Red Hat Ceph Storage 2

Ceph Object Gateway for Production

Planning, designing and deploying Ceph Storage clusters and Ceph Object Gateway clusters for production.

Last Updated: 2018-04-03

Red Hat Ceph Storage 2 Ceph Object Gateway for Production

Planning, designing and deploying Ceph Storage clusters and Ceph Object Gateway clusters for production.

Abstract

This guide covers planning a cluster, considering hardware, developing storage strategies, configuring gateways and load balancers and using the Ceph Object Gateway.

Table of Contents

CHAPTER 1. INTRODUCTION

Welcome to the **Ceph Object Gateway for Production** guide. This guide covers topics for building Ceph Storage clusters and Ceph Object Gateway clusters for production use.

1.1. AUDIENCE

This guide is for those who intend to deploy a Ceph Object Gateway environment for production. It provides a sequential series of topics for planning, designing and deploying a production Ceph Storage cluster and Ceph Object Gateway cluster with links to general Ceph documentation where appropriate.

1.2. ASSUMPTIONS

This guide assumes the reader has a basic understanding of the Ceph Storage Cluster and the Ceph Object Gateway. Readers with no Ceph experience should consider setting up a small Ceph test environment or using the Ceph Sandbox Environment to get familiar with Ceph concepts before proceeding with this guide.

This guide assumes a single-site cluster consisting of a single Ceph Storage cluster and multiple Ceph Object Gateway instances in the same zone. This guide assumes the single-site cluster will expand to a multi-zone and multi-site cluster by repeating the procedures in this guide for each zone group and zone with the naming modifications required for secondary zone groups and zones.

1.3. SCOPE

This guide covers the following topics when setting up a Ceph Storage Cluster and a Ceph Object Gateway for production:

- Planning a Cluster
- Considering Hardware
- Deploying a Cluster
- Developing Storage Strategies
- Configuring Gateways
- Additional Use Cases

NOTE

This document is intended to complement the hardware, installation, administration and Ceph Object Gateway guides. This guide does not replace the other guides.

CHAPTER 2. PLANNING A CLUSTER

Planning a cluster for use with the Ceph Object Gateway involves several important considerations:

- Identifying use cases
- Selecting data durability methods
- Considering multi-site deployment

These factors will have a significant influence when considering hardware. Consider these factors carefully before selecting hardware.

2.1. IDENTIFYING USE CASES

Ceph Storage is capable of serving many different types of storage use cases. For Ceph Object Storage, the typical use cases are:

- **Throughput-optimized:** A throughput-optimized cluster seeks to ensure fast data access. Host bus adapters (HBAs), storage media with fast sequential read/write characteristics and high network bandwidth provide capability for applications such as graphics, streaming audio and streaming video. Throughput-optimized optimized clusters also consider whether write performance is a consideration. Throughput-optimized clusters that use SSDs for journaling realize substantially better write performance, which can be important for applications like storing CCTV streams. Throughput-optimized clusters should consider the throughput characteristics of a Host Bus Adapter (HBA) controller and network throughput for intensive applications such as streaming 4K video. HBA-based hardware controllers offer significant performance improvements over on-board controllers.
- **Capacity-optimized:** A capacity-optimized cluster seeks to ensure the lowest cost per terabyte of storage. Capacity-optimized clusters often use the least expensive storage media and often avoid the added expense of separate SSD journals for applications such as archiving infrequently accessed legacy financial records, old emails, etc.
- **IOPS-optimized:** An IOPS-optimized cluster seeks to provide high performance for read- and write-intensive workloads. While IOPS-optimized workloads are not as common for Ceph Object Gateways, they can be supported with SSD, Flash memory or NVMe CRUSH hierarchies.

Carefully consider the storage use case(s) **BEFORE** considering hardware, because it can significantly impact the price and performance of the cluster. For example, if the use case is capacity-optimized and the hardware is better suited to a throughput-optimized use case, the hardware will be more expensive than necessary. Conversely, if the use case is throughput-optimized and the hardware is better suited to a capacity-optimized use case, the cluster may suffer from poor performance.

Also, note that since Ceph Object Gateway supports storage policies, it is possible to create CRUSH hierarchies for **ALL** of the foregoing scenarios and invoke them with storage policies supported in the APIs. See Creating Data Placement Strategies for details.

2.2. SELECTING DATA DURABILITY METHODS

Cluster design should also consider the data durability strategy. Ceph Storage uses either replication or erasure coding to ensure data durability.

Replication stores one or more redundant copies of the data across failure domains in case of a hardware failure. However, redundant copies of data can become expensive at scale. For example, to store 1 petabyte of data with triple replication would require a cluster with at least 3 petabytes of storage capacity.

The Erasure coding section of the Storage Strategies Guide describes how erasure coding stores data as data chunks and coding chunks. In the event of a lost data chunk, erasure coding can recover the lost data chunk with the remaining data chunks and coding chunks. Erasure coding is substantially more economical than replication. For example, using erasure coding with 8 data chunks and 3 coding chunks provides the same redundancy as 3 copies of the data. However, such an encoding scheme uses approximately 1.5x of the initial data stored compared to 3x with replication.

NOTE

ONLY the data storage pool can use erasure coding. Pools storing system data and bucket indexes use replication.

2.3. CONSIDERING MULTI-SITE DEPLOYMENT

Another important aspect of designing a cluster is to determine if the cluster will be in one data center site or span multiple data center sites. Multi-site clusters benefit from geographically distributed failover and disaster recovery, such as long-term power outages, earthquakes, hurricanes, floods or other disasters. Additionally, multi-site clusters in an active-active configuration can direct client applications to the closest available cluster in the manner of content delivery networks. Placing data as close to the client as possible is increasingly important for throughput intensive workloads such as streaming 4k video.

For details of multi-site clusters, see the Multi-site sections of the Object Gateway for Red Hat Enterprise Linux guide.

NOTE

Red Hat recommends identifying realm, zone group and zone names BEFORE creating Ceph Storage pools. Some pool names should be pre-pended with the zone name by convention.

CHAPTER 3. CONSIDERING HARDWARE

Considering hardware is an important part of building Ceph Storage clusters and Ceph Object Gateway clusters for production environments. High-level considerations include:

- Considering Storage Sizing
- Considering Storage Density
- Considering Uninterrupted Power Supplies
- Considering Network Hardware
- Selecting Hardware for Use Cases
- Selecting SSDs for Bucket Indexes
- Selecting SSDs for Monitor Nodes

IMPORTANT

Consider these factors **BEFORE** identifying and purchasing computing and networking hardware for the cluster.

3.1. CONSIDERING STORAGE SIZING

One of the most important factors in designing a cluster is to determine the storage requirements (sizing). Ceph Storage is designed to scale into petabytes and beyond. The following examples are common sizes for Ceph storage clusters.

- **Small:** 250 terabytes
- **Medium:** 1 petabyte
- **Large:** 2 petabytes or more.

Sizing should include current needs and the needs of the near future. Consider the rate at which the gateway client will add new data to the cluster. That may differ from use-case to use-case. For example, recording CCTV video, 4k video or medical imaging may add significant amounts of data far more quickly then less storage intensive information such as financial market data. Additionally, consider that data durability methods such as replication versus erasure coding will have a significant impact on the storage media required.

For additional information on sizing, see the Red Hat Ceph Storage Hardware Guide and its associated links for selecting OSD hardware.

3.2. CONSIDERING STORAGE DENSITY

Another important aspect of cluster design includes storage density. Generally, a cluster should store data across at least 10 nodes to ensure reasonable performance when replicating, backfilling and recovery. If a node fails, with at least 10 nodes in the cluster, only 10% of the data has to move to the surviving nodes. If the number of nodes is substantially less, a higher percentage of the data must move to the surviving nodes. Additionally, the `full_ratio` and `near_full_ratio` need to be set to accommodate a node failure to ensure that the cluster can write data. For this reason, it is is important to consider storage density. Higher storage density isn’t necessarily a good idea.

