Red Hat Ceph Storage Planning

Planning involves considering the supported compatibility, physical configuration, and various storage strategy prerequisites before working with Red Hat Ceph Storage.

Considerations and recommendations

Use this information to understand all considerations and recommendations before you run your Red Hat Ceph Storage cluster.

Red Hat Ceph Storage can be used for different workloads based on a particular business need or set of requirements. Doing the necessary planning before installing a Red Hat Ceph Storage is critical to the success of running a Ceph storage cluster efficiently and achieving the business requirements.

For the latest architectural requirements for a supportable cluster, see Red Hat Ceph Storage: Supported configurations.

> **Note:** For more Red Hat Ceph Storage planning help, contact your Red Hat Support account team.

Basic considerations

Use these basic considerations for running your Red Hat Ceph Storage cluster.

Before using Red Hat Ceph Storage, it is important to develop a proper storage strategy for your data. A storage strategy is a method of storing data that serves a particular use case. If storing volumes and images for a cloud platform, such as OpenStack, is necessary, you can store data on faster Serial Attached SCSI (SAS) drives with Solid State Drives (SSD) for journals. By contrast, if you need to store object data for an S3- or Swift-compliant gateway, you can choose to use something more economical, like traditional Serial Advanced Technology Attachment (SATA) drives.

Both cloud platform and S3- and Swift-compliant gateway scenarios are supported within the same Ceph storage cluster. When a storage cluster needs to support both scenarios, a fast storage strategy to the cloud platform and traditional storage for the object store must be provided.

One of the most important steps in a successful Ceph deployment is identifying a price-to-performance profile suitable for the storage cluster's use case and workload. It is important to choose the right hardware for the use case. For example:

- An IOPS-optimized hardware for a cold storage application can increase hardware costs unnecessarily.
- Capacity-optimitized hardware might have a more attractive price point but can lead to slow performance with IOPS-intensive workloads.

For more hardware considerations, see "Minimum hardware considerations" on page 15.

Use cases

Red Hat Ceph Storage use cases can support multiple storage strategies. Use cases, cost versus benefit performance tradeoffs, and data durability are the primary considerations that help develop a sound storage strategy.

Ceph provides massive storage capacity, and it supports numerous use cases, such as:

- The Ceph Block Device client is a leading storage backend for cloud platforms that provides limitless storage for volumes and images with high-performance features like copy-on-write cloning.
- The Ceph Object Gateway client is a leading storage backend for cloud platforms that provides a RESTful S3-compliant and Swift-compliant object storage for objects like audio, bitmap, video, and other data.
- The Ceph File System (CephFS) for traditional file storage.

Cost versus benefit of performance

Faster is better. Bigger is better. High durability is better. However, there is a price for each superlative quality, and a corresponding cost versus benefit tradeoff. Consider the following use cases from a performance perspective:

- SSDs can provide fast storage for relatively small amounts of data and journaling. Storing a database or object index can benefit from a pool of very fast SSDs, but proves too expensive for other data.
- SAS drives with SSD journaling provide fast performance at an economical price for volumes and images.
- SATA drives without SSD journaling provide cheap storage with lower overall performance.

When you create a CRUSH hierarchy of OSDs, you need to consider the use case and an acceptable cost versus performance tradeoff.

Data durability

In large-scale storage clusters, hardware failure is an expectation, not an exception. However, data loss and service interruption remain unacceptable. For this reason, data durability is important. Ceph addresses data durability with multiple replica copies of an object or with erasure coding and multiple coding chunks. Multiple copies or multiple coding chunks present an extra cost versus benefit tradeoff: it is cheaper to store fewer copies or coding chunks, but it can lead to the inability to service write requests in a degraded state. Generally, one object with two more copies, or two coding chunks can allow a storage cluster to service writes in a degraded state while the storage cluster recovers.

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.

Erasure coding stores data as data chunks and coding chunks. If there is 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, by using erasure coding with eight data chunks and three coding chunks provides the same redundancy as three copies of the data. However, such an encoding scheme uses approximately 1.5-times the initial data stored compared to three-times with replication.

The CRUSH algorithm aids this process by ensuring that Ceph stores extra copies or coding chunks in different locations within the storage cluster. This ensures that the failure of a single storage device or host does not lead to losing all copies or coding chunks necessary to preclude data loss. You can plan a storage strategy with cost versus benefit tradeoffs, and data durability in mind, then present it to a Ceph client as a storage pool.

Important:

- ONLY the data storage pool can use erasure coding. Pools storing service data and bucket indexes use replication.
- Erasure coding is supported only for Ceph Object Gateway bucket data pools and CephFS data pools. Index and metadata pools must use replication. Erasure-coded Ceph Block Device pools are supported only for archival data.
- Ceph's object copies or coding chunks make RAID solutions obsolete. Do not use RAID, as Ceph already handles data durability. A degraded RAID has a negative impact on performance and recovering data that uses RAID is substantially slower than using deep copies or erasure coding chunks.

Workload considerations

One of the key benefits of a Ceph storage cluster is the ability to support different types of workloads within the same storage cluster using performance domains.

Performance domains allow Red Hat Ceph Storage clusters to support different workloads within the same cluster. Each domain uses hardware optimized for its performance profile. Storage administrators can deploy pools on the appropriate domain to meet application requirements. Selecting correctly sized and optimized servers for these domains is essential for cluster design.

To the Ceph client interface that reads and writes data, a Ceph storage cluster appears as a simple pool where the client stores data. However, the storage cluster performs many complex operations in a manner that is

visible to the client interface. Ceph clients and Ceph object storage daemons, referred to as Ceph OSDs, or simply OSDs, both use the Controlled Replication Under Scalable Hashing (CRUSH) algorithm for the storage and retrieval of objects. Ceph OSDs can run in containers within the storage cluster.

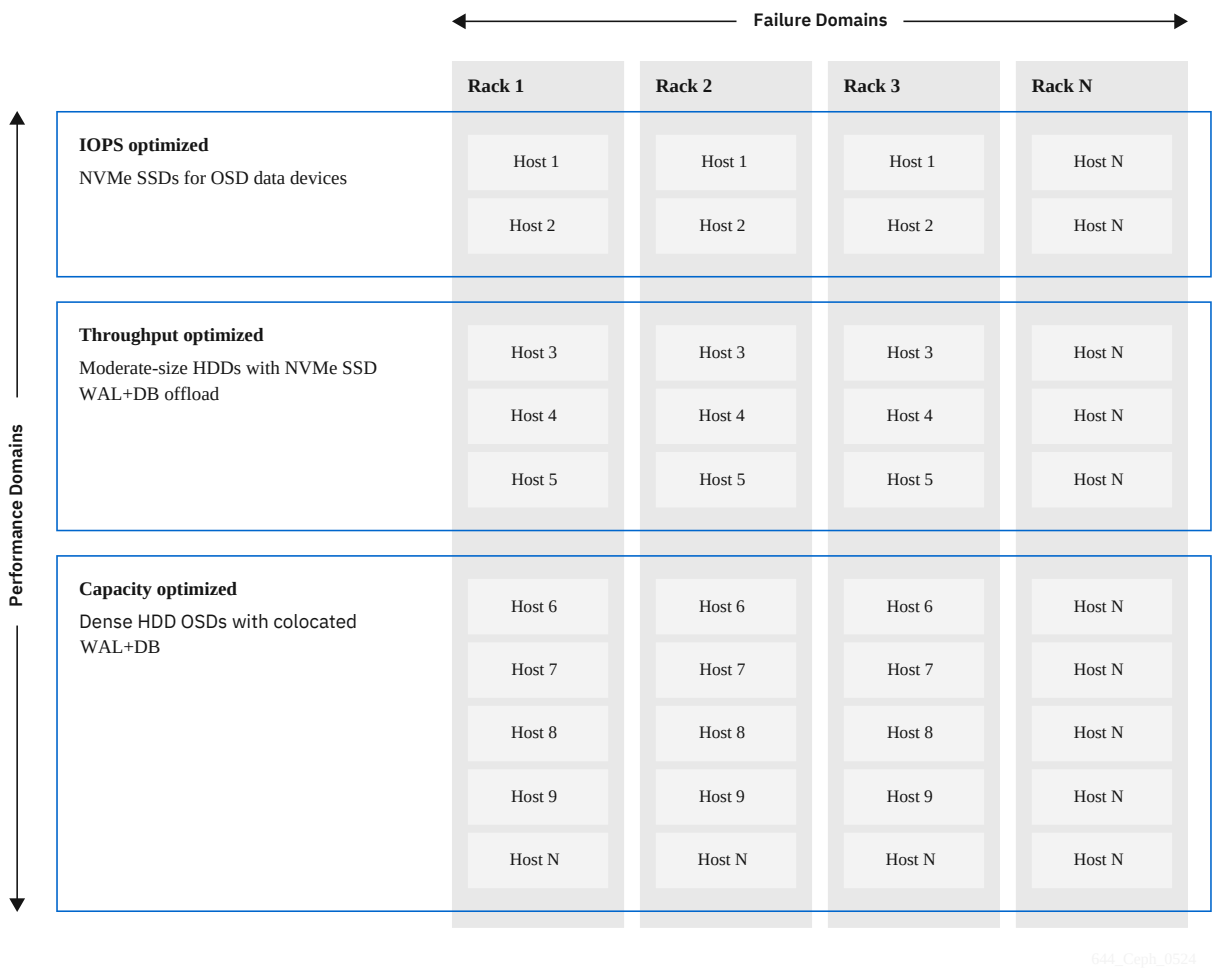
A CRUSH map describes a topography of cluster resources, and the map exists both on client hosts as well as Ceph Monitor hosts within the cluster. Ceph clients and Ceph OSDs both use the CRUSH map and the CRUSH algorithm. Ceph clients communicate directly with OSDs, eliminating a centralized object lookup and a potential performance bottleneck. With awareness of the CRUSH map and communication with their peers, OSDs can handle replication, backfilling, and recovery—allowing for dynamic failure recovery.

Ceph uses the CRUSH map to implement failure domains. Ceph also uses the CRUSH map to implement performance domains, which take the performance profile of the underlying hardware into consideration. The CRUSH map describes how Ceph stores data, and it is implemented as a simple hierarchy, specifically an acyclic graph, and a ruleset. The CRUSH map can support multiple hierarchies to separate one type of hardware performance profile from another. Ceph implements performance domains with device "classes".

For example, you can have these performance domains coexisting in the same Red Hat Ceph Storage cluster:

- Hard disk drives (HDDs) are typically appropriate for cost and capacity-focused workloads.
- Throughput-sensitive workloads typically use HDDs with Ceph write journals on solid-state drives (SSDs).
- IOPS-intensive workloads, such as MySQL and MariaDB, often use SSDs, including those with NVMe, SAS, or SATA interfaces.

Figure: Performance and Failure Domains



Workloads

Red Hat Ceph Storage is optimized for three primary workloads:

Important: Carefully consider the workload being run by Red Hat Ceph Storage clusters **BEFORE** considering what hardware to purchase because it can significantly impact the price and performance of the storage cluster. For example, if the workload is capacity-optimized and the hardware is better suited to a throughput-optimized workload, then hardware will be more expensive than necessary. Conversely, if the workload is throughput-optimized and the hardware is better suited to a capacity-optimized workload, then the storage cluster can suffer from poor performance.

IOPS optimized

Input, output per second (IOPS) optimization deployments are suitable for cloud computing operations, such as running MYSQL or MariaDB instances as virtual machines on OpenStack. IOPS optimized deployments require higher performance storage such as 15k RPM SAS drives and separate SSD journals to handle frequent write operations. Some high IOPS scenarios use all flash storage to improve IOPS and total throughput.
An IOPS-optimized storage cluster has the following properties:

- Lowest cost per IOPS.
- Highest IOPS per GB.
- 99th percentile latency consistency.

Uses for an IOPS-optimized storage cluster are:

- Typically block storage.
- 3x replication for hard disk drives (HDDs) or 2x replication for solid-state drives (SSDs).
- MySQL on OpenStack clouds.

Throughput optimized

Throughput-optimized deployments are suitable for serving up significant amounts of data, such as graphic, audio, and video content. Throughput-optimized deployments require high-bandwidth networking hardware, controllers, and hard disk drives with fast sequential read and write characteristics. If fast data access is a requirement, then use a throughput-optimized storage strategy. Also, if fast write performance is a requirement, using SSDs for journals can substantially improve write performance.
A throughput-optimized storage cluster has the following properties:

- Lowest cost per MBps (throughput).
- Highest MBps per TB.
- Highest MBps per BTU.
- Highest MBps per Watt.
- 97th percentile latency consistency.

Uses for a throughput-optimized storage cluster are:

- Block or object storage.
- 3x replication.
- Active performance storage for video, audio, and images.
- Streaming media, such as 4k video.

Capacity optimized

Capacity-optimized deployments are suitable for storing significant amounts of data as inexpensively as possible. Capacity-optimized deployments typically trade performance for a more attractive price point. For example, capacity-optimized deployments often use slower and less expensive SATA drives and colocate journals rather than using SSDs for journaling.
A cost and capacity-optimized storage cluster has the following properties:

- Lowest cost per TB.
- Lowest BTU per TB.
- Lowest Watts required per TB.

A cost and capacity-optimized storage cluster has the following properties:

- Typically object storage.
- Erasure coding for maximizing usable capacity
- Object archive.
- Video, audio, and image object repositories.

Network considerations for Red Hat Ceph Storage

An important aspect of a cloud storage solution is that storage clusters can run out of IOPS due to network latency, and other factors. The storage cluster can run out of throughput due to bandwidth constraints long before the storage clusters run out of storage capacity. As a result, the network hardware configuration must support the chosen workloads to meet price versus performance requirements.

Storage administrators prefer that a storage cluster recovers as quickly as possible. Carefully consider bandwidth requirements for the storage cluster network, be mindful of network link oversubscription, and separate the intra-cluster traffic from the client-to-cluster traffic. Network performance is increasingly important when considering the use of Solid State Disks (SSD), flash, NVMe, and other high performing storage devices.

Ceph supports a public network and a storage cluster network. The public network handles client traffic and communication with Ceph Monitors. The storage cluster network handles Ceph OSD heartbeats, replication, backfilling, and recovery traffic. Use a **minimum** of a single 10 Gb/s Ethernet link for storage hardware, and another 10 Gb/s Ethernet links can be added for connectivity and throughput.

Important:

- Allocate bandwidth to the storage cluster network, such that it is a multiple of the public network by using the **`osd_pool_default_size`** parameter as the basis for the multiple on replicated pools. Run the public and storage cluster networks on separate network cards.
- Use 10 Gb/s Ethernet for Red Hat Ceph Storage deployments in production. A 1 Gb/s Ethernet network is not suitable for production storage clusters.

In the case of a drive failure, replicating 1 TB of data across a 1 Gb/s network takes 3 hours and replicating 10 TB across a 1 Gb/s network takes 30 hours. Using 10 TB is the typical drive configuration. By contrast, with a 10 Gb/s Ethernet network, the replication times would be 20 minutes for 1 TB and 1 hour for 10 TB.

Note: When a Ceph OSD fails, the storage cluster recovers by replicating the data that it contained to other Ceph OSDs within the pool.

The failure of a larger domain such as a rack means that the storage cluster uses considerably more bandwidth. When building a storage cluster consisting of multiple racks, which is common for large storage implementations, consider using as much network bandwidth between switches in a "fat tree" design for optimal performance. A typical 10 Gb/s Ethernet switch has 48 10 Gb/s ports and four 40 Gb/s ports. Use the 40 Gb/s ports on the spine for maximum throughput. Alternatively, consider aggregating unused 10 Gb/s ports with QSFP+ and SFP+ cables into more 40 Gb/s ports to connect to other rack and spine routers. LACP mode 4 can be used to bond network interfaces. Use jumbo frames with a maximum transmission unit (MTU) of 9000, especially on the backend or cluster network.

Before installing and testing a Red Hat Ceph Storage cluster, verify the network throughput. 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 10 Gb/s Ethernet for the front side network. For large clusters, consider using 40 Gb/s ethernet for the backend or cluster network.

Important: For network optimization, use jumbo frames for a better CPU per bandwidth ratio, and a non-blocking network switch back-plane. Red Hat Ceph Storage requires the same MTU value throughout all networking devices in the communication path, end-to-end for both public and cluster networks. Verify

that the MTU value is the same on all hosts and networking equipment in the environment before using a Red Hat Ceph Storage cluster in production.

Reference

- Configuring multiple public networks to the cluster
- Configuring a private network
- Configuring a public network

Considerations for using a RAID controller with OSD hosts

Plan RAID controller usage carefully for Ceph OSD hosts to avoid performance issues, data corruption, and unnecessary complexity.

Note: Just a Bunch of Drives (JBOD), also known as Integrated Target (IT) mode, HBAs are recommended for SAS/SATA OSDs. RAID-capable HBAs do not offer significant benefit yet incur additional cost, monitoring demands, operational complexity, and decreased reliability.
When using OSD drives, avoid using RAID HBAs. Alternatively, the Linux boot volume can be mirrored by using MD software RAID (Linux `mdadm` utility).

If RAID controllers are already in use or required for specific configurations, review the following considerations to ensure optimal performance and reliability:

- If an OSD host has a RAID controller with 1 - 2 Gb of cache installed, enabling the write-back cache might result in increased small I/O write throughput. However, the cache must be nonvolatile.
- Most modern RAID controllers have super capacitors that provide enough power to drain volatile memory to nonvolatile NAND memory during a power-loss event. It is important to understand how a particular controller and its firmware behave after power is restored.
- Some RAID controllers require manual intervention. Hard disks typically advertise to the operating system whether their disk caches can be enabled or disabled by default. However, certain RAID controllers and some firmware do not provide such information. Verify that disk level caches are disabled to avoid file system corruption.
- Create a single RAID 0 volume with write-back for each Ceph OSD data drive with write-back cache enabled.
- If Serial Attached SCSI (SAS) or SATA connected Solid-state Drive (SSD) disks are also present on the RAID controller, then investigate whether the controller and firmware support *pass-through* mode. Enabling *pass-through* mode helps avoid caching logic, and generally results in lower latency for fast media.

Related information

Avoid using RAID and SAN solutions
Server and rack solutions

Tuning considerations for Red Hat Ceph Storage

Production Red Hat Ceph Storage clusters benefit from tuning operating system and BIOS / BMC settings. BMC refers to an embedded service processor used to configure and manage servers independently of the operating system, for example Dell iDRAC or HPE iLO.

When adjusting settings, be sure to apply the adjustments consistently across all hosts in the storage cluster. For more information, contact Red Hat Support.

There are various optimizations to help improve performance and reliability for many Red Hat Ceph Storage deployments. Coordinate with networking and infrastructure teams to align with fleet management. You can use TuneD or the `sysctl` utility for operation system tuning.

The following optimizations improve performance and reliability for many Red Hat Ceph Storage deployments.

TuneD

TuneD manages performance profiles that adjust system settings for specific use cases.

sysctl

The **`sysctl`** utility manages Linux kernel tunables. The following settings have shown performance and reliability benefits. Evaluate these settings against your environment and automation.

BIOS and BMC settings

Available settings and their possible values vary by server chassis manufacturer and model. Use the following recommendations as guidelines, and adapt them to your hardware and environment.

Thermal and performance profiles

Many modern server chassis include performance and thermal profile settings that affect CPU core management. By default, these are set to `Balanced`. Select the `Performance` or equivalent profile to improve throughput and reduce latency. This change may increase power consumption.

Cooling

Server chassis BMCs often adjust fan speeds dynamically based on detected hardware. The default settings may not provide optimal cooling. Select **Enhanced Cooling** or a medium fan offset for more effective cooling of storage devices, with the tradeoff of increased noise.

Hyper-Threading

Red Hat Ceph Storage is an integer workload that benefits from Hyper-Threading. Each physical CPU core appears to the operating system as two virtual cores (vcores) or *threads*. Keep Hyper-Threading enabled for most deployments.

Operating system tuning with TuneD

TuneD manages performance profiles that adjust system settings for specific use cases.

1. Install TuneD.

```
# dnf install -y tuned
# tuned-adm active
Current active profile: balanced
```

2. Activate the `network-latency` profile.
   The **`network-latency`** profile prevents deep CPU C-state transitions. These transitions are slow and can cause performance degradation or packet loss.

```
# tuned-adm profile network-latency
# tuned-adm active
Current active profile: network-latency
```

3. Verify the system C-states.
   On the **idle stats** tab, CPUs should report being in the following states most of the time: `C0` , `C1`, or `C1E`. The CPU should never be in the `C6` state. This improves throughput and latency, but may increase power consumption and heat generation.

4. After verifying that TuneD is running with the correct profile, install the `powertop` utility to verify that TuneD is running correctly.

```
# dnf install -y powertop
# powertop
```

A rolling reboot is recommended to apply all tunings.

> **Note:** Do not modify `/etc/sysctl.conf` or `/etc/sysctl.d/99-sysctl.conf` . Audit existing `/etc/sysctl.d` files to avoid overrides.

Operating system tuning with `sysctl`

The **`sysctl`** utility manages Linux kernel tunables. The following settings have shown performance and reliability benefits. Evaluate these settings against your environment and automation.

1. Create the `/etc/sysctl.d/99-ceph.conf` file with the following contents:

```
# Memory tuning
vm.min_free_kbytes = 4194304
vm.vfs_cache_pressure = 10
vm.swappiness = 1
vm.zone_reclaim_mode = 0
vm.dirty_ratio = 80
vm.dirty_background_ratio = 3

# I/O performance
fs.aio-max-nr = 50000000

# Network performance
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 1
net.ipv4.tcp_dsack = 0
net.ipv4.tcp_fack = 0
net.ipv4.tcp_congestion_control = bbr
net.core.default_qdisc = fq
net.ipv4.tcp_mtu_probing = 1
net.ipv4.tcp_syncookies = 1
net.core.somaxconn = 5000
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.tcp_fin_timeout = 20
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_low_latency = 1
net.ipv4.tcp_adv_win_scale = 1
net.ipv4.tcp_slow_start_after_idle = 0

# High-impact memory and backlog tuning
net.core.netdev_max_backlog = 250000
net.ipv4.tcp_max_tw_buckets = 2000000
net.ipv4.tcp_max_syn_backlog = 100000
net.ipv4.udp_mem = 4096 87380 33554432
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.tcp_wmem = 4096 65536 33554432
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864

# Connection tracking
net.netfilter.nf_conntrack_max = 1048576
net.nf_conntrack_max = 1048576
```

2. Complete the `nf_conntrack` table expansion by running the following commands:

```
echo 131072 > /sys/module/nf_conntrack/parameters/hashsize
echo "options nf_conntrack hashsize=131072" >/etc/modprobe.d/nf_conntrack.conf
```

   Ensure that your fleet automation persists these files by using Ansible, or a similar fleet configuration management tool.

A rolling reboot is recommended to apply all tunings.

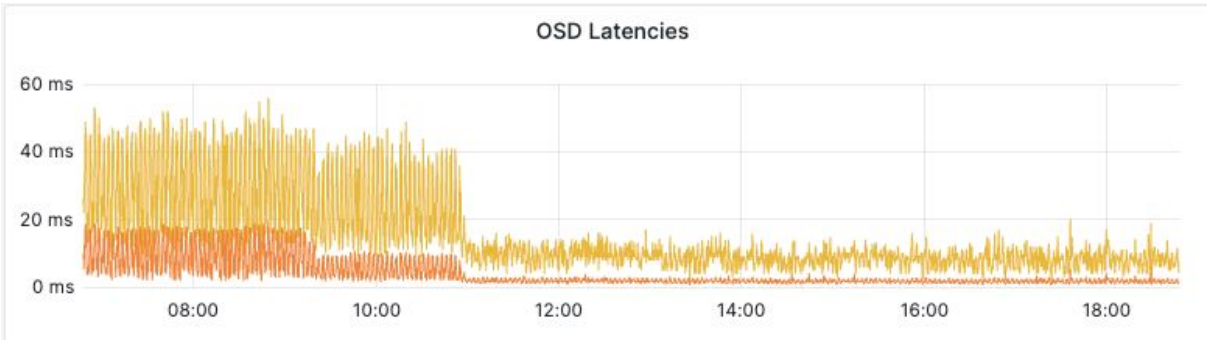
> **Note:** Do not modify `/etc/sysctl.conf` or `/etc/sysctl.d/99-sysctl.conf`. Audit existing `/etc/sysctl.d` files to avoid overrides.

Volatile HDD cache

Modern hard drives include up to 256 MB of cache. This cache is volatile and lacks Power Loss Protection (PLP). If power is lost, unwritten data may be corrupted. When volatile write cache is enabled, the Linux kernel flushes every write synchronously to media. This prevents data loss but often reduces OSD performance, especially average and tail write latencies.

You can improve performance by disabling the volatile cache. This cache is different from controller-level caches, such as those on RAID HBAs. The impact varies by drive and controller. Test changes in a lab or pre-GA environment before production deployment. Monitor `hdparm` and `sdparm` latency before and after changes by using your cluster dashboard.

“Figure: OSD Latencies dashboard output” on page 8 an example OSD latency output on the dashboard.
*Figure: **OSD Latencies** dashboard output*



Note: Persistence across reboot depends on drive firmware.

Use the `hdparm` and `sdparm` commands for live testing of cache settings.

Example commands for disabling the volatile cache

```
hdparm -W 0 /dev/sdXX   # SATA drives
sdparm --clear=WCE /dev/sdXX   # SAS drives
```

Example commands for enabling the volatile cache

```
hdparm -W 1 /dev/sdXX   # SATA drives
sdparm --set=WCE /dev/sdXX   # SAS drives
```

To disable volatile write cache reliably at boot, create a `udev` rule.

The following is an example for all SAS drives reporting to the manufacturer `SPINNERSRUS`.

```
cat <<EOF >/etc/udev/rules.d/90-wce.rules SUBSYSTEM=="block", ENV{SCSI_VENDOR}
=="SPINNERSRUS", ACTION=="add|change", RUN+="/usr/bin/sdparm --clear=WCE /dev/%k" EOF
```

To apply the changes immediately, reload udev.

```
udevadm control --reload
```

Colocation

Use this information to understand how colocation works and its advantages.

You can colocate containerized Ceph daemons on the same host.

