



Red Hat OpenStack Platform 16.0

Deploying an overcloud with containerized Red Hat Ceph

Configuring the director to deploy and use a containerized Red Hat Ceph cluster

Red Hat OpenStack Platform 16.0 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

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This guide provides information about 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	4
1.1. INTRODUCTION TO CEPH STORAGE	4
1.2. REQUIREMENTS	4
1.2.1. Ceph Storage node requirements	5
1.3. ADDITIONAL RESOURCES	6
CHAPTER 2. PREPARING OVERCLOUD NODES	7
2.1. CLEANING CEPH STORAGE NODE DISKS	7
2.2. REGISTERING NODES	7
2.3. MANUALLY TAGGING NODES INTO PROFILES	10
2.4. DEFINING THE ROOT DISK FOR MULTI-DISK CLUSTERS	11
2.5. USING THE OVERCLOUD-MINIMAL IMAGE TO AVOID USING A RED HAT SUBSCRIPTION ENTITLEMENT	13
CHAPTER 3. DEPLOYING CEPH SERVICES ON DEDICATED NODES	15
3.1. CREATING A CUSTOM ROLES FILE	15
3.2. CREATING A CUSTOM ROLE AND FLAVOR FOR THE CEPH MON SERVICE	15
3.3. CREATING A CUSTOM ROLE AND FLAVOR FOR THE CEPH MDS SERVICE	17
CHAPTER 4. CUSTOMIZING THE STORAGE SERVICE	19
4.1. ENABLING THE CEPH METADATA SERVER	20
4.2. ENABLING THE CEPH OBJECT GATEWAY	20
4.3. CONFIGURING THE BACKUP SERVICE TO USE CEPH	21
4.4. CONFIGURING MULTIPLE BONDED INTERFACES FOR CEPH NODES	22
4.4.1. Configuring bonding module directives	24
CHAPTER 5. CUSTOMIZING THE CEPH STORAGE CLUSTER	26
5.1. SETTING CEPH-ANSIBLE GROUP VARIABLES	27
5.2. CEPH CONTAINERS FOR RED HAT OPENSTACK PLATFORM WITH CEPH STORAGE	27
5.3. MAPPING THE CEPH STORAGE NODE DISK LAYOUT	27
5.3.1. Using BlueStore	28
5.3.2. Referring to devices with persistent names	29
5.4. ASSIGNING CUSTOM ATTRIBUTES TO DIFFERENT CEPH POOLS	30
5.5. MAPPING THE DISK LAYOUT TO NON-HOMOGENEOUS CEPH STORAGE NODES	31
5.6. INCREASING THE RESTART DELAY FOR LARGE CEPH CLUSTERS	34
CHAPTER 6. DEPLOYING SECOND-TIER CEPH STORAGE ON OPENSTACK	35
6.1. CREATE A CRUSH MAP	35
6.2. MAPPING THE OSDS	35
6.3. SETTING THE REPLICATION FACTOR	36
6.4. DEFINING THE CRUSH HIERARCHY	36
6.5. DEFINING CRUSH MAP RULES	39
6.6. CONFIGURING OSP POOLS	40
6.7. CONFIGURING BLOCK STORAGE TO USE THE NEW POOL	41
6.8. VERIFYING CUSTOMIZED CRUSH MAP	41
CHAPTER 7. CREATING THE OVERCLOUD	42
7.1. ASSIGNING NODES AND FLAVORS TO ROLES	42
7.2. INITIATING OVERCLOUD DEPLOYMENT	43
CHAPTER 8. POST-DEPLOYMENT	46
8.1. ACCESSING THE OVERCLOUD	46
8.2. MONITORING CEPH STORAGE NODES	46

CHAPTER 9. REBOOTING THE ENVIRONMENT	48
9.1. REBOOTING A CEPH STORAGE (OSD) CLUSTER	48
CHAPTER 10. SCALING THE CEPH STORAGE CLUSTER	50
10.1. SCALING UP THE CEPH STORAGE CLUSTER	50
10.2. SCALING DOWN AND REPLACING CEPH STORAGE NODES	52
10.3. ADDING AN OSD TO A CEPH STORAGE NODE	54
10.4. REMOVING AN OSD FROM A CEPH STORAGE NODE	55
10.5. HANDLING DISK FAILURE	57
APPENDIX A. SAMPLE ENVIRONMENT FILE: CREATING A CEPH STORAGE CLUSTER	58
APPENDIX B. SAMPLE CUSTOM INTERFACE TEMPLATE: MULTIPLE BONDED INTERFACES	60

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, including integration with Red Hat Ceph Storage (both Ceph Storage clusters created with the director or existing Ceph Storage clusters).

This guide contains instructions for deploying a containerized Red Hat Ceph Storage cluster with your overcloud. Director uses Ansible playbooks provided through the **ceph-ansible** package to deploy a containerized Ceph cluster. The director also manages the configuration and scaling operations of the cluster.

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. INTRODUCTION TO 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 contains integration information for Ceph Block storage and the Ceph Object Gateway (RGW). It does not include information about Ceph File (CephFS) storage.

1.2. REQUIREMENTS

This guide contains information supplementary to the [Director Installation and Usage](#) guide. The [Requirements](#) section from that guide also applies to this guide. Implement these requirements as necessary.

Before you deploy a containerized Ceph Storage cluster with your overcloud, your environment must contain the following configuration:

- An undercloud host with the Red Hat OpenStack Platform director installed. See [Installing the Undercloud](#).
- Any additional hardware recommended for Red Hat Ceph Storage. For more information about recommended hardware, see the [Red Hat Ceph Storage Hardware Guide](#).



IMPORTANT

The Ceph Monitor service installs on the overcloud Controller nodes, so you must provide adequate resources to avoid performance issues. Ensure that 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. For a medium to large Ceph installation, provide at least 500 GB of Ceph monitor data. This space is necessary to avoid levelDB growth if the cluster becomes unstable.

If you use the Red Hat OpenStack Platform director to create Ceph Storage nodes, note the following requirements.

1.2.1. Ceph Storage node requirements

Ceph Storage nodes are responsible for providing object storage in a Red Hat OpenStack Platform environment.

Placement Groups (PGs)

Ceph uses placement groups to facilitate dynamic and efficient object tracking at scale. In the case of OSD failure or cluster rebalancing, Ceph can move or replicate a placement group and its contents, which means a Ceph cluster can re-balance and recover efficiently. The default placement group count that director creates is not always optimal so it is important to calculate the correct placement group count according to your requirements. You can use the placement group calculator to calculate the correct count: [Placement Groups \(PGs\) per Pool Calculator](#)

Processor

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

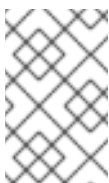
Memory

Red Hat typically recommends a baseline of 16 GB of RAM per OSD host, with an additional 2 GB of RAM per OSD daemon.

Disk layout

Sizing is dependent on your storage requirements. Red Hat recommends that your Ceph Storage node configuration includes three or more disks in a layout similar to the following example:

- **/dev/sda** - The root disk. The director copies the main overcloud image to the disk. Ensure that the disk has a minimum of 40 GB of available disk space.
- **/dev/sdb** - The journal disk. This disk divides into partitions for Ceph OSD journals. For example, **/dev/sdb1**, **/dev/sdb2**, and **/dev/sdb3**. 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 that you need at least two disks for a supported Ceph Storage node.

