Red Hat OpenStack Platform 10

Red Hat Ceph Storage for the Overcloud

Configuring an Overcloud to Use Red Hat Ceph Storage

Last Updated: 2020-05-26

Red Hat OpenStack Platform 10 Red Hat Ceph Storage for the Overcloud

Configuring an Overcloud to Use Red Hat Ceph Storage

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide provides information on using the Red Hat OpenStack Platform director to create an Overcloud that uses Red Hat Ceph Storage. This includes recommendations for your Red Hat Ceph Storage environment and instructions on how to implement an Overcloud with Ceph Storage nodes.

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat OpenStack Platform director creates a cloud environment called the **Overcloud**. The director provides the ability to configure extra features for an Overcloud. One of these extra features includes integration with Red Hat Ceph Storage. This includes both Ceph Storage clusters created with the director or existing Ceph Storage clusters. This guide provides information for integrating Ceph Storage into your Overcloud through the director and configuration examples.

1.1. DEFINING CEPH STORAGE

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

Ceph OSD (Object Storage Daemon)

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

Ceph Monitor

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

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

IMPORTANT

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

1.2. USING CEPH STORAGE IN RED HAT OPENSTACK PLATFORM

Red Hat OpenStack Platform director provides two main methods for integrating Red Hat Ceph Storage into an Overcloud.

Creating an Overcloud with its own Ceph Storage Cluster

The director has the ability to create a Ceph Storage Cluster during the creation on the Overcloud. The director creates a set of Ceph Storage nodes that use the Ceph OSD to store the data. In addition, the director installs the Ceph Monitor service on the Overcloud's Controller nodes. This means if an organization creates an Overcloud with three highly available controller nodes, the Ceph Monitor also becomes a highly available service.

Integrating a Existing Ceph Storage into an Overcloud

If you already have an existing Ceph Storage Cluster, you can integrate this during an Overcloud deployment. This means you manage and scale the cluster outside of the Overcloud configuration.

1.3. SETTING REQUIREMENTS

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

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

Processor

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

Memory

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

Disk Space

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

Disk Layout

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

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

IMPORTANT

Erase all existing partitions on the disks targeted for journaling and OSDs before deploying the Overcloud. In addition, the Ceph Storage OSDs and journal disks require GPT disk labels, which you can configure as a part of the deployment. See Section 2.10, "Formatting Ceph Storage Node Disks to GPT" for more information.

Network Interface Cards

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

Intelligent Platform Management Interface (IPMI)

Each Ceph node requires IPMI functionality on the server's motherboard.

This guide also requires the following:

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

IMPORTANT

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

1.4. DEFINING THE SCENARIOS

This guide uses two scenarios:

- The first scenario creates an Overcloud with a Ceph Storage Cluster. This means the director deploys the Ceph Storage Cluster.
- The second scenario integrates an existing Ceph Storage Cluster with an Overcloud. This means you manage the Ceph Storage Cluster separate from Overcloud management.

CHAPTER 2. CREATING AN OVERCLOUD WITH CEPH STORAGE NODES

This chapter describes how to use the director to create an Overcloud that includes its own Ceph Storage Cluster. For instructions on how to create an Overcloud and integrate it with an existing Ceph Storage Cluster, see Chapter 3, *Integrating an Existing Ceph Storage Cluster with an Overcloud* instead.

The scenario described in this chapter consists of nine nodes in the Overcloud:

- Three Controller nodes with high availability. This includes the Ceph Monitor service on each node.
- Three Red Hat Ceph Storage nodes in a cluster. These nodes contain the Ceph OSD service and act as the actual storage.
- Three Compute nodes.

All machines in this scenario are bare metal systems using IPMI for power management. These nodes do not require an operating system because the director copies a Red Hat Enterprise Linux 7 image to each node.

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. For this example, we use 192.0.2.0/24 as the Provisioning subnet with the following IP address assignments:

Node Name	IP Address	MAC Address	IPMI IP Address
Director	192.0.2.1	aa:aa:aa:aa:aa:aa	
Controller 1	DHCP defined	b1:b1:b1:b1:b1:b1	192.0.2.205
Controller 2	DHCP defined	b2:b2:b2:b2:b2:b2	192.0.2.206
Controller 3	DHCP defined	b3:b3:b3:b3:b3:b3	192.0.2.207
Compute 1	DHCP defined	c1:c1:c1:c1:c1:c1	192.0.2.208
Compute 2	DHCP defined	c2:c2:c2:c2:c2:c2	192.0.2.209
Compute 3	DHCP defined	c3:c3:c3:c3:c3:c3	192.0.2.210
Ceph 1	DHCP defined	d1:d1:d1:d1:d1:d1	192.0.2.211
Ceph 2	DHCP defined	d2:d2:d2:d2:d2:d2	192.0.2.212
Ceph 3	DHCP defined	d3:d3:d3:d3:d3:d3	192.0.2.213

2.1. INITIALIZING THE STACK USER

Log into the director host as the **stack** user and run the following command to initialize your director configuration:

```
$ source ~/stackrc
```

This sets up environment variables containing authentication details to access the director's CLI tools.

