Red Hat Ceph Storage Overview

Learn about the architecture, data security, and hardening concepts for Red Hat Ceph Storage.

Introduction to Red Hat Ceph Storage

Red Hat Ceph Storage cluster is a distributed data object store designed to provide excellent performance, reliability and scalability.

Red Hat Ceph Storage is a scalable, open, software-defined storage platform that combines an enterprise-hardened version of the Ceph storage system, with a Ceph management platform, deployment utilities, and support services. Red Hat Ceph Storage is designed for cloud infrastructure and web-scale object storage.

Distributed object stores are the future of storage, because they accommodate unstructured data, and because clients can use modern object interfaces and legacy interfaces simultaneously.

For example:

- APIs in many languages (C/C++, Java, Python)
- RESTful interfaces (S3/Swift)
- Block device interface
- File system interface

The power of Red Hat Ceph Storage cluster can transform your organization's IT infrastructure and your ability to manage vast amounts of data, especially for cloud computing platforms like Red Hat Enterprise Linux OSP. The cluster delivers extraordinary scalability–thousands of clients accessing petabytes to exabytes of data and beyond.

At the heart of every Ceph deployment is the Red Hat Ceph Storage cluster. It consists of three types of daemons:

Ceph OSD Daemon

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

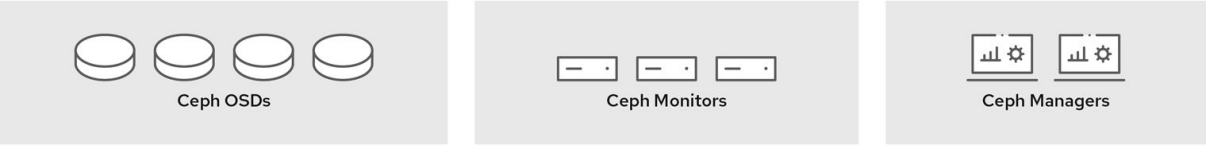
Ceph Monitor

A Ceph Monitor maintains a master copy of the Red Hat Ceph Storage cluster map with the current state of the cluster. Monitors require high consistency, and use Paxos to ensure agreement about the state of the cluster.

Ceph Manager

The Ceph Manager maintains detailed information about placement groups, process metadata and host metadata in lieu of the Ceph Monitor—significantly improving performance at scale. The Ceph Manager handles execution of many of the read-only Ceph CLI queries, such as placement group statistics. The Ceph Manager also provides the RESTful monitoring APIs.

Figure: Daemons



Ceph client interfaces read data from and write data to the Red Hat Ceph Storage cluster. Clients need the following data to communicate with the Red Hat Ceph Storage cluster:

- The Ceph configuration file, or the cluster name (usually `ceph`) and the monitor address.
- The pool name.

– The user name and the path to the secret key.

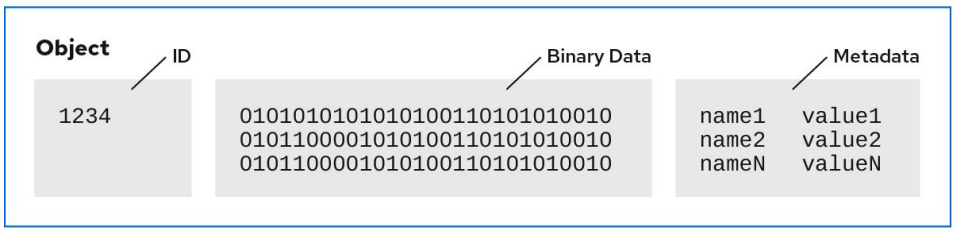
Ceph clients maintain object IDs and the pool names where they store the objects. However, they do not need to maintain an object-to-OSD index or communicate with a centralized object index to look up object locations. Then, Ceph clients provide an object name and pool name to `librados`, which computes an object's placement group and the primary OSD for storing and retrieving data using the CRUSH (Controlled Replication Under Scalable Hashing) algorithm. The Ceph client connects to the primary OSD where it may perform read and write operations. There is no intermediary server, broker or bus between the client and the OSD.

When an OSD stores data, it receives data from a Ceph client, whether the client is a Ceph Block Device, a Ceph Object Gateway, a Ceph File System or another interface and it stores the data as an object.

> **Note:** An object ID is unique across the entire cluster, not just an OSD's storage media.

Ceph OSDs store all data as objects in a flat namespace. There are no hierarchies of directories. An object has a cluster-wide unique identifier, binary data, and metadata consisting of a set of name/value pairs.

Figure: Object



Ceph clients define the semantics for the client's data format. For example, the Ceph Block Device maps a block device image to a series of objects stored across the cluster.

> **Note:** Objects consisting of a unique ID, data, and name/value paired metadata can represent both structured and unstructured data, as well as legacy and leading edge data storage interfaces.

Red Hat Ceph Storage clusters consist of the following types of nodes:

Ceph Monitor

Each Ceph Monitor node runs the `ceph-mon` daemon, which maintains a primary copy of the storage cluster map. The storage cluster map includes the storage cluster topology. A client connecting to the Ceph storage cluster retrieves the current copy of the storage cluster map from the Ceph Monitor, enabling the client to read from and write data to the storage cluster.

> **Important:** The storage cluster can run with just one Ceph Monitor; however, to ensure high availability in a production storage cluster, Red Hat supports deployments with at least three Ceph Monitor nodes. Deploy a total of 5 Ceph Monitors for storage clusters exceeding 750 Ceph OSDs.

Ceph Manager

The Ceph Manager daemon, `ceph-mgr`, co-exists with the Ceph Monitor daemons running on Ceph Monitor nodes to provide extra services. The Ceph Manager provides an interface for other monitoring and management systems using Ceph Manager modules. Running the Ceph Manager daemons is a requirement for normal storage cluster operations.

Ceph OSD

Each Ceph Object Storage Device (OSD) node runs the `ceph-osd` daemon, which interacts with logical disks that are attached to the node. The storage cluster stores data on these Ceph OSD nodes.
Ceph can run with few OSD nodes, of which the default is three, but production storage clusters realize better performance beginning at modest scales. For example, 50 Ceph OSDs in a storage cluster. Ideally, a

Ceph storage cluster has multiple OSD nodes, allowing for the possibility to isolate failure domains by configuring the CRUSH map.

Ceph MDS

Each Ceph Metadata Server (MDS) node runs the `ceph-mds` daemon, which manages metadata related to files stored on the Ceph File System (CephFS). The Ceph MDS daemon also coordinates access to the shared storage cluster.

Ceph Object Gateway

Ceph Object Gateway node runs the `ceph-radosgw` daemon, and is an object storage interface built on top of `librados` to provide applications with a RESTful access point to the Ceph storage cluster. The Ceph Object Gateway supports two interfaces:

S3

Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.

Swift

Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.

Note:

- For more information about Ceph architecture, see Architecture.
- For the minimum hardware recommendations, see Hardware.

Understanding Red Hat Ceph Storage terminology

Understand terms and definitions for the Red Hat Ceph Storage storage system.

BlueStore

BlueStore stores objects directly on raw block devices or partitions, and does not interact with mounted file systems. BlueStore uses RocksDB's key/value database to map object names to block locations on disk.

Bucket

In the context of Ceph Object Gateway, a bucket is a group of objects. In a filesystem-based analogy in which objects are the counterpart of files, buckets are the counterpart of directories. Multi-site sync policies can be set on buckets to provide fine-grained control of data movement from one zone to another zone. The concept of the bucket has been taken from AWS S3. See also "Ceph Object Gateway" on page 4.

Ceph Block Device

Also called *RADOS Block Device* and *RBD*. A software instrument that orchestrates the storage of block-based data in Red Hat Ceph Storage. Ceph Block Device splits block-based application data into "chunks". RADOS stores these chunks as objects. Ceph Block Device orchestrates the storage of those objects across the storage cluster.

Ceph Block Storage

One of the three kinds of storage supported by Red Hat Ceph Storage (the other two are object storage and file storage). Ceph Block Storage is the block storage product, which refers to block-storage related services and capabilities when used in conjunction with the collection of the following:

- `librbd`, a python module that provides file-like access to RBD images.
- A hypervisor such as QEMU or Xen.
- A hypervisor abstraction layer such as `libvirt`.

Ceph client

Any of the Red Hat Ceph Storage components that can access a Ceph Storage Cluster. This includes the Ceph Object Gateway, the Ceph Block Device, the Ceph File System, and their corresponding libraries. It also includes kernel modules, and FUSEs (Filesystems in USERspace).

Ceph client libraries

The collection of libraries that can be used to interact with components of the Red Hat Ceph Storage Cluster.

Ceph cluster map

See cluster map.

Ceph File System

The Ceph File System, or *CephFS*, is a POSIX-compliant file system built on top of Ceph's distributed object store, RADOS. See also "RADOS" on page 7.

CephFS

See "Ceph File System" on page 4.

Ceph kernel modules

The collection of kernel modules that can be used to interact with the Red Hat Ceph Storage cluster, such as `ceph.ko` and `rbd.ko`.

Ceph Manager

The Ceph Manager daemon. Also referred to as *ceph-mgr*. The Ceph Manager is a daemon that runs alongside monitor daemons to provide monitoring and interfacing to external monitoring and management systems.

Ceph Metadata Server

The Ceph MetaData Server daemon. Also referred to as *ceph-mds* and *MDS*. The MDS daemon must be running in any Red Hat Ceph Storage cluster that runs the CephFS. The MDS stores all filesystem metadata.

Ceph Monitor

A daemon that maintains a map of the state of the cluster. This cluster state includes the monitor map, the manager map, the OSD map, and the CRUSH map. a Red Hat Ceph Storage cluster must contain a minimum of three running monitors to be both redundant and highly-available. Ceph monitors and the nodes on which they run are often referred to as *mons*.

Ceph node

A Ceph node is a unit of the Ceph Cluster that communicates with other nodes in the Ceph Cluster in order to replicate and redistribute data. All of the nodes together are called the Ceph Storage Cluster. Ceph nodes include OSDs, Ceph Monitors, Ceph Managers, and MDSes. The term *node* is usually equivalent to *host* in the Red Hat Ceph Storage documentation. If you have a running Ceph Cluster, you can list all of the nodes in it by running the **`ceph node ls all`** command.

Ceph Object Gateway

Also called *RADOS Gateway* and *RGW*. An object storage interface built on top of `librados`. Ceph Object Gateway provides a RESTful gateway between applications and Ceph storage clusters. See also "RADOS Gateway" on page 7.

Ceph Object Storage

A Ceph Object Store consists of a Ceph Storage Cluster and a Ceph Object Gateway (RGW).

Ceph Object Store

See "Ceph Object Storage" on page 4.

Ceph OSD

Ceph Object Storage Daemon (OSD). The Ceph OSD software, which interacts with logical disks.

Ceph OSD Daemon

See "Ceph OSD" on page 4.

Ceph stack

A collection of two or more components of Red Hat Ceph Storage.

Ceph Storage Cluster

The collection of Ceph Monitors, Ceph Managers, Ceph Metadata Servers, and OSDs that work together to store and replicate data for use by applications, Ceph Users, and Ceph Clients. Ceph Storage Clusters receive data from Red Hat Ceph Storage Clients.

CephX

The Ceph authentication protocol. CephX authenticates users and daemons. CephX operates like Kerberos, but it has no single point of failure.

Clone

A full data copy from an existing volume or disk drive to a new volume or disk drive. Clearly distinguish from a virtual machine clone, which copies both storage and configuration.

Cloud stacks

Third party cloud provisioning platforms such as OpenStack, CloudStack, OpenNebula, and Proxmox VE.

Cloud storage cluster

The collection of Ceph Monitors, Ceph Managers, Ceph Metadata Servers, and OSDs that work together to store and replicate data for use by applications, Ceph Users, and Ceph Clients. Ceph Storage Clusters receive data from Ceph Clients.

Cluster map

The cluster map represents all components that are being used in a Ceph cluster that together report the state of the Ceph cluster. The components include the following: monitor map, OSD map, placement group map, MDS map, and CRUSH map.

Container storage interface

Container Storage Interface (CSI) is a software interface for Kubernetes and other container orchestrators. The CSI allows storage providers to create plug-ins so that their applications can be exposed to containerized applications as persistent storage without a need for users to understand all of the specifics and complexity of the underlying storage infrastructure. Using the CSI, users can address requests against the CSI API in a common way.

CRUSH

Controlled Replication Under Scalable Hashing (*CRUSH*). The algorithm that Red Hat Ceph Storage uses to compute object storage locations.

CRUSH rule

The CRUSH data placement rule that applies to a particular pool or pools. See also "CRUSH" on page 5.

DAS

Direct-Attached Storage (*DAS*). Storage that is attached directly to the computer accessing it, without passing through a network. Contrast with NAS and SAN.

Distributed file system

A file system that presents files from a number of different storage devices, potentially on many different machines and in many different locations, as a single interface to an end user or consuming service.

Distributed volume

A storage volume that distributes data across multiple storage nodes and resources.

Ephemeral storage

A temporary storage location that allows pods to run internal resources required for their functioning. Ephemeral storage only exists when a pod exists. It is non-persistent storage and non-sharable with other pods processes.

Epoch

In the context of Ceph Object Gateway, an epoch is a group of time periods.

File System in User Space

File System In User Space (FUSE). A software interface for Linux and Linux-like systems that lets non-privileged users create and configure their own file systems without touching kernel code.

FQDN

See "Fully Qualified Domain Name" on page 5.

Fully Qualified Domain Name

Fully Qualified Domain Name (FQDN). A domain name that is applied to a node in a network and that specifies the node's exact location in the tree hierarchy of the DNS.
In the context of Ceph cluster administration, FQDNs are often applied to hosts. In this documentation, the term *FQDN* is used mostly to distinguish between FQDNs and relatively simpler hostnames, which do not specify the exact location of the host in the tree hierarchy of the DNS but merely name the host.

FUSE

See "File System in User Space" on page 5.

Georeplication

A way to distribute data over long distances and over latency networks with the goal to provide a copy of the data for disaster recovery purposes.

Hybrid OSD

Refers to an OSD that has both HDD and SSD drives.

Logical volume

A logical volume (LV) is a virtual block storage device that a file system, database, or application can use. To create a Logical Volume Manager (LVM) logical volume, the physical volumes are combined into a volume group. This creates a pool of disk space out of which LVM logical volumes can be allocated.

LVM cache

A caching mechanism used to improve the performance of a logical volume (LV). Typically, a smaller and faster device is used to improve I/O performance of a larger and slower LV. See also “Logical volume” on page 6.

LVM tags

Logical Volume Manager tags. Extensible metadata for LVM volumes and groups. They are used to store Ceph-specific information about devices and its relationship with OSDs.

MDS

See “Ceph Metadata Server” on page 4.

MGR

See “Ceph Manager” on page 4.

MON

See “Ceph Monitor” on page 4.

Object bucket

A logical container for objects to be stored in.

Object bucket claim

A request by a user or application for a S3-compatible bucket that is used for workload processing.

Object storage

A repository of data objects for unstructured data.

OSD

See “Ceph OSD” on page 4.

OSD FSID

This is a unique identifier used to identify an OSD. It is found in the OSD path in a file called `osd_fsid`. The term *FSID* is used interchangeably with *UUID*. See also “OSD UUID” on page 6.

OSD ID

The integer that defines an OSD. It is generated by the monitors during the creation of each OSD.

OSD UUID

This is the unique identifier of an OSD. The term *UUID* is used interchangeably with *FSID*. See also “OSD FSID” on page 6.

Period

In the context of Ceph Object Gateway, a period is the configuration state of the *Realm*. The period stores the configuration state of a multi-site configuration. When the period is updated, the “epoch” is said thereby to have been changed. See also “RADOS Gateway” on page 7 and “Realm” on page 7.

Persistent volume

A storage volume for data in an OpenShift cluster that provides persistent storage, rather than the ephemeral storage provided by default. A request for this kind of storage is a Persistent Volume Claim (PVC).
See also “Persistent volume claim” on page 6.

Persistent volume claim

Persistent volume claim (PVC). A request by a user or application for storage that is associated to a persistent volume or disk drive.

RADOS

Reliable Autonomic Distributed Object Store (*RADOS*). RADOS is the object store that provides a scalable service for variably-sized objects. The RADOS object store is the core component of a Ceph cluster. It is the core set of storage software which stores the user data (MON and OSD). See also “Ceph Monitor” on page 4 and “Ceph OSD” on page 4.

RADOS cluster

A proper subset of the Ceph Cluster consisting of OSDs, Ceph Monitors, and Ceph Managers.

RADOS Gateway

Also called *Ceph Object Gateway* or *RGW*. The component of Ceph that provides a gateway to both the Amazon S3 RESTful API and the OpenStack Swift API. See “Ceph Object Gateway” on page 4.

RBD

RADOS Block Device. See “Ceph Block Device” on page 3.

Realm

In the context of Ceph Object Gateway, a realm is a globally unique namespace that consists of one or more zonegroups.

RGW

See “RADOS Gateway” on page 7.

Scale out

Adding servers to achieve better performance, availability, and in the case of storage, more capacity.

Scale up

Increasing capacity or performance by adding more storage devices or CPUs into a server.

Scrubs

The processes by which Red Hat Ceph Storage ensures data integrity. During the process of scrubbing, Red Hat Ceph Storage generates a catalog of all objects in a placement group, then ensures that none of the objects are missing or mismatched by comparing each primary object against its replicas, which are stored across other OSDs. Any placement group (PG) is determined to have a copy of an object that is different than the other copies or is missing entirely is marked `inconsistent` (that is, the PG is marked “inconsistent”).
There are two kinds of scrubbing: light scrubbing and deep scrubbing (also called “normal scrubbing” and “deep scrubbing”, respectively). Light scrubbing is performed daily and does nothing more than confirm that a given object exists and that its metadata is correct. Deep scrubbing is performed weekly and reads the data and uses checksums to ensure data integrity.

Scrubbing

See “Scrubs” on page 7.

Secrets

Secrets are credentials used to perform digital authentication whenever privileged users must access systems that require authentication. Secrets can be passwords, API keys, tokens, SSH keys, private certificates, or encryption keys.

Shard

A shard is a small part of a larger container. Shards replicate independently in parallel with other shards. Operations on shards take less time than operations on the whole container, which makes replication and maintenance more reliable.

Storage class

A storage class defines the type of storage and eventual storage specifics. Storage classes provide a way for administrators to describe the classes of storage they offer. Classes can map to storage type, quality of service levels, backup policies, storage profiles, storage types, and storage features.
This is different than a Kubernetes StorageClass object.

Teuthology

The collection of software that performs scripted tests on Ceph.

Zone

In the context of Ceph Object Gateway, a zone is a logical group that consists of one or more Ceph Object Gateway instances. A zone’s configuration state is stored in the period. See also “Ceph Object Gateway” on page 4.

Architecture

Get to know the architecture information for Ceph Storage Clusters and their clients.

Ceph architecture

Red Hat Ceph Storage cluster is a distributed data object store designed to provide excellent performance, reliability and scalability. The power of Red Hat Ceph Storage cluster can transform your organization's IT infrastructure and your ability to manage vast amounts of data, especially for cloud computing platforms like Red Hat Enterprise Linux OSP. The cluster delivers extraordinary scalability–thousands of clients accessing petabytes to exabytes of data and beyond.

Distributed object stores are the future of storage, because they accommodate unstructured data, and because clients can use modern object interfaces and legacy interfaces simultaneously.

For example:

- APIs in many languages (C/C++, Java, Python)
- RESTful interfaces (S3/Swift)
- Block device interface
- Filesystem interface

At the heart of every Ceph deployment is the Red Hat Ceph Storage cluster. It consists of three types of daemons:

Ceph OSD Daemon

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

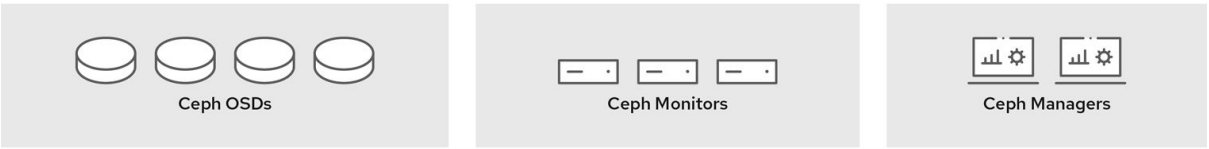
Ceph Monitor

A Ceph Monitor maintains a master copy of the Red Hat Ceph Storage cluster map with the current state of the cluster. Monitors require high consistency, and use Paxos to ensure agreement about the state of the cluster.

Ceph Manager

The Ceph Manager maintains detailed information about placement groups, process metadata and host metadata in lieu of the Ceph Monitor—significantly improving performance at scale. The Ceph Manager handles execution of many of the read-only Ceph CLI queries, such as placement group statistics. The Ceph Manager also provides the RESTful monitoring APIs.

Figure: Daemons



Ceph client interfaces read data from and write data to the Red Hat Ceph Storage cluster. Clients need the following data to communicate with the Red Hat Ceph Storage cluster:

- The Ceph configuration file, or the cluster name (usually `ceph`) and the monitor address.
- The pool name.
- The user name and the path to the secret key.