Network Interface Cards

A minimum of one 1 Gbps Network Interface Cards, although Red Hat recommends that you use at least two NICs in a production environment. Use additional network interface cards for bonded

interfaces or to delegate tagged VLAN traffic. Red Hat recommends that you use a 10 Gbps interface for storage nodes, especially if you want to create an OpenStack Platform environment that serves a high volume of traffic.

Power management

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

1.3. 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 contains all the default settings that the playbooks apply:

- `/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. Instead, use heat environment files to override the defaults set by these playbooks. If you edit the **ceph-ansible** playbooks directly, your deployment will fail.

For more information about the playbook collection, see the documentation for this project (<http://docs.ceph.com/ceph-ansible/master/>) to learn more about the playbook collection.

Alternatively, for information about the default settings applied by director for containerized Ceph Storage, see the heat templates in `/usr/share/openstack-tripleo-heat-templates/deployment/ceph-ansible`.



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 8 image to each node. Additionally, the Ceph Storage services on these nodes 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. You must delete all metadata from the disks to allow the director to set GPT labels on them.

You can configure 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 runs an additional step to boot the nodes and clean the disks each time the node is set to **available**. This process adds an additional power cycle after the first introspection and before each deployment. The Bare Metal Provisioning service uses the **wipefs --force --all** command to perform the clean.

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



WARNING

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

2.2. REGISTERING NODES

Import a node inventory file (`instackenv.json`) in JSON format to the director so that the director can communicate with the nodes. This inventory file contains hardware and power management details that the director can use to register nodes:

```
{
  "nodes":[
    {
      "mac":[
        "b1:b1:b1:b1:b1:b1"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"ipmi",
    }
  ]
}
```

```
"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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "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":"ipmi",
  "pm_user":"admin",
  "pm_password":"p@55w0rd!",
  "pm_addr":"192.0.2.213"
}
]
}

```

Procedure

1. After you create the inventory file, save the file to the home directory of the stack user (**/home/stack/instackenv.json**).
2. Initialize the stack user, then import the **instackenv.json** inventory file into the director:

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

The **openstack overcloud node import** command imports the inventory file and registers each node with the director.

3. 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 NODES INTO PROFILES

After you register each node, you must inspect the hardware and tag the node into a specific profile. Use profile tags to match your nodes to flavors, and then assign flavors to deployment roles.

To inspect and tag new nodes, complete the following 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 the nodes that are in a managed state. In this example, all nodes are in a managed state.
- The **--provide** option resets all nodes to an **active** state after introspection.



IMPORTANT

Ensure that this process completes successfully. 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. The addition of the **profile** option tags the nodes into each respective profile.



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.

For example, a typical deployment contains three profiles: **control**, **compute**, and **ceph-storage**. Run the following commands to 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-c7fdbc12129a 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 you can use to tag a node for the Ceph MON and Ceph MDS services. See [Chapter 3, Deploying Ceph services on dedicated nodes](#) for details.

2.4. DEFINING THE ROOT DISK FOR MULTI-DISK CLUSTERS

Director must identify the root disk during provisioning in the case of nodes with multiple disks. For example, most Ceph Storage nodes use multiple disks. By default, the director writes the overcloud image to the root disk during the provisioning process

There are several properties that you can define 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

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

Complete the following steps to specify the root device using its serial number.

Procedure

1. Check the disk information from the hardware introspection of each node. Run the following command to display the disk information of 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"
  }
]
```

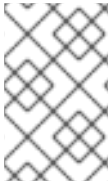
2. Run the **openstack baremetal node set --property root_device=** command to set the root disk for a node. Include the most appropriate hardware attribute value to define the root disk.


```
(undercloud) $ openstack baremetal node set --property
root_device='{“serial”:“<serial_number>”}' <node-uuid>
```

For example, to set the root device to disk 2, which has the serial number **61866da04f380d001ea4e13c12e36ad6** run the following command:

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

+



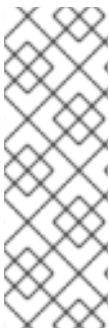
NOTE

Ensure that you configure the BIOS of each node to include booting from the root disk that you choose. Configure the boot order to boot from the network first, then to boot from the root disk.

Director identifies the specific disk to use as the root disk. When you run the **openstack overcloud deploy** command, director provisions and writes the overcloud image to the root disk.

2.5. USING THE OVERCLOUD-MINIMAL IMAGE TO AVOID USING A RED HAT SUBSCRIPTION ENTITLEMENT

By default, director writes the QCOW2 **overcloud-full** image to the root disk during the provisioning process. The **overcloud-full** image uses a valid Red Hat subscription. However, you can also use the **overcloud-minimal** image, for example, to provision a bare OS where you do not want to run any other OpenStack services or consume one of your subscription entitlements. A common use case is if you want to provision nodes with only Ceph daemons. For this, and other similar cases, you can use the **overcloud-minimal** image option to avoid reaching the limit of your paid Red Hat subscriptions.



NOTE

A Red Hat OpenStack Platform subscription contains Open vSwitch (OVS), but core services, such as OVS, are not available when you use the **overcloud-minimal** image. OVS is not required to deploy Ceph Storage nodes. Instead of using 'ovs_bond' to define bonds, use 'linux_bond'. For more information about **linux_bond**, see https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.0/html-single/advanced_overcloud_customization/index#linux-bonding-options.

Procedure

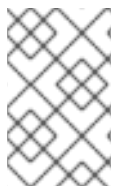
1. To configure director to use the **overcloud-minimal** image, create an environment file that contains the following image definition:

```
parameter_defaults:
  <roleName>Image: overcloud-minimal
```

2. Replace **<roleName>** with the name of the role and append **Image** to the name of the role. The following example shows an **overcloud-minimal** image for Ceph storage nodes:

```
parameter_defaults:  
  CephStorageImage: overcloud-minimal
```

3. Pass the environment file to the **openstack overcloud deploy** command.



NOTE

The **overcloud-minimal** image supports only standard Linux bridges and not OVS because OVS is an OpenStack service that requires a Red Hat OpenStack Platform subscription entitlement.

CHAPTER 3. DEPLOYING 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 Red Hat recommends that you deploy the Ceph MON and Ceph MDS services on dedicated nodes to improve the performance of your Ceph cluster. Create a custom role for services that you want to isolate on dedicated nodes.



NOTE

For more information about custom roles, see [Creating a New Role](#) in the [Advanced Opencloud 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`

3.1. CREATING A CUSTOM ROLES FILE

To create a custom role file, complete the following steps:

Procedure

1. Make a copy of the `roles_data.yaml` file in `/home/stack/templates/` so that you can add custom roles:

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

2. Include the new custom role file in the `openstack overcloud deploy` command.

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

Complete the following steps to create a custom role `CephMon` and flavor `ceph-mon` for the Ceph MON role. You must already have a copy of the default roles data file as described in [Chapter 3, Deploying Ceph services on dedicated nodes](#).

Procedure

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 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
[...]

```

- At the end of the `roles_data_custom.yaml` file, add a custom **CephMon** role that contains the Ceph MON service and all the other required node services:

```

- 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

```

- Run the `openstack flavor create` command to define a new flavor named **ceph-mon** for the **CephMon** role:

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

**NOTE**

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

- 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 details about this command, run `openstack flavor set --help`.

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

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

For more information about tagging nodes, see [Section 2.3, “Manually tagging nodes into profiles”](#). For more information about custom role profiles, see [Tagging Nodes Into Profiles](#).