The following are the advantages of colocating some of the services that Ceph offers:

- Significant improvement in total cost of ownership (TCO) at small scale
- Reduction from six hosts to three for the minimum configuration
- Easier upgrade
- Better resource isolation

How colocation works

With the help of the `cephadm` orchestrator, you can colocate one daemon from the following list with one or more OSD daemons (`ceph-osd`):

- Ceph Monitor (`ceph-mon`) and Ceph Manager (`ceph-mgr`) daemons
- RBD Mirror (`rbd-mirror`)
- Observability Stack (Grafana)

Additionally, for Ceph Object Gateway (`radosgw`) and Ceph File System (`ceph-mds`), you can colocate either with an OSD daemon plus a daemon from the previous list, excluding RBD mirror.

Note:

- Colocating two of the same kind of daemons on a specific node is not supported.
- Because `ceph-mon` and `ceph-mgr` work together closely they do not count as two separate daemons for the purposes of colocation.
- Colocate the Ceph Object Gateway with Ceph OSD containers to increase performance.

Colocation examples

The following examples demonstrate minimum cluster sizes that comply with the listed rules.

Example 1

“Colocated daemons: example 1” on page 10 and “Figure: Colocated daemons: example 1” on page 10 provide an example of colocated daemons with the following specifications:

Media

Full flash systems (SSDs)

Use case

Block (Ceph Block Device) and File (CephFS), or Object (Ceph Object Gateway)

Number of nodes

3

Replication scheme

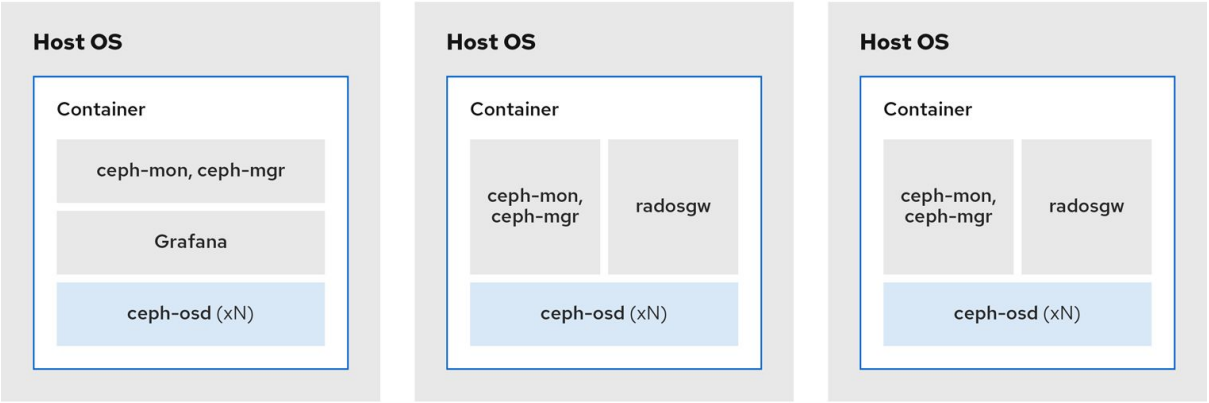
2

Colocated daemons: example 1

Host	Daemon	Daemon	Daemon
host1	OSD	Monitor/Manager	Grafana
host2	OSD	Monitor/Manager	RGW or CephFS
host3	OSD	Monitor/Manager	RGW or CaphFS

Note: The minimum size for a storage cluster with three replicas is four nodes. Similarly, the size of a storage cluster with two replicas is a three node cluster. It is a requirement to have a certain number of nodes for the replication factor with an extra node in the cluster to avoid extended periods with the cluster in a degraded state.

Figure: Colocated daemons: example 1



Example 2

“Colocated daemons: example 2” on page 11 and “Figure: Colocated daemons: example 2” on page 11 provide an example of colocated daemons with the following specifications:

Media

Full flash systems (SSDs) or spinning devices (HDDs)

Use case

Block (Ceph Block Device), File (CephFS), and Object (Ceph Object Gateway)

Number of nodes

4

Replication scheme

3

Colocated daemons: example 2			
Host	**Daemon**	**Daemon**	**Daemon**
host1	OSD	Grafana	CephFS
host2	OSD	Monitor/Manager	RGW
host3	OSD	Monitor/Manager	RGW
host4	OSD	Monitor/Manager	CephFS

Figure: Colocated daemons: example 2

The differences between colocation and non-colocated daemons

“Figure: Example of colocated daemons” on page 12 and “Figure: Non-colocated Daemons” on page 13 show the differences between storage clusters with colocated and non-colocated daemons.

Figure: Example of colocated daemons

Figure: Non-colocated Daemons

Operating system requirements

Before running Red Hat Ceph Storage, be sure to comply with all operating system requirements listed here.

- Red Hat Enterprise Linux entitlements are included in the Red Hat Ceph Storage subscription.
- For the latest supported Red Hat Enterprise Linux versions, see Compatibility matrix.
- Red Hat Ceph Storage is supported on container-based deployments only.
- Use the same architecture and deployment type across all nodes.
  For example, do not use a mixture of nodes with both AMD64 and Intel 64 architectures or a mixture of nodes with container-based deployments.

  > **Important:** Red Hat Ceph Storage does not support clusters with heterogeneous architectures or deployment types.

- By default, SELinux is set to `Enforcing` mode and the `ceph-selinux` packages are installed. For details, see Using SELinux for your OS version, on the Red Hat Customer Portal.

Deployment recommendations

Use the following recommendations when deploying the operating system for Red Hat Ceph Storage clusters. These recommendations are based on internal testing and lab environments and are intended to provide a consistent and optimized baseline. Adjust these settings based on your workload and infrastructure requirements.

> **Note:** These recommendations reflect common deployment practices from internal lab environments. Additional tuning might be required based on workload and infrastructure requirements.

Software profile

- Use the Red Hat Enterprise Linux Minimal installation profile to reduce package footprint.
- Install only the required packages for your deployment.
- Ensure that the `tuned` package is installed.

Storage and filesystem layout

- Deploy the operating system on a RAID-protected device, such as hardware RAID or MDRAID.
- Consider using RAID 1 for the operating system devices.
- Separate `/var/lib/ceph` to its own filesystem to prevent Ceph or monitoring services from filling the root filesystem.

CPU and performance tuning

- Ensure that the `tuned` profile is set to `network-latency`.
- For performance-sensitive workloads, consider setting the CPU governor to `performance` by using the `cpupower` command.
- If you use Intel CPUs, evaluate:
  - Setting the CPU mode to performance in the BIOS.
  - Disabling the Intel idle driver based on workload requirements.

Memory

Networking recommendations

- Use high-speed networking, such as 100 Gb, for all-flash nodes when possible.
- Use NIC bonding or teaming to aggregate network links if supported by your deployment topology.
- Configure the transmit hash policy to `layer3+4` for bonded interfaces.

Firewall and system services

- Use `firewalld` to manage firewall rules.
- Ceph deployment tools configure required rules automatically.

Minimum hardware considerations

Before running Red Hat Ceph Storage, be sure to comply with all minimum hardware requirements listed here.

Red Hat Ceph Storage can run on non-proprietary commodity hardware. Small production clusters and development clusters can run without performance optimization with modest hardware.

> **Note:** Disk space requirements are based on the Ceph daemons' default path under `/var/lib/ceph/` directory.

For more information about the Red Hat Ceph Storage internal components and the strategies around those components, see Storage strategies.

Important: Hardware accelerated compression in Ceph Object Gateway requires RHEL 9.4 on a Sapphire or Emerald Rapids Xeon CPU (or newer) with QAT devices. For more information, see Which 4th Gen Intel® Xeon® Scalable Processors Do Support Intel® QuickAssist Technology (Intel® QAT)? on Intel Product Support.

Red Hat Ceph Storage is supported on Intel and AMD x86-64 microprocessors.

- Red Hat Ceph Storage 9 introduces full support for clusters built with ARM (aarch64).

Use the following information for each process container and their minimum recommended requirements for various components.

For the latest architectural requirements for a supportable cluster, see Red Hat Ceph Storage: Supported configurations .

`ceph-osd-container`

- 1 × AMD64 or Intel 64 CPU vcore/hyperthread per OSD container.
- Minimum of 5 GB of RAM per OSD container.
- 1 × OS disk per host.
- 1 × storage drive per OSD container.

  Note: The storage drive cannot be shared with the OS disk.

- Optional WAL and DB offload for HDD OSDs.
  - Red Hat recommended.
  - 1 × SSD or NVMe or Optane partition or lvm per daemon.
  - Sizing is 4% of `block.data` for BlueStore for object, file, and mixed workloads and 1% of `block.data` for the BlueStore for Block Device.
- Minimum of 2 × 10 GB Ethernet NICs network connection.

`ceph-mon-container`

- 1 × AMD64 or Intel 64 CPU vcore/hyperthread per `mon-container`.
- 3 GB of RAM per `mon-container`.
- 10 GB of disk space per `mon-container`.

  Note: 50 GB disk space is recommended.

- Prometheus requires 20 GB to 50 GB under `/var/lib/ceph/` directory created as a separate file system to protect the contents under `/var/` directory.

`ceph-mgr-container`

- 1 × AMD64 or Intel 64 CPU vcore/hyperthread per `mgr-container`.
- 3 GB RAM per `mgr-container`.
- 2 × 1 GB Ethernet NICs network connection.

  Note: 10 GB network is recommended.

`ceph-radosgw-container`

- 1 × AMD64 or Intel 64 CPU vcore/hyperthread per `radosgw-container`.

- 1 GB RAM per daemon.
- 5 GB disk space per daemon.
- 1 × 1 GB Ethernet NICs network connection.

`ceph-mds-container`

- 1 × AMD64 or Intel 64 CPU vcore/hyperthread per `mds-container`.
  This number is highly dependent on the configurable MDS cache size. The RAM requirement is typically twice as much as the amount set in the `mds_cache_memory_limit` configuration setting.

  > **Note:** This is the memory for the daemon, not the overall system memory.

- 3 GB RAM per `mds-container`.
- 2 GB disk space per `mds-container`, plus considering any additional space required for possible debug logging.

  > **Note:** Minimum of 20 GiB of available space in the CephFS metadata pool.

- Minimum of 2 × 1 GB Ethernet NICs network connection.

  > **Note:**
  > - 10 GB network is recommended.
  > - This is the same network as the OSD containers. If you have a 10 GB network on your OSDs you should use the same on your MDS so that the MDS is not disadvantaged when it comes to latency.

Hardware

Get a high level guidance on selecting hardware for use with Red Hat Ceph Storage.

Software-defined storage presents many advantages to organizations that seek scale-out solutions to meet demanding applications and escalating storage needs. With a proven methodology and extensive testing with multiple vendors, Red Hat simplifies the process of selecting hardware to meet the demands of any environment. Importantly, the guidelines and example systems that are listed in this document are not a substitute for quantifying the impact of production workloads on sample systems.

Red Hat Ceph Storage significantly reduces the cost of storing enterprise data and helps organizations manage exponential data growth. The software is a robust and modern petabyte-scale storage platform for public or private cloud deployments.

Red Hat offers mature interfaces for enterprise block and object storage These interfaces make Red Hat Ceph Storage an optimal solution for active archive, rich media, and cloud infrastructure workloads characterized by tenant-agnostic OpenStack® environments[1]. Delivered as a unified, software-defined, scale-out storage platform, Red Hat Ceph Storage allows businesses to focus on improving application innovation and availability by offering capabilities such as:

- Scaling to hundreds of petabytes[2].
- No single point of failure in the cluster.
- Lower capital expenses (CapEx) by running on commodity server hardware.
- Lower operational expenses (OpEx) with self-managing and self-healing properties.

1. Ceph is and has been the leading storage for OpenStack according to several semi-annual OpenStack user surveys.
2. See Yahoo Cloud Object Store - Object Storage at Exabyte Scale for details.

Red Hat Ceph Storage can run on myriad industry-standard hardware configurations to satisfy diverse needs. To simplify and accelerate the cluster design process, Red Hat conducts extensive performance and suitability testing with participating hardware vendors. This testing allows evaluation of selected hardware under load and generates essential performance and sizing data for diverse workloads ultimately simplifying Ceph hardware selection. Multiple hardware vendors now provide server and rack-level solutions optimized for Red Hat Ceph Storage deployments with IOPS, throughput, and cost and capacity-optimized solutions as available options.

General principles for selecting hardware

As a storage administrator, you must select the appropriate hardware for running a production Red Hat Ceph Storage cluster. When selecting hardware for Red Hat Ceph Storage, review these following general principles. These principles will help save time, avoid common mistakes, save money and achieve a more effective solution.

For a list of supported configurations, see .

Prerequisites

- A planned use for Red Hat Ceph Storage.
- Linux System Administration Advance level with Red Hat Enterprise Linux certification.
- Storage administrator with Ceph Certification.

Identify performance use case

One of the most important steps in a successful Ceph deployment is identifying a price-to-performance profile suitable for the cluster's use case and workload. It is important to choose the correct hardware for the use case. For example, choosing IOPS-optimized hardware for a cloud storage application increases hardware costs unnecessarily. This is as opposed to selecting capacity-optimized hardware for its more attractive price point in an IOPS-intensive workload likely leads to unhappy users complaining about slow performance.

The following lists the primary use cases for Ceph.

IOPS optimized

IOPS optimized deployments are suitable for cloud computing operations, such as running MYSQL or MariaDB instances as virtual machines on OpenStack. IOPS optimized deployments require higher performance storage such as 15k RPM SAS drives and separate SSD journals to handle frequent write operations. Some high IOPS scenarios use all flash storage to improve IOPS and total throughput.

Throughput optimized

Throughput-optimized deployments are suitable for serving up significant amounts of data, such as graphic, audio and video content. Throughput-optimized deployments require networking hardware, controllers, and hard disk drives with acceptable total throughput characteristics. In cases where write performance is a requirement, SSD journals substantially improve write performance.

Capacity optimized

Capacity-optimized deployments are suitable for storing significant amounts of data as inexpensively as possible. Capacity-optimized deployments typically trade performance for a more attractive price point. For example, capacity-optimized deployments often use slower and less expensive SATA drives and colocate journals rather than using SSDs for journaling.

Use the information that is detailed within "General principles for selecting hardware" on page 18 and its subsections provides examples of Red Hat tested hardware suitable for these use cases.

Consider storage density

Hardware planning should include distributing Ceph daemons and other processes that use Ceph across many hosts to maintain high availability in the event of hardware faults. Balance storage density considerations with the need to rebalance the cluster in the event of hardware faults.

A common hardware selection mistake is to use very high storage density in small clusters, which can overload networking during backfill and recovery operations.

Identical hardware configuration

Create pools and define CRUSH hierarchies such that the OSD hardware within the pool is identical. Using the same hardware within a pool provides a consistent performance profile, simplifies provisioning, and streamlines troubleshooting.

Identical hardware configuration includes the following:

- Same controller.
- Same drive size.
- Same RPMs.
- Same seek times.
- Same I/O.
- Same network throughput.
- Same journal configuration.

Network considerations for Red Hat Ceph Storage

Carefully consider bandwidth requirements for the storage cluster network. An important aspect of a cloud storage solution is that storage clusters can run out of IOPS due to network latency, and other factors. Also, the storage cluster can run out of throughput due to bandwidth constraints long before the storage clusters run out of storage capacity. This means that the network hardware configuration must support the chosen workloads to meet price versus performance requirements.

Storage administrators prefer that a storage cluster recovers as quickly as possible. Carefully consider bandwidth requirements for the storage cluster network, be mindful of network link oversubscription, and segregate the intra-cluster traffic from the client-to-cluster traffic. Also consider that network performance is increasingly important when considering the use of Solid State Disks (SSD), flash, NVMe, and other high performing storage devices.

Ceph supports a public network and a storage cluster network. The public network handles client traffic and communication with Ceph Monitors. The storage cluster network handles Ceph OSD heartbeats, replication, backfilling, and recovery traffic. At a *minimum*, a single 10 GB Ethernet link should be used for storage hardware, and you can add additional 10 GB Ethernet links for connectivity and throughput.

> **Important:** Red Hat recommends allocating bandwidth to the storage cluster network, such that it is a multiple of the public network using the `osd_pool_default_size` as the basis for the multiple on replicated pools. Red Hat also recommends running the public and storage cluster networks on separate network cards.
> Red Hat recommends using 10 GB Ethernet for Red Hat Ceph Storage deployments in production. A 1 GB Ethernet network is not suitable for production storage clusters.

In the case of a drive failure, replicating 1 TB of data across a 1 GB Ethernet network takes 3 hours, and 3 TB takes 9 hours. Using 3 TB is the typical drive configuration. By contrast, with a 10 GB Ethernet network, the replication times would be 20 minutes and 1 hour. Remember that when a Ceph OSD fails, the storage cluster will recover by replicating the data it contained to other Ceph OSDs within the pool.

The failure of a larger domain such as a rack means that the storage cluster utilizes considerably more bandwidth. When building a storage cluster consisting of multiple racks, which is common for large storage implementations, consider utilizing as much network bandwidth between switches in a "fat tree" design for optimal performance. A typical 10 GB Ethernet switch has 48 10 GB ports and four 40 GB ports. Use the 40 GB ports on the spine for maximum throughput. Alternatively, consider aggregating unused 10 GB ports with QSFP+ and SFP+ cables into more 40 GB ports to connect to other rack and spine routers. Also, consider using LACP mode 4 to bond network interfaces. Additionally, use jumbo frames, with a maximum transmission unit (MTU) of 9000, especially on the backend or cluster network.

Before installing and testing a Red Hat Ceph Storage cluster, verify the network throughput. 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 10 GB ethernet for the front side network. For large clusters, consider using 40 GB ethernet for the backend or cluster network.

Important: For network optimization, Red Hat recommends using jumbo frames for a better CPU per bandwidth ratio, and a non-blocking network switch back-plane. Red Hat Ceph Storage requires the same MTU value throughout all networking devices in the communication path, end-to-end for both public and cluster networks. Verify that the MTU value is the same on all hosts and networking equipment in the environment before using a Red Hat Ceph Storage cluster in production.

Avoid using RAID and SAN solutions

Ceph can replicate or erasure code objects. RAID duplicates this functionality on the block level and reduces available capacity. Consequently, RAID is an unnecessary expense. Additionally, a degraded RAID will have a **negative** impact on performance.

Important: Red Hat does not support SAN as a storage backend for OSDs.

Important: Red Hat recommends that each hard drive be exported separately from the RAID controller as a single volume with write-back caching enabled.

This requires a battery-backed, or a non-volatile flash memory device on the storage controller. It is important to make sure the battery is working, as most controllers will disable write-back caching if the memory on the controller can be lost as a result of a power failure. Periodically, check the batteries and replace them if necessary, as they do degrade over time. See the storage controller vendor's documentation for details. Typically, the storage controller vendor provides storage management utilities to monitor and adjust the storage controller configuration without any downtime.

Using Just a Bunch of Drives (JBOD) in independent drive mode with Ceph is supported when using all Solid State Drives (SSDs), or for configurations with high numbers of drives per controller. For example, 60 drives attached to one controller. In this scenario, the write-back caching can become a source of I/O contention. Since JBOD disables write-back caching, it is ideal in this scenario. One advantage of using JBOD mode is the ease of adding or replacing drives and then exposing the drive to the operating system immediately after it is physically plugged in.

Summary of common mistakes when selecting hardware

Use the examples provided within this documentation of Red Hat tested configurations for different workloads to avoid some of the most common hardware selection mistakes.

These mistakes include the following:

- Repurposing underpowered legacy hardware for use with Ceph.
- Using dissimilar hardware in the same pool.
- Using 1 Gbps networks instead of 10Gbps or greater.
- Neglecting to setup both public and cluster networks.
- Using RAID instead of JBOD.
- Selecting drives on a price basis without regard to performance or throughput.
- Journaling on OSD data drives when the use case calls for an SSD journal.
- Having a disk controller with insufficient throughput characteristics.

Optimize workload performance domains

One of the key benefits of Ceph storage is the ability to support different types of workloads within the same cluster using Ceph performance domains. Dramatically different hardware configurations can be associated with each performance domain. Ceph system administrators can deploy storage pools on the appropriate performance domain, providing applications with storage tailored to specific performance and cost profiles.

Selecting appropriately sized and optimized servers for these performance domains is an essential aspect of designing a Red Hat Ceph Storage cluster.

The following lists provide the criteria Red Hat uses to identify optimal Red Hat Ceph Storage cluster configurations on storage servers. These categories are provided as general guidelines for hardware purchases and configuration decisions, and can be adjusted to satisfy unique workload blends. Actual hardware configurations chosen will vary depending on specific workload mix and vendor capabilities.

IOPS optimized

An IOPS-optimized storage cluster typically has the following properties:

- Lowest cost per IOPS.
- Highest IOPS per GB.
- 99th percentile latency consistency.

Typically uses for an IOPS-optimized storage cluster are:

- Typically block storage.
- 3x replication for hard disk drives (HDDs) or 2x replication for solid state drives (SSDs).
- MySQL on OpenStack clouds.

Throughput optimized

A throughput-optimized storage cluster typically has the following properties:

- Lowest cost per MBps (throughput).
- Highest MBps per TB.
- Highest MBps per BTU.
- Highest MBps per Watt.
- 97th percentile latency consistency.

Typically uses for an throughput-optimized storage cluster are:

- Block or object storage.
- 3x replication.
- Active performance storage for video, audio, and images.
- Streaming media.

Cost and capacity optimized

A cost- and capacity-optimized storage cluster typically has the following properties:

- Lowest cost per TB.
- Lowest BTU per TB.
- Lowest Watts required per TB.

Typically uses for an cost- and capacity-optimized storage cluster are:

- Typically object storage.
- Erasure coding common for maximizing usable capacity
- Object archive.
- Video, audio, and image object repositories.

How performance domains work

To the Ceph client interface that reads and writes data, a Ceph storage cluster appears as a simple pool where the client stores data. However, the storage cluster performs many complex operations in a manner that is completely transparent to the client interface. Ceph clients and Ceph object storage daemons (Ceph OSDs, or

simply OSDs) both use the controlled replication under scalable hashing (CRUSH) algorithm for storage and retrieval of objects. OSDs run on OSD hosts—the storage servers within the cluster.

A CRUSH map describes a topography of cluster resources, and the map exists both on client nodes as well as Ceph Monitor (MON) nodes within the cluster. Ceph clients and Ceph OSDs both use the CRUSH map and the CRUSH algorithm. Ceph clients communicate directly with OSDs, eliminating a centralized object lookup and a potential performance bottleneck. With awareness of the CRUSH map and communication with their peers, OSDs can handle replication, backfilling, and recovery—allowing for dynamic failure recovery.

Ceph uses the CRUSH map to implement failure domains. Ceph also uses the CRUSH map to implement performance domains, which simply take the performance profile of the underlying hardware into consideration. The CRUSH map describes how Ceph stores data, and it is implemented as a simple hierarchy (acyclic graph) and a ruleset. The CRUSH map can support multiple hierarchies to separate one type of hardware performance profile from another.

The following examples describe performance domains.

- Hard disk drives (HDDs) are typically appropriate for cost- and capacity-focused workloads.
- Throughput-sensitive workloads typically use HDDs with Ceph write journals on solid state drives (SSDs).
- IOPS-intensive workloads such as MySQL and MariaDB often use SSDs.

All of these performance domains can coexist in a Ceph storage cluster.

Server and rack solutions

Ceph provides both optimized server-level and rack-level solution SKUs. Hardware vendors provide both optimized server-level and rack-level solution SKUs. Validated through joint testing with Red Hat, these solutions offer predictable price-to-performance ratios for Ceph deployments, with a convenient modular approach to expand Ceph storage for specific workloads.

Many hardware vendors now offer both Ceph-optimized servers and rack-level solutions that are designed for distinct workload profiles. Red Hat works with multiple storage server vendors to test and evaluate specific cluster options for different cluster sizes and workload profiles. This work is to simplify the hardware selection process and reduce risk for organizations. Red Hat's exacting methodology combines performance testing with proven guidance for a broad range of cluster capabilities and sizes.

With appropriate storage servers and rack-level solutions, Red Hat Ceph Storage can provide storage pools that serve various workloads from throughput-sensitive and cost and capacity-focused workloads to emerging IOPS intensive workloads.

IOPS-optimized solutions

With the growing use of flash storage, organizations increasingly host IOPS-intensive workloads on Ceph storage clusters enabling emulation of high-performance public cloud solutions with private cloud storage. These workloads commonly involve structured data from MySQL-, MariaDB-, or PostgreSQL-based applications.

"Typical IOPS-optimized server configuration" on page 22 lists a typical server configuration.

Typical IOPS-optimized server configuration

Element	Specification
CPU	5 physical cores (10 hyperthreads) per NVMe SSD
RAM	24 GB baseline, plus 6 GB per OSD
Networking	10-Gigabit Ethernet (GbE) per 2 OSDs
OSD media	High performance enterprise NVMe SSDs, read-intensive or mixed-use
OSDs	One OSD per SSD for most deployments
BlueStore WAL/DB	High performance enterprise NVMe SSD, colocated on the OSD
Controller	Native PCIe bus

"Solutions SKUs for IOPS-optimized Ceph workloads, by cluster size" on page 23 provides SKU solutions for IOPs-optimized Ceph workloads, by cluster size.

For more information, see Supermicro® storage products.

Solutions SKUs for IOPS-optimized Ceph workloads, by cluster size			
Vendor	**Small (250 TB)**	**Medium (1 PB)**	**Large (2 PB+)**
SuperMicro [1]	SSG-122B, SSG-222B, SSG-121, SSG-110P-NTR10	SSG-122B-NE316R, SSG-110P-NTR10	SSG-222B-NE3X24R

Throughput-optimized solutions

Throughput-optimized Ceph solutions are usually centered around semi-structured or unstructured data. Large-block sequential I/O is typical.

"Typical throughput-optimized server configuration" on page 23 lists a typical server configuration.

Typical throughput-optimized server configuration	
Element	**Specification**
CPU	– 2 physical cores (4 hyperthreads) per OSD – 1 physical core (2 hyperthreads) per HDD
RAM	24 GB baseline, plus 6 GB per OSD
Networking	Redundant 25GE or 100GE links
OSD media	7,200 RPM Enterprise HDDs or NVMe SSDs
OSDs	One per storage drive
BlueStore WAL/DB	Offloaded to shared high-performance NVMe SSDs
Controller	RAID HBAs are not recommended; for SAS/SATA drives a JBOD style HBA is preferred.

Several vendors provide pre-configured server and rack-level solutions for throughput-optimized Ceph workloads. Extensive testing and evaluation of servers from Supermicro and Quanta Cloud Technologies (QCT) has been conducted.

This table describes individual OSD servers.

