Redbooks
ibm.com/redbooks

IBM Storage Ceph Concepts and Architecture Guide

Vasfi Gucer

Jussi Lehtinen

Jean-Charles (JC) Lopez

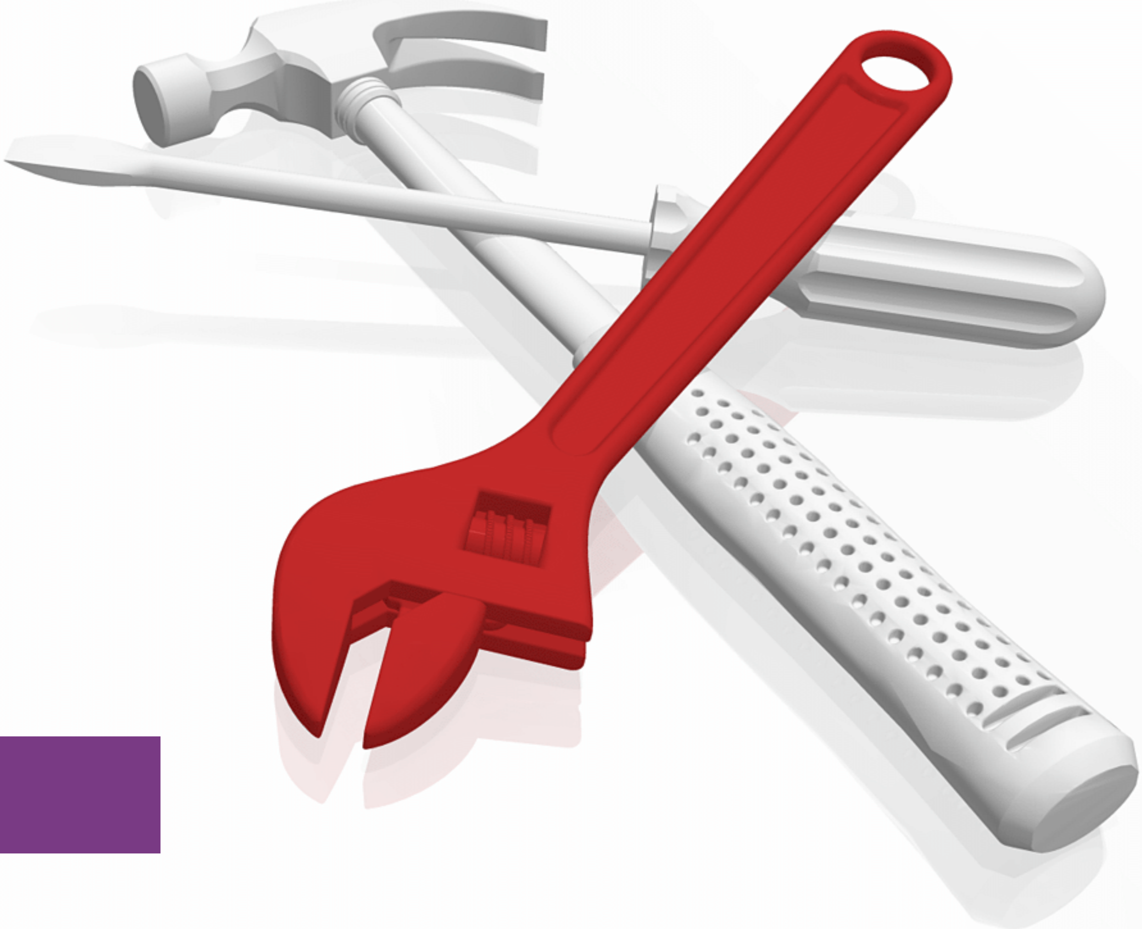
Christopher Maestas

Franck Malterre

Suha Ondokuzmayis

Daniel Parkes

John Shubeck



Storage

IBM

Redpaper

IBM Redbooks

IBM Storage Ceph Concepts and Architecture Guide

June 2024

REDP-5721-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (June 2024)