2.2. REGISTERING NODES

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

```
{
    "nodes":[
        {
            "mac":[
                "b1:b1:b1:b1:b1:b1"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
        },
        {
            "mac":[
                "b2:b2:b2:b2:b2:b2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        },
        {
            "mac":[
                "b3:b3:b3:b3:b3:b3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.207"
        },
        {
            "mac":[
```

```
            "c1:c1:c1:c1:c1:c1"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"pxe_ipmitool",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.0.2.208"
    },
    {
        "mac":[
            "c2:c2:c2:c2:c2:c2"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"pxe_ipmitool",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.0.2.209"
    },
    {
        "mac":[
            "c3:c3:c3:c3:c3:c3"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"pxe_ipmitool",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.0.2.210"
    },
    {
        "mac":[
            "d1:d1:d1:d1:d1:d1"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"pxe_ipmitool",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.0.2.211"
    },
    {
        "mac":[
            "d2:d2:d2:d2:d2:d2"
        ],
        "cpu":"4",
        "memory":"6144",
```

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

After creating the template, save the file to the stack user's home directory (/**home/stack/instackenv.json**), then import it into the director. Use the following command to accomplish this:

```
$ openstack baremetal import --json ~/instackenv.json
```

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

Assign the kernel and ramdisk images to all nodes:

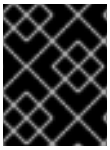
```
$ openstack baremetal configure boot
```

The nodes are now registered and configured in the director.

2.3. INSPECTING THE HARDWARE OF NODES

After registering the nodes, inspect the hardware attribute of each node. Run the following command to inspect the hardware attributes of each node:

```
$ openstack baremetal introspection bulk start
```



IMPORTANT

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

2.4. MANUALLY TAGGING THE NODES

After registering and inspecting the hardware of each node, tag them into specific profiles. These profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role.

Retrieve a list of your nodes to identify their UUIDs:

```
$ ironic node-list
```

To manually tag a node to a specific profile, add a profile option to the **properties/capabilities** parameter for each node. For example, to tag three nodes to use a controller profile and one node to use a compute profile, use the following commands:

```
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update da0cc61b-4882-45e0-9f43-fab65cf4e52b add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update b9f70722-e124-4650-a9b1-aade8121b5ed add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 68bf8f29-7731-4148-ba16-efb31ab8d34f add
properties/capabilities='profile:ceph-storage,boot_option:local'
```

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

NOTE

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

2.5. DEFINING THE ROOT DISK FOR CEPH STORAGE NODES

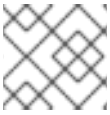
Most Ceph Storage nodes use multiple disks. This means the director needs to identify the disk to use for the root disk when provisioning a Ceph Storage node. There are several properties you can use to help identify the root disk:

- **model** (String): Device identifier.
- **vendor** (String): Device vendor.
- **serial** (String): Disk serial number.
- **wwn** (String): Unique storage identifier.
- **size** (Integer): Size of the device in GB.

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

First, collect a copy of each node's hardware information that the director obtained from the introspection. This information is stored in the OpenStack Object Storage server (swift). Download this information to a new directory:

```
$ mkdir swift-data
$ cd swift-data
$ export SWIFT_PASSWORD=`sudo crudini --get /etc/ironic-inspector/inspector.conf swift password`
$ for node in $(ironic node-list | grep -v UUID| awk '{print $2}'); do swift -U service:ironic -K
$SWIFT_PASSWORD download ironic-inspector inspector_data-$node; done
```



NOTE

This example uses the **crudini** command, which is available in the **crudini** package.

This downloads the data from each **inspector_data** object from introspection. All objects use the node UUID as part of the object name:

```
$ ls -1
inspector_data-15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
inspector_data-46b90a4d-769b-4b26-bb93-50eaefcdb3f4
inspector_data-662376ed-faa8-409c-b8ef-212f9754c9c7
inspector_data-6fc70fe4-92ea-457b-9713-eed499eda206
inspector_data-9238a73a-ec8b-4976-9409-3fcff9a8dca3
inspector_data-9cbfe693-8d55-47c2-a9d5-10e059a14e07
inspector_data-ad31b32d-e607-4495-815c-2b55ee04cdb1
inspector_data-d376f613-bc3e-4c4b-ad21-847c4ec850f8
```

Check the disk information for each node. The following command displays each node ID and the disk information:

```
$ for node in $(ironic node-list | grep -v UUID| awk '{print $2}'); do echo "NODE: $node" ; cat
inspector_data-$node | jq '.inventory.disks' ; echo "-----" ; done
```

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

```
NODE: 15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
```

```
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
-----
```

For this example, set the root device to disk 2, which has **61866da04f37fc001ea4e31e121cfb45** as the serial number. This requires a change to the **root_device** parameter for the node definition:

```
$ ironic node-update 15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3 add properties/root_device='{"serial":
"61866da04f37fc001ea4e31e121cfb45"}'
```

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

IMPORTANT

Do not use **name** to set the root disk as this value can change when the node boots.

2.6. ENABLING CEPH STORAGE IN THE OVERCLOUD

The Overcloud image already contains the Ceph services and the necessary Puppet modules to automatically configure both the Ceph OSD nodes and the Ceph Monitor on Controller clusters. The Overcloud's Heat template collection also contains the necessary procedures to enable your Ceph Storage configuration. However, the director requires some details to enable Ceph Storage and pass on the intended configuration. To pass this information, copy the **storage-environment.yaml** environment file to your stack user's templates directory.

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

Modify the following options in the copy of storage-environment.yaml:

CinderEnableIscsiBackend

Enables the iSCSI backend. Set to **false**.