Individual OSD servers			
Vendor	**Small (250 TB)**	**Medium (1 PB)**	**Large (2 PB+)**
SuperMicro	SSG-110P-NTR10, SYS-112B-WR, SSG-122B-NE316R	SSG-122B-NE316R	SSG-222B-NE3X24R
QCT[1]	QxStor RCT-200	QxStor RCT-400	QxStor RCT-400

[1] QxStor Ceph Storage Open Source Edition

This table describes other servers configurable for throughput-optimized Ceph OSD workloads.

Other configurable servers for throughput-optimized Ceph OSD workloads			
Vendor	**Small (250 TB)**	**Medium (1 PB)**	**Large (2 PB+)**
Dell	R670	R670, R770	R7715
Cisco	C220 M8	C240 M8	C245 M8
Lenovo	SR645 V3	SR645 V3	SR665 V3

Cost and capacity-optimized solutions

Cost- and capacity-optimized solutions typically focus on higher capacity, or longer archival scenarios. Data can be either semi-structured or unstructured. Workloads include media archives, big data analytics archives, and machine image backups. Large-block sequential I/O is typical.

"Typical cost- and capacity-optimized server configuration" on page 24 lists a typical configuration.

Typical cost- and capacity-optimized server configuration	
Element	**Specification**
CPU	2 physical cores (4 hyperthreads) per OSD
RAM	24 GB baseline, plus 5 GB per OSD
Networking	Redundant 10 GE, 25 GE, or 100 GE links
OSD media	7,200 RPM Enterprise HDDs or QLC-class SSDs
OSDs	One per storage drive
BlueStore WAL/DB	Colocated on the storage drive
Controller	RAID HBAs are not recommended; for SAS/SATA drives a JBOD style HBA is preferred.

Other configurable servers for cost- and capacity-optimized workloads			
Vendor	**Small (250 TB)**	**Medium (1 PB)**	**Large (2 PB+)**
Dell	R470	R570	R7715
Cisco	C225 M8	C245 M8	C245M8
Lenovo	SR630 V3	SR645 V3	SR665 V3

For more information, see the following related information:

- *Deploying MySQL Databases on Red Hat Ceph Storage*
- *Intel® Data Center Blocks for Cloud – Red Hat OpenStack Platform with Red Hat Ceph Storage*

Recommended minimum hardware requirements for the Red Hat Ceph Storage Dashboard

The Red Hat Ceph Storage Dashboard has minimum hardware requirements.

The following are minimum requirements for using the Red Hat Ceph Storage Dashboard:

- 4 core processor at 2.5 GHz or higher
- 8 GB RAM
- 50 GB hard disk drive
- 1 Gigabit Ethernet network interface

Related information

High-level monitoring

Storage strategies

A storage strategy is a method of storing data that serves a particular use case. Creating storage strategies for Red Hat Ceph Storage clusters includes creating CRUSH hierarchies, estimating the number of placement groups, determining which type of storage pool to create, and managing pools.

–

Red Hat Ceph Storage supports the following, among other, storage strategies within a single cluster:

Storage for OpenStack volumes and images

When storing volumes and images for a cloud platform such as OpenStack, consider the workload characteristics. Cinder handles random small-block I/O, while Glance serves sequential, high-volume reads. These workloads are not well-suited to HDDs, whether SATA or SAS. Use conventional TLC SSDs in replicated pools. Because tri-mode HBAs are costly, an NVMe-only server may offer better performance at a lower total cost than a SAS-capable server.

Storage for S3- or Swift-compatible object data

For object data storage in S3- or Swift-compatible services, a more economical strategy may be appropriate. SATA HDDs or QLC SSDs can be used, depending on object size and access patterns. Very small objects (hundreds of KB or less) are not ideal for rotational media and should be stored on SSDs. Object storage clusters must include a modest pool of TLC or 4KB QLC SSDs for indexes and metadata. For mixed workloads, use multiple storage classes: replicated TLC or 4 KB QLC SSDs for small objects, and erasure-coded SATA HDDs or QLC SSDs for cost-effective bulk storage.

Interacting with the Ceph storage cluster is made simple from the perspective of the Ceph client. Storage strategies are invisible to the Ceph client in all but the storage capacity and performance.

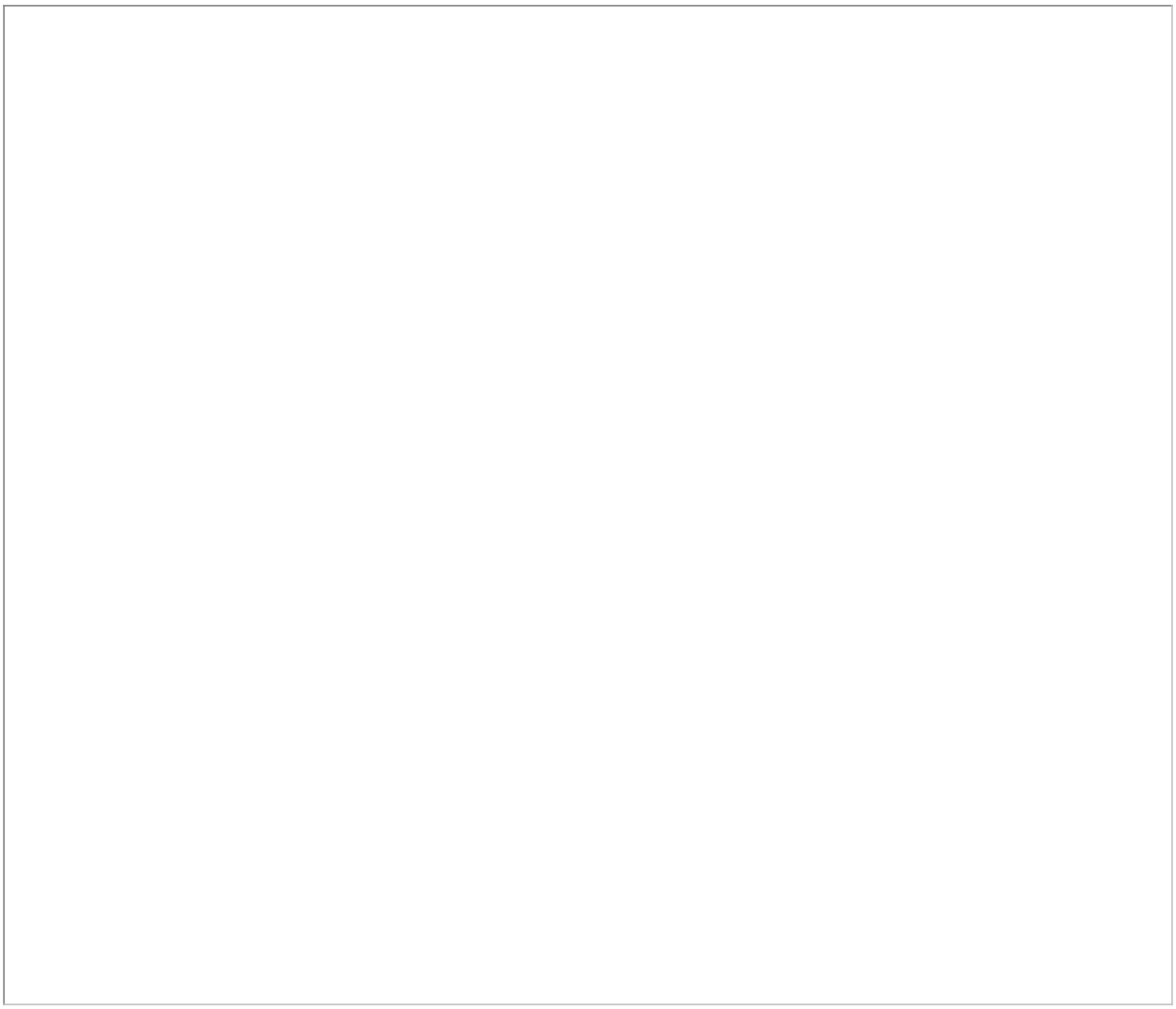
The interface enables the Ceph client to select one of the defined storage strategies. The interaction is as follows:

1. Connect to the cluster.
2. Create a pool I/O context.

“Figure: Ceph storage architecture data flow” on page 25 shows the logical data flow starting from the client into the Red Hat Ceph Storage cluster.

The diagram illustrates the Ceph object storage workflow, starting from data creation by the Ceph client, which sends data to RADOS for conversion and RADOS Object creation. RADOS Objects are organized into pools, divided into placement groups (`pg_num = 4`), and then mapped by the CRUSH algorithm to Object Storage Daemons (OSDs) for writing and replication.

Figure: Ceph storage architecture data flow



Storage strategies include:

- Storage hardware components, including HDDs, SSDs, and other related storage devices.
- The CRUSH maps that set up performance and failure domains for the storage media.
- The number of placement groups.
- The pool interface.

Red Hat Ceph Storage supports multiple storage strategies. Use cases, cost/benefit performance tradeoffs, and data durability are the primary considerations that drive storage strategies.

Use cases

Ceph provides massive storage capacity, and it supports numerous use cases. For example, the Ceph Block Device client is a leading storage backend for cloud platforms like OpenStack that provides limitless storage for volumes and images with high-performance features like copy-on-write cloning. Likewise, Ceph can provide container-based storage for OpenShift environments. By contrast, the Ceph Object Gateway client is a leading storage backend for cloud platforms that provides RESTful S3-compliant and Swift-compliant object storage for objects like audio, bitmap, video, and other data.

Cost/benefit performance tradeoff

Bigger is better. High durability is better. However, there is a price for each superlative quality, and a corresponding cost/benefit tradeoff. Consider the following use cases from a performance perspective:

SSDs can provide fast storage for relatively small amounts of data and journaling. Storing a database or object index might benefit from a pool of fast SSDs, but prove too expensive for other data. SAS drives with SSD journaling provide fast performance at an economical price for volumes and images. SATA drives without SSD journaling provide cheap storage with lower overall performance. When you create a CRUSH hierarchy of OSDs, you need to consider the use case and an acceptable cost/performance tradeoff.

Durability

In large-scale clusters, hardware failure is an expectation, not an exception. However, data loss and service interruption remain unacceptable. For this reason, data durability is important . Ceph addresses data durability with multiple deep copies of an object or with erasure coding and multiple coding chunks. Multiple copies or multiple coding chunks present an extra cost/benefit tradeoff. It is cheaper to store fewer copies or coding chunks, but it might lead to the inability to service write requests in a degraded state. Generally, one object with two more copies (that is, `size = 3`) or two coding chunks might allow a cluster to service writes in a degraded state while the cluster recovers. The CRUSH algorithm aids this process by ensuring that Ceph stores more copies or coding chunks in different locations within the cluster. This ensures that the failure of a single storage device or node does not lead to a loss of all the copies or coding chunks necessary to preclude data loss.

You can capture use cases, cost/benefit performance tradeoffs and data durability in a storage strategy and present it to a Ceph client as a storage pool.

> **Important:** Ceph's object copies or coding chunks make RAID obsolete. Do not use RAID because Red Hat Ceph Storage already handles data durability. A degraded RAID has a negative impact on performance and recovering data by using RAID is substantially slower than using deep copies or erasure coding chunks.

Configuring storage strategies

Configuring storage strategies is about assigning Ceph OSDs to a CRUSH hierarchy, defining the number of placement groups for a pool, and creating a pool.

1. Define a storage strategy - Storage strategies require you to analyze your use case, cost/benefit performance tradeoffs and data durability. Then, you create OSDs suitable for that use case. For example, you can create SSD-backed OSDs for a high-performance pool; SAS drive/SSD journal-backed OSDs for high-performance block device volumes and images; or SATA-backed OSDs for low-cost storage. Ideally, each OSD for a use case must have the same hardware configuration so that you have a consistent performance profile.
2. Define a CRUSH hierarchy - Ceph rules select a node, usually the `root`, in a CRUSH hierarchy, and identify the appropriate OSDs for storing placement groups and the objects they contain. Create a CRUSH hierarchy and a CRUSH rule for your storage strategy. CRUSH hierarchies get assigned directly to a pool by the CRUSH rule setting.
3. Calculate placement groups - Ceph shards a pool into placement groups. You do not have to manually set the number of placement groups for your pool. PG autoscaler sets an appropriate number of placement groups for your pool that remains within a healthy maximum number of placement groups when you assign multiple pools to the same CRUSH rule.
4. Create a pool - Finally, you must create a pool and determine whether it uses replicated or erasure-coded storage. Set the number of placement groups for the pool, the rule for the pool and the durability, such as size or `K+M` coding chunks.

AFTER COMPLETING THE TASK

Remember, the pool is the Ceph client's interface to the storage cluster, but the storage strategy is transparent to the Ceph client, except for capacity and performance.

CRUSH admin overview

The Controlled Replication Under Scalable Hashing (CRUSH) algorithm determines how to store and retrieve data by computing data storage locations.

CRUSH introduction

The CRUSH map for your storage cluster describes your device locations within CRUSH hierarchies and a rule for each hierarchy that determines how Ceph stores data.

The CRUSH map contains at least one hierarchy of nodes and leaves. The nodes of a hierarchy, called "buckets" in Ceph, are any aggregation of storage locations as defined by their type. For example, rows, racks, chassis, hosts, and devices. Each leaf of the hierarchy consists essentially of one of the storage devices in the list of

storage devices. A leaf is always contained in one node or "bucket." A CRUSH map also has a list of rules that determine how CRUSH stores and retrieves data.

> **Note:** Storage devices are added to the CRUSH map when adding an OSD to the cluster.

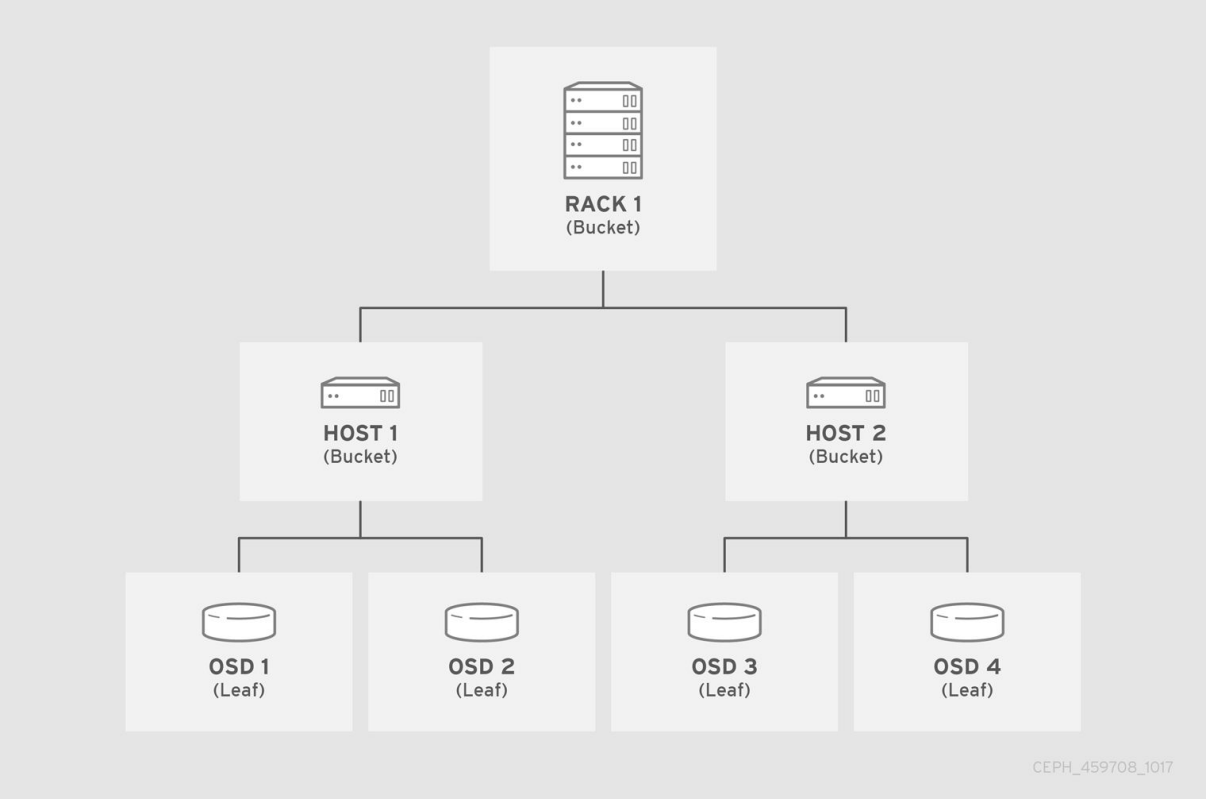
The CRUSH algorithm distributes data objects among storage devices according to a per-device weight value, approximating a uniform probability distribution. CRUSH distributes objects and their replicas or erasure-coding chunks according to the hierarchical cluster map an administrator defines. The CRUSH map represents the available storage devices and the logical buckets that contain them for the rule, and by extension each pool that uses the rule.

To map placement groups to OSDs across failure domains or performance domains, a CRUSH map defines a hierarchical list of bucket types; that is, under `types` in the generated CRUSH map. The purpose of creating a bucket hierarchy is to segregate the leaf nodes by their failure domains or performance domains or both. Failure domains include hosts, chassis, racks, power distribution units, pods, rows, rooms, and data centers. Performance domains include failure domains and OSDs of a particular configuration. For example, SSDs, SAS drives with SSD journals, SATA drives, and so on. Devices have the notion of a `class`, such as `hdd`, `ssd` and `nvme` to more rapidly build CRUSH hierarchies with a class of devices.

With the exception of the leaf nodes representing OSDs, the rest of the hierarchy is arbitrary, and you can define it according to your own needs if the default types do not suit your requirements. We recommend adapting your CRUSH map bucket types to your organization's hardware naming conventions and using instance names that reflect the physical hardware names. Your naming practice can make it easier to administer the cluster and troubleshoot problems when an OSD or other hardware malfunctions and the administrator needs remote or physical access to the host or other hardware.

In the following example, the bucket hierarchy has four leaf buckets (`osd 1-4`), two node buckets (`host 1-2`) and one rack node (`rack 1`).

Figure: CRUSH hierarchy



Since leaf nodes reflect storage devices declared under the `devices` list at the beginning of the CRUSH map, there is no need to declare them as bucket instances. The second lowest bucket type in the hierarchy usually aggregates the devices; that is, it is usually the computer containing the storage media, and uses whatever term administrators prefer to describe it, such as "node", "computer", "server," "host", "machine", and so on. In high density environments, it is increasingly common to see multiple hosts/nodes per card and per chassis. Make

sure to account for card and chassis failure too, for example, the need to pull a card or chassis if a node fails can result in bringing down numerous hosts/nodes and their OSDs.

When declaring a bucket instance, specify its type, give it a unique name as a string, assign it an optional unique ID expressed as a negative integer, specify a weight relative to the total capacity or capability of its items, specify the bucket algorithm such as `straw2`, and the hash that is usually `0` reflecting hash algorithm `rjenkins1`. A bucket can have one or more items. The items can consist of node buckets or leaves. Items can have a weight that reflects the relative weight of the item.

Dynamic data placement

Ceph Clients and Ceph OSDs both use the CRUSH map and the CRUSH algorithm.

- **Ceph Clients:** By distributing CRUSH maps to Ceph clients, CRUSH empowers Ceph clients to communicate with OSDs directly. This means that Ceph clients avoid a centralized object look-up table that could act as a single point of failure, a performance bottleneck, a connection limitation at a centralized look-up server and a physical limit to the storage cluster's scalability.
- **Ceph OSDs:** By distributing CRUSH maps to Ceph OSDs, Ceph empowers OSDs to handle replication, backfilling and recovery. This means that the Ceph OSDs handle storage of object replicas (or coding chunks) on behalf of the Ceph client. It also means that Ceph OSDs know enough about the cluster to re-balance the cluster (backfilling) and recover from failures dynamically.

CRUSH failure domain

Having multiple object replicas or `M` erasure coding chunks helps prevent data loss, but it is not sufficient to address high availability. By reflecting the underlying physical organization of the Ceph Storage Cluster, CRUSH can model, and thereby address, potential sources of correlated device failures. By encoding the cluster's topology into the cluster map, CRUSH placement policies can separate object replicas or erasure coding chunks across different failure domains while still maintaining the desired pseudo-random distribution.

For example, to address the possibility of concurrent failures, it might be desirable to ensure that data replicas or erasure coding chunks are on devices using different shelves, racks, power supplies, controllers or physical locations. This helps to prevent data loss and allows the cluster to operate in a degraded state.

CRUSH performance domain

Ceph can support multiple hierarchies to separate one type of hardware performance profile from another type of hardware performance profile. For example, CRUSH can create one hierarchy for hard disk drives and another hierarchy for SSDs. *Performance domains* are hierarchies that take the performance profile of the underlying hardware into consideration. Performance domains are increasingly popular due to the need to support different performance characteristics. Operationally, these are just CRUSH maps with more than one `root` type bucket.

Some use case examples of CRUSH performance domains include object storage, cold storage, and SSD-backed pools.

- **Object storage:** Ceph hosts that serve as an object storage back end for S3 and Swift interfaces might take advantage of less expensive storage media such as SATA drives that might not be suitable for VMs—reducing the cost per gigabyte for object storage, while separating more economical storage hosts from more performing ones intended for storing volumes and images on cloud platforms. HTTP tends to be the bottleneck in object storage systems.
- **Cold storage**: Systems designed for cold storage—infrequently accessed data, or data retrieval with relaxed performance requirements—might take advantage of less expensive storage media and erasure coding. However, erasure coding might require a bit of additional RAM and CPU, and thus differ in RAM and CPU requirements from a host used for object storage or VMs.
- **SSD-backed pools:** SSDs are expensive, but they provide significant advantages over hard disk drives. SSDs have no seek time and they provide high total throughput. In addition to using SSDs for journaling, a cluster can support SSD-backed pools. Common use cases include high performance SSD pools. For example, it is possible to map the `.rgw.buckets.index` pool for the Ceph Object Gateway to SSDs instead of SATA drives.

A CRUSH map supports the notion of a device `class`. Ceph can discover aspects of a storage device and automatically assign a class such as `hdd`, `ssd` or `nvme`. However, CRUSH is not limited to these defaults. For example, CRUSH hierarchies might also be used to separate different types of workloads. For example, an SSD

might be used for a journal or write-ahead log, a bucket index or for raw object storage. CRUSH can support different device classes, such as `ssd-bucket-index` or `ssd-object-storage` so Ceph does not use the same storage media for different workloads—making performance more predictable and consistent.

Behind the scenes, Ceph generates a crush root for each device-class. These roots should only be modified by setting or changing device classes on OSDs. You can view the generated roots using the following command:

Example

```
[ceph: root@host01 /]# ceph osd crush tree --show-shadow

 ID   CLASS  WEIGHT    TYPE NAME
-24    ssd   4.54849  root default~ssd
-19    ssd   0.90970      host ceph01~ssd
  8    ssd   0.90970          osd.8
-20    ssd   0.90970      host ceph02~ssd
  7    ssd   0.90970          osd.7
-21    ssd   0.90970      host ceph03~ssd
  3    ssd   0.90970          osd.3
-22    ssd   0.90970      host ceph04~ssd
  5    ssd   0.90970          osd.5
-23    ssd   0.90970      host ceph05~ssd
  6    ssd   0.90970          osd.6
 -2    hdd  50.94173  root default~hdd
 -4    hdd   7.27739      host ceph01~hdd
 10    hdd   7.27739          osd.10
-12    hdd  14.55478      host ceph02~hdd
  0    hdd   7.27739          osd.0
 12    hdd   7.27739          osd.12
 -6    hdd  14.55478      host ceph03~hdd
  4    hdd   7.27739          osd.4
 11    hdd   7.27739          osd.11
-10    hdd   7.27739      host ceph04~hdd
  1    hdd   7.27739          osd.1
 -8    hdd   7.27739      host ceph05~hdd
  2    hdd   7.27739          osd.2
 -1         55.49022  root default
 -3          8.18709      host ceph01
 10    hdd   7.27739          osd.10
  8    ssd   0.90970          osd.8
-11         15.46448      host ceph02
  0    hdd   7.27739          osd.0
 12    hdd   7.27739          osd.12
  7    ssd   0.90970          osd.7
 -5         15.46448      host ceph03
  4    hdd   7.27739          osd.4
 11    hdd   7.27739          osd.11
 3     ssd   0.90970         osd.3
 -9          8.18709      host ceph04
  1    hdd   7.27739          osd.1
  5    ssd   0.90970          osd.5
 -7          8.18709      host ceph05
  2    hdd   7.27739          osd.2
  6    ssd   0.90970          osd.6
```

Using different device classes

To create performance domains, use device classes and a single CRUSH hierarchy.

To create performance domains, add OSDs to the CRUSH hierarchy, then do the following:

1. Add a class to each device. For example:

   Syntax

   ```
   ceph osd crush set-device-class <class> <osdId> [<osdId>]
   ceph osd crush set-device-class hdd osd.0 osd.1 osd.4 osd.5
   ceph osd crush set-device-class ssd osd.2 osd.3 osd.6 osd.7
   ```

2. Then, create rules to use the devices.

   Syntax

   ```
   ceph osd crush rule create-replicated <rule-name> <root> <failure-domain-type>
  <class>
   ceph osd crush rule create-replicated cold default host hdd
   ceph osd crush rule create-replicated hot default host ssd
   ```

3. Finally, set pools to use the rules.

   Syntax

```
ceph osd pool set <poolname> crush_rule <rule-name>
ceph osd pool set cold_tier crush_rule cold
ceph osd pool set hot_tier crush_rule hot
```

Note: There is no need to manually edit the CRUSH map.

CRUSH hierarchy

The CRUSH map is a directed acyclic graph, so it can accommodate multiple hierarchies, for example, performance domains.

The easiest way to create and modify a CRUSH hierarchy is with the Ceph CLI; however, you can also decompile a CRUSH map, edit it, recompile it, and activate it.

When declaring a bucket instance with the Ceph CLI, you must specify its type and give it a unique string name. Ceph automatically assigns a bucket ID, sets the algorithm to `straw2`, sets the hash to `0` reflecting `rjenkins1` and sets a weight. When modifying a decompiled CRUSH map, assign the bucket a unique ID expressed as a negative integer (optional), specify a weight relative to the total capacity/capability of its item(s), specify the bucket algorithm (usually `straw2`), and the hash (usually `0`, reflecting hash algorithm `rjenkins1`).

A bucket can have one or more items. The items can consist of node buckets (for example, racks, rows, hosts) or leaves (for example, an OSD disk). Items can have a weight that reflects the relative weight of the item.

