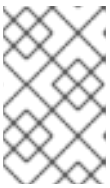
10. Check the tags available for the **base** image:

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --product "OSP13 Containers"
```

This displays tags for the OpenStack Platform container images.

11. Return to the undercloud and generate an environment file for the images on your Satellite server. The following is an example command for generating the environment file:

```
(undercloud) $ openstack overcloud container image prepare \
  --namespace=satellite6.example.com:5000 \
  --prefix=acme-osp13_containers- \
  --tag-from-label {version}-{release} \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

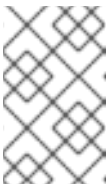


NOTE

This version of the **openstack overcloud container image prepare** command targets the Satellite server. It uses different values than the **openstack overcloud container image prepare** command used in a previous step.

When running this command, include the following data:

- **--namespace** - The URL and port of the registry on the Satellite server. The registry port on Red Hat Satellite is 5000. For example, **--namespace=satellite6.example.com:5000**.



NOTE

If you are using Red Hat Satellite version 6.10, you do not need to specify a port. The default port of **443** is used. For more information, see "How can we adapt RHOSP13 deployment to Red Hat Satellite 6.10?".

- **--prefix=** - The prefix is based on a Satellite 6 convention for labels, which uses lower case characters and substitutes spaces for underscores. The prefix differs depending on whether you use content views:
  - If you use content views, the structure is **[org]-[environment]-[content view]-[product]-**. For example: **acme-production-myosp13-osp13_containers-**.
  - If you do not use content views, the structure is **[org]-[product]-**. For example: **acme-osp13_containers-**.
- **--tag-from-label {version}-{release}** - Identifies the latest tag for each image.
- **-e** - Include any environment files for optional services.
- **-r** - Include a custom roles file.
- **--set ceph_namespace**, **--set ceph_image**, **--set ceph_tag** - If using Ceph Storage, include the additional parameters to define the Ceph Storage container image location.

Note that **ceph_image** now includes a Satellite-specific prefix. This prefix is the same value as the **--prefix** option. For example:

```
--set ceph_image=acme-osp13_containers-rhceph-3-rhel7
```

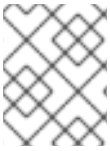
This ensures the overcloud uses the Ceph container image using the Satellite naming convention.

12. The **overcloud_images.yaml** file contains the image locations on the Satellite server. Include this file with your deployment.

The registry configuration is ready.

2.7. MODIFYING CONTAINERS IMAGES

Red Hat provides a set of pre-built container images through the Red Hat Container Catalog (**registry.redhat.io**). It is possible to modify these images and add additional layers to them. This is useful for adding RPMs for certified 3rd party drivers to the containers.



NOTE

To ensure continued support for modified OpenStack Platform container images, ensure that the resulting images comply with the "Red Hat Container Support Policy" .

This example shows how to customize the latest **openstack-keystone** image. However, these instructions can also apply to other images:

Procedure

1. Pull the image you aim to modify. For example, for the **openstack-keystone** image:

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

2. Check the default user on the original image. For example, for the **openstack-keystone** image:

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

NOTE

The **openstack-keystone** image uses **root** as the default user. Other images use specific users. For example, **openstack-glance-api** uses **glance** for the default user.

3. Create a **Dockerfile** to build an additional layer on an existing container image. The following is an example that pulls the latest OpenStack Identity (keystone) image from the Container Catalog and installs a custom RPM file to the image:

```
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. Build and tag the new image. For example, to build with a local **Dockerfile** stored in the **/home/stack/keystone** directory and tag it to your undercloud's local registry:

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

5. Push the resulting image to the undercloud's local registry:

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

6. Edit your overcloud container images environment file (usually **overcloud_images.yaml**) and change the appropriate parameter to use the custom container image.

WARNING

The Container Catalog publishes container images with a complete software stack built into it. When the Container Catalog releases a container image with updates and security fixes, your existing custom container will **not** include these updates and will require rebuilding using the new image version from the Catalog.

CHAPTER 3. DEPLOYING AND UPDATING AN OVERCLOUD WITH CONTAINERS

This chapter provides info on how to create a container-based overcloud and keep it updated.

3.1. DEPLOYING AN OVERCLOUD

This procedure demonstrates how to deploy an overcloud with minimum configuration. The result will be a basic two-node overcloud (1 Controller node, 1 Compute node).

Procedure

1. Source the **stackrc** file:

```
$ source ~/stackrc
```

2. Run the **deploy** command and include the file containing your overcloud image locations (usually **overcloud_images.yaml**):

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

3. Wait until the overcloud completes deployment.

3.2. UPDATING AN OVERCLOUD

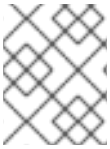
For information on updating a containerized overcloud, see the *Keeping Red Hat OpenStack Platform Updated* guide.

CHAPTER 4. WORKING WITH CONTAINERIZED SERVICES

This chapter provides some examples of commands to manage containers and how to troubleshoot your OpenStack Platform containers

4.1. MANAGING CONTAINERIZED SERVICES

The overcloud runs most OpenStack Platform services in containers. In certain situations, you might need to control the individual services on a host. This section provides some common **docker** commands you can run on an overcloud node to manage containerized services. For more comprehensive information on using **docker** to manage containers, see Working with Docker formatted containers in the *Getting Started with Containers* guide.



NOTE

Before running these commands, check that you are logged into an overcloud node and not running these commands on the undercloud.

Listing containers and images

To list running containers:

```
$ sudo docker ps
```

To also list stopped or failed containers, add the **--all** option:

```
$ sudo docker ps --all
```

To list container images:

```
$ sudo docker images
```

Inspecting container properties

To view the properties of a container or container images, use the **docker inspect** command. For example, to inspect the **keystone** container:

```
$ sudo docker inspect keystone
```

Managing basic container operations

To restart a containerized service, use the **docker restart** command. For example, to restart the **keystone** container:

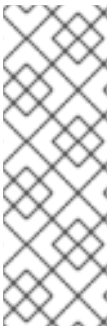
```
$ sudo docker restart keystone
```

To stop a containerized service, use the **docker stop** command. For example, to stop the **keystone** container:

```
$ sudo docker stop keystone
```

To start a stopped containerized service, use the **docker start** command. For example, to start the **keystone** container:

```
$ sudo docker start keystone
```



NOTE

Any changes to the service configuration files within the container revert after restarting the container. This is because the container regenerates the service configuration based upon files on the node's local file system in **/var/lib/config-data/puppet-generated/**. For example, if you edit **/etc/keystone/keystone.conf** within the **keystone** container and restart the container, the container regenerates the configuration using **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** on the node's local file system, which overwrites any the changes made within the container before the restart.

Monitoring containers

To check the logs for a containerized service, use the **docker logs** command. For example, to view the logs for the **keystone** container:

```
$ sudo docker logs keystone
```

Accessing containers

To enter the shell for a containerized service, use the **docker exec** command to launch **/bin/bash**. For example, to enter the shell for the **keystone** container:

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

To enter the shell for the **keystone** container as the root user:

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

To exit from the container:

```
# exit
```

4.2. TROUBLESHOOTING CONTAINERIZED SERVICES

If a containerized service fails during or after overcloud deployment, use the following recommendations to determine the root cause for the failure:

NOTE

Before running these commands, check that you are logged into an overcloud node and not running these commands on the undercloud.

Checking the container logs

Each container retains standard output from its main process. This output acts as a log to help determine what actually occurs during a container run. For example, to view the log for the **keystone** container, use the following command:

```
$ sudo docker logs keystone
```

In most cases, this log provides the cause of a container's failure.

Inspecting the container

In some situations, you might need to verify information about a container. For example, use the following command to view **keystone** container data:

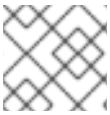
```
$ sudo docker inspect keystone
```

This provides a JSON object containing low-level configuration data. You can pipe the output to the **jq** command to parse specific data. For example, to view the container mounts for the **keystone** container, run the following command:

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

You can also use the **--format** option to parse data to a single line, which is useful for running commands against sets of container data. For example, to recreate the options used to run the **keystone** container, use the following **inspect** command with the **--format** option:

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



NOTE

The **--format** option uses Go syntax to create queries.

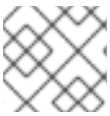
Use these options in conjunction with the **docker run** command to recreate the container for troubleshooting purposes:

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

Running commands in the container

In some cases, you might need to obtain information from within a container through a specific Bash command. In this situation, use the following **docker** command to execute commands within a running container. For example, to run a command in the **keystone** container:

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



NOTE

The **-ti** options run the command through an interactive pseudoterminal.

Replace **<COMMAND>** with your desired command. For example, each container has a health check script to verify the service connection. You can run the health check script for **keystone** with the following command:

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

To access the container's shell, run **docker exec** using **/bin/bash** as the command:

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

Exporting a container

When a container fails, you might need to investigate the full contents of the file. In this case, you can export the full file system of a container as a **tar** archive. For example, to export the **keystone** container's file system, run the following command:

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

This command create the **keystone.tar** archive, which you can extract and explore.

CHAPTER 5. COMPARING SYSTEMD SERVICES TO CONTAINERIZED SERVICES

This chapter provides some reference material to show how containerized services differ from Systemd services.

5.1. SYSTEMD SERVICE COMMANDS VS CONTAINERIZED SERVICE COMMANDS

The following table shows some similarities between Systemd-based commands and their Docker equivalents. This helps identify the type of service operation you aim to perform.

Function	Systemd-based	Docker-based
List all services	**systemctl list-units -t service**	**docker ps --all**
List active services	**systemctl list-units -t service --state active**	**docker ps**
Check status of service	**systemctl status openstack-nova-api**	**docker ps --filter "name=nova_api$" --all**
Stop service	**systemctl stop openstack-nova-api**	**docker stop nova_api**
Start service	**systemctl start openstack-nova-api**	**docker start nova_api**
Restart service	**systemctl restart openstack-nova-api**	**docker restart nova_api**
Show service configuration	**systemctl show openstack-nova-api** **systemctl cat openstack-nova-api**	**docker inspect nova_api**
Show service logs	**journalctl -u openstack-nova-api**	**docker logs nova_api**

5.2. SYSTEMD SERVICES VS CONTAINERIZED SERVICES

The following table shows Systemd-based OpenStack services and their container-based equivalents.

OpenStack service	Systemd services	Docker containers

OpenStack service	Systemd services	Docker containers
aodh	**openstack-aodh-evaluator** **openstack-aodh-listener** **openstack-aodh-notifier** **httpd (openstack-aodh-api)**	**aodh_listener** **aodh_api** **aodh_notifier** **aodh_evaluator**
ceilometer	**openstack-ceilometer-central** **openstack-ceilometer-collector** **openstack-ceilometer-notification** **httpd (openstack-ceilometer-api)**	**ceilometer_agent_notification** **ceilometer_agent_central**
cinder	**openstack-cinder-api** **openstack-cinder-scheduler** **openstack-cinder-volume**	**cinder_scheduler** **cinder_api** **openstack-cinder-volume-docker-0**
glance	**openstack-glance-api** **openstack-glance-registry**	**glance_api**
gnocchi	**openstack-gnocchi-metricd** **openstack-gnocchi-statsd** **httpd (openstack-gnocchi-api)**	**gnocchi_statsd** **gnocchi_api** **gnocchi_metricd**
heat	**openstack-heat-api-cfn** **openstack-heat-api-cloudwatch** **openstack-heat-api** **openstack-heat-engine**	**heat_api_cfn** **heat_engine** **heat_api**
horizon	**httpd (openstack-dashboard)**	**horizon**
keystone	**httpd (openstack-keystone)**	**keystone**

OpenStack service	Systemd services	Docker containers
neutron	**neutron-dhcp-agent** **neutron-l3-agent** **neutron-metadata-agent** **neutron-openvswitch-agent** **neutron-server**	**neutron_ovs_agent** **neutron_l3_agent** **neutron_metadata_agent** **neutron_dhcp** **neutron_api**
nova	**openstack-nova-api** **openstack-nova-conductor** **openstack-nova-consoleauth** **openstack-nova-novncproxy** **openstack-nova-scheduler**	**nova_metadata** **nova_api** **nova_conductor** **nova_vnc_proxy** **nova_consoleauth** **nova_api_cron** **nova_scheduler** **nova_placement**
panko		**panko_api**

OpenStack service	Systemd services	Docker containers
swift	**openstack-swift-account-auditor** **openstack-swift-account-reaper** **openstack-swift-account-replicator** **openstack-swift-account** **openstack-swift-container-auditor** **openstack-swift-container-replicator** **openstack-swift-container-updater** **openstack-swift-container** **openstack-swift-object-auditor** **openstack-swift-object-expirer** **openstack-swift-object-replicator** **openstack-swift-object-updater** **openstack-swift-object** **openstack-swift-proxy**	**swift_proxy** **swift_account_server** **swift_container_auditor** **swift_object_expirer** **swift_object_updater** **swift_container_replicator** **swift_account_auditor** **swift_object_replicator** **swift_container_server** **swift_rsync** **swift_account_reaper** **swift_account_replicator** **swift_object_auditor** **swift_object_server** **swift_container_update**

5.3. SYSTEMD LOG LOCATIONS VS CONTAINERIZED LOG LOCATIONS

The following table shows Systemd-based OpenStack logs and their equivalents for containers. All container-based log locations are available on the physical host and are mounted to the container.

OpenStack service	Systemd service logs	Docker container logs
aodh	**/var/log/aodh/**	**/var/log/containers/aodh/** **/var/log/containers/httpd/aodh-api/**

OpenStack service	Systemd service logs	Docker container logs
ceilometer	**/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/**
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/**
databases	**/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/**

OpenStack service	Systemd service logs	Docker container logs
nova	**/var/log/nova/**	**/var/log/containers/nova/** **/var/log/containers/httpd/nova-api/** **/var/log/containers/httpd/nova-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/**
swift	**/var/log/swift/**	**/var/log/containers/swift/**

5.4. SYSTEMD CONFIGURATION VS CONTAINERIZED CONFIGURATION

The following table shows Systemd-based OpenStack configuration and their equivalents for containers. All container-based configuration locations are available on the physical host, are mounted to the container, and are merged (via **kolla**) into the configuration within each respective container.

OpenStack service	Systemd service configuration	Docker container configuration
aodh	**/etc/aodh/**	**/var/lib/config-data/puppet-generated/aodh/**
ceilometer	**/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/**

OpenStack service	Systemd service configuration	Docker container configuration
haproxy	**/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/**
databases	**/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/nova_placement/etc/nova/**
panko		**/var/lib/config-data/puppet-generated/panko/etc/panko**
rabbitmq	**/etc/rabbitmq/**	**/var/lib/config-data/puppet-generated/rabbitmq/etc/rabbitmq/**

OpenStack service	Systemd service configuration	Docker container configuration
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/**