Another factor that favors more nodes over higher storage density is erasure coding. When writing an object using erasure coding and using `node` as the minimum CRUSH failure domain, the cluster will need as many nodes as data and coding chunks. For example, a cluster using `k=8, m=3` should have at least 11 nodes so that each data or coding chunk is stored on a separate node.

Hot-swapping is also an important consideration. Most modern servers support drive hot-swapping. However, some hardware configurations require removing more than one drive to replace a drive. Red Hat recommends avoiding such configurations, because they can bring down more OSDs than required when swapping out failed disks.

3.3. CONSIDERING NETWORK HARDWARE

A major advantage of Ceph Storage is that it allows scaling capacity, IOPS and throughput independently. An important aspect of a cloud storage solution is that clusters can run out of IOPS due to network latency and other factors or run out of throughput due to bandwidth constraints long before the clusters run out of storage capacity. This means that the network hardware configuration must support the use case(s) in order to meet price/performance targets. Network performance is increasingly important when considering the use of SSDs, flash, NVMe, and other high performance storage methods.

Another important consideration of Ceph Storage is that it supports a front side or public network for client and monitor data, and a back side or cluster network for heart beating, data replication and recovery. This means that the back side or cluster network will **always** require more network resources than the front side or public network. Depending upon whether the data pool uses replication or erasure coding for data durability, the network requirements for the back side or cluster network should be quantified appropriately.

Finally, verify network throughput before installing and testing Ceph. Most performance-related problems in Ceph usually begin with a networking issue. Simple network issues like a kinked or bent Cat-6 cable could result in degraded bandwidth. Use a minimum of 10Gbe for the front side network. For large clusters, consider using 40Gbe for the backend or cluster network. Alternatively, use LCAP mode 4 to bond networks. Additionally, use jumbo frames (MTU 9000), especially on the backend or cluster network.

3.4. CONSIDERING UNINTERRUPTED POWER SUPPLIES

Since Ceph writes are atomic—all or nothing—it isn't a requirement to invest in uninterruptable power supplies (UPS) for Ceph OSD nodes. However, Red Hat recommends investing in UPSs for Ceph Monitor nodes. Monitors use `leveldb`, which is sensitive to synchronous write latency. A power outage could cause corruption, requiring technical support to restore the state of the cluster.

Ceph OSDs may benefit from the use of a UPS if a storage controller uses a writeback cache. In this scenario, a UPS may help prevent filesystem corruption during a power outage if the controller doesn't flush the writeback cache in time.

3.5. SELECTING HARDWARE FOR USE CASES

A major advantage of Ceph Storage is that it can be configured to support many use cases. Generally, Red Hat recommends configuring OSD hosts identically for a particular use case. The three primary use cases for a Ceph Storage cluster are:

- IOPS optimized
- Throughput optimized

- Capacity optimized

Since these use cases typically have different drive, HBA controller and networking requirements among other factors, configuring a series of identical hosts to facilitate all of these use cases with a single node configuration is possible, but is not necessarily recommended.

Using the same hosts to facilitate multiple CRUSH hierarchies will involve the use of logical, rather than actual host names in the CRUSH map. Additionally, deployment tools such as Ansible would need to consider a group for each use case, rather than deploying all OSDs in the default `[osds]` group.

NOTE

Generally, it is easier to configure and manage hosts that serve a single use case, such as high IOPS, high throughput, or high capacity.

3.6. SELECTING SSDS FOR BUCKET INDEXES

When selecting OSD hardware for use with a Ceph Object Gateway—irrespective of the use case—Red Hat recommends considering an OSD node that has at least one SSD drive used **exclusively** for the bucket index pool. This is particularly important when buckets will contain a large number of objects.

A bucket index entry is approximately 200 bytes of data, stored as an object map (omap) in `leveldb`. While this is a trivial amount of data, some uses of Ceph Object Gateway can result in tens or hundreds of millions of objects in a single bucket. By mapping the bucket index pool to a CRUSH hierarchy of SSD nodes, the reduced latency provides a dramatic performance improvement when buckets contain very large numbers of objects.

IMPORTANT

In a production cluster, a typical OSD node will have at least one SSD for the bucket index, AND at least on SSD for the journal.

3.7. SELECTING SSDS FOR MONITOR NODES

Ceph monitors use `leveldb`, which is sensitive to synchronous write latency. Red Hat strongly recommends using SSDs to store monitor data. Ensure that the selected SSDs have sufficient sequential write and throughput characteristics.

CHAPTER 4. DEPLOYING A CLUSTER

The initial deployment of a production cluster is identical to deploying a proof-of-concept system. The only material difference is that the initial deployment will use production-grade hardware. First, follow the prerequisites section of the Installation Guide for Red Hat Enterprise Linux and execute the appropriate steps for each node. The following sections provide additional guidance relevant to production clusters.

4.1. NAMING HOSTS

When naming hosts, consider their use case and performance profile. For example, if the hosts will store client data, consider naming them according to their hardware configuration and performance profile. For example:

- `data-ssd-1`, `data-ssd-2`
- `hot-storage-1`, `hot-storage-2`
- `sata-1`, `sata-2`
- `sas-ssd-1`, `sas-ssd-2`

The naming convention may make it easier to manage the cluster and troubleshoot hardware issues as they arise.

If the host contains hardware for multiple use cases—for example, the host contains SSDs for data, SAS drives with SSDs for journals, and SATA drives with co-located journals for cold storage—choose a generic name for the host. For example:

- `osd-node-1` `osd-node-2`

Generic host names can be extended when using logical host names in the CRUSH hierarchy as needed. For example:

- `osd-node-1-ssd` `osd-node-1-sata` `osd-node-1-sas-ssd` `osd-node-1-bucket-index`
- `osd-node-2-ssd` `osd-node-2-sata` `osd-node-2-sas-ssd` `osd-node-2-bucket-index`

See Using Logical Host Names in a CRUSH Map for additional details.

4.2. TUNING THE KERNEL

Production clusters benefit from tuning the operating system, specifically limits and memory allocation. Ensure that adjustments are set for all nodes within the cluster. Consult Red Hat support for additional guidance.

4.2.1. Adjusting TCMalloc

Under heavy multi-threaded memory allocation workloads, TCMalloc can consume significant amounts of CPU and reduce IOPS when it doesn't have enough thread cache available. Red Hat recommends increasing the amount of thread cache beyond the default 32MB.

To change the TCMalloc cache setting, edit **`/etc/sysconfig/ceph`**, and use the **`TCMALLOC_MAX_TOTAL_THREAD_CACHE_BYTES`** setting to adjust the cache size. For example, increasing the cache from 64MB to 128MB can substantially increase IOPS while reducing CPU overhead.

To release the memory that TCMalloc has allocated, but which is not being used by the Ceph daemon itself, execute the following:

```
# ceph tell osd.* heap release
```

4.2.2. Reserving Free Memory for OSDs

To help prevent insufficient memory-related errors during OSD memory allocation requests, set the **`vm.min_free_kbytes`** option in the **`sysctl.conf`** file on OSD nodes. This option specifies the amount of physical memory to keep in reserve. The recommended settings are based on the amount of system RAM. For example:

- For 64GB RAM, reserve 1GB.

  ```
  vm.min_free_kbytes = 1048576
  ```

- For 128GB RAM, reserve 2GB.

  ```
  vm.min_free_kbytes = 2097152
  ```

- For 256GB RAM, reserve 3GB.

  ```
  vm.min_free_kbytes = 3145728
  ```

4.2.3. Increasing File Descriptors

The Ceph Object Gateway may hang if it runs out of file descriptors. Modify **`/etc/security/limits.conf`** on Ceph Object Gateway nodes to increase the file descriptors for the Ceph Object Gateway. For example:

```
ceph       soft    nproc     unlimited
```

4.2.4. Adjusting ulimit On Large Clusters

For system administrators that will run Ceph administrator commands on large clusters—for example, 1024 OSDs or more—create an **`/etc/security/limits.d/50-ceph.conf`** file on each node that will run administrator commands with the following contents:

```
<username>       soft    nproc     unlimited
```

Replace **`<username>`** with the name of the non-root account that will run Ceph administrator commands.