When modifying a decompiled CRUSH map, you can declare a node bucket with the following syntax:

```
[bucket-type] [bucket-name] {
    id [a unique negative numeric ID]
    weight [the relative capacity/capability of the item(s)]
    alg [the bucket type: uniform | list | tree | straw2 ]
    hash [the hash type: 0 by default]
    item [item-name] weight [weight]
}
```

For example, we would define two host buckets and one rack bucket. The OSDs are declared as items within the host buckets:

```
host node1 {
    id -1
    alg straw2
    hash 0
    item osd.0 weight 1.00
    item osd.1 weight 1.00
}

host node2 {
    id -2
    alg straw2
    hash 0
    item osd.2 weight 1.00
    item osd.3 weight 1.00
}

rack rack1 {
    id -3
    alg straw2
    hash 0
    item node1 weight 2.00
    item node2 weight 2.00
}
```

Note: In the foregoing example, note that the rack bucket does not contain any OSDs. Rather it contains lower level host buckets, and includes the sum total of their weight in the item entry.

CRUSH location

A CRUSH location is the position of an OSD in terms of the CRUSH map's hierarchy.

When you express a CRUSH location on the command line interface, a CRUSH location specifier takes the form of a list of name/value pairs describing the OSD's position. For example, if an OSD is in a particular row, rack, chassis and host, and is part of the `default` CRUSH tree, its crush location could be described as:

```
root=default row=a rack=a2 chassis=a2a host=a2a1
```

Note:

1. The order of the keys does not matter.
2. The key name (left of =) must be a valid CRUSH `type`. By default these include `root`, `datacenter`, `room`, `row`, `pod`, `pdu`, `rack`, `chassis` and `host`. You might edit the CRUSH map to change the types to suit your needs.
3. You do not need to specify all the buckets/keys. For example, by default, Ceph automatically sets a `ceph-osd` daemon's location to be `root=default host={HOSTNAME}` (based on the output from `hostname -s`).

Adding a bucket

Add a bucket type for each CRUSH hierarchy.

To add a bucket instance to your CRUSH hierarchy, specify the bucket name and its type. Bucket names must be unique in the CRUSH map.

```
ceph osd crush add-bucket {name} {type}
```

If you plan to use multiple hierarchies, for example, for different hardware performance profiles, consider naming buckets based on their type of hardware or use case.

For example, you could create a hierarchy for solid state drives (`ssd`), a hierarchy for SAS disks with SSD journals (`hdd-journal`), and another hierarchy for SATA drives (`hdd`):

```
ceph osd crush add-bucket ssd-root root
ceph osd crush add-bucket hdd-journal-root root
ceph osd crush add-bucket hdd-root root
```

The Ceph CLI outputs:

```
added bucket ssd-root type root to crush map
added bucket hdd-journal-root type root to crush map
added bucket hdd-root type root to crush map
```

Important: Using colons (`:`) in bucket names is not supported.

Add an instance of each bucket type you need for your hierarchy. The following example demonstrates adding buckets for a row with a rack of SSD hosts and a rack of hosts for object storage.

```
ceph osd crush add-bucket ssd-row1 row
ceph osd crush add-bucket ssd-row1-rack1 rack
ceph osd crush add-bucket ssd-row1-rack1-host1 host
ceph osd crush add-bucket ssd-row1-rack1-host2 host
ceph osd crush add-bucket hdd-row1 row
ceph osd crush add-bucket hdd-row1-rack1 rack
ceph osd crush add-bucket hdd-row1-rack1-host1 host
ceph osd crush add-bucket hdd-row1-rack1-host2 host
ceph osd crush add-bucket hdd-row1-rack1-host3 host
ceph osd crush add-bucket hdd-row1-rack1-host4 host
```

Once you have completed these steps, view your tree.

```
ceph osd tree
```

Notice that the hierarchy remains flat. You must move your buckets into a hierarchical position after you add them to the CRUSH map.

Moving a bucket

When you create your initial cluster, Ceph has a default CRUSH map with a root bucket named `default` and your initial OSD hosts appear under the `default` bucket. When you add a bucket instance to your CRUSH map, it appears in the CRUSH hierarchy, but it does not necessarily appear under a particular bucket. Moving a bucket instance to a particular location in your CRUSH hierarchy, in these cases, is necessary.

To move a bucket instance to a particular location in your CRUSH hierarchy, specify the bucket name and its type.

Example

```
ceph osd crush move ssd-row1 root=ssd-root
ceph osd crush move ssd-row1-rack1 row=ssd-row1
ceph osd crush move ssd-row1-rack1-host1 rack=ssd-row1-rack1
ceph osd crush move ssd-row1-rack1-host2 rack=ssd-row1-rack1
```

Once you have completed these steps, you can view your tree.

```
ceph osd tree
```

> **Note:** You can also use `ceph osd crush create-or-move` to create a location while moving an OSD.

Removing a bucket

Learn how to remove a bucket.

To remove a bucket instance from your CRUSH hierarchy, specify the bucket name. For example:

```
ceph osd crush remove {bucket-name}
```

Or:

```
ceph osd crush rm {bucket-name}
```

> **Note:** The bucket must be empty in order to remove it.

If you are removing higher level buckets (for example, a root like `default`), check to see if a pool uses a CRUSH rule that selects that bucket. If so, you need to modify your CRUSH rules; otherwise, peering fails.

CRUSH bucket algorithms

When you create buckets using the Ceph CLI, Ceph sets the algorithm to `straw2` by default. Ceph supports four bucket algorithms, each representing a tradeoff between performance and reorganization efficiency.

If you are unsure of which bucket type to use, we recommend using a `straw2` bucket. The bucket algorithms are uniform, list, tree, and straw2.

1. **Uniform:** Uniform buckets aggregate devices with **exactly** the same weight. For example, when firms commission or decommission hardware, they typically do so with many machines that have exactly the same physical configuration (for example, bulk purchases). When storage devices have exactly the same weight, you can use the `uniform` bucket type, which allows CRUSH to map replicas into uniform buckets in constant time. With non-uniform weights, you should use another bucket algorithm.
2. **List**: List buckets aggregate their content as linked lists. Based on the RUSH (Replication Under Scalable Hashing) $_{P}$ algorithm, a list is a natural and intuitive choice for an **expanding cluster**: either an object is

relocated to the newest device with some appropriate probability, or it remains on the older devices as before. The result is optimal data migration when items are added to the bucket. Items removed from the middle or tail of the list, however, can result in a significant amount of unnecessary movement, making list buckets most suitable for circumstances in which they **never, or very rarely shrink**.

3. **Tree**: Tree buckets use a binary search tree. They are more efficient than listing buckets when a bucket contains a larger set of items. Based on the RUSH (Replication Under Scalable Hashing) $_R$ algorithm, tree buckets reduce the placement time to zero (log $_n$), making them suitable for managing much larger sets of devices or nested buckets.
4. **Straw2 (default):** List and Tree buckets use a divide and conquer strategy in a way that either gives certain items precedence, for example, those at the beginning of a list or obviates the need to consider entire subtrees of items at all. That improves the performance of the replica placement process, but can also introduce suboptimal reorganization behavior when the contents of a bucket change due an addition, removal, or re-weighting of an item. The `straw2` bucket type allows all items to fairly "compete" against each other for replica placement through a process analogous to a draw of straws.

Ceph OSDs in CRUSH

Once you have a CRUSH hierarchy for the OSDs, add OSDs to the CRUSH hierarchy. You can also move or remove OSDs from an existing hierarchy.

"CRUSH rule values" on page 34 describes the Ceph CLI usage values.

CRUSH rule values

Value	Description	Type	Required	Example
`id`	The numeric ID of the OSD.	Integer	Yes	`0`
`name`	The full name of the OSD.	String	Yes	`osd.0`
`weight`	The CRUSH weight for the OSD.	Double	Yes	`2.0`
`root`	The name of the root bucket of the hierarchy or tree in which the OSD resides.	Key-value pair.	Yes	`root=default`, `root=replicated_rule`, and so on
`bucket-type`	One or more name-value pairs, where the name is the bucket type and the value is the bucket's name. You can specify a CRUSH location for an OSD in the CRUSH hierarchy.	Key-value pair.	No	`datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1`

Viewing OSDs in CRUSH

The **`ceph osd crush tree`** command outputs CRUSH buckets and items in a tree view. Use this command to determine a list of OSDs in a particular bucket. It gives output similar to the **`ceph osd tree`** command.

To return additional details, run the following:

```
# ceph osd crush tree -f json-pretty
```

The command returns an output similar to the following:

```
[
    {
        "id": -2,
```

```
        "name": "ssd",
        "type": "root",
        "type_id": 10,
        "items": [
            {
                "id": -6,
                "name": "dell-per630-11-ssd",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 6,
                        "name": "osd.6",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.099991,
                        "depth": 2
                    }
                ]
            },
            {
                "id": -7,
                "name": "dell-per630-12-ssd",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 7,
                        "name": "osd.7",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.099991,
                        "depth": 2
                    }
                ]
            },
            {
                "id": -8,
                "name": "dell-per630-13-ssd",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 8,
                        "name": "osd.8",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.099991,
                        "depth": 2
                    }
                ]
            }
        ]
    },
    {
        "id": -1,
        "name": "default",
        "type": "root",
        "type_id": 10,
        "items": [
            {
                "id": -3,
                "name": "dell-per630-11",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 0,
                        "name": "osd.0",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.449997,
                        "depth": 2
                    },
                    {
                        "id": 3,
                        "name": "osd.3",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.289993,
                        "depth": 2
```

```
                    }
                ]
            },
            {
                "id": -4,
                "name": "dell-per630-12",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 1,
                        "name": "osd.1",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.449997,
                        "depth": 2
                    },
                    {
                        "id": 4,
                        "name": "osd.4",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.289993,
                        "depth": 2
                    }
                ]
            },
            {
                "id": -5,
                "name": "dell-per630-13",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 2,
                        "name": "osd.2",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.449997,
                        "depth": 2
                    },
                    {
                        "id": 5,
                        "name": "osd.5",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.289993,
                        "depth": 2
                    }
                ]
            }
        ]
    }
]
```


Adding an OSD to CRUSH

Adding a Ceph OSD to a CRUSH hierarchy is the final step before you might start an OSD (rendering it `up` and `in`) and Ceph assigns placement groups to the OSD. You must prepare a Ceph OSD before you add it to the CRUSH hierarchy.

Deployment utilities, such as the Ceph Orchestrator, can prepare a CephOSD before adding it to the CRUSH hierarchy. For example creating a Ceph OSD on a single node:

Syntax

```
ceph orch daemon add osd HOST:DEVICE,[DEVICE]
```

The CRUSH hierarchy is notional, so the `ceph osd crush add` command allows you to add OSDs to the CRUSH hierarchy wherever you wish. The location you specify *should* reflect its actual location. If you specify at least one bucket, the command places the OSD into the most specific bucket you specify, *and* it moves that bucket underneath any other buckets you specify.

To add an OSD to a CRUSH hierarchy:

Syntax

```
ceph osd crush add ID_OR_NAME WEIGHT [BUCKET_TYPE=BUCKET_NAME ...]
```

Important: If you specify only the root bucket, the command attaches the OSD directly to the root. However, CRUSH rules expect OSDs to be inside of hosts or chassis, and host or chassis *should* be inside of other buckets reflecting your cluster topology.

If you specify only the root bucket, the command attaches the OSD directly to the root. However, CRUSH rules expect OSDs to be inside of hosts or chassis, and host or chassis *should* be inside of other buckets reflecting your cluster topology.

The following example adds `osd.0` to the hierarchy:

```
ceph osd crush add osd.0 1.0 root=default datacenter=dc1 room=room1 row=foo rack=bar
host=foo-bar-1
```

Note: You can also use `ceph osd crush set` or `ceph osd crush create-or-move` to add an OSD to the CRUSH hierarchy.

Moving an OSD within a CRUSH Hierarchy

If the storage cluster topology changes, you can move an OSD in the CRUSH hierarchy to reflect its actual location.

Important: Moving an OSD in the CRUSH hierarchy means that Ceph will recompute which placement groups get assigned to the OSD, potentially resulting in significant redistribution of data

To move an OSD within the CRUSH hierarchy:

Syntax

```
ceph osd crush set ID_OR_NAME WEIGHT root=POOL_NAME  [BUCKET_TYPE=BUCKET_NAME...]
```

Note: You can also use `ceph osd crush create-or-move` to move an OSD within the CRUSH hierarchy.

Removing an OSD from a CRUSH Hierarchy

Removing an OSD from a CRUSH hierarchy is the first step when you want to remove an OSD from your cluster. When you remove the OSD from the CRUSH map, CRUSH recomputes which OSDs get the placement groups and data re-balances accordingly.

To remove an OSD from the CRUSH map of a running cluster, run the following:

Syntax

```
ceph osd crush remove NAME
```

Device class

The Ceph CRUSH map provides a lot of flexibility in controlling data placement.

The flexibility in controlling data placement is one of Ceph's greatest strengths. Early Ceph deployments used hard disk drives almost exclusively. Today, Ceph clusters are frequently built with multiple types of storage devices: HDD, SSD, NVMe, or even various classes of the foregoing. For example, it is common in Ceph Object Gateway deployments to have storage policies where clients can store data on slower HDDs and other storage policies for storing data on fast SSDs. Ceph Object Gateway deployments might even have a pool backed by fast

SSDs for bucket indices. Additionally, OSD nodes also frequently have SSDs used exclusively for journals or write-ahead logs that do NOT appear in the CRUSH map. These complex hardware scenarios historically required manually editing the CRUSH map, which can be time-consuming and tedious. It is not required to have different CRUSH hierarchies for different classes of storage devices.

CRUSH rules work in terms of the CRUSH hierarchy. However, if different classes of storage devices reside in the same hosts, the process becomes more complicated—requiring users to create multiple CRUSH hierarchies for each class of device, and then disable the `osd crush update on start` option that automates much of the CRUSH hierarchy management. Device classes eliminate this tediousness by telling the CRUSH rule what class of device to use, dramatically simplifying CRUSH management tasks.

> **Note:** The **`ceph osd tree`** command has a column reflecting a device class.

Reference

For more information, see "Using different device classes" on page 30 and "CRUSH storage strategies examples" on page 51.

Setting a device class

Use this information to set a device class for an OSD.

To set a device class for an OSD, run the following:

Syntax

```
ceph osd crush set-device-class  CLASS OSD_ID [OSD_ID..]
```

Example

```
[ceph: root@host01 /]# ceph osd crush set-device-class hdd osd.0 osd.1
[ceph: root@host01 /]# ceph osd crush set-device-class ssd osd.2 osd.3
[ceph: root@host01 /]# ceph osd crush set-device-class bucket-index osd.4
```

> **Note:** Ceph might assign a class to a device automatically. However, class names are simply arbitrary strings. There is no requirement to adhere to `hdd`, `ssd` or `nvme`. In the foregoing example, a device class named `bucket-index` might indicate an SSD device that a Ceph Object Gateway pool uses exclusively bucket index workloads. To change a device class that was already set, use `ceph osd crush rm-device-class` first.

Removing a device class

Use this information to remove a device class for an OSD.

Remove a device class for an OSD by running the following command:

```
ceph osd crush rm-device-class OSD_ID [OSD_ID..]
```

For example,

```
[ceph: root@host01 /]# ceph osd crush rm-device-class 20
done removing class of osd(s): 20
[ceph: root@host01 /]# ceph osd crush rm-device-class 19 21
done removing class of osd(s): 19,21
```

Renaming a device class

Use this information to rename a device class for OSDs.

To rename a device class for all OSDs that use that class, run the following:

Syntax

```
ceph osd crush class rename OLD_NAME NEW_NAME
```

Example

```
[ceph: root@host01 /]# ceph osd crush class rename hdd sas15k
```

Listing a device class

Use this information to list device classes within a CRUSH map.

To list device classes in the CRUSH map, run the following:

Syntax

```
ceph osd crush class ls
```

The output will look something like this:

Example

```
[
    "hdd",
    "ssd",
    "bucket-index"
]
```

Listing OSDs of a device class

Use this information to list oSDs that belong to a device class.

To list all OSDs that belong to a particular class, run the following:

Syntax

```
ceph osd crush class ls-osd CLASS
```

Example

```
[ceph: root@host01 /]# ceph osd crush class ls-osd hdd
```

The output is simply a list of OSD numbers. For example:

```
0
1
2
3
4
5
6
```

Listing CRUSH rules by class

Use this information to list CRUSH rules that reference the same class.

To list all CRUSH rules that reference the same class, run the following:

Syntax

```
ceph osd crush rule ls-by-class CLASS
```

Example

```
[ceph: root@host01 /]# ceph osd crush rule ls-by-class hdd
```

CRUSH weights

The CRUSH algorithm assigns a weight value in terabytes (by convention) per OSD device. The assigned weight is done with the objective of approximating a uniform probability distribution for write requests that assign new data objects to placement groups (PGs) and PGs to OSDs.

As a result, it is best practice to create CRUSH hierarchies with devices of the same type and size, and to assign the same weight. In addition, use devices with the same I/O and throughput characteristics so that you will also have uniform performance characteristics in your CRUSH hierarchy, even though performance characteristics do not affect data distribution.

Since using uniform hardware is not always practical, you might incorporate OSD devices of different sizes and use a relative weight so that Ceph will distribute more data to larger devices and less data to smaller devices.

Setting CRUSH weights of OSDs

Set CRUSH weights in terabytes with a CRUSH map.

To set an OSD CRUSH weight in terabytes within the CRUSH map, run the following command, with the value information provided in “OSD CRUSH reweight values” on page 40:

```
ceph osd crush reweight NAME WEIGHT
```

OSD CRUSH reweight values

Value	Description	Type	Required	Example
`name`	The full name of the OSD.	String	Yes.	`osd.0`
`weight`	The CRUSH weight for the OSD. This should be the size of the OSD in Terabytes, where `1.0` is 1 Terabyte. **Note:** This setting is used when creating an OSD or adjusting the CRUSH weight immediately after adding the OSD. It usually does not change over the life of the OSD.	Double	Yes.	`2.0`

Setting a bucket’s OSD weights

Set all Ceph OSD weights under a bucket by running a single command.

Using the **`ceph osd crush reweight`** command can be time-consuming. You can set (or reset) all Ceph OSD weights under a bucket (row, rack, node, and so on) by running the **`ceph osd crush reweight-subtree`** command.

```
ceph osd crush reweight-subtree NAME
```

Where *NAME* is the name of the CRUSH bucket.

Set an OSD's `in` weight

In some cases, it is necessary to change the `in` weight of a particular OSD. Use this information to set an OSD's `in` weight.

For the purposes of `ceph osd in` and `ceph osd out`, an OSD is either `in` the cluster or `out` of the cluster. That is how a monitor records an OSD's status. However, even though an OSD is `in` the cluster, it might be experiencing a malfunction such that you do not want to rely on it as much until you fix it (for example, replace a storage drive, change out a controller, and so on).

You can increase or decrease the `in` weight of a particular OSD (that is, without changing its weight in terabytes) by executing:

Syntax

```
ceph osd reweight ID WEIGHT
```

Where:

- `id` is the OSD number.
- `weight` is a range from 0.0-1.0, where `0` is not `in` the cluster (that is, it does not have any PGs assigned to it) and 1.0 is `in` the cluster (that is, the OSD receives the same number of PGs as other OSDs).

Setting the OSDs weight by utilization

CRUSH is designed to approximate a uniform probability distribution for write requests that assign new data objects PGs and PGs to OSDs. Despite the CRUSH design, it is possible for clusters to become imbalanced for various reasons. If this occurs, set the OSD weight by utilization.

OSD weight imbalance can occur from various reasons, for example:

- **Multiple Pools:** You can assign multiple pools to a CRUSH hierarchy, but the pools might have different numbers of placement groups, size (number of replicas to store), and object size characteristics.
- **Custom Clients:** Ceph clients such as block device, object gateway, and filesystem share data from their clients and stripe the data as objects across the cluster as uniform-sized smaller RADOS objects. So except for the foregoing scenario, CRUSH usually achieves its goal. However, there is another case where a cluster can become imbalanced: namely, using `librados` to store data without normalizing the size of objects. This scenario can lead to imbalanced clusters (for example, storing 100 1-MB objects and 10 4-MB objects will make a few OSDs have more data than the others).
- **Probability:** A uniform distribution will result in some OSDs with more PGs and some with less. For clusters with a large number of OSDs, the statistical outliers will be further out.

You can reweight OSDs by utilization by executing the following:

Syntax

```
ceph osd reweight-by-utilization [THRESHOLD] [WEIGHT_CHANGE_AMOUNT] [NUMBER_OF_OSDS] [--no-
increasing]
```

Example

```
[ceph: root@host01 /]# ceph osd test-reweight-by-utilization 110 .5 4 --no-increasing
```

Where:

- `threshold` is a percentage of utilization such that OSDs facing higher data storage loads will receive a lower weight and thus fewer PGs assigned to them. The default value is `120`, reflecting 120%. Any value from `100+` is a valid threshold. Optional.
- `weight_change_amount` is the amount to change the weight. Valid values are greater than `0.0 - 1.0`. The default value is `0.05`. Optional.
- `number_of_OSDs` is the maximum number of OSDs to reweight. For large clusters, limiting the number of OSDs to reweight prevents significant rebalancing. Optional.

- `no-increasing` is **off** by default. Increasing the osd weight is allowed when using the `reweight-by-utilization` or `test-reweight-by-utilization` commands. If this option is used with these commands, it prevents the OSD weight from increasing, even if the OSD is underutilized. Optional.

> **Important:** Executing `reweight-by-utilization` is recommended and somewhat inevitable for large clusters. Utilization rates might change over time, and as your cluster size or hardware changes, the weightings might need to be updated to reflect changing utilization. If you elect to reweight by utilization, you might need to re-run this command as utilization, hardware or cluster size change.

Executing this or other weight commands that assign a weight will override the weight assigned by this command (for example, `osd reweight-by-utilization`, `osd crush weight`, `osd weight`, `in` or `out`).

Setting an OSD's weight by placement group distribution

In CRUSH hierarchies with a smaller number of OSDs, it's possible for some OSDs to get more placement groups (PGs) than other OSDs, resulting in a higher load. You can reweight OSDs by PG distribution to address this situation.

Reweight OSDs by PG distribution, by running the following:

Syntax

```
osd reweight-by-pg POOL_NAME
```

Where:

- `poolname` is the name of the pool. Ceph will examine how the pool assigns PGs to OSDs and reweight the OSDs according to this pool's PG distribution. Note that multiple pools could be assigned to the same CRUSH hierarchy. Reweighting OSDs according to one pool's distribution could have unintended effects for other pools assigned to the same CRUSH hierarchy if they do not have the same size (number of replicas) and PGs.

Recalculating a CRUSH tree bucket weight

When manually editing CRUSH map weights, recalculate the CRUSH bucket tree weight.

CRUSH tree buckets should be the sum of their leaf weights. If you manually edit the CRUSH map weights, you should run the following to ensure that the CRUSH bucket tree accurately reflects the sum of the leaf OSDs under the bucket.

Syntax

```
osd crush reweight-all
```

Primary affinity

When a Ceph Client reads or writes data, it always contacts the primary OSD in the acting set. Sometimes an OSD is not well suited to act as a primary compared to other OSDs (for example, it has a slow disk or a slow controller). To prevent performance bottlenecks (especially on read operations) while maximizing utilization of your hardware, you can set a Ceph OSD's primary affinity so that CRUSH is less likely to use the OSD as a primary in an acting set.

For example, in set `[2, 3, 4]`, `osd.2` is the primary.

Syntax

```
ceph osd primary-affinity OSD_ID WEIGHT
```

Primary affinity is `1` by default (*that is,* an OSD might act as a primary). You might set the OSD primary range from `0-1`, where `0` means that the OSD might **NOT** be used as a primary and `1` means that an OSD might be used as a primary. When the weight is `< 1`, it is less likely that CRUSH will select the Ceph OSD Daemon to act as a primary.

CRUSH rules

CRUSH rules define how a Ceph client selects buckets and the primary OSD within them to store objects, and how the primary OSD selects buckets and the secondary OSDs to store replicas or coding chunks.

In some cases, you might create a rule that selects a pair of target OSDs backed by SSDs for two object replicas, and another rule that selects three target OSDs backed by SAS drives in different data centers for three replicas.

A rule takes the following form, where the values information is provided in "CRUSH rule values" on page 43:

```
rule <rulename> {

    id <unique number>
    type [replicated | erasure]
    min_size <min-size>
    max_size <max-size>
    step take <bucket-type> [class <class-name>]
    step [choose|chooseleaf] [firstn|indep] <N> <bucket-type>
    step emit
}
```

CRUSH rule values

Value	Description	Type	Required	Example/Default
`id`	A unique whole number for identifying the rule. Purpose: A component of the rule mask.	Integer	Yes.	Default: `0`
`type`	Describes a rule for either a storage drive replicated or erasure coded. Purpose: A component of the rule mask. Valid value: `replicated`	String	Yes.	Default: `replicated`
`min_size`	If a pool makes fewer replicas than this number, CRUSH will not select this rule. Purpose: A component of the rule mask.	Integer	Yes.	Default: `1`
`max_size`	If a pool makes more replicas than this number, CRUSH will not select this rule. Purpose: A component of the rule mask.	Integer	Yes.	Default: `10`

Value	Description	Type	Required	Example/Default
`step take <bucket-name> [class <class-name>]`	Takes a bucket name, and begins iterating down the tree. Purpose: A component of the rule.	N/A	Yes	Example: `step take data step take data class ssd`
`step choose firstn <num> type <bucket-type>`	Selects the number of buckets of the given type. The number is usually the number of replicas in the pool (that is, pool size). – If `<num> == 0`, choose `pool-num-replicas` buckets (all available). – If `<num> > 0 && < pool-num-replicas`, choose that many buckets. – If `<num> < 0`, it means `pool-num-replicas - {num}`. Purpose: A component of the rule. Prerequisite: Follows `step take` or `step choose`.	N/A	N/A	Example: `step choose firstn 1 type row`

Value	Description	Type	Required	Example/Default
`step chooseleaf firstn <num> type <bucket-type>`	Selects a set of buckets of `{bucket-type}` and chooses a leaf node from the subtree of each bucket in the set of buckets. The number of buckets in the set is usually the number of replicas in the pool (that is, pool size). – If `<num> == 0`, choose `pool-num-replicas` buckets (all available). – If `<num> > 0 && < pool-num-replicas`, choose that many buckets. – If `<num> < 0`, it means `pool-num-replicas - <num>`. Purpose: A component of the rule. Usage removes the need to select a device using two steps. Prerequisite: Follows `step take` or `step choose`.	N/A	N/A	Example: `step chooseleaf firstn 0 type row`
`step emit`	Outputs the current value and empties the stack. Typically used at the end of a rule, but might also be used to pick from different trees in the same rule. Purpose: A component of the rule. Prerequisite: Follows `step choose`.			Example: `step emit`

Value	Description	Type	Required	Example/Default
`firstn versus indep`	Controls the replacement strategy CRUSH uses when OSDs are marked down in the CRUSH map. If this rule is to be used with replicated pools it should be `firstn` and if it is for erasure-coded pools it should be `indep`.	N/A	N/A	You have a PG stored on OSDs 1, 2, 3, 4, 5 in which 3 goes down.. In the first scenario, with the `firstn` mode, CRUSH adjusts its calculation to select 1 and 2, then selects 3 but discovers it is down, so it retries and selects 4 and 5, and then goes on to select a new OSD 6. The final CRUSH mapping change is from *1, 2, 3, 4, 5* to *1, 2, 4, 5, 6*. In the second scenario, with `indep` mode on an erasure-coded pool, CRUSH attempts to select the failed OSD 3, tries again and picks out 6, for a final transformation from *1, 2, 3, 4, 5* to *1, 2, 6, 4, 5*. **Important:** A given CRUSH rule can be assigned to multiple pools, but it is not possible for a single pool to have multiple CRUSH rules.