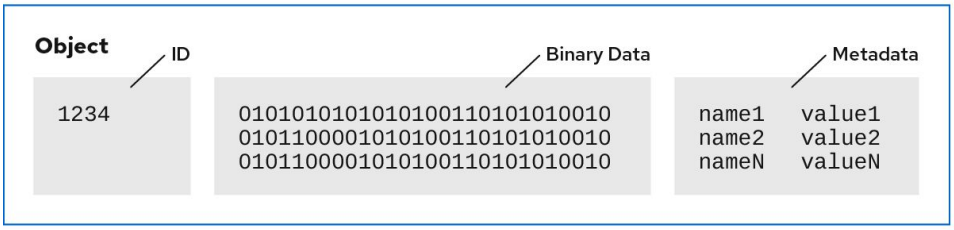
Ceph clients maintain object IDs and the pool names where they store the objects. However, they do not need to maintain an object-to-OSD index or communicate with a centralized object index to look up object locations. Then, Ceph clients provide an object name andI pool name to `librados`, which computes an object's placement group and the primary OSD for storing and retrieving data using the CRUSH (Controlled Replication Under Scalable Hashing) algorithm. The Ceph client connects to the primary OSD where it may perform read and write operations. There is no intermediary server, broker or bus between the client and the OSD.

When an OSD stores data, it receives data from a Ceph client—whether the client is a Ceph Block Device, a Ceph Object Gateway, a Ceph Filesystem or another interface and it stores the data as an object.

> **Note:** An object ID is unique across the entire cluster, not just an OSD's storage media.

Ceph OSDs store all data as objects in a flat namespace. There are no hierarchies of directories. An object has a cluster-wide unique identifier, binary data, and metadata consisting of a set of name/value pairs.

Figure: Object



Ceph clients define the semantics for the client's data format. For example, the Ceph block device maps a block device image to a series of objects stored across the cluster.

> **Note:** Objects consisting of a unique ID, data, and name/value paired metadata can represent both structured and unstructured data, as well as legacy and leading edge data storage interfaces.

Ceph client components

Ceph clients differ materially in how they present data storage interfaces. A Ceph Block Device presents block storage that mounts just like a physical storage drive. A Ceph gateway presents an object storage service with S3-compliant and Swift-compliant RESTful interfaces with its own user management. However, all Ceph clients use the Reliable Autonomic Distributed Object Store (RADOS) protocol to interact with the Red Hat Ceph Storage cluster.

Ceph clients all have the same basic needs:

- The Ceph configuration file, and the Ceph monitor address.
- The pool name.
- The user name and the path to the secret key.

Ceph clients tend to follow some similar patterns, such as object-watch-notify and striping. The following sections describe a little bit more about RADOS, librados and common patterns used in Ceph clients.

Prerequisites

Be sure to have a basic understanding of distributed storage systems before learning about the Ceph client components.

Ceph client native protocol

Modern applications need a simple object storage interface with asynchronous communication capability. The Red Hat Ceph Storage Cluster provides a simple object storage interface with asynchronous communication capability.

The interface provides direct, parallel access to objects throughout the cluster.

- Pool Operations
- Snapshots
- Read/Write Objects

- Create or Remove
- Entire Object or Byte Range
- Append or Truncate

– Create/Set/Get/Remove XATTRs

– Create/Set/Get/Remove Key/Value Pairs

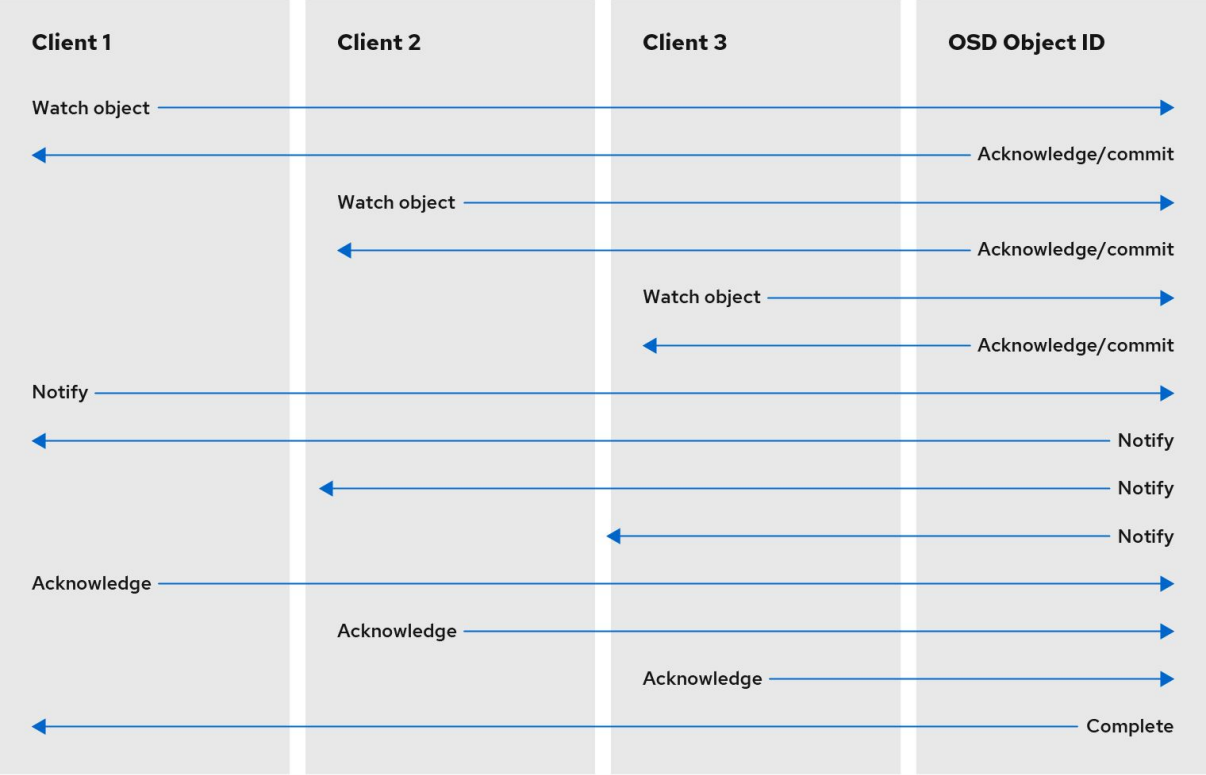
– Compound operations and dual-ack semantics

Ceph client object watch and notify

A Ceph client can register a persistent interest with an object and keep a session to the primary OSD open. The client can send a notification message and payload to all watchers and receive notification when the watchers receive the notification. This enables a client to use any object as a synchronization/communication channel.

“Figure: Ceph client object watch and notify” on page 10 illustrates the Ceph client object watch and notify workflow.

Figure: Ceph client object watch and notify



Ceph client `Mandatory Exclusive Locks`

Mandatory Exclusive Locks is a feature that locks an RBD to a single client, if multiple mounts are in place. Enabling this feature means that only one client can modify an RBD device at a time and provides protection for failed clients.

Mandatory Exclusive Locks helps address the write conflict situation when multiple mounted clients try to write to the same object. This feature is built on `object-watch-notify` explained in the previous section. So, when writing, if one client first establishes an exclusive lock on an object, another mounted client will first check to see if a peer has placed a lock on the object before writing.

With this feature enabled, only one client can modify an RBD device at a time, especially when changing internal RBD structures during operations like `snapshot create/delete`. It also provides some protection for failed

clients. For instance, if a virtual machine seems to be unresponsive and you start a copy of it with the same disk elsewhere, the first one will be blacklisted in Ceph and unable to corrupt the new one.

Mandatory Exclusive Locks are not enabled by default. You have to explicitly enable it with `--image-feature` parameter when creating an image.

Example

```
[root@mon ~]# rbd create --size 102400 mypool/myimage --image-feature 13
```

In this example, the numeral 13 is a summation of 1, 4, and 8, where 1 enables layering support, 4 enables exclusive locking support, and 8 enables object map support. So, the command creates a 100 GB RBD image, enable layering, exclusive lock, and object map.

Mandatory Exclusive Locks is also a prerequisite for `object map`. Without enabling exclusive locking support, object map support cannot be enabled.

Mandatory Exclusive Locks also does some ground work for mirroring.

Ceph client object map

Object map is a feature that tracks the presence of backing RADOS objects when a client writes to an RBD image. When a write occurs, that write is converted to an offset within a backing RADOS object. When the object map feature is enabled, the presence of these RADOS objects is tracked, allowing Ceph to know if the objects actually exist.

Object map is kept in-memory on the librbd client so it can avoid querying the OSDs for objects that it knows don't exist. In other words, object map is an index of the objects that actually exist.

Object map is beneficial for certain operations, viz:

- Resize
- Export
- Copy
- Flatten
- Delete
- Read

A shrink **resize** operation is like a partial delete where the trailing objects are deleted.

An **export** operation knows which objects are to be requested from RADOS.

A **copy** operation knows which objects exist and need to be copied. It does not have to iterate over potentially hundreds and thousands of possible objects.

A **flatten** operation performs a copy-up for all parent objects to the clone so that the clone can be detached from the parent i.e, the reference from the child clone to the parent snapshot can be removed. So, instead of all potential objects, copy-up is done only for the objects that exist.

A **delete** operation deletes only the objects that exist in the image.

A **read** operation skips the read for objects it knows doesn't exist.

So, for operations like resize, shrinking only, exporting, copying, flattening, and deleting, these operations would need to issue an operation for all potentially affected RADOS objects, whether they exist or not. With object map enabled, if the object doesn't exist, the operation need not be issued.

For example, if we have a 1 TB sparse RBD image, it can have hundreds and thousands of backing RADOS objects. A delete operation without object map enabled would need to issue a `remove object` operation for each potential object in the image. But if object map is enabled, it only needs to issue `remove object` operations for the objects that exist.

Object map is valuable against clones that don't have actual objects but get objects from parents. When there is a cloned image, the clone initially has no objects and all reads are redirected to the parent. So, object map can improve reads as without the object map, first it needs to issue a read operation to the OSD for the clone, when that fails, it issues another read to the parent — with object map enabled. It skips the read for objects it knows doesn't exist.

Object map is not enabled by default. You have to explicitly enable it with `--image-features` parameter when creating an image. Also, `Mandatory Exclusive Locks` is a prerequisite for `object map`. Without enabling exclusive locking support, object map support cannot be enabled. To enable object map support when creating a image, run:

Example

```
[root@mon ~]# rbd create --size 102400 mypool/myimage --image-feature 13
```

In this example, the numeral 13 is a summation of 1, 4, and 8, where 1 enables layering support, 4 enables exclusive locking support, and 8 enables object map support. So, the command creates a 100 GB RBD image, enable layering, exclusive lock and object map.

Ceph client data striping

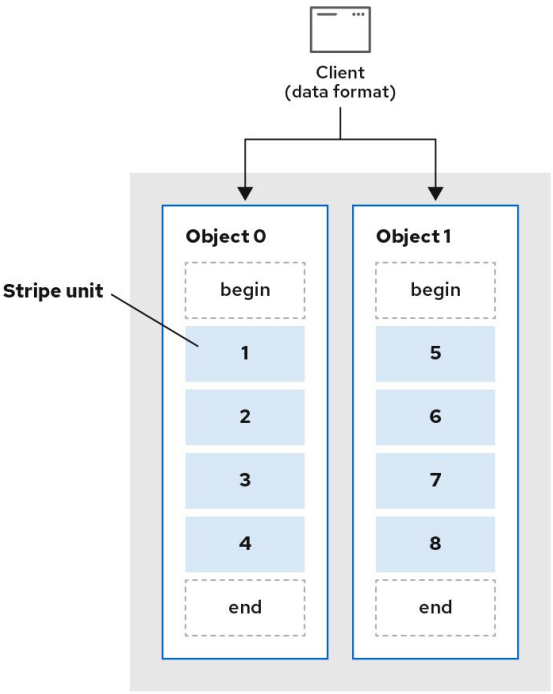
Storage devices have throughput limitations, which impact performance and scalability. As a result, storage systems often support striping to increase throughput and performance. Striping is the storing sequential pieces of information across multiple storage devices. The most common form of data striping comes from RAID. The RAID type most similar to Ceph's striping is RAID 0, or a *striped volume*. Ceph's striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery.

Ceph provides three types of clients: Ceph Block Device, Ceph Filesystem, and Ceph Object Storage. A Ceph Client converts its data from the representation format it provides to its users, such as a block device image, RESTful objects, CephFS filesystem directories, into objects for storage in the Ceph Storage Cluster.

> **Tip:** The objects Ceph stores in the Ceph Storage Cluster are not striped. Ceph Object Storage, Ceph Block Device, and the Ceph Filesystem stripe their data over multiple Ceph Storage Cluster objects. Ceph Clients that write directly to the Ceph storage cluster using `librados` must perform the striping, and parallel I/O for themselves to obtain these benefits.

The simplest Ceph striping format involves a stripe count of 1 object. Ceph Clients write stripe units to a Ceph Storage Cluster object until the object is at its maximum capacity, and then create another object for additional stripes of data. The simplest form of striping may be sufficient for small block device images, S3 or Swift objects. However, this simple form doesn't take maximum advantage of Ceph's ability to distribute data across placement groups, and consequently does not improve performance very much. The following diagram depicts the simplest form of striping:

Figure: Data Striping

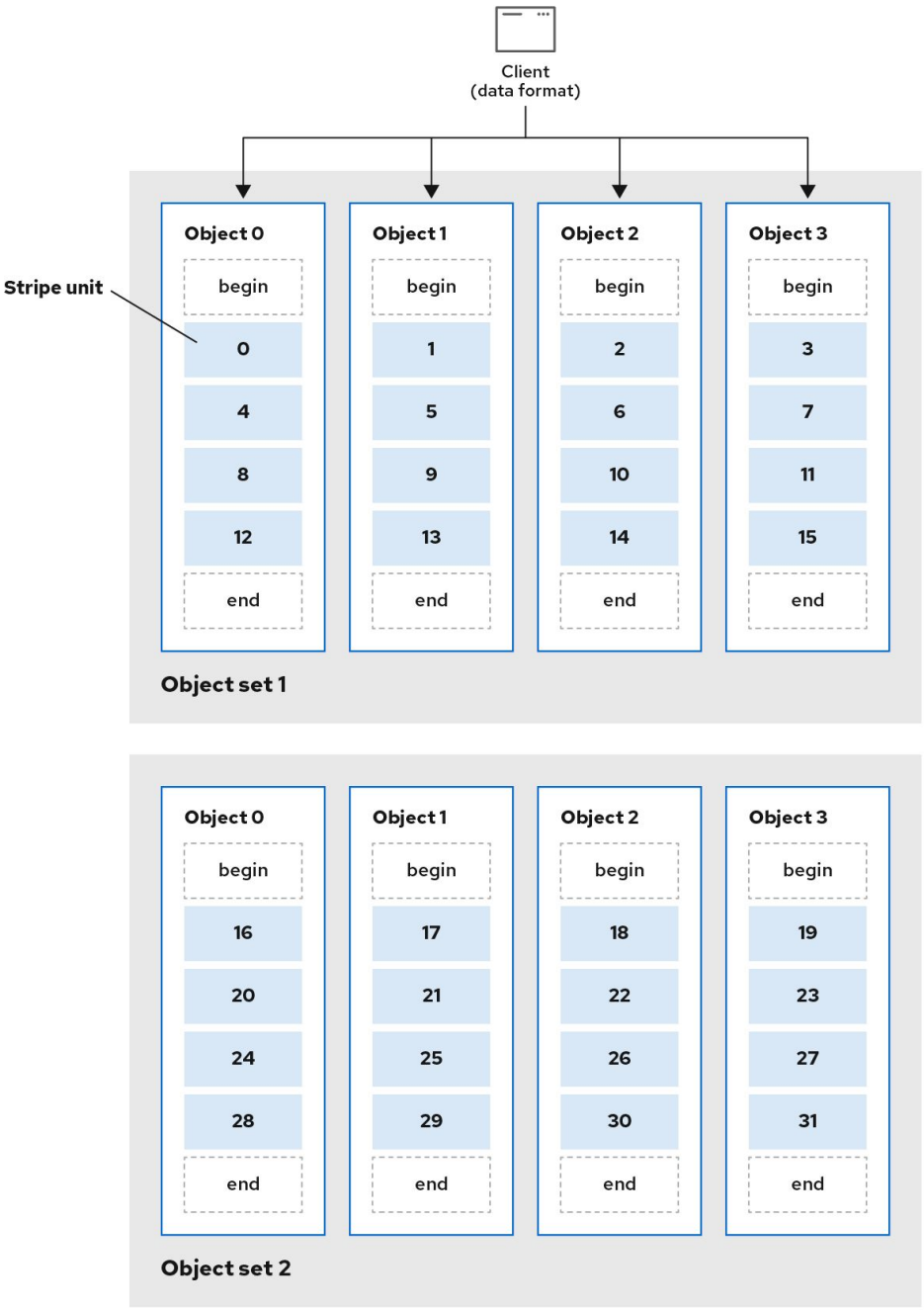


If you anticipate large images sizes, large S3 or Swift objects for example, video, you may see considerable read/write performance improvements by striping client data over multiple objects within an object set. Significant write performance occurs when the client writes the stripe units to their corresponding objects in parallel. Since objects get mapped to different placement groups and further mapped to different OSDs, each write occurs in parallel at the maximum write speed. A write to a single disk would be limited by the head movement for example, 6ms per seek and bandwidth of that one device for example, 100MB/s. By spreading that write over multiple objects, which map to different placement groups and OSDs, Ceph can reduce the number of seeks per drive and combine the throughput of multiple drives to achieve much faster write or read speeds.

> **Note:** Striping is independent of object replicas. Since CRUSH replicates objects across OSDs, stripes get replicated automatically.

In the following diagram, client data gets striped across an object set (`object set 1` in the following diagram) consisting of 4 objects, where the first stripe unit is `stripe unit 0` in `object 0`, and the fourth stripe unit is `stripe unit 3` in `object 3`. After writing the fourth stripe, the client determines if the object set is full. If the object set is not full, the client begins writing a stripe to the first object again, see `object 0` in the following diagram. If the object set is full, the client creates a new object set, see `object set 2` in the following diagram, and begins writing to the first stripe, with a stripe unit of 16, in the first object in the new object set, see `object 4` in the diagram.

Figure: Client Data Stripping



Three important variables determine how Ceph stripes data:

Object Size

Objects in the Ceph Storage Cluster have a maximum configurable size, such as 2 MB, or 4 MB. The object size should be large enough to accommodate many stripe units, and should be a multiple of the stripe unit.

> **Important:** Red Hat recommends a safe maximum value of 16 MB.

Stripe Width

Stripes have a configurable unit size, for example 64 KB. The Ceph Client divides the data it will write to objects into equally sized stripe units, except for the last stripe unit. A stripe width should be a fraction of the Object Size so that an object may contain many stripe units.

Stripe Count

The Ceph Client writes a sequence of stripe units over a series of objects determined by the stripe count. The series of objects is called an object set. After the Ceph Client writes to the last object in the object set, it returns to the first object in the object set.

Important: Test the performance of your striping configuration before putting your cluster into production. You **CANNOT** change these striping parameters after you stripe the data and write it to objects.

Once the Ceph Client has striped data to stripe units and mapped the stripe units to objects, Ceph's CRUSH algorithm maps the objects to placement groups, and the placement groups to Ceph OSD Daemons before the objects are stored as files on a storage disk.

Note: Since a client writes to a single pool, all data striped into objects get mapped to placement groups in the same pool. So they use the same CRUSH map and the same access controls.

Ceph on-wire encryption

You can enable encryption for all Ceph traffic over the network with the introduction of the messenger version 2 protocol. The `secure` mode setting for messenger v2 encrypts communication between Ceph daemons and Ceph clients, giving you end-to-end encryption.

The second version of Ceph's on-wire protocol, `msgr2`, includes several new features:

- A secure mode encrypting all data moving through the network.
- Encapsulation improvement of authentication payloads.
- Improvements to feature advertisement and negotiation.

The Ceph daemons bind to multiple ports allowing both the legacy, v1-compatible, and the new, v2-compatible, Ceph clients to connect to the same storage cluster. Ceph clients or other Ceph daemons connecting to the Ceph Monitor daemon will try to use the `v2` protocol first, if possible, but if not, then the legacy `v1` protocol will be used. By default, both messenger protocols, `v1` and `v2`, are enabled. The new v2 port is 3300, and the legacy v1 port is 6789, by default.

The messenger v2 protocol has two configuration options that control whether the v1 or the v2 protocol is used:

`ms_bind_msgr1`

This option controls whether a daemon binds to a port speaking the v1 protocol; it is `true` by default.

`ms_bind_msgr2`

This option controls whether a daemon binds to a port speaking the v2 protocol; it is `true` by default.

Similarly, two options control based on IPv4 and IPv6 addresses used:

`ms_bind_ipv4`

This option controls whether a daemon binds to an IPv4 address; it is `true` by default.

`ms_bind_ipv6`

This option controls whether a daemon binds to an IPv6 address; it is `true` by default.

The `msgr2` protocol supports two connection modes:

- `crc`
  - Provides strong initial authentication when a connection is established with `cephx`.
  - Provides a `crc32c` integrity check to protect against bit flips.
  - Does not provide protection against a malicious man-in-the-middle attack.
  - Does not prevent an eavesdropper from seeing all post-authentication traffic.
- `secure`
  - Provides strong initial authentication when a connection is established with `cephx`.

- Provides full encryption of all post-authentication traffic.
- Provides a cryptographic integrity check.

The default mode is `crc`.

Ensure that you consider cluster CPU requirements when you plan the Red Hat Ceph Storage deployment to include encryption overhead.

> **Important:** Using `secure` mode is supported by Ceph clients using `librbd`, such as OpenStack Nova, Glance, and Cinder.