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

Complete the following steps to create a custom role **CephMDS** and flavor **ceph-mds** for the Ceph MDS role. You must already have a copy of the default roles data file as described in [Chapter 3, *Deploying Ceph services on dedicated nodes*](#).

Procedure

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 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. In the next step, you add this service to the new custom role.

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

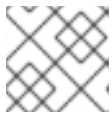
```
- name: CephMDS
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - 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 you can set with either the Ceph MON or Ceph Client service. If you deploy Ceph MDS on a dedicated node without the Ceph MON service, you must also include the Ceph Client service in the new **CephMDS** role.

4. Run the **openstack flavor create** command to 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 details about this command, run **openstack flavor create --help**.

5. Map the new **ceph-mds** 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 details about this command, run **openstack flavor set --help**.

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

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

For more information about tagging nodes, see [Section 2.3, “Manually tagging nodes into profiles”](#). For more information about custom role profiles, see [Tagging Nodes Into 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 director uses the `/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml` environment file to create a Ceph cluster and integrate it with your overcloud during deployment. This cluster features 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 also applies basic, default settings to the deployed Ceph cluster. You must also define any additional configuration in a custom environment file:

Procedure

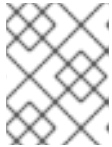
1. Create the file `storage-config.yaml` in `/home/stack/templates/`. In this example, the `~/templates/storage-config.yaml` file contains most of the overcloud-related custom settings for your environment. Parameters that you include in the custom environment file override the corresponding default settings from the `/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml` file.
2. Add a `parameter_defaults` section to `~/templates/storage-config.yaml`. This section contains custom settings for your overcloud. For example, to set `vxlan` as the network type of the networking service (`neutron`), add the following snippet to your custom environment file:

```
parameter_defaults:
  NeutronNetworkType: vxlan
```

3. If necessary, set the following options under `parameter_defaults` according to your requirements:

Option	Description	Default value
<code>CinderEnableiscsiBackend</code>	Enables the iSCSI backend	false
<code>CinderEnableRbdBackend</code>	Enables the Ceph Storage backend	true
<code>CinderBackupBackend</code>	Sets ceph or swift as the backend for volume backups. For more information, see Section 4.3, "Configuring the Backup Service to use Ceph" .	ceph
<code>NovaEnableRbdBackend</code>	Enables Ceph Storage for Nova ephemeral storage	true
<code>GlanceBackend</code>	Defines which backend the Image service should use: rbd (Ceph), swift , or file	rbd

Option	Description	Default value
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 custom environment file change depending on the settings that you apply in the following sections. See [Appendix A, Sample environment file: creating a Ceph Storage cluster](#) for a completed example.

The following subsections contain information about overriding the common default storage service settings that the director applies.

4.1. ENABLING THE CEPH METADATA SERVER

The Ceph Metadata Server (MDS) runs the **ceph-mds** daemon, which manages metadata related to files stored on CephFS. CephFS can be consumed through NFS. For more information about using CephFS through NFS, see [File System Guide](#) and [CephFS via NFS Back End Guide for the Shared File System Service](#).

**NOTE**

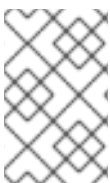
Red Hat supports deploying Ceph MDS only with the CephFS through NFS back end for the Shared File System service.

Procedure

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

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

For more information, see [Section 7.2, "Initiating overcloud deployment"](#). For more information about the Ceph Metadata Server, see [Configuring Metadata Server Daemons](#).

**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 more information, see [Section 3.3, "Creating a custom role and flavor for the Ceph MDS service"](#).

4.2. ENABLING THE CEPH OBJECT GATEWAY

The Ceph Object Gateway (RGW) provides applications with an interface to object storage capabilities

within a Ceph Storage cluster. When you deploy RGW, you can replace the default Object Storage service (**swift**) with Ceph. For more information, see [Object Gateway Configuration and Administration Guide](#).

Procedure

To enable RGW in your deployment, invoke the following environment file when you create the overcloud:

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

For more information, see [Section 7.2, "Initiating overcloud deployment"](#).

By default, Ceph Storage allows 250 placement groups per OSD. When you enable RGW, Ceph Storage creates six additional pools that are required by RGW. The new pools are:

- `.rgw.root`
- `default.rgw.control`
- `default.rgw.meta`
- `default.rgw.log`
- `default.rgw.buckets.index`
- `default.rgw.buckets.data`



NOTE

In your deployment, **default** is replaced with the name of the zone to which the pools belongs.

Therefore, when you enable RGW, be sure to set the default **pg_num** using the **CephPoolDefaultPgNum** parameter to account for the new pools. For more information about how to calculate the number of placement groups for Ceph pools, see [Section 5.4, "Assigning custom attributes to different Ceph pools"](#).

The Ceph Object Gateway is a direct 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 more information, see the [Block Storage Backup Guide](#).

4.3. CONFIGURING THE BACKUP SERVICE TO USE CEPH

The Block Storage Backup service (**cinder-backup**) is disabled by default. To enable the Block Storage Backup service, complete the following steps:

Procedure

Invoke the following environment file when you create your overcloud:

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

For more information, see the [Block Storage Backup Guide](#).

4.4. CONFIGURING MULTIPLE BONDED INTERFACES FOR CEPH NODES

Use a bonded interface to combine multiple NICs and add redundancy to a network connection. If you have enough NICs on your Ceph nodes, you can create multiple bonded interfaces on each node to expand redundancy capability.

You can then use a bonded interface for each network connection that the node requires. This provides both redundancy and a dedicated connection for each network.

The simplest implementation of bonded interfaces 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 the corresponding 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 \(PG\)](#) in the *Red Hat Ceph Architecture Guide*.

To configure multiple bonded interfaces, you must create a new network interface template, as the director does not provide any sample templates that you can use to deploy multiple bonded NICs. However, the director does provide a template that deploys a single bonded interface. This template is `/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml`. You can define an additional bonded interface for your additional NICs in this template.



NOTE

For more information about creating custom interface templates, see [Creating Custom Interface Templates](#) in the *Advanced Overcloud Customization* guide.

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

```
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 in 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**.
- 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 file. For more information about configuring bonding module directives, 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 example, the bonded interface uses **nic2** (set as the primary interface) and **nic3**.
- 6 The **br-bond** bridge has two other members: a VLAN for both front-end (**StorageNetwork**) and back-end (**StorageMgmtNetwork**) storage networks.
- 7 The **device** parameter defines which device a VLAN should use. In this example, both VLANs 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, which increases throughput and reliability for both storage network connections.

When you customize the `/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` file for this purpose, Red Hat recommends that you 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, **nic4** and **nic5**, and is 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 more 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 you add and configure the bonded interfaces, use the **BondInterfaceOvsOptions** parameter to set the directives that you want each bonded interface to use. You can find this information in the **parameters**: section of the `/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` file. 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'**:

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

For more information about other supported bonding options, see [Appendix C. Open vSwitch Bonding Options](#) in the *Advanced OpenCloud Optimization* guide. 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

Director deploys containerized Red Hat Ceph Storage using a default configuration. You can customize Ceph Storage by overriding the default settings.

Prerequisites

To deploy containerized Ceph Storage you must include the `/usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml` file during overcloud deployment. This environment file defines the following resources:

- **CephAnsibleDisksConfig** - This resource maps the Ceph Storage node disk layout. For more information, see [Section 5.3, “Mapping the Ceph Storage node disk layout”](#) .
- **CephConfigOverrides** - This resource applies all other custom settings to your Ceph Storage cluster.