Listing CRUSH rules

Use this information to list CRUSH rules from the command-line.

To list CRUSH rules from the command line, run the following:

Syntax

```
ceph osd crush rule list
ceph osd crush rule ls
```

Dumping CRUSH rules

Use this information to ldump the contents of a specific CRUSH rule from the command-line.

To dump the contents of a specific CRUSH rule, run the following:

Syntax

```
ceph osd crush rule dump NAME
```

Adding CRUSH rules

Use this information to add CRUSH rules from the command-line.

To add a CRUSH rule, you must specify a rule name, the root node of the hierarchy you wish to use, the type of bucket you want to replicate across (for example, *rack*, *row*, and so on and the mode for choosing the bucket.

Syntax

```
ceph osd crush rule create-simple RULENAME ROOT BUCKET_NAME FIRSTN_OR_INDEP
```

Ceph creates a rule with `chooseleaf` and one bucket of the type you specify.

Example

```
[ceph: root@host01 /]# ceph osd crush rule create-simple deleteme default host firstn
```

Create the following rule:

```
{ "id": 1,
  "rule_name": "deleteme",
  "type": 1,
  "min_size": 1,
  "max_size": 10,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_firstn",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
```

Creating CRUSH rules for replicated pools

Use this information to create a CRUSH rule for a replicated pool from the command-line.

To create a CRUSH rule for a replicated pool, run the following:

Syntax

```
ceph osd crush rule create-replicated NAME ROOT FAILURE_DOMAIN CLASS
```

Where:

- `<name>`: The name of the rule.
- `<root>`: The root of the CRUSH hierarchy.
- `<failure-domain>`: The failure domain. For example: `host` or `rack`.
- `<class>`: The storage device class. For example: `hdd` or `ssd`.

Example

```
[ceph: root@host01 /]# ceph osd crush rule create-replicated fast default host ssd
```

Creating CRUSH rules for erasure coded pools

To add a CRUSH rule for use with an erasure-coded pool, you might specify a rule name and an erasure coded profile.

- Specify the rule name and an erasure-coded profile.

  ```
  ceph osd crush rule create-erasure RULE_NAME PROFILE_NAME
  ```

  For example,

  ```
  [ceph: root@host01 /]# ceph osd crush rule create-erasure default default
  ```

  For more information, see “Erasure code profiles” on page 86.

Removing CRUSH rules

Use this information to remove a CRUSH rule from the command-line.

To remove a rule, run the following and specify the CRUSH rule name:

Syntax

```
ceph osd crush rule rm NAME
```

CRUSH tuning

Ceph provides the ability to adjust certain parameters of the CRUSH algorithm, also known as CRUSH tuning.

Important considerations

Before adjusting CRUSH parameters, be sure to consider the following:

- Adjusting CRUSH values might result in the shift of some placement groups (PGs) between storage nodes. If the Ceph cluster is already storing a lot of data, be prepared for some fraction of the data to move.
- The `ceph-osd` and `ceph-mon` daemons will start requiring the feature bits of new connections as soon as they receive an updated map. However, already-connected clients are effectively grandfathered in, and will misbehave if they do not support the new feature. Make sure when you upgrade your Ceph Storage Cluster daemons that you also update your Ceph clients.
- If the CRUSH tunables are set to non-legacy values and then later changed back to the legacy values, `ceph-osd` daemons will not be required to support the feature. However, the OSD peering process requires examining and understanding old maps. Therefore, you should not run old versions of the `ceph-osd` daemon if the cluster has previously used non-legacy CRUSH values, even if the latest version of the map has been switched back to using the legacy defaults.
- Clusters running supported releases should set optimal tunables, by using the following command:

  ```
  ceph osd crush tunables optimal
  ```

Tuning by changing to a known profile

Tuning by changing to a known profile is the preferred method.

> **Important:** Before you tune CRUSH, ensure that all Ceph clients and all Ceph daemons use the same version. If you have recently upgraded, ensure that you have restarted daemons and reconnected clients.

Set the CRUSH tunables after you upgrade, or if you receive a warning. Starting with version v0.74, Ceph issues a health warning if the CRUSH tunables are not set to their optimal values, the optimal values are the default as of v0.73.

To adjust the CRUSH tunables, change to a known profile.

You can select a profile on a running cluster with the following command:

```
ceph osd crush tunables PROFILE
```

Note: This might result in some data movement.

Available known profiles are as follows:

legacy

The legacy behavior from v0.47 (pre-Argonaut) and earlier.

argonaut

The legacy values supported by v0.48 (Argonaut) release.

bobtail

The values supported by the v0.56 (Bobtail) release.

firefly

The values supported by the v0.80 (Firefly) release.

hammer

The values supported by the v0.94 (Hammer) release.

jewel

The values supported by the v10.0.2 (Jewel) release.

optimal

The current best values.

default

The current default values for a new cluster.

If a health warning is emitted that the CRUSH tunables are not set to their optimal values, run one of the following commands:

- Adjust the tunables on the existing cluster.
  Adjusting the tunables on the cluster results in up to 10% data movement. This is the preferred route, but should be taken with care on a production cluster where the data movement might affect performance.Enable tunalbes by using the **`crush tunables`** command.

  ```
  ceph osd crush tunables optimal
  ```

  If at any time you need to revert to an earlier profile, use the following command:

  ```
  ceph osd crush tunables PROFILE
  ```

  Examples of when reverting might be necessary are the following:

  - Too much load.
  - Not enough progress.
  - An issue with client compatibility, such as old kernel cephfs or rbd clients or pre-bobtail librados clients.

  The following example restores the pre-v0.48 (Argonaut) values.

  ```
  ceph osd crush tunables legacy
  ```

- Add the `warn on legacy crush tunables = false` option to the `mon` section of the `ceph.conf` file.

  ```
  mon warn on legacy crush tunables = false
  ```

After changing the configuration file, either restart the monitors or apply the following option to the running monitors:

```
ceph tell mon.\* injectargs --no-mon-warn-on-legacy-crush-tunables
```

Tuning by modifying the CRUSH map

> **Important:** Only use this option if you can ensure that all clients are running recent code.

Adjust the tunables by extracting the CRUSH map, modifying the values, and then reinjecting the map into the cluster.

1. Extract the latest CRUSH map.

   ```
   ceph osd getcrushmap -o /tmp/crush
   ```

2. Adjust tunables.

   > **Important:** Use this option with extreme care.

   ```
   crushtool -i /tmp/crush --set-choose-local-tries 0 --set-choose-local-fallback-tries 0
   --set-choose-total-tries 50 -o /tmp/crush.new
   ```

   In addition to the values specified here, you also need to specify the **`--enable-unsafe-tunables`** argument to **`crushtool`** for this to work.

3. Reinject modified map.

   ```
   ceph osd setcrushmap -i /tmp/crush.new
   ```

Setting CRUSH legacy values

Legacy values for the CRUSH tunables can be set, when required.

- The special **`--enable-unsafe-tunables`** option is required.
- Be careful running old versions of the `ceph-osd` daemon after reverting to legacy values

```
crushtool -i /tmp/crush --set-choose-local-tries 2 --set-choose-local-fallback-tries 5 --
set-choose-total-tries 19 --set-chooseleaf-descend-once 0 --set-chooseleaf-vary-r 0 -o /
tmp/crush.legacy
```

Edit a CRUSH map

Generally, modifying your CRUSH map at runtime with the Ceph CLI is more convenient than editing the CRUSH map manually. However, there are times when you might choose to edit it, such as changing the default bucket types, or using a bucket algorithm other than `straw2`.

To edit an existing CRUSH map:

1. Get the CRUSH map, as described in “Getting the CRUSH map” on page 51.
2. Decompile the CRUSH map, as described in “Decompiling the CRUSH map” on page 51 .
3. Edit at least one of the devices, and buckets and rules.
4. Recompile the CRUSH map, as described in “Compiling the CRUSH map” on page 51.
5. Set the CRUSH map, as described in “Setting a CRUSH map” on page 51.

To activate a CRUSH Map rule for a specific pool, identify the common rule number and specify that rule number for the pool when creating the pool.

Getting the CRUSH map

Getting the CRUSH map is the necessary first step in editing the map.

To get the CRUSH map for your cluster, run the following:

Syntax

```
ceph osd getcrushmap -o COMPILED_CRUSHMAP_FILENAME
```

Ceph will output (-o) a compiled CRUSH map to the file name you specified. Since the CRUSH map is in a compiled form, you must decompile it first before you can edit it.

Decompiling the CRUSH map

Decompile the CRUSH map

To decompile a CRUSH map, run the following:

Syntax

```
crushtool -d COMPILED_CRUSHMAP_FILENAME -o DECOMPILED_CRUSHMAP_FILENAME
```

Ceph decompiles (-d) the compiled CRUSH map and send the output (-o) to the file name you specified.

Compiling the CRUSH map

To compile a CRUSH map, run the following:

Syntax

```
crushtool -c DECOMPILED_CRUSHMAP_FILENAME -o COMPILED_CRUSHMAP_FILENAME
```

Ceph will store a compiled CRUSH map to the file name you specified.

Setting a CRUSH map

To set the CRUSH map for your cluster, run the following:

Syntax

```
ceph osd setcrushmap -i COMPILED_CRUSHMAP_FILENAME
```

Ceph inputs the compiled CRUSH map of the file name you specified as the CRUSH map for the cluster.

CRUSH storage strategies examples

Set the CRUSH rules with device classes.

CRUSH can handle scenarios when you want to have most pools default to OSDs backed by large hard disk drives or some pools mapped to OSDs are backed by fast solid-state drives (SSDs). Use device classes to set rules for CRUSH to handle specific scenarios.

1. Add a class to each device.

   ```
   ceph osd crush set-device-class CLASS OSD_ID [OSD_ID]
   ```

   For example,

   ```
   [ceph:root@host01 /]# ceph osd crush set-device-class hdd osd.0 osd.1 osd.4 osd.5
   [ceph:root@host01 /]# ceph osd crush set-device-class ssd osd.2 osd.3 osd.6 osd.7
   ```

2. Create rules to use the devices.

   ```
   ceph osd crush rule create-replicated RULENAME ROOT FAILURE_DOMAIN_TYPE DEVICE_CLASS
   ```

   For example,

```
[ceph:root@host01 /]# ceph osd crush rule create-replicated cold default host hdd
[ceph:root@host01 /]# ceph osd crush rule create-replicated hot default host ssd
```

3. Set the pools to use the rules.

```
ceph osd pool set POOL_NAME crush_rule RULENAME
```

For example,

```
[ceph:root@host01 /]# ceph osd pool set cold crush_rule hdd
[ceph:root@host01 /]# ceph osd pool set hot crush_rule ssd
```

You do not need to manually edit the CRUSH map because one hierarchy can serve multiple classes of devices.
For example,

```
device 0 osd.0 class hdd
device 1 osd.1 class hdd
device 2 osd.2 class ssd
device 3 osd.3 class ssd
device 4 osd.4 class hdd
device 5 osd.5 class hdd
device 6 osd.6 class ssd
device 7 osd.7 class ssd

  host ceph-osd-server-1 {
      id -1
      alg straw2
      hash 0
      item osd.0 weight 1.00
      item osd.1 weight 1.00
      item osd.2 weight 1.00
      item osd.3 weight 1.00
  }

  host ceph-osd-server-2 {
      id -2
      alg straw2
      hash 0
      item osd.4 weight 1.00
      item osd.5 weight 1.00
      item osd.6 weight 1.00
      item osd.7 weight 1.00
  }

  root default {
      id -3
      alg straw2
      hash 0
      item ceph-osd-server-1 weight 4.00
      item ceph-osd-server-2 weight 4.00
  }


  rule cold {
      ruleset 0
      type replicated
      min_size 2
      max_size 11
      step take default class hdd
      step chooseleaf firstn 0 type host
      step emit
  }


  rule hot {
      ruleset 1
      type replicated
      min_size 2
      max_size 11
      step take default class ssd
      step chooseleaf firstn 0 type host
      step emit
  }
```

Placement groups

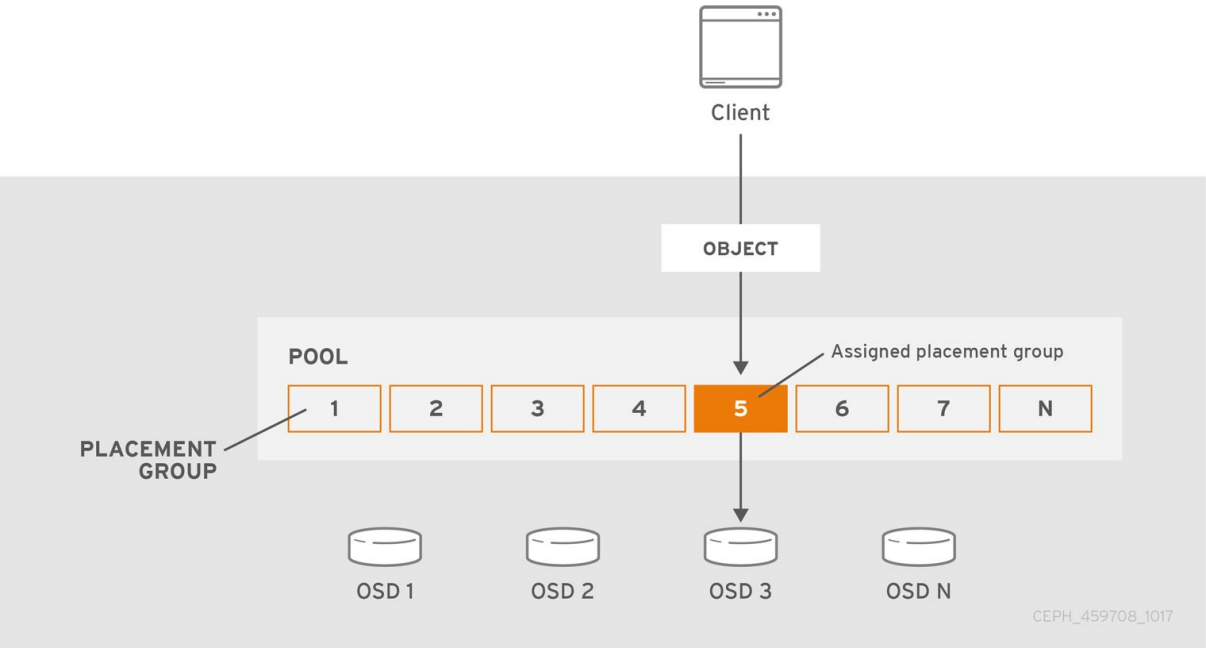
Placement groups (PGs) are invisible to Ceph clients, but they play an important role in Ceph storage clusters. Ceph assigns objects to placement groups, and placement groups to OSDs to make re-balancing dynamic and efficient.

A Ceph Storage Cluster might require many thousands of OSDs to reach an exabyte level of storage capacity. Ceph clients store objects in pools, which are a logical subset of the overall cluster. The number of objects stored in a pool might easily run into the millions and beyond. A system with millions of objects or more cannot realistically track placement on a per-object basis and still perform well. Ceph assigns objects to placement groups, and placement groups to OSDs to make re-balancing dynamic and efficient.

About placement groups

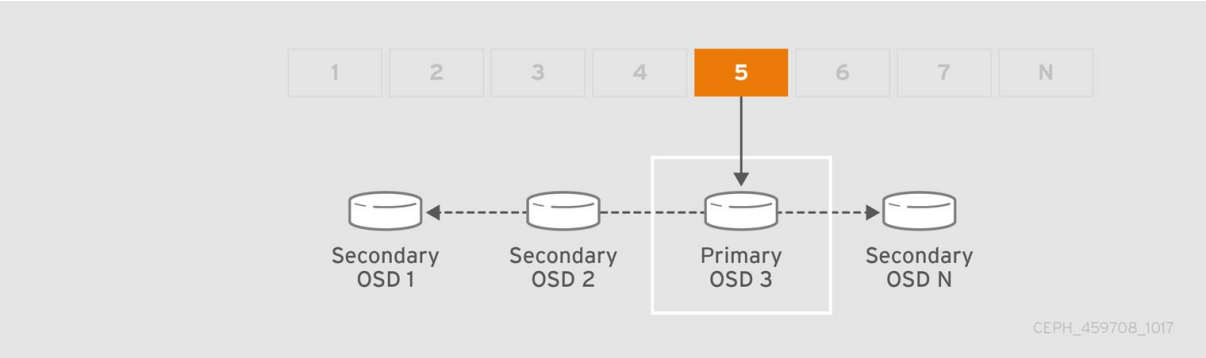
Tracking object placement on a per-object basis within a pool is computationally expensive at scale. To facilitate high performance at scale, Ceph subdivides a pool into placement groups, assigns each individual object to a placement group, and assigns the placement group to a *primary* OSD. If an OSD fails or the cluster re-balances, Ceph can move or replicate an entire placement group, meaning all of the objects in the placement groups, without having to address each object individually. This allows a Ceph cluster to re-balance or recover efficiently.

Figure: About PGs



When CRUSH assigns a placement group to an OSD, it calculates a series of OSDs—the first being the primary. The `osd_pool_default_size` setting minus `1` for replicated pools, and the number of coding chunks `M` for erasure-coded pools determine the number of OSDs storing a placement group that can fail without losing data permanently. Primary OSDs use CRUSH to identify the secondary OSDs and copy the placement group's contents to the secondary OSDs. For example, if CRUSH assigns an object to a placement group, and the placement group is assigned to OSD 5 as the primary OSD, if CRUSH calculates that OSD 1 and OSD 8 are secondary OSDs for the placement group, the primary OSD 5 will copy the data to OSDs 1 and 8. By copying data on behalf of clients, Ceph simplifies the client interface and reduces the client workload. The same process allows the Ceph cluster to recover and rebalance dynamically.

Figure: CRUSH hierarchy



When the primary OSD fails and gets marked out of the cluster, CRUSH assigns the placement group to another OSD, which receives copies of objects in the placement group. Another OSD in the `Up Set` will assume the role of the primary OSD.

When you increase the number of object replicas or coding chunks, CRUSH will assign each placement group to additional OSDs as required.

> **Note:** PGs do not own OSDs. CRUSH assigns many placement groups to each OSD pseudo-randomly to ensure that data gets distributed evenly across the cluster.

Placement group states

When you check the storage cluster's status with the `ceph -s` or `ceph -w` commands, Ceph reports on the status of the placement groups (PGs). A PG has one or more states. The optimum state for PGs in the PG map is an `active + clean` state. Use this information to get to know the different placement group states.

`activating`

The PG is peered, but not yet active.

`active`

Ceph processes requests to the PG.

`backfill_toofull`

A backfill operation is waiting because the destination OSD is over the backfillfull ratio.

`backfill_unfound`

Backfill stopped due to unfound objects.

`backfill_wait`

The PG is waiting in line to start backfill.

`backfilling`

Ceph is scanning and synchronizing the entire contents of a PG instead of inferring what contents need to be synchronized from the logs of recent operations. Backfill is a special case of recovery.

`clean`

Ceph replicated all objects in the PG accurately.

`creating`

Ceph is still creating the PG.

`deep`

Ceph is checking the PG data against stored checksums.

`degraded`

Ceph has not replicated some objects in the PG accurately yet.

`down`

A replica with necessary data is down, so the PG is offline. A PG with less than `min_size` replicas is marked as down. Use `ceph health detail` to understand the backing OSD state.

forced_backfill

High backfill priority of that PG is enforced by user.

forced_recovery

High recovery priority of that PG is enforced by user.

incomplete

Ceph detects that a PG is missing information about writes that might have occurred, or does not have any healthy copies. If you see this state, try to start any failed OSDs that might contain the needed information. In the case of an erasure coded pool, temporarily reducing `min_size` might allow recovery.

inconsistent

Ceph detects inconsistencies in one or more replicas of an object in the PG, such as objects are the wrong size, objects are missing from one replica after recovery finished.

peering

The PG is undergoing the peering process. A peering process should clear off without much delay, but if it stays and the number of PGs in a peering state does not reduce in number, the peering might be stuck.

peered

The PG has peered, but cannot serve client IO due to not having enough copies to reach the pool's configured `min_size` parameter. Recovery might occur in this state, so the PG might heal up to `min_size` eventually.

recovering

Ceph is migrating or synchronizing objects and their replicas.

recovery_toofull

A recovery operation is waiting because the destination OSD is over its full ratio.

recovery_unfound

Recovery stopped due to unfound objects.

recovery_wait

The PG is waiting in line to start recovery.

remapped

The PG is temporarily mapped to a different set of OSDs from what CRUSH specified.

repair

Ceph is checking the PG and repairing any inconsistencies it finds, if possible.

replay

The PG is waiting for clients to replay operations after an OSD crashed.

snaptrim

Trimming snaps.

snaptrim_error

Error stopped trimming snaps.

snaptrim_wait

Queued to trim snaps.

scrubbing

Ceph is checking the PG metadata for inconsistencies.

splitting

Ceph is splitting the PG into multiple PGs.

stale

The PG is in an unknown state; the monitors have not received an update for it since the PG mapping changed.

undersized

The PG has fewer copies than the configured pool replication level.

unknown

The `ceph-mgr` has not yet received any information about the PG's state from an OSD since Ceph Manager started up.

Related information

What are the possible Placement Group states in an RHCS/Ceph cluster on Red Hat Customer Portal

Placement group tradeoffs

Data durability and data distribution among all OSDs call for more placement groups but their number should be reduced to the minimum required for maximum performance to conserve CPU and memory resources.

Data durability

Ceph strives to prevent the permanent loss of data. However, after an OSD fails, the risk of permanent data loss increases until the data it had is fully recovered. Permanent data loss, though rare, is still possible.

The following scenario describes how Ceph could permanently lose data in a single placement group with three copies of the data:

- An OSD fails and all copies of the object it contains are lost. For all objects within a placement group stored on the OSD, the number of replicas suddenly drops from three to two.
- Ceph starts recovery for each placement group stored on the failed OSD by choosing a new OSD to re-create the third copy of all objects for each placement group.
- The second OSD containing a copy of the same placement group fails before the new OSD is fully populated with the third copy. Some objects will then only have one surviving copy.
- Ceph picks yet another OSD and keeps copying objects to restore the desired number of copies.
- The third OSD containing a copy of the same placement group fails before recovery is complete. If this OSD contained the only remaining copy of an object, the object is lost permanently.

Hardware failure isn't an exception, but an expectation. To prevent the foregoing scenario, ideally the recovery process should be as fast as reasonably possible. The size of your cluster, your hardware configuration and the number of placement groups play an important role in total recovery time.

Small clusters don't recover as quickly.

In a cluster containing 10 OSDs with 512 placement groups in a three replica pool, CRUSH will give each placement group three OSDs. Each OSD will end up hosting `(512 * 3) / 10 = ~150` placement groups. When the first OSD fails, the cluster will start recovery for all 150 placement groups simultaneously.

It is likely that Ceph stored the remaining 150 placement groups randomly across the 9 remaining OSDs. Therefore, each remaining OSD is likely to send copies of objects to all other OSDs and also receive some new objects, because the remaining OSDs become responsible for some of the 150 placement groups now assigned to them.

The total recovery time depends upon the hardware supporting the pool. For example, in a 10 OSD cluster, if a host contains one OSD with a 1 TB SSD, and a 10 GB/s switch connects each of the 10 hosts, the recovery time will take `M` minutes. By contrast, if a host contains two SATA OSDs and a 1 GB/s switch connects the five hosts, recovery will take substantially longer. Interestingly, in a cluster of this size, the number of placement groups has almost no influence on data durability. The placement group count could be 128 or 8192 and the recovery would not be slower or faster.

However, growing the same Ceph cluster to 20 OSDs instead of 10 OSDs is likely to speed up recovery and therefore improve data durability significantly. Why? Each OSD now participates in only 75 placement groups instead of 150. The 20 OSD cluster will still require all 19 remaining OSDs to perform the same amount of copy operations in order to recover. In the 10 OSD cluster, each OSDs had to copy approximately 100 GB. In the 20 OSD cluster each OSD only has to copy 50 GB each. If the network was the bottleneck, recovery will happen twice as fast. In other words, recovery time decreases as the number of OSDs increases.

In large clusters, PG count is important!

If the exemplary cluster grows to 40 OSDs, each OSD will only host 35 placement groups. If an OSD dies, recovery time will decrease unless another bottleneck precludes improvement. However, if this cluster grows to 200 OSDs, each OSD will only host approximately 7 placement groups. If an OSD dies, recovery will happen between at most of 21 `(7 * 3)` OSDs in these placement groups: recovery takes longer than when there were 40 OSDs, meaning the number of placement groups should be increased!

> **Important:** No matter how short the recovery time, there is a chance for another OSD storing the placement group to fail while recovery is in progress.

In the 10 OSD cluster described, if any OSD fails, then approximately 8 placement groups (that is `75 pgs / 9 osds` being recovered) will only have one surviving copy. And if any of the 8 remaining OSDs fail, the last objects of one placement group are likely to be lost (that is `8 pgs / 8 osds` with only one remaining copy being recovered). This is why starting with a somewhat larger cluster is preferred (for example, 50 OSDs).