NOTE

The root user’s ulimit is already set to "unlimited" by default on RHEL.

4.2.5. Adjusting PID Count

Hosts with high numbers of OSDs may spawn a lot of threads, especially during recovery and re-balancing. Many Linux kernels default to a relatively small maximum number of threads. Check the default settings to see if they are suitable.

```
cat /proc/sys/kernel/pid_max
```

Consider setting **`kernel.pid_max`** to a higher number of threads. The theoretical maximum is 4,194,303 threads. For example, add the following to the **`/etc/sysctl.conf`** file to set it to the maximum:

```
kernel.pid_max = 4194303
```

To effect the changes without rebooting, execute:

```
# sysctl -p
```

To verify the changes, execute:

```
# sysctl -a | grep kernel.pid_max
```

4.3. CONFIGURING ANSIBLE GROUPS

This procedure is only pertinent for deploying Ceph using Ansible. The **`ceph-ansible`** package is already configured with a default **`osds`** group. If the cluster will only have one use case and storage policy, proceed with the procedure documented in the Installing Ceph Using Ansible section of the Installation Guide for Red Hat Enterprise Linux.

If the cluster will support multiple use cases and storage policies, create a group for each one. See Configuring OSD Settings section of the Installation Guide for Red Hat Enterprise Linux for high level details.

Each use case should copy **`/usr/share/ceph-ansible/group_vars/osd.sample`** to a file named for the group name. For example, if the cluster has IOPS-optimized, throughput-optimized and capacity-optimized use cases, create separate files representing the groups for each use case. For example:

```
cd /usr/share/ceph-ansible/group_vars/
cp osds.sample osds-iops
cp osds.sample osds-throughput
cp osds.sample osds-capacity
```

Then, configure each file according to the use case.

Once the group variable files are configured, edit the **`site.yml`** file to ensure that it includes each new group. For example:

```
- hosts: osds-iops
  gather_facts: false
  become: True
  roles:
  - ceph-osd

- hosts: osds-throughput
  gather_facts: false
```

```
    become: True
    roles:
    - ceph-osd

  - hosts: osds-capacity
    gather_facts: false
    become: True
    roles:
    - ceph-osd
```

Finally, in the **`/etc/ansible/hosts`** file, place the OSD nodes associated to a group under the corresponding group name. For example:

```
[osds-iops]
<ceph-host-name> devices="[ '<device_1>', '<device_2>' ]"

[osds-throughput]
<ceph-host-name> devices="[ '<device_1>', '<device_2>' ]"

[osds-capacity]
<ceph-host-name> devices="[ '<device_1>', '<device_2>' ]"
```

4.4. DEPLOYING CEPH

Once the pre-requisites and initial tuning are complete, consider deploying a Ceph cluster. When deploying a production cluster, Red Hat recommends setting up the initial monitor cluster and enough OSD nodes to reach an **`active + clean`** state. See Storage Cluster Installation for details.

Then, install the Ceph CLI client on an administration node. See Ceph CLI installation for details.

Once the initial cluster is running, consider adding the settings in the following sections to the Ceph configuration file.

NOTE

If deployment uses a tool such as Ansible, add the following settings to the deployment tool's configuration. For example, see overriding Ceph default settings for examples on how to modify Ceph settings using Ansible.

4.4.1. Setting the Journal Size

Set the journal size for the Ceph cluster. Configuration tools such as Ansible may have a default value. Generally, the journal size should find the product of the synchronization interval and the slower of the disk and network throughput, and multiply the product by two (2).

See Journal Settings for details.

4.4.2. Adjusting Backfill & Recovery Settings

I/O is negatively impacted by both backfilling and recovery operations, leading to poor performance and unhappy end users. To help accommodate I/O demand during a cluster expansion or recovery, set the following options and values in the Ceph Configuration file:

```
[osd]
```

```
osd_max_backfills = 1
osd_recovery_max_active = 1
osd_recovery_op_priority = 1
```

4.4.3. Adjusting the Cluster Map Size

When the cluster has thousands of OSDs, download the cluster map and check its file size. By default, the **`ceph-osd`** daemon caches 500 previous osdmaps. Even with deduplication, the map may consume a lot of memory per daemon. Tuning the cache size in the Ceph configuration file may help reduce memory consumption significantly. For example:

```
[global]
osd_map_message_max=10

[osd]
osd_map_cache_size=20
osd_map_max_advance=10
osd_map_share_max_epochs=10
osd_pg_epoch_persisted_max_stale=10
```

4.4.4. Adjusting Scrubbing

By default, Ceph performs light scrubbing daily and deep scrubbing weekly. Light scrubbing checks object sizes and checksums to ensure that PGs are storing the same object data. Over time, disk sectors can go bad irrespective of object sizes and checksums. Deep scrubbing checks an object's content with that of its replicas to ensure that the actual contents are the same. In this respect, deep scrubbing ensures data integrity in the manner of **`fsck`**, but the procedure imposes an I/O penalty on the cluster. Even light scrubbing can impact I/O.

The default settings may allow Ceph OSDs to initiate scrubbing at inopportune times such as peak operating times or periods with heavy loads. End users may experience latency and poor performance when scrubbing operations conflict with end user operations.

To prevent end users from experiencing poor performance, Ceph provides a number of scrubbing settings that can limit scrubbing to periods with lower loads or during off-peak hours. See scrubbing for details.

If the cluster experiences high loads during the day and low loads late at night, consider restricting scrubbing to night time hours. For example:

```
[osd]
osd_scrub_begin_hour = 23   #23:01H, or 10:01PM.
osd_scrub_end_hour = 6      #06:01H or 6:01AM.
```

If time constraints aren't an effective method of determining a scrubbing schedule, consider using the **`osd_scrub_load_threshold`**. The default value is **`0.5`**, but it could be modified for low load conditions. For example:

```
[osd]
osd_scrub_load_threshold = 0.25
```

4.4.5. Expanding the Cluster

Once the initial cluster is running and in an **`active+clean`** state, add additional OSD nodes and Ceph Object Gateway nodes to the cluster. Apply the steps detailed in Tuning the Kernel to each node. See Adding and Removing OSD Nodes for details on adding nodes.

For each OSD node added to the cluster, add OSDs to the cluster for each drive in the node that will store client data. See Adding an OSD for additional details. When using Ansible to add OSD nodes, refer to Configuring Ansible Groups, and add the OSD nodes to the appropriate group if the cluster will support multiple use cases.

For each Ceph Object Gateway node, install a gateway instance. See Ceph Object Gateway Installation for details.

Once the cluster returns to an **`active+clean`** state, remove any overrides and proceed with Developing Storage Strategies.

NOTE

Step 3 of **Adding a Node** and Step 10 of **Adding an OSD With the Command Line Interface** will be revisited in topics beginning with Developing CRUSH Hierarchies.

CHAPTER 5. DEVELOPING STORAGE STRATEGIES

One of the more challenging aspects of setting up Ceph Storage clusters and Ceph Object Gateways for production use is defining effective storage strategies. Storage strategies involve the following factors:

- Developing CRUSH Hierarchies
- Creating CRUSH Roots
- Using Logical Host Names in a CRUSH Map
- Creating CRUSH Rulesets
- Creating the Gateway's Root Pool
- Configuring a Realm
- Creating System Pools
- Creating Data Placement Strategies

See the Storage Strategies guide for general guidance on storage strategies and command line usage.

