Red Hat OpenStack Platform 12

Deploying an Overcloud with Containerized Red Hat Ceph

Configuring the Director to Deploy and Use a Containerized Red Hat Ceph Cluster

Last Updated: 2019-03-05

Red Hat OpenStack Platform 12 Deploying an Overcloud with Containerized Red Hat Ceph

Configuring the Director to Deploy and Use a Containerized Red Hat Ceph Cluster

OpenStack Team
rhos-docs@redhat.com

Abstract

This guide provides information on using the Red Hat OpenStack Platform director to create an Overcloud with a containerized Red Hat Ceph Storage cluster. This includes instructions for customizing your Ceph cluster through the director.

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat OpenStack Platform director creates a cloud environment called the *overcloud*. The director provides the ability to configure extra features for an Overcloud. One of these extra features includes integration with Red Hat Ceph Storage. This includes both Ceph Storage clusters created with the director or existing Ceph Storage clusters.

The Red Hat Ceph cluster described in this guide features *containerized Ceph Storage*. For more information about containerized services in OpenStack, see "Configuring a Basic Overcloud with the CLI Tools" in the Director *Installation and Usage Guide*.

1.1. DEFINING CEPH STORAGE

Red Hat Ceph Storage is a distributed data object store designed to provide excellent performance, reliability, and scalability. Distributed object stores are the future of storage, because they accommodate unstructured data, and because clients can use modern object interfaces and legacy interfaces simultaneously. At the heart of every Ceph deployment is the **Ceph Storage Cluster**, which consists of two types of daemons:

Ceph OSD (Object Storage Daemon)

Ceph OSDs store data on behalf of Ceph clients. Additionally, Ceph OSDs utilize the CPU and memory of Ceph nodes to perform data replication, rebalancing, recovery, monitoring and reporting functions.

Ceph Monitor

A Ceph monitor maintains a master copy of the Ceph storage cluster map with the current state of the storage cluster.

For more information about Red Hat Ceph Storage, see the Red Hat Ceph Storage Architecture Guide.

IMPORTANT

This guide only integrates Ceph Block storage and the Ceph Object Gateway (RGW). It does not include Ceph File (CephFS) storage.

1.2. DEFINING THE SCENARIO

This guide provides instructions for deploying a *containerized* Red Hat Ceph cluster with your overcloud. To do this, the director uses Ansible playbooks provided through the `ceph-ansible` package. The director also manages the configuration and scaling operations of the cluster.

1.3. SETTING REQUIREMENTS

This guide acts as supplementary information for the Director Installation and Usage guide. This means the Requirements section also applies to this guide. Implement these requirements as necessary.

If using the Red Hat OpenStack Platform director to create Ceph Storage nodes, note the following requirements for these nodes:

Processor

64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.

Memory

Memory requirements depend on the amount of storage space. Ideally, use at minimum 1 GB of memory per 1 TB of hard disk space.

Disk Space

Storage requirements depends on the amount of memory. Ideally, use at minimum 1 GB of memory per 1 TB of hard disk space.

Disk Layout

The recommended Red Hat Ceph Storage node configuration requires at least three or more disks in a layout similar to the following:

- **`/dev/sda`** - The root disk. The director copies the main Overcloud image to the disk.
- **`/dev/sdb`** - The journal disk. This disk divides into partitions for Ceph OSD journals. For example, **`/dev/sdb1`**, **`/dev/sdb2`**, **`/dev/sdb3`**, and onward. The journal disk is usually a solid state drive (SSD) to aid with system performance.
- **`/dev/sdc`** and onward - The OSD disks. Use as many disks as necessary for your storage requirements.

> **NOTE**
>
> Red Hat OpenStack Platform director uses **`ceph-ansible`**, which does not support installing the OSD on the root disk of Ceph Storage nodes. This means you need at least two or more disks for a supported Ceph Storage node.

Network Interface Cards

A minimum of one 1 Gbps Network Interface Cards, although it is recommended to use at least two NICs in a production environment. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic. It is recommended to use a 10 Gbps interface for storage node, especially if creating an OpenStack Platform environment that serves a high volume of traffic.

Power Management

Each Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server's motherboard.

This guide also requires the following:

- An Undercloud host with the Red Hat OpenStack Platform director installed. See Installing the Undercloud.
- Any additional hardware recommendation for Red Hat Ceph Storage. See the Red Hat Ceph Storage Hardware Guide for these recommendations.

> **IMPORTANT**
>
> The Ceph Monitor service is installed on the Overcloud's Controller nodes. This means you must provide adequate resources to alleviate performance issues. Ensure the Controller nodes in your environment use at least 16 GB of RAM for memory and solid-state drive (SSD) storage for the Ceph monitor data.

1.4. ADDITIONAL RESOURCES

The **`/usr/share/openstack-tripleo-heat-templates/environments/ceph-`**

ansible/ceph-ansible.yaml environment file instructs the director to use playbooks derived from the ceph-ansible project. These playbooks are installed in **/usr/share/ceph-ansible/** of the undercloud. In particular, the following file lists all the default settings applied by the playbooks:

- **/usr/share/ceph-ansible/group_vars/all.yml.sample**

> **WARNING**
>
> While **ceph-ansible** uses playbooks to deploy containerized Ceph Storage, do not edit these files to customize your deployment. Doing this will result in a failed deployment. Rather, use Heat environment files to override the defaults set by these playbooks.

You can also consult the documentation of this project (http://docs.ceph.com/ceph-ansible/master/) to learn more about the playbook collection.

Alternatively, you can also consult the Heat templates in **/usr/share/openstack-tripleo-heat-templates/docker/services/ceph-ansible/** for information about the default settings applied by director for containerized Ceph Storage.

> **NOTE**
>
> Reading these templates requires a deeper understanding of how environment files and Heat templates work in director. See Understanding Heat Templates and Environment Files for reference.

Lastly, for more information about containerized services in OpenStack, see "Configuring a Basic Overcloud with the CLI Tools" in the Director *Installation and Usage Guide*.

CHAPTER 2. PREPARING OVERCLOUD NODES

All nodes in this scenario are bare metal systems using IPMI for power management. These nodes do not require an operating system because the director copies a Red Hat Enterprise Linux 7 image to each node; in addition, the Ceph Storage services on the nodes described here are containerized. The director communicates to each node through the Provisioning network during the introspection and provisioning processes. All nodes connect to this network through the native VLAN.

2.1. CLEANING CEPH STORAGE NODE DISKS

The Ceph Storage OSDs and journal partitions require GPT disk labels. This means the additional disks on Ceph Storage require conversion to GPT before installing the Ceph OSD services. For this to happen, all metadata must be deleted from the disks; this will allow the director to set GPT labels on them.

You can set the director to delete all disk metadata by default by adding the following setting to your **/home/stack/undercloud.conf** file:

```
clean_nodes=true
```

With this option, the Bare Metal Provisioning service will run an additional step to boot the nodes and *clean* the disks each time the node is set to **available**. This adds an additional power cycle after the first introspection and before each deployment. The Bare Metal Provisioning service uses **wipefs --force --all** to perform the clean.