CinderEnableRbdBackend

Enables the Ceph Storage backend. Set to **true**.

CinderEnableNfsBackend

Enables the NFS backend. Set to **false**.

NovaEnableRbdBackend

Enables Ceph Storage for Nova ephemeral storage. Set to **true**.

GlanceBackend

Defines the backend to use for Glance. Set to **rbd** to use Ceph Storage for images.

Next, modify each entry in the **resource_registry** to point to the absolute path of each resource:

```
resource_registry:
  OS::TripleO::Services::CephMon: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-osd.yaml
  OS::TripleO::Services::CephClient: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-client.yaml
```

NOTE

The **storage-environment.yaml** also contains some options to configure Ceph Storage directly through Heat. However, these options are not necessary in this scenario since the director creates these nodes and automatically defines the configuration values.

2.7. MAPPING THE CEPH STORAGE NODE DISK LAYOUT

The default mapping uses the root disk for Ceph Storage. However, most production environments use multiple separate disks for storage and partitions for journaling. In this situation, you define a storage map as part of the **storage-environment.yaml** file copied previously.

Edit the **storage-environment.yaml** file and the following snippet to the **parameter_defaults**:

```
ExtraConfig:
  ceph::profile::params::osds:
```

This adds extra Hiera data to the Overcloud, which Puppet uses as custom parameters during configuration. Use the **ceph::profile::params::osds** parameter to map the relevant disks and journal partitions. For example, a Ceph node with four disks might have the following assignments:

- **/dev/sda** - The root disk containing the Overcloud image
- **/dev/sdb** - The disk containing the journal partitions. This is usually a solid state disk (SSD) to aid with system performance.
- **/dev/sdc** and **/dev/sdd** - The OSD disks

For this example, the mapping might contain the following:

```
ceph::profile::params::osds:
    '/dev/sdc':
      journal: '/dev/sdb'
    '/dev/sdd':
      journal: '/dev/sdb'
```

If you do not want a separate disk for journals, use co-located journals on the OSD disks. Pass a blank value to the journal parameters:

```
ceph::profile::params::osds:
  '/dev/sdb': {}
  '/dev/sdc': {}
  '/dev/sdd': {}
```

NOTE

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

```
ceph::profile::params::osds:
  '/dev/disk/by-path/pci-0000:00:17.0-ata-2-part1':
    journal: '/dev/nvme0n1'
  '/dev/disk/by-path/pci-0000:00:17.0-ata-2-part2':
    journal: '/dev/nvme0n1'
```

This will ensure that the block device mapping is consistent throughout deployments.

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

You can also deploy Ceph nodes with different types of disks (for example, SSD and SATA disks on the same physical host). In a typical Ceph deployment, this is configured through CRUSH maps, as described in Placing Different Pools on Different OSDS . If you are mapping such a deployment, add the following line to the **ExtraConfig** section of the **storage-environment.yaml**:

```
ceph::osd_crush_update_on_start: false
```

Afterwards, save the **~/templates/storage-environment.yaml** file so that when we deploy the Overcloud, the Ceph Storage nodes use our disk mapping. We include this file in our deployment to initiate our storage requirements.

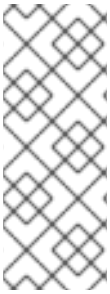
2.8. DEPLOY THE CEPH OBJECT GATEWAY

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

To enable a Ceph Object Gateway in your deployment, add the following snippet to the **resource_registry** of your environment file (namely, **~/templates/storage-environment.yaml**):

```
  OS::TripleO::Services::CephRgw: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-rgw.yaml
  OS::TripleO::Services::SwiftProxy: OS::Heat::None
  OS::TripleO::Services::SwiftStorage: OS::Heat::None
  OS::TripleO::Services::SwiftRingBuilder: OS::Heat::None
```

In addition to deploying the Ceph Object Gateway, this snippet also disables the default Object Storage service. (**swift**).



NOTE

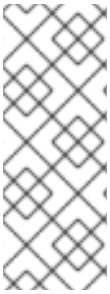
These resources are also found in **/usr/share/openstack-tripleo-heat-templates/environments/ceph-radosgw.yaml**; you can also invoke this environment file directly during deployment. In this document, the resources are defined directly in **/home/stack/templates/storage-environment.yaml**, as doing so centralizes all resources and parameters to one environment file (shown in Appendix A, *Sample Environment File: Creating a Ceph Cluster*).

The Ceph Object Gateway acts as a drop-in replacement for the default Object Storage service. As such, all other services that normally use **swift** can seamlessly start using the Ceph Object Gateway instead without further configuration. For example, when configuring the Block Storage Backup service (**cinder-backup**) to use the Ceph Object Gateway, set **swift** as the target back end (see Section 2.9, "Configuring the Backup Service to Use Ceph").

2.9. CONFIGURING THE BACKUP SERVICE TO USE CEPH

The Block Storage Backup service (**cinder-backup**) is disabled by default. You can enable it by adding the following line to the **resource_registry** of your environment file (namely, **~/templates/storage-environment.yaml**):

```
OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-templates/puppet/services/pacemaker/cinder-backup.yaml
```



NOTE

This resource is also defined in **/usr/share/openstack-tripleo-heat-templates/environments/cinder-backup.yaml**, which you can also invoke directly during deployment. In this document, the resource is defined directly in **/home/stack/templates/storage-environment.yaml** instead, as doing so centralizes all resources and parameters to one environment file (shown in Appendix A, *Sample Environment File: Creating a Ceph Cluster*).