Address Changes

For both versions of the messenger protocol to co-exist in the same storage cluster, the address formatting has changed:

Syntax for old address format

```
IP_ADDR:PORT/CLIENT_ID
```

For example, `1.2.3.4:5678/91011`

Syntax for new address format

```
PROTOCOL_VERSION:IP_ADDR:PORT/CLIENT_ID
```

For example, `v2:1.2.3.4:5678/91011`, where *PROTOCOL_VERSION* can be either `v1` or `v2`.

Because the Ceph daemons now bind to multiple ports, the daemons display multiple addresses instead of a single address. Here is an example from a dump of the monitor map:

```
epoch 1
fsid 50fcf227-be32-4bcb-8b41-34ca8370bd17
last_changed 2021-12-12 11:10:46.700821
created 2021-12-12 11:10:46.700821
min_mon_release 14 (nautilus)
0: [v2:10.0.0.10:3300/0,v1:10.0.0.10:6789/0] mon.a
1: [v2:10.0.0.11:3300/0,v1:10.0.0.11:6789/0] mon.b
2: [v2:10.0.0.12:3300/0,v1:10.0.0.12:6789/0] mon.c
```

Also, the `mon_host` configuration option and specifying addresses on the command line using `-m` supports the new address format.

Connection Phases

There are four phases for making an encrypted connection:

Banner

On connection, both the client and the server send a banner. Currently, the Ceph banner is `ceph 0 0n`.

Authentication Exchange

All data, sent or received, is contained in a frame for the duration of the connection. The server decides if authentication has completed, and what the connection mode will be. The frame format is fixed, and can be in three different forms depending on the authentication flags being used.

Message Flow Handshake Exchange

The peers identify each other and establish a session. The client sends the first message, and the server will reply with the same message. The server can close connections if the client talks to the wrong daemon. For new sessions, the client and server proceed to exchanging messages. Client cookies are used to identify a session, and can reconnect to an existing session.

Message Exchange

The client and server start exchanging messages, until the connection is closed.

Reference

See the "Data security and hardening" on page 27 for details on enabling the `msgr2` protocol.

Core Ceph components

A Red Hat Ceph Storage cluster can have a large number of Ceph nodes for limitless scalability, high availability and performance. Use this information to learn how CRUSH enables Ceph to perform seamless operations.

Each node leverages non-proprietary hardware and intelligent Ceph daemons that communicate with each other to:

- Write and read data
- Compress data
- Ensure durability by replicating or erasure coding data
- Monitor and report on cluster health--also called 'heartbeating'
- Redistribute data dynamically--also called 'backfilling'
- Ensure data integrity; and,
- Recover from failures.

To the Ceph client interface that reads and writes data, an Red Hat Ceph Storage cluster looks like a simple pool where it stores data. However, `librados` and the storage cluster perform many complex operations in a manner that is completely transparent to the client interface. Ceph clients and Ceph OSDs both use the CRUSH (Controlled Replication Under Scalable Hashing) algorithm.

Prerequisites

Be sure to have a basic understanding of distributed storage systems before learning about the Ceph client components.

Pools

The Ceph storage cluster stores data objects in logical partitions called *pools*. Ceph administrators can create pools for particular types of data, such as for Ceph Block Devices, Ceph Object Gateways, or simply just to separate one group of users from another. Pools play an important role in data durability, performance, and high availability to Red Hat Ceph Storage.

From the perspective of a Ceph client, the storage cluster is very simple. When a Ceph client reads or writes data using an I/O context, it *always* connects to a storage pool in the Ceph storage cluster. The client specifies the pool name, a user and a secret key, so the pool appears to act as a logical partition with access controls to its data objects.

A Ceph pool is not only a logical partition for storing object data. A pool plays a critical role in how the Ceph storage cluster distributes and stores data. However, these complex operations are transparent to the Ceph client.

Ceph pools define:

Pool type

In early versions of Ceph, a pool simply maintained multiple deep copies of an object. Today, Ceph can maintain multiple copies of an object, or it can use erasure coding to ensure durability. The data durability method is pool-wide, and does not change after creating the pool. The pool type defines the data durability method when creating the pool. Pool types are completely transparent to the client.

Placement groups

In an exabyte scale storage cluster, a Ceph pool might store millions of data objects or more. Ceph must handle many types of operations, including data durability via replicas or erasure code chunks, data integrity by scrubbing or CRC checks, replication, rebalancing and recovery. Consequently, managing data on a per-object basis presents a scalability and performance bottleneck. Ceph addresses this bottleneck by sharding a pool into placement groups. The CRUSH algorithm computes the placement group for storing an object and computes the Acting Set of OSDs for the placement group. CRUSH puts each object into a placement group. Then, CRUSH stores each placement group in a set of OSDs. System administrators set the placement group count when creating or modifying a pool.

CRUSH ruleset

CRUSH can detect failure domains and performance domains. CRUSH can identify OSDs by storage media type and organize OSDs hierarchically into nodes, racks, and rows. CRUSH enables Ceph OSDs to store object copies across failure domains. For example, copies of an object may get stored in different server rooms, aisles, racks and nodes. If a large part of a cluster fails, such as a rack, the cluster can still operate in a degraded state until the cluster recovers.
Additionally, CRUSH enables clients to write data to particular types of hardware, such as SSDs, hard drives with SSD journals, or hard drives with journals on the same drive as the data. The CRUSH ruleset determines failure domains and performance domains for the pool. Administrators set the CRUSH ruleset when creating a pool.

> **Note:** An administrator CANNOT change a pool's ruleset after creating the pool.

Durability

In exabyte scale storage clusters, hardware failure is an expectation and not an exception. When using data objects to represent larger grained storage interfaces such as a block device, losing one or more data objects for that larger grained interface can compromise the integrity of the larger grained storage entity potentially rendering it useless. So data loss is intolerable. Ceph provides high data durability in two ways:

- Replica pools store multiple deep copies of an object using the CRUSH failure domain to physically separate one data object copy from another. That is, copies get distributed to separate physical hardware. This increases durability during hardware failures.
- Erasure coded pools store each object as `K+M` chunks, where `K` represents data chunks and `M` represents coding chunks. The sum represents the number of OSDs used to store the object and the `M` value represents the number of OSDs that can fail and still restore data should the `M` number of OSDs fail.

Ceph authentication

To identify users and protect against man-in-the-middle attacks, Ceph provides its `cephx` authentication system, which authenticates users and daemons.

> **Note:** The `cephx` protocol does not address data encryption for data transported over the network or data stored in OSDs.

Cephx uses shared secret keys for authentication, meaning both the client and the monitor cluster have a copy of the client's secret key. The authentication protocol enables both parties to prove to each other that they have a copy of the key without actually revealing it. This provides mutual authentication, which means the cluster is sure the user possesses the secret key, and the user is sure that the cluster has a copy of the secret key.

Cephx

The `cephx` authentication protocol operates in a manner similar to Kerberos.

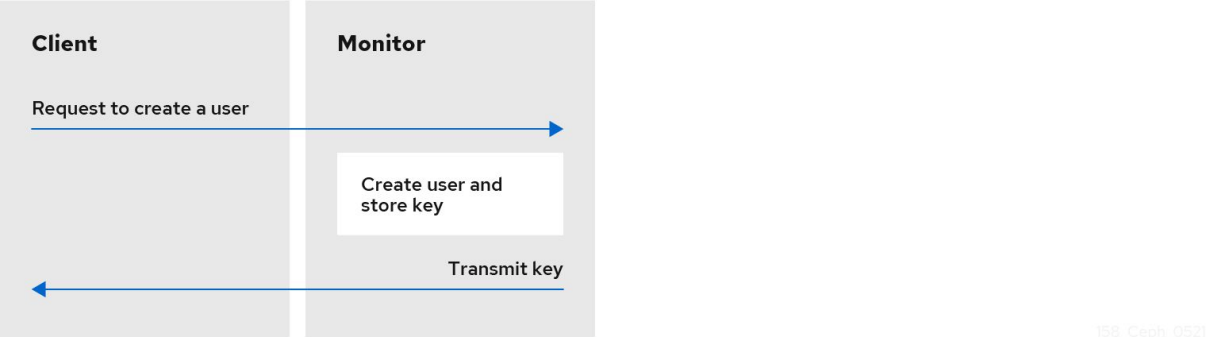
A user/actor invokes a Ceph client to contact a monitor. Unlike Kerberos, each monitor can authenticate users and distribute keys, so there is no single point of failure or bottleneck when using `cephx`. The monitor returns an authentication data structure similar to a Kerberos ticket that contains a session key for use in obtaining Ceph services. This session key is itself encrypted with the user's permanent secret key, so that only the user can request services from the Ceph monitors. The client then uses the session key to request its desired services from the monitor, and the monitor provides the client with a ticket that will authenticate the client to the OSDs that actually handle data. Ceph monitors and OSDs share a secret, so the client can use the ticket provided by the monitor with any OSD or metadata server in the cluster. Like Kerberos, `cephx` tickets expire, so an attacker cannot use an expired ticket or session key obtained surreptitiously. This form of authentication will prevent attackers with access to the communications medium from either creating bogus messages under another user's identity or altering another user's legitimate messages, as long as the user's secret key is not divulged before it expires.

To use `cephx`, an administrator must set up users first. In the following diagram, the `client.admin` user invokes `ceph auth get-or-create-key` from the command line to generate a username and secret key. Ceph's `auth`

subsystem generates the username and key, stores a copy with the monitor(s) and transmits the user's secret back to the `client.admin` user. This means that the client and the monitor share a secret key.

> **Note:** The `client.admin` user must provide the user ID and secret key to the user in a secure manner.

Figure: CephX



Placement groups

Storing millions of objects in a cluster and managing them individually is resource intensive. Ceph uses placement groups (PGs) to make managing a huge number of objects more efficient.

A PG is a subset of a pool that serves to contain a collection of objects. Ceph shards a pool into a series of PGs. Then, the CRUSH algorithm takes the cluster map and the status of the cluster into account and distributes the PGs evenly and pseudo-randomly to OSDs in the cluster.

Here is how it works:

When a system administrator creates a pool, CRUSH creates a user-defined number of PGs for the pool. Generally, the number of PGs should be a reasonably fine-grained subset of the data. For example, 100 PGs per OSD per pool would mean that each PG contains approximately 1% of the pool's data.

The number of PGs has a performance impact when Ceph needs to move a PG from one OSD to another OSD. If the pool has too few PGs, Ceph will move a large percentage of the data simultaneously and the network load will adversely impact the cluster's performance. If the pool has too many PGs, Ceph will use too much CPU and RAM when moving tiny percentages of the data and thereby adversely impact the cluster's performance. For details on calculating the number of PGs to achieve optimal performance, see PG Count

Ceph ensures against data loss by storing replicas of an object or by storing erasure code chunks of an object. Since Ceph stores objects or erasure code chunks of an object within PGs, Ceph replicates each PG in a set of OSDs called the *Acting Set* for each copy of an object or each erasure code chunk of an object. A system administrator can determine the number of PGs in a pool and the number of replicas or erasure code chunks. However, the CRUSH algorithm calculates which OSDs are in the acting set for a particular PG.

The CRUSH algorithm and PGs make Ceph dynamic. Changes in the cluster map or the cluster state may result in Ceph moving PGs from one OSD to another automatically.

Here are a few examples:

- **Expanding the Cluster:** When adding a new host and its OSDs to the cluster, the cluster map changes. Since CRUSH evenly and pseudo-randomly distributes PGs to OSDs throughout the cluster, adding a new host and its OSDs means that CRUSH will reassign some of the pool's placement groups to those new OSDs. That means that system administrators do not have to rebalance the cluster manually. Also, it means that the new OSDs contain approximately the same amount of data as the other OSDs. This also means that new OSDs do not contain newly written OSDs, preventing *hot spots* in the cluster.
- **An OSD Fails:** When an OSD fails, the state of the cluster changes. Ceph temporarily loses one of the replicas or erasure code chunks, and needs to make another copy. If the primary OSD in the acting set fails, the next OSD in the acting set becomes the primary and CRUSH calculates a new OSD to store the additional copy or erasure code chunk.

By managing millions of objects within the context of hundreds to thousands of PGs, the Ceph storage cluster can grow, shrink and recover from failure efficiently.

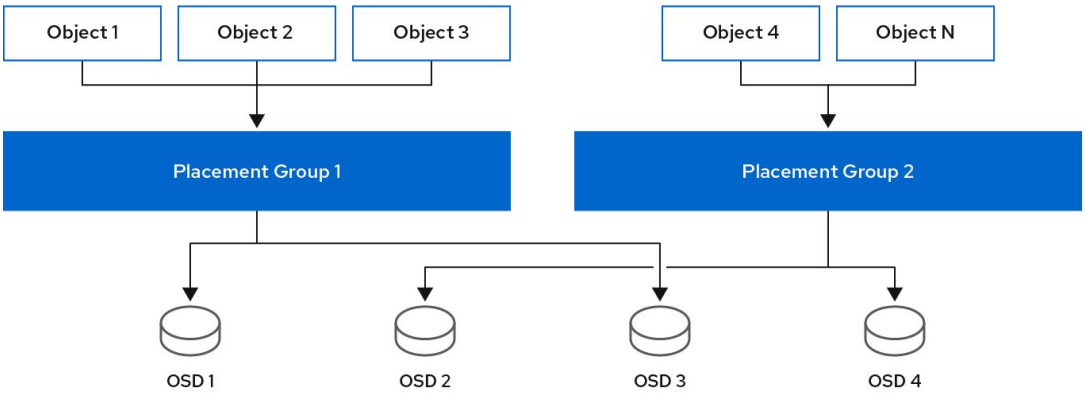
For Ceph clients, the CRUSH algorithm via `librados` makes the process of reading and writing objects very simple. A Ceph client simply writes an object to a pool or reads an object from a pool. The primary OSD in the acting set can write replicas of the object or erasure code chunks of the object to the secondary OSDs in the acting set on behalf of the Ceph client.

If the cluster map or cluster state changes, the CRUSH computation for which OSDs store the PG will change too. For example, a Ceph client may write object `foo` to the pool `bar`. CRUSH will assign the object to PG `1.a`, and store it on `OSD 5`, which makes replicas on `OSD 10` and `OSD 15` respectively. If `OSD 5` fails, the cluster state changes. When the Ceph client reads object `foo` from pool `bar`, the client via `librados` will automatically retrieve it from `OSD 10` as the new primary OSD dynamically.

The Ceph client via `librados` connects directly to the primary OSD within an acting set when writing and reading objects. Since I/O operations do not use a centralized broker, network oversubscription is typically NOT an issue with Ceph.

The following diagram depicts how CRUSH assigns objects to PGs, and PGs to OSDs. The CRUSH algorithm assigns the PGs to OSDs such that each OSD in the acting set is in a separate failure domain, which typically means the OSDs will always be on separate server hosts and sometimes in separate racks.

Figure: Placement Groups



CRUSH ruleset

Ceph assigns a CRUSH ruleset to a pool. When a Ceph client stores or retrieves data in a pool, Ceph identifies the CRUSH ruleset, a rule within the rule set, and the top-level bucket in the rule for storing and retrieving data. As Ceph processes the CRUSH rule, it identifies the primary OSD that contains the placement group for an object. That enables the client to connect directly to the OSD, access the placement group and read or write object data.

To map placement groups to OSDs, a CRUSH map defines a hierarchical list of bucket types. The list of bucket types are located under `types` in the generated CRUSH map. The purpose of creating a bucket hierarchy is to segregate the leaf nodes by their failure domains and/or performance domains, such as drive type, hosts, chassis, racks, power distribution units, pods, rows, rooms, and data centers.

With the exception of the leaf nodes representing OSDs, the rest of the hierarchy is arbitrary. Administrators may define it according to their own needs if the default types don't suit their requirements. CRUSH supports a directed acyclic graph that models the Ceph OSD nodes, typically in a hierarchy. So Ceph administrators can support multiple hierarchies with multiple root nodes in a single CRUSH map. For example, an administrator can create a hierarchy representing higher cost SSDs for high performance, and a separate hierarchy of lower cost hard drives with SSD journals for moderate performance.

Input/output operations

Ceph clients retrieve a *Cluster Map* from a Ceph monitor, bind to a pool, and perform input/output (I/O) on objects within placement groups in the pool. The pool's CRUSH ruleset and the number of placement groups are the main factors that determine how Ceph will place the data.

With the latest version of the cluster map, the client knows about all of the monitors and OSDs in the cluster and their current state. **However, the client doesn't know anything about object locations.**

The only inputs required by the client are the object ID and the pool name. It is simple: Ceph stores data in named pools. When a client wants to store a named object in a pool it takes the object name, a hash code, the number of PGs in the pool and the pool name as inputs; then, CRUSH (Controlled Replication Under Scalable Hashing) calculates the ID of the placement group and the primary OSD for the placement group.

Ceph clients use the following steps to compute PG IDs.

1. The client inputs the pool ID and the object ID. For example, `pool = liverpool` and `object-id = john`.
2. CRUSH takes the object ID and hashes it.
3. CRUSH calculates the hash modulo of the number of PGs to get a PG ID. For example, `58`.
4. CRUSH calculates the primary OSD corresponding to the PG ID.
5. The client gets the pool ID given the pool name. For example, the pool `liverpool` is pool number `4`.
6. The client prepends the pool ID to the PG ID. For example, `4.58`.
7. The client performs an object operation such as write, read, or delete by communicating directly with the Primary OSD in the Acting Set.

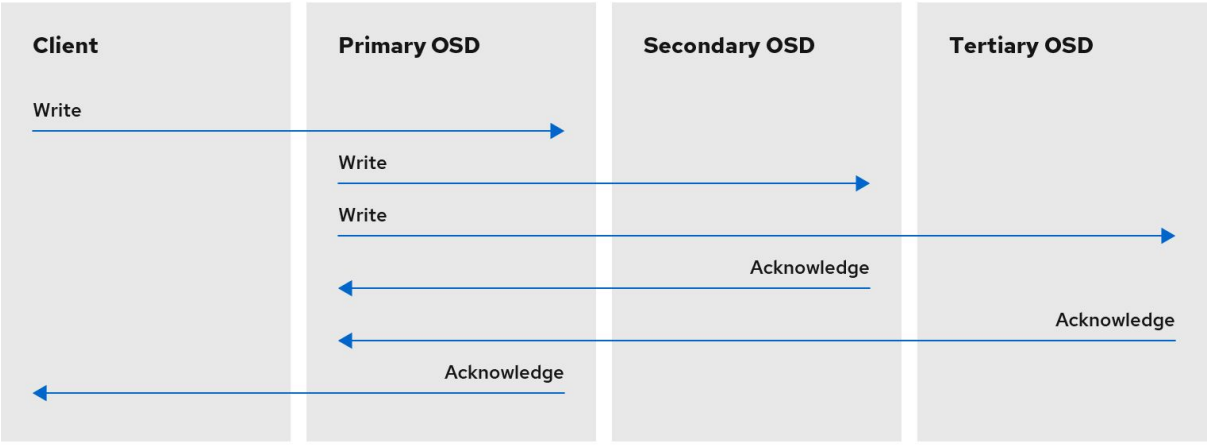
The topology and state of the Ceph storage cluster are relatively stable during a session. Empowering a Ceph client via `librados` to compute object locations is much faster than requiring the client to make a query to the storage cluster over a chatty session for each read/write operation. The CRUSH algorithm allows a client to compute where objects *should* be stored, and **enables the client to contact the primary OSD in the acting set directly** to store or retrieve data in the objects. Since a cluster at the exabyte scale has thousands of OSDs, network oversubscription between a client and a Ceph OSD is not a significant problem. If the cluster state changes, the client can simply request an update to the cluster map from the Ceph monitor.

Replication

Similar to Ceph clients, Ceph OSDs can contact Ceph monitors to retrieve the latest copy of the cluster map. Ceph OSDs also use the CRUSH algorithm, but they use it to compute where to store replicas of objects. In a typical write scenario, a Ceph client uses the CRUSH algorithm to compute the placement group ID and the primary OSD in the Acting Set for an object. When the client writes the object to the primary OSD, the primary OSD finds the number of replicas that it should store.

The value is found in the `osd_pool_default_size` setting. Then, the primary OSD takes the object ID, pool name and the cluster map and uses the CRUSH algorithm to calculate the IDs of secondary OSDs for the acting set. The primary OSD writes the object to the secondary OSDs. When the primary OSD receives an acknowledgment from the secondary OSDs and the primary OSD itself completes its write operation, it acknowledges a successful write operation to the Ceph client.

Figure: Replicated IO



With the ability to perform data replication on behalf of Ceph clients, Ceph OSD Daemons relieve Ceph clients from that duty, while ensuring high data availability and data safety.

Note: The primary OSD and the secondary OSDs are typically configured to be in separate failure domains. CRUSH computes the IDs of the secondary OSDs with consideration for the failure domains.

Data copies

In a replicated storage pool, Ceph needs multiple copies of an object to operate in a degraded state. Ideally, a Ceph storage cluster enables a client to read and write data even if one of the OSDs in an acting set fails. For this reason, Ceph defaults to making three copies of an object with a minimum of two copies clean for write operations. Ceph will still preserve data even if two OSDs fail. However, it will interrupt write operations.

In an erasure-coded pool, Ceph needs to store chunks of an object across multiple OSDs so that it can operate in a degraded state. Similar to replicated pools, ideally an erasure-coded pool enables a Ceph client to read and write in a degraded state.

Important: The following *jerasure* coding values for *k*, and *m* are supported:

- k=8 m=3
- k=8 m=4
- k=4 m=2

Erasure-coding

Ceph can load one of many erasure code algorithms. The earliest and most commonly used is the `Reed-Solomon` algorithm. An erasure code is a forward error correction (FEC) code. FEC code transforms a message of `K` chunks into a longer message that is called a code word of `N` chunks, such that Ceph can recover the original message from a subset of the `N` chunks.