After setting this option, run the **openstack undercloud install** command to execute this configuration change.

> **WARNING**
>
> The **wipefs --force --all** will delete all data and metadata on the disk, but does not perform a *secure erase*. A secure erase takes much longer.

2.2. REGISTERING NODES

A node definition template (**instackenv.json**) is a JSON format file and contains the hardware and power management details for registering nodes. For example:

```
{
    "nodes":[
        {
            "mac":[
                "b1:b1:b1:b1:b1:b1"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
```

```
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
        },
        {
            "mac":[
                "b2:b2:b2:b2:b2:b2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        },
        {
            "mac":[
                "b3:b3:b3:b3:b3:b3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.207"
        },
        {
            "mac":[
                "c1:c1:c1:c1:c1:c1"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.208"
        },
        {
            "mac":[
                "c2:c2:c2:c2:c2:c2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.209"
        },
        {
```

```
            "mac":[
                "c3:c3:c3:c3:c3:c3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.210"
        },
        {
            "mac":[
                "d1:d1:d1:d1:d1:d1"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.211"
        },
        {
            "mac":[
                "d2:d2:d2:d2:d2:d2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.212"
        },
        {
            "mac":[
                "d3:d3:d3:d3:d3:d3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.213"
        }
    ]
}
```

After creating the template, save the file to the stack user's home directory (**/home/stack/instackenv.json**). Initialize the stack user, then import **instackenv.json** into the director:

```
$ source ~/stackrc
$ openstack overcloud node import ~/instackenv.json
```

This imports the template and registers each node from the template into the director.

Assign the kernel and ramdisk images to each node:

```
$ openstack overcloud node configure <node>
```

The nodes are now registered and configured in the director.

2.3. MANUALLY TAGGING THE NODES

After registering each node, you will need to inspect the hardware and tag the node into a specific profile. Profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role.

To inspect and tag new nodes, follow these steps:

1. Trigger hardware introspection to retrieve the hardware attributes of each node:

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

   - The **--all-manageable** option introspects only nodes in a managed state. In this example, it is all of them.
   - The **--provide** option resets all nodes to an **active** state after introspection.

   > **IMPORTANT**
   >
   > Make sure this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

2. Retrieve a list of your nodes to identify their UUIDs:

   ```
   $ openstack baremetal node list
   ```

3. Add a **profile** option to the **properties/capabilities** parameter for each node to manually tag a node to a specific profile.
   For example, a typical deployment will use three profiles: **control**, **compute**, and **ceph-storage**. The following commands tag three nodes for each profile:

   ```
   $ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
   properties/capabilities='profile:control,boot_option:local'
   $ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
   properties/capabilities='profile:control,boot_option:local'
   $ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
   properties/capabilities='profile:control,boot_option:local'
   $ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
   properties/capabilities='profile:compute,boot_option:local'
   ```

```
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update da0cc61b-4882-45e0-9f43-fab65cf4e52b add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update b9f70722-e124-4650-a9b1-aade8121b5ed add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 68bf8f29-7731-4148-ba16-efb31ab8d34f add
properties/capabilities='profile:ceph-storage,boot_option:local'
```

TIP

You can also configure a new custom profile that will allow you to tag a node for the Ceph MON and Ceph MDS services. See Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes* for details.

The addition of the **`profile`** option tags the nodes into each respective profiles.

NOTE

As an alternative to manual tagging, use the Automated Health Check (AHC) Tools to automatically tag larger numbers of nodes based on benchmarking data.

2.4. DEFINING THE ROOT DISK FOR CEPH STORAGE NODES

Most Ceph Storage nodes use multiple disks. This means the director needs to identify the disk to use for the root disk when provisioning a Ceph Storage node.

There are several properties you can use to help the director identify the root disk:

- **`model`** (String): Device identifier.
- **`vendor`** (String): Device vendor.
- **`serial`** (String): Disk serial number.
- **`hctl`** (String): Host:Channel:Target:Lun for SCSI.
- **`size`** (Integer): Size of the device in GB.
- **`wwn`** (String): Unique storage identifier.
- **`wwn_with_extension`** (String): Unique storage identifier with the vendor extension appended.
- **`wwn_vendor_extension`** (String): Unique vendor storage identifier.
- **`rotational`** (Boolean): True for a rotational device (HDD), otherwise false (SSD).
- **`name`** (String): The name of the device, for example: /dev/sdb1.

IMPORTANT

Only use **name** for devices with persistent names. Do not use **name** to set the root disk for other device because this value can change when the node boots.

In this example, you specify the drive to deploy the overcloud image using the serial number of the disk to determine the root device.

Check the disk information from each node's hardware introspection. The following command displays the disk information from a node:

```
(undercloud) $ openstack baremetal introspection data save 1a4e30da-b6dc-
499d-ba87-0bd8a3819bc0 | jq ".inventory.disks"
```

For example, the data for one node might show three disks:


```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
```


For this example, set the root device to disk 2, which has **61866da04f380d001ea4e13c12e36ad6** as the serial number. This requires a change to the **root_device** parameter for the node definition:

```
(undercloud) $ openstack baremetal node set --property
root_device='{"serial": "61866da04f380d001ea4e13c12e36ad6"}' 1a4e30da-
b6dc-499d-ba87-0bd8a3819bc0
```

NOTE

Make sure to configure the BIOS of each node to include booting from the chosen root disk. The recommended boot order is network boot, then root disk boot.

This helps the director identify the specific disk to use as the root disk. When we initiate our Overcloud creation, the director provisions this node and writes the Overcloud image to this disk. The other disks are used for mapping our Ceph Storage nodes.

CHAPTER 3. DEPLOYING OTHER CEPH SERVICES ON DEDICATED NODES

By default, the director deploys the Ceph MON and Ceph MDS services on the Controller nodes. This is suitable for small deployments. However, with larger deployments we advise that you deploy the Ceph MON and Ceph MDS services on dedicated nodes to improve the performance of your Ceph cluster. You can do this by creating a *custom role* for either one.

> **NOTE**
>
> For detailed information about custom roles, see Creating a New Role from the Advanced Overcloud Customization guide.

The director uses the following file as a default reference for all overcloud roles:

- **`/usr/share/openstack-tripleo-heat-templates/roles_data.yaml`**

Copy this file to **`/home/stack/templates/`** so you can add custom roles to it:

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

You invoke the **`/home/stack/templates/roles_data_custom.yaml`** file later during overcloud creation (Section 6.2, "Initiating Overcloud Deployment"). The following sub-sections describe how to configure custom roles for either Ceph MON and Ceph MDS services.

3.1. CREATING A CUSTOM ROLE AND FLAVOR FOR THE CEPH MON SERVICE

This section describes how to create a custom role (named **`CephMon`**) and flavor (named **`ceph-mon`**) for the Ceph MON role. You should already have a copy of the default roles data file as described in Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes*.

1. Open the **`/home/stack/templates/roles_data_custom.yaml`** file.

2. Remove the service entry for the Ceph MON service (namely, **OS::TripleO::Services::CephMon**) from under the Controller role.