Next, configure the **cinder-backup** service to store backups in Ceph. This involves configuring the service to use Ceph Object Storage (assuming you are also deploying the Ceph Object Gateway, as in Section 2.8, "Deploy the Ceph Object Gateway"). To do so, add the following line to the **parameter_defaults** of your environment file:

```
CinderBackupBackend: swift
```

NOTE

If you are not deploying the Ceph Object Gateway and wish to use the Ceph Block Device as your backup target instead, use:

```
CinderBackupBackend: ceph
```

2.10. FORMATTING CEPH STORAGE NODE DISKS TO GPT

The Ceph Storage OSDs and journal partitions require GPT disk labels. This means the additional disks on Ceph Storage require conversion to GPT labels before installing the Ceph OSD. To accomplish this, the node must execute a script to perform this operation on first boot. You include this script as part of a

Heat template in your Overcloud creation. For example, the following heat template (**wipe-disks.yaml**) runs a script that checks all disks on Ceph Storage node and converts all of them (except the disk containing the root file system) to GPT.

```
heat_template_version: 2014-10-16

description: >
  Wipe and convert all disks to GPT (except the disk containing the root file system)

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: wipe_disk}

  wipe_disk:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: wipe-disk.sh}

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

This Heat template makes reference to a Bash script called **wipe-disk.sh**. This script contains your procedure to wipe the non-root disks. The following script is an example of **wipe-disk.sh** that wipes all disks except for the root disk:

```
#!/bin/bash
if [[ `hostname` = *"ceph"* ]]
then
  echo "Number of disks detected: $(lsblk -no NAME,TYPE,MOUNTPOINT | grep "disk" | awk '{print
$1}' | wc -l)"
  for DEVICE in `lsblk -no NAME,TYPE,MOUNTPOINT | grep "disk" | awk '{print $1}'`
  do
    ROOTFOUND=0
    echo "Checking /dev/$DEVICE..."
    echo "Number of partitions on /dev/$DEVICE: $(expr $(lsblk -n /dev/$DEVICE | awk '{print $7}' | wc
-l) - 1)"
    for MOUNTS in `lsblk -n /dev/$DEVICE | awk '{print $7}'`
    do
      if [ "$MOUNTS" = "/" ]
      then
        ROOTFOUND=1
      fi
    done
    if [ $ROOTFOUND = 0 ]
    then
      echo "Root not found in /dev/${DEVICE}"
      echo "Wiping disk /dev/${DEVICE}"
      sgdisk -Z /dev/${DEVICE}
      sgdisk -g /dev/${DEVICE}
    else
      echo "Root found in /dev/${DEVICE}"
```

```
    fi
  done
fi
```

To include the Heat template in your environment, register it as the **NodeUserData** resource in your **storage-environment.yaml** file:

```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/firstboot/wipe-disks.yaml
```

2.11. CONFIGURING MULTIPLE BONDED INTERFACES PER CEPH NODE

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

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

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

Front-end storage network (StorageNet)

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

Back-end storage network (StorageMgmtNet)

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

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

NOTE

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

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

```
type: ovs_bridge // 1
name: br-bond
members:
  -
    type: ovs_bond // 2
    name: bond1 // 3
    ovs_options: {get_param: BondInterfaceOvsOptions} 4
```

```
    members: // 5
      -
        type: interface
        name: nic2
        primary: true
      -
        type: interface
        name: nic3
  -
    type: vlan // 6
    device: bond1 // 7
    vlan_id: {get_param: StorageNetworkVlanID}
    addresses:
      -
        ip_netmask: {get_param: StorageIpSubnet}
  -
    type: vlan
    device: bond1
    vlan_id: {get_param: StorageMgmtNetworkVlanID}
    addresses:
      -
        ip_netmask: {get_param: StorageMgmtIpSubnet}
```

1. A single bridge named **br-bond** holds the bond defined by this template. This line defines the bridge type, namely OVS.

2. The first member of the **br-bond** bridge is the bonded interface itself, named **bond1**. This line defines the bond type of **bond1**, which is also OVS.

3. The default bond is named **bond1**, as defined in this line.

4. The **ovs_options** entry instructs director to use a specific set of *bonding module directives*. Those directives are passed through the **BondInterfaceOvsOptions**, which you can also configure in this same file. For instructions on how to configure this, see Section 2.11.1, "Configuring Bonding Module Directives".

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

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

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

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

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

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