This edition applies to IBM Storage Ceph Version 7.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at `https://www.ibm.com/legal/copytrade.shtml`

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

IBM®
IBM Cloud®
IBM Spectrum®
Redbooks®
Redbooks (logo) ®

The following terms are trademarks of other companies:

ITIL is a Registered Trade Mark of AXELOS Limited.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat, Ansible, Ceph, OpenShift, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

VMware, and the VMware logo are registered trademarks or trademarks of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM Storage Ceph is an IBM® supported distribution of the open-source Ceph platform that provides massively scalable object, block, and file storage in a single system.

IBM Storage Ceph is designed to infuse AI with enterprise resiliency, consolidate data with software simplicity, and run on multiple hardware platforms to provide flexibility and lower costs.

Engineered to be self-healing and self-managing with no single point of failure, it includes storage analytics for critical insights into growing amounts of data. IBM Storage Ceph can be used as a simple and efficient way to build a data lakehouse for IBM watsonx.data and for next-generation AI workloads.

This IBM Redpaper publication explains the concepts and architecture of IBM Storage Ceph in a clear and concise way. For more information about how to implement IBM Storage Ceph for real-life solutions, see *IBM Storage Ceph Solutions Guide*, REDP-5715.

The target audience for this publication is IBM Storage Ceph architects, IT specialists, and technologists.

Authors

This paper was produced by a team of specialists from around the world.

Vasfi Gucer works as the Storage Team Leader on the IBM Redbooks® Team. He has more than 30 years of experience in the areas of systems management, networking hardware, and software. He writes extensively and teaches IBM classes worldwide about IBM products. For the past decade, his primary focus has been on storage, cloud computing, and cloud storage technologies. Vasfi is also an IBM Certified Senior IT Specialist, Project Management Professional (PMP), IT Infrastructure Library (ITIL) V2 Manager, and ITIL V3 Expert.

Jussi Lehtinen is a Principal Storage subject matter expert for IBM Object Storage who works for IBM Infrastructure in the Nordics, the Benelux, and Eastern Europe. He has over 35 years of experience working in IT, with the last 25 years with Storage. He holds a bachelor’s degree in Management and Computer Studies from Webster University in Geneva, Switzerland.

Jean-Charles (JC) Lopez has been in storage for 80% of his professional career, and has been working in software-defined storage since 2013. JC helps people understand how they can move to a containerized environment when it comes to data persistence and protection. His go-to solutions are Fusion, Ceph, and Red Hat OpenShift. He takes part in the Upstream Community and downstream versions of Ceph and Red Hat OpenShift.

Christopher Maestas is the Chief Executive Architect for IBM File and Object Storage Solutions with over 25 years of experience deploying and designing IT systems for clients in various spaces. He has experience in scaling performance and availability with various file systems technologies. He has developed benchmark frameworks to test out systems for reliability and validate research performance data. He also has led global enablement sessions online and face to face, where he described how best to position mature technologies like IBM Spectrum® Scale with emerging technologies in the cloud, object, container, or AI spaces.

Franck Malterre is an IT professional with a background of over 25 years in designing, implementing, and maintaining large x86 physical and virtualized environments. For the last 10 years, he has specialized in the IBM File and Object Storage Solution by developing IBM Cloud® Object Storage, IBM Storage Scale, and IBM Storage Ceph live demonstrations and proofs of concept for IBM personnel and IBM Business Partners.

Suha Ondokuzmayis began his career in 2001 as an assembly language developer intern at Turkish Airlines, which was followed by working at Turkcell, Turkish Telekom, IBM, and Veritas Technologies within different departments, all with infrastructure-related responsibilities. He joined İşbank in 2013, where he has worked with IT infrastructures, IaaS and PaaS platforms, and block, file, and object storage product offerings, and backup operations. He holds a Master of Science degree in Management Information Systems from the University of Istanbul.