5.1. DEVELOPING CRUSH HIERARCHIES

When deploying a Ceph cluster and an Object Gateway, typically the object gateway will have a default zone group and zone. The Ceph storage cluster will have default pools, which in turn will use a CRUSH map with a default CRUSH hierarchy and a default CRUSH ruleset.

IMPORTANT

The default `rbd` pool may use the default CRUSH ruleset. **DO NOT** delete the default ruleset or hierarchy if Ceph clients have used them to store client data.

For general details on CRUSH hierarchies, see the CRUSH Administration section of the Storage Strategies guide.

Production gateways typically use a custom realm, zone group and zone named according to the use and geographic location of the gateways. Additionally, the Ceph cluster will have a CRUSH map that has multiple CRUSH hierarchies.

- **System Pools:** At least one CRUSH hierarchy will be for system pools and potentially for data. The system pools include `.rgw.root` and the system pools associated with the zone. System pools typically fall under a single CRUSH hierarchy, and use replication for data durability. A data pool may also use the CRUSH hierarchy, but the pool will usually be configured with erasure coding for data durability.
- **Bucket Index:** At least one CRUSH hierarchy **SHOULD** be for the bucket index pool, where the CRUSH hierarchy maps to SSD drives. Bucket indices can be a performance bottleneck. It is strongly recommended to use SSD drives in this CRUSH hierarchy. **DO NOT** create partitions for bucket indices on SSDs used for OSD journals . Additionally, a bucket index should be configured with bucket sharding. See Creating a Bucket Index Pool and supporting links for details.
- **Placement Pools:** The placement pools for each placment target include the bucket index, the

data bucket and the bucket extras. These pools may fall under separate CRUSH hierarchies. Since Ceph Object Gateway can support multiple storage policies, the bucket pools of the storage policies may be associated with different CRUSH hierarchies, reflecting different use cases such as IOPS-optimized, throughput-optimized, and capacity-optimized respectively. The bucket index pool **SHOULD** use its own CRUSH hierarchy to map the bucket index pool to higher performance SSD drives.

5.1.1. Creating CRUSH Roots

From the command line on the administration node, create CRUSH roots in the CRUSH map for each CRUSH hierarchy. There **MUST** be at least one CRUSH hierarchy for system pools that may also potentially serve data storage pools. There **SHOULD** be at least one CRUSH hierarchy for the bucket index pool, mapped to SSDs or other high speed data storage media.

For details on CRUSH hierarchies, see CRUSH Hierarchies. To manually edit a CRUSH map, see Editing a CRUSH Map.

In the following examples, the hosts named **`data0`**, **`data1`** and **`data2`** use extended logical names such as **`data0-sas-ssd`**, **`data0-index`** and so forth in the CRUSH map, because there are multiple CRUSH hierarchies pointing to the same physical hosts.

A typical CRUSH root might represent nodes with SAS drives and SSDs for journals. For example:

```
##
# SAS-SSD ROOT DECLARATION
##

root sas-ssd {
  id -1   # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
  item data2-sas-ssd weight 4.000
  item data1-sas-ssd weight 4.000
  item data0-sas-ssd weight 4.000
}
```

A CRUSH root for bucket indexes **SHOULD** represent SSD drives or other high speed storage media used exclusively for bucket indexes. For example:

```
##
# INDEX ROOT DECLARATION
##

root index {
  id -2    # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
  item data2-index weight 1.000
  item data1-index weight 1.000
  item data0-index weight 1.000
}
```

NOTE

Create a separate CRUSH hierarchy with different SSD drives to store data in SSDs. **DO NOT** use the same SSD drive to store journals, bucket indexes and data.

5.1.2. Using Logical Host Names in a CRUSH Map

In the CRUSH map, host names must be unique and used only once. When the host serves multiple CRUSH hierarchies and use cases, a CRUSH map may use logical host names instead of the actual host name in order to ensure the host name is only used once. For example, a node may have multiple types of drives such as SSDs, SAS drives with SSD journals and SATA drives with co-located journals. To create multiple CRUSH hierarchies for the same host, the hierarchies will need to use logical host names in lieu of the actual host names so the bucket names are unique within the CRUSH hierarchy. For example, if the host name is **`data2`**, the CRUSH hierarchy might use logical names such as **`data2-sas-ssd`** and **`data2-index`**. For example:

```
host data2-sas-ssd {
  id -11   # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 1.000
  item osd.1 weight 1.000
  item osd.2 weight 1.000
  item osd.3 weight 1.000
}
```

In the foregoing example, the host **`data2`** uses the logical name **`data2-sas-ssd`** to map the SAS drives with journals on SSDs into one hierarchy. The OSD IDs **`osd.0`** through **`osd.3`** in the forgoing example represent SAS drives using SSD journals in a high throughput hardware configuration. These OSD IDs differ from the OSD ID in the following example.

In the following example, the host **`data2`** uses the logical name **`data2-index`** to map the SSD drive for a bucket index into a second hierarchy. The OSD ID **`osd.4`** in the following example represents an SSD drive or other high speed storage media used exclusively for a bucket index pool.

```
host data2-index {
  id -21   # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
  item osd.4 weight 1.000
}
```

IMPORTANT

When using logical host names, ensure that one of the following settings is present in the Ceph configuration file to prevent the OSD startup scripts from using the actual host names upon startup and thereby failing to locate data in CRUSH map.

When the CRUSH map uses logical host names, as in the foregoing examples, prevent the OSD startup scripts from identifying the hosts according to their actual host names at initialization. In the **`[global]`** section of the Ceph configuration file, add the following setting:

```
osd_crush_update_on_start = false
```

An alternative method of defining a logical host name is to define the location of the CRUSH map for each OSD in the **`[osd.<ID>]`** sections of the Ceph configuration file. This will override any locations the OSD startup script defines. From the foregoing examples, the entries might look like the following:

```
[osd.0]
osd crush location = "host=data2-sas-ssd"

[osd.1]
osd crush location = "host=data2-sas-ssd"

[osd.2]
osd crush location = "host=data2-sas-ssd"

[osd.3]
osd crush location = "host=data2-sas-ssd"

[osd.4]
osd crush location = "host=data2-index"
```

IMPORTANT

If one of the foregoing approaches isn't used when a CRUSH map uses logical host names rather than actual host names, on restart, the Ceph Storage Cluster will assume that the OSDs map to the actual host names, and the actual host names will not be found in the CRUSH map, and Ceph Storage Cluster clients will not find the OSDs and their data.

5.1.3. Creating CRUSH Rulesets

Like the default CRUSH hierarchy, the CRUSH map also contains a default CRUSH ruleset, usually **`ruleset 0`**.

NOTE

The default **`rbd`** pool may use this ruleset. DO NOT delete the default ruleset if other pools have used it to store customer data.

See CRUSH Rules for general details on CRUSH Rules. To manually edit a CRUSH map, see Editing a CRUSH Map.

For each CRUSH hierarchy, create a CRUSH rule. The following example illustrates a rule for the CRUSH hierarchy that will store the system pools, including **`.rgw.root`**. In this example, the root **`sas-ssd`** serves as the main CRUSH hierarchy. It uses **`ruleset 1`** to distinguish itself from the default **`ruleset 0`**. The **`step take sas-ssd`** line tells the pool to use the **`sas-ssd`** root created in CRUSH Root, whose child buckets contain OSDs with SAS drives and SSD journals in a high throughput hardware configuration. The **`type rack`** portion of **`step chooseleaf`** is the failure domain. In the following example, it is a rack.

```
##
# SYSTEM RULE DECLARATION
##
```

```
rule rgw-system {
 ruleset 1
 type replicated
 min_size 1
 max_size 10
 step take sas-ssd
 step chooseleaf firstn 0 type rack
 step emit
}
```

NOTE

In the foregoing example, if data gets replicated three times, there should be at least three racks in the cluster containing a similar number of OSD nodes.

