Red Hat Ceph Storage Edge clusters

Edge clusters are a solution for cost-efficient object storage configurations.

Red Hat supports the following minimum configuration of a Red Hat Ceph Storage cluster:

- A three node cluster with two replicas for SSDs.
- A four-node cluster with three replicas for HDDs.
- A four-node cluster with EC pool with 2+2 configuration.
- A four-node cluster with 8+6 CRUSH MSR configuration.

With smaller clusters, the utilization goes down because of the amount of usage and the loss of resiliency.

Pools overview

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

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

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

Creating a cluster handle and connecting to the cluster

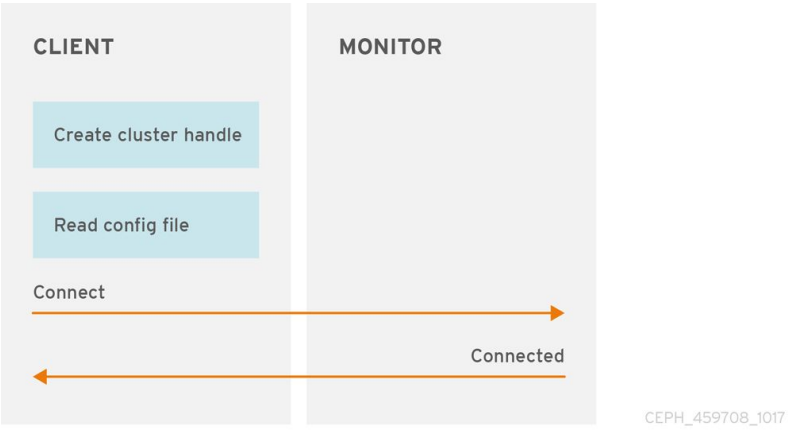
To connect to the Ceph storage cluster, the Ceph client needs the following details:

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

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

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

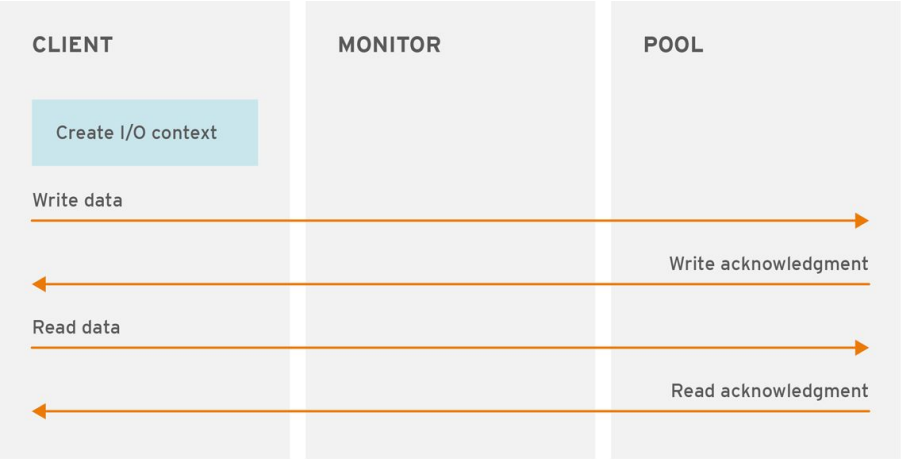
Figure: Create handle



Creating a pool I/O context

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

Figure: I/O Context



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

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

Resilience

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

Placement groups

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

CRUSH rules

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

Quotas

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

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

Resilient and non-resilient data pools

Resilient data pools enable data to replicate or encode its data. A non-resilient data pool does not enable to replicate or encode its data. Non-resilient pools are also known as *replical*.

A non-resilient pool is not protected from data loss. A small cluster with non-resilient data pools does its own replication and does not need replication from Red Hat Ceph Storage as it does its own replication.

Cluster utilization with data pools

With smaller configurations, the cluster utilization reduces because of the loss of resiliency. The recovery is limited by host utilization and might potentially impact production input/output operations per second (IOPS). When there is a single node of failure, you can add a third node to limit the host utilization and for Red Hat Ceph Storage to recover to full replication.

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.