Daniel Parkes works in the IBM Storage Ceph Product Management team, where he focuses on the IBM Storage Ceph Object Storage offering.

John Shubeck is an IT professional with over 42 years of industry experience across both the customer and technology provider perspectives. John serves in the IBM Advanced Technology Group as a Senior Storage Technical Specialist on IBM Object Storage platforms across the Americas. He has participated in many technology conferences during the past 20 years as a facilitator, presenter, teacher, and panelist.

Thanks to the following people for their contributions to this project:

Kenneth David Hartsoe, Elias Luna, Henry Vo, Wade Wallace, William West
IBM US

Marcel Hergaarden
IBM Netherlands

The team extends its gratitude to the Upstream Community, IBM, and Red Hat Ceph Documentation teams for their contributions to continuously improve Ceph documentation.

The authors would like to express their sincere gratitude to Anthony D'Atri of Red Hat Inc for his valuable contributions to this publication through his thorough review of the book.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on LinkedIn:

  https://www.linkedin.com/groups/2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/subscribe

- Stay current on recent Redbooks publications with RSS Feeds:

  https://www.redbooks.ibm.com/rss.html

1

Introduction

This chapter introduces the origins of Ceph and the basic architectural concepts that are used by this software-defined solution.

This chapter has the following sections:

- History
- Ceph and storage challenges
- What is new with IBM Storage Ceph 7.0

1.1 History

The Ceph project emerged from a critical observation: the Lustre architecture was inherently limited by its metadata lookup mechanism. In Lustre, locating a specific file requires querying a dedicated software component that is called the Metadata Server (MDS). This centralized approach proved to be a bottleneck under heavy workloads, which hindered the overall performance and scalability of the storage system.

To address the inherent Lustre architecture problem, Sage Weil developed Ceph, a distributed storage system that uses a pseudo-random placement algorithm *Controlled Replication Under Scalable Hashing (CRUSH)* to distribute data across a heterogeneous cluster. The novel algorithm leverages a sophisticated calculation to optimally place and redistribute data across the cluster, which minimizes data movement when the cluster's state changes. While Ceph employs a distributed approach to metadata management, it does utilize a metadata pool to store essential file system information.

It is designed to distribute the data across all devices in the cluster to avoid the classic bias of favoring empty devices to write new data. During cluster expansion, this approach will likely generate a bottleneck because new empty devices are favored to receive all the new writes while generating unbalanced data distribution because the old data is not redistributed across all the devices in the storage cluster.

As CRUSH is designed to distribute and maintain the distribution of the data throughout the lifecycle of the storage cluster (expansion, reduction, or failure), it favors an equivalent mixture of old and new data on each physical disk of the cluster, which leads to a more even distribution of the I/Os across all the physical disk devices.

To enhance data distribution, the solution was designed to break down large elements (for example, a 100 GiB file) into smaller elements, each assigned a specific placement through the CRUSH algorithm. Therefore, reading a large file leverages multiple physical disk drives rather than a single disk drive as if the file had been kept as a single element.

Sage Weil prototyped the new algorithm and doing so created a new distributed storage software solution: Ceph. The name *Ceph* was chosen as a reference to the ocean and the life that it harbors. Ceph is a short for *cephalopod*.

> **Note:** Cephalopods have multiple arms or tentacles that can operate independently yet in a coordinated manner. This reflects Ceph's ability to distribute data and operations across numerous storage devices in a cluster, working in parallel for high performance and redundancy.

On January 2023, all Ceph developers and product managers were moved from Red Hat to IBM to provide greater resources for the future of the project. The Ceph project at IBM remains an open-source project. Code changes still follow the upstream first rule, and Ceph is the base for the IBM Storage Ceph software-defined storage product.

Figure 1-1 on page 3 represents the milestones of the Ceph project over the past two decades.