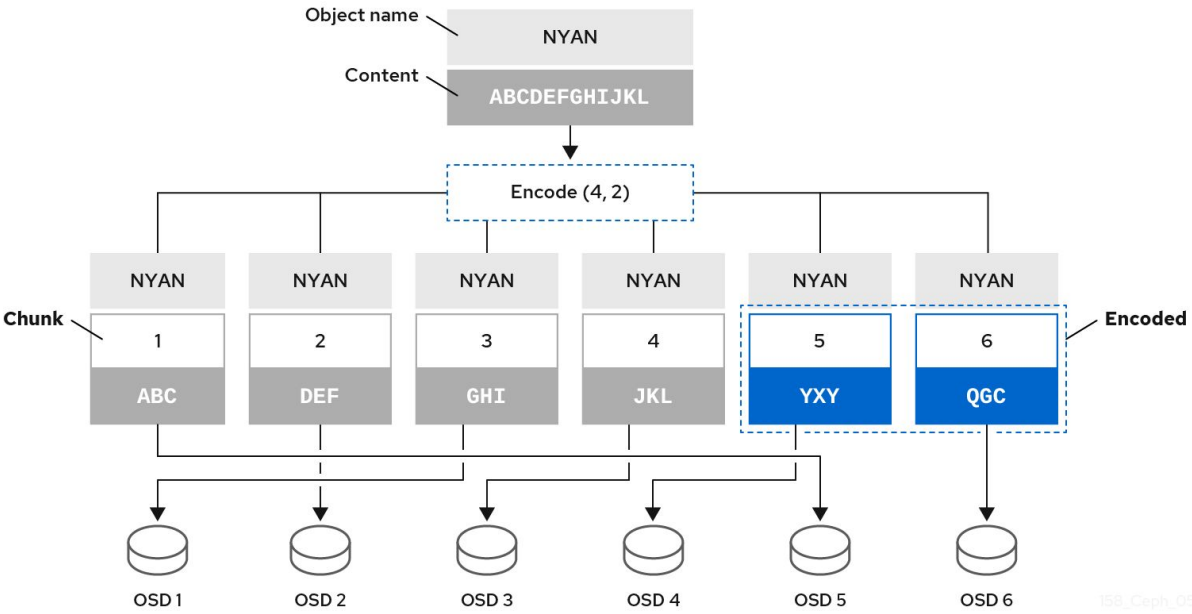
More specifically, `N = K+M` where the variable `K` is the original number of data chunks. The variable `M` stands for the extra or redundant chunks that the erasure code algorithm adds to provide protection from failures. The variable `N` is the total number of chunks that are created after the erasure coding process. The value of `M` is `N-K` which means that the algorithm computes `N-K` redundant chunks from `K` original data chunks. This approach ensures that that Ceph can access all the original data. The system is resilient to arbitrary `N-K` failures. For instance, in a 10 `K` of 16 `N` configuration, or erasure coding `10/16`, the erasure code algorithm adds six extra chunks to the 10 base chunks `K`. For example, in a `M = K-N` or `16-10 = 6` configuration, Ceph will spread the 16 chunks `N` across 16 OSDs. The original file might be reconstructed from the 10 verified `N` chunks even if 6 OSDs fail—ensuring that the Red Hat Ceph Storage cluster will not lose data, and thereby ensures a high level of fault tolerance.

Like replicated pools, in an erasure-coded pool, the primary OSD in the up set receives all write operations. In replicated pools, Ceph makes a deep copy of each object in the placement group on the secondary OSDs in the set. For erasure coding, the process is a bit different. An erasure coded pool stores each object as `K+M` chunks. It is divided into `K` data chunks and `M` coding chunks. The pool is configured to have a size of `K+M` so that Ceph stores each chunk in an OSD in the acting set. Ceph stores the rank of the chunk as an attribute of the object. The primary OSD is responsible for encoding the payload into `K+M` chunks and sends them to the other OSDs. The primary OSD is also responsible for maintaining an authoritative version of the placement group logs.

For example, in a typical configuration, a system administrator creates an erasure-coded pool to use six OSDs and sustain the loss of two of them. That is, (`K+M = 6`) such that (`M = 2`).

When Ceph writes the object **NYAN** containing `ABCDEFGHIJKL` to the pool, the erasure encoding algorithm splits the content into four data chunks by simply dividing the content into four parts: `ABC`, `DEF`, `GHI`, and `JKL`. The algorithm will pad the content if the content length is not a multiple of `K`. The function also creates two coding chunks: the fifth with `YXY` and the sixth with `QGC`. Ceph stores each chunk on an OSD in the acting set with a shard_id corresponding to the acting set position, where it stores the chunks in objects that have the same name, **NYAN**, but reside on different OSDs. For example, Chunk 1 contains `ABC` and Ceph stores it on **OSD5** while chunk 5 contains `YXY` and Ceph stores it on **OSD4**.

Figure: Ceph client object watch and notify



In a recovery scenario, the client attempts to read the object **NYAN** from the erasure-coded pool by reading chunks 1 through 6. The OSD informs the algorithm that chunks 2 and 6 are missing. For example, the primary OSD might not read chunk 6 because the **OSD6** is out, and might not read chunk 2, because **OSD2** was the slowest and its chunk was not taken into account. However, when the algorithm has four chunks, it reads the four chunks: chunk 1 containing `ABC`, chunk 3 containing `GHI`, chunk 4 containing `JKL`, and chunk 5 containing `YXY`. Then, it rebuilds the original content of the object `ABCDEFGHIJKL`, and original content of chunk 6, which contained `QGC`.

Splitting data into chunks is independent from object placement. The CRUSH ruleset along with the erasure-coded pool profile determines the placement of chunks on the OSDs. For instance, that uses the Locally Repairable Code (`lrc`) plug-in in the erasure code profile creates extra chunks and requires fewer OSDs to recover from. For example, in an `lrc` profile configuration `K=4 M=2 L=3`, the algorithm creates six chunks (`K+M`), just as the `jerasure` plug-in would, but the locality value (`L=3`) requires that the algorithm create 2 more chunks locally. The algorithm creates the additional chunks as such, `(K+M)/L`. If the OSD containing chunk 0 fails, this chunk can be recovered by using chunks 1, 2 and the first local chunk. In this case, the algorithm only requires 3 chunks for recovery instead of 5.

Note: Using erasure-coded pools disables Object Map.

Important: For an erasure-coded pool with 2+2 configuration, replace the input string from ABCDEFGHIJKL to ABCDEF and replace the coding chunks from 4 to 2.

Multi-step-retry (MSR) is a type of CRUSH rule in the Ceph cluster that defines how data is distributed across storage devices. MSR ensures efficient data retrieval, balancing, and fault tolerance. For more information, see “Creating CRUSH rules” on page 0

Reference

- For more information about CRUSH, the erasure-coding profiles, and plug-ins, see Storage strategies.
- For more details on Object Map, see Ceph client object map.

`ObjectStore`

`ObjectStore` provides a low-level interface to an OSD’s raw block device. When a client reads or writes data, it interacts with the `ObjectStore` interface. Ceph write operations are essentially ACID transactions: that is, they provide *Atomicity*, *Consistency*, *Isolation*, and *Durability*.

`ObjectStore` ensures that a `Transaction` is all-or-nothing to provide Atomicity. The `ObjectStore` also handles object semantics. An object stored in the storage cluster has a unique identifier, object data and metadata. So `ObjectStore` provides `Consistency` by ensuring that Ceph object semantics are correct. `ObjectStore` also provides the **Isolation** portion of an ACID transaction by invoking a `Sequencer` on write operations to ensure that Ceph write operations occur sequentially. In contrast, an OSDs replication or erasure coding functionality provides the **Durability** component of the ACID transaction. Since `ObjectStore` is a low-level interface to storage media, it also provides performance statistics.

Ceph implements several concrete methods for storing data:

BlueStore

A production grade implementation using a raw block device to store object data.

Memstore

A developer implementation for testing read/write operations directly in RAM.

K/V Store

An internal implementation for Ceph's use of key/value databases.

Since administrators will generally only address `BlueStore`, the following sections will only describe those implementations in greater detail.

BlueStore

`BlueStore` is the next generation storage implementation for Ceph. As the market for storage devices now includes solid state drives or SSDs and non-volatile memory over PCI Express or NVMe, their use in Ceph reveals some of the limitations of the `FileStore` storage implementation. While `FileStore` has many improvements to facilitate SSD and NVMe storage, other limitations remain. Among them, increasing placement groups remains computationally expensive, and the double write penalty remains. Whereas, `FileStore` interacts with a file system on a block device, `BlueStore` eliminates that layer of indirection and directly consumes a raw block device for object storage. `BlueStore` uses the very light weight `BlueFS` file system on a small partition for its k/v databases. `BlueStore` eliminates the paradigm of a directory representing a placement group, a file representing an object and file XATTRs representing metadata. `BlueStore` also eliminates the double write penalty of `FileStore`, so write operations are nearly twice as fast with `BlueStore` under most workloads.

`BlueStore` stores data as:

Object Data

In `BlueStore`, Ceph stores objects as blocks directly on a raw block device. The portion of the raw block device that stores object data does NOT contain a filesystem. The omission of the filesystem eliminates a layer of indirection and thereby improves performance. However, much of the `BlueStore` performance improvement comes from the block database and write-ahead log.

Block Database

In `BlueStore`, the block database handles the object semantics to guarantee **Consistency**. An object's unique identifier is a key in the block database. The values in the block database consist of a series of block addresses that refer to the stored object data, the object's placement group, and object metadata. The block database may reside on a `BlueFS` partition on the same raw block device that stores the object data, or it may reside on a separate block device, usually when the primary block device is a hard disk drive and an SSD or NVMe will improve performance. The block database provides a number of improvements over `FileStore`; namely, the key/value semantics of `BlueStore` do not suffer from the limitations of filesystem XATTRs. `BlueStore` may assign objects to other placement groups quickly within the block database without the overhead of moving files from one directory to another, as is the case in `FileStore`. `BlueStore` also introduces new features. The block database can store the checksum of the stored object data and its metadata, allowing full data checksum operations for each read, which is more efficient than periodic scrubbing to detect bit rot. `BlueStore` can compress an object and the block database can store the algorithm used to compress an object—ensuring that read operations select the appropriate algorithm for decompression.

Write-ahead Log

In `BlueStore`, the write-ahead log ensures **Atomicity**, similar to the journaling functionality of `FileStore`. Like `FileStore`, `BlueStore` logs all aspects of each transaction. However, the `BlueStore` write-ahead log or WAL can perform this function simultaneously, which eliminates the double write penalty of `FileStore`.

Consequently, `BlueStore` is nearly twice as fast as `FileStore` on write operations for most workloads. BlueStore can deploy the WAL on the same device for storing object data, or it may deploy the WAL on another device, usually when the primary block device is a hard disk drive and an SSD or NVMe will improve performance.

Note: It is only helpful to store a block database or a write-ahead log on a separate block device if the separate device is faster than the primary storage device. For example, SSD and NVMe devices are generally faster than HDDs. Placing the block database and the WAL on separate devices may also have performance benefits due to differences in their workloads.

Self management operations

Ceph clusters perform a lot of self monitoring and management operations automatically.

Ceph OSDs can, for example, check the cluster health and report back to the Ceph monitors. By using CRUSH to assign objects to placement groups and placement groups to a set of OSDs, Ceph OSDs can use the CRUSH algorithm to rebalance the cluster or recover from OSD failures dynamically.

Heartbeat

Ceph OSDs join a cluster and report to Ceph Monitors on their status.

At the lowest level, the Ceph OSD status is `up` or `down` reflecting whether or not it is running and able to service Ceph client requests. If a Ceph OSD is `down` and `in` the Ceph storage cluster, this status may indicate the failure of the Ceph OSD. If a Ceph OSD is not running for example, it crashes the Ceph OSD cannot notify the Ceph Monitor that it is `down`. The Ceph Monitor can ping a Ceph OSD daemon periodically to ensure that it is running. However, heartbeating also empowers Ceph OSDs to determine if a neighboring OSD is `down`, to update the cluster map and to report it to the Ceph Monitors. This means that Ceph Monitors can remain light weight processes.

Peering

Ceph stores copies of placement groups on multiple OSDs. Each copy of a placement group has a status. These OSDs *peer* check each other to ensure that they agree on the status of each copy of the placement group. Peering issues usually resolve themselves.

Note: When Ceph monitors agree on the state of the OSDs storing a placement group, that does not mean that the placement group has the latest contents.

When Ceph stores a placement group in an acting set of OSDs, refer to them as *Primary*, *Secondary*, and so forth. By convention, the *Primary* is the first OSD in the *Acting Set*. The *Primary* that stores the first copy of a placement group is responsible for coordinating the peering process for that placement group. The *Primary* is the **ONLY** OSD that will accept client-initiated writes to objects for a given placement group where it acts as the *Primary*.

An *Acting Set* is a series of OSDs that are responsible for storing a placement group. An *Acting Set* may refer to the Ceph OSD Daemons that are currently responsible for the placement group, or the Ceph OSD Daemons that were responsible for a particular placement group as of some epoch.

The Ceph OSD daemons that are part of an *Acting Set* may not always be `up`. When an OSD in the *Acting Set* is `up`, it is part of the *Up Set*. The *Up Set* is an important distinction, because Ceph can remap PGs to other Ceph OSDs when an OSD fails.

Note: In an *Acting Set* for a PG containing `osd.25`, `osd.32` and `osd.61`, the first OSD, `osd.25`, is the *Primary*. If that OSD fails, the *Secondary*, `osd.32`, becomes the *Primary*, and Ceph will remove `osd.25` from the *Up Set*.

Rebalancing and recovery

When an administrator adds a Ceph OSD to a Ceph storage cluster, Ceph updates the cluster map. This change to the cluster map also changes object placement, because the modified cluster map changes an input for the CRUSH calculations. CRUSH places data evenly, but pseudo randomly. So only a small amount of data moves when an administrator adds a new OSD. The amount of data is usually the number of new OSDs divided by the total amount of data in the cluster. For example, in a cluster with 50 OSDs, 1/50th or 2% of the data might move when adding an OSD.

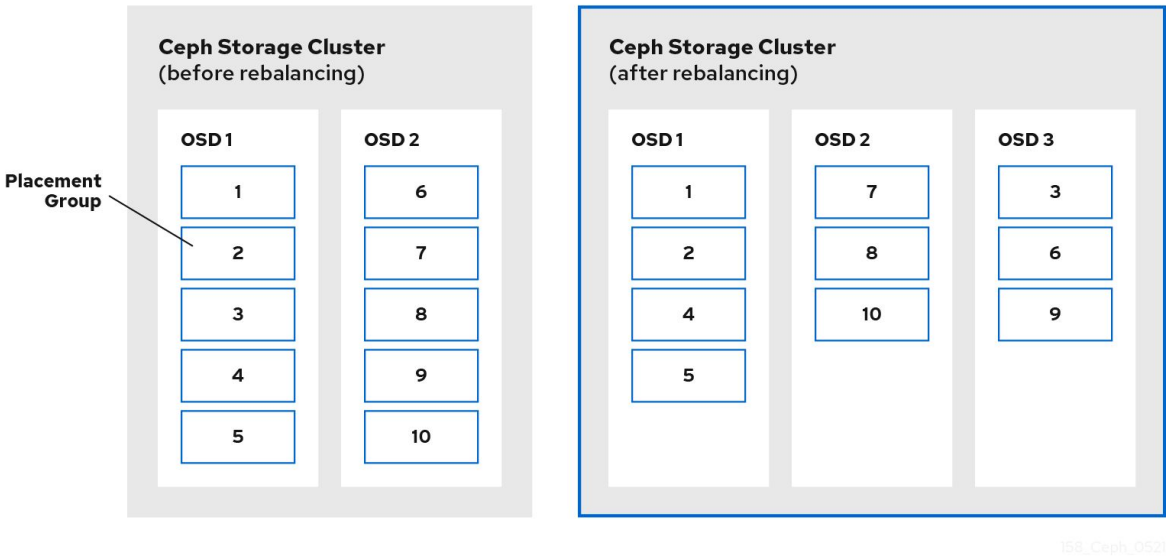
"Figure: Rebalancing and recovery" on page 26 depicts the rebalancing process where some, but not all of the PGs migrate from existing OSDs, OSD 1 and 2 in the diagram, to the new OSD, OSD 3, in the diagram. Even when rebalancing, CRUSH is stable. Many of the placement groups remain in their original configuration, and each OSD gets some added capacity, so there are no load spikes on the new OSD after the cluster rebalances.

There are 2 types of balancer in Ceph: capacity balancing and read balancing.

Capacity balancing

Capacity balancing is a functional need. When one device is full, the system can not take write requests anymore. To avoid filling up devices, it is important to balance capacity across the devices in a fair way. Each device must get a capacity proportional to its size so all devices have the same fullness level. Capacity balancing creates fair share workloads on the OSDs for write requests from a performance perspective. Capacity balancing creates fair share workloads on the OSDs for write requests from a performance perspective. Capacity balancing requires data movement and is a time-consuming operation as it takes time to balance the system.

Figure: Rebalancing and recovery



For optimal write performance, ensure that all devices are homogeneous (same size and performance).

> **Note:** You can run the optimizer on heterogeneous devices, but the results would not be optimal.

Read balancing (Technology Preview)

> **Important:** Technology Preview features are not supported with production service level agreements (SLAs), might not be functionally complete, and does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

Read balancing is a performance need. It helps the system perform better by ensuring that each device gets its fair share of primary OSDs so read requests get distributed across OSDs in the cluster, evenly.

Unbalanced read requests lead to bad performance under load due to the weakest link in the chain effect and to reduced cluster read bandwidth. Read balancing is cheap and the operation is fast as there is no data movement involved. It is a metadata operation, where the osdmap is updated to change which participating OSD in a PG is primary.

Read balancing supports replicated pools only, erasure coded pools are not supported. Read balancing does not take into account the device class of the OSDs and the availability zones of DR solutions. An offline tool is available to use the read balancing feature and you need to run the procedure on each pool of the cluster. You need to run the read balancing procedure again after every autoscale change.

For optimal read performance, ensure that all devices are homogeneous (same size and performance) and that you have balanced the capacity.

> **Note:** You can run the optimizer on heterogeneous devices, but the results would not be optimal.

Data integrity

As part of maintaining data integrity, Ceph provides numerous mechanisms to guard against bad disk sectors and bit rot.

Scrubbing

Ceph OSD Daemons can scrub objects within placement groups. That is, Ceph OSD Daemons can compare object metadata in one placement group with its replicas in placement groups stored on other OSDs. Scrubbing usually performed daily catches bugs or storage errors. Ceph OSD Daemons also perform deeper scrubbing by comparing data in objects bit-for-bit. Deep scrubbing usually performed weekly finds bad sectors on a drive that weren't apparent in a light scrub.

CRC Checks

When using `BlueStore`, Ceph can ensure data integrity by conducting a cyclical redundancy check (CRC) on write operations; then, store the CRC value in the block database. On read operations, Ceph can retrieve the CRC value from the block database and compare it with the generated CRC of the retrieved data to ensure data integrity instantly.

High availability

In addition to the high scalability enabled by the CRUSH algorithm, Ceph must also maintain high availability.

Maintaining high availability means that Ceph clients must be able to read and write data even when the cluster is in a degraded state, or when a monitor fails.

Clustering the Ceph Monitor

For added reliability and fault tolerance, Ceph supports a cluster of monitors.

Before Ceph clients can read or write data, they must contact a Ceph Monitor to obtain the most recent copy of the cluster map. A Red Hat Ceph Storage cluster can operate with a single monitor; however, this introduces a single point of failure. That is, if the monitor goes down, Ceph clients cannot read or write data.

In a cluster of Ceph Monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. For this reason, Ceph must have agreement among various monitor instances regarding the state of the storage cluster. Ceph always uses a majority of monitors and the Paxos algorithm to establish a consensus among the monitors about the current state of the storage cluster. Ceph Monitors nodes require NTP to prevent clock drift.

Storage administrators usually deploy Ceph with an odd number of monitors so determining a majority is efficient. For example, a majority may be 1, 2:3, 3:5, 4:6, and so forth.

Data security and hardening

Use this information to learn about data security and hardening information for Red Hat Ceph Storage Clusters and their clients. The information here also provides advice and good practices information for hardening the security of Red Hat Ceph Storage, with a focus on the Ceph Orchestrator using `cephadm` for Red Hat Ceph Storage deployments.

Important: While following these instructions helps harden the security of your environment, security and compliance is not guaranteed from following these recommendations.

Security is an important concern and should be a strong focus of any Red Hat Ceph Storage deployment. Data breaches and downtime are costly and difficult to manage, laws may require passing audits and compliance processes, and projects have an expectation of a certain level of data privacy and security. The information in this section provides a general introduction to security for Red Hat Ceph Storage, as well as the role of Red Hat in supporting your system's security.

This information provides only a general introduction to security for Red Hat Ceph Storage. Contact Red Hat Support for additional help.

Introduction to Red Hat Ceph Storage

Red Hat Ceph Storage is a highly scalable and reliable object storage solution, which is typically deployed in conjunction with cloud computing solutions like OpenStack, as a standalone storage service, or as network attached storage using interfaces.