When the size of the cluster grows to 20 OSDs, the number of placement groups damaged by the loss of three OSDs drops. The second OSD lost will degrade approximately 2 (that is `35 pgs / 19 osds` being recovered) instead of 8 and the third OSD lost will only lose data if it is one of the two OSDs containing the surviving copy. In other words, if the probability of losing one OSD is `0.0001%` during the recovery time frame, it goes from `8 * 0.0001%` in the cluster with 10 OSDs to `2 * 0.0001%` in the cluster with 20 OSDs. Having 512 or 4096 placement groups is roughly equivalent in a cluster with less than 50 OSDs as far as data durability is concerned.

> **Tip:** More OSDs means faster recovery and a lower risk of cascading failures leading to the permanent loss of a placement group and its objects.

When you add an OSD to the cluster, it might take a long time to populate the new OSD with placement groups and objects. However there is no degradation of any object and adding the OSD has no impact on data durability.

Data distribution

Ceph seeks to avoid hot spots, that is, some OSDs receive substantially more traffic than other OSDs. Ideally, CRUSH assigns objects to placement groups evenly so that when the placement groups get assigned to OSDs (also pseudo randomly), the primary OSDs store objects such that they are evenly distributed across the cluster and hot spots and network over-subscription problems cannot develop because of data distribution.

Since CRUSH computes the placement group for each object, but does not actually know how much data is stored in each OSD within this placement group, **the ratio between the number of placement groups and the number of OSDs might influence the distribution of the data significantly.**

For instance, if there was only one placement group with ten OSDs in a three replica pool, Ceph would only use three OSDs to store data because CRUSH would have no other choice. When more placement groups are available, CRUSH is more likely to evenly spread objects across OSDs. CRUSH also evenly assigns placement groups to OSDs.

As long as there are one or two orders of magnitude more placement groups than OSDs, the distribution should be even. For instance, 256 placement groups for 3 OSDs, 512 or 1024 placement groups for 10 OSDs, and so forth.

The ratio between OSDs and placement groups usually solves the problem of uneven data distribution for Ceph clients that implement advanced features like object striping. For example, a 4 TB block device might get sharded up into 4 MB objects.

The ratio between OSDs and placement groups does not address uneven data distribution in other cases, because CRUSH does not take object size into account. Using the `librados` interface to store some relatively small objects and some very large objects can lead to uneven data distribution. For example, one million 4K objects totaling 4 GB are evenly spread among 1000 placement groups on 10 OSDs. They will use `4 GB / 10 = 400 MB` on each OSD. If one 400 MB object is added to the pool, the three OSDs supporting the placement group in which the object has been placed will be filled with `400 MB + 400 MB = 800 MB` while the seven others will remain occupied with only 400 MB.

Resource usage

For each placement group, OSDs and Ceph monitors need memory, network, and CPU at all times, and even more during recovery. Sharing this overhead by clustering objects within a placement group is one of the main reasons placement groups exist.

Minimize the number of placement groups to save significant amounts of resources.

Placement group count

The number of placement groups (PGs) in a pool plays a significant role in how a cluster peers, distributes data, and rebalances.

Small clusters do not see as many performance improvements compared to large clusters by increasing the number of placement groups. However, clusters that have many pools accessing the same OSDs might need to carefully consider PG count so that Ceph OSDs use resources efficiently.

> **Note:** Use 100–200 placement group replicas per OSD.

Placement group calculator

The placement group (PG) calculator calculates the number of placement groups for you and addresses specific use cases.

The PG calculator is helpful when using Ceph clients like the Ceph Object Gateway where there are many pools typically using the same rule (CRUSH hierarchy). You might still calculate PGs manually using the guidelines in “Placement group count for small clusters” on page 58 and “Calculating placement group count” on page 58. However, the PG calculator is the preferred method of calculating PGs.

See Ceph Placement Groups (PGs) per Pool Calculator on the Red Hat Customer Portal for details.

Configuring default placement group count

When you create a pool, you also create a number of placement groups for the pool. While Ceph uses the default value of `8`, this should increased along with the default value.

If you don’t specify the number of placement groups, Ceph uses the default value of `8`, which is unacceptably low. You can increase the number of placement groups for a pool, but we recommend setting reasonable default values too.

```
osd pool default pg num = 100
osd pool default pgp num = 100
```

You need to set both the number of placement groups (total), and the number of placement groups used for objects (used in PG splitting). They should be equal.

Placement group count for small clusters

Small clusters don’t benefit from large numbers of placement groups, therefore it is important to use the PG calculator with small clusters.

As the number of OSDs increase, choosing the correct value for `pg_num` and `pgp_num` becomes more important because it has a significant influence on the behavior of the cluster as well as the durability of the data when something goes wrong (that is the probability that a catastrophic event leads to data loss).

For more information about the PG calculator, see “Placement group calculator” on page 58.

Calculating placement group count

Plan for 100–200 PG replicas per OSD in large clusters to balance durability, resource usage, and data distribution.

Clusters with more than 50 OSDs should plan for each OSD to host 100-200 placement group replicas. This balances resource usage, data durability, and uniform data distribution. If you have fewer than 50 OSDs, choosing among the PG count for small clusters as is recommended, as described in the previous section.

For a single pool of objects, you can use the following formula to get a baseline:

```
pg_num = (num_of_OSDs × ratio) / replication
ratio = (pg_num × replication) / num_of_OSDs
```

The PG ratio is the average number of PG replicas on each OSD.

Where pool size is either the number of replicas for replicated pools or the `K+M` sum for erasure coded pools (as returned by `ceph osd erasure-code-profile get`).

The result must be rounded to a power of 2.

> **Note:** When in doubt, err on the side of the higher power of 2.

The result should be rounded up to the nearest power of two. Rounding up is optional, but recommended for CRUSH to evenly balance the number of objects among placement groups.

For a cluster with 200 OSDs and a pool size of 3 replicas, you would estimate your number of PGs as follows:

```
   (200 * 200)
   ----------- = 13333
        3
```

In this example, the nearest power of 2 is 16384.

With 16384 placement groups, 49152 placement group replicas are distributed across 200 OSDs. Thus in this single-pool example, each OSD will on average host 245 placement group replicas. Consider the number of pools you are likely to use in your cluster, since additional pools will create additional placement groups too. With multiple RADOS pools, we solve for the aggregate number of placement group replicas.

The placement group calculator referenced in the above section is a valuable tool for manually calculating `pg_num` values when multiple data pools are present in a cluster, again targeting 100–200 PG replicas per OSD. Most clusters benefit by using the PG autoscaler however, which is described in the next section.

Ensure that you have a reasonable maximum placement group count. For more information, see “Maximum placement group count” on page 59

Maximum placement group count

Balance placement groups (PGs) across pools and OSDs to reduce variance and avoid resource strain or slow peering.

When using multiple data pools to store objects, ensure that the number of PGs per pool is balanced with the number of PGs per OSD. This helps maintain a reasonable total PG count and minimizes variance per OSD, avoiding excessive resource usage and slow peering.

For example, in a Red Hat Ceph Storage cluster with 10 pools, each configured with 512 PGs across 10 OSDs, the system manages 5,120 PGs total, which is 512 PGs per OSD. This configuration may be acceptable depending on your hardware. However, creating 1,000 pools with 512 PGs each results in ~50,000 PGs per OSD, which significantly increases resource demands. Excessive PGs per OSD can degrade performance, especially during rebalancing or recovery, and may lead to `00Mkilling` errors.

Red Hat Ceph Storage includes safeguards against excessive PG counts. By default, an OSD will not activate if it exceeds 250 PG replicas. You can adjust this limit using the following configuration command:

```
ceph config set mon_max_pg_per_osd 600
```

Setting the value to 600 provides a balance between protection and flexibility when OSDs fail or are added.

> **Tip:** Ceph Object Gateways deploy with 10-15 pools. To maintain a reasonable PG count, consider using fewer than 100 PGs per OSD.

Auto-scaling placement groups

Auto-scaling the number of PGs can make managing the cluster easier. The number of placement groups (PGs) in a pool plays a significant role in how a cluster peers, distributes data, and rebalances.

The auto-scaler analyzes pools and adjusts on a per-subtree basis.

Because each pool can map to a different CRUSH rule, and each rule can distribute data across different devices, Ceph considers utilization of each subtree of the hierarchy independently. For example, a pool that maps to OSDs of class `ssd`, and a pool that maps to OSDs of class `hdd`, will each have optimal PG counts that depend on the number of those respective device types.

The `pg-autoscaling` command provides recommendations for scaling PGs, or automatically scales PGs based on how the cluster is being used.

- To enable, or disable auto-scaling, see "Setting placement group auto-scaling modes" on page 62.
- To view placement group scaling recommendations, see "Viewing placement group scaling recommendations" on page 64.
- To set placement group auto-scaling, see "Configuring placement group auto-scaling" on page 65.
- To manually update autoscaler profile, see Manually updating autoscaler profile for a pool.
- To update the autoscaler globally, see "Updating noautoscale flag" on page 66.
- To set target pool size, see "Specifying target pool size" on page 67.

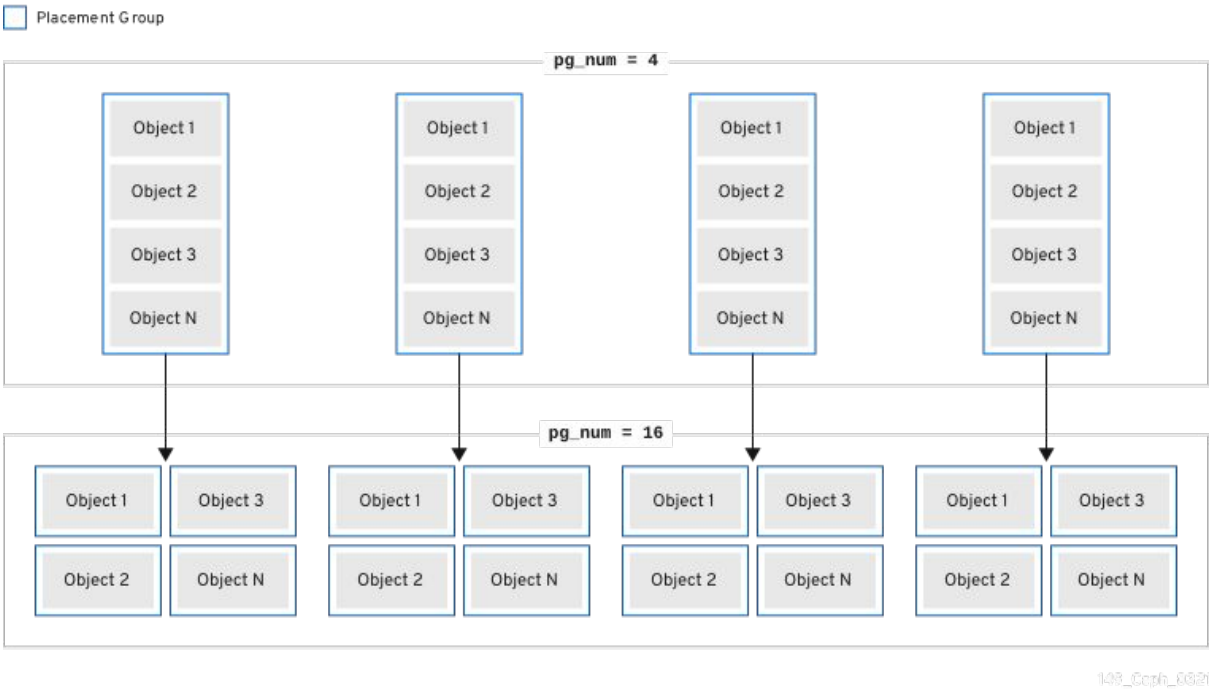
Placement group splitting and merging

Red Hat Ceph Storage can split existing placement groups (PGs) into smaller PGs, which increases the total number of PGs for a given pool. Red Hat Ceph Storage can also merge two existing PGs into a larger PG, which decreases the total number of PGs.

Splitting

Splitting existing placement groups (PGs) allows a small Red Hat Ceph Storage cluster to scale over time as storage requirements increase. The PG auto-scaling feature can increase the `pg_num` value, which causes the existing PGs to split as the storage cluster expands. If the PG auto-scaling feature is disabled, then you can manually increase the `pg_num` value, which triggers the PG split process to begin. For example, increasing the `pg_num` value from `4` to `16`, will split into four pieces. Increasing the `pg_num` value will also increase the `pgp_num` value, but the `pgp_num` value increases at a gradual rate. This gradual increase is done to minimize the impact to a storage cluster's performance and to a client's workload, because migrating object data adds a significant load to the system. By default, Ceph queues and moves no more than 5% of the object data that is in a "misplaced" state. This default percentage can be adjusted with the `target_max_misplaced_ratio` option.

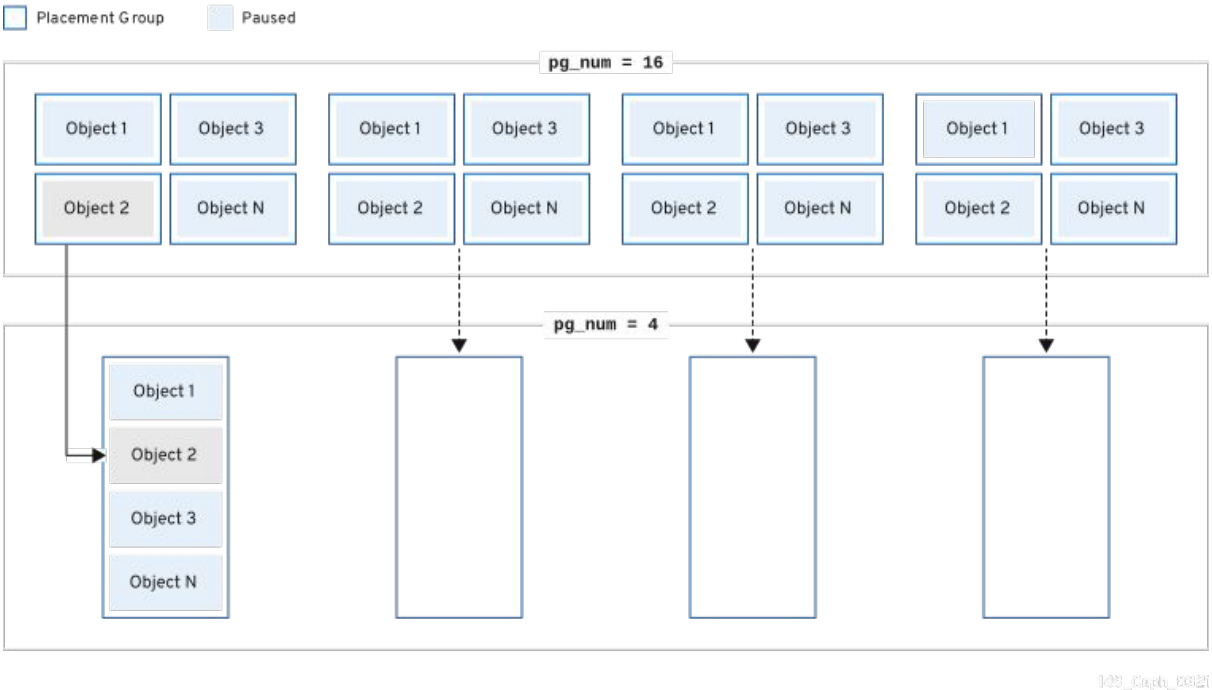
Figure: Splitting



Merging

Merging two PGs together can be useful, especially when the relative amount of objects in a pool decreases over time, or when the initial number of PGs chosen was too large. While merging PGs can be useful, it is also a complex and delicate process. When doing a merge, pausing I/O to the PG occurs, and only one PG is merged at a time to minimize the impact to a storage cluster's performance. Ceph works slowly on merging the object data until the new `pg_num` value is reached.

Figure: Merging



Setting placement group auto-scaling modes

Each pool in the Red Hat Ceph Storage cluster has a `pg_autoscale_mode` property for PGs that you can set to `off`, `on`, or `warn`.

- `off`: Disables auto-scaling for the pool. It is up to the administrator to choose an appropriate PG number for each pool. For more information, see "Placement group count" on page 58.
- `on`: Enables automated adjustments of the PG count for the given pool.
- `warn`: Raises health alerts when the PG count needs adjustment.

> **Note:** `pg_autoscale_mode` is on by default. Upgraded storage clusters retain the existing `pg_autoscale_mode` setting. The `pg_auto_scale` mode is `on` for the newly created pools. PG count is automatically adjusted, and `ceph status` might display a recovering state during PG count adjustment.

The autoscaler uses the `bulk` flag to determine which pool should start with a full complement of PGs and only scales down when the usage ratio across the pool is not even. However, if the pool does not have the `bulk` flag, the pool starts with minimal PGs and only when there is more usage in the pool.

> **Note:** The autoscaler identifies any overlapping roots and prevents the pools with such roots from scaling because overlapping roots can cause problems with the scaling process.

Procedure

- Enable auto-scaling on an existing pool:

  Syntax

  ```
  ceph osd pool set POOL_NAME pg_autoscale_mode on
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph osd pool set testpool pg_autoscale_mode on
  ```

- Enable auto-scaling on a newly created pool:

  Syntax

  ```
  ceph config set global osd_pool_default_pg_autoscale_mode MODE
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph config set global osd_pool_default_pg_autoscale_mode on
  ```

- Create a pool with the `bulk` flag:

  Syntax

  ```
  ceph osd pool create POOL_NAME [--bulk]
  ```

  Example

  ```
  [ceph: root@host01 /]#  ceph osd pool create testpool --bulk
  ```

- Set or unset the `bulk` flag for an existing pool:

Important: The values must be written as `true`, `false`, `1`, or `0`. `1` is equivalent to `true` and `0` is equivalent to `false`. If written with different capitalization, or with other content, an error is emitted. The following is an example of the command written with the wrong syntax:

```
[ceph: root@host01 /]# ceph osd pool set ec_pool_overwrite bulk
TrueError EINVAL: expecting value 'true', 'false', '0', or '1'
```

Syntax

```
ceph osd pool set POOL_NAME bulk true/false/1/0
```

Example

```
[ceph: root@host01 /]#  ceph osd pool set testpool bulk true
```

- Get the `bulk` flag of an existing pool:

  Syntax

  ```
  ceph osd pool get POOL_NAME bulk
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph osd pool get testpool bulk
bulk: true
  ```

Setting minimum and maximum number of placement groups for pools

Specify the minimum and maximum value of placement groups (PGs) in order to limit the auto-scaling range.

If a minimum value is set, Ceph does not automatically reduce, or recommend to reduce, the number of PGs to a value below the set minimum value.

If a minimum value is set, Ceph does not automatically increase, or recommend to increase, the number of PGs to a value above the set maximum value.

The minimum and maximum values can be set together or separately.

In addition to the this procedure, the **`ceph osd pool create`** command has two command-line options that can be used to specify the minimum or maximum PG count at the time of pool creation.

```
ceph osd pool create --pg-num-min NUMBER
ceph osd pool create --pg-num-max NUMBER
```

For example:

```
ceph osd pool create --pg-num-min 50
ceph osd pool create --pg-num-max 150
```

Prerequisites

- A running Red Hat Ceph Storage cluster
- Root-level access to the node

Procedure

- Set the minimum number of PGs for a pool.

  Syntax

  ```
  ceph osd pool set POOL_NAME pg_num_min NUMBER
  ```

Example

```
[ceph: root@host01 /]# ceph osd pool set testpool pg_num_min 50
```

- Set the maximum number of PGs for a pool.

  Syntax

  ```
  ceph osd pool set POOL_NAME pg_num_max NUMBER
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph osd pool set testpool pg_num_max 150
  ```

Reference

For more information, see:

- Setting placement group auto-scaling modes
- Placement Group count

Viewing placement group scaling recommendations

You can view the pool, its relative utilization, and any suggested changes to the placement group (PG) count within the storage cluster.

Before you begin, make sure that you have the following prerequisites in place:

- A running Red Hat Ceph Storage cluster.
- A `cephadm` shell or shell on a node with the cluster's admin key.

View each pool, its relative utilization, and any suggested changes to the PG count by using the **`pool autoscale-status`** command.

```
ceph osd pool autoscale-status
```

The command output looks similar to the following:

```
[ceph: root@host01 /]# ceph osd pool autoscale-status
OOL                     SIZE  TARGET SIZE  RATE  RAW CAPACITY   RATIO  TARGET RATIO
 EFFECTIVE RATIO  BIAS  PG_NUM  NEW PG_NUM  AUTOSCALE  BULK
  device_health_metrics      0                 3.0        374.9G  0.0000
                1.0       1              on         False
  cephfs.cephfs.meta     24632                 3.0        374.9G  0.0000
                4.0      32              on         False
  cephfs.cephfs.data         0                 3.0        374.9G  0.0000
                1.0      32              on         False
  .rgw.root                1323                 3.0        374.9G  0.0000
                1.0      32              on         False
  default.rgw.log          3702                 3.0        374.9G  0.0000
                1.0      32              on         False
  default.rgw.control        0                 3.0        374.9G  0.0000
                1.0      32              on         False
  default.rgw.meta         382                 3.0        374.9G  0.0000
                4.0       8              on         False
```

See the following output definitions for understanding the scaling status.

SIZE

The amount of data stored in the pool.

TARGET SIZE

If present, the amount of data the administrator expects to eventually be stored in this pool. The system uses the larger of the two values for its calculation.

RATE

The multiplier for the pool that determines how much raw storage capacity the pool uses. For example, a 3-replica pool has a ratio of 3.0, while a k=4,m=2 erasure coded pool has a ratio of 1.5.

RAW CAPACITY

The total amount of raw storage capacity on the OSDs responsible for storing the pool's data.

RATIO

The ratio of the total capacity that the pool is consuming, calculated as:
ratio = size × rate ÷ raw capacity

TARGET RATIO

If present, the ratio of storage the administrator expects the pool to consume relative to other pools with target ratios set. If both target size bytes and ratio are specified, the ratio takes precedence. The default value is 0 unless specified during pool creation. The higher the `--target_ratio`, the larger the PGs expected for the pool.

EFFECTIVE RATIO

The target ratio after adjusting in two ways:

1. Subtracting capacity expected to be used by pools with target size set.
2. Normalizing target ratios among pools with target ratio set so they collectively target the remaining space.

For example, four pools with target ratio 1.0 would each have an effective ratio of 0.25. The system uses the larger of the actual ratio and the effective ratio for its calculation.

BIAS

A multiplier to manually adjust a pool's PG count based on prior expectations. The default value is 1.0 unless specified during pool creation. The higher the `--bias`, the larger the PGs expected for the pool.

PG_NUM

The current number of PGs for the pool, or the number the pool is working towards if a `pg_num` change is in progress.

NEW PG_NUM

If present, the suggested number of PGs (`pg_num`). It is always a power of 2 and only appears if the suggested value differs from the current value by more than a factor of 3.

AUTOSCALE

The pool's `pg_autoscale_mode`, which can be `on`, `off`, or `warn`.

BULK

Determines which pool should start with a full complement of PGs. `BULK` only scales down when usage ratio across the pool is uneven. Pools without this flag start with minimal PGs and scale up with increased usage. Valid values are `true`, `false`, `1`, or `0`, where `1` is equivalent to `true`. The default is `false`. Set during or after pool creation.

For more information on using the `bulk` flag, see "Creating pools" on page 71 and "Configuring placement group auto-scaling" on page 65.

Configuring placement group auto-scaling

Allowing the cluster to automatically scale placement groups (PGs) based on cluster usage is the simplest approach to scaling PGs.

Red Hat Ceph Storage takes the total available storage and the target number of PGs for the whole system, compares how much data is stored in each pool, and apportions the PGs accordingly.

Configuring placement group autoscaling solves for the number of PG replicas per OSD.

> **Note:** This is not the same as dividing the pool's `pg_num` value by the number of OSDs.

A replicated `size=3` pool has 3×`pg_num` PG replicas.

An erasure-coded pool has (k + m)×`pg_num` replicas.

Each OSD's current number of PG replicas, the PG ratio, can be seen by using the **ceph osd df** command.

In the following example, `osd.217` holds 127 PG replicas.

```
[ceph: root@host01 /]# ceph osd df | head
ID  CLASS WEIGHT    REWEIGHT SIZE   RAW    USE    DATA   OMAP   META    AVAIL %USE VAR PGS
STATUS
217 hdd   18.53969  1.00000  19 TiB 15 TiB 14 TiB 21 KiB 53 GiB 3.9 TiB 79.20 1.01 127 up
```

The command only makes changes to a pool whose current number of PGs (`pg_num`) is more than three times off from the calculated or suggested PG number. This threshold can be modified by adjusting the value at pool granularity.

```
ceph osd pool set POOL threshold THRESHOLD
```

The following example adjusts the threshold for a pool named `pool01` to 2.5.

```
[ceph: root@host01 /]# ceph osd pool set pool01 threshold 2.5
```

The target number of PG replicas per OSD is determined by the **mon_target_pg_per_osd** central configuration option. The default value is `100` but most clusters benefit from setting to 250.

```
ceph config set global mon_target_pg_per_osd VALUE
```

For example,

```
[ceph: root@host01 /]# ceph config set global mon_target_pg_per_osd 250
```

> **Important:** When raising the target number of PG replicas per OSD, it is important to also raise the central configuration option **mon_max_pg_per_osd**. This value is a failsafe that guards against accidents. The default value is 250, but it is recommended to raise the value to 600.
>
> ```
> ceph config set global mon_max_pg_per_osd 600
> ```

Updating `noautoscale` flag

If you want to enable or disable the autoscaler for all the pools at same time, you can use the `noautoscale` global flag. This global flag is useful during upgradation of the storage cluster when some OSDs are bounced or when the cluster is under maintenance. You can set the flag before any activity and unset it once the activity is complete.

By default, the `noautoscale` flag is set to `off`. When this flag is set, then all the pools have `pg_autoscale_mode` as `off` and all the pools have the autoscaler disabled.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to all the nodes.

Procedure

1. Get the value of the `noautoscale` flag:

   Example

   ```
   [ceph: root@host01 /]# ceph osd pool get noautoscale
   ```

2. Set the `noautoscale` flag before any activity:

   Example

   ```
   [ceph: root@host01 /]# ceph osd pool set noautoscale
   ```

3. Unset the `noautoscale` flag on completion of the activity:

   Example

   ```
   [ceph: root@host01 /]# ceph osd pool unset noautoscale
   ```

Specifying target pool size

Specify target pool size.