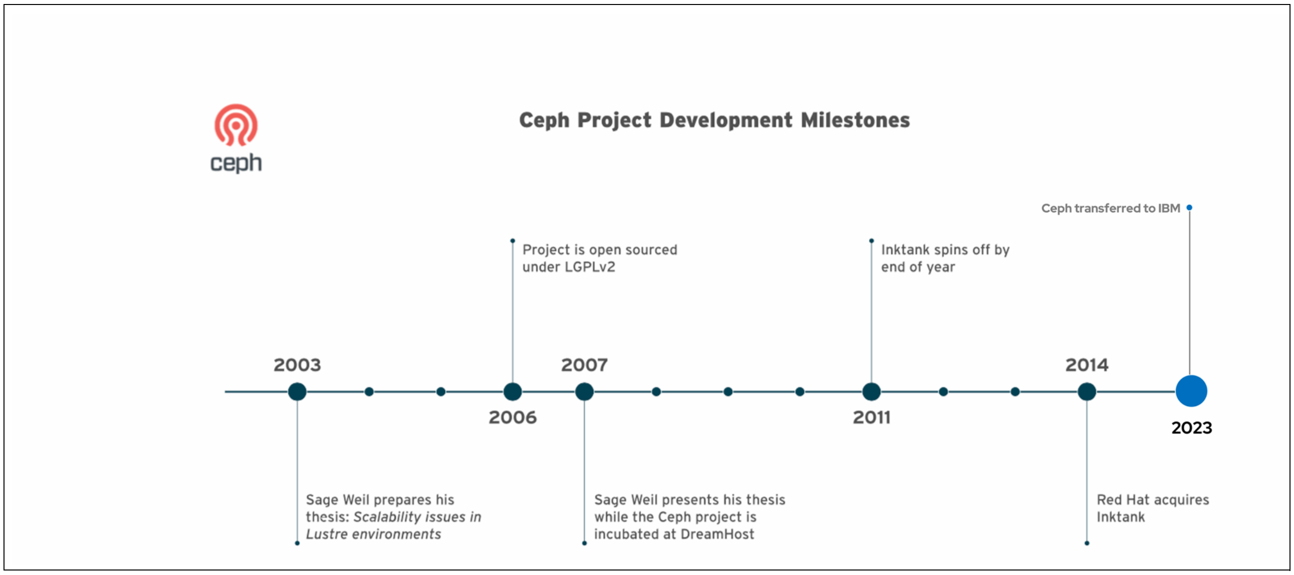

ceph
Ceph Project Development Milestones
Ceph transferred to IBM
Project is open sourced under LGPLv2
Inktank spins off by end of year
2003
2007
2014
2006
2011
2023
Sage Weil prepares his thesis: *Scalability issues in Lustre environments*
Sage Weil presents his thesis while the Ceph project is incubated at DreamHost
Red Hat acquires Inktank

Figure 1-1 Ceph project timeline

All Ceph community versions were assigned the name of a member of the Cephalopoda natural sciences family. The first letter of the name helps identify the version.

Table 1-1 represents all Ceph community version names with the matching Inktank, Red Hat Ceph Storage, or IBM Storage Ceph version that is leveraging it.

Table 1-1 Upstream and downstream Ceph versions

Name	Upstream Version	Release	End of life	Downstream product
Argonaut	0.48	2012-07-03	N/A	
Bobtail	0.56	2013-01-01	N/A	Inktank Ceph Enterprise 1.0
Cuttlefish	0.61	2013-05-07	N/A	
Dumpling	0.67	2013-08-01	2015-05-01	Inktank Ceph Enterprise 1.1
Emperor*	0.72	2013-11-01	2014-05-01	
Firefly	0.80	2014-05-01	2016-04-01	Red Hat Ceph Storage 1.2 Inktank Ceph Enterprise 1.2
Giant*	0.87	2014-10-01	2015-04-01	
Hammer	0.94	2015-04-01	2017-08-01	Red Hat Ceph Storage 1.3
Infernos*	9.2.1	2015-11-01	2016-04-01	
Jewel	10.2	2016-04-01	2018-07-01	Red Hat Ceph Storage 2
Kurgan*	11.2	2017-01-01	2017-08-01	
Luminous	12.2	2017-08-01	2020-03-01	Red Hat Ceph Storage 3
Mimic*	13.2	2018-06-01	2020-07-22	
Nautilus	14.2	2019-03-19	2021-06-30	Red Hat Ceph Storage 4
Octopus*	15.2	2020-03-23	2022-08-09	