Use these resources to override any defaults that the director sets for containerized Ceph Storage.

Procedure

1. Enable the Red Hat Ceph Storage 4 Tools repository:

```
$ sudo subscription-manager repos --enable=rhceph-4-tools-for-rhel-8-x86_64-rpms
```

2. Install the **ceph-ansible** package on the undercloud:

```
$ sudo dnf install ceph-ansible
```

3. To customize your Ceph Storage cluster, define custom parameters in a new environment file, for example, `/home/stack/templates/ceph-config.yaml`. You can apply Ceph Storage cluster settings with the following syntax in the **parameter_defaults** section of your environment file:

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



NOTE

You can apply the **CephConfigOverrides** parameter to the **[global]** section of the **ceph.conf** file, as well as any other section, such as **[osd]**, **[mon]**, and **[client]**. If you specify a section, the **key:value** data goes into the specified section. If you do not specify a section, the data goes into the **[global]** section by default. For information about Ceph Storage configuration, customization, and supported parameters, see [Red Hat Ceph Storage Configuration Guide](#) .

4. Replace **KEY** and **VALUE** with the Ceph cluster settings that you want to apply. For example, in the **global** section, **max_open_files** is the **KEY** and **131072** is the corresponding **VALUE**:

```
parameter_defaults:
  CephConfigOverrides:
    global:
```

```
max_open_files: 131072
osd:
  osd_scrub_during_recovery: false
```

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

```
[global]
max_open_files = 131072
[osd]
osd_scrub_during_recovery = false
```

5.1. SETTING CEPH-ANSIBLE GROUP VARIABLES

The **ceph-ansible** tool is a playbook used to install and manage Ceph Storage clusters.

The **ceph-ansible** tool has a **group_vars** directory that defines configuration options and the default settings for those options. Use the **group_vars** directory to set Ceph Storage parameters.

For information about the **group_vars** directory, see [Installing a Red Hat Ceph Storage cluster](#) in the *Installation Guide*.

To change the variable defaults in director, use the **CephAnsibleExtraConfig** parameter to pass the new values in heat environment files. For example, to set the **ceph-ansible** group variable **journal_size** to 40960, create an environment file with the following **journal_size** definition:

```
parameter_defaults:
  CephAnsibleExtraConfig:
    journal_size: 40960
```



IMPORTANT

Change **ceph-ansible** group variables with the override parameters; do not edit group variables directly in the **/usr/share/ceph-ansible** directory on the undercloud.

5.2. CEPH CONTAINERS FOR RED HAT OPENSTACK PLATFORM WITH CEPH STORAGE

A Ceph container is required to configure OpenStack Platform to use Ceph, even with an external Ceph cluster. To be compatible with Red Hat Enterprise Linux 8, OpenStack Platform 15 requires Red Hat Ceph Storage 4. The Ceph Storage 4 container is hosted at registry.redhat.io, a registry which requires authentication.

You can use the heat environment parameter **ContainerImageRegistryCredentials** to authenticate at registry.redhat.io, as described in [Container image preparation parameters](#).

5.3. MAPPING THE CEPH STORAGE NODE DISK LAYOUT

When you deploy containerized Ceph Storage, you must map the disk layout and specify dedicated block devices for the Ceph OSD service. You can perform this mapping in the environment file that you created earlier to define your custom Ceph parameters: **/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	lvm NOTE: The default value is lvm .	The lvm value allows ceph-ansible to use ceph-volume to configure OSDs and BlueStore WAL devices.
devices	Yes	NONE. Variable must be set.	A list of block devices that you want to use for OSDs on the node.
dedicated_devices	Yes (only if osd_scenario is non-collocated)	devices	A list of block devices that maps each entry in the devices parameter to a dedicated journaling block device. You can use this variable only when osd_scenario=non-collocated .
dmccrypt	No	false	Sets whether data stored on OSDs is encrypted (true) or unencrypted (false).
osd_objectstore	No	bluestore NOTE: The default value is bluestore .	Sets the storage back end used by Ceph.

5.3.1. Using BlueStore

To specify the block devices that you want to use as Ceph OSDs, use a variation of the following snippet:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
      - /dev/nvme0n1
    osd_scenario: lvm
    osd_objectstore: bluestore
```

Because **/dev/nvme0n1** is in a higher performing device class, the example parameter defaults produce three OSDs that run on **/dev/sdb**, **/dev/sdc**, and **/dev/sdd**. The three OSDs use **/dev/nvme0n1** as a BlueStore WAL device. The ceph-volume tool does this by using the **batch** subcommand. The same

setup is duplicated for each Ceph storage node and assumes uniform hardware. If the BlueStore WAL data resides on the same disks as the OSDs, then change the parameter defaults:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
    osd_scenario: lvm
    osd_objectstore: bluestore
```

5.3.2. Referring to devices with persistent names

In some nodes, disk paths, such as `/dev/sdb` and `/dev/sdc`, may not point to the same block device during reboots. If this is the case with your **CephStorage** nodes, specify each disk with the `/dev/disk/by-path/` symlink to ensure that the block device mapping is consistent throughout deployments:

```
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
```

Because you must set the list of OSD devices prior to overcloud deployment, it may not be possible to identify and set the PCI path of disk devices. In this case, gather the `/dev/disk/by-path/symlink` data for block devices during introspection.

In the following example, run the first command to download the introspection data from the undercloud Object Storage service (swift) for the server **b08-h03-r620-hci** and saves the data in a file called **b08-h03-r620-hci.json**. Run the second command to grep for "by-path". The output of this command contains the unique `/dev/disk/by-path` values that you can use to identify disks.

```
(undercloud) [stack@b08-h02-r620 ironic]$ openstack baremetal introspection data save b08-h03-r620-hci | jq . > b08-h03-r620-hci.json
(undercloud) [stack@b08-h02-r620 ironic]$ grep by-path b08-h03-r620-hci.json
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:0:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:1:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:3:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:4:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:5:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:6:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:7:0",
  "by_path": "/dev/disk/by-path/pci-0000:02:00.0-scsi-0:2:0:0",
```

For more information about naming conventions for storage devices, see [Overview of persistent naming attributes](#) in the *Managing storage devices* guide.

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

5.4. 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
      application: rbd
```

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.

If you use the **CephPools** parameter, you must also specify the application type. The application type for Compute, Block Storage, and Image Storage should be **rbd**, as shown in the examples, but depending on what the pool will be used for, you may need to specify a different application type. For example, the application type for the gnocchi metrics pool is **openstack_gnocchi**. See [Enable Application](#) in the *Storage Strategies Guide* for more information.

If you do not use the **CephPools** parameter, director sets the appropriate application type automatically, but only for the default pool list.

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
      application: rbd
```

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.

**WARNING**

Red Hat Ceph Storage 3 (Luminous) introduces a hard limit on the maximum number of PGs an OSD can have, which is 200 by default. Do not override this parameter beyond 200. If there is a problem because the Ceph PG number exceeds the maximum, adjust the **pg_num** per pool to address the problem, not the **mon_max_pg_per_osd**.