All Red Hat Ceph Storage deployments consist of a storage cluster commonly referred to as the Ceph Storage Cluster or RADOS (Reliable Autonomous Distributed Object Store), which consists of three types of daemons:

- **Ceph Monitors (`ceph-mon`):** Ceph monitors provide a few critical functions such as establishing an agreement about the state of the cluster, maintaining a history of the state of the cluster such as whether an OSD is up and running and in the cluster, providing a list of pools through which clients write and read data, and providing authentication for clients and the Ceph Storage Cluster daemons.
- **Ceph Managers (`ceph-mgr`):** Ceph manager daemons track the status of peering between copies of placement groups distributed across Ceph OSDs, a history of the placement group states, and metrics about the Ceph cluster. They also provide interfaces for external monitoring and management systems.
- **Ceph OSDs (`ceph-osd`):** Ceph Object Storage Daemons (OSDs) store and serve client data, replicate client data to secondary Ceph OSD daemons, track and report to Ceph Monitors on their health and on the health of neighboring OSDs, dynamically recover from failures, and backfill data when the cluster size changes, among other functions.

All Red Hat Ceph Storage deployments store end-user data in the Ceph Storage Cluster or RADOS (Reliable Autonomous Distributed Object Store). Generally, users **DO NOT** interact with the Ceph Storage Cluster directly; rather, they interact with a Ceph client.

There are three primary Ceph Storage Cluster clients:

- **Ceph Object Gateway (`radosgw`):** The Ceph Object Gateway, also known as RADOS Gateway, `radosgw` or `rgw` provides an object storage service with RESTful APIs. Ceph Object Gateway stores data on behalf of its clients in the Ceph Storage Cluster or RADOS.
- **Ceph Block Device (`rbd`):** The Ceph Block Device provides copy-on-write, thin-provisioned, and cloneable virtual block devices to a Linux kernel via Kernel RBD (`krbd`) or to cloud computing solutions like OpenStack via `librbd`.
- **Ceph File System (`cephfs`):** The Ceph File System consists of one or more Metadata Servers (`mds`), which store the inode portion of a file system as objects on the Ceph Storage Cluster. Ceph file systems can be mounted via a kernel client, a FUSE client, or via the `libcephfs` library for cloud computing solutions like OpenStack.

Additional clients include `librados`, which enables developers to create custom applications to interact with the Ceph Storage cluster and command line interface clients for administrative purposes.

Supporting software

An important aspect of Red Hat Ceph Storage security is to deliver solutions that have security built-in upfront, that Red Hat supports over time.

Specific steps which Red Hat takes with Red Hat Ceph Storage include:

- Maintaining upstream relationships and community involvement to help focus on security from the start.
- Selecting and configuring packages based on their security and performance track records.

- Building binaries from associated source code (instead of simply accepting upstream builds).
- Applying a suite of inspection and quality assurance tools to prevent an extensive array of potential security issues and regressions.
- Digitally signing all released packages and distributing them through cryptographically authenticated distribution channels.
- Providing a single, unified mechanism for distributing patches and updates.

In addition, Red Hat maintains a dedicated security team that analyzes threats and vulnerabilities against our products, and provides relevant advice and updates through the Customer Portal. This team determines which issues are important, as opposed to those that are mostly theoretical problems. The Red Hat Product Security team maintains expertise in, and makes extensive contributions to the upstream communities associated with our subscription products. A key part of the process, Red Hat Security Advisories, deliver proactive notification of security flaws affecting Red Hat solutions, along with patches that are frequently distributed on the same day the vulnerability is first published.

Threat and vulnerability management

Red Hat Ceph Storage is typically deployed in conjunction with cloud computing solutions, so it can be helpful to think about a Red Hat Ceph Storage deployment abstractly as one of many series of components in a larger deployment. These deployments typically have shared security concerns, which this guide refers to as *security zones*. Threat actors and vectors are classified based on their motivation and access to resources. The intention is to provide you with a sense of the security concerns for each zone, depending on your objectives.

Threat actors

A threat actor is an abstract way to refer to a class of adversary that you might attempt to defend against. The more capable the actor, the more rigorous the security controls that are required for successful attack mitigation and prevention. Security is a matter of balancing convenience, defense, and cost, based on requirements.

In some cases, it's impossible to secure a Red Hat Ceph Storage deployment against all threat actors described here. When deploying Red Hat Ceph Storage, you must decide where the balance lies for your deployment and usage.

As part of your risk assessment, you must also consider the type of data you store and any accessible resources, as this will also influence certain actors. However, even if your data is not appealing to threat actors, they could simply be attracted to your computing resources.

Nation-State Actors

This is the most capable adversary. Nation-state actors can bring tremendous resources against a target and have capabilities beyond that of any other actor. It is difficult to defend against these actors without stringent controls in place, both human and technical.

Serious Organized Crime

This class describes highly capable and financially driven groups of attackers. They are able to fund in-house exploit development and target research. In recent years, the rise of organizations such as the Russian Business Network, a massive cyber-criminal enterprise, has demonstrated how cyber attacks have become a commodity. Industrial espionage falls within the serious organized crime group.

Highly Capable Groups

This refers to hacktivist-type organizations that are not typically commercially funded but can pose a serious threat to service providers and cloud operators.

Motivated Individuals Acting Alone

These attackers come in many guises, such as rogue or malicious employees, disaffected customers, or small-scale industrial espionage.

Script Kiddies

These attackers do not target a specific organization but run automated vulnerability scanning and exploitation. They are often a nuisance; however, compromise by one of these actors is a major risk to an organization's reputation.

The following practices can help mitigate some of the risks identified:

Security Updates

You must consider the end-to-end security posture of your underlying physical infrastructure, including networking, storage, and server hardware. These systems will require their own security hardening practices. For your Red Hat Ceph Storage deployment, you should have a plan to regularly test and deploy security updates.

Product Updates

Red Hat recommends running product updates as they become available. Updates are typically released every six weeks (and occasionally more frequently). Red Hat endeavors to make point releases and z-stream releases fully compatible within a major release in order to not require additional integration testing.

Access Management

Access management includes authentication, authorization, and accounting. Authentication is the process of verifying the user's identity. Authorization is the process of granting permissions to an authenticated user. Accounting is the process of tracking which user performed an action. When granting system access to users, apply the *principle of least privilege*, and only grant users the granular system privileges they actually need. This approach can also help mitigate the risks of both malicious actors and typographical errors from system administrators.

Manage Insiders

You can help mitigate the threat of malicious insiders by applying careful assignment of role-based access control (minimum required access), using encryption on internal interfaces, and using authentication and authorization security such as centralized identity management. You can also consider additional non-technical options, such as separation of duties and irregular job role rotation.

Security zones

A security zone comprises users, applications, servers, or networks that share common trust requirements and expectations within a system. Typically they share the same authentication, authorization requirements, and users. Although you might refine these zone definitions further, this section refers to four distinct security zones, three of which form the bare minimum that is required to deploy a security-hardened Red Hat Ceph Storage cluster.

The following list are the security zones from least to most trusted:

- **Public Security Zone:** The public security zone is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks that are external to your OpenStack deployment over which you have no authority. Any data with confidentiality or integrity requirements that traverse this zone should be protected using compensating controls such as encryption. The public security zone **SHOULD NOT** be confused with the Ceph Storage Cluster's front- or client-side network, which is referred to as the `public_network` in Red Hat Ceph Storage and is usually **NOT** part of the public security zone or the Ceph client security zone.
- **Ceph Client Security Zone:** With Red Hat Ceph Storage, the Ceph client security zone refers to networks accessing Ceph clients such as Ceph Object Gateway, Ceph Block Device, Ceph Filesystem, or `librados`. The Ceph client security zone is typically behind a firewall separating itself from the public security zone. However, Ceph clients are not always protected from the public security zone. It is possible to expose the Ceph Object Gateway's S3 and Swift APIs in the public security zone.
- **Storage Access Security Zone:** The storage access security zone refers to internal networks providing Ceph clients with access to the Ceph Storage Cluster. We use the phrase *storage access security zone* so that this document is consistent with the terminology used in the OpenStack Platform Security and Hardening Guide. The storage access security zone includes the Ceph Storage Cluster's front- or client-side network, which is referred to as the `public_network` in Red Hat Ceph Storage.
- **Ceph Cluster Security Zone:** The Ceph cluster security zone refers to the internal networks providing the Ceph Storage Cluster's OSD daemons with network communications for replication, heartbeating, backfilling, and recovery. The Ceph cluster security zone includes the Ceph Storage Cluster's backside network, which is referred to as the `cluster_network` in Red Hat Ceph Storage.

  These security zones can be mapped separately, or combined to represent the majority of the possible areas of trust within a given Red Hat Ceph Storage deployment. Security zones should be mapped out against your specific deployment topology. The zones and their trust requirements will vary depending upon whether the storage cluster is operating in a standalone capacity or is serving a public, private, or hybrid cloud.

  For a visual representation of these security zones, see Security-Optimized Architecture.

Reference

For more information, see Network Communication

Connecting security zones

Any component that spans across multiple security zones with different trust levels or authentication requirements must be carefully configured. These connections are often the weak points in network architecture, and should always be configured to meet the security requirements of the highest trust level of any of the zones being connected. In many cases, the security controls of the connected zones should be a primary concern due to the likelihood of attack. The points where zones meet do present an opportunity for attackers to migrate or target their attack to more sensitive parts of the deployment.

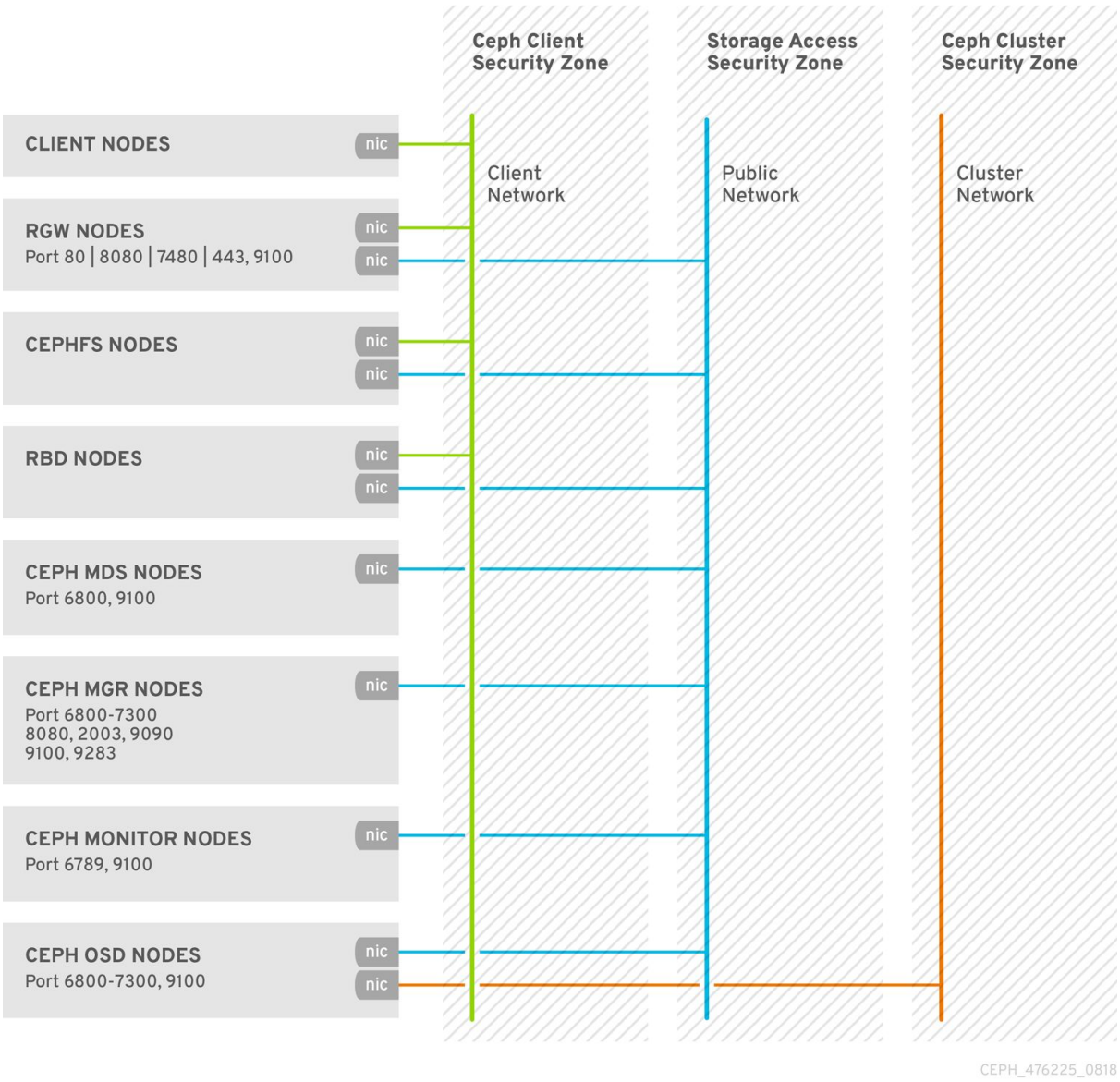
In some cases, Red Hat Ceph Storage administrators might want to consider securing integration points at a higher standard than any of the zones in which the integration point resides. For example, the Ceph Cluster Security Zone can be isolated from other security zones easily, because there is no reason for it to connect to other security zones. By contrast, the Storage Access Security Zone must provide access to port `6789` on Ceph monitor nodes, and ports `6800-7300` on Ceph OSD nodes. However, port `3000` should be exclusive to the Storage Access Security Zone, because it provides access to Ceph Grafana monitoring information that should be exposed to Ceph administrators only. A Ceph Object Gateway in the Ceph Client Security Zone will need to access the Ceph Cluster Security Zone's monitors (port `6789`) and OSDs (ports `6800-7300`), and may expose its S3 and Swift APIs to the Public Security Zone such as over HTTP port `80` or HTTPS port `443`; yet, it may still need to restrict access to the admin API.

The design of Red Hat Ceph Storage is such that the separation of security zones is difficult. As core services usually span at least two zones, special consideration must be given when applying security controls to them.

Security-optimized architecture

A Red Hat Ceph Storage cluster's daemons typically run on nodes that are subnet isolated and behind a firewall, which makes it relatively simple to secure a cluster. By contrast, Red Hat Ceph Storage clients such as Ceph Block Device (rbd), Ceph Filesystem (cephfs), and Ceph Object Gateway (rgw) access the Red Hat storage cluster, but expose their services to other cloud computing platforms.

"Figure: Security-optimized architecture" on page 31 illustrates the security-optimized architecture for the different Red Hat Ceph Storage cluster daemon and client types.
Figure: Security-optimized architecture



Encryption and key management

The Red Hat Ceph Storage cluster typically resides in its own network security zone, especially when using a private storage cluster network. In some cases, there are security requirements to assure the confidentiality or integrity of network traffic, and where Red Hat Ceph Storage uses encryption and key management.

> **Note:** Security zone separation might be insufficient for protection if an attacker gains access to Ceph clients on the public network.

The following cases are examples of when encryption and key management are used:

- SSH
- SSL Termination
- Messenger v2 protocol
- Encryption in Transit
- Compression modes of messenger v2 protocol

- Encryption at Rest
- Key rotation

SSH

All nodes in the Red Hat Ceph Storage cluster use SSH as part of deploying the cluster.

On each node:

- A `cephadm` user exists with password-less root privileges.
- The SSH service is enabled and by extension port 22 is open.
- A copy of the `cephadm` user's public SSH key is available.

> **Important:** Any person with access to the `cephadm` user by extension has permission to run commands as `root` on any node in the Red Hat Ceph Storage cluster.

For more information, see How `cephadm` works

SSL Termination

The Ceph Object Gateway may be deployed in conjunction with HAProxy and `keepalived` for load balancing and failover. Understand the implications of terminating SSL with HAProxy and `keepalived` and their encryption status.

The Ceph Object Gateway may be deployed in conjunction with HAProxy and `keepalived` for load balancing and failover. Earlier versions of Civetweb do not support SSL and later versions support SSL with some performance limitations.

You can configure the Beast front-end web server to use the OpenSSL library to provide Transport Layer Security (TLS).

When using HAProxy and `keepalived` to terminate SSL connections, the HAProxy and `keepalived` components use encryption keys.

When using HAProxy and `keepalived` to terminate SSL, the connection between the load balancer and the Ceph Object Gateway is **NOT** encrypted.

Reference

For more information, see the following:

- "Configuring HTTPS for the Ceph Object Gateway" on page 0
- High availability service

Messenger v2 protocol

Messenger v2 (`msgr2`) is the new protocol for Ceph clients, or other Ceph daemons, to connect to the Ceph Monitor daemon. Ceph daemons bind to multiple ports, allowing both the legacy v1-compatible, and the new, v2-compatible Ceph clients to connect to the same storage cluster.

The second version of Ceph's on-wire protocol, `msgr2`, has the following features:

- A secure mode encrypting all data moving through the network.
- Encapsulation improvement of authentication payloads, enabling future integration of new authentication modes.
- Improvements to feature advertisement and negotiation.

Ceph clients or other Ceph daemons connecting to the Ceph Monitor daemon uses the `v2` protocol first, if possible, but if not, then the legacy `v1` protocol is used. By default, both messenger protocols, `v1` and `v2`, are enabled. The new v2 port is 3300, and the legacy v1 port is 6789, by default.

The messenger v2 protocol has two configuration options that control whether the v1 or the v2 protocol is used:

- `ms_bind_msgr1` - This option controls whether a daemon binds to a port speaking the v1 protocol; it is `true` by default.
- `ms_bind_msgr2` - This option controls whether a daemon binds to a port speaking the v2 protocol; it is `true` by default.

Similarly, two options control based on IPv4 and IPv6 addresses used:

- `ms_bind_ipv4` - This option controls whether a daemon binds to an IPv4 address; it is `true` by default.
- `ms_bind_ipv6` - This option controls whether a daemon binds to an IPv6 address; it is `true` by default.

> **Note:** The ability to bind to multiple ports has paved the way for dual-stack IPv4 and IPv6 support.

The `msgr2` protocol supports two connection modes:

- `crc`
  - Provides strong initial authentication when a connection is established with `cephx`.
  - Provides a `crc32c` integrity check to protect against bit flips.
  - Does not provide protection against a malicious man-in-the-middle attack.
  - Does not prevent an eavesdropper from seeing all post-authentication traffic.
- `secure`
  - Provides strong initial authentication when a connection is established with `cephx`.
  - Provides full encryption of all post-authentication traffic.
  - Provides a cryptographic integrity check.

The default mode is `crc`.

Ceph Object Gateway Encryption

Also, the Ceph Object Gateway supports encryption with customer-provided keys using its S3 API.

> **Important:** To comply with regulatory compliance standards requiring strict encryption in transit, administrators **MUST** deploy the Ceph Object Gateway with client-side encryption.

Ceph Block Device Encryption

System administrators integrating Ceph as a backend for OpenStack Platform 13 **MUST** encrypt Ceph Block Device volumes using `dm_crypt` for RBD Cinder to ensure on-wire encryption within the Ceph storage cluster.

> **Important:** To comply with regulatory compliance standards requiring strict encryption in transit, system administrators **MUST** use `dmcrypt` for RBD Cinder to ensure on-wire encryption within the Ceph storage cluster.

Reference

For more information, see Configuring

Encryption in transit

Encryption for all Ceph traffic over the network is enabled by default, with the introduction of the messenger version 2 protocol. The `secure` mode setting for messenger v2 encrypts communication between Ceph daemons and Ceph clients, providing end-to-end encryption.

You can check for encryption of the messenger v2 protocol with the `ceph config dump` command, `netstat -Ip | grep ceph-osd` command, or verify the Ceph daemon on the v2 ports.

Reference

For more information, see the following:

- “SSL Termination” on page 33
- “Messenger v2 protocol” on page 33
- S3 server-side encryption
- Server side encryption

Compression modes of messenger v2 protocol

The messenger v2 protocol supports the compression feature. Compression is not enabled by default.

Compressing and encrypting the same message is not recommended as the level of security of messages between peers is reduced. If encryption is enabled, a request to enable compression is ignored until the configuration option `ms_osd_compress_mode` is set to `true`.

It supports two compression modes:

- `force`
  - In multi-availability zones deployment, compresses replication messages between OSDs saves latency.
  - In the public cloud, minimizes message size, thereby reducing network costs to cloud provider.
  - Instances on public clouds with NVMe provides low network bandwidth relative to the device bandwidth. Does not provide protection against a malicious man-in-the-middle attack.
- `none`
  - The messages are transmitted without compression.

To ensure that compression of the message is enabled, run the `debug_ms` command and check some debug entries for connections. Also, you can run the `ceph config get` command to get details about the different configuration options for the network messages.

Reference

- Network configuration options

Encryption at Rest

Red Hat Ceph Storage supports encryption at rest for both the Ceph Storage Cluster as well as for Ceph Object Gateway in various scenarios.

1. **Ceph Storage Cluster:** The Ceph Storage Cluster supports Linux Unified Key Setup or LUKS encryption of Ceph OSDs and their corresponding journals, write-ahead logs, and metadata databases. In this scenario, Ceph will encrypt all data at rest irrespective of whether the client is a Ceph Block Device, Ceph Filesystem, or a custom application built on `librados`.
2. **Ceph Object Gateway:** The Ceph storage cluster supports encryption of client objects. Additionally, the data transmitted is between the Ceph Object Gateway and the Ceph Storage Cluster is in encrypted form.

Ceph Storage Cluster Encryption

The Ceph storage cluster supports encrypting data stored in Ceph OSDs. Red Hat Ceph Storage can encrypt logical volumes with `lvm` by specifying `dmcrypt`; that is, `lvm`, invoked by `ceph-volume`, encrypts an OSD’s logical volume, not its physical volume. It can encrypt non-LVM devices like partitions using the same OSD key. Encrypting logical volumes allows for more configuration flexibility.