TIP

The **`type replicated`** setting has **NOTHING** to do with data durability, the number of replicas or the erasure coding. Only **`replicated`** is supported.

The following example illustrates a rule for the CRUSH hierarchy that will store the data pool. In this example, the root **`sas-ssd`** serves as the main CRUSH hierarchy—the same CRUSH hierarchy as the system rule. It uses **`ruleset 2`** to distinguish itself from the default **`ruleset 0`** and **`ruleset 1`**. The **`step take sas-ssd`** line tells the pool to use the **`sas-ssd`** root created in CRUSH Root, whose child buckets contain OSDs with SAS drives and SSD journals in a high throughput hardware configuration. The **`type host`** portion of **`step chooseleaf`** is the failure domain. In the following example, it is a host. Notice that the rule uses the same CRUSH hierarchy, but a different failure domain.

```
##
# THROUGHPUT RULE DECLARATION
##

rule rgw-throughput {
 ruleset 2
 type replicated
 min_size 1
 max_size 10
 step take sas-ssd
 step chooseleaf firstn 0 type host
 step emit
}
```

NOTE

In the foregoing example, if the pool uses erasure coding with a a larger number of data and encoding chunks than the default, there should be at least as many racks in the cluster containing a similar number of OSD nodes to facilitate the erasure coding chunks. For smaller clusters, this may not be practical, so the foregoing example uses **`host`** as the CRUSH failure domain.

The following example illustrates a rule for the CRUSH hierarchy that will store the index pool. In this example, the root **`index`** serves as the main CRUSH hierarchy. It uses **`ruleset 3`** to distinguish itself from **ruleset 0** through **ruleset 2**. The **step take index** line tells the pool to use the **index** root created in CRUSH Root, whose child buckets contain OSDs SSD drives for high IOPS. The **type rack** portion of **step chooseleaf** is the failure domain. In the following example, it is a rack.

```
##
# INDEX RULE DECLARATION
##

rule rgw-index {
 ruleset 3
 type replicated
 min_size 1
 max_size 10
 step take index
 step chooseleaf firstn 0 type rack
 step emit
}
```

5.2. CREATING THE ROOT POOL

The Ceph Object Gateway configuration gets stored in a pool named **.rgw.root**, including realms, zone groups and zones. By convention, its name is not prepended with the zone name.

.rgw.root

If the Ceph Storage Cluster is running, create an **.rgw.root** pool using the new ruleset. See the Ceph Placement Groups (PGs) per Pool Calculator and Placement Groups for details on the number or PGs. See Create a Pool for details on creating a pool. In this instance, the pool will use **replicated** and **NOT erasure** for data durability. For example:

```
# ceph osd pool create .rgw.root 32 32 replicated sas-ssd
```

NOTE

For system pools, including **.rgw.root**, the suggested PG count from the Ceph Placement Groups (PGs) per Pool Calculator is substantially less than the target PGs per OSD. Also, ensure the number of OSDs is set in step 3 of the calculator.

Once this pool gets created, the Ceph Object Gateway can store its configuration data in the pool.

5.3. CONFIGURING A REALM

The Ceph Storage pools supporting the Ceph Object Gateway apply to a zone within a zone group. By default, Ceph Object Gateway will define a default zone group and zone.

For the master zone group and zone, Red Hat recommends creating a new realm, zone group and zone. Then, delete the default zone and its pools if they were already generated. Use Configuring a Master Zone as a best practice, because this configures the cluster for Multi Site operation.

1. Create a realm. See Realms for additional details.

2. Create a master zone group. See Zone Groups for additional details on zone groups.

3. Create a master zone. See Zones for additional details on zones.

4. Delete the default zone group and zone. You **MAY** delete default pools if they were created, and are not storing client data. **DO NOT** delete the `.rgw.root` pool.

5. Create a System User.

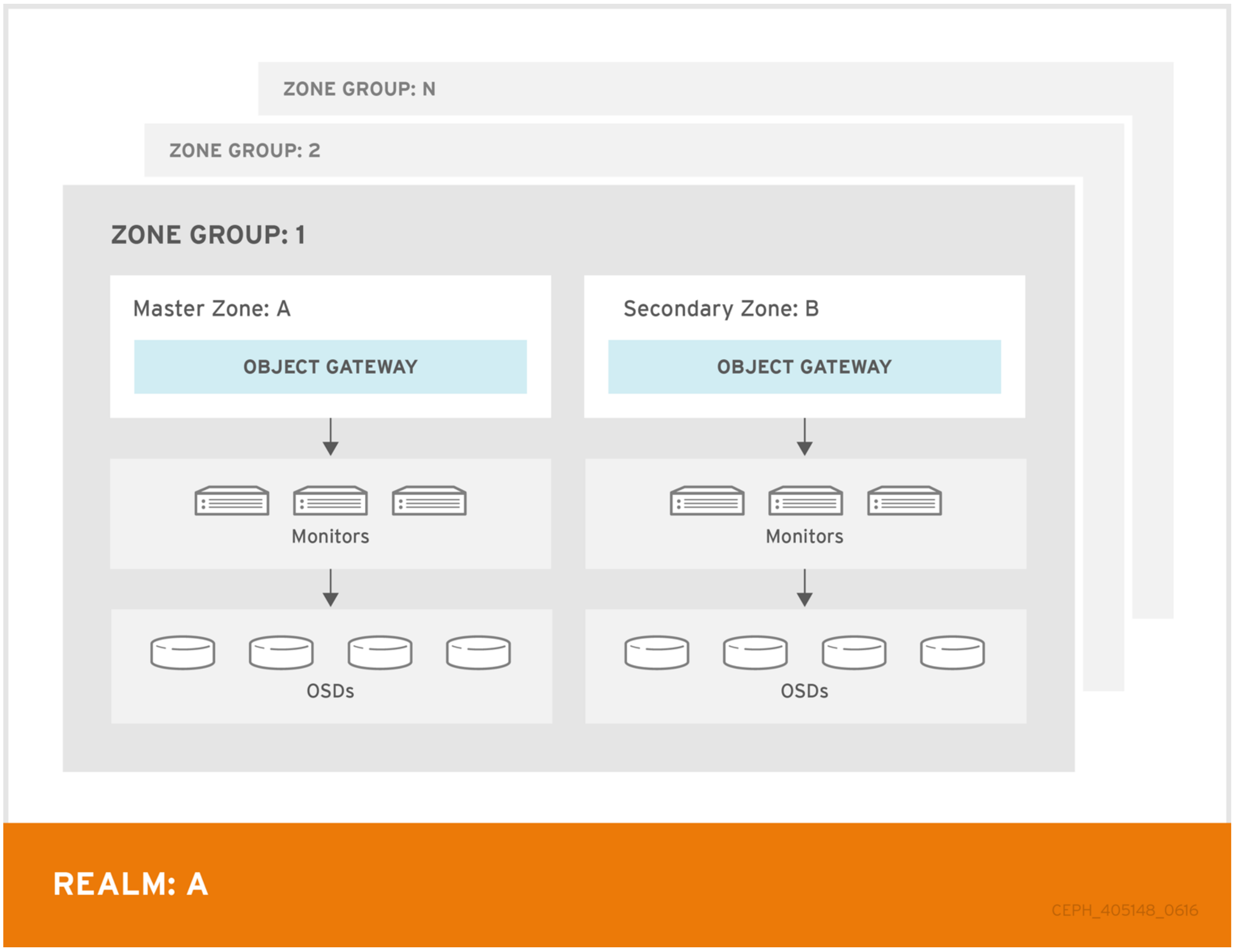
6. Update the period.

7. Update the Ceph Configuration file.

NOTE

This procedure omits the step of starting the gateway, since the gateway may create the pools manually. To specify specific CRUSH rulesets and data durability methods, create the pools manually.