3. Add the **OS::TripleO::Services::CephClient** service to the Controller role:

```
[...]
- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMds
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
[...]
```

4. At the end of **roles_data_custom.yaml**, add a custom **CephMon** role containing the Ceph MON service and all the other required node services. For example:

```
- name: CephMon
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephMon
```

5. Using the **openstack flavor create** command, define a new flavor named **ceph-mon** for this role:

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

NOTE

For more details about this command, run **openstack flavor create --help**.

6. Map this flavor to a new profile, also named **ceph-mon**:

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-mon" ceph-mon
```

NOTE

For more details about this command, run **openstack flavor set --help**.

7. Tag nodes into the new **ceph-mon** profile:

```
$ ironic node-update UUID add properties/capabilities='profile:ceph-
mon,boot_option:local'
```

See Section 2.3, “Manually Tagging the Nodes” for more details about tagging nodes. See also Tagging Nodes Into Profiles for related information on custom role profiles.

3.2. CREATING A CUSTOM ROLE AND FLAVOR FOR THE CEPH MDS SERVICE

This section describes how to create a custom role (named **CephMDS**) and flavor (named **ceph-mds**) for the Ceph MDS role. You should already have a copy of the default roles data file as described in Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes*.

1. Open the **/home/stack/templates/roles_data_custom.yaml** file.

2. Remove the service entry for the Ceph MDS service (namely, **OS::TripleO::Services::CephMds**) from under the Controller role:

```
[...]
- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    # - OS::TripleO::Services::CephMds 1
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
[...]
```

1 Comment out this line. This same service will be added to a custom role in the next step.

3. At the end of **roles_data_custom.yaml**, add a custom **CephMDS** role containing the Ceph MDS service and all the other required node services. For example:

```
- name: CephMDS
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephMds
    - OS::TripleO::Services::CephClient 1
```

1. The Ceph MDS service requires the admin keyring, which can be set by either Ceph MON or Ceph Client service. As we are deploying Ceph MDS on a dedicated node (without the Ceph MON service), include the Ceph Client service on the role as well.

4. Using the **openstack flavor create** command, define a new flavor named **ceph-mds** for this role:

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

NOTE

For more details about this command, run **openstack flavor create --help**.

5. Map this flavor to a new profile, also named **ceph-mds**:

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-mds" ceph-mds
```

NOTE

For more details about this command, run **openstack flavor set --help**.

Tag nodes into the new **ceph-mds** profile:

```
$ ironic node-update UUID add properties/capabilities='profile:ceph-
mds,boot_option:local'
```

See Section 2.3, “Manually Tagging the Nodes” for more details about tagging nodes. See also Tagging Nodes Into Profiles for related information on custom role profiles.

CHAPTER 4. CUSTOMIZING THE STORAGE SERVICE

The Heat template collection provided by the director already contains the necessary templates and environment files to enable a basic Ceph Storage configuration.

The **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** environment file will create a Ceph cluster and integrate it with your overcloud upon deployment. This cluster will feature *containerized* Ceph Storage nodes. For more information about containerized services in OpenStack, see "Configuring a Basic Overcloud with the CLI Tools" in the Director *Installation and Usage Guide*.

The Red Hat OpenStack director will also apply basic, default settings to the deployed Ceph cluster. You need a *custom environment file* to pass custom settings to your Ceph cluster. To create one:

1. Create the file **storage-config.yaml** in **/home/stack/templates/**. For the purposes of this document, **~/templates/storage-config.yaml** will contain most of the overcloud-related custom settings for your environment. It will override all the default settings applied by the director to your overcloud.

2. Add a **parameter_defaults** section to **~/templates/storage-config.yaml**. This section will contain custom settings for your overcloud. For example, to set **vxlan** as the network type of the networking service (**neutron**):

```
parameter_defaults:
  NeutronNetworkType: vxlan
```

3. If needed, set the following options under **parameter_defaults** as you see fit:

Option	Description	Default value
CinderEnableIscsiBackend	Enables the iSCSI backend	false
CinderEnableRbdBackend	Enables the Ceph Storage back end	true
CinderBackupBackend	Sets ceph or swift as the back end for volume backups; see Section 4.3, "Configuring the Backup Service to Use Ceph" for related details	ceph
NovaEnableRbdBackend	Enables Ceph Storage for Nova ephemeral storage	true
GlanceBackend	Defines which back end the Image service should use: **rbd** (Ceph), **swift**, or **file**	rbd
GnocchiBackend	Defines which back end the Telemetry service should use: **rbd** (Ceph), **swift**, or **file**	rbd

NOTE

You can omit an option from **~/templates/storage-config.yaml** if you intend to use the default setting.

The contents of your environment file will change depending on the settings you apply in the sections that follow. See Appendix A, *Sample Environment File: Creating a Ceph Cluster* for a finished example.

The following subsections explain how to override common default storage service settings applied by the director.

4.1. ENABLING THE CEPH METADATA SERVER (MDS) [TECHNOLOGY PREVIEW]

IMPORTANT

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

The *Ceph Metadata Server* (MDS) runs the **ceph-mds** daemon, which manages metadata related to files stored on the Ceph File System (CephFS). For related information about CephFS, see Ceph File System Guide (Technology Preview) and CephFS Back End Guide for the Shared File System Service.

To enable the Ceph Metadata Server, invoke the following environment file when creating your overcloud:

- **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-mds.yaml**

See Section 6.2, "Initiating Overcloud Deployment" for more details. For more information about the Ceph Metadata Server, see Installing and Configuring Ceph Metadata Servers (MDS).

NOTE

By default, the Ceph Metadata Server will be deployed on the Controller node. You can deploy the Ceph Metadata Server on its own dedicated node; for instructions, see Section 3.2, "Creating a Custom Role and Flavor for the Ceph MDS Service".

4.2. ENABLING THE CEPH OBJECT GATEWAY

The *Ceph Object Gateway* provides applications with an interface to object storage capabilities within a Ceph storage cluster. Upon deploying the Ceph Object Gateway, you can then replace the default Object Storage service (**swift**) with Ceph. For more information, see Object Gateway Guide for Red Hat Enterprise Linux.

To enable a Ceph Object Gateway in your deployment, invoke the following environment file when creating your overcloud:

- **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-rgw.yaml**

See Section 6.2, “Initiating Overcloud Deployment” for more details.

The Ceph Object Gateway acts as a drop-in replacement for the default Object Storage service. As such, all other services that normally use **swift** can seamlessly start using the Ceph Object Gateway instead without further configuration. For example, when configuring the Block Storage Backup service (**cinder-backup**) to use the Ceph Object Gateway, set **ceph** as the target back end (see Section 4.3, “Configuring the Backup Service to Use Ceph”).

4.3. CONFIGURING THE BACKUP SERVICE TO USE CEPH

The Block Storage Backup service (**cinder-backup**) is disabled by default. To enable it, invoke the following environment file when creating your overcloud:

- **/usr/share/openstack-tripleo-heat-templates/environments/cinder-backup.yaml**

See Section 6.2, “Initiating Overcloud Deployment” for more details.