```
  type: ovs_bridge
  name: br-bond
  members:
    -
      type: linux_bond
      name: bond1
      bonding_options: {get_param: BondInterfaceOvsOptions} // (1)
      members:
        -
          type: interface
          name: nic2
          primary: true
        -
          type: interface
          name: nic3
    -
      type: vlan
      device: bond1
      vlan_id: {get_param: StorageNetworkVlanID}
      addresses:
        -
          ip_netmask: {get_param: StorageIpSubnet}
-
  type: ovs_bridge
  name: br-bond2
  members:
    -
      type: linux_bond
      name: bond2
      bonding_options: {get_param: BondInterfaceOvsOptions}
      members:
        -
          type: interface
          name: nic4
          primary: true
        -
          type: interface
          name: nic5
    -
      type: vlan
      device: bond1
      vlan_id: {get_param: StorageMgmtNetworkVlanID}
      addresses:
        -
          ip_netmask: {get_param: StorageMgmtIpSubnet}
```

(1) As **bond1** and **bond2** are both Linux bonds (instead of OVS), they use **bonding_options** instead of **ovs_options** to set bonding directives. For related information, see Section 2.11.1, "Configuring Bonding Module Directives".

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

2.11.1. Configuring Bonding Module Directives

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

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

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

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

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

2.12. CUSTOMIZING THE CEPH STORAGE CLUSTER

It is possible to override the default configuration parameters for Ceph Storage nodes using the **ExtraConfig** hook to define data to pass to the Puppet configuration. There are two methods to pass this data:

Method 1: Modifying Puppet Defaults

You customize parameters provided to the **ceph** Puppet module during the overcloud configuration. These parameters are a part of the **ceph::profile::params** Puppet class defined in **/etc/puppet/modules/ceph/manifests/profile/params.conf**. For example, following environment file snippet customizes the default **osd_journal_size** parameter from the **ceph::profile::params** class and overrides any default:

```
parameter_defaults:
  ExtraConfig:
    ceph::profile::params::osd_journal_size: 2048
```

Add this content to an environment file (for example, **ceph-settings.yaml**) and include it when you run the **openstack overcloud deploy** command in Section 2.13, "Creating the Overcloud". For example:

```
$ openstack overcloud deploy --templates --ceph-storage-scale <number of nodes> -e
/home/stack/templates/storage-environment.yaml -e /home/stack/templates/ceph-settings.yaml
```

Method 2: Arbitrary Configuration Defaults

If Method 1 does not include a specific parameter you need to configure, it is possible to provide arbitrary Ceph Storage parameters using the **ceph::conf::args** Puppet class. This class accepts parameter names using a **stanza/key** format and **value** to define the parameter's value. These settings configure the **ceph.conf** file on each node. For example, to change the **max_open_files** parameter in the **global** section of the **ceph.conf** file, use the following structure in an environment file:

```
parameter_defaults:
  ExtraConfig:
    ceph::conf::args:
      global/max_open_files:
        value: 131072
```

Add this content to an environment file (for example, **ceph-settings.yaml**) and include it when you run the **openstack overcloud deploy** command in Section 2.13, "Creating the Overcloud". For example:

```
$ openstack overcloud deploy --templates --ceph-storage-scale <number of nodes> -e
/home/stack/templates/storage-environment.yaml -e /home/stack/templates/ceph-settings.yaml
```

The resulting **ceph.conf** file should be populated with the following:

```
[global]
max_open_files = 131072
```

2.12.1. 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 Section 2.12, "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** resource, as in:

```
parameter_defaults:
  CephPools:
    POOL:
      size: 5
      pg_num: 128
      pgp_num: 128
```

Replace *POOL* with the name of the pool you want to configure with the **size**, **pg_num**, and **pgp_num** settings that follow.

2.13. CREATING THE OVERCLOUD

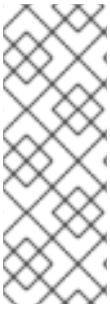
The creation of the Overcloud requires additional arguments for the **openstack overcloud deploy** command. For example:

```
$ openstack overcloud deploy --templates -e /home/stack/templates/storage-environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-scale 3 --control-flavor control --compute-flavor compute --ceph-storage-flavor ceph-storage --ntp-server pool.ntp.org
```

The above command uses the following options:

- **--templates** - Creates the Overcloud from the default Heat template collection.
- **-e /home/stack/templates/storage-environment.yaml** - Adds an additional environment file to the Overcloud deployment. In this case, it is the storage environment file containing our Ceph Storage configuration.
- **--control-scale 3** - Scale the Controller nodes to three.
- **--compute-scale 3** - Scale the Compute nodes to three.
- **--ceph-storage-scale 3** - Scale the Ceph Storage nodes to three.
- **--control-flavor control** - Use a specific flavor for the Controller nodes.
- **--compute-flavor compute** - Use a specific flavor for the Compute nodes.
- **--ceph-storage-flavor ceph-storage** - Use a specific flavor for the Compute nodes.
- **--ntp-server pool.ntp.org** - Sets our NTP server.

See Appendix A, *Sample Environment File: Creating a Ceph Cluster* for an overview of all the settings used in **/home/stack/templates/storage-environment.yaml**.



NOTE

For a full list of options, run:

```
$ openstack help overcloud deploy
```

For more information, see Setting Overcloud Parameters 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
$ heat stack-list --show-nested
```

2.14. ACCESSING THE OVERCLOUD

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

```
$ source ~/overcloudrc
```

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

```
$ source ~/stackrc
```

2.15. MONITORING CEPH STORAGE NODES

After completing the Overcloud creation, it is recommended to check the status of the Ceph Storage Cluster to make sure it is working properly. To accomplish this, log into a Controller node as the **heat-admin** user from the director.

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

Check the health of the cluster:

```
$ sudo ceph health
```

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

Check the status of the Ceph Monitor quorum:

```
$ sudo ceph quorum_status
```

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

Check if all Ceph OSDs are running:

```
$ ceph osd stat
```

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

2.16. REBOOTING THE ENVIRONMENT

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

Make sure to boot the nodes in the following order:

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

- **Boot all Ceph Storage nodes**- This ensures the Ceph OSD cluster can connect to the active Ceph Monitor cluster on the Controller nodes.