By setting up a new realm, zone group and zone, the cluster is now prepared for expansion to a multi site cluster where there are multiple zones within the zone group. This means that the cluster can be expanded and configured for failover, and disaster recovery. See Expanding the Cluster with Multi Site for additional details.



In Red Hat Ceph Storage 2, multi site configurations are active-active by default. When deploying a multi site cluster, the zones and their underlying Ceph storage clusters may be in different geographic regions. Since each zone has a deep copy of each object in the same namespace, users can access the copy from the zone that is physically the closest to them, reducing latency. However, the cluster may be configured in active-passive mode if the secondary zones are intended only for failover and disaster recovery.

NOTE

Using a zone group with multiple zones is supported. Using multiple zone groups is a technology preview only, and is not supported in production.

5.4. CREATING SYSTEM POOLS

The Ceph Object Gateway uses many pools for various system functions, and a separate set of placement pools for storing bucket indexes, data and other information.

Since it is computationally expensive to peer a pool's placement groups, Red Hat generally recommends that the Ceph Object Gateway's system pools use substantially fewer placement groups than data storage pools.

The system pools store objects related to system control, garbage collection, logging, user information, usage, etc. By convention, these pool names have the zone name prepended to the pool name.

- **`.<zone-name>.rgw.control`**: The control pool.
- **`.<zone-name>.rgw.gc`**: The garbage collection pool, which contains hash buckets of objects to be deleted.
- **`.<zone-name>.log`**: The log pool contains logs of all bucket/container and object actions such as create, read, update and delete.
- **`.<zone-name>.intent-log`**: The intent log pool contains a copy of an object update request to facilitate undo/redo if a request fails.
- **`.<zone-name>.users.uid`**: The user ID pool contains a map of unique user IDs.
- **`.<zone-name>.users.keys`**: The keys pool contains access keys and secret keys for each user ID.
- **`.<zone-name>.users.email`**: The email pool contains email addresses associated to a user ID.
- **`.<zone-name>.users.swift`**: The Swift pool contains the Swift subuser information for a user ID.
- **`.<zone-name>.usage`**: The usage pool contains a usage log on a per user basis.

Execute the Get a Zone procedure to see the pool names.

```
# radosgw-admin zone get [--rgw-zone=<zone>]
```

When **`radosgw-admin`** creates a zone, the pool names **SHOULD** be prepended with the zone name. For example, a zone named **`us-west`** **SHOULD** have pool names that look something like this:

```
{ "domain_root": ".rgw.root",
  "control_pool": ".us-west.rgw.control",
  "gc_pool": ".us-west.rgw.gc",
    "log_pool": ".us-west.log",
    "intent_log_pool": ".us-west.intent-log",
    "usage_log_pool": ".us-west.usage",
    "user_keys_pool": ".us-west.users.keys",
    "user_email_pool": ".us-west.users.email",
    "user_swift_pool": ".us-west.users.swift",
    "user_uid_pool": ".us-west.users.uid",
    "system_key": { "access_key": "", "secret_key": ""},
    "placement_pools": [
      {  "key": "default-placement",
         "val": { "index_pool": ".us-west.rgw.buckets.index",
                  "data_pool": ".us-west.rgw.buckets",
                  "data_extra_pool": ".us-west.rgw.buckets.non-ec"
                  "index_type": 0
                }
      }
    ]
  }
```

Beginning with **`control_pool`** and ending with **`user_uid_pool`**, create the pools using the pool names in the zone, provided the zone name is prepended to the pool name. Following the previous examples, pool creation might look something like this:

```
# ceph osd pool create .us-west.rgw.control 32 32 replicated rgw-system
...
# ceph osd pool create .us-west.users.uid 32 32 replicated rgw-system
```

From previous examples, the **`rgw-system`** ruleset represents a CRUSH hierarchy of SAS drives with SSD journals and **`rack`** as the CRUSH failure domain. See Creating CRUSH Roots, and Creating CRUSH Rulesets for preceding examples.

See the Ceph Placement Groups (PGs) per Pool Calculator and Placement Groups for details on the number of PGs. See Create a Pool for details on creating a pool.

NOTE

For system pools the suggested PG count from the calculator is substantially less than the target PGs per OSD. Ensure that step 3 of the calculator specifies the correct number of OSDs.

Generally, the **`.rgw.root`** pool and the system pools should use the same CRUSH hierarchy and use at least **node** as the failure domain in the CRUSH ruleset. Like the **`.rgw.root`** pool, the system pools should use **`replicated`** for data durability, **NOT** **`erasure`**.

5.5. CREATING DATA PLACEMENT STRATEGIES

The Ceph Object Gateway has a default storage policy called **`default-placement`**. If the cluster has only one storage policy, the **`default-placement`** policy will suffice. This default placement policy is referenced from the zone group configuration and defined in the zone configuration.

See Storage Policies for additional details.

For clusters that support multiple use cases, such as IOPS-optimized, throughput-optimized or capacity-optimized, a set of placement targets in the zone group configuration and a set of placement pools in the zone configuration represent each storage policy.

The examples in the following sections illustrate how to create a storage policy and make it the default policy. This example also assumes the default policy will use a throughput-optimized hardware profile. Topics include:

- Creating a Bucket Index Pool
- Creating a Data Pool
- Creating a Bucket Extras Pool
- Configuring Placement Targets in a Zone Group
- Configuring Placement Pools in a Zone
- Data Placement Summary

5.5.1. Creating a Bucket Index Pool

By default, Ceph Object Gateway maps a bucket's objects to a bucket index, which enables a gateway client to request a list of objects in a bucket among other things. While common use cases may involve quotas where users have a bucket and a limited number of objects per bucket, buckets can store innumerable objects. When buckets store millions of objects, bucket index performance benefits substantially from using SSDs or other high performance storage media to store its data. Additionally, bucket sharding also dramatically improves performance.

See the Ceph Placement Groups (PGs) per Pool Calculator and Placement Groups for details on the number of PGs. See Create a Pool for details on creating a pool.

NOTE

The PG per Pool Calculator recommends a smaller number of PGs per pool for the bucket index pool; however, the PG count is approximately twice the number of PGs as the system pools.

To create a bucket index pool, execute **`ceph osd pool create`** with the pool name, the number of PGs and PGPs, the **`replicated`** data durability method, and the name of the ruleset. For example:

```
# ceph osd pool create .us-west.rgw.buckets.index 64 64 replicated rgw-
index
```

From previous examples, the **`rgw-index`** ruleset represents a CRUSH hierarchy of SSD drives and **`rack`** as the CRUSH failure domain. See Selecting SSDs for Bucket Indexes, Creating CRUSH Roots, and Creating CRUSH Rulesets for preceding examples.

IMPORTANT

If buckets will store more than 100k objects, configure bucket sharding to ensure that bucket index performance doesn't degrade as the number of objects in the bucket increases. See Configuring Bucket Sharding. Also see Bucket Index Resharding for details on resharding a bucket if the original configuration is no longer suitable.

5.5.2. Creating a Data Pool

The data pool is where Ceph Object Gateway stores the object data for a particular storage policy. The data pool should have a full complement of PGs, not the reduced number of PGs for system pools. The data pool **SHOULD** consider using erasure coding, as it is substantially more efficient than replication and can significantly reduce the capacity requirements while maintaining data durability.

To use erasure coding, create an erasure code profile. See Erasure Code Profiles in the Storage Strategies Guide for more details.

IMPORTANT

Choosing the correct profile is important because you cannot change the profile after you create the pool. To modify a profile, you must create a new pool with a different profile and migrate the objects from the old pool to the new pool.

The default configuration is two data chunks and one encoding chunk, which means only one OSD can be lost. For higher resiliency, consider a larger number of data and encoding chunks. For example, some large very scale systems use 8 data chunks and 3 encoding chunks, which allows three OSDs to fail without losing data.