When you enable **cinder-backup** (as in Section 4.2, “Enabling the Ceph Object Gateway”), you can configure it to store backups in Ceph. This involves adding the following line to the **parameter_defaults** of your environment file (namely, **~/templates/storage-config.yaml**):

```
CinderBackupBackend: ceph
```

4.4. CONFIGURING MULTIPLE BONDED INTERFACES PER CEPH NODE

Using a *bonded interface* allows you to combine multiple NICs to add redundancy to a network connection. If you have enough NICs on your Ceph nodes, you can take this a step further by creating *multiple bonded interfaces* per node.

With this, you can then use a bonded interface for *each* network connection required by the node. This provides both redundancy and a dedicated connection for each network.

The simplest implementation of this involves the use of two bonds, one for each storage network used by the Ceph nodes. These networks are the following:

Front-end storage network (StorageNet)

The Ceph client uses this network to interact with its Ceph cluster.

Back-end storage network (StorageMgmtNet)

The Ceph cluster uses this network to balance data in accordance with the *placement group* policy of the cluster. For more information, see Placement Groups (PG) (from the Red Hat Ceph Architecture Guide).

Configuring this involves customizing a network interface template, as the director does not provide any sample templates that deploy multiple bonded NICs. However, the director does provide a template that deploys a single bonded interface — namely, **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml**. You can add a bonded interface for your additional NICs by defining it there.

> **NOTE**
>
> For more detailed instructions on how to do this, see Creating Custom Interface Templates (from the Advanced Overcloud Customization guide). That section also explains the different components of a bridge and bonding definition.

The following snippet contains the default definition for the single bonded interface defined by **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml**:

```
type: ovs_bridge // 1
name: br-bond
members:
  -
    type: ovs_bond // 2
    name: bond1 // 3
    ovs_options: {get_param: BondInterfaceOvsOptions} 4
    members: // 5
      -
        type: interface
        name: nic2
        primary: true
      -
        type: interface
        name: nic3
  -
    type: vlan // 6
    device: bond1 // 7
    vlan_id: {get_param: StorageNetworkVlanID}
    addresses:
      -
        ip_netmask: {get_param: StorageIpSubnet}
  -
    type: vlan
    device: bond1
    vlan_id: {get_param: StorageMgmtNetworkVlanID}
    addresses:
      -
        ip_netmask: {get_param: StorageMgmtIpSubnet}
```

1. A single bridge named **br-bond** holds the bond defined by this template. This line defines the bridge type, namely OVS.
2. The first member of the **br-bond** bridge is the bonded interface itself, named **bond1**. This line defines the bond type of **bond1**, which is also OVS.
3. The default bond is named **bond1**, as defined in this line.
4. The **ovs_options** entry instructs director to use a specific set of *bonding module directives*. Those directives are passed through the **BondInterfaceOvsOptions**, which you can also configure in this same file. For instructions on how to configure this, see Section 4.4.1, “Configuring Bonding Module Directives”.
5.

The **members** section of the bond defines which network interfaces are bonded by **bond1**. In this case, the bonded interface uses **nic2** (set as the primary interface) and **nic3**.

6 The **br-bond** bridge has two other members: namely, a VLAN for both front-end (**StorageNetwork**) and back-end (**StorageMgmtNetwork**) storage networks.

7 The **device** parameter defines what device a VLAN should use. In this case, both VLANs will use the bonded interface **bond1**.

With at least two more NICs, you can define an additional bridge and bonded interface. Then, you can move one of the VLANs to the new bonded interface. This results in added throughput and reliability for both storage network connections.

When customizing **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml** for this purpose, it is advisable to also use Linux bonds (**type: linux_bond**) instead of the default OVS (**type: ovs_bond**). This bond type is more suitable for enterprise production deployments.

The following edited snippet defines an additional OVS bridge (**br-bond2**) which houses a new Linux bond named **bond2**. The **bond2** interface uses two additional NICs (namely, **nic4** and **nic5**) and will be used solely for back-end storage network traffic:

```
  type: ovs_bridge
  name: br-bond
  members:
    -
      type: linux_bond
      name: bond1
      bonding_options: {get_param: BondInterfaceOvsOptions} // 1
      members:
        -
          type: interface
          name: nic2
          primary: true
        -
          type: interface
          name: nic3
    -
      type: vlan
      device: bond1
      vlan_id: {get_param: StorageNetworkVlanID}
      addresses:
        -
          ip_netmask: {get_param: StorageIpSubnet}
-
  type: ovs_bridge
  name: br-bond2
  members:
    -
      type: linux_bond
      name: bond2
      bonding_options: {get_param: BondInterfaceOvsOptions}
      members:
        -
          type: interface
```

```
          name: nic4
          primary: true
        -
          type: interface
          name: nic5
    -
      type: vlan
      device: bond1
      vlan_id: {get_param: StorageMgmtNetworkVlanID}
      addresses:
        -
          ip_netmask: {get_param: StorageMgmtIpSubnet}
```

1. As **bond1** and **bond2** are both Linux bonds (instead of OVS), they use **bonding_options** instead of **ovs_options** to set bonding directives. For related information, see Section 4.4.1, "Configuring Bonding Module Directives".

For the full contents of this customized template, see Appendix B, *Sample Custom Interface Template: Multiple Bonded Interfaces*.

4.4.1. Configuring Bonding Module Directives

After adding and configuring the bonded interfaces, use the **BondInterfaceOvsOptions** parameter to set what directives each should use. You can find this in the **parameters:** section of **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml**. The following snippet shows the default definition of this parameter (namely, empty):

```
BondInterfaceOvsOptions:
    default: ''
    description: The ovs_options string for the bond interface. Set
                 things like lacp=active and/or bond_mode=balance-slb
                 using this option.
    type: string
```

Define the options you need in the **default:** line. For example, to use 802.3ad (mode 4) and a LACP rate of 1 (fast), use **'mode=4 lacp_rate=1'**, as in:

```
BondInterfaceOvsOptions:
    default: 'mode=4 lacp_rate=1'
    description: The bonding_options string for the bond interface. Set
                 things like lacp=active and/or bond_mode=balance-slb
                 using this option.
    type: string
```

See Appendix C. Open vSwitch Bonding Options (from the Advanced Overcloud Optimization guide) for other supported bonding options. For the full contents of the customized **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml** template, see Appendix B, *Sample Custom Interface Template: Multiple Bonded Interfaces*.

CHAPTER 5. CUSTOMIZING THE CEPH STORAGE CLUSTER

Deploying containerized Ceph Storage involves the use of **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** during overcloud deployment (as described in Chapter 4, *Customizing the Storage Service*). This environment file also defines the following resources:

- **CephAnsibleDisksConfig** - used to map the Ceph Storage node disk layout. See Section 5.1, "Mapping the Ceph Storage Node Disk Layout" for more details.
- **CephConfigOverrides** - used to apply all other custom settings to your Ceph cluster.

Use these resources to override any defaults set by the director for containerized Ceph Storage.