Ceph uses LUKS v1 rather than LUKS v2, because LUKS v1 has the broadest support among Linux distributions.

When creating an OSD, `lvm` will generate a secret key and pass the key to the Ceph Monitors securely in a JSON payload via `stdin`. The attribute name for the encryption key is `dmcrypt_key`.

Important: System administrators must explicitly enable encryption.

By default, Ceph does not encrypt data stored in Ceph OSDs. System administrators must enable `dmcrypt` to encrypt data stored in Ceph OSDs. When using a Ceph Orchestrator service specification file for adding Ceph OSDs to the storage cluster, set the following option in the file to encrypt Ceph OSDs:

Example

```
...
encrypted: true
...
```

Note: LUKS and `dmcrypt` only address encryption for data at rest, not encryption for data in transit.

Ceph Object Gateway Encryption

The Ceph Object Gateway supports encryption with customer-provided keys using its S3 API. When using customer-provided keys, the S3 client passes an encryption key along with each request to read or write encrypted data. It is the customer's responsibility to manage those keys. Customers must remember which key the Ceph Object Gateway used to encrypt each object. For full information about Ceph Object Gateway encryption, see Server side encryption.

Reference

For more information, see the following:

- S3 API server-side encryption

Enabling key rotation

Key rotation helps ensure that current industry and security compliance requirements are met.

PREREQUISITE

Before you begin, be sure that you have a running Red Hat Ceph Storage cluster and admin user privileges.

Note: The active Ceph cluster includes nodes within the Ceph client role with parallel key changes.

Ceph and Ceph Object Gateway daemons within the Ceph cluster have a secret key. This key is used to connect to and authenticate with the cluster. You can update an active security key within an active Ceph cluster with minimal service interruption, by using the key rotation feature.

For more information about enabling key rotation on Ceph dashboard, see User capabilities.

1. Rotate the key.

   ```
   ceph orch daemon rotate-key NAME
   ```

   For example,

   ```
   [ceph: root@host01 /]# ceph orch daemon rotate-key mgr.ceph-key-host01
   Scheduled to rotate-key mgr.ceph-key-host01 on host 'my-host-host01-installer'
   ```

2. If using a daemon other than MDS, OSD, or MGR, restart the daemon to switch to the new key. MDS, OSD, and MGR daemons do not require daemon restart.

   ```
   ceph orch restart SERVICE_TYPE
   ```

   For example,

```
[ceph: root@host01 /]# ceph orch restart rgw
```

CephX key rotation

CephX key rotation updates authentication keys to use stronger cipher types after a cluster upgrade.

> **Important:** `aes256k` key types are only supported on version 9.1z1 or later.

CephX is the authentication mechanism used by Ceph clients and daemons. Keys are associated with each entity and used for authentication.

After upgrading Ceph, existing keys might use older cipher types such as `aes`. Newer versions introduce stronger ciphers such as `aes256k`.

Service daemon keys for `mon`, `mgr`, `mds`, and `osd` are rotated automatically by `cephadm` during Ceph upgrade. Client CephX keys are not rotated automatically by `cephadm`.

During the transition, the cluster supports both old and new ciphers.

Identity and access management

Red Hat Ceph Storage provides identity and access management for Ceph Storage Cluster users, Ceph Object Gateway users, Ceph Object Gateway LDAP and AD, and for Ceph Object Gateway OpenStack Keystone.

Ceph Storage cluster user access

To identify users and protect against man-in-the-middle attacks, Ceph provides its `cephx` authentication system to authenticate users and daemons.

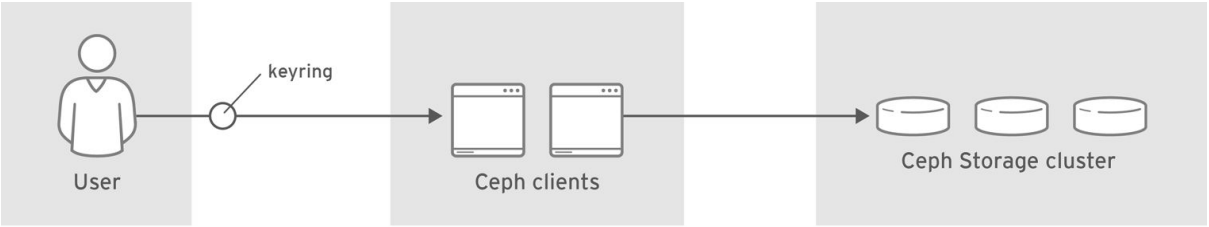
For more information about `cephx`, see Ceph user management.

> **Important:** The `cephx` protocol **DOES NOT** address data encryption in transport or encryption at rest.

Cephx uses shared secret keys for authentication, meaning both the client and the monitor cluster have a copy of the client's secret key. The authentication protocol is such that both parties are able to prove to each other they have a copy of the key without actually revealing it. This provides mutual authentication, which means the cluster is sure the user possesses the secret key, and the user is sure that the cluster has a copy of the secret key.

In "Figure: User access" on page 37, users are either individuals or system actors such as applications, which use Ceph clients to interact with the Red Hat Ceph Storage cluster daemons.

Figure: User access



Ceph runs with authentication and authorization enabled by default. Ceph clients may specify a user name and a keyring containing the secret key of the specified user, usually by using the command line. If the user and keyring are not provided as arguments, Ceph will use the `client.admin` administrative user as the default. If a keyring is not specified, Ceph will look for a keyring by using the `keyring` setting in the Ceph configuration.

Important: To harden a Ceph cluster, keyrings **SHOULD ONLY** have read and write permissions for the current user and `root`. The keyring containing the `client.admin` administrative user key must be restricted to the `root` user.

Reference

For more information, see the following:

- Configuring
- Ceph authentication configuration

Ceph Object Gateway user access

The Ceph Object Gateway provides a RESTful application programming interface (API) service with its own user management that authenticates and authorizes users to access S3 and Swift APIs containing user data.

Authentication consists of:

- **S3 User:** An access key and secret for a user of the S3 API.
- **Swift User:** An access key and secret for a user of the Swift API. The Swift user is a subuser of an S3 user. Deleting the S3 *parent* user will delete the Swift user.
- **Administrative User:** An access key and secret for a user of the administrative API. Administrative users should be created sparingly, as the administrative user will be able to access the Ceph Admin API and run its functions, such as creating users, and giving them permissions to access buckets or containers and their objects among other things.

The Ceph Object Gateway stores all user authentication information in Ceph Storage cluster pools. Additional information may be stored about users including names, email addresses, quotas, and usage.

Reference

For more information, see the following:

- User management
- Creating an administrative user

Ceph Object Gateway LDAP or AD authentication

Red Hat Ceph Storage supports Light-weight Directory Access Protocol (LDAP) servers for authenticating Ceph Object Gateway users. When configured to use LDAP or Active Directory (AD), Ceph Object Gateway defers to an LDAP server to authenticate users of the Ceph Object Gateway.

Ceph Object Gateway controls whether to use LDAP. However, once configured, it is the LDAP server that is responsible for authenticating users.

To secure communications between the Ceph Object Gateway and the LDAP server, Red Hat recommends deploying configurations with LDAP Secure or LDAPS.

Important: When using LDAP, ensure that access to the `rgw_ldap_secret = PATH_TO_SECRET_FILE` secret file is secure.

Reference

For more information, see the following:

- Configuring LDAP and Ceph Object Gateway
-

Ceph Object Gateway OpenStack Keystone authentication

Red Hat Ceph Storage supports using OpenStack Keystone to authenticate Ceph Object Gateway Swift API users. The Ceph Object Gateway can accept a Keystone token, authenticate the user and create a corresponding Ceph Object Gateway user. When Keystone validates a token, the Ceph Object Gateway considers the user authenticated.

Ceph Object Gateway controls whether to use OpenStack Keystone for authentication. However, once configured, it is the OpenStack Keystone service that is responsible for authenticating users.

Configuring the Ceph Object Gateway to work with Keystone requires converting the OpenSSL certificates that Keystone uses for creating the requests to the `nss db` format.

Reference

For more information, see the following:

- Ceph Object Gateway and OpenStack Keystone

Ceph Dashboard SSO using OpenID Connect (OIDC) protocol

The Red Hat Ceph Management stack is built by using a modular, service-based architecture that relies on nginx and `oauth2-proxy` components.

The Ceph Management gateway (`mgmt-gateway`) provides unified access, improved user experience, and high availability for both the Ceph Dashboard and monitoring.

The OAuth2 Proxy (`oauth2-proxy`) service provides enhanced security, seamless SSO, and centralized authentication management.

Enable the `mgmt-gateway` service by itself to provide a single-entry point to the cluster, high availability for monitoring and dashboard, as well as for improved security. Enable the service together with the `oauth2-proxy` to use SSO.

With the Ceph Management gateway (`mgmt-gateway`) and the OAuth2 Proxy (`oauth2-proxy`) in place, nginx automatically directs the user through the `oauth2-proxy` to the configured Identity Provider (*IdP*), when single sign-on (SSO) is configured. The IdP then runs the authentication and login.

This configuration provides the following advantages:

- Unified access.
- Improved user experience.
- High availability for the Ceph Dashboard and monitoring.
- Enhanced security.
- Seamless SSO.
- Centralized authentication management.

For more information about this modular SSO solution and implementation with the Ceph Dashboard, see Using single sign-on with the dashboard.

For more information about dashboard SSO options, see Using single sign-on with the dashboard

Using the Ceph Management gateway (`mgmt-gateway`)

The Ceph Management gateway (`mgmt-gateway`) service provides a modular, service-based architecture design.`cephadm` manages the `mgmt-gateway` service, which is built on top of nginx. This gateway acts as the front-end and single entry point to the Ceph cluster, providing unified access to all Ceph applications, including the Ceph Dashboard and the monitoring stack.

Using nginx enhances security and simplifies access management due to its robust community support and high-security standards. The `mgmt-gateway` service acts as a reverse proxy that routes requests to the appropriate Ceph application instances. Use the `mgmt-gateway` either by itself or with the OAuth2 Proxy service for authentication to enable single-sign on (SSO) access to the Ceph Dashboard. For more information about the OAuth2 Proxy service, see “Using the OAuth2 Proxy (oauth2-proxy) service” on page 48.

Using the Ceph Management gateway provides multiple benefits.

Unified access

Consolidated access through nginx improves security and provide a single entry point to services.

Improved user experience

Users no longer needs to keep track of where each application is running (IP or host).

High availability for Ceph Dashboard

nginx high availability (HA) mechanisms are used to provide high availability for the Ceph Dashboard.

High availability for monitoring

nginx high availability mechanisms are used to provide high availability for monitoring.

High availability for mg mt-gateway

Users can have continuous access to Ceph management tools, such as the Ceph Dashboard, Prometheus, Grafana, and Alertmanager, by enabling high availability (HA) for the Ceph Management gateway (`mgmt-gateway`). For more information about enabling high availability (HA) for the Ceph Management gateway, see “Enabling high availability for Ceph Management gateway” on page 44

Deploying the `mgmt-gateway` provides enhanced security. Once the `mgmt-gateway` service is deployed users cannot access monitoring services without authentication through the Ceph Dashboard. After the service is deployed, Prometheus, Grafana, and Alertmanager applications are accessible through links on the Ceph Dashboard. To get to the application links, go to **Administration › Services** on the Ceph Dashboard.

> **Note:** In `mgmt-gateway` mode, Prometheus and Alertmanager require the user to define a unique username and password. The default is `admin:admin`. If OAuth2-proxy SSO is not enabled, the user must manually define these credentials.

nginx HA mechanisms are used to provide high availability for all the Ceph management applications including the Ceph Dashboard and monitoring stack. The `mgmt-gateway` service handles manager failover transparently and redirects the user to the active manager.

With monitoring, the Ceph Management service takes care of handling HA when several instances of Prometheus, Alertmanager or Grafana are available. The reverse proxy automatically detects healthy instances and uses them to process user requests.

> **Important:** Design high availability configurations carefully to avoid introducing single points of failure.

Ceph Management gateway limitations

Using the `mgmt-gateway` service has the following limitations:

- Services must bind to the appropriate ports based on the applications being proxied. Ensure that there are no port conflicts that might disrupt service availability.
- The `mgmt-gateway` service internally makes use of the nginx reverse proxy.
  Use the following to find the default container image is used by default:

  ```
  DEFAULT_NGINX_IMAGE = 'quay.io/ceph/NGINX_IMAGE'
  ```

  Storage administrators can change the image being used by changing the **`container_image_nginx`**`cephadm`module option. If other daemons were running, you must redeploy the daemons to use the new image.

  ```
  ceph config set mgr mgr/cephadm/container_image_nginx NEW_NGINX_IMAGE
  ceph orch redeploy mgmt-gateway
  ```

Reconfigure the dashboard to use `cephadm`-signed certificates

Replace custom dashboard certificates with `cephadm`-signed certificates required for mTLS when the management gateway is enabled.

PREREQUISITE

Conditions

Perform this task only if the dashboard currently uses custom (non-cephadm) certificates.

Impact

Reloading the dashboard briefly interrupts access to the Ceph Dashboard.

While reconfiguring the dashboard can be done at any time, be sure to reconfigure enabling the Ceph Management gateway.

1. Generate cephadm-signed certificates for the dashboard.

```
ceph orch certmgr generate-certificates dashboard \

tee \
>(jq -r '.cert' > dashboard.cert.pem) \
>(jq -r '.key' > dashboard.key.pem) \
> /dev/null
```

   Two files are created in the current directory. dashboard.cert.pem and dashboard.key.pem.

2. Set the dashboard certificate.

```
ceph dashboard set-ssl-certificate -i dashboard.cert.pem
```

3. Set the dashboard key.

```
ceph dashboard set-ssl-certificate-key -i dashboard.key.pem
```

4. Reload the dashboard module to apply the new certificates.

```
ceph mgr module disable dashboard; ceph mgr module enable dashboard --force
```

AFTER COMPLETING THE TASK

Verify the dashboard is using the new cephadm-signed certificates and that all services start without errors.

- Open the dashboard URL and confirm HTTPS loads without certificate errors.
- Check the MGR logs for successful dashboard startup and TLS messages.
- (Optional) Validate that backend services (Prometheus, Alertmanager) connect without mTLS errors after enabling mgmt-gateway.

Enabling the Ceph Management gateway

Enable the Ceph Management gateway for SSO access to the Dashboard and the Ceph cluster. Enabling the mgmt-gateway service either with the cephadm CLI commands or by using a service specification file.

PREREQUISITE

Before you begin, if the dashboard uses custom certificates, you must replace them with cephadm-signed certificates before you enable the management gateway. For more information, see “Reconfigure the dashboard to use cephadm-signed certificates” on page 40.
After deploying the mgmt-gateway service, direct access to services like Prometheus, Grafana, and Alertmanager is no longer allowed. These services are now accessible only through the Ceph Dashboard by the provided links in **Administration › Services**.

Enabling the Ceph Management gateway with the command-line interface

- Deploy the mgmt-gateway service.

```
ceph orch apply mgmt-gateway [--placement=DESTINATION_HOST] [--enable-auth=true]
```

> **Note:** The **--enable-auth=true** parameter is mandatory to enable SSO with the oauth2-proxy.

For example,

```
[ceph: root@host01 /]# ceph orch apply mgmt-gateway --placement=host01
```

AFTER COMPLETING THE TASK

Verify that the service has been deployed, as expected.

1. Run the **ceph orch ls** command to get the service status.
2. Run the **ceph orch ps** command to get the status of the corresponding daemons.

Enabling the Ceph Management gateway with a service specification file

PREREQUISITE

Before enabling the Ceph Management gateway, be sure that the following are on each Ceph node that the `mgmt-gateway` service will run on.

- The port for gateway service use.
- A running Red Hat Ceph Storage cluster.
- (Optional) SSL protocols being used.
- (Optional) SSL ciphers.
- (Optional) SSL certificates and certificate keys.

For more information about SSL protocols, ciphers, certificates, and certificate keys, see the *Deploying web servers and reverse proxies* in the Red Hat Enterprise Linux documentation.

1. Create a YAML file for the `mgmt-gateway` service.
   For example,

   ```
   [root@host01 ~]# touch mgmt-gateway.yaml
   ```

2. Edit the YAML file to include the following details.

   Important: The following fields are optional: **ssl_protocols**, **ssl_ciphers**, and **ssl_certificate**. When omitted, the `mgmt-gateway` service uses a safe and secure configuration. Changing these fields are permitted but should be done with care as this can compromise the security of your system.

   ```
   service_type: mgmt-gateway
   placement:
     hosts:
       - HOST
   spec:
    port: PORT
    ssl_protocols:   # Optional
      - TLSv1.3
    ssl_ciphers:     # Optional
      - AES128-SHA
      - AES256-SHA
      - RC4-SHA
    ssl_certificate: |   # Optional
      -----BEGIN CERTIFICATE-----
      < YOU CERT DATA HERE >
      -----END CERTIFICATE-----
    ssl_certificate_key: |
     -----BEGIN RSA PRIVATE KEY-----
      < YOU PRIV KEY DATA HERE >
     -----END RSA PRIVATE KEY-----
   ```

   Important: Use the TLSv1.3 SSL protocol. TLSv1.3 includes a set of secure ciphers by default. While TLSv1.2 is supported, it is crucial to use only a subset of secure ciphers when using this protocol. Using weak or outdated ciphers can significantly compromise the security of your system.

   For example,

```
service_type: mgmt-gateway
service_id: gateway
placement:
  hosts:
    - ceph0
spec:
 port: 5000
 ssl_protocols:
   - TLSv1.3
   - ...
 ssl_ciphers:
   - AES128-SHA
   - AES256-SHA
   - ...
 ssl_certificate: |
   -----BEGIN CERTIFICATE-----
   MIIDtTCCAp2gAwIBAgIYMC4xNzc1NDQxNjEzMzc2MjMyXzxvQ7EcMA0GCSqGSIb3
   DQEBCwUAMG0xCzAJBgNVBAYTAlVTMQ0wCwYDVQQIDARVdGFoMRcwFQYDVQQHDA5T
   [...]
   -----END CERTIFICATE-----
ssl_certificate_key: |
   -----BEGIN PRIVATE KEY-----
   MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQC5jdYbjtNTAKW4
   /CwQr/7wOiLGzVxChn3mmCIF3DwbL/qvTFTX2d8bDf6LjGwLYloXHscRfxszX/4h
   [...]
   -----END PRIVATE KEY-----
```

“mgmt-gateway specific fields in the spec file” on page 43 lists fields that are specific to the `mgmt-gateway` service section of the spec file (`ceph.deployment.service_spec.MgmtGatewaySpec`).

`mgmt-gateway` specific fields in the spec file

Field	Description
`disable_https`	Is a flag to disable HTTPS. If `True`, the server will use unsecure HTTP.
`enable_auth`	Is a flag to enable SSO auth. Requires `oauth2-proxy` to be active for SSO authentication.
`networks`	A list of network identities instructing the daemons to only bind on the particular networks in that list. In case the cluster is distributed across multiple networks. You can add multiple networks.
`placement`	For the orchestrator to deploy a service, it needs to know where to deploy daemons, and how many to deploy. This is the role of a placement specification. Placement specifications can either be passed as command line arguments or in a YAML files. For more information, see Managing services.
`port`	The port number on which the server will listen.
`server_tokens`	Flag control server tokens in responses: `on`, `off`, `build`, `string`.
`ssl_cert`	A multi-line string that contains the SSL certificate.
`ssl_key`	A multi-line string that contains the SSL key.
`ssl_ciphers`	List of supported secure SSL ciphers. Changing this list can reduce system security.
`ssl_prefer_server_ciphers`	Prefer server ciphers over client ciphers: `on`, `off`.
`ssl_protocols`	A list of supported SSL protocols (as supported by nginx).
`ssl_session_cache`	Duration of an SSL/TLS session is cached: `off`, `none`, `[builtin[:size]]`, `[shared:name:size]`.

Field	Description
ssl_session_tickets	A multi-option flag to control session tickets: on, off.
ssl_session_timeout	The duration for SSL session timeout. Syntax: time (for example, 5m).
ssl_stapling	Flag to enable or disable SSL stapling: on, off.
ssl_stapling_verify	Flag to control verification of SSL stapling: on, off.

3. Apply the specification file.

```
[root@host01 ~]# ceph orch apply -i mgmt-gateway.yaml
```

AFTER COMPLETING THE TASK

Verify that the service has been deployed, as expected.

1. Run the **ceph orch ls** command to get the service status.
2. Run the **ceph orch ps** command to get the status of the corresponding daemons.

Enabling high availability for Ceph Management gateway

Ensure continuous access to Ceph management tools, such as the Ceph Dashboard, Prometheus, Grafana, and Alertmanager, by enabling high availability (HA) for the Ceph Management gateway (mgmt-gateway).

PREREQUISITE

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

- A subset of hosts have the mgmt label set on them. These are the hosts where the daemons are deployed. Use the **ceph orch host ls** command to see which hosts have the mgmt labels set.
- Have an available Virtual IP (VIP).

By deploying multiple mgmt-gateway instances in an active/standby setup, the system can automatically switch to a backup instance if there is failure. Change to a backup instance by using the keepalived for failover. The oauth2-proxy service operates statelessly, with nginx as a load balancer to evenly distribute traffic. They following are the key components of HA for mgmt-gateway:

Keepalived

Provides failover support.

OAuth2-Proxy

Manages authentication.

Nginx

Runs as a load balancer and reverse-proxy for Ceph Management stack services.

Virtual IP (VIP)

Helps ensure smooth access by other external and internal services to the mgmt-gateway.