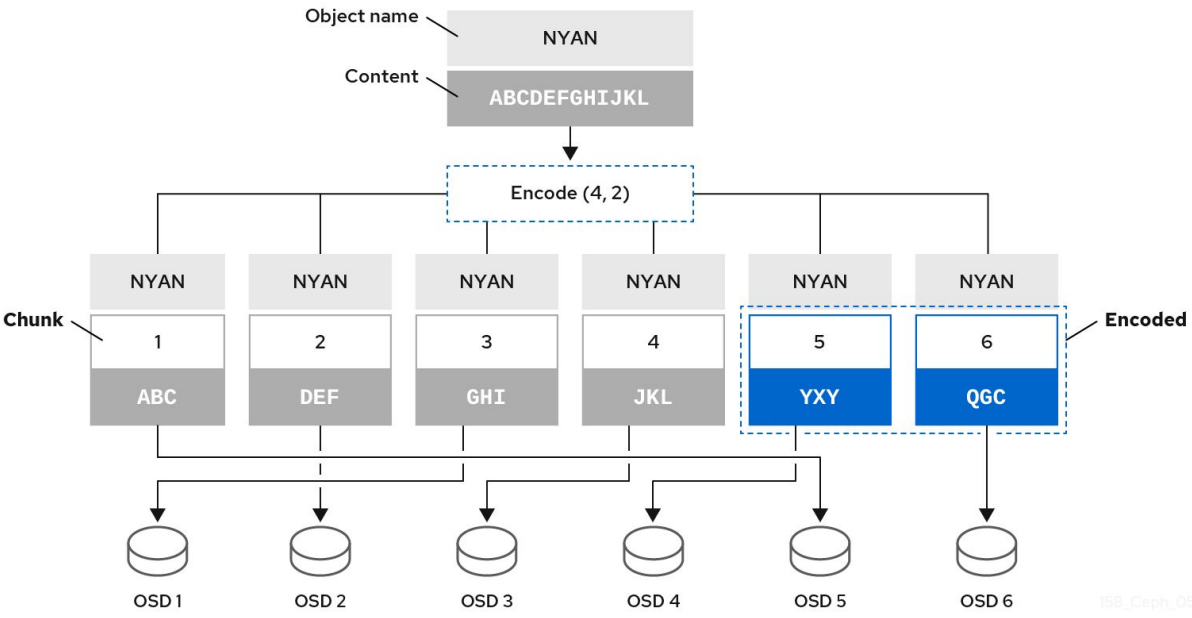
More specifically, `N = K+M` where the variable `K` is the original number of data chunks. The variable `M` stands for the extra or redundant chunks that the erasure code algorithm adds to provide protection from failures. The variable `N` is the total number of chunks that are created after the erasure 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.

Erasure code pools overview

Ceph uses replicated pools by default, meaning that Ceph copies every object from a primary OSD node to one or more secondary OSDs. The erasure-coded pools reduce the amount of disk space required to ensure data durability but it is computationally a bit more expensive than replication.

Ceph storage strategies involve defining data durability requirements. Data durability means the ability to sustain the loss of one or more OSDs without losing data.

Ceph stores data in pools and there are two types of the pools:

- replicated
- erasure-coded

Erasure coding is a method of storing an object in the Ceph storage cluster durably where the erasure code algorithm breaks the object into data chunks (k) and coding chunks (m), and stores those chunks in different OSDs.

In an event of the failure of an OSD, Ceph retrieves the remaining data (k) and coding (m) chunks from the other OSDs and the erasure code algorithm restores the object from those chunks.

Note: Use min_size for erasure-coded pools to be K+1 or more to prevent loss of writes and data. For a 2+2 pool, the min_size should be k+1 or 3.

Erasure coding uses storage capacity more efficiently than replication. The n-replication approach maintains n copies of an object (3 times by default in Ceph), whereas erasure coding maintains only k + m chunks. For example, 3 data and 2 coding chunks use 1.5 times the storage space of the original object.

While erasure coding uses less storage overhead than replication, the erasure code algorithm uses more RAM and CPU than replication when it accesses or recovers objects. Erasure coding is advantageous when data storage must be durable and fault tolerant, but do not require fast read performance (for example, cold storage, historical records, and so on).

For the mathematical and detailed explanation on how erasure code works in Ceph, see the Erasure coding.

Ceph creates a default erasure code profile when initializing a cluster with **k=2** and **m=2**, This mean that Ceph will spread the object data over four OSDs **(k+m = 4)** and Ceph can lose one of those OSDs without losing data. To know more about erasure code profiling see “Erasure code profiles” on page 0 section.