5.5. MAPPING THE DISK LAYOUT TO NON-HOMOGENEOUS CEPH STORAGE NODES

By default, all nodes of a role which will host Ceph OSDs (indicated by the **OS::TripleO::Services::CephOSD** service in **roles_data.yaml**), for example **CephStorage** or **ComputeHCI** nodes, will use the global **devices** and **dedicated_devices** lists set in [Section 5.3, “Mapping the Ceph Storage node disk layout”](#). This assumes that all of these servers have homogeneous hardware. If a subset of these servers do not have homogeneous hardware, then director needs to be aware that each of these servers has different **devices** and **dedicated_devices** lists. This is known as a *node-specific disk configuration*.

To pass director a node-specific disk configuration, a Heat environment file, such as **node-spec-overrides.yaml**, must be passed to the **openstack overcloud deploy** command and the file’s content must identify each server by a machine unique UUID and a list of local variables which override the global variables.

The machine unique UUID may be extracted for each individual server or from the Ironic database.

To locate the UUID for an individual server, log in to the server and run:

```
dmidecode -s system-uuid
```

To extract the UUID from the Ironic database, run the following command on the undercloud:

```
openstack baremetal introspection data save NODE-ID | jq .extra.system.product.uuid
```

**WARNING**

If the **undercloud.conf** does not have **inspection_extras = true** prior to undercloud installation or upgrade and introspection, then the machine unique UUID will not be in the Ironic database.

**IMPORTANT**

The machine unique UUID is not the Ironic UUID.

A valid **node-spec-overrides.yaml** file may look like the following:

```
parameter_defaults:
  NodeDataLookup: {"32E87B4C-C4A7-418E-865B-191684A6883B": {"/dev/sdc"}}
```

All lines after the first two lines must be valid JSON. An easy way to verify that the JSON is valid is to use the **jq** command. For example:

1. Remove the first two lines (**parameter_defaults:** and **NodeDataLookup:**) from the file temporarily.
2. Run **cat node-spec-overrides.yaml | jq .**

As the **node-spec-overrides.yaml** file grows, **jq** may also be used to ensure that the embedded JSON is valid. For example, because the **devices** and **dedicated_devices** list should be the same length, use the following to verify that they are the same length before starting the deployment.

```
(undercloud) [stack@b08-h02-r620 tht]$ cat node-spec-c05-h17-h21-h25-6048r.yaml | jq '[] | .devices
| length'
33
30
33
(undercloud) [stack@b08-h02-r620 tht]$ cat node-spec-c05-h17-h21-h25-6048r.yaml | jq '[] |
.dedicated_devices | length'
33
30
33
(undercloud) [stack@b08-h02-r620 tht]$
```

In the above example, the **node-spec-c05-h17-h21-h25-6048r.yaml** has three servers in rack c05 in which slots h17, h21, and h25 are missing disks. A more complicated example is included at the end of this section.

After the JSON has been validated add back the two lines which makes it a valid environment YAML file (**parameter_defaults:** and **NodeDataLookup:**) and include it with a **-e** in the deployment.

In the example below, the updated Heat Environment File uses NodeDataLookup for Ceph deployment. All of the servers had a devices list with 35 disks except one of them had a disk missing. This environment file overrides the default devices list for only that single node and gives it the list of 34 disks it should use instead of the global list.

```
parameter_defaults:
# c05-h01-6048r is missing scsi-0:2:35:0 (00000000-0000-0000-0000-0CC47A6EFD0C)
NodeDataLookup: {
  "00000000-0000-0000-0000-0CC47A6EFD0C": {
    "devices": [
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:1:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:32:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:2:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:3:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:4:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:5:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:6:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:33:0",
      "/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:7:0",
```



```

"/dev/disk/by-path/pci-0000:84:00.0-nvme-1",
"/dev/disk/by-path/pci-0000:84:00.0-nvme-1",
"/dev/disk/by-path/pci-0000:84:00.0-nvme-1",
"/dev/disk/by-path/pci-0000:84:00.0-nvme-1",
"/dev/disk/by-path/pci-0000:84:00.0-nvme-1"
]
}
}

```

5.6. INCREASING THE RESTART DELAY FOR LARGE CEPH CLUSTERS

During deployment, Ceph services such as OSDs and Monitors, are restarted and the deployment does not continue until the service is running again. Ansible waits 15 seconds (the delay) and checks 5 times for the service to start (the retries). If the service does not restart, the deployment stops so the operator can intervene.

Depending on the size of the Ceph cluster, you may need to increase the retry or delay values. The exact names of these parameters and their defaults are as follows:

```

health_mon_check_retries: 5
health_mon_check_delay: 15
health_osd_check_retries: 5
health_osd_check_delay: 15

```

Procedure

1. Update the **CephAnsibleExtraConfig** parameter to change the default delay and retry values:

```

parameter_defaults:
  CephAnsibleExtraConfig:
    health_osd_check_delay: 40
    health_osd_check_retries: 30
    health_mon_check_delay: 20
    health_mon_check_retries: 10

```

This example makes the cluster check 30 times and wait 40 seconds between each check for the Ceph OSDs, and check 20 times and wait 10 seconds between each check for the Ceph MONs.

2. To incorporate the changes, pass the updated **yaml** file with **-e** using **openstack overcloud deploy**.

CHAPTER 6. DEPLOYING SECOND-TIER CEPH STORAGE ON OPENSTACK

Using OpenStack director, you can deploy different Red Hat Ceph Storage performance tiers by adding new Ceph nodes dedicated to a specific tier in a Ceph cluster.

For example, you can add new object storage daemon (OSD) nodes with SSD drives to an existing Ceph cluster to create a Block Storage (cinder) backend exclusively for storing data on these nodes. A user creating a new Block Storage volume can then choose the desired performance tier: either HDDs or the new SSDs.

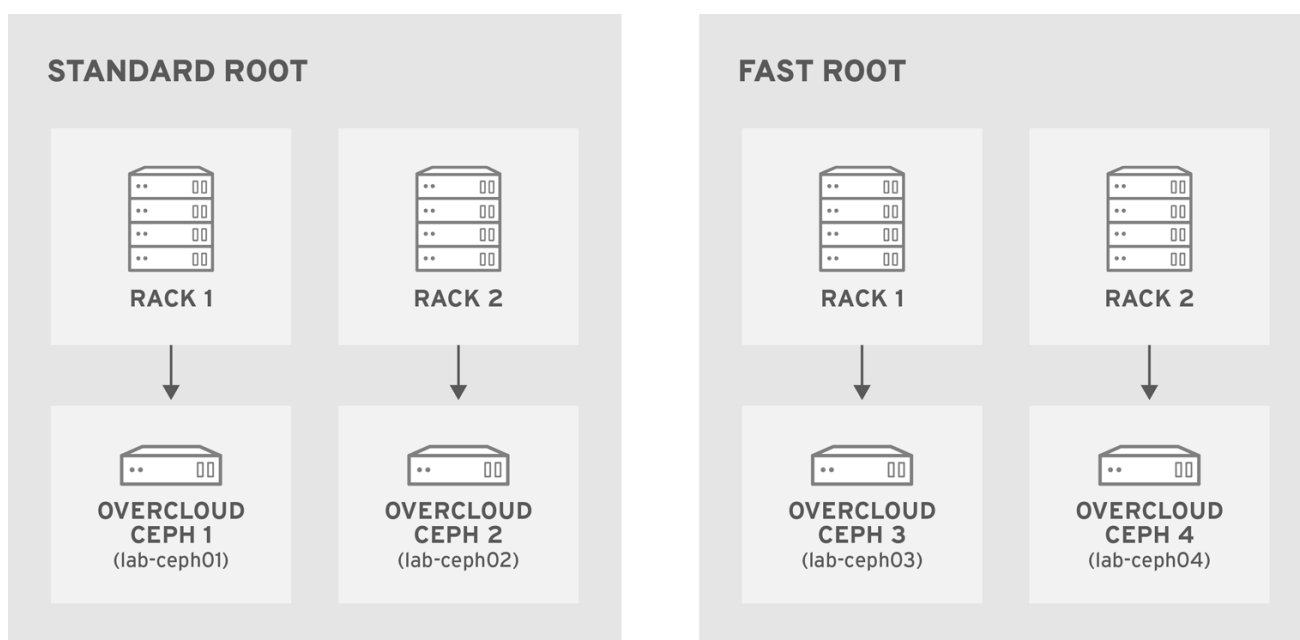
This type of deployment requires Red Hat OpenStack Platform director to pass a customized CRUSH map to ceph-ansible. The CRUSH map allows you to split OSD nodes based on disk performance, but you can also use this feature for mapping physical infrastructure layout.