Enable Enabling HA to help ensure that tool access is always available, even during a component failure. Enable high availability service either with the cephadm CLI commands or by using a service specification file.

After deploying the mgmt-gateway service, direct access to services like Prometheus, Grafana, and Alertmanager is no longer allowed. These services are now accessible only through the Ceph Dashboard by the links that are provided in **Administration › Services**.

> **Note:** To enable high availability (HA) for the management gateway, the ingress service must also be configured—this is demonstrated in the comprehensive example provided. Without ingress, the virtual IP (VIP) will not function correctly.

Enabling HA for Ceph Management gateway with the command-line interface

– Deploy the `mgmt-gateway` service in a high-availability configuration.

> **Note:** To enable Single Sign-On (SSO), configure and deploy an OAuth2-proxy service. For more information about OAuth2-proxy service, see "Using the OAuth2 Proxy (oauth2-proxy) service" on page 48

To enable SSO with the `oauth2-proxy`, the **`--enable-auth=true`** parameter is mandatory.

```
ceph orch apply mgmt-gateway --virtual_ip VIP --enable-auth=true --
placement="label:mgmt"
```

For example,

```
[ceph: root@host01 /]# ceph orch apply mgmt-gateway --virtual_ip 192.168.100.220 --
enable-auth=true --placement="label:mgmt"
```

AFTER COMPLETING THE TASK

Verify that the service is deployed, as expected.

1. Run the **`ceph orch ls`** command to get the service status.
2. Run the **`ceph orch ps`** command to get the status of the corresponding daemons.

Enabling HA for Ceph Management gateway with a service specification file

PREREQUISITE

Before enabling the Ceph Management gateway, be sure that the following are on each Ceph node that the `mgmt-gateway` service will run on.

– The port for gateway service use.

– A running Red Hat Ceph Storage cluster.

– (Optional) SSL protocols and ciphers for secure communication.

– (Optional) SSL certificates and private key data for secure connections.

For more information about SSL protocols, ciphers, certificates, and certificate keys, see the *Deploying web servers and reverse proxies* in the Red Hat Enterprise Linux documentation.

1. Create a YAML file for the `mgmt-gateway` service.
   For example,

   ```
   [root@host01 ~]# touch mgmt-gateway.yaml
   ```

2. Edit the YAML file to include the following details.

   > **Important:** The following fields are optional: **`ssl_protocols`**, **`ssl_ciphers`**, and **`ssl_cert`**. When omitted, the `mgmt-gateway` service uses a safe and secure configuration. Change these fields with care, as they can compromise the security of your system.

   ```
   service_type: mgmt-gateway
   placement:
     hosts: - ceph-node-1
   spec:
     enable_auth: true
     virtual_ip: 192.168.100.220
     ssl_protocols:
       - TLSv1.3
     ssl_ciphers:
       - AES128-SHA
       - AES256-SHA
     ssl_cert: |
       -----BEGIN CERTIFICATE-----
       < YOUR CERT DATA HERE >
       -----END CERTIFICATE-----
   ```

```
ssl_key: |
  -----BEGIN PRIVATE KEY-----
  < YOUR PRIVATE KEY DATA HERE >
  -----END PRIVATE KEY-----
```

Important: SSL-related fields such as **ssl_protocols**, **ssl_ciphers**, and **ssl_cert** are optional but recommended for secure configurations. If omitted, default secure settings are used. However, incorrect modifications can weaken your system's security. Ensure that any changes are in line with current security best practices.

"mgmt-gateway specific fields in the spec file" on page 46 lists fields that are specific to the `mgmt-gateway` service section of the spec file (`ceph.deployment.service_spec.MgmtGatewaySpec`).

`mgmt-gateway` specific fields in the spec file

Field	Description
`disable_https`	Is a flag to disable HTTPS. If `True`, the server uses unsecure HTTP.
`enable_auth`	Is a flag to enable SSO auth. Requires `oauth2-proxy` to be active for SSO authentication.
`networks`	A list of network identities that instruct the daemons to only bind on the particular networks in that list. In case the cluster is distributed across multiple networks, you can add multiple networks.
`placement`	For the orchestrator to deploy a service, it needs to know where to deploy daemons, and how many to deploy. This is the role of a placement specification. Placement specifications can either be passed as command line arguments or in a YAML files. For more information, see Managing services.
`virtual_ip`	The high availability virtual IP number on which the server will reside.
`server_tokens`	Flag control server tokens in responses: `on`, `off`, `build`, `string`.
`ssl_cert`	A multi-line string that contains the SSL certificate.
`ssl_key`	A multi-line string that contains the SSL key.
`ssl_ciphers`	List of supported secure SSL ciphers. Changing this list can reduce system security.
`ssl_prefer_server_ciphers`	Prefer server ciphers over client ciphers: `on`, `off`.
`ssl_protocols`	A list of supported SSL protocols (as supported by nginx).
`ssl_session_cache`	Duration that an SSL/TLS session is cached: `off`, `none`, `[builtin[:size]]`, `[shared:name:size]`.
`ssl_session_tickets`	A multi-option flag to control session tickets: `on`, `off`.
`ssl_session_timeout`	The duration for SSL session timeout. Syntax: time (for example, `5m`).
`ssl_stapling`	Flag to enable or disable SSL stapling: `on`, `off`.
`ssl_stapling_verify`	Flag to control verification of SSL stapling: `on`, `off`.

3. Apply the specification file.

```
[root@host01 ~]# ceph orch apply -i mgmt-gateway.yaml
```

Applying the specification file applies the HA configuration including the virtual IP and optional SSL.

AFTER COMPLETING THE TASK

Verify that the service has been deployed, as expected.

1. Run the **ceph orch ls** command to get the service status.
2. Run the **ceph orch ps** command to get the status of the corresponding daemons.

Example configurations for `mgmt-gateway` with high availability

Use these examples to configure multiple `mgmt-gateway` instances with high availability (HA) in an active/standby configuration with keepalived for seamless failover.

Example `mgmt-gateway` configuration

```
service_type: mgmt-gateway
placement:
  label: mgmt
spec:
  enable_auth: true
  virtual_ip: 192.168.100.220  # HA Virtual IP
```

Example of ingress configuration for keepalived

```
service_type: ingress
service_id: ingress-mgmt-gw
placement:
  label: mgmt
virtual_ip: 192.168.100.220
backend_service: mgmt-gateway
keepalive_only: true
```

Example for deploying the `mgmt-gateway` in a high availability configuration

The following is a complete example that shows all the components required to deploy the Ceph management gateway in HA mode with SSO enabled. The configuration includes the nodes, the management gateway, `oauth2-proxy`, and ingress.

```
---
service_type: host
hostname: ceph-node-1
addr: 192.168.100.101
labels:
  - mgmt
---
service_type: host
hostname: ceph-node-2
addr: 192.168.100.102
labels:
  - mgmt
---
service_type: mgmt-gateway
placement:
  label: mgmt
spec:
  enable_auth: true
  virtual_ip: <your-virtual-ip>
---
service_type: oauth2-proxy
placement:
  label: mgmt
spec:
  provider_display_name: "<your-oidc-provider>"
  client_id: <your-client-id>
  client_secret: <your-client-secret>
  oidc_issuer_url: <your-oidc-url>
  allowlist_domains:
    - 192.168.100.1:8080
---
service_type: ingress
service_id: ingress-mgmt-gw
placement:
  label: mgmt
virtual_ip: <your-virtual-ip>
```

```
backend_service: mgmt-gateway
keepalive_only: true
```

Using the OAuth2 Proxy (`oauth2-proxy`) service

The OAuth2 Proxy service provides an advanced method for managing authentication and access control for Ceph applications. The oauth2-proxy service integrates with external Identity Providers (*IdPs*) to provide secure, flexible authentication with the OpenID Connect (*OIDC*) protocol. oauth2-proxy acts as an authentication gateway, ensuring that access to Ceph applications are tightly controlled.

Using the OAuth2 Proxy service with authentication requires using the Ceph Management gateway (`mgmt-gateway`). For more information, see "Using the Ceph Management gateway (mgmt-gateway)" on page 39.

Using the `oauth2-proxy` service provides multiple benefits.

Enhanced security

Provides robust authentication through integration with external IdPs by using the OIDC protocol.

Seamless single sign-on (*SSO*)

Enables seamless SSO across all Ceph monitoring applications, improving user access control. For more information about enabling OAuth2 SSO for the Ceph Dashboard, see Enabling OAuth2 single sign-on.

Centralized authentication

Centralizes authentication management, reducing complexity and improving control over access.

Deploying the `oauth2-proxy` service provides enhanced security. Once the `oauth2-proxy` service is deployed all access to Ceph applications must be authenticated through the external IdP by using OIDC. Authentication prevents unauthorized users from accessing sensitive information. Users are redirected to the IdP for login and then returned to the requested application. This setup ensures secure access and integrates seamlessly with the Ceph management stack. The Prometheus, Alertmanager, and Grafana Ceph applications all require authentication through the required IdP.

The `oauth2-proxy` service uses the OAuth Provider open-source project, enabling easier service integration with a variety of external IdPs, providing a secure and flexible authentication mechanism. For a full list of valid OAuth providers, see **OAuth Provider Configuration** on OAuth2 Proxy Docs.

OAuth2 Proxy limitations

The `oauth2-proxy` service does not support high availability configurations.

> **Important:** Proper configuration of the IdP and OAuth2 Proxy parameters is crucial to avoid authentication failures. Misconfiguration can lead to access issues.

Creating an admin account with Red Hat Single Sign-On 7.6.0

Create an admin account to synchronize OAuth2 service to the Ceph dashboard.

PREREQUISITE

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

- A running Red Hat Ceph Storage cluster.
- Dashboard is installed.
- Admin-level access to the dashboard.
- Users are added to the dashboard.
- Root-level access on all the hosts.
- Java OpenJDK installed. For more information, see the Installing a JRE on RHEL by using yum section of the *Installing and using OpenJDK 8 for RHEL* guide for OpenJDK on Red Hat Documentation.
- Red Hat Single Sign-On installed from a ZIP file. For more information, see *Installing RH-SSO from a ZIP file* in the Red Hat Single Sign-On 7.6 Server Installation and Configuration Guide on Red Hat Documentation.

Use Red Hat Single Sign-on (SSO) to synchronize users to the Ceph dashboard. See Syncing users to the Ceph dashboard using Red Hat Single Sign-On.

For more information, see the following:

- Creating roles.
- Creating users.

1. Download the *Red Hat Single Sign-On 7.6.0 Server* on the system where Red Hat Ceph Storage is installed. For download package and information, see Software Details for *Red Hat Single Sign-On 7.6.0 Server* on the Red Hat Customer Portal.
2. Extract the folder.

   ```
   unzip rhsso-7.6.0.zip
   ```

   For example,

   ```
   [root@host01 ~]# unzip rhsso-7.6.0.zip
   ```

3. Go to the `standalone/configuration` directory under the newly created `rhsso-7.6.0` directory and open the `standalone.xml` file for editing.
   For example,

   ```
   [root@host01 ~]# cd standalone/configuration
   [root@host01 configuration]# vi standalone.xml
   ```

4. Replace the **`localhost`** and `127.0.0.1` with the IP address of the server where the Red Hat Single-Sign On is installed.
5. From the `bin` directory of the newly created `rhsso-7.6.0` folder, run the `add-user-keycloak` script to add the initial administrator user.

   ```
   ./add-user-keycloak.sh -u admin
   ```

   For example,

   ```
   [root@host01 bin]# add-user-keycloak.sh -u admin
   ```

6. Start the server.
   From the `bin` directory of `rh-sso-7.6` folder, run the `standalone` boot script.

   ```
   ./standalone.sh
   ```

   For example,

   ```
   [root@host01 bin]# ./standalone.sh
   ```

7. Log in to the admin account in `https:IP_ADDRESS:8080/auth` with the new username and password created.

   > **Note:** You must create an admin account only the first time that you log into the console.

Setting up IdP for OAuth2-proxy using Red Hat SSO with OAuth2 protocol

Complete these steps from the Red Hat Single Sign-On console.

1. Create a realm.
   The realm must be **Enabled**.
2. Go to **Realm Settings** and complete the required settings.
   - Toggle **Enabled** to **ON**.
   - Toggle **User-Managed Access** to **ON**.

- Click the link address of the OpenID Endpoint Configuration to retrieve the required endpoint for OAuth2 configuration. The issuer endpoint is used during OAuth2 service configuration.

3. Create a client.
   Set the **Client Protocol** as **openid-connect**.
4. Update the client settings.
   - Set the **Access Type** as **confidential**.
   - Enter the **Valid Redirect URIs**. For example: `<https://<ip|hostname>/oauth2/callback>` and `<https://<ip|hostname>/oauth2/sign_out>`
5. Verify the client credentials.
   Check that the **Client Authenticator** is set as **Client Id and Secret**. The secret is used as the **`client_secret`** parameter when configuring the `oauth2-proxy` service.
6. Create a new administrator role.
   Enter `administrator` as the role name.
7. Update the administrator information.
   a. Go to **Manage › Users › Add User**.
   Complete the **Username**, **Email**, **First Name**, **Last Name**, and toggle **Email Verified** to **ON**.
   b. From **Credentials**, set a new password credentials for the administrator user.
   c. From **Role Mappings**, set the **Client Roles** as the client that was created in step "3" on page 50 and click **Add selected** for **administrator** role created in step "6" on page 50.

Enabling the OAuth2 Proxy service

Enable the OAuth2 Proxy service for SSO access to the Dashboard and the Ceph cluster. Enabling the `oauth2-proxy` service either with the `cephadm` CLI commands or by using a service specification file.

PREREQUISITE

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

- Ceph Management gateway (`mgmt-gateway`) service is enabled and **`--enable-auth`** is set to `true`. For more information, see "Enabling the Ceph Management gateway" on page 41.
- You should either have an admin account with Red Hat Single-Sign-On 7.6.0 with OAuth2 protocol running or IBM Security Verify with OAuth2 protocol running. For more information, see Configuring IBM Security Verify as the identity provider using OAuth2 protocol.

Find the `oauth2-proxy` container image by running the **`ceph config get`** command.

```
ceph config get mgr mgr/cephadm/container_image_oauth2_proxy
```

Storage administrators can specify a custom image by changing the **`container_image_oauth2_proxy`** `cephadm` module option. If other daemons were running, you must redeploy the daemons to use the new image.

```
ceph config set mgr mgr/cephadm/container_image_oauth2_proxy NEW_OAUTH2_PROXY_IMAGE
ceph orch redeploy oauth2_proxy
```

Enabling the OAuth2 Proxy service with the command-line interface

1. Deploy the `oauth2-proxy` service.

   ```
   ceph orch apply oauth2-proxy [--placement=DESTINATION_HOST]
   ```

   For example,

   ```
   [ceph: root@host01 /]# ceph orch apply oauth2-proxy [--placement=host01]
   ```

2. Verify that the service was correctly deployed.

Enabling the OAuth2 Proxy service with a service specification file

PREREQUISITE

Before enabling the OAuth2 Proxy service, make sure that the following are on each Ceph node that the `oauth2-proxy` service will run on.

- Your client ID.
- The OIDC issuer URL.
- Client secret.
- The relevant domain addresses to allow.

  > **Note:** The domain can be the same or different from the **`oidc_issuer_url`**.

- A running Red Hat Ceph Storage cluster.
- (Optional) The host HTTPS address and host port.
- (Optional) Cookie secret.
- (Optional) SSL certificates and certificate keys.

For more information about SSL protocols, ciphers, certificates, and certificate keys, see the *Deploying web servers and reverse proxies* section in the Red Hat Enterprise Linux documentation.

1. Create a YAML file for the `oauth2-proxy` service.
   For example,

   ```
   [root@host01 ~]# touch oauth2-proxy.yaml
   ```

2. Edit the YAML file to include the following details.

   ```
   service_type: oauth2-proxy
   service_id: auth-proxy
   placement:
     hosts:
       - <target-node>
   spec:
    https_address: HTTPS_ADDRESS:PORT
    provider_display_name: MY OIDC PROVIDER
    client_id: CLIENT_ID
    oidc_issuer_url: OIDC ISSUER URL
    allowlist_domains:
       - HTTPS_ADDRESS:PORT
    client_secret: CLIENT_SECRET
    cookie_secret: COOKIE_SECRET
    ssl_cert: |
      -----BEGIN CERTIFICATE-----
      < YOU CERT DATA HERE >
      -----END CERTIFICATE-----
    ssl_key: |
     -----BEGIN RSA PRIVATE KEY-----
      < YOU PRIV KEY DATA HERE >
     -----END RSA PRIVATE KEY-----
   ```

   For example,

```
service_type: oauth2-proxy
service_id: auth-proxy
placement:
  hosts:
    - ceph0
spec:
 https_address: "0.0.0.0:4180"
 provider_display_name: "My OIDC Provider"
 client_id: "your-client-id"
 oidc_issuer_url: "http://192.168.100.1:5556/realms/ceph"
 allowlist_domains:
    - 192.168.100.1:8080
    - 192.168.200.1:5000
 client_secret: "your-client-secret"
 cookie_secret: "your-cookie-secret"
 ssl_cert: |
   -----BEGIN CERTIFICATE-----
   MIIDtTCCAp2gAwIBAgIYMC4xNzc1NDQxNjEzMzc2MjMyXzxvQ7EcMA0GCSqGSIb3
   DQEBCwUAMG0xCzAJBgNVBAYTAlVTMQ0wCwYDVQQIDARVdGFoMRcwFQYDVQQHDA5T
   [...]
   -----END CERTIFICATE-----
ssl_key: |
   -----BEGIN PRIVATE KEY-----
   MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQC5jdYbjtNTAKW4
   /CwQr/7wOiLGzVxChn3mmCIF3DwbL/qvTFTX2d8bDf6LjGwLYloXHscRfxszX/4h
   [...]
   -----END PRIVATE KEY-----
```

“oauth2-proxy specific fields in the spec file” on page 52 lists fields that are specific to the `oauth2-proxy` service section of the spec file (`ceph.deployment.service_spec.OAuth2ProxySpec`).

`oauth2-proxy` specific fields in the spec file

Field	Description
`allowlist_domains`	List of allowed domains for safe redirection after login or logout, preventing unauthorized redirects. This can be the same or different from the **`oidc_issuer_url`**.
`client_id`	The client ID for authenticating with the identity provider.
`client_secret`	The client secret for authenticating with the identity provider.
`cookie_secret`	The secret key is used for signing cookies. Its length must be 16 bytes, 24 bytes, or 32 bytes to create an AES cipher.
`https_address`	The address for HTTPS connections, which are formatted as `host:port`.
`networks`	A list of network identities that instruct the daemons to only bind on the particular networks in that list. You can add multiple networks if the cluster is distributed across multiple networks.
`oidc_issuer_url`	The URL of the OpenID Connect (OIDC) issuer.
`placement`	For the orchestrator to deploy a service, it needs to know where to deploy daemons, and how many to deploy. This is the role of a placement specification. Placement specifications can either be passed as command line arguments or in a YAML files. For more information, see Managing services.
`provider_display_name`	The display name for the identity provider (IdP) in the UI.
`redirect_url`	Optional. The URL oauth2-proxy will redirect to after a successful login. If not provided, the`cephadm`automatically calculates the value of this URL.

Field	Description
ssl_cert	Optional. The multi-line SSL certificate for encrypting communications.
ssl_key	Optional. The multi-line SSL certificate private key for decrypting communications.

3. Apply the specification file.

```
[root@host01 ~]# ceph orch apply -i oauth2-proxy.yaml
```

Infrastructure security

A proper Red Hat Ceph Storage security plan requires Red Hat Enterprise Linux security and SELinux prerequisites.

For full information, see the following Red Hat Enterprise Linux 9 user documentation on Red Hat Documentation.

- *Security Hardening Guide*
- *Using SELinux Guide*

Administration

Use command line tools to administer Red Hat Ceph Storage clusters. The CLI tools require an administrator key for administrator access privileges to the cluster.