> **Important:** Configure only the `.rgw.buckets` pool as erasure-coded and all other Ceph Object Gateway pools as replicated, otherwise an attempt to create a new bucket fails with the following error:
>
> `set_req_state_err err_no=95 resorting to 500`
>
> The reason for this is that erasure-coded pools do not support the `omap` operations and certain Ceph Object Gateway metadata pools require the `omap` support.

Creating a sample erasure-coded pool

Create an erasure-coded pool and specify the placement groups.

The **`ceph osd pool create`** command creates an erasure-coded pool with the default profile, unless another profile is specified. Profiles define the redundancy of data by setting two parameters, **k**, and **k**. These parameters define the number of chunks a piece of data is split and the number of coding chunks are created.

The simplest erasure-coded pool is equivalent to RAID5 and requires at least four hosts. You can create an erasure-coded pool with 2+2 profile.

1. Set the following configuration for an erasure-coded pool on four nodes with 2+2 configuration.

   ```
   ceph config set mon mon_osd_down_out_subtree_limit host
   ```

   > **Important:** This is not needed for an erasure-coded pool in general.

   > **Note:** For an EC cluster with four nodes the value of K+M is 2+2. If a node fails completely, it does not recover as four chunks and only three nodes are available. When you set the value of **`mon_osd_down_out_subtree_limit`** to `host`, during a host down scenario, it prevents the OSDs from marked out, so as to prevent the data from re balancing and the waits until the node is up again.

2. For an erasure-coded pool with a 2+2 configuration, set the profile.

   ```
   ceph osd erasure-code-profile set ec22 k=2 m=2 crush-failure-domain=host
   ```

   ```
   [ceph: root@host01 /]# ceph osd erasure-code-profile set ec22 k=2 m=2 crush-failure-
   domain=host

   Pool : ceph osd pool create test-ec-22 erasure ec22
   ```

3. Create an erasure-coded pool.

   ```
   ceph osd pool create ecpool 32 32 erasure
   pool 'ecpool' created
   $ echo ABCDEFGHI | rados --pool ecpool put NYAN -
   $ rados --pool ecpool get NYAN -
   ABCDEFGHI
   ```

   `32` is the number of placement groups.

Bluestore server-side compression

BlueStore applies compression based on pool settings, the configured compression mode, and object characteristics. Each blob (data fragment) is evaluated individually.

How compression is applied depends on pool settings and object access patterns. Object characteristics determine which blob size BlueStore uses, not whether compression is enabled:

- If an object is immutable, append-only, or expected to have sequential reads, BlueStore uses `compression_max_blob_size`.
- If the object is expected to have random reads or writes, BlueStore uses `compression_min_blob_size`.

Benefits

Following are the benefits of using the Bluestore server-side compression feature:

- Store more data without adding physical capacity.
- Reduce raw disk usage and infrastructure costs.
- Enable compression selectively per pool to balance performance and efficiency.