A newly created pool consumes a small fraction of the total cluster capacity and appears to the system that it will need a small number of PGs. However, in most cases, cluster administrators know which pools are expected to consume most of the system capacity over time. If you provide this information, known as the `target size` to Red Hat Ceph Storage, such pools can use a more appropriate number of PGs (`pg_num`) from the beginning. This approach prevents subsequent changes in `pg_num` and the overhead associated with moving data around when making those adjustments.

You can specify target size of a pool by either using the absolute size of the pool or the total cluster capacity.

Specifying target size using the absolute size of the pool

Specify the target size of a pool by using the absolute size of the pool.

1. Set the `target size` using the absolute size of the pool in bytes:

   ```
   ceph osd pool set pool-name target_size_bytes value
   ```

   For example, to instruct the system that `mypool` is expected to consume 100T of space:

   ```
   $ ceph osd pool set mypool target_size_bytes 100T
   ```

You can also set the target size of a pool at creation time by adding the optional `--target-size-bytes <bytes>` argument to the **`ceph osd pool create`** command.

Specifying target size using the total cluster capacity

Specify the target size of a pool by using the total cluster capacity.

1. Set the `target size` using the ratio of the total cluster capacity:

   Syntax

   ```
   ceph osd pool set pool-name target_size_ratio ratio
   ```

   Example

   ```
   [ceph: root@host01 /]# ceph osd pool set mypool target_size_ratio 1.0
   ```

   tells the system that the pool `mypool` is expected to consume 1.0 relative to the other pools with `target_size_ratio` set. If `mypool` is the only pool in the cluster, this means an expected use of 100% of the total capacity. If there is a second pool with `target_size_ratio` as 1.0, both pools would expect to use 50% of the cluster capacity.

You can also set the target size of a pool at creation time by adding the optional `--target-size-ratio <ratio>` argument to the `ceph osd pool create` command.

NOTE If you specify impossible target size values, for example, a capacity larger than the total cluster, or ratios that sum to more than 1.0, the cluster raises a POOL_TARGET_SIZE_RATIO_OVERCOMMITTED or POOL_TARGET_SIZE_BYTES_OVERCOMMITTED health warning. If you specify both target_size_ratio and target_size_bytes for a pool, the cluster considers only the ratio, and raises a POOL_HAS_TARGET_SIZE_BYTES_AND_RATIO health warning.

Placement group command line interface

The `ceph` CLI allows you to set and get the number of placement groups for a pool, view the PG map and retrieve PG statistics.

Setting number of placement groups in a pool

To set the number of placement groups in a pool, you must specify the number of placement groups at the time you create the pool.

Once you set placement groups for a pool, you can increase or decrease the number of placement groups. For more information, see “Creating pools” on page 71.

To change the number of placement groups, use the following command:

Syntax

```
ceph osd pool set POOL_NAME pg_num PG_NUM
```

Example

```
[ceph: root@host01 /]# ceph osd pool set pool1 pg_num 60
set pool 2 pg_num to 60
```

Once you increase or decrease the number of placement groups, you must also adjust the number of placement groups for placement (`pgp_num`) before your cluster rebalances. The `pgp_num` should be equal to the `pg_num`. To increase the number of placement groups for placement, run the following:

Syntax

```
ceph osd pool set POOL_NAME pgp_num PGP_NUM
```

Example

```
[ceph: root@host01 /]# ceph osd pool set pool1 pgp_num 60
set pool 2 pgp_num to 60
```

Getting number of placement groups in a pool

Learn how to get the number of placement groups in a pool.

To get the number of placement groups in a pool, run the following:

Syntax

```
ceph osd pool get POOL_NAME pg_num
```

Example

```
[ceph: root@host01 /]# ceph osd pool get testpool 60
```

Getting statistics for placement groups

Learn how to get the statistics for the placement groups in a storage cluster.

To get the statistics for the placement groups in your storage cluster, run the following:

Syntax

```
ceph pg dump [--format FORMAT]
```

Valid formats are `plain` (default) and `json`.

Getting statistics for stuck placement groups

Learn how to get the statistics for all placements groups stuck in a specific state.

To get the statistics for all placement groups stuck in a specified state, run the following:

Syntax

```
ceph pg dump_stuck {inactive|unclean|stale|undersized|degraded [inactive|unclean|stale|
undersized|degraded...]} INTEGER
```

Inactive Placement groups cannot process reads or writes because they are waiting for an OSD with the most up-to-date data to come up and in.

Unclean Placement groups contain objects that are not replicated the desired number of times. They should be recovering.

Stale Placement groups are in an unknown state - the OSDs that host them have not reported to the monitor cluster in a while (configured by `mon_osd_report_timeout`).

Valid formats are `plain` (default) and `json`. The threshold defines the minimum number of seconds the placement group is stuck before including it in the returned statistics (default 300 seconds).

Getting placement group maps

Learn how to get the placement group map for a particular placement group.

To get the placement group map for a particular placement group, run the following:

Syntax

```
ceph pg map PG_ID
```

Example

```
[ceph: root@host01 /]# ceph pg map 1.6c
osdmap e13 pg 1.6c (1.6c) -> up [1,0] acting [1,0]
```

Ceph returns the placement group map, the placement group, and the OSD status:

Getting a placement group statistics

Learn how to get statistics for a particular placement group.

Retrieve statistics for a particular placement group:

Syntax

```
ceph pg PG_ID query
```

Scrubbing placement groups

Learn how to scrub a placement group.

To scrub a placement group, run the following:

Syntax

```
ceph pg scrub PG_ID
```

Ceph checks the primary and any replica nodes, generates a catalog of all objects in the placement group and compares them to ensure that no objects are missing or mismatched, and their contents are consistent. Assuming the replicas all match, a final semantic sweep ensures that all of the snapshot-related object metadata is consistent. Errors are reported via logs.

Marking `unfound` objects

If the cluster has lost one or more objects, and you have decided to abandon the search for the lost data, you must mark the unfound objects as `lost`.

If all possible locations have been queried and objects are still lost, you might have to give up on the lost objects. This is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered.

Currently the only supported option is "revert", which will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. To mark the "unfound" objects as "lost", run the following:

Syntax

```
ceph pg PG_ID mark_unfound_lost revert|delete
```

> **Important:** Use this feature with caution, because it might confuse applications that expect the object(s) to exist.

Pools overview

Ceph clients store data in pools. When you create pools, you create an I/O interface for clients to store data.

From a Ceph client perspective, (that is, block device, gateway, and the rest), interacting with the Ceph storage cluster is remarkably simple where you perform the following actions:

- Create a cluster handle.
- Connect the cluster handle to the cluster.
- Create an I/O context for reading and writing objects and their extended attributes.

Creating a cluster handle and connecting to the cluster

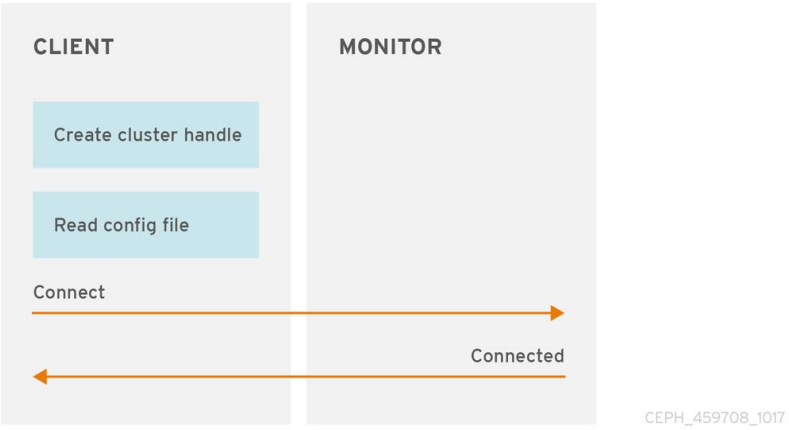
To connect to the Ceph storage cluster, the Ceph client needs the following details:

- The cluster name (which Ceph by default) - not using usually because it sounds ambiguous.
- An initial monitor address.

Ceph clients usually retrieve these parameters that use the default path for the Ceph configuration file and then read it from the file. You can also specify the parameters on the command line interface.

The Ceph client also provides a username and secret key (authentication is `on` by default) for the clients to contact the Ceph monitor cluster and retrieve a recent copy of the cluster map, including its monitors, OSDs, and pools.

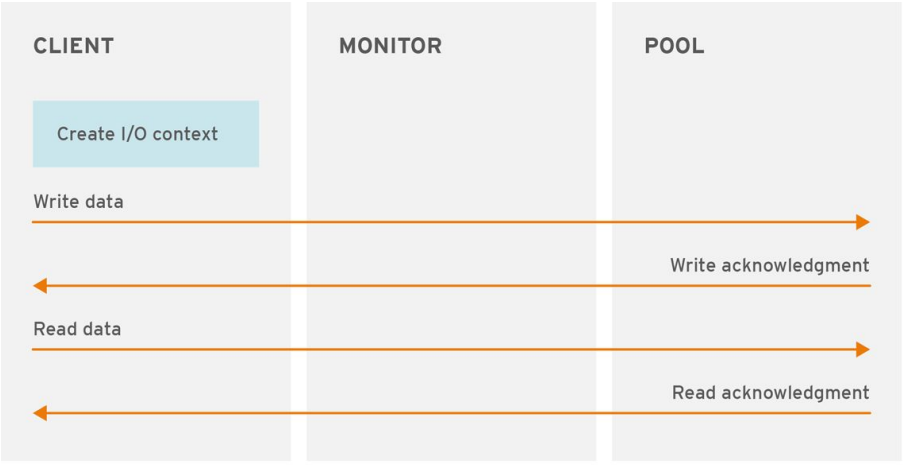
Figure: Create handle



Creating a pool I/O context

To read and write data, the Ceph client creates an I/O context to a specific pool in the Ceph storage cluster. If the specified user has the permission to the pool, the Ceph client can read from and write to the specified pool.

Figure: I/O Context



The architecture of Ceph enables the storage cluster to provide a simple interface to Ceph clients so that clients can select one of the storage strategies you define simply by specifying a pool name and creating an I/O context. Storage strategies are invisible to the Ceph client in all but capacity and performance. Similarly, the complexities of Ceph clients (mapping objects into a block device representation, providing an S3/Swift RESTful service) are invisible to the Ceph storage cluster.

A pool provides you with resilience, placement groups, CRUSH rules, and quotas.

Resilience

Set how many OSD are allowed to fail without losing data. For replicated pools, it is the wanted number of copies/replicas of an object. A typical configuration stores an object and one more copy (that is, `size = 2`), but you can determine the number of copies/replicas. For erasure coded pools, it is the number of coding chunks (that is `m=2` in the erasure code profile).

Placement groups

Set the number of placement groups for the pool. A typical configuration uses approximately 50-100 placement groups per OSD to provide optimal balancing without using up too many computing resources. When setting up multiple pools, ensure that you set a reasonable number of placement groups for both the pool and the cluster as a whole.

CRUSH rules

When you store data in a pool, a CRUSH rule mapped to the pool enables CRUSH to identify the rule for the placement of each object and its replicas or chunks for erasure coded pools in your cluster. You can create a custom CRUSH rule for your pool.

Quotas

When you set quotas on a pool with the **`ceph osd pool set-quota`** command you can limit the maximum number of objects or the maximum number of bytes stored in the specified pool.

You can list, create, and remove pools. You can also view the utilization statistics for each pool.

Listing pool

Learn how to list pools in a cluster.

To list your cluster's pools, run:

```
ceph osd lspools
```

Creating pools

Learn how to create a pool. Pools can be replicated, erasure-coded, or as bulk.

PREREQUISITE

See Pools, placement groups, and CRUSH configuration.

Note: The system administrators must expressly enable a pool to receive I/O operations from Ceph clients. See "Enabling a client application" on page 76 for details. Failure to enable a pool results in a `HEALTH_WARN` status.

When you create a pool, set the number of placement groups to a reasonable value (for example to 100). Consider the total number of placement groups per OSD too. Placement groups are computationally expensive, so performance degrades when you have many pools with many placement groups, for example, 50 pools with 100 placement groups each. The point of diminishing returns depends upon the power of the OSD host.

Based on the requirement, you can create a replicated pool, an erasure-coded pool, or a bulk pool.

- "Creating a replicated pool" on page 74
- "Creating an erasure-coded pool" on page 74
- "Creating a bulk pool" on page 74

In the examples, replace the variable names as follows:

POOL_NAME

Description

The name of the pool. It must be unique.

Type

String

Required

Yes. If not specified, it is set to the default value or manually set with the **`ceph config set`** command.

Default

`ceph`

PG_NUM

Description

The total number of placement groups for the pool. For more information about calculating a suitable number, see "Placement groups" on page 53 and Ceph Placement Groups (PGs) per Pool Calculator on the Red Hat Customer Portal. The default value `8` is not suitable for most systems.

Type

Integer

Required

Yes

Default

`8`

PGP_NUM

Description

The total number of placement groups for placement purposes. This value must be equal to the total number of placement groups, except for placement group splitting scenarios.

Type

Integer

Required

Yes. If not specified it is set to the value listed or to the default value.

Default

8

replicated* or *erasure

Description

The pool type can be either `replicated` to recover from lost OSDs by keeping multiple copies of the objects or `erasure` to get a generalized RAID5 capability. The replicated pools require more raw storage but implement all Ceph operations. The erasure-coded pools require less raw storage but implement only a subset of the available operations.

Type

String

Required

No

Default

`replicated`

crush-rule-name

Description

The name of the crush rule for the pool. The rule must exist. For replicated pools, the name is the rule that is specified by the `osd_pool_default_crush_rule` configuration setting. For erasure-coded pools the name is `erasure-code` if you specify the default erasure code profile or *POOL_NAME* otherwise. Ceph creates this rule with the specified name implicitly if the rule doesn't exist.

Type

String

Required

No

Default

Uses `erasure-code` for an erasure-coded pool. For replicated pools, it uses the value of the `osd_pool_default_crush_rule` variable from the Ceph configuration.

expected-num-objects

Description

The expected number of objects for the pool. Ceph splits the placement groups at pool creation time to avoid the latency impact to perform runtime directory splitting.

Type

Integer

Required

No

Default

`0`, no splitting at the pool creation time.

erasure-code-profile

Description

For erasure-coded pools only. Use the erasure code profile. It must be an existing profile as defined by the `osd erasure-code-profile set` variable in the Ceph configuration file. For more information, see "Erasure code profiles" on page 86.

Type

String

Required

No

Creating a replicated pool

- To create a replicated pool, use the following command:

```
ceph osd pool create POOL_NAME PG_NUM PGP_NUM [replicated] [CRUSH_RULE_NAME]
[EXPECTED_NUMBER_OBJECTS]
```

Creating an erasure-coded pool

- To create an erasure-coded pool, use the following command:

```
ceph osd pool create POOL_NAME PG_NUM PGP_NUM [erasure] ERASURE_CODE_PROFILE
         [CRUSH_RULE_NAME] [EXPECTED_NUMBER_OBJECTS]
```

Creating a bulk pool

- To create bulk pool, use the following command:

```
ceph osd pool create POOL_NAME [--bulk]
```

Setting pool quota

You can set pool quotas for the maximum number of bytes or the maximum number of objects per pool or for both.

Syntax

```
ceph osd pool set-quota POOL_NAME [max_objects OBJECT_COUNT>] [max_bytes BYTES]
```

Example

```
[ceph: root@host01 /]# ceph osd pool set-quota data max_objects 10000
```

To remove a quota, set its value to `0`.

> **Note:** In-flight write operations might overrun pool quotas for a short time until Ceph propagates the pool usage across the cluster. This is normal behavior. Enforcing pool quotas on in-flight write operations would impose significant performance penalties.

Deleting a pool

Learn how to delete a pool.

To delete a pool, run:

Syntax

```
ceph osd pool delete POOL_NAME [POOL_NAME --yes-i-really-really-mean-it]
```

> **Important:** To protect data, storage administrators cannot delete pools by default. Set the `mon_allow_pool_delete` configuration option before deleting pools.

If a pool has its own rule, consider removing it after deleting the pool. If a pool has users strictly for its own use, consider deleting those users after deleting the pool.

Renaming a pool

Learn how to rename a pool.

To rename a pool, run:

Syntax

```
ceph osd pool rename CURRENT_POOL_NAME NEW_POOL_NAME
```

If you rename a pool and you have per-pool capabilities for an authenticated user, you must update the user's capabilities (that is, caps) with the new pool name.

Migrating a pool

Sometimes it is necessary to migrate all objects from one pool to another. This is done in cases such as needing to change parameters that cannot be modified on a specific pool. For example, needing to reduce the number of placement groups of a pool.

PREREQUISITE

- If using the **`rados cppool`** command:
  - Read-only access to the pool is required.
  - Only use this command if you do not have RBD images and its snaps and `user_version` consumed by librados.
- If using the local drive RADOS commands, verify that sufficient cluster space is available. Two, three, or more copies of data will be present as per pool replication factor.

> **Important:** When a workload is using *only* Ceph Block Device images, use the following procedures:
> - "Moving images between pools" on page 0
> - "Migrating pools" on page 0

The migration methods described for Ceph Block Device are more recommended than those documented here. using the cppool does not preserve all snapshots and snapshot related metadata, resulting in an unfaithful copy of the data. For example, copying an RBD pool does not completely copy the image. In this case, snaps are not present and will not work properly. The cppool also does not preserve the `user_version` field that some librados users may rely on.

If migrating a pool is necessary and your user workloads contain images other than Ceph Block Devices, continue with one of the procedures documented here.

Migrating directly

Copy all objects with the **`rados cppool`** command.

> **Important:** Read-only access to the pool is required during copy.

```
ceph osd pool create NEW_POOL PG_NUM [ <other new pool parameters> ]
rados cppool SOURCE_POOL NEW_POOL
ceph osd pool rename SOURCE_POOL NEW_SOURCE_POOL_NAME
ceph osd pool rename NEW_POOL SOURCE_POOL
```

For example,

```
[ceph: root@host01 /]# ceph osd pool create pool1 250
[ceph: root@host01 /]# rados cppool pool2 pool1
[ceph: root@host01 /]# ceph osd pool rename pool2 pool3
[ceph: root@host01 /]# ceph osd pool rename pool1 pool2
```

Migrating through a local drive

1. Use the **`rados export`** and **`rados import`** commands and a temporary local directory to save all exported data.

```
ceph osd pool create NEW_POOL PG_NUM [ <other new pool parameters> ]
rados export --create SOURCE_POOL FILE_PATH
rados import FILE_PATH NEW_POOL
```

For example,

```
[ceph: root@host01 /]# ceph osd pool create pool1 250
[ceph: root@host01 /]# rados export --create pool2 <path of export file>
[ceph: root@host01 /]# rados import <path of export file> pool1
```

2. Stop all I/O to the source pool.
3. Resynchronize all modified objects.

```
rados export --workers 5 SOURCE_POOL FILE_PATH
rados import --workers 5 FILE_PATH NEW_POOL
```

For example,

```
[ceph: root@host01 /]# rados export --workers 5 pool2 <path of export file>
[ceph: root@host01 /]# rados import --workers 5 <path of export file> pool1
```

Viewing pool statistics

Learn how to view a pool's utilization statistics.

To show a pool's utilization statistics, run the following command:

```
rados df
```

Setting pool values

Set pool values.

To set a value to a pool, run the following command:

```
ceph osd pool set POOL_NAME KEY VALUE
```

For more information about the available key-value pairs, see "Pool values" on page 79.

Getting pool values

Get a value from a pool.

To get a value from a pool, run the following command:

```
ceph osd pool get POOL_NAME KEY
```

For more information about the available key-value pairs, see "Pool values" on page 79.

Enabling a client application

Red Hat Ceph Storage provides additional protection for pools to prevent unauthorized types of clients from writing data to the pool. This means that system administrators must expressly enable pools to receive I/O operations from Ceph Block Device, Ceph Object Gateway, Ceph Filesystem or for a custom application.

To enable a client application to conduct I/O operations on a pool, execute the following:

Syntax

```
ceph osd pool application enable POOL_NAME APP {--yes-i-really-mean-it}
```

Where `APP` is:

- `cephfs` for the Ceph Filesystem.

- `rbd` for the Ceph Block Device
- `rgw` for the Ceph Object Gateway

Important: A pool that is not enabled will generate a `HEALTH_WARN` status.
In that scenario, the output for `ceph health detail -f json-pretty` gives the following output:

```
{
    "checks": {
        "POOL_APP_NOT_ENABLED": {
            "severity": "HEALTH_WARN",
            "summary": {
                "message": "application not enabled on 1 pool(s)"
            },
            "detail": [
                {
                    "message": "application not enabled on pool 'POOL_NAME'"
                },
                {
                    "message": "use 'ceph osd pool application enable POOL_NAME APP',
where APP is 'cephfs', 'rbd', 'rgw', or freeform for custom applications."
                }
            ]
        }
    },
    "status": "HEALTH_WARN",
    "overall_status": "HEALTH_WARN",
    "detail": [
        "'ceph health' JSON format has changed in luminous. If you see this your
monitoring system is scraping the wrong fields. Disable this with 'mon health
preluminous compat warning = false'"
    ]
}
```

Note:

- Specify a different `APP` value for a custom application.
- Initialize pools for the Ceph Block Device with `rbd pool init POOL_NAME`.

Disabling a client application

To disable a client application from conducting I/O operations on a pool, run the following:

Syntax

```
ceph osd pool application disable POOL_NAME APP {--yes-i-really-mean-it}
```

Where `APP` is:

- `cephfs` for the Ceph Filesystem.
- `rbd` for the Ceph Block Device
- `rgw` for the Ceph Object Gateway

Note: Specify a different `APP` value for a custom application.

Setting application metadata

Set client application metadata on a pool to provide the functionality to set key-value pairs describing attributes of the client application.

To set client application metadata on a pool, run the following:

```
ceph osd pool application set POOL_NAME APP KEY VALUE
```

Replace *APP* with one of the following:

- `cephfs` for Ceph File System.
- `rbd` for Ceph Block Device
- `rgw` for Ceph Object Gateway

Note:

- Specify a different *APP* value for a custom application.
- This command is different than the **ceph osd pool application enable POOL APP** command.

Removing application metadata

To remove client application metadata on a pool, run the following:

Syntax

```
ceph osd pool application rm POOL_NAME APP KEY
```

Where `APP` is:

- `cephfs` for the Ceph Filesystem.
- `rbd` for the Ceph Block Device
- `rgw` for the Ceph Object Gateway

Note: Specify a different `APP` value for a custom application.

Setting the number of object replicas

Learn how to set the number of object replicas.

To set the number of object replicas on a replicated pool, run the following command:

Syntax

```
ceph osd pool set POOL_NAME size NUMBER_OF_REPLICAS
```

You can run this command for each pool.

- The **NUMBER_OF_REPLICAS** parameter includes the object itself. If you want to include the object and two copies of the object for a total of three instances of the object, specify `3`.
  Example

  ```
  [ceph: root@host01 /]# ceph osd pool set data size 3
  ```

- An object might accept I/O operations in degraded mode with fewer replicas than specified by the `pool size` setting. To set a minimum number of required replicas for I/O, use the `min_size` setting.
  Example

  ```
  ceph osd pool set data min_size 2
  ```

  This ensures that no object in the data pool will receive I/O with fewer replicas than specified by the `min_size` setting.

Getting the number of object replicas

Learn how to get the number of object replicas.

To get the number of object replicas, run the following command:

```
ceph osd dump | grep 'replicated size'
```

Ceph will list the pools, with the `replicated size` attribute highlighted. By default, Ceph creates two replicas of an object, that is a total of three copies, or a size of 3.

Pool values

This information contains key-values pairs that you can set or get.

For further information, see "Setting pool values" on page 76 and "Getting pool values" on page 76.

Pool values				
Value	**Description**	**Type**	**Required**	**Default**
size	Specifies the number of replicas for objects in the pool. See the "Setting the number of object replicas" on page 78 section for further details. Applicable for the replicated pools only.	Integer	No	None
min_size	Specifies the minimum number of replicas required for I/O. See the "Setting the number of object replicas" on page 78 section for further details. For erasure-coded pools, this should be set to a value greater than `k`. If I/O is allowed at the value `k`, then there is no redundancy and data is lost in the event of a permanent OSD failure. For more information, see "Erasure code pools overview" on page 85.	Integer	No	None
crash_replay_interval	Specifies the number of seconds to allow clients to replay acknowledged, but uncommitted requests.	Integer	No	None

Value	Description	Type	Required	Default
pg_num	The total number of placement groups for the pool. See the Pools, placement groups, and CRUSH configuration options for details on calculating a suitable number. **Note:** The default value 8 is not suitable for most systems.	Integer	Yes	8
pgp_num	The total number of placement groups for placement purposes. This should be equal to the total number of placement groups, except for placement group splitting scenarios. Valid range: Equal to or less than what specified by the `pg_num` variable.	Integer	Yes. Picks up default or Ceph configuration value if not specified.	None
crush_rule	The rule to use for mapping object placement in the cluster.	String	No	None

Value	Description	Type	Required	Default
hashpspool	Enable or disable the **HASHPSPOOL** flag on a given pool. With this option enabled, pool hashing and placement group mapping are changed to improve the way pools and placement groups overlap. Valid settings: `1` enables the flag, `0` disables the flag. **Important:** Do not enable this option on production pools of a cluster with a large amount of OSDs and data. All placement groups in the pool would have to be remapped causing too much data movement.	Integer	No	None

Value	Description	Type	Required	Default
fast_read	On a pool that uses erasure coding, if this flag is enabled, the read request issues subsequent reads to all shards, and waits until it receives enough shards to decode to serve the client. In the case of the `jerasure` and `isa erasure` plug-ins, once the first K replies return, the client's request is served immediately using the data decoded from these replies. This helps to allocate some resources for better performance. Currently this flag is only supported for erasure coding pools.	Boolean	No	`0`
allow_ec_overwrites	Whether writes to an erasure coded pool can update part of an object, so the Ceph Filesystem and Ceph Block Device can use it.	Boolean	No	None
compression_algorithm	Sets inline compression algorithm to use with the BlueStore storage backend. This setting overrides the **`bluestore_compression_algorithm`** configuration setting. Valid settings: `lz4`, `snappy`, `zlib`, `zstd`	String	No	None

Value	Description	Type	Required	Default
compression_mode	Sets the policy for the inline compression algorithm for the BlueStore storage backend. This setting overrides the **bluestore_compression_algorithm** configuration setting. Valid settings: `none`, `passive`, `aggressive`, `force`	String	No	None
compression_min_blob_size	BlueStore does not compress chunks smaller than this size. This setting overrides the **bluestore_compression_min_blob_size** configuration setting.	Unsigned Integer	No	None
compression_max_blob_size	BlueStore breaks chunks larger than this size into smaller blobs of **compression_max_blob_size** before compressing the data.	Unsigned Integer	No	None
nodelete	Set or unset the **NODELETE** flag on a given pool. Valid settings: `1` enables the flag, `0` disables the flag.	Integer	No	None
nopgchange	Set or unset the **NOPGCHANGE** flag on a given pool.	Integer	No	None
nosizechange	Set or unset the **NOSIZECHANGE** flag on a given pool. Valid settings: `1` enables the flag, `0` disables the flag.	Integer	No	None
write_fadvise_dontneed	Set or unset the **WRITE_FADVISE_DONTNEED** flag on a given pool. Valid settings: `1` enables the flag, `0` disables the flag.	Integer	No	None

Value	Description	Type	Required	Default
noscrub	Set or unset the **NOSCRUB** flag on a given pool. Valid settings: `1` enables the flag, `0` disables the flag.	Integer	No	None
nodeep-scrub	Set or unset the **NODEEP_SCRUB** flag on a given pool. Valid settings: `1` enables the flag, `0` disables the flag.	Integer	No	None
scrub_min_interval	The minimum interval in seconds for pool scrubbing when load is low. If it is `0`, Ceph uses the **osd_scrub_min_interval** configuration setting.	Double	No	`0`
scrub_max_interval	The maximum interval in seconds for pool scrubbing irrespective of cluster load. If it is `0`, Ceph uses the **osd_scrub_max_interval** configuration setting.	Double	No	`0`
deep_scrub_interval	The interval in seconds for pool *deep scrubbing*. If it is `0`, Ceph uses the `osd_deep_scrub_interval` configuration setting.	Double	No	`0`
peering_crush_bucket_count	The value is used along with peering_crush_bucket_barrier to determined whether the set of OSDs in the chosen acting set can peer with each other, based on the number of distinct buckets there are in the acting set.	Integer	No	None

Value	Description	Type	Required	Default
peering_crush_bucket_target	This value is used along with peering_crush_bucket_barrier and size to calculate the value `bucket_max` which limits the number of OSDs in the same bucket from getting chose to be in the acting set of a PG.	Integer	No	None
peering_crush_bucket_barrier	The type of bucket a pool is stretched across. For example, rack, row, or datacenter.	String	No	None

Erasure code pools overview

Ceph uses replicated pools by default, meaning that Ceph copies every object from a primary OSD node to one or more secondary OSDs. The erasure-coded pools reduce the amount of disk space required to ensure data durability but it is computationally a bit more expensive than replication.