IMPORTANT

Each data and encoding chunk SHOULD get stored on a different node or host at a minimum. For smaller clusters, this makes using **`rack`** impractical as the minimum CRUSH failure domain when using a larger number of data and encoding chunks. Consequently, it is common for the data pool to use a separate CRUSH hierarchy with **`host`** as the minimum CRUSH failure domain. Red Hat recommends **`host`** as the minimum failure domain. If erasure code chunks get stored on OSDs within the same host, a host failure such as a failed journal or network card could lead to data loss.

To create a data pool, execute **`ceph osd pool create`** with the pool name, the number of PGs and PGPs, the **`erasure`** data durability method, the erasure code profile and the name of the ruleset. For example:

```
# ceph osd pool create .us-west.rgw.buckets.throughput 8192 8192 erasure
8k3m rgw-throughput
```

From previous examples, the **`rgw-throughput`** ruleset represents a CRUSH hierarchy of SAS drives with SSD journals and **`host`** as the CRUSH failure domain. See Creating CRUSH Roots, and Creating CRUSH Rulesets for preceding examples.

5.5.3. Creating a Bucket Extras Pool

The **`data_extra_pool`** is for data that cannot use erasure coding. For example, multi-part uploads allow uploading a large object such as a movie in multiple parts. These parts must first be stored without erasure coding. Erasure coding will apply to the whole object, not the partial uploads.

NOTE

The PG per Pool Calculator recommends a smaller number of PGs per pool for the **`data_extra_pool`**; however, the PG count is approximately twice the number of PGs as the system pools and the same as the bucket index pool.

To create a data extra pool, execute **`ceph osd pool create`** with the pool name, the number of PGs and PGPs, the **`replicated`** data durability method, and the name of the ruleset. For example:

```
# ceph osd pool create .us-west.rgw.buckets.non-ec 64 64 replicated rgw-
system
```

5.5.4. Configuring Placement Targets in a Zone Group

Once the pools are created, create the placement target in the zone group. To retrieve the zone group, execute the following to output the zone group configuration to a file called **`zonegroup.json`**:

```
# radosgw-admin zonegroup get [--rgw-zonegroup=<zonegroup>] >
zonegroup.json
```

The file contents will look something like this:

```
{
  "id": "90b28698-e7c3-462c-a42d-4aa780d24eda",
  "name": "us",
  "api_name": "us",
  "is_master": "true",
  "endpoints": [
    "http:\/\/rgw1:80"
  ],
  "hostnames": [],
  "hostnames_s3website": [],
  "master_zone": "9248cab2-afe7-43d8-a661-a40bf316665e",
  "zones": [
      {
        "id": "9248cab2-afe7-43d8-a661-a40bf316665e",
        "name": "us-east",
        "endpoints": [
            "http:\/\/rgw1"
        ],
        "log_meta": "true",
        "log_data": "true",
        "bucket_index_max_shards": 0,
        "read_only": "false"
      },
      {
        "id": "d1024e59-7d28-49d1-8222-af101965a939",
        "name": "us-west",
        "endpoints": [
            "http:\/\/rgw2:80"
        ],
        "log_meta": "false",
        "log_data": "true",
        "bucket_index_max_shards": 0,
        "read_only": "false"
      }
  ],
  "placement_targets": [
      {
        "name": "default-placement",
            "tags": []
        }
    ],
    "default_placement": "default-placement",
    "realm_id": "ae031368-8715-4e27-9a99-0c9468852cfe"
}
```

The **placement_targets** section will list each storage policy. By default, it will contain a placement target called **default-placement**. The default placement target is identified immediately after the **placement_targets** section.

Assuming a placement target called **throughput-optimized**, with **throughput-optimized** as the default target, the **placement_targets** section and the **default_placement** setting of the zone group configuration should be modified to something like this:

```
{
  ...
  "placement_targets": [
      {
        "name": "throughput-optimized",
        "tags": []
      }
  ],
  "default_placement": "throughput-optimized",
  ...
}
```

Finally, set the zone group configuration with the settings from the modified **zonegroup.json** file; then, update the period. For example:

```
# radosgw-admin zonegroup set [--rgw-zonegroup=<zonegroup>] --infile
zonegroup.json
# radosgw-admin period update --commit
```

5.5.5. Configuring Placement Pools in a Zone

Once the zone group has the new **throughput-optimized** placement target, map the placement pools for **throughput-optimized** in the zone configuration. This step will replace the mapping for **default-placement** to its associated pools with a **throughput-optimized** set of placement pools.

Execute the Get a Zone procedure to see the pool names.

```
# radosgw-admin zone get [--rgw-zone=<zone>] > zone.json
```

Assuming a zone named **us-west**, the file contents will look something like this:

```
{ "domain_root": ".rgw.root",
  "control_pool": ".us-west.rgw.control",
  "gc_pool": ".us-west.rgw.gc",
  "log_pool": ".us-west.log",
  "intent_log_pool": ".us-west.intent-log",
  "usage_log_pool": ".us-west.usage",
  "user_keys_pool": ".us-west.users.keys",
  "user_email_pool": ".us-west.users.email",
  "user_swift_pool": ".us-west.users.swift",
  "user_uid_pool": ".us-west.users.uid",
  "system_key": { "access_key": "", "secret_key": ""},
  "placement_pools": [
    {  "key": "default-placement",
       "val": { "index_pool": ".us-west.rgw.buckets.index",
                "data_pool": ".us-west.rgw.buckets",
                "data_extra_pool": ".us-west.rgw.buckets.non-ec"
                "index_type": 0
              }
    }
  ]
}
```

The **`placement_pools`** section of the zone configuration defines sets of placement pools. Each set of placement pools defines a storage policy. Modify the file to remove the **`default-placement`** entry, and replace it with a **`throughput-optimized`** entry with the pools created in the preceding steps. For example:

```
{
...
"placement_pools": [
    {  "key": "throughput-optimized",
       "val": { "index_pool": ".us-west.rgw.buckets.index",
                "data_pool": ".us-west.rgw.buckets.throughput"}
                "data_extra_pool": ".us-west.rgw.buckets.non-ec",
                "index_type": 0
    }
  ]
}
```

Finally, set the zone configuration with the settings from the modified **`zone.json`** file; then, update the period. For example:

```
# radosgw-admin zone set --rgw-zone={zone-name} --infile zone.json
# radosgw-admin period update --commit
```

NOTE

The **`index_pool`** points to the index pool and CRUSH hierarchy with SSDs or other high-performance storage, the **`data_pool`** points to a pool with a full complement of PGs, and a CRUSH hierarchy of high-throughput host bus adapters, SAS drives and SSDs for journals.

5.5.6. Data Placement Summary

When processing client requests, the Ceph Object Gateway will use the the new **`throughput-optimized`** target as the default storage policy. Use this procedure to establish the same target in different zones and zone groups in a multi-site configuration, replacing the zone name for the pools as appropriate.

Use this procedure to establish additional storage policies. The naming for each target and set of placement pools is arbitrary. It could be **`fast`**, **`streaming`**, **`cold-storage`** or any other suitable name. However, each set must have a corresponding entry under **`placement_targets`** in the zone group, and one of the targets **MUST** be referenced in the **`default_placement`** setting; and, the zone must have a corresponding set of pools configured for each policy.

Client requests will always use the default target, unless the client request specifies **`X-Storage-Policy`** and a different target. See Create a Container for an object gateway client usage example.

CHAPTER 6. CONFIGURING GATEWAYS

The final steps in preparing the Ceph Object Gateway for production involve configuring Civetweb, firewall ports, the DNS and load balancers. Topics include:

- Configuring Civetweb
- Configuring Firewall Ports
- Configuring DNS Wildcards
- Configuring Load Balancers

6.1. CONFIGURING CIVETWEB

Depending on the choices made during installation of the Ceph Object Gateway, the Ceph configuration file will already have entries for each instance of the Ceph Object Gateway with additional modifications from the steps involved in Configuring a Realm.