Name	Upstream Version	Release	End of life	Downstream product
Pacific	16.2	2021-03-31	2023-10-01	IBM Storage Ceph 5 (start with 5.3) Red Hat Ceph Storage 5
Quincy	17.2	2022-04-19	2024-06-01	IBM Storage Ceph 6 (start with 6.1) Red Hat Ceph Storage 6
Reef	18.2	2023-08-07	2025-08-01	IBM Storage Ceph 7 (start with 7.0) Red Hat Ceph Storage 7

Ceph releases marked with an asterisk in Table 1-1 indicate development versions with short lifespans. These releases, lacking a corresponding downstream product, were not used by Inktank, Red Hat, or IBM to create long-term supported products. Starting with the Pacific release, the Ceph project adopted a new release cycle: yearly Long Term Support (LTS) versions without intermediate development releases. Consequently, Red Hat/IBM major releases now directly mirror community major releases.

Table 1-2 shows the downstream product lifecycle.

Table 1-2 Downstream product lifecycle

Product	General availability date	End of life	Extended Support Available until
Inktank Ceph Enterprise 1.1		2015-07-31	N/A
Inktank Ceph Enterprise 1.2 Red Hat Ceph Storage 1.2		2016-05-31	N/A
Red Hat Ceph Storage 1.3		2018-06-30	N/A
Red Hat Ceph Storage 2		2019-12-16	2021-12-17
Red Hat Ceph Storage 3		2021-02-28	2023-06-27
Red Hat Ceph Storage 4		2023-03-31	2025-04-30
Red Hat Ceph Storage 5.3 IBM Storage Ceph 5.3	2023-03-10	2024-08-31	2027-07-31
Red Hat Ceph Storage 6.1 IBM Storage Ceph 6.1	2023-08-18	2026-03-20	2028-03-20
Red Hat Ceph Storage 7.0 IBM Storage Ceph 7.0	2023-12-08		

Following the transfer of the Ceph project to IBM, Red Hat Ceph Storage will be OEM, starting with Red Hat Ceph Storage 6.

Figure 1-2 represents the different IBM Storage Ceph versions at the time of writing.

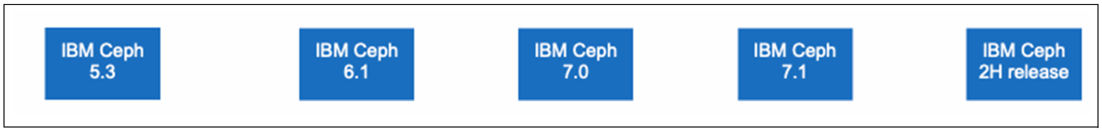


Figure 1-2 IBM Storage Ceph versions

1.2 Ceph and storage challenges

As storage has evolved over the decades, it has consistently faced a common set of challenges:

- Data keeps growing
- Technology changes
- Data organization, access, and costs
- Data added value

Data keeps growing

The evolution of IT has transformed data from character-based to multimedia, leading to an exponential increase in data volume and a decentralized data creation process across diverse terminals, effectively shattering the traditional notion of structured data.

Technology changes

The rapid advancement of the IT lifecycle has rendered the once-dominant, centralized data center obsolete, giving rise to a more decentralized architecture that hinges on unprecedented levels of application and server communication. This shift marks a stark departure from the era of centralized processing units that are housed within a single room, where passive terminals served as mere conduits for accessing computational power.