The **/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** environment file uses playbooks provided by the **ceph-ansible** package. As such, you need to install this package on your undercloud first:

```
$ sudo yum install ceph-ansible
```

To customize your Ceph cluster, define your custom parameters in a new environment file, namely **/home/stack/templates/ceph-config.yaml**. You can arbitrarily apply global Ceph cluster settings using the following syntax in the **parameter_defaults** section of your environment file:

```
parameter_defaults:
  CephConfigOverrides:
    KEY: VALUE
```

Replace *KEY*: *VALUE* with the Ceph cluster settings you want to apply. For example, consider the following snippet:

For example, consider the following snippet:

```
parameter_defaults:
  CephConfigOverrides
    journal_size: 2048
    max_open_files: 131072
```

This will result in the following settings defined in the configuration file of your Ceph cluster:

```
[global]
journal_size: 2048
max_open_files: 131072
```

NOTE

See the Red Hat Ceph Storage Configuration Guide for detailed information about supported parameters.

5.1. MAPPING THE CEPH STORAGE NODE DISK LAYOUT

When you deploy containerized Ceph Storage, you need to map the disk layout and specify dedicated block devices for the Ceph OSD service. You can do this in the environment file you created earlier to define your custom Ceph parameters — namely, **`/home/stack/templates/ceph-config.yaml`**.

Use the **`CephAnsibleDisksConfig`** resource in **`parameter_defaults`** to map your disk layout. This resource uses the following variables:

Variable	Required?	Default value (if unset)	Description
osd_scenario	Yes	collocated	Sets the journaling scenario; as in, whether OSDs should be created with journals that are either: - co-located on the same device (**`collocated`**), or - stored on dedicated devices (**`non-collocated`**).
devices	Yes	*NONE. Variable must be set.*	A list of block devices to be used on the node for OSDs.
dedicated_devices	Yes (only if **`osd_scenario`** is **`non-collocated`**)	devices	A list of block devices that maps each entry under devices to a dedicated journaling block device. This variable is only usable when **`osd_scenario=non-collocated`**.
dmcrypt	No	false	Sets whether data stored on OSDs are encrypted (**`true`**) or not (**`false`**).
osd_objectstore	No	filestore	Sets the storage back end used by Ceph. Currently, Red Hat Ceph only supports **`filestore`**.

IMPORTANT

The default journaling scenario is set to **osd_scenario=collocated**, which has lower hardware requirements consistent with most testing environments. In a typical production environment, however, journals are stored on dedicated devices (**osd_scenario=non-collocated**) to accommodate heavier I/O workloads. For related information, see Identifying a Performance Use Case.

List each block device to be used by OSDs as a simple list under the **devices** variable. For example:

```
devices:
  - /dev/sda
  - /dev/sdb
  - /dev/sdc
  - /dev/sdd
```

If **osd_scenario=non-collocated**, you must also map each entry in devices to a corresponding entry in dedicated_devices. For example, given the following snippet in **/home/stack/templates/ceph-config.yaml**:

```
osd_scenario: non-collocated
devices:
  - /dev/sda
  - /dev/sdb
  - /dev/sdc
  - /dev/sdd

dedicated_devices:
  - /dev/sdf
  - /dev/sdf
  - /dev/sdg
  - /dev/sdg

dmcrypt: true
```

Each Ceph Storage node in the resulting Ceph cluster would have the following characteristics:

- **/dev/sda** will have **/dev/sdf1** as its journal
- **/dev/sdb** will have **/dev/sdf2** as its journal
- **/dev/sdc** will have **/dev/sdg1** as its journal
- **/dev/sdd** will have **/dev/sdg2** as its journal
- Data stored on OSDs will be encrypted.

In some nodes, disk paths (for example, **/dev/sdb**, **/dev/sdc**) may not point to the exact same block device during reboots. If this is the case with your **CephStorage** nodes, specify each disk through its **/dev/disk/by-path/** symlink. For example:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
```

```
      - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:10:0
      - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:11:0


    dedicated_devices
      - /dev/nvme0n1
      - /dev/nvme0n1
```

This ensures that the block device mapping is consistent throughout deployments.

Because the list of OSD devices must be set prior to overcloud deployment, it may not be possible to identify and set the PCI path of disk devices. In this case, deploy the node without using the disks in question (as a compute node, for example) and run the following command on the deployed node. Use the output to define the PCI path of the disk device.

```
[root@overcloud-novacompute-0 ~]# ls -l /dev/disk/by-path/
total 0
lrwxrwxrwx. 1 root root  9 Jul 11 20:12 pci-0000:00:11.5-ata-1.0 ->
../../sda
lrwxrwxrwx. 1 root root 10 Jul 11 20:12 pci-0000:00:11.5-ata-1.0-part1 ->
../../sda1
lrwxrwxrwx. 1 root root  9 Jul 11 20:12 pci-0000:00:11.5-ata-2.0 ->
../../sdb
lrwxrwxrwx. 1 root root  9 Jul 11 20:12 pci-0000:00:11.5-ata-3.0 ->
../../sdc
lrwxrwxrwx. 1 root root  9 Jul 11 20:12 pci-0000:00:11.5-ata-4.0 ->
../../sdd
lrwxrwxrwx. 1 root root 13 Jul 11 20:12 pci-0000:1a:00.0-nvme-1 ->
../../nvme0n1
[root@overcloud-novacompute-0 ~]#
```

For more information about naming conventions for storage devices, see Persistent Naming.

For more details about each journaling scenario and disk mapping for containerized Ceph Storage, see the OSD Scenarios section of the project documentation for ceph-ansible.

5.2. ASSIGNING CUSTOM ATTRIBUTES TO DIFFERENT CEPH POOLS

By default, Ceph pools created through the director have the same placement group (**pg_num** and **pgp_num**) and sizes. You can use either method in Chapter 5, *Customizing the Ceph Storage Cluster* to override these settings globally; that is, doing so will apply the same values for all pools.

You can also apply different attributes to each Ceph pool. To do so, use the **CephPools** parameter, as in:

```
parameter_defaults:
  CephPools:
    - name: POOL
      pg_num: 128
```

Replace *POOL* with the name of the pool you want to configure along with the **pg_num** setting to indicate number of placement groups. This overrides the default **pg_num** for the specified pool.

You can also create new custom pools through the **CephPools** parameter. For example, to add a pool called **custompool**:

```
parameter_defaults:
  CephPools:
    - name: custompool
      pg_num: 128
```

This creates a new custom pool in addition to the default pools.

TIP

For typical pool configurations of common Ceph use cases, see the Ceph Placement Groups (PGs) per Pool Calculator. This calculator is normally used to generate the commands for manually configuring your Ceph pools. In this deployment, the director will configure the pools based on your specifications.

CHAPTER 6. CREATING THE OVERCLOUD

Once your custom environment files are ready, you can specify which flavors/nodes each node should use and then execute the deployment. The following subsections explain both steps in greater detail.

6.1. ASSIGNING NODES AND FLAVORS TO ROLES

Planning an overcloud deployment involves specifying how many nodes and which flavors to assign to each role. Like all Heat template parameters, these role specifications are declared in the **`parameter_defaults`** section of your environment file (in this case, **`~/templates/storage-config.yaml`**).

For this purpose, use the following parameters:

Table 6.1. Roles and Flavors for Overcloud Nodes

Heat Template Parameter	Description
ControllerCount	The number of Controller nodes to scale out
OvercloudControlFlavor	The flavor to use for Controller nodes (**`control`**)
ComputeCount	The number of Compute nodes to scale out
OvercloudComputeFlavor	The flavor to use for Compute nodes (**`compute`**)
CephStorageCount	The number of Ceph storage (OSD) nodes to scale out
OvercloudCephStorageFlavor	The flavor to use for Ceph Storage (OSD) nodes (**`ceph-storage`**)
CephMonCount	The number of dedicated Ceph MON nodes to scale out
OvercloudCephMonFlavor	The flavor to use for dedicated Ceph MON nodes (**`ceph-mon`**)
CephMdsCount	The number of dedicated Ceph MDS nodes to scale out
OvercloudCephMdsFlavor	The flavor to use for dedicated Ceph MDS nodes (**`ceph-mds`**)

IMPORTANT

The **`CephMonCount`**, **`CephMdsCount`**, **`OvercloudCephMonFlavor`**, and **`OvercloudCephMdsFlavor`** parameters (along with the **`ceph-mon`** and **`ceph-mds`** flavors) will only be valid if you created a custom **`CephMON`** and **`CephMds`** role, as described in Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes*.

For example, to configure the overcloud to deploy three nodes for each role (Controller, Compute, Ceph-Storage, and CephMon), add the following to your **parameter_defaults**:

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
  CephMdsCount: 3
  OvercloudCephMdsFlavor: ceph-mds
```

NOTE

See Creating the Overcloud with the CLI Tools from the Director Installation and Usage guide for a more complete list of Heat template parameters.

6.2. INITIATING OVERCLOUD DEPLOYMENT

The creation of the Overcloud requires additional arguments for the **openstack overcloud deploy** command. For example:

```
$ openstack overcloud deploy --templates -r
/home/stack/templates/roles_data_custom.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-ansible.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-rgw.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-mds.yaml
  -e /usr/share/openstack-tripleo-heat-templates/environments/cinder-
backup.yaml \
  -e /home/stack/templates/storage-config.yaml \
  -e /home/stack/templates/ceph-config.yaml \
  --ntp-server pool.ntp.org
```

The above command uses the following options:

- **--templates** - Creates the Overcloud from the default Heat template collection (namely, **/usr/share/openstack-tripleo-heat-templates/**).
- **-r /home/stack/templates/roles_data_custom.yaml** - Specifies the customized roles definition file from Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes*, which adds custom roles for either Ceph MON or Ceph MDS services. These roles allow either service to be installed on dedicated nodes.
- **-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml** - Sets the director to create a Ceph cluster. In particular, this environment file will deploy a Ceph cluster with *containerized* Ceph Storage nodes.

- **-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-rgw.yaml** - Enables the Ceph Object Gateway, as described in Section 4.2, "Enabling the Ceph Object Gateway".
- **-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-mds.yaml** - Enables the Ceph Metadata Server, as described in Section 4.1, "Enabling the Ceph Metadata Server (MDS) [Technology Preview]".
- **-e /usr/share/openstack-tripleo-heat-templates/environments/cinder-backup.yaml** - Enables the Block Storage Backup service (**cinder-backup**), as described in Section 4.3, "Configuring the Backup Service to Use Ceph".
- **-e /home/stack/templates/storage-config.yaml** - Adds the environment file containing your custom Ceph Storage configuration.
- **-e /home/stack/templates/ceph-config.yaml** - Adds the environment file containing your custom Ceph cluster settings, as described in Chapter 5, *Customizing the Ceph Storage Cluster*.
- **--ntp-server pool.ntp.org** - Sets our NTP server.

TIP

You can also use an *answers file* to invoke all your templates and environment files. For example, you can use the following command to deploy an identical overcloud:

```
$ openstack overcloud deploy -r
/home/stack/templates/roles_data_custom.yaml \
  --answers-file /home/stack/templates/answers.yaml --ntp-server
pool.ntp.org
```

In this case, the answers file **/home/stack/templates/answers.yaml** contains:

```
templates: /usr/share/openstack-tripleo-heat-templates/
environments:
  - /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-ansible.yaml
  - /usr/share/openstack-tripleo-heat-templates/environments/ceph-rgw.yaml
  - /usr/share/openstack-tripleo-heat-templates/environments/ceph-mds.yaml
  - /usr/share/openstack-tripleo-heat-templates/environments/cinder-
backup.yaml
  - /home/stack/templates/storage-config.yaml
  - /home/stack/templates/ceph-config.yaml
```

See Including Environment Files in Overcloud Creation for more details.

For a full list of options, run:

```
$ openstack help overcloud deploy
```

For more information, see Creating the Overcloud with the CLI Tools in the Director Installation and Usage guide.

The Overcloud creation process begins and the director provisions your nodes. This process takes some time to complete. To view the status of the Overcloud creation, open a separate terminal as the **stack** user and run:

```
$ source ~/stackrc
$ openstack stack list --nested
```

CHAPTER 7. POST-DEPLOYMENT

The following subsections describe several post-deployment operations for managing the Ceph cluster.

7.1. ACCESSING THE OVERCLOUD

The director generates a script to configure and help authenticate interactions with your Overcloud from the director host. The director saves this file (**overcloudrc**) in your **stack** user's home directory. Run the following command to use this file:

```
$ source ~/overcloudrc
```

This loads the necessary environment variables to interact with your Overcloud from the director host's CLI. To return to interacting with the director's host, run the following command:

```
$ source ~/stackrc
```

7.2. MONITORING CEPH STORAGE NODES

After completing the Overcloud creation, we recommend that you check the status of the Ceph Storage Cluster to ensure it is working properly. To do this, log into a Controller node as the **heat-admin** user:

```
$ nova list
$ ssh heat-admin@192.168.0.25
```

Check the health of the cluster:

```
$ sudo ceph health
```

If the cluster has no issues, the command reports back **HEALTH_OK**. This means the cluster is safe to use.

Check the status of the Ceph Monitor quorum:

```
$ sudo ceph quorum_status
```

This shows the monitors participating in the quorum and which one is the leader.

Check if all Ceph OSDs are running:

```
$ ceph osd stat
```

For more information on monitoring Ceph Storage clusters, see Monitoring in the *Red Hat Ceph Storage Administration Guide*.

7.3. REBOOTING THE ENVIRONMENT

A situation might occur where you need to reboot the environment. For example, when you might need to modify the physical servers, or you might need to recover from a power outage. In this situation, it is important to make sure your Ceph Storage nodes boot correctly.

Make sure to boot the nodes in the following order:

- **Boot all Ceph Monitor nodes first** - This ensures the Ceph Monitor service is active in your high availability cluster. By default, the Ceph Monitor service is installed on the Controller node. If the Ceph Monitor is separate from the Controller in a custom role, make sure this custom Ceph Monitor role is active.
- **Boot all Ceph Storage nodes** - This ensures the Ceph OSD cluster can connect to the active Ceph Monitor cluster on the Controller nodes.