The most common configuration change from the default configuration is changing the default port **`7480`** to another port such as **`8080`** or **`80`**. See Changing the Default Port.

There are additional settings particular to Civetweb. See Civetweb Configuration Options for details.

There are additional settings which may be overridden. See Object Gateway Configuration Reference for details.

NOTE

The section on Additional Use Cases will provide detailed configuration examples for using Ceph Object Gateway with third party components.

6.2. CONFIGURING FIREWALL PORTS

When changing the default port for Civetweb, ensure that the corresponding ports are open for client access. See Configuring the Firewall for details.

6.3. CONFIGURING DNS WILDCARDS

S3-style subdomains incorporate the bucket name as a CNAME extension. Add a wildcard to the DNS to facilitate S3-style subdomains. See Adding a Wildcard to DNS for details.

6.4. CONFIGURING LOAD BALANCERS

A zone will typically have multiple instances of a Ceph Object Gateway to handle production loads and to maintain high availability. Production clusters typically use a load balancer to allocate requests among gateway instances.

Additionally, earlier versions of Civetweb do not support HTTPS. A load balancer can be configured to accept SSL requests, terminate the SSL connection and pass the request over HTTP to the gateway instances.

Ceph Storage aims to maintain high availability. For this reason, Red Hat recommends using HAProxy/keepalived. See HAProxy/keepalived Configuration for details.

CHAPTER 7. ADDITIONAL USE CASES

Once the cluster is up and running, there are additional use cases to consider.

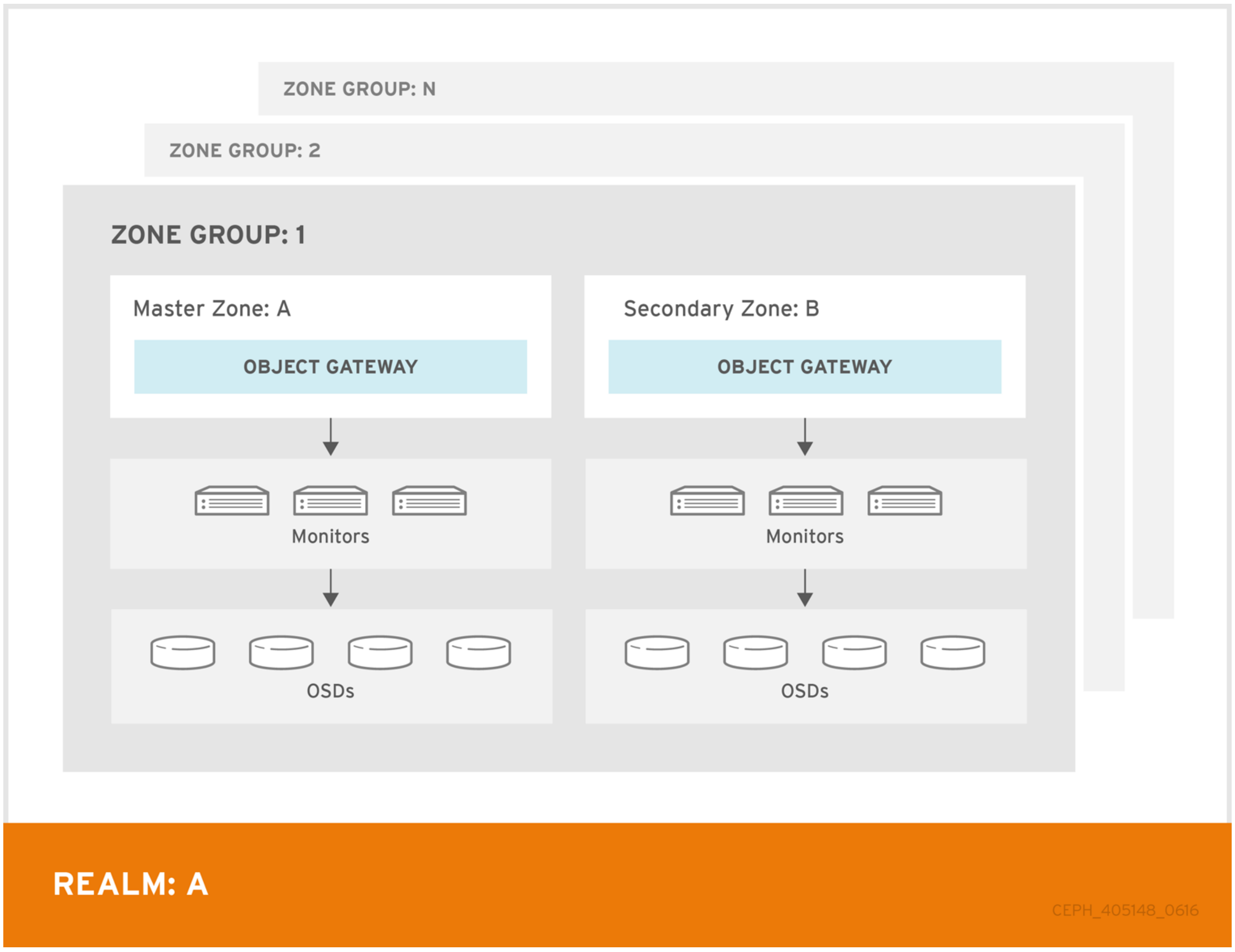
- Expanding the Cluster with Multi-Site
- Migrating Data with NFS-Ganesha
- Configuring the Cluster for Static Webhosting
- Configuring the Cluster with LDAP/AD
- Configuring the Cluster to use Keystone

7.1. EXPANDING THE CLUSTER WITH MULTI-SITE

When developing storage strategies, the procedure for Creating a Realm ensured that the cluster is already configured to use multi-site with its own realm, master zone group and master zone.

A typical production cluster will have a secondary zone with its own Ceph Storage Cluster in a separate physical location to act as a backup in the event of a disaster. To set up a secondary zone, repeat the procedures in this guide. Generally, the secondary zone should have the same hardware configuration and sizing as the master zone. See Configuring a Secondary Zone for additional details.

Adding a secondary zone adds Failover and Disaster Recovery capabilities to the cluster.



7.2. MIGRATING DATA WITH NFS GANESHA

If the Ceph Object Gateway and Ceph Storage Cluster replaces a filesystem-based storage solution, consider using Ceph's NFS-Ganesha solution to migrate data from the file system into Ceph Object Gateway.

See Exporting the Namespace to NFS-Ganesha (TECH PREVIEW).

7.3. CONFIGURING THE CLUSTER FOR STATIC WEBHOSTING

Traditional web hosting sometimes involves setting up a web server for each website, which can use resources inefficiently when content doesn't change dynamically.

Ceph Object Gateway can host static web sites in S3 buckets—that is, sites that do not use server-side services like PHP, servlets, databases, nodejs and the like. This approach is substantially more economical than setting up VMs with web servers for each site.

See Configuring Gateways for Static Web Hosting for additional details.

7.4. CONFIGURING THE CLUSTER FOR LDAP/AD

Organizations deploying Ceph Object Gateway for their users and applications may choose to use Light-weight Directory Access Protocol (LDAP) or Microsoft Active Directory (AD) to authenticate with the Ceph Object Gateway in lieu of creating Ceph Object Gateway users.

Using LDAP/AD means that Ceph Object Gateway can integrate with an organizations LDAP/AD single sign-on initiatives.

See the Ceph Object Gateway with LDAP/AD Guide for details.

7.5. CONFIGURING THE CLUSTER TO USE OPENSTACK KEYSTONE

When deploying the Ceph Object Gateway in lieu of OpenStack Swift, it is possible to configure the gateway to use OpenStack Keystone to authenticate users in lieu of creating Ceph Object Gateway users.

See Using Keystone to Authenticate Ceph Object Gateway Users for details.