Use the following process to reboot the Ceph Storage nodes:

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

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

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

3. Reboot the node:

```
$ sudo reboot
```

4. Wait until the node boots.

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

```
$ sudo ceph -s
```

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

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

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

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

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

```
$ sudo ceph status
```

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

```
$ sudo systemctl restart 'ceph*'
```

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

```
$ sudo ceph status
```

2.17. REPLACING CEPH STORAGE NODES

If a Ceph Storage node fails, you must disable and rebalance the faulty node before removing it from the overcloud to prevent data loss. This procedure explains the process for replacing a Ceph Storage node.

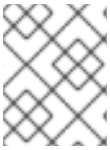
NOTE

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

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

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

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



NOTE

In the example, host overcloud-cephstorage-0 hosts two OSDs: osd.0 and osd.1. Adapt this procedure to suit your environment.

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

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

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

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

After the Ceph cluster finishes rebalancing, log in to the faulty Ceph Storage node as the **heat-admin** user and stop the node.

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

Prevent the OSDs from starting during the next reboot.

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

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

Remove the Ceph Storage node from the CRUSH map so that it no longer receives data.

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

Remove the OSD authentication key.

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

Remove the OSD from the cluster.

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

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

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

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

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

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

```
$ heat stack-list
```

Identify the UUIDs of the Ceph Storage node to delete.

```
$ nova list
```

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

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

IMPORTANT

If you passed any extra environment files when you created the overcloud, pass them again here using the **-e** or **--environment-file** option to avoid making undesired changes to the overcloud.

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

Add new nodes to the director's node pool and deploy them as Ceph Storage nodes. Use the **--ceph-storage-scale** option to define the total number of Ceph Storage nodes in the overcloud. For example, if you removed a faulty node from a three-node cluster and you want to replace it, use **--ceph-storage-scale 3** to return the number of Ceph Storage nodes to its original value.

```
$ openstack overcloud deploy --templates --ceph-storage-scale 3 -e [ENVIRONMENT_FILES]
```

IMPORTANT

If you passed any extra environment files when you created the overcloud, pass them again here using the **-e** or **--environment-file** option to avoid making undesired changes to the overcloud.

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

Log in to a Controller node as the **heat-admin** user and check the status of the Ceph Storage node.

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

Confirm that the value in the **osdmap** section matches the number of desired nodes in your cluster.

The failed Ceph Storage node has now been replaced with a new node.

2.18. ADDING AND REMOVING OSD DISKS FROM CEPH STORAGE NODES

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

- "Adding an OSD"
- "Removing an OSD"

CHAPTER 3. INTEGRATING AN EXISTING CEPH STORAGE CLUSTER WITH AN OVERCLOUD

This chapter describes how to create an Overcloud and integrate it with an existing Ceph Storage Cluster. For instructions on how to create both Overcloud and Ceph Storage Cluster, see Chapter 2, *Creating an Overcloud with Ceph Storage Nodes* instead.

The scenario described in this chapter consists of six nodes in the Overcloud:

- Three Controller nodes with high availability.
- Three Compute nodes.

The director will integrate a separate Ceph Storage Cluster with its own nodes into the Overcloud. You manage this cluster independently from the Overcloud. For example, you scale the Ceph Storage cluster using the Ceph management tools and not through the OpenStack Platform director.

All OpenStack machines are bare metal systems using IPMI for power management. These nodes do not require an operating system because the director copies a Red Hat Enterprise Linux 7 image to each node.

The director communicates to the Controller and Compute nodes through the Provisioning network during the introspection and provisioning processes. All nodes connect to this network through the native VLAN. For this example, we use 192.0.2.0/24 as the Provisioning subnet with the following IP address assignments:

Node Name	IP Address	MAC Address	IPMI IP Address	
Director	192.0.2.1	aa:aa:aa:aa:aa:aa		
Controller 1	DHCP defined	b1:b1:b1:b1:b1:b1	192.0.2.205	
Controller 2	DHCP defined	b2:b2:b2:b2:b2:b2	192.0.2.206	
Controller 3	DHCP defined	b3:b3:b3:b3:b3:b3	192.0.2.207	
Compute 1	DHCP defined	c1:c1:c1:c1:c1:c1	192.0.2.208	
Compute 2	DHCP defined	c2:c2:c2:c2:c2:c2	192.0.2.209	
Compute 3	DHCP defined	c3:c3:c3:c3:c3:c3	192.0.2.210	

3.1. CONFIGURING THE EXISTING CEPH STORAGE CLUSTER

1. Create the following pools in your Ceph cluster relevant to your environment:
   - **volumes**: Storage for OpenStack Block Storage (cinder)
   - **images**: Storage for OpenStack Image Storage (glance)
   - **vms**: Storage for instances

- **backups**: Storage for OpenStack Block Storage Backup (cinder-backup)
- **metrics**: Storage for OpenStack Telemetry Metrics (gnocchi)
  Use the following commands as a guide:

```
[root@ceph ~]# ceph osd pool create volumes PGNUM
[root@ceph ~]# ceph osd pool create images PGNUM
[root@ceph ~]# ceph osd pool create vms PGNUM
[root@ceph ~]# ceph osd pool create backups PGNUM
[root@ceph ~]# ceph osd pool create metrics PGNUM
```

  Replace *PGNUM* with the number of *placement groups*. We recommend approximately 100 per OSD. For example, the total number of OSDs multiplied by 100 divided by the number of replicas (**osd pool default size**). You can also use the Ceph Placement Groups (PGs) per Pool Calculator to determine a suitable value.