Use the following process to reboot the Ceph Storage nodes:

1. Log into a Ceph MON or Controller node and disable Ceph Storage cluster rebalancing temporarily:

   ```
   $ sudo ceph osd set noout
   $ sudo ceph osd set norebalance
   ```

2. Select the first Ceph Storage node to reboot and log into it.

3. Reboot the node:

   ```
   $ sudo reboot
   ```

4. Wait until the node boots.

5. Log into the node and check the cluster status:

   ```
   $ sudo ceph -s
   ```

   Check that the **`pgmap`** reports all **`pgs`** as normal (**`active+clean`**).

6. Log out of the node, reboot the next node, and check its status. Repeat this process until you have rebooted all Ceph storage nodes.

7. When complete, log into a Ceph MON or Controller node and enable cluster rebalancing again:

   ```
   $ sudo ceph osd unset noout
   $ sudo ceph osd unset norebalance
   ```

8. Perform a final status check to verify the cluster reports **`HEALTH_OK`**:

   ```
   $ sudo ceph status
   ```

If a situation occurs where all Overcloud nodes boot at the same time, the Ceph OSD services might not start correctly on the Ceph Storage nodes. In this situation, reboot the Ceph Storage OSDs so they can connect to the Ceph Monitor service.

Verify a **`HEALTH_OK`** status of the Ceph Storage node cluster with the following command:

```
$ sudo ceph status
```

7.4. SCALING UP THE CEPH CLUSTER

You can scale up the number of Ceph Storage nodes in your overcloud by re-running the deployment with the number of Ceph Storage nodes you need.

Before doing so, ensure that you have enough nodes for the updated deployment. These nodes must be registered with the director and tagged accordingly.

Registering New Ceph Storage Nodes

To register new Ceph storage nodes with the director, follow these steps:

1. Log into the director host as the **stack** user and initialize your director configuration:

```
$ source ~/stackrc
```

2. Define the hardware and power management details for the new nodes in a new node definition template; for example, **instackenv-scale.json**.

3. Import this file to the OpenStack director:

```
$ openstack overcloud node import ~/instackenv-scale.json
```

Importing the node definition template registers each node defined there to the director.

4. Assign the kernel and ramdisk images to all nodes:

```
$ openstack overcloud node configure
```

NOTE

For more information about registering new nodes, see Section 2.2, “Registering Nodes”.

Manually Tagging New Nodes

After registering each node, you will need to inspect the hardware and tag the node into a specific profile. Profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role.

To inspect and tag new nodes, follow these steps:

1. Trigger hardware introspection to retrieve the hardware attributes of each node:

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

- The **--all-manageable** option introspects only nodes in a managed state. In this example, it is all of them.
- The **--provide** option resets all nodes to an **active** state after introspection.

IMPORTANT

Make sure this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

2. Retrieve a list of your nodes to identify their UUIDs:

```
$ openstack baremetal node list
```

3. Add a **profile** option to the **`properties/capabilities`** parameter for each node to manually tag a node to a specific profile.
   For example, the following commands tag three additional nodes with the **`ceph-storage`** profile:

```
$ ironic node-update 551d81f5-4df2-4e0f-93da-6c5de0b868f7 add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 5e735154-bd6b-42dd-9cc2-b6195c4196d7 add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 1a2b090c-299d-4c20-a25d-57dd21a7085b add
properties/capabilities='profile:ceph-storage,boot_option:local'
```

TIP

If the nodes you just tagged and registered use multiple disks, you can set the director to use a specific root disk on each node. See Section 2.4, “Defining the Root Disk for Ceph Storage Nodes” for instructions on how to do so.

Re-deploying the Overcloud with Additional Ceph Storage Nodes

After registering and tagging the new nodes, you can now scale up the number of Ceph Storage nodes by re-deploying the overcloud. When you do, set the **`CephStorageCount`** parameter in the **`parameter_defaults`** of your environment file (in this case, **`~/templates/storage-config.yaml`**). In Section 6.1, “Assigning Nodes and Flavors to Roles”, the overcloud is configured to deploy with 3 Ceph Storage nodes. To scale it up to 6 nodes instead, use:

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 6
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
```

Upon re-deployment with this setting, the overcloud should now have 6 Ceph Storage nodes instead of 3.

7.5. SCALING DOWN AND REPLACING CEPH STORAGE NODES

In some cases, you may need to scale down your Ceph cluster, or even replace a Ceph Storage node (for example, if a Ceph Storage node is faulty). In either situation, you need to disable and rebalance any Ceph Storage node you are removing from the Overcloud to ensure no data loss. This procedure explains the process for replacing a Ceph Storage node.

NOTE

This procedure uses steps from the *Red Hat Ceph Storage Administration Guide* to manually remove Ceph Storage nodes. For more in-depth information about manual removal of Ceph Storage nodes, see Adding and Removing OSD Nodes from the *Red Hat Ceph Storage Administration Guide*.

Log into either a Controller node or a Ceph Storage node as the **heat-admin** user. The director's **stack** user has an SSH key to access the **heat-admin** user.

List the OSD tree and find the OSDs for your node. For example, your node to remove might contain the following OSDs:

```
-2 0.09998     host overcloud-cephstorage-0
0 0.04999         osd.0                         up  1.00000
1.00000
1 0.04999         osd.1                         up  1.00000
1.00000
```

Disable the OSDs on the Ceph Storage node. In this case, the OSD IDs are 0 and 1.

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 0
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 1
```

The Ceph Storage cluster begins rebalancing. Wait for this process to complete. You can follow the status using the following command:

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph -w
```

Once the Ceph cluster completes rebalancing, log into the Ceph Storage node you are removing (in this case, **overcloud-cephstorage-0**) as the **heat-admin** user and stop the node.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo systemctl disable ceph-osd@0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo systemctl disable ceph-osd@1
```

While logged into **overcloud-cephstorage-0**, remove it from the CRUSH map so that it no longer receives data.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove osd.1
```

Remove the OSD authentication key.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.1
```

Remove the OSD from the cluster.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 1
```

Leave the node and return to the director host as the **stack** user.

```
[heat-admin@overcloud-cephstorage-0 ~]$ exit
[stack@director ~]$
```

Disable the Ceph Storage node so the director does not reprovision it.

```
[stack@director ~]$ ironic node-list
[stack@director ~]$ ironic node-set-maintenance UUID true
```

Removing a Ceph Storage node requires an update to the **overcloud** stack in the director using the local template files. First identify the UUID of the Overcloud stack:

```
$ heat stack-list
```

Identify the UUIDs of the Ceph Storage node to delete:

```
$ nova list
```

Run the following command to delete the node from the stack and update the plan accordingly:

```
$ openstack overcloud node delete --stack STACK_UUID --templates -e
ENVIRONMENT_FILE NODE_UUID
```

IMPORTANT

If you passed any extra environment files when you created the overcloud, pass them again here using the **-e** option to avoid making undesired changes to the overcloud. For more information, see Modifying the Overcloud Environment (from Director Installation and Usage).

Wait until the stack completes its update. Monitor the stack update using the **heat stack-list --show-nested** command.

Add new nodes to the director's node pool and deploy them as Ceph Storage nodes. Use the **CephStorageCount** parameter in the **parameter_defaults** of your environment file (in this case, **~/templates/storage-config.yaml**) to define the total number of Ceph Storage nodes in the Overcloud. For example:

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
```

See Section 6.1, "Assigning Nodes and Flavors to Roles" for details on how to define the number of nodes per role.