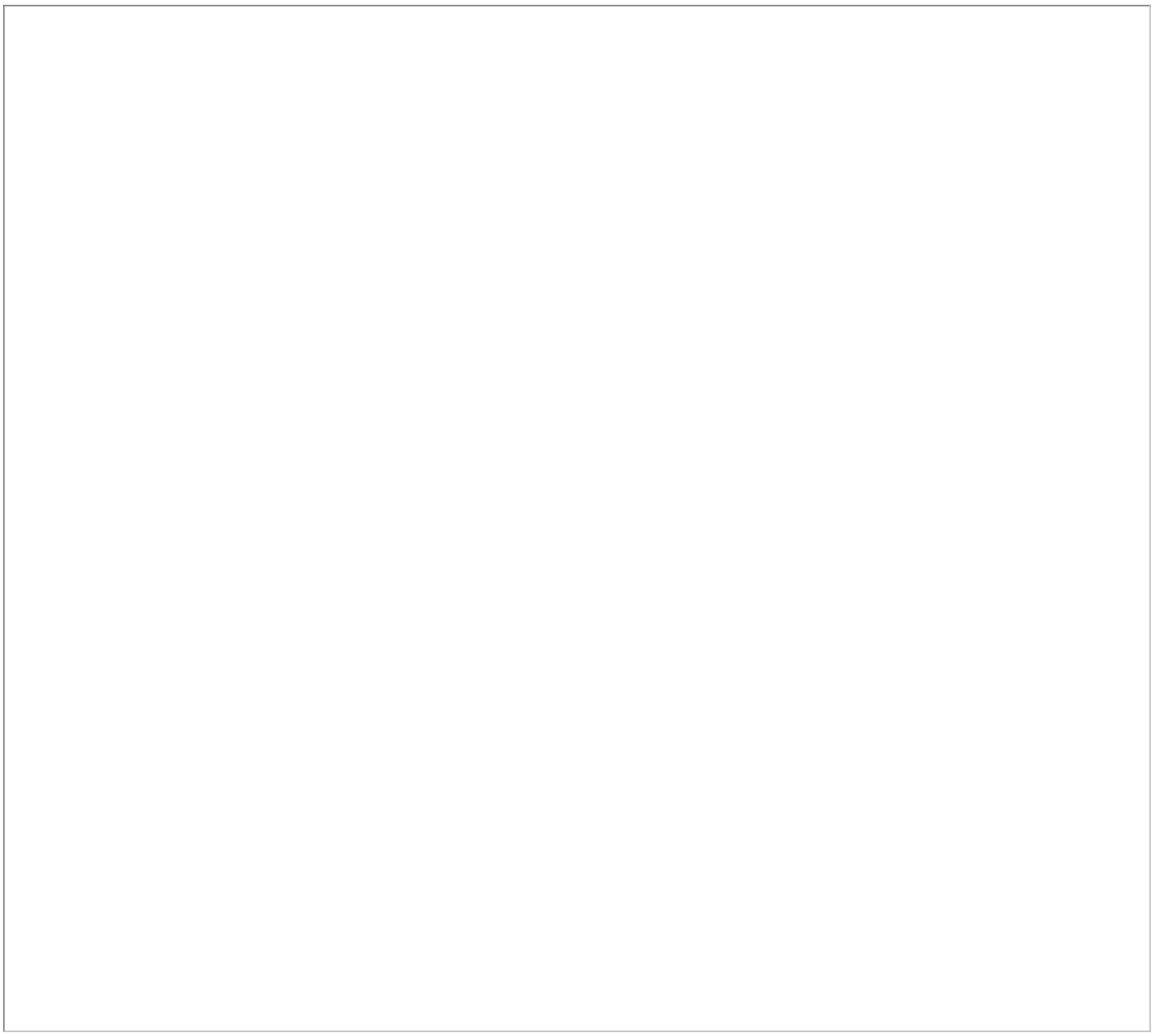
Considerations

Compression introduces CPU overhead and may slightly increase read and write latency. In some cases, small overwrites on compressed objects can increase space usage because overlapping data fragments require both the old and new blobs to be retained. Enable compression only on pools or workloads where data-reduction benefits outweigh performance costs, and avoid enabling it cluster-wide without testing workload patterns.

How compression works

BlueStore applies compression based on pool settings and object characteristics. Each blob (data fragment) is evaluated individually. Compression is applied when an object is marked as immutable or append-only, not expected to have random writes or reads, and expected to have sequential reads. If these conditions are met, BlueStore uses the `compression_max_blob_size` parameter; otherwise, it uses `compression_min_blob_size`. Blob size determines how data is divided and aligned before compression. Larger blobs generally improve compression efficiency, while smaller blobs improve read performance because less data must be decompressed.

Figure 1 shows how BlueStore selects a blob size based on object properties.
Figure: Bluestore compression workflow



BlueStore does not use a continuous size range; instead, it selects one of the two blob sizes based on allocation hints sent through the `rados_set_alloc_hint2` API call. Ceph clients such as RBD, CephFS, and RGW already use these hints to optimize performance.

Compression behavior

BlueStore compresses each write operation according to its configured mode, ratio threshold, and blob-size limits. In *none*mode, compression is never applied. In *passive* mode, it occurs only if the client provides a compressible hint. In *aggressive* mode, BlueStore compresses unless the client marks data as incompressible. In *force* mode, BlueStore always attempts to compress data.