Ceph storage strategies involve defining data durability requirements. Data durability means the ability to sustain the loss of one or more OSDs without losing data.

Ceph stores data in pools and there are two types of the pools:

- replicated
- erasure-coded

Erasure coding is a method of storing an object in the Ceph storage cluster durably where the erasure code algorithm breaks the object into data chunks (`k`) and coding chunks (`m`), and stores those chunks in different OSDs.

In an event of the failure of an OSD, Ceph retrieves the remaining data (`k`) and coding (`m`) chunks from the other OSDs and the erasure code algorithm restores the object from those chunks.

> **Note:** Use `min_size` for erasure-coded pools to be `K+1` or more to prevent loss of writes and data. For a 2+2 pool, the `min_size` should be k+1 or 3.

Erasure coding uses storage capacity more efficiently than replication. The n-replication approach maintains `n` copies of an object (3 times by default in Ceph), whereas erasure coding maintains only `k + m` chunks. For example, 3 data and 2 coding chunks use 1.5 times the storage space of the original object.

While erasure coding uses less storage overhead than replication, the erasure code algorithm uses more RAM and CPU than replication when it accesses or recovers objects. Erasure coding is advantageous when data storage must be durable and fault tolerant, but do not require fast read performance (for example, cold storage, historical records, and so on).

For the mathematical and detailed explanation on how erasure code works in Ceph, see the Erasure coding.

Ceph creates a default erasure code profile when initializing a cluster with **k=2** and **m=2**, This mean that Ceph will spread the object data over four OSDs **(k+m = 4)** and Ceph can lose one of those OSDs without losing data. To know more about erasure code profiling see “Erasure code profiles” on page 86 section.

> **Important:** Configure only the `.rgw.buckets` pool as erasure-coded and all other Ceph Object Gateway pools as replicated, otherwise an attempt to create a new bucket fails with the following error:

```
set_req_state_err err_no=95 resorting to 500
```

The reason for this is that erasure-coded pools do not support the `omap` operations and certain Ceph Object Gateway metadata pools require the `omap` support.

Creating a sample erasure-coded pool

Create an erasure-coded pool and specify the placement groups.

The **`ceph osd pool create`** command creates an erasure-coded pool with the default profile, unless another profile is specified. Profiles define the redundancy of data by setting two parameters, **k**, and **k**. These parameters define the number of chunks a piece of data is split and the number of coding chunks are created.

The simplest erasure-coded pool is equivalent to RAID5 and requires at least four hosts. You can create an erasure-coded pool with 2+2 profile.

1. Set the following configuration for an erasure-coded pool on four nodes with 2+2 configuration.

   ```
   ceph config set mon mon_osd_down_out_subtree_limit host
   ```

   Important: This is not needed for an erasure-coded pool in general.

   Note: For an EC cluster with four nodes the value of K+M is 2+2. If a node fails completely, it does not recover as four chunks and only three nodes are available. When you set the value of **`mon_osd_down_out_subtree_limit`** to `host`, during a host down scenario, it prevents the OSDs from marked out, so as to prevent the data from re balancing and the waits until the node is up again.

2. For an erasure-coded pool with a 2+2 configuration, set the profile.

   ```
   ceph osd erasure-code-profile set ec22 k=2 m=2 crush-failure-domain=host
   ```

   ```
   [ceph: root@host01 /]# ceph osd erasure-code-profile set ec22 k=2 m=2 crush-failure-
   domain=host

   Pool : ceph osd pool create test-ec-22 erasure ec22
   ```

3. Create an erasure-coded pool.

   ```
   ceph osd pool create ecpool 32 32 erasure
   pool 'ecpool' created
   $ echo ABCDEFGHI | rados --pool ecpool put NYAN -
   $ rados --pool ecpool get NYAN -
   ABCDEFGHI
   ```

   `32` is the number of placement groups.

Erasure code profiles

Ceph defines an erasure-coded pool with a *profile*. Ceph uses a profile when creating an erasure-coded pool and the associated CRUSH rule.

Ceph creates a default erasure code profile when initializing a cluster and it provides the same level of redundancy as two copies in a replicated pool. This default profile defines **k=2** and **m=2**, meaning Ceph spreads the object data over four OSDs (**k+m=4**) and Ceph can lose one of those OSDs without losing data. EC 2+2 requires a minimum deployment footprint of 4 nodes (5 nodes recommended) and can cope with the temporary loss of 1 OSD node.

To display the default profile use the following command:

```
$ ceph osd erasure-code-profile get default
k=2
m=2
plugin=jerasure
technique=reed_sol_van
```

You can create a new profile to improve redundancy without increasing raw storage requirements. For instance, a profile with **k=8** and **m=4** can sustain the loss of four (**m=4**) OSDs by distributing an object on 12 (**k+m=12**) OSDs. Ceph divides the object into **8** chunks and computes **4** coding chunks for recovery. For example, if the object size is 8 MB, each data chunk is 1 MB and each coding chunk has the same size as the data chunk, that is also 1 MB. The object is not lost even if four OSDs fail simultaneously.

The most important parameters of the profile are `k`, `m`, and `crush-failure-domain`, because they define the storage overhead and the data durability.

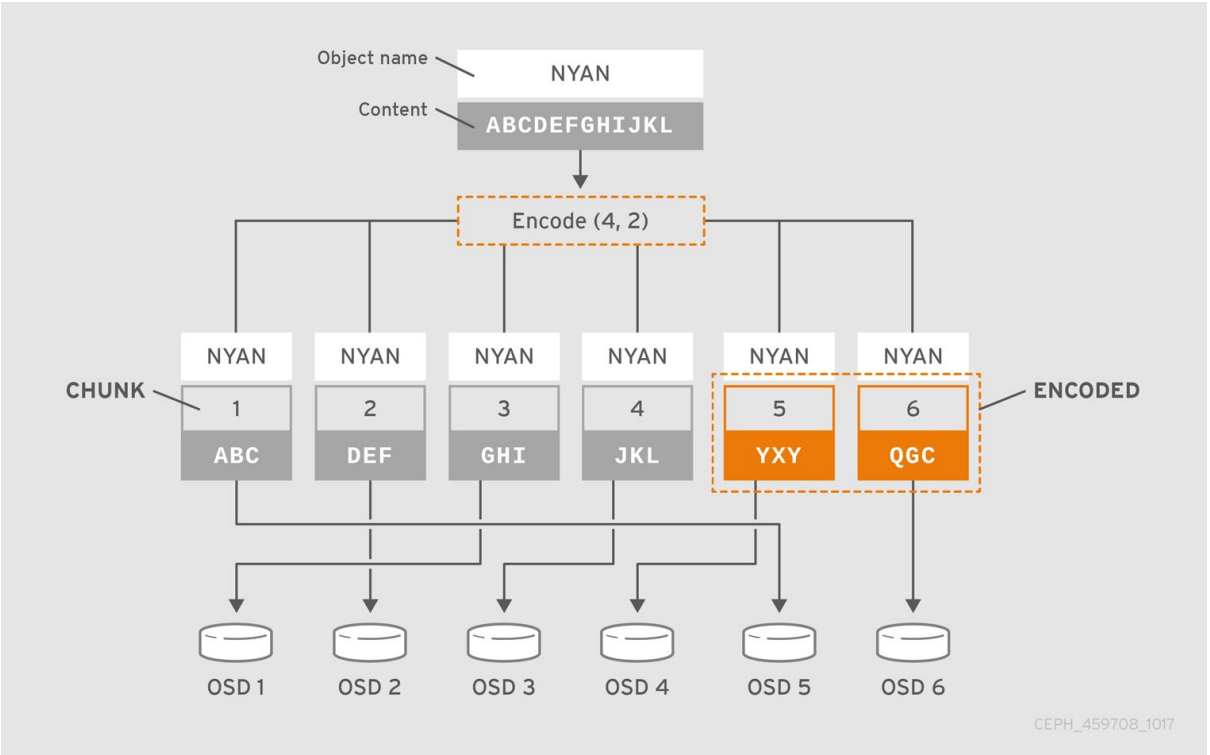
> **Important:** Choosing the correct profile is important because you cannot change the profile after you create the pool. To modify a profile, you must create a new pool with a different profile and migrate the objects from the old pool to the new pool.

For instance, if the desired architecture must sustain the loss of two racks with a storage overhead of 40% overhead, the following profile can be defined:

```
$ ceph osd erasure-code-profile set myprofile \
   k=4 \
   m=2 \
   crush-failure-domain=rack
$ ceph osd pool create ecpool 12 12 erasure *myprofile*
$ echo ABCDEFGHIJKL | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHIJKL
```

The primary OSD will divide the *NYAN* object into four (*k=4*) data chunks and create two additional chunks (*m=2*). The value of *m* defines how many OSDs can be lost simultaneously without losing any data. The *crush-failure-domain=rack* will create a CRUSH rule that ensures no two chunks are stored in the same rack.

Figure: Erasure code



Important:

- The following *jerasure* coding values are supported for *k*, and *m*:
  - k=8 m=3
  - k=8 m=4
  - k=4 m=2
- If the number of OSDs lost equals the number of coding chunks (`m`), some placement groups in the erasure coding pool will go into incomplete state. If the number of OSDs lost is less than `m`, no placement groups will go into incomplete state. In either situation, no data loss will occur. If placement groups are in incomplete state, temporarily reducing `min_size` of an erasure coded pool will allow recovery.

Supported erasure coding profiles lists the supported erasure coding profiles.

Supported erasure coding profiles

k	M	Supported Plugins	Minimum OSD Hosts (K+M)	Recommended OSD Hosts	Minimum OSDs	Redundancy Domain
2	2	jerasure, isa	4	4 + hosts (K+M)	4 (1 per node)	Loss of 1 nod 2 OSDs
2	2	jerasure, isa	5	5 + hosts(K+M+1)	5 (1 per node)	Loss of up to OSDs
4	2	jerasure, isa	6	7 + hosts (K+M+1)	7 (1 per node)	Loss of up to OSDs
5	2	jerasure, isa	7	8 + hosts (K+M+1)	8 (1 per node)	Loss of up to OSDs
6	2	jerasure, isa	8	9 + hosts (K+M+1)	9 (1 per node)	Loss of up to OSDs
8	3	jerasure, isa	11	12 + hosts (K+M+1)	12 (1 per node)	Loss of up to OSDs
8	4	jerasure, isa	12	13 + hosts (K+M+1)	13 (1 per node)	Loss of up to OSDs

k	M	Supported Plugins	Minimum OSD Hosts (K+M)	Recommended OSD Hosts	Minimum OSDs	Redundancy Domain
8	6	jerasure, isa	4	4 hosts{}	4 (4 per node)	Loss of 1 nod OR 6 OSDs

Note:
For clusters initialized on Red Hat Ceph Storage 9 or later, the default erasure code profile uses the ISA-L plugin instead of Jerasure.

For clusters upgraded to 9, the existing default Jerasure profile is preserved. You can create new profiles using either the ISA-L or Jerasure plugin depending on performance and compatibility requirements.

Creating a new `erasure-code-profile`

Learn how to create a new erasure code profile.

To create a new erasure code profile, use the following syntax, with the value information provided in “Erasure code profile values” on page 89.

```
ceph osd erasure-code-profile set NAME \
         [<directory=DIRECTORY>] \
         [<plugin=PLUGIN>] \
         [<stripe_unit=STRIPE_UNIT>] \
         [<CRUSH_DEVICE_CLASS>]\
         [<CRUSH_FAILURE_DOMAIN>]\
         [<key=value> ...] \
         [--force]
```

Erasure code profile values

Value	Description	Type	Required	Default
`directory`	Set the directory name from which the erasure code plug-in is loaded.	String	No.	`/usr/lib/ceph/erasure-code`

Value	Description	Type	Required	Default
`plugin`	Use the erasure code plug-in to compute coding chunks and recover missing details. For more information, see “Erasure code plugins” on page 94.	String	No.	`isa` (for new clusters created on Ceph 9.0 or later), `jerasure` (for clusters upgraded from earlier versions) Jerasure
`stripe_unit`	The amount of data in a data chunk, per stripe. For example, a profile with 2 data chunks and `stripe_unit=4K` would put the range 0-4K in chunk 0, 4K-8K in chunk 1, then 8K-12K in chunk 0 again. This should be a multiple of 4K for best performance. The default value is taken from the monitor config option `osd_pool_erasure_code_stripe_unit` when a pool is created. The stripe_width of a pool using this profile will be the number of data chunks multiplied by this `stripe_unit`.	String	No.	`4K`
`crush-device-class`	The device class, such as `hdd` or `ssd`.	String	No.	`none`, meaning CRUSH uses all devices regardless of class.
`crush-failure-domain`	The failure domain, such as `host` or `rack`.	String	No.	`host`
`key`	The semantic of the remaining key-value pairs is defined by the erasure code plug-in.	String	No.	N/A
`--force`	Override an existing profile by the same name.	String	No.	N/A

Note: The plugin default depends on how the cluster was created. New 9.0 clusters use ISA-L; upgraded clusters continue with Jerasure unless specified otherwise.

Removing an existing `erasure-code-profile`

Learn how to remove an erasure code profile.

- Remove an erasure code profile.

  ```
  ceph osd erasure-code-profile rm NAME
  ```

  where *NAME* is the name of the profile.

  Important: If the profile is referenced by a pool, the deletion fails.

  Warning: Removing an erasure code profile using **`osd erasure-code-profile rm`** command does not automatically delete the associated CRUSH rule that is associated with the erasure code profile. Manually remove the associated CRUSH rule using **`ceph osd crush rule remove RULE_NAME`** command to avoid unexpected behavior.

Fetching details for an `erasure-code-profile`

Learn how to display an erasure code profile.

To display an erasure code profile:

Syntax

```
ceph osd erasure-code-profile get NAME
```

where *NAME* is the name of the profile.

Listing all `erasure-code-profile` on cluster

Learn how to list the names of all erasure code profiles.

To list the names of all erasure code profiles:

Syntax

```
ceph osd erasure-code-profile ls
```

Erasure coding with overwrites

By default, erasure coded pools only work with the Ceph Object Gateway, which performs full object writes and appends.

Using erasure coded pools with overwrites allows Ceph Block Devices and CephFS store their data in an erasure coded pool:

Syntax

```
ceph osd pool set ERASURE_CODED_POOL_NAME allow_ec_overwrites true
```

Example

```
[ceph: root@host01 /]# ceph osd pool set ec_pool allow_ec_overwrites true
```

Enabling erasure coded pools with overwrites can only reside in a pool using BlueStore OSDs. Since BlueStore's checksumming is used to detect bit rot or other corruption during deep scrubs. Using FileStore with erasure coded overwrites is unsafe, and yields lower performance when compared to BlueStore.

Erasure coded pools do not support omap. To use erasure coded pools with Ceph Block Devices and CephFS, store the data in an erasure coded pool, and the metadata in a replicated pool.

For Ceph Block Devices, use the `--data-pool` option during image creation:

Syntax

```
rbd create --size IMAGE_SIZE_M|G|T --data-pool _ERASURE_CODED_POOL_NAME
REPLICATED_POOL_NAME/IMAGE_NAME
```

Example

```
[ceph: root@host01 /]# rbd create --size 1G --data-pool ec_pool rep_pool/image01
```

If using erasure coded pools for CephFS, then setting the overwrites must be done in a file layout.

Erasure coding optimizations

Erasure coding (EC) optimizations improve performance for small I/O operations and reduce storage overhead in erasure-coded pools by eliminating padding.

Starting with the Red Hat Ceph Storage 9 release, EC optimizations are enabled by default for newly created pools. This behavior is controlled by the global flag described below.

Workload impact
EC optimizations affect workloads differently depending on how applications store and access data:

RADOS Block Device (RBD) and Ceph File System (CephFS)

Gain higher performance and reduced capacity usage.

Ceph Object Gateway with large sequential objects

Experience minimal benefit.

Ceph Object Gateway with many small objects or random access

Gain improved performance and reduced capacity usage.

Configuration
When you enable EC optimizations, you cannot disable them because they change how data is stored. You can configure EC optimizations at either the pool level or the global level:

Pool level

Use the `allow_ec_optimizations` option to enable EC optimizations for a specific pool.

Global level

Use the `osd_pool_default_flag_ec_optimizations` option to set EC optimizations as the default for all newly created erasure-coded pools.

Requirements
You must meet the following requirements before you enable EC optimizations:

- Upgrade all Monitors and OSDs to Red Hat Ceph Storage 9 or later.
- Clients and gateways do not require upgrading.
- Supported with the following EC config combinations:

  Plugin

  Jerasure — Technique: `reed_sol_van`

  Plugin

  ISA-L — Technique: `reed_sol_van` or `cauchy`

> **Note:** Any EC pool created with the above plugins and technique will support EC enhancements, irrespective of `k, m` values. The appropriate technique is automatically selected based on the `k,m` configuration.

Limitations
Before enabling EC optimizations, review the following limitations to avoid configuration errors:

- Red Hat Ceph Storage blocks the operation with an error if you attempt to enable EC optimizations with an unsupported plugin or technique.

Enabling coding optimizations

Enable erasure coding (EC) optimizations, also known as Fast EC, on a pool to improve performance for small I/O operations and reduce capacity usage.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- All monitors and OSDs are upgraded to Red Hat Ceph Storage 9 or later.
- The erasure-coded pool is configured using one of the following supported combinations:
  Plugin

  Jerasure — Technique: `reed_sol_van`

  Plugin

  ISA-L — Technique: `reed_sol_van` or `cauchy`

  > **Note:** Any EC pool created with the above plugins and technique will support EC enhancements, irrespective of `k, m` values. The appropriate technique is automatically selected based on the `k,m` configuration.

- Gateways and clients do not require upgrading.

1. Run the following command to enable EC optimizations on a pool.

   ```
   ceph osd pool set EC_POOL allow_ec_optimizations true
   ```

2. Configure optimizations as the default for new pools by setting the **`osd_pool_default_flag_ec_optimizations`** configuration parameter.

   > **Note:** Starting with Red Hat Ceph Storage 9, this parameter is set to `true` by default. This enables EC optimizations automatically for newly created pools that use supported erasure-coded configurations.

   For example,

   ```
   ceph config set global osd_pool_default_flag_ec_optimizations true
   ```

Result

The pool is optimized for workloads with small objects or random read/write operations. EC pools become more suitable for Ceph Block Device and CephFS workloads. Ceph Object Gateway workloads with many small objects benefit, while large sequential objects see minimal improvement.

> **Note:**

- Once enabled, EC optimizations cannot be disabled because they change how data is stored.
- EC optimizations are supported only with the Jerasure and ISA-L plugins using the reed_sol_van or cauchy technique. Unsupported combinations are blocked with an error.

Erasure code plugins

Ceph supports erasure coding with a plug-in architecture, which means you can create erasure-coded pools by using different erasure coding algorithms.

Starting with Red Hat Ceph Storage 9, the default erasure code plug-in for new clusters changes from Jerasure to ISA-L. ISA-L (Intel® Intelligent Storage Acceleration Library) provides optimized Reed-Solomon–based erasure coding with improved encoding and recovery performance on modern CPUs.Clusters that are upgraded to Red Hat Ceph Storage 9 continue to use the existing Jerasure-based default profile. You can still create custom profiles using either plug-in.

Ceph supports the following erasure code plug-ins:

- ISA-L - Default for new clusters created in Red Hat Ceph Storage 9 and later.
- Jerasure - Default for clusters created before Red Hat Ceph Storage 9 and still supported.
- Intel Storage Acceleration (ISA) library (Default) Jerasure

Creating a new erasure code profile using Jerasure erasure code plugin

The ISA-L plugin provides high-performance Reed-Solomon erasure coding using the Intel® Intelligent Storage Acceleration Library. Starting with Red Hat Ceph Storage 9, new clusters created on this release use ISA-L as the default erasure code plugin.

The ISA-L plug-in encapsulates the JerasureH library. For detailed information about the parameters, see the Jerasure documentation.

To create a new erasure code profile using the Jerasure plug-in, run the following command, with the value information provided in “New EC profile using Jerasure values” on page 94:

```
ceph osd erasure-code-profile set NAME \
     plugin=jerasure \
     k=DATA_CHUNKS \
     m=DATA_CHUNKS \
     technique=TECHNIQUE \
     [crush-root=ROOT] \
     [crush-failure-domain=BUCKET_TYPE] \
     [directory=DIRECTORY] \
     [--force]
```

New EC profile using Jerasure values

Value	Description	Type	Required	Example/Default
k	Each object is split in data-chunks parts, each stored on a different OSD.	Integer	Yes.	Example: 4
m	Compute coding chunks for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.	Integer	Yes.	Example: 2

Value	Description	Type	Required	Example/Default
`technique`	The more flexible technique is **reed_sol_van**; it is enough to set `k` and `m`. The **cauchy_good** technique can be faster but you need to choose the `packetsize` carefully. All of `reed_sol_r6_op`, `liberation`, `blaum_roth`, `liber8tion` are RAID6 equivalents in the sense that they can only be configured with `m=2`. Valid settings are as follows: `reed_sol_van`, `reed_sol_r6_op`, `cauchy_orig`, `cauchy_good`, `liberation`, `blaum_roth`, `liber8tion`	String	No.	Default: `reed_sol_van`
`packetsize`	The encoding will be done on packets of bytes size at a time. Choosing the correct packet size is difficult. The Jerasure documentation contains extensive information on this topic.	Integer	No.	Default: `2048`
`crush-root`	The name of the CRUSH bucket used for the first step of the rule. For instance **step take default**.	String	No.	Default: default
`crush-failure-domain`	Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is `host`, no two chunks will be stored on the same host. It is used to create a rule step such as step chooseleaf host.	String	No.	Default: `host`

Value	Description	Type	Required	Example/Default
`directory`	Set the **directory** name from which the erasure code plug-in is loaded.	String	No.	Default: `/usr/lib/ceph/erasure-code`
`--force`	Override an existing profile by the same name.	String	No.	N/A

Creating a new erasure code profile using the ISA-L erasure code plugin

The ISA-L plugin provides high-performance Reed-Solomon erasure coding using the Intel® Intelligent Storage Acceleration Library. Starting with Red Hat Ceph Storage 9, new clusters created on this release use ISA-L as the default erasure code plugin. Clusters upgrading to Red Hat Ceph Storage 9 continue using their existing Jerasure-based default profile, but they may create new ISA-L or Jerasure profiles as needed.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- The cluster is running Red Hat Ceph Storage 9 or later.
- You have administrator privileges.
- All OSD nodes have ISA-L support installed.

1. To create an ISA-L–based erasure code profile, run the following commands:

```
ceph osd erasure-code-profile set <NAME> \
     plugin=isa \
     k=DATA_CHUNKS \
     m=CODING_CHUNKS \
     technique=TECHNIQUE \
     [crush-root=ROOT] \
     [crush-failure-domain=BUCKET_TYPE] \
     [directory=DIRECTORY] \
     [--force]
```

   The following example creates a 4+2 ISA-L profile:

```
ceph osd erasure-code-profile set isal_4_2 \
     plugin=isa \
     k=4 \
     m=2 \
     technique=reed_sol_van \
     crush-failure-domain=host
```

2. Create a pool using the profile:

```
ceph osd pool create ecpool 12 12 erasure isal_4_2
```

Result

The ISA-L erasure code profile is created and available for use. You can verify the profile by running the following command:

```
ceph osd erasure-code-profile get <NAME>
```

Controlling CRUSH placement

Learn how to control CRUSH placement.

The default CRUSH rule provides OSDs that are on different hosts. For instance:

```
chunk nr    01234567

step 1      _cDD_cDD
```

```
step 2      cDDD____
step 3      ____cDDD
```

needs exactly *8* OSDs, one for each chunk. If the hosts are in two adjacent racks, the first four chunks can be placed in the first rack and the last four in the second rack. Recovering from the loss of a single OSD does not require using bandwidth between the two racks.

For instance:

```
crush-steps='[ [ "choose", "rack", 2 ], [ "chooseleaf", "host", 4 ] ]'
```

creates a rule that selects two crush buckets of type *rack* and for each of them choose four OSDs, each of them located in a different bucket of type *host*.

The rule can also be created manually for finer control.