2. Create a **client.openstack** user in your Ceph cluster with the following capabilities:
   - cap_mon: **allow r**
   - cap_osd: **allow class-read object_prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow rwx pool=backups, allow rwx pool=metrics**
     Use the following command as a guide:

```
[root@ceph ~]# ceph auth add client.openstack mon 'allow r' osd 'allow class-read object_prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow rwx pool=backups, allow rwx pool=metrics'
```

3. Next, note the *Ceph client key* created for the **client.openstack** user:

```
[root@ceph ~]# ceph auth list
...
client.openstack
 key: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
 caps: [mon] allow r
 caps: [osd] allow class-read object_prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow rwx pool=backups, allow rwx pool=metrics
...
```

   The **key** value here (**AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==**) is your Ceph client key.

4. Finally, note the *file system ID* of your Ceph Storage cluster. This value is specified with the **fsid** setting in the configuration file of your cluster (under the **[global]** section):

```
[global]
fsid = 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
...
```

NOTE

For more information about the Ceph Storage cluster configuration file, see Configuration Reference (from the Red Hat Ceph Storage Configuration Guide).

The Ceph client key and file system ID will both be used later in Section 3.6, "Enabling Integration with the Existing Ceph Storage Cluster".

3.2. INITIALIZING THE STACK USER

Log into the director host as the **stack** user and run the following command to initialize your director configuration:

```
$ source ~/stackrc
```

This sets up environment variables containing authentication details to access the director's CLI tools.

3.3. REGISTERING NODES

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

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        },
        {
            "mac":[
                "dd:dd:dd:dd:dd:dd"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
```

```
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.207"
        },
        {
            "mac":[
                "ee:ee:ee:ee:ee:ee"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.208"
        }
        {
            "mac":[
                "ff:ff:ff:ff:ff:ff"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.209"
        }
        {
            "mac":[
                "gg:gg:gg:gg:gg:gg"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.210"
        }
    ]
}
```

After creating the template, save the file to the stack user's home directory (/**home**/**stack**/**instackenv.json**), then import it into the director. Use the following command to accomplish this:

```
$ openstack baremetal import --json ~/instackenv.json
```

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

Assign the kernel and ramdisk images to all nodes:

```
$ openstack baremetal configure boot
```

The nodes are now registered and configured in the director.

3.4. INSPECTING THE HARDWARE OF NODES

After registering the nodes, inspect the hardware attribute of each node. Run the following command to inspect the hardware attributes of each node:

```
$ openstack baremetal introspection bulk start
```

IMPORTANT

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

3.5. MANUALLY TAGGING THE NODES

After registering and inspecting the hardware of each node, tag them into specific profiles. These profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role.

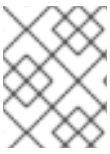
Retrieve a list of your nodes to identify their UUIDs:

```
$ ironic node-list
```

To manually tag a node to a specific profile, add a profile option to the **properties/capabilities** parameter for each node. For example, to tag three nodes to use a controller profile and one node to use a compute profile, use the following commands:

```
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
```

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



NOTE

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

3.6. ENABLING INTEGRATION WITH THE EXISTING CEPH STORAGE CLUSTER

Create a copy of **/usr/share/openstack-tripleo-heat-templates/environments/puppet-ceph-external.yaml** to a directory in your **stack** user's home directory:

```
[stack@director ~]# mkdir templates
[stack@director ~]# cp /usr/share/openstack-tripleo-heat-templates/environments/puppet-ceph-external.yaml ~/templates/.
```

Edit the file and set the following parameters:

- Set the **CephExternal** resource definition to an absolute path:

```
OS::TripleO::Services::CephExternal: /usr/share/openstack-tripleo-heat-templates/puppet/services/ceph-external.yaml
```

- Set the following three parameters using your Ceph Storage environment details:
  - **CephClientKey**: the Ceph client key of your Ceph Storage cluster. This is the value of **key** you retrieved earlier in Section 3.1, "Configuring the Existing Ceph Storage Cluster" (for example, **AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==**).
  - **CephExternalMonHost**: a comma-delimited list of the IPs of all MON hosts in your Ceph Storage cluster.
  - **CephClusterFSID**: the file system ID of your Ceph Storage cluster. This is the value of **fsid** in your Ceph Storage cluster configuration file, which you retrieved earlier in Section 3.1, "Configuring the Existing Ceph Storage Cluster" (for example, **4b5c8c0a-ff60-454b-a1b4-9747aa737d19**).
- If necessary, also set the name of the OpenStack pools and the client user using the following parameters and values:
  - **CephClientUserName: openstack**
  - **NovaRbdPoolName: vms**
  - **CinderRbdPoolName: volumes**
  - **GlanceRbdPoolName: images**
  - **CinderBackupRbdPoolName: backups**
  - **GnocchiRbdPoolName: metrics**

3.7. BACKWARDS COMPATIBILITY WITH OLDER VERSIONS OF RED HAT CEPH STORAGE

If you are integrating Red Hat OpenStack Platform with an external Ceph Storage Cluster from an earlier version (that is, Red Hat Ceph Storage 1.3), you need to enable backwards compatibility. To do so, first create an environment file in **/home/stack/templates/** containing the following:

```
parameter_defaults:
  ExtraConfig:
    ceph::conf::args:
      client/rbd_default_features:
        value: "1"
```

Include this file in your overcloud deployment, described in Section 3.8, "Creating the Overcloud".

3.8. CREATING THE OVERCLOUD

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

```
$ openstack overcloud deploy --templates -e /home/stack/templates/puppet-ceph-external.yaml --control-scale 3 --compute-scale 3 --ceph-storage-scale 0 --control-flavor control --compute-flavor compute --neutron-network-type vxlan --ntp-server pool.ntp.org
```

The above command uses the following options:

- **--templates** - Creates the Overcloud from the default Heat template collection.
- **-e /home/stack/templates/puppet-ceph-external.yaml** - Adds an additional environment file to the Overcloud deployment. In this case, it is the storage environment file containing the configuration for the existing Ceph Storage Cluster.
- **--control-scale 3** - Scale the Controller nodes to three.
- **--compute-scale 3** - Scale the Compute nodes to three.
- **--ceph-storage-scale 0** - Scale the Ceph Storage nodes to zero. This ensures the director does not create any Ceph Storage nodes.
- **--control-flavor control** - Use a specific flavor for the Controller nodes.
- **--compute-flavor compute** - Use a specific flavor for the Compute nodes.
- **--neutron-network-type vxlan** - Sets the **neutron** networking type.
- **--ntp-server pool.ntp.org** - Sets our NTP server.

NOTE

For a full list of options, run:

```
$ openstack help overcloud deploy
```

For more information, see Setting Overcloud Parameters 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
$ heat stack-list --show-nested
```

This configures the Overcloud to use your external Ceph Storage cluster. Note that you manage this cluster independently from the Overcloud. For example, you scale the Ceph Storage cluster using the Ceph management tools and not through the OpenStack Platform director.

3.9. ACCESSING THE OVERCLOUD

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

```
$ source ~/overcloudrc
```

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

```
$ source ~/stackrc
```

CHAPTER 4. CONCLUSION

This concludes the creation and configuration of Overcloud with Red Hat Ceph Storage. For general Overcloud post-creation functions, see Performing Tasks after Overcloud Creation in the Director Installation and Usage guide.

APPENDIX A. SAMPLE ENVIRONMENT FILE: CREATING A CEPH CLUSTER

The following custom environment file uses many of the options described throughout Chapter 2, *Creating an Overcloud with Ceph Storage 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-environment.yaml

```
resource_registry: // 1
  OS::TripleO::Services::CephMon: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-osd.yaml
  OS::TripleO::Services::CephClient: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-client.yaml
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/puppet/services/pacemaker/cinder-backup.yaml  // 2
  OS::TripleO::Services::CephRgw: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-rgw.yaml  // 3
  OS::TripleO::Services::SwiftProxy: OS::Heat::None
  OS::TripleO::Services::SwiftStorage: OS::Heat::None
  OS::TripleO::Services::SwiftRingBuilder: OS::Heat::None
  OS::TripleO::NodeUserData: /home/stack/templates/firstboot/wipe-disks.yaml // 4

parameter_defaults: // 5
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderBackupBackend: swift // 6
  ExtraConfig:
    ceph::profile::params::osds: // 7
      '/dev/sdc':
        journal: '/dev/sdb'
      '/dev/sdd':
        journal: '/dev/sdb'
    CephPools: // 8
      volumes:
        size: 5
        pg_num: 128
        pgp_num: 128
```

1. The **resource_registry** section defines resources linked to heat templates. The first three entries (**CephMon**, **CephOSD**, and **CephClient**) link the heat templates used to define the different components of the Ceph cluster (MON, OSD, and client).

2. The **OS::TripleO::Services::CinderBackup** entry calls the heat template required to deploy the Block Storage Backup service. You can set the backup target later on in the **parameter_defaults** section.

3. The **OS::TripleO::Services::CephRgw** entry calls the heat template required to deploy the Ceph Object Gateway, which in turn provides the means for OpenStack to use Ceph Object Storage.

4. The **OS::TripleO::NodeUserData:** entry uses a template (**/home/stack/templates/firstboot/wipe-disks.yaml**) that checks all disks on Ceph Storage node and converts all of them (except the disk containing the root file system) to GPT using a custom script. For more details, see Section 2.10, "Formatting Ceph Storage Node Disks to GPT".

5. The **parameter_defaults** section modifies the default values for parameters in all templates. Most of the entries listed here are described in Section 2.6, "Enabling Ceph Storage in the Overcloud" .

6. As you are deploying the Ceph Object Gateway, you can then use Ceph Object Storage as a backup target. To configure this, set **CinderBackupBackend** to **swift**. See Section 2.8, "Deploy the Ceph Object Gateway" for details.

7. The **ceph::profile::params::osds::** section defines a custom disk layout, as described in Section 2.7, "Mapping the Ceph Storage Node Disk Layout" .

8. The **CephPools** section sets custom attributes for any Ceph pool. This example sets custom **size**, **pg_num**, and **pgp_num** attributes for the **volumes** pool. See Section 2.12.1, "Assigning Custom Attributes to Different Ceph Pools" for more details.

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}
```