The `compression_required_ratio` parameter controls the minimum acceptable ratio. For example, if set to 0.7, compression occurs only when the resulting data is 70 percent or smaller than the original. If the threshold is not met, the data is stored uncompressed. This check is performed for each blob individually to prevent unnecessary CPU usage and space amplification.

Blob size is governed by `compression_min_blob_size` and `compression_max_blob_size`, and BlueStore supports the snappy, zlib, lz4, and zstd algorithms. When configured appropriately, BlueStore balances capacity savings with performance efficiency across Red Hat Ceph Storage clusters.

Configuring BlueStore compression for a specific pool

Use these commands to configure compression settings for a specific pool. These settings override global defaults.

1. Set the compression algorithm for the pool.

```
ceph osd pool set POOL_NAME compression_algorithm ALGORITHM
```

For example,

```
[ceph: root@host01 /]# ceph osd pool set test_pool compression_algorithm snappy
set pool 2 compression_algorithm to snappy
```

2. Set the compression mode for the pool.

```
ceph osd pool set POOL_NAME compression_mode MODE
```

For example,

```
[ceph: root@host01 /]# ceph osd pool set test_pool compression_mode force
set pool 2 compression_mode to force
```

3. Set the required compression ratio for the pool.

```
ceph osd pool set POOL_NAME compression_required_ratio RATIO
```

> **Note:** Compression occurs only when compressed data size is less than or equal to 70% of the original size.

For example,

```
[ceph: root@host01 /]# ceph osd pool set test_pool compression_required_ratio 0.7
```

4. Set minimum and maximum blob sizes for the pool.

```
ceph osd pool set POOL_NAME compression_min_blob_size SIZE
ceph osd pool set POOL_NAME compression_max_blob_size SIZE
```

5. Verify the per-pool compression settings.

```
ceph osd pool get POOL_NAME OPTION_NAME
```

For example,

```
[ceph: root@host01 /]# ceph osd pool get test_pool compression_algorithm
compression_algorithm: snappy
```

Configuring global BlueStore compression

Use the **`ceph config set`** commands to configure compression across all OSDs in the cluster.

1. Set the global compression algorithm.

```
ceph config set osd bluestore_compression_algorithm ALGORITHM
```

The default value is **snappy**. This parameter is optional.

2. Set the global compression mode.

```
ceph config set osd bluestore_compression_mode MODE
```

The supported *MODE* inputs are:

- none (default)
- `passive`
- `aggressive`
- `force`

3. Set the global compression ratio.

```
ceph config set osd bluestore_compression_required_ratio RATIO
```

The default value is `0.875`. A value of `0.7` means that compression is applied only if the compressed data is 70% or smaller than the original.

4. Set the minimum and maximum blob sizes.

```
ceph config set osd bluestore_compression_min_blob_size SIZE
ceph config set osd bluestore_compression_max_blob_size SIZE
```

The following are the default blob sizes:

SSD

65536 bytes (minimum and maximum)

HDD

8192 bytes (minimum) and 65536 bytes (maximum)

Removing global BlueStore compression

Remove global BlueStore compression settings when you want all OSDs to revert to their default values. Removing these options stops applying the overridden compression behavior for future writes; existing data is not modified.

1. Remove the global compression algorithm.
   This command overrides any set algorithms so that OSDs stop forcing a specific compression algorithm.

```
ceph config rm osd bluestore_compression_algorithm
```

2. Remove the global compression mode.
   This command clears the compression mode (`none`, `passive`, `aggressive`, or `force`) and returns OSDs to the default mode.

```
ceph config rm osd bluestore_compression_mode
```

3. Remove the global required ratio.
   This command removes the threshold used to decide whether compressed data should be kept.

```
ceph config rm osd bluestore_compression_required_ratio
```

4. Remove the minimum blob size.
   This command removes the configured minimum blob size so that OSDs use the default value based on the device type.

```
ceph config rm osd bluestore_compression_min_blob_size
```

5. Remove the maximum blob size.
   This command removes the configured maximum blob size so that OSDs use the default value based on the device type.

```
ceph config rm osd bluestore_compression_max_blob_size
```

Cluster topology and colocation

Understand the topology needed and the factors to be considers for collocation for an edge cluster.

For information on cluster topology, hyper convergence with OpenStack, collocating nodes On OpenStack, and limitations of OpenStack minimum configuration, see Ceph configuration overrides for HCI.

For more information on colocation, see Colocation.