Data organization, access, and costs

Historically, data was centralized due to the limitations of early storage technologies like magnetic disks, tapes, and punch cards. Access to this data was often restricted and required physical handling of the storage medium. Preserving the availability and integrity of this diverse data over time was a crucial responsibility of the IT department.

Data added value

The dispersion of data across diverse infrastructure, users, and terminal types has made the generation of value from data increasingly dependent on the ability to process large volumes of data from multiple sources and in multiple formats, employing advanced techniques like Artificial Intelligence and machine learning. The key to unlocking value lies in enabling real-time data accessibility from multiple locations without the need for human intervention.

Ceph is no exception to these challenges, and was designed from the ground up to be highly available (HA), with no single point of failure, and highly scalable with limited day-to-day operational requirements other than the replacement of failed physical resources, such as nodes or drives. Ceph provides a complete, software-defined storage solution as an alternative to proprietary storage arrays.

1.2.1 Ceph approach

To address these challenges, Ceph relies on a layered architecture with an object store at its core. This core, which is known as the Reliable Autonomic Object Store (RADOS), provides the following features:

- Stores the data with data protection and consistency as goal number one.
- Follows a scale-out model to cope with growth and changes.
- Presents no single point of failure.
- Provides CRUSH as a customizable, metadata-less placement algorithm.

- Runs as a software-defined storage solution on commodity hardware.
- Open source to avoid vendor lock-in.

1.2.2 Ceph storage types

Ceph is known as a unified storage solution and supports the following types of storage:

- Block storage through a virtual block device solution that is known as RADOS Block Device (RBD).
- File storage through a POSIX compliant shared file system solution that is known as Ceph File System (CephFS).
- Object storage through an OpenStack Swift and Amazon S3 gateway that is known as the RADOS Gateway (RGW).

All the different types of storage are segregated and do not share data between them while eventually sharing the physical storage where they are physically stored. However, a custom CRUSH configuration enables you to separate the physical nodes and disks that are used by each of them.

All the different types of storage are identified and implemented as Ceph access methods on top of the native RADOS API. This API is known as `librados`.

Ceph is written entirely in C and C++ except for some API wrappers that are language-specific (For example, a Python wrapper for `librados` or `librbd`).

IBM Storage Ceph is positioned for the use cases that are shown in Figure 1-3. IBM is committed to support additional ones in upcoming versions, such as NVMe over Fabric for a full integration with VMware.

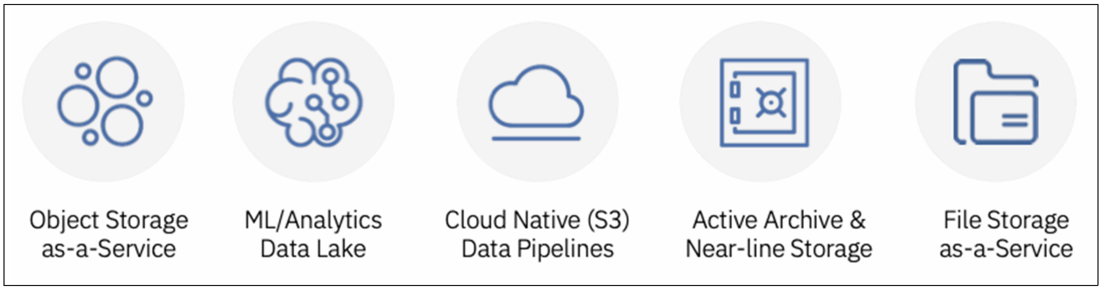

Object Storage as-a-Service
ML/Analytics Data Lake
Cloud Native (S3) Data Pipelines
Active Archive & Near-line Storage
File Storage as-a-Service

Figure 1-3 IBM Storage Ceph use cases

1.3 What is new with IBM Storage Ceph 7.0

IBM Storage Ceph 7.0 is a major release that introduces several new features and enhancements that can improve the performance, scalability, and security of your Ceph storage cluster. This section delves into some of the new features that are introduced with this upcoming version. These features are subject to change before the GA date.