Upon updating your environment file, re-deploy the overcloud as normal:

```
$ openstack overcloud deploy --templates -e ENVIRONMENT_FILES
```

The director provisions the new node and updates the entire stack with the new node's details.

Log into a Controller node as the **heat-admin** user and check the status of the Ceph Storage node. For example:

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph status
```

Confirm that the value in the **osdmap** section matches the number of desired nodes in your cluster. The Ceph Storage node you removed has now been replaced with a new node.

7.6. ADDING AND REMOVING OSD DISKS FROM CEPH STORAGE NODES

In situations when an OSD disk fails and requires a replacement, use the standard instructions from the *Red Hat Ceph Storage Administration Guide*:

- "Adding an OSD"
- "Changing an OSD Drive"
- "Removing an OSD"

APPENDIX A. SAMPLE ENVIRONMENT FILE: CREATING A CEPH CLUSTER

The following custom environment file uses many of the options described throughout Chapter 2, *Preparing Overcloud Nodes*. This sample does not include any commented-out options. For an overview on environment files, see Environment Files (from the Advanced Overcloud Customization guide).

/home/stack/templates/storage-config.yaml

```
parameter_defaults: // 1
  CinderBackupBackend: ceph // 2
  CephAnsibleDisksConfig: 3
    osd_scenario: non-collocated
    dmcrypt: true
    devices:
      - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:10:0
      - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:11:0
    dedicated_devices:
      - /dev/nvme0n1
      - /dev/nvme0n1
  ControllerCount: 3 // 4
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
  CephMdsCount: 3
  OvercloudCephMdsFlavor: ceph-mds
  NeutronNetworkType: vxlan // 5
```

1. The **`parameter_defaults`** section modifies the default values for parameters in all templates. Most of the entries listed here are described in Chapter 4, *Customizing the Storage Service*.

2. If you are deploying the Ceph Object Gateway, you can use Ceph Object Storage (**`ceph-rgw`**) as a backup target. To configure this, set **`CinderBackupBackend`** to **`swift`**. See Section 4.2, "Enabling the Ceph Object Gateway" for details.

3. The **`CephAnsibleDisksConfig`** section defines a custom disk layout, as described in Section 5.1, "Mapping the Ceph Storage Node Disk Layout".

4. For each role, the **`*Count`** parameters assign a number of nodes while the **`Overcloud*Flavor`** parameters assign a flavor. For example, **`ControllerCount: 3`** assigns 3 nodes to the Controller role, and **`OvercloudControlFlavor: control`** sets each of those roles to use the **`control`** flavor. See Section 6.1, "Assigning Nodes and Flavors to Roles" for details.

NOTE

The **`CephMonCount`**, **`CephMdsCount`**, **`OvercloudCephMonFlavor`**, and **`OvercloudCephMdsFlavor`** parameters (along with the **`ceph-mon`** and **`ceph-mds`** flavors) will only be valid if you created a custom **`CephMON`** and **`CephMds`** role, as described in Chapter 3, *Deploying Other Ceph Services on Dedicated Nodes*.

5 **`NeutronNetworkType:`** sets the network type that the **`neutron`** service should use (in this case, **`vxlan`**).

APPENDIX B. SAMPLE CUSTOM INTERFACE TEMPLATE: MULTIPLE BONDED INTERFACES

The following template is a customized version of **`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml`**. It features multiple bonded interfaces to isolate back-end and front-end storage network traffic, along with redundancy for both connections (as described in Section 4.4, "Configuring Multiple Bonded Interfaces Per Ceph Node"). It also uses custom bonding options (namely, **`'mode=4 lacp_rate=1'`**, as described in Section 4.4.1, "Configuring Bonding Module Directives").

/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml (custom)

```
heat_template_version: 2015-04-30

description: >
  Software Config to drive os-net-config with 2 bonded nics on a bridge
  with VLANs attached for the ceph storage role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  BondInterfaceOvsOptions:
    default: 'mode=4 lacp_rate=1'
    description: The bonding_options string for the bond interface. Set
                 things like lacp=active and/or bond_mode=balance-slb
                 using this option.
    type: string
```

```
    constraints:
      - allowed_pattern: "^((?!balance.tcp).)*$"
        description: |
          The balance-tcp bond mode is known to cause packet loss and
          should not be used in BondInterfaceOvsOptions.
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: 20
    description: Vlan ID for the internal_api network traffic.
    type: number
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage mgmt network traffic.
    type: number
  TenantNetworkVlanID:
    default: 50
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  ControlPlaneSubnetCidr: # Override this via parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this via parameter_defaults
    description: The default route of the control plane network.
    type: string
  ExternalInterfaceDefaultRoute: # Not used by default in this template
    default: '10.0.0.1'
    description: The default route of the external network.
    type: string
  ManagementInterfaceDefaultRoute: # Commented out by default in this
template
    default: unset
    description: The default route of the management network.
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations)
that will be added to resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this via parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string

resources:
  OsNetConfigImpl:
```

```
type: OS::Heat::StructuredConfig
properties:
  group: os-apply-config
  config:
    os_net_config:
      network_config:
        -
          type: interface
          name: nic1
          use_dhcp: false
          dns_servers: {get_param: DnsServers}
          addresses:
            -
              ip_netmask:
                list_join:
                  - '/'
                  - - {get_param: ControlPlaneIp}
                    - {get_param: ControlPlaneSubnetCidr}
          routes:
            -
              ip_netmask: 169.254.169.254/32
              next_hop: {get_param: EC2MetadataIp}
            -
              default: true
              next_hop: {get_param: ControlPlaneDefaultRoute}
        -
          type: ovs_bridge
          name: br-bond
          members:
            -
              type: linux_bond
              name: bond1
              bonding_options: {get_param: BondInterfaceOvsOptions}
              members:
                -
                  type: interface
                  name: nic2
                  primary: true
                -
                  type: interface
                  name: nic3
            -
              type: vlan
              device: bond1
              vlan_id: {get_param: StorageNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: StorageIpSubnet}
        -
          type: ovs_bridge
          name: br-bond2
          members:
            -
              type: linux_bond
              name: bond2
              bonding_options: {get_param: BondInterfaceOvsOptions}
```

```
                  members:
                    -
                      type: interface
                      name: nic4
                      primary: true
                    -
                      type: interface
                      name: nic5
                -
                  type: vlan
                  device: bond1
                  vlan_id: {get_param: StorageMgmtNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: StorageMgmtIpSubnet}
outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```