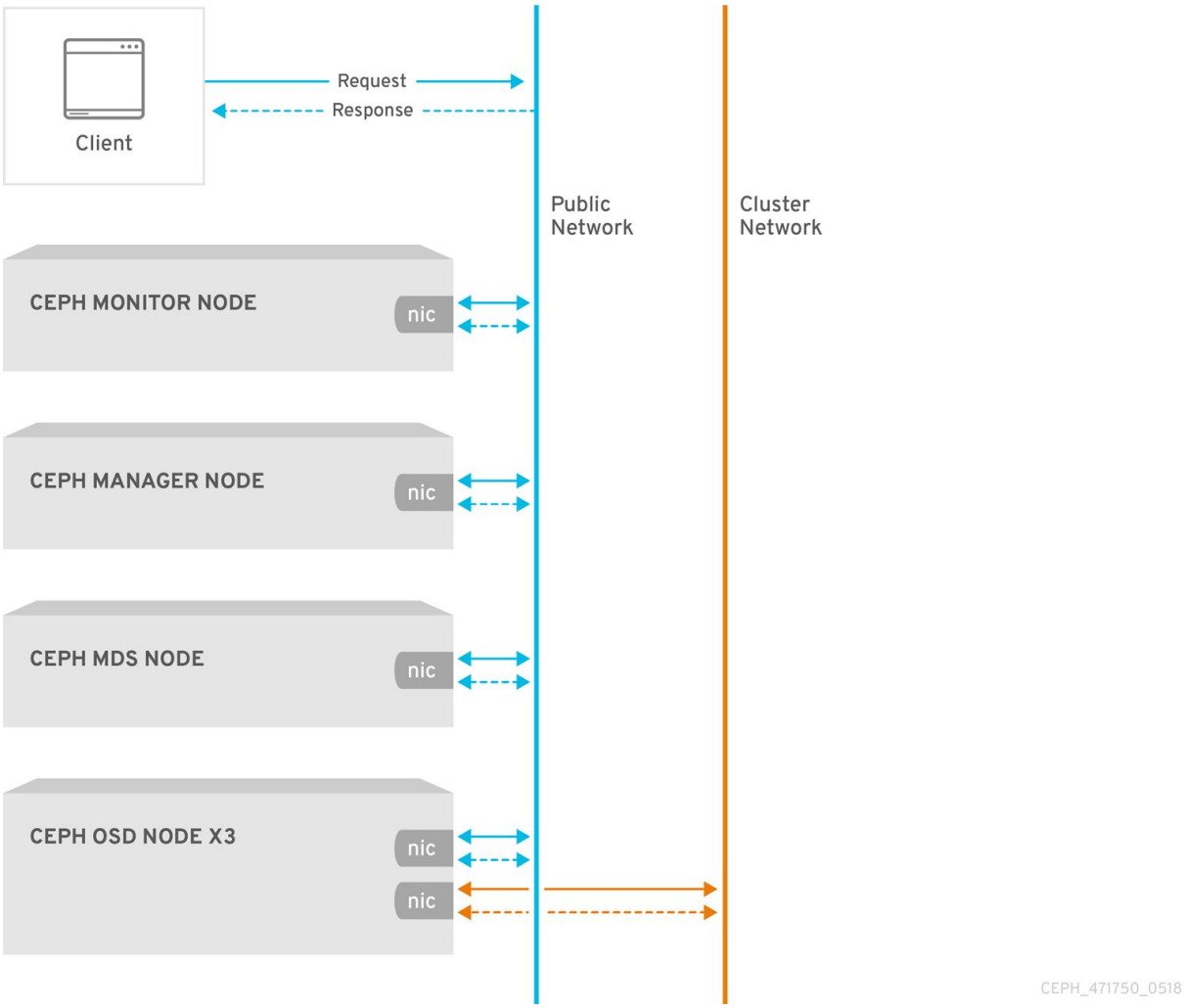
By default, Ceph stores the administrator key in the `/etc/ceph` directory. The default file name is `ceph.client.admin.keyring`. Take steps to secure the keyring so that only a user with administrative privileges to the cluster can access the keyring.

Network communication

Red Hat Ceph Storage provides two networks, both a public and a cluster network.

All Ceph daemons and Ceph clients require access to the public network, which is part of the *storage access security zone*. By contrast, **ONLY** the OSD daemons require access to the cluster network, which is part of the *Ceph cluster security zone*.

Figure: Network architecture



The Ceph configuration contains `public_network` and `cluster_network` settings. For hardening purposes, specify the IP address and the netmask using CIDR notation. Specify multiple comma-delimited IP address and netmask entries if the cluster will have multiple sub-nets.

```
public_network = <public-network/netmask>[,<public-network/netmask>]

cluster_network = <cluster-network/netmask>[,<cluster-network/netmask>]
```

Reference

For more information, see the following:

- Ceph network configuration

Hardening the network service

System administrators deploy Red Hat Ceph Storage clusters on Red Hat Enterprise Linux 9 Server. SELinux is on by default and the firewall blocks all inbound traffic except for the SSH service port 22; however, you **MUST** ensure that this is the case so that no other unauthorized ports are open or unnecessary services are enabled.

On each server node, run the following:

1. Start the `firewalld` service, enable it to run on boot, and ensure that it is running:

```
# systemctl enable firewalld
# systemctl start firewalld
# systemctl status firewalld
```

2. Take an inventory of all open ports.

```
# firewall-cmd --list-all
```

On a new installation, the `sources:` section should be blank indicating that no ports have been opened specifically. The `services` section should indicate `ssh` indicating that the SSH service (and port 22) and `dhcpv6-client` are enabled.

```
sources:
services: ssh dhcpv6-client
```

3. Ensure SELinux is running and `Enforcing`.

```
# getenforce
Enforcing
```

If SELinux is `Permissive`, set it to `Enforcing`.

```
# setenforce 1
```

If SELinux is not running, enable it. For more information, see *Red Hat Enterprise Linux 9 Using SELinux*.

Each Ceph daemon uses one or more ports to communicate with other daemons in the Red Hat Ceph Storage cluster. In some cases, you may change the default port settings. Administrators typically only change the default port with the Ceph Object Gateway or `ceph-radosgw` daemon.

Ceph Ports		
TCP/UDP Port	**Daemon**	**Configuration Option**
`6789, 3300`	`ceph-mon`	N/A
`6800-7300`	`ceph-osd`	`ms_bind_port_min` to `ms_bind_port_max`
`6800-7300`	`ceph-mgr`	`ms_bind_port_min` to `ms_bind_port_max`
`6800`	`ceph-mds`	N/A
`8080`	`ceph-radosgw`	`rgw_frontends`

The Ceph Storage Cluster daemons include `ceph-mon`, `ceph-mgr`, and `ceph-osd`. These daemons and their hosts comprise the Ceph cluster security zone, which should use its own subnet for hardening purposes.

The Ceph clients include `ceph-radosgw`, `ceph-mds`, `ceph-fuse`, `libcephfs`, `rbd`, `librbd`, and `librados`. These daemons and their hosts comprise the storage access security zone, which should use its own subnet for hardening purposes.

On the Ceph Storage Cluster zone's hosts, consider enabling only hosts running Ceph clients to connect to the Ceph Storage Cluster daemons. For example:

```
firewall-cmd --zone=<zone-name> --add-rich-rule="rule family="ipv4" \
source address="<ip-address>/<netmask>" port protocol="tcp" \
port="<port-number>" accept"
```

Replace `<zone-name>` with the zone name, `<ipaddress>` with the IP address, `<netmask>` with the subnet mask in CIDR notation, and `<port-number>` with the port number or range. Repeat the process with the `--permanent` flag so that the changes persist after reboot. For example:

```
firewall-cmd --zone=<zone-name> --add-rich-rule="rule family="ipv4" \
source address="<ip-address>/<netmask>" port protocol="tcp" \
port="<port-number>" accept" --permanent
```

Reporting

Red Hat Ceph Storage provides system monitoring and reporting. System administrators can retrieve configuration settings at runtime.

Red Hat Ceph Storage provides basic system monitoring and reporting with the `ceph-mgr` daemon plug-ins, namely, the RESTful API, the dashboard, and other plug-ins such as `Prometheus` and `Zabbix`. Ceph collects this information using `collectd` and sockets to retrieve settings, configuration details, and statistical information.

In addition to default system behavior, system administrators may configure `collectd` to report on security matters, such as configuring the `IP-Tables` or `ConnTrack` plug-ins to track open ports and connections respectively.

Reference

For more information, see Viewing the Ceph configuration at runtime.

Auditing administrator actions

An important aspect of system security is to periodically audit administrator actions on the cluster.

Red Hat Ceph Storage stores a history of administrator actions in the `/var/log/ceph/`*`CLUSTER_FSID`*`/ceph.audit.log` file.

Example

```
[root@host04 ~]# cat /var/log/ceph/6c58dfb8-4342-11ee-a953-fa163e843234/ceph.audit.log
```

Each entry will contain:

- **Timestamp:** Indicates when the command was run.
- **Monitor Address:** Identifies the monitor modified.
- **Client Node:** Identifies the client node initiating the change.
- **Entity:** Identifies the user making the change.
- **Command:** Identifies the command used.

The following is an output of the Ceph audit log:

```
2023-09-01T10:20:21.445990+0000 mon.host01 (mon.0) 122301 : audit [DBG] from='mgr.14189
10.0.210.22:0/1157748332' entity='mgr.host01.mcadea' cmd=[{"prefix": "config generate-
minimal-conf"}]: dispatch
2023-09-01T10:20:21.446972+0000 mon.host01 (mon.0) 122302 : audit [INF] from='mgr.14189
10.0.210.22:0/1157748332' entity='mgr.host01.mcadea' cmd=[{"prefix": "auth get", "entity":
"client.admin"}]: dispatch
2023-09-01T10:20:21.453790+0000 mon.host01 (mon.0) 122303 : audit [INF] from='mgr.14189
10.0.210.22:0/1157748332' entity='mgr.host01.mcadea'
2023-09-01T10:20:21.457119+0000 mon.host01 (mon.0) 122304 : audit [DBG] from='mgr.14189
10.0.210.22:0/1157748332' entity='mgr.host01.mcadea' cmd=[{"prefix": "osd tree", "states":
["destroyed"], "format": "json"}]: dispatch
2023-09-01T10:20:30.671816+0000 mon.host01 (mon.0) 122305 : audit [DBG] from='mgr.14189
10.0.210.22:0/1157748332' entity='mgr.host01.mcadea' cmd=[{"prefix": "osd blocklist ls",
"format": "json"}]: dispatch
```

In distributed systems such as Ceph, actions may begin on one instance and get propagated to other nodes in the cluster. When the action begins, the log indicates `dispatch`. When the action ends, the log indicates `finished`.

Data retention

Red Hat Ceph Storage stores user data, but usually in an indirect manner. Customer data retention may involve other applications, such as the OpenStack Platform.

Ceph Storage Cluster

The Ceph Storage Cluster, often referred to as the Reliable Autonomic Distributed Object Store or RADOS, stores data as objects within pools. In most cases, these objects are the atomic units representing client data,

such as Ceph Block Device images, Ceph Object Gateway objects, or Ceph File System files. However, custom applications built on top of `librados` might bind to a pool and store data too.

Cephx controls access to the pools storing object data. However, Ceph Storage Cluster users are typically Ceph clients, and not users. Consequently, users generally DO NOT have the ability to write, read or delete objects directly in a Ceph Storage Cluster pool.

Ceph Block Device

The most popular use of Red Hat Ceph Storage, the Ceph Block Device interface, also referred to as RADOS Block Device, or RBD, creates virtual volumes, images, and compute instances and stores them as a series of objects within pools. Ceph assigns these objects to placement groups and distributes or places them pseudo-randomly in OSDs throughout the cluster.

Depending upon the application consuming the Ceph Block Device interface, usually OpenStack Platform, users may create, modify, and delete volumes and images. Ceph handles the create, retrieve, update, and delete operations of each individual object.

Deleting volumes and images destroys the corresponding objects in an unrecoverable manner. However, residual data artifacts may continue to reside on storage media until overwritten. Data may also remain in backup archives.

Ceph File System

The Ceph File System interface creates virtual file systems and stores them as a series of objects within pools. Ceph assigns these objects to placement groups and distributes or places them pseudo-randomly in OSDs throughout the cluster.

Typically, the Ceph File System uses two pools:

- **Metadata:** The metadata pool stores the data of the Ceph Metadata Server (MDS), which generally consists of inodes; that is, the file ownership, permissions, creation date and time, last modified or accessed date and time, parent directory, and so on.
- **Data:** The data pool stores file data. Ceph may store a file as one or more objects, typically representing smaller chunks of file data such as extents.

Depending upon the application consuming the Ceph File System interface, usually OpenStack Platform, users may create, modify, and delete files in a Ceph File System. Ceph handles the create, retrieve, update, and delete operations of each individual object representing the file.

Deleting files destroys the corresponding objects in an unrecoverable manner. However, residual data artifacts may continue to reside on storage media until overwritten. Data may also remain in backup archives.

Ceph Object Gateway

From a data security and retention perspective, the Ceph Object Gateway interface has some important differences when compared to the Ceph Block Device and Ceph File System interfaces. The Ceph Object Gateway provides a service to users. The Ceph Object Gateway may store user authentication information, user data, and logging.

- **User Authentication Information:** User authentication information generally consists of user IDs, user access keys, and user secrets. It may also comprise a user's name and email address if provided. Ceph Object Gateway will retain user authentication data unless the user is explicitly deleted from the system.
- **User Data:** User data generally comprises user- or administrator-created buckets or containers, and the user-created S3 or Swift objects contained within them. The Ceph Object Gateway interface creates one or more Ceph Storage cluster objects for each S3 or Swift object and stores the corresponding Ceph Storage cluster objects within a data pool. Ceph assigns the Ceph Storage cluster objects to placement groups and distributes or places them pseudo-randomly in OSDs throughout the cluster. The Ceph Object Gateway may also store an index of the objects contained within a bucket or index to enable services such as listing the contents of an S3 bucket or Swift container. Additionally, when implementing multi-part uploads, the Ceph Object Gateway may temporarily store partial uploads of S3 or Swift objects.

  Users may create, modify, and delete buckets or containers, and the objects contained within them in a Ceph Object Gateway. Ceph handles the create, retrieve, update, and delete operations of each individual Ceph Storage cluster object representing the S3 or Swift object.

Deleting S3 or Swift objects destroys the corresponding Ceph Storage cluster objects in an unrecoverable manner. However, residual data artifacts may continue to reside on storage media until overwritten. Data may also remain in backup archives.

- **Logging:** Ceph Object Gateway also stores logs of user operations that the user intends to accomplish and operations that have been run. This data provides traceability about who created, modified or deleted a bucket or container, or an S3 or Swift object residing in an S3 bucket or Swift container. When users delete their data, the logging information is not affected and will remain in storage until deleted by a system administrator or removed automatically by expiration policy.

Bucket Lifecycle

Ceph Object Gateway also supports bucket lifecycle features, including object expiration. Data retention regulations like the General Data Protection Regulation may require administrators to set object expiration policies and disclose them to users among other compliance factors.

Multi-site

Ceph Object Gateway is often deployed in a multi-site context whereby a user stores an object at one site and the Ceph Object Gateway creates a replica of the object in another cluster possibly at another geographic location. For example, if a primary cluster fails, a secondary cluster may resume operations. In another example, a secondary cluster may be in a different geographic location, such as an edge network or content-delivery network such that a client may access the closest cluster to improve response time, throughput, and other performance characteristics. In multi-site scenarios, administrators must ensure that each site has implemented security measures. Additionally, if geographic distribution of data would occur in a multi-site scenario, administrators must be aware of any regulatory implications when the data crosses political boundaries.

Federal Information Processing Standard (FIPS)

Red Hat Ceph Storage uses FIPS validated cryptography modules when run on the latest certified Red Hat Enterprise Linux version.

Enable FIPS mode on Red Hat Enterprise Linux either during system installation or after the installation process.

For container deployments, follow the instructions in the Red Hat Enterprise Linux 9 Security Hardening Guide on Red Hat Documentation.

Reference

For more information, see the following:

- *US Government Standards*

Call Home

Call Home connects the system to support personnel who can monitor and respond to system events to ensure that your system remains up and running. You can use Call Home to send automatic delivery of inventory, status, and last contact information, and IBM Storage Insights support.

Call Home is enabled by default during system setup to improve response time for issues on the system. The service automatically sends diagnostic data to support personnel, who can then help quickly identify and resolve issues that could disrupt system operations.

> **Important:** Service events must be enabled for critical event support cases to be opened automatically. For enabling service events, see Enabling service events.

By using Call Home, you agree to allow IBM and its subsidiaries to store and use your contact information and certain system information anywhere they do business worldwide as described in the Program license agreement and documentation. For more information, please refer to the Program license agreement and documentation.

You can setup Call Home with the Ceph Dashboard, `cephadm`, and the `call_home_agent` manager module. View reports that are sent to IBM Support through the Ceph dashboard or using the **`ceph callhome show <report-type>`** command on the command-line interface (CLI).

Call Home periodically sends cluster inventory, status, and last contact information to IBM Support. This information helps support teams better understand the system environment when a service request is opened.

In addition, Call Home uses cloud services to send notifications in certain cases directly to a centralized file repository that contains troubleshooting information that is gathered from your system. Support personnel can review this information to assist with troubleshooting when needed. For more information and assistance with troubleshooting, contact IBM Support.

When enabled, Call Home provides the following benefits:

- Helps to pro-actively detect issues
  - Call Home data helps IBM Support identify issues and recommend corrective actions
- Generates reports based on customer insights for support and development
- Detects critical system warnings or failures and pro-actively opens support tickets

For more information, see IBM Storage Ceph Call Home on IBM Support.

If required, you can disable Call Home. For more information, see Disabling Call Home using Dashboard or Disabling Call Home using CLI.

For details about configuring Call Home see Configuring Call Home using the command-line interface and Configuring IBM Call Home and IBM Storage Insights.

IBM Storage Insights

IBM Storage Insights is a cloud-based service that is a part of the IBM Storage family of products. When using Call Home, you can set up and use IBM Storage Insights to gain an overview of all the deployed Ceph clusters and their statuses. Using IBM Storage Insights can help with data-driven decisions, based on actual usage of the Ceph product.

Red Hat Ceph Storage gives user access to IBM Storage Insights Pro.

> **Important:** To use IBM Storage Insights, Call Home must be enabled on your Red Hat Ceph Storage system. Data being sent to and from IBM Storage Insights Red Hat Ceph Storage uses IBM Call Home as an intermediary.

Using IBM Storage Insights streamlines the support and storage usage by implementing the following features:

- IBM Storage Insights Support can retrieve diagnostic information from a Ceph cluster.
- IBM Storage Insights receives the same important events as Call Home. When an important event occurs within the Ceph cluster, the Call Home manager module generates a report that is sent directly to IBM Storage Insights. An important event is defined as changes that can impact the availability or performance of the overall cluster, such as a monitoring alert or a change in the cluster composition.

IBM Storage Insights is enabled separately from Call Home, but depends on Call Home enablement.

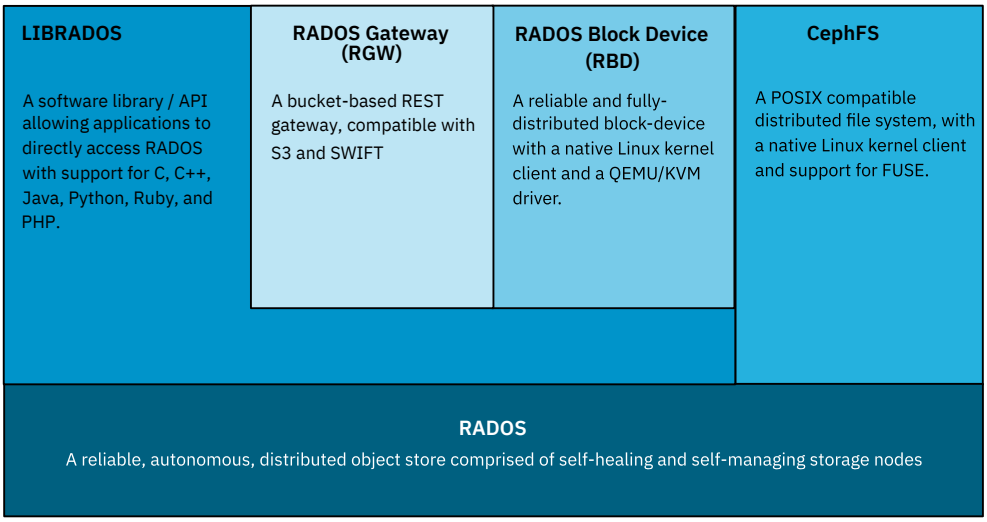
- For more information about configuring by using the CLI, see Enabling IBM Storage Insights support.
- For more information about configuring by using the Ceph Dashboard, see Configuring IBM Call Home and .

Ceph File System

Ceph File System (CephFS) is a distributed file system that integrates seamlessly with the Ceph storage architecture. By leveraging the Ceph RADOS (Reliable Autonomic Distributed Object Store), CephFS provides a scalable and robust file system interface, adhering to POSIX standards.

“Figure: Ceph cluster integration” on page 59 illustrates integration of Ceph within the Ceph cluster.

Figure: Ceph cluster integration



CephFS-based file sharing offers the same desirable features as other storage services on the Ceph platform, including scalability, high availability, and operational efficiency. The following are key Ceph components that enable CephFS to achieve these benefits:

Object Storage Daemons (OSDs)

OSDs are responsible for storing the actual file data in CephFS. Files and directories are represented as objects within RADOS, with OSDs managing data storage, retrieval, and replication to ensure durability and availability.

Ceph Monitors (MONs)

MONs maintain the cluster map and monitor the health of the Ceph cluster. They track the state of OSDs and other components, ensuring operational stability and fault tolerance. For CephFS, MONs facilitate the mapping of file system operations to the underlying object storage.

Ceph Managers (MGRs)

MGRs handle cluster management, monitoring, and administrative tasks. They provide an interface for managing the Ceph cluster and integrate with external services like OpenStack Manila and Kubernetes CSI for additional functionalities.

Ceph Metadata Servers (MDSs)

MDSs manage CephFS metadata, including file and directory operations, access control, and namespace organization. They coordinate with OSDs to ensure consistent and efficient metadata operations.

Scalability

CephFS scales horizontally with the Ceph cluster, allowing for the addition of OSDs, MONs, and MDSs to increase storage capacity and performance, accommodating large data volumes and high client request rates.

Fault tolerance

Inheriting Ceph fault tolerance, CephFS ensures data durability and availability through replication and erasure coding provided by RADOS, even in the event of hardware failures.

Unified storage platform

CephFS integrates with other Ceph storage services such as block storage, object storage, offering a unified platform that simplifies management and supports diverse storage needs.

Use cases for CephFS

CephFS supports a range of use cases, demonstrating its scalability and flexibility to meet various storage needs. These include:

Native CephFS clients

Clients can mount CephFS shares directly using the kernel client (kcephfs) or FUSE (ceph-fuse), benefiting from Ceph's high throughput and low latency.

Kubernetes storage

CephFS can serve as backend storage for Kubernetes via the Container Storage Interface (CSI) driver, facilitating dynamic provisioning and management of volumes for containerized applications.