The following sections demonstrate how to deploy four nodes where two of the nodes use SSDs and the other two use HDDs. The example is kept simple to communicate a repeatable pattern. However, a production deployment should use more nodes and more OSDs to be supported as per the [Red Hat Ceph Storage hardware selection guide](#).

6.1. CREATE A CRUSH MAP

The CRUSH map allows you to put OSD nodes into a CRUSH root. By default, a “default” root is created and all OSD nodes are included in it.

Inside a given root, you define the physical topology, rack, rooms, and so forth, and then place the OSD nodes in the desired hierarchy (or bucket). By default, no physical topology is defined; a flat design is assumed as if all nodes are in the same rack.



OPENSTACK_481504_1118

See [Crush Administration](#) in the *Storage Strategies Guide* for details about creating a custom CRUSH map.

6.2. MAPPING THE OSDS

Complete the following step to map the OSDs.

Procedure

1. Declare the OSDs/journal mapping:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sda
      - /dev/sdb
    dedicated_devices:
      - /dev/sdc
      - /dev/sdc
    osd_scenario: non-collocated
    journal_size: 8192
```

6.3. SETTING THE REPLICATION FACTOR

Complete the following step to set the replication factor.



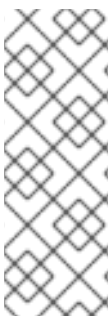
NOTE

This is normally supported only for full SSD deployment. See [Red Hat Ceph Storage: Supported configurations](#).

Procedure

1. Set the default replication factor to two. This example splits four nodes into two different roots.

```
parameter_defaults:
  CephPoolDefaultSize: 2
```



NOTE

If you upgrade a deployment that uses gnocchi as the backend, you might encounter deployment timeout. To prevent this timeout, use the following **CephPool** definition to customize the gnocchi pool:

```
parameter_defaults
  CephPools: {"name": metrics, "pg_num": 128, "pgp_num": 128, "size": 1}
```

6.4. DEFINING THE CRUSH HIERARCHY

Director provides the data for the CRUSH hierarchy, but ceph-ansible actually passes that data by getting the CRUSH mapping through the Ansible inventory file. Unless you keep the default root, you must specify the location of the root for each node.

For example if node lab-ceph01 (provisioning IP 172.16.0.26) is placed in **rack1** inside the **fast_root**, the Ansible inventory should resemble the following:

172.16.0.26:

```
osd_crush_location: {host: lab-ceph01, rack: rack1, root: fast_root}
```

When you use `director` to deploy Ceph, you don't actually write the Ansible inventory; it is generated for you. Therefore, you must use **NodeDataLookup** to append the data.

`NodeDataLookup` works by specifying the system product UUID stored on the motherboard of the systems. The Bare Metal service (`ironic`) also stores this information after the introspection phase.

To create a CRUSH map that supports second-tier storage, complete the following steps:

Procedure

1. Run the following commands to retrieve the UUIDs of the four nodes:

```
for ((x=1; x<=4; x++)); \
{ echo "Node overcloud-ceph0${x}"; \
openstack baremetal introspection data save overcloud-ceph0${x} | jq
.extra.system.product.uuid; }
Node overcloud-ceph01
"32C2BC31-F6BB-49AA-971A-377EFDDB111"
Node overcloud-ceph02
"76B4C69C-6915-4D30-AFFD-D16DB74F64ED"
Node overcloud-ceph03
"FECF7B20-5984-469F-872C-732E3FEF99BF"
Node overcloud-ceph04
"5FFEFA5F-69E4-4A88-B9EA-62811C61C8B3"
```

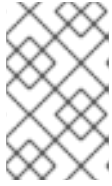


NOTE

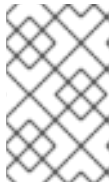
In the example, `overcloud-ceph0[1-4]` are the Ironic nodes names; they will be deployed as **lab-ceph0[1-4]** (via `HostnameMap.yaml`).

2. Specify the node placement as follows:

Root	Rack	Node
standard_root	rack1_std	overcloud-ceph01 (lab-ceph01)
	rack2_std	overcloud-ceph02 (lab-ceph02)
fast_root	rack1_fast	overcloud-ceph03 (lab-ceph03)
	rack2_fast	overcloud-ceph04 (lab-ceph04)

**NOTE**

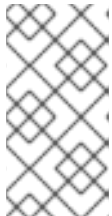
You cannot have two buckets with the same name. Even if **lab-ceph01** and **lab-ceph03** are in the same physical rack, you cannot have two buckets called **rack1**. Therefore, we named them **rack1_std** and **rack1_fast**.

**NOTE**

This example demonstrates how to create a specific route called "standard_root" to illustrate multiple custom roots. However, you could have kept the HDDs OSD nodes in the default root.

- Use the following **NodeDataLookup** syntax:

```
NodeDataLookup: {"SYSTEM_UUID": {"osd_crush_location": {"root": "$MY_ROOT", "rack": "$MY_RACK", "host": "$OVERCLOUD_NODE_HOSTNAME"}}
```

**NOTE**

You must specify the system UUID and then the CRUSH hierarchy from top to bottom. Also, the **host** parameter must point to the node's overcloud host name, not the Bare Metal service (ironic) node name. To match the example configuration, enter the following:

```
parameter_defaults:
  NodeDataLookup: {"32C2BC31-F6BB-49AA-971A-377EFDDB111":
{"osd_crush_location": {"root": "standard_root", "rack": "rack1_std", "host": "lab-ceph01"}},
    "76B4C69C-6915-4D30-AFFD-D16DB74F64ED": {"osd_crush_location": {"root":
"standard_root", "rack": "rack2_std", "host": "lab-ceph02"}},
    "FECF7B20-5984-469F-872C-732E3FEF99BF": {"osd_crush_location": {"root":
"fast_root", "rack": "rack1_fast", "host": "lab-ceph03"}},
    "5FFFEFA5F-69E4-4A88-B9EA-62811C61C8B3": {"osd_crush_location": {"root":
"fast_root", "rack": "rack2_fast", "host": "lab-ceph04"}}
```

- Enable CRUSH map management at the ceph-ansible level:

```
parameter_defaults:
  CephAnsibleExtraConfig:
    create_crush_tree: true
```

- Use scheduler hints to ensure the Bare Metal service node UUIDs correctly map to the hostnames:

```
parameter_defaults:
  CephStorageCount: 4
  OvercloudCephStorageFlavor: ceph-storage
  CephStorageSchedulerHints:
    'capabilities:node': 'ceph-%index%'
```

- Tag the Bare Metal service nodes with the corresponding hint:

```
openstack baremetal node set --property capabilities='profile:ceph-storage,node:ceph-0,boot_option:local' overcloud-ceph01
```

```
openstack baremetal node set --property capabilities=profile:ceph-storage,'node:ceph-1,boot_option:local' overcloud-ceph02
```

```
openstack baremetal node set --property capabilities='profile:ceph-storage,node:ceph-2,boot_option:local' overcloud-ceph03
```

```
openstack baremetal node set --property capabilities='profile:ceph-storage,node:ceph-3,boot_option:local' overcloud-ceph04
```



NOTE

For more information about predictive placement, see [Assigning Specific Node IDs](#) in the *Advanced Overcloud Customization* guide.

6.5. DEFINING CRUSH MAP RULES

Rules define how the data is written on a cluster. After the CRUSH map node placement is complete, define the CRUSH rules.

Procedure

1. Use the following syntax to define the CRUSH rules:

```
parameter_defaults:
  CephAnsibleExtraConfig:
    crush_rules:
      - name: $RULE_NAME
        root: $ROOT_NAME
        type: $REPLICAT_DOMAIN
        default: true/false
```

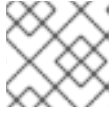


NOTE

Setting the default parameter to **true** means that this rule will be used when you create a new pool without specifying any rule. There may only be one default rule.

In the following example, rule **standard** points to the OSD nodes hosted on the **standard_root** with one replicate per rack. Rule **fast** points to the OSD nodes hosted on the **standard_root** with one replicate per rack:

```
parameter_defaults:
  CephAnsibleExtraConfig:
    crush_rule_config: true
    crush_rules:
      - name: standard
        root: standard_root
        type: rack
        default: true
      - name: fast
        root: fast_root
        type: rack
        default: false
```

**NOTE**

You must set **crush_rule_config** to **true**.

6.6. CONFIGURING OSP POOLS

Ceph pools are configured with a CRUSH rules that define how to store data. This example features all built-in OSP pools using the **standard_root** (the standard rule) and a new pool using **fast_root** (the fast rule).

Procedure

1. Use the following syntax to define or change a pool property:

```
- name: $POOL_NAME
  pg_num: $PG_COUNT
  rule_name: $RULE_NAME
  application: rbd
```

2. List all OSP pools and set the appropriate rule (standard, in this case), and create a new pool called **tier2** that uses the **fast** rule. This pool will be used by Block Storage (cinder).

```
parameter_defaults:
  CephPools:
    - name: tier2
      pg_num: 64
      rule_name: fast
      application: rbd

    - name: volumes
      pg_num: 64
      rule_name: standard
      application: rbd

    - name: vms
      pg_num: 64
      rule_name: standard
      application: rbd

    - name: backups
      pg_num: 64
      rule_name: standard
      application: rbd

    - name: images
      pg_num: 64
      rule_name: standard
      application: rbd

    - name: metrics
      pg_num: 64
      rule_name: standard
      application: openstack_gnocchi
```

6.7. CONFIGURING BLOCK STORAGE TO USE THE NEW POOL

Add the Ceph pool to the **cinder.conf** file to enable Block Storage (cinder) to consume it:

Procedure

1. Update **cinder.conf** as follows:

```
parameter_defaults:
  CinderRbdExtraPools:
    - tier2
```

6.8. VERIFYING CUSTOMIZED CRUSH MAP

After the **openstack overcloud deploy** command creates or updates the overcloud, complete the following step to verify that the customized CRUSH map was correctly applied.



NOTE

Be careful if you move a host from one route to another.

Procedure

1. Connect to a Ceph monitor node and run the following command:

```
# ceph osd tree
ID WEIGHT  TYPE NAME                UP/DOWN REWEIGHT PRIMARY-AFFINITY
-7 0.39996 root standard_root
-6 0.19998  rack rack1_std
-5 0.19998  host lab-ceph02
 1 0.09999  osd.1      up 1.00000 1.00000
 4 0.09999  osd.4      up 1.00000 1.00000
-9 0.19998  rack rack2_std
-8 0.19998  host lab-ceph03
 0 0.09999  osd.0      up 1.00000 1.00000
 3 0.09999  osd.3      up 1.00000 1.00000
-4 0.19998 root fast_root
-3 0.19998  rack rack1_fast
-2 0.19998  host lab-ceph01
 2 0.09999  osd.2      up 1.00000 1.00000
 5 0.09999  osd.5      up 1.00000 1.00000
```

CHAPTER 7. CREATING THE OVERCLOUD

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

7.1. 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 7.1. 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 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.

7.2. INITIATING OVERCLOUD DEPLOYMENT



NOTE

During undercloud installation, set **generate_service_certificate=false** in the **undercloud.conf** file. Otherwise, you must inject a trust anchor when you deploy the overcloud, as described in [Enabling SSL/TLS on Overcloud Public Endpoints](#) in the *Advanced Overcloud Customization* guide.

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 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”](#).
- **-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 8. POST-DEPLOYMENT

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

8.1. ACCESSING THE OVERCLOUD

The director generates a script to configure and help authenticate interactions with your overcloud from the undercloud. 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 undercloud CLI. To return to interacting with the undercloud, run the following command:

```
$ source ~/stackrc
```

8.2. MONITORING CEPH STORAGE NODES

After you create the overcloud, check the status of the Ceph Storage Cluster to ensure that it works correctly.

Procedure

1. Log in to a Controller node as the **heat-admin** user:

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

2. Check the health of the cluster:

```
$ sudo podman exec ceph-mon-$(hostname) ceph health
```

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

3. Log in to an overcloud node that runs the Ceph monitor service and check the status of all OSDs in the cluster:

```
$ sudo podman exec ceph-mon-$(hostname) ceph osd tree
```

4. Check the status of the Ceph Monitor quorum:

```
$ sudo podman exec ceph-mon-$(hostname) ceph quorum_status
```

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

5. Verify that all Ceph OSDs are running:

```
$ sudo podman exec ceph-mon-$(hostname) ceph osd stat
```

For more information on monitoring Ceph Storage clusters, see [Monitoring](#) in the *Red Hat Ceph Storage Administration Guide*.

CHAPTER 9. 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.

9.1. REBOOTING A CEPH STORAGE (OSD) CLUSTER

Complete the following steps to reboot a cluster of Ceph Storage (OSD) nodes.

Procedure

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

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
```

2. Select the first Ceph Storage node that you want to reboot and log in to the node.
3. Reboot the node:

```
$ sudo reboot
```

4. Wait until the node boots.
5. Log into the node and check the cluster status:

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

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 re-enable cluster rebalancing:

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
```

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

```
$ sudo podman exec -it ceph-mon-controller-0 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
```

CHAPTER 10. SCALING THE CEPH STORAGE CLUSTER

10.1. SCALING UP THE CEPH STORAGE 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 in to the undercloud 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 you register each node, you must inspect the hardware and tag the node into a specific profile. Use profile tags to match your nodes to flavors, and then assign flavors to deployment roles.

To inspect and tag new nodes, complete the following 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 the nodes that are in a managed state. In this example, all nodes are in a managed state.
- The **--provide** option resets all nodes to an **active** state after introspection.

**IMPORTANT**

Ensure that this process completes successfully. This process usually takes 15 minutes for bare metal nodes.

- Retrieve a list of your nodes to identify their UUIDs:

```
$ openstack baremetal node list
```

- Add a profile option to the **properties/capabilities** parameter for each node to manually tag a node to a specific profile. The addition of the **profile** option tags the nodes into each respective profile.

**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.

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 multi-disk clusters”](#) 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 7.1, “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.

10.2. 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 [Starting, stopping, and restarting Ceph daemons that run in containers](#) and [Removing a Ceph OSD using the command-line interface](#).

1. Log in to a Controller node as the **heat-admin** user. The director's **stack** user has an SSH key to access the **heat-admin** user.
2. List the OSD tree and find the OSDs for your node. For example, the node you want 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
```

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

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

4. The Ceph Storage cluster begins rebalancing. Wait for this process to complete. Follow the status using the following command:

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph -w
```

5. After the Ceph cluster completes rebalancing, log in to 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 podman exec ceph-mon-$HOSTNAME
systemctl disable ceph-osd@0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME
systemctl disable ceph-osd@1
```

6. Stop the OSDs.

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

7. While logged in to the Controller node, remove the OSDs from the CRUSH map so that they no longer receive data.


```
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
osd crush remove osd.0
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
osd crush remove osd.1
```

- Remove the OSD authentication key.

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
auth del osd.0
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
auth del osd.1
```

- Remove the OSD from the cluster.

```
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
osd rm 0
[heat-admin@overcloud-controller-0 ~]$ sudo podman exec ceph-mon-$HOSTNAME ceph
osd rm 1
```

- Leave the node and return to the undercloud as the **stack** user.

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

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

```
[stack@director ~]$ openstack baremetal node list
[stack@director ~]$ openstack baremetal node maintenance set UUID
```

- 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:

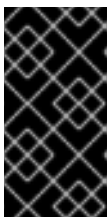
```
$ openstack stack list
```

- Identify the UUIDs of the Ceph Storage node you want to delete:

```
$ openstack server list
```

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

```
$ openstack overcloud node delete --stack overcloud 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](#) in the [Director Installation and Usage](#) guide.

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

16. 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
```



NOTE

See [Section 7.1, "Assigning nodes and flavors to roles"](#) for details on how to define the number of nodes per role.

17. After you update 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.

18. Log in to 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
```

19. 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.

10.3. ADDING AN OSD TO A CEPH STORAGE NODE

This procedure demonstrates how to add an OSD to a node. For more information about Ceph OSDs, see [Ceph OSDs](#) in the *Red Hat Ceph Storage Operations Guide*.

Procedure

1. Notice the following heat template deploys Ceph Storage with three OSD devices:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
  osd_scenario: lvm
  osd_objectstore: bluestore
```

- To add an OSD, update the node disk layout as described in [Section 5.3, “Mapping the Ceph Storage node disk layout”](#). In this example, add **/dev/sde** to the template:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
      - /dev/sde
    osd_scenario: lvm
    osd_objectstore: bluestore
```

- Run **openstack overcloud deploy** to update the overcloud.



NOTE

This example assumes that all hosts with OSDs have a new device called **/dev/sde**. If you do not want all nodes to have the new device, update the heat template as shown and see [Section 5.5, “Mapping the disk layout to non-homogeneous Ceph Storage nodes”](#) for information about how to define hosts with a differing **devices** list.

10.4. REMOVING AN OSD FROM A CEPH STORAGE NODE

This procedure demonstrates how to remove an OSD from a node. It assumes the following about the environment:

- A server (**ceph-storage0**) has an OSD (**ceph-osd@4**) running on **/dev/sde**.
- The Ceph monitor service (**ceph-mon**) is running on **controller0**.
- There are enough available OSDs to ensure the storage cluster is not at its near-full ratio.

For more information about Ceph OSDs, see [Ceph OSDs](#) in the *Red Hat Ceph Storage Operations Guide*.

Procedure

- SSH into **ceph-storage0** and log in as **root**.
- Disable and stop the OSD service:

```
[root@ceph-storage0 ~]# systemctl disable ceph-osd@4
[root@ceph-storage0 ~]# systemctl stop ceph-osd@4
```

- Disconnect from **ceph-storage0**.
- SSH into **controller0** and log in as **root**.
- Identify the name of the Ceph monitor container:

```
[root@controller0 ~]# podman ps | grep ceph-mon
ceph-mon-controller0
[root@controller0 ~]#
```

6. Enable the Ceph monitor container to mark the undesired OSD as **out**:

```
[root@controller0 ~]# podman exec ceph-mon-controller0 ceph osd out 4
```



NOTE

This command causes Ceph to rebalance the storage cluster and copy data to other OSDs in the cluster. The cluster temporarily leaves the **active+clean** state until rebalancing is complete.

7. Run the following command and wait for the storage cluster state to become **active+clean**:

```
[root@controller0 ~]# podman exec ceph-mon-controller0 ceph -w
```

8. Remove the OSD from the CRUSH map so that it no longer receives data:

```
[root@controller0 ~]# podman exec ceph-mon-controller0 ceph osd crush remove osd.4
```

9. Remove the OSD authentication key:

```
[root@controller0 ~]# podman exec ceph-mon-controller0 ceph auth del osd.4
```

10. Remove the OSD:

```
[root@controller0 ~]# podman exec ceph-mon-controller0 ceph osd rm 4
```

11. Disconnect from **controller0**.

12. SSH into the undercloud as the **stack** user and locate the heat environment file in which you defined the **CephAnsibleDisksConfig** parameter.

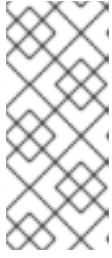
13. Notice the heat template contains four OSDs:

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
      - /dev/sde
    osd_scenario: lvm
    osd_objectstore: bluestore
```

14. Modify the template to remove **/dev/sde**.

```
parameter_defaults:
  CephAnsibleDisksConfig:
    devices:
      - /dev/sdb
      - /dev/sdc
      - /dev/sdd
    osd_scenario: lvm
    osd_objectstore: bluestore
```

-
- 15. Run **openstack overcloud deploy** to update the overcloud.



NOTE

This example assumes that you removed the **/dev/sde** device from all hosts with OSDs. If you do not remove the same device from all nodes, update the heat template as shown and see [Section 5.5, “Mapping the disk layout to non-homogeneous Ceph Storage nodes”](#) for information about how to define hosts with a differing **devices** list.

10.5. HANDLING DISK FAILURE

If a disk fails, see [Handling a Disk Failure](#) in the Red Hat Ceph Storage *Operations Guide*.

APPENDIX A. SAMPLE ENVIRONMENT FILE: CREATING A CEPH STORAGE 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: ❶
  CinderBackupBackend: ceph ❷
  CephAnsibleDisksConfig: ❸
    osd_scenario: lvm
    osd_objectstore: bluestore
    dmccrypt: 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
      - /dev/nvme0n1
  ControllerCount: 3 ❹
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
  CephMdsCount: 3
  OvercloudCephMdsFlavor: ceph-mds
  NeutronNetworkType: vxlan ❺
```

- ❶ 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*](#).
- ❷ 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.
- ❸ The **CephAnsibleDisksConfig** section defines a custom disk layout for deployments using BlueStore.
- ❹ 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 7.1, “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 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 [\[\]](#)). It also uses custom bonding options (namely, `'mode=4 lacp_rate=1'`, as described in [xref:multibonded-nics-ovs-opts\[\]](#)).

`/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}
```