1.3.1 Write Once, Read Many compliance certification

Cohasset Associates, Inc. evaluated IBM Storage Ceph Object Lock against electronic records requirements that are mandated by various regulatory bodies. They concluded that IBM Storage Ceph, when configured and used with Object Lock, adheres to the electronic record keeping system stipulations of SEC and Financial Industry Regulatory Authority (FINRA) rules, as listed below:

- SEC 17a-4(f)
- SEC 18a-6(e)
- FINRA 4511(c)
- CFTC 1.31(c)-(d)

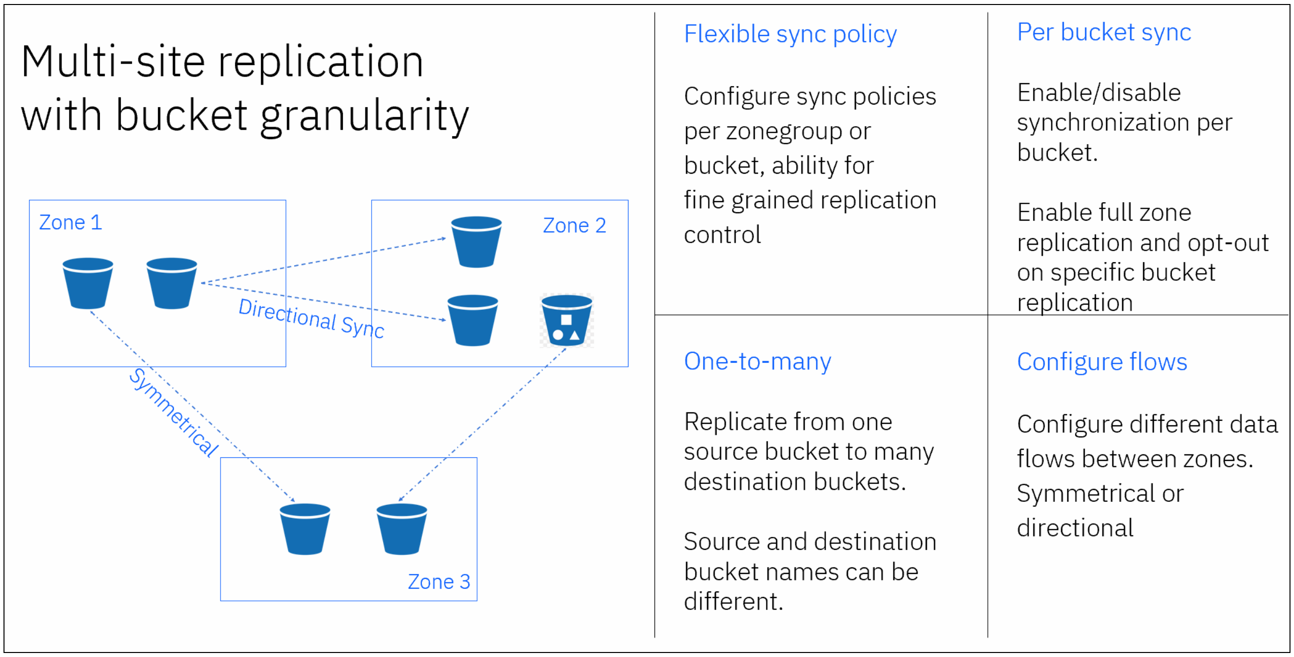
> **Note:** The US Securities and Exchange Commission (SEC) stipulates record keeping requirements, including retention periods. FINRA rules regulate member brokerage firms and exchange member markets.

For more information, see *IBM Storage Ceph Compliance Assessment*.

1.3.2 Multi-site replication with bucket granularity

This feature enables the replication of individual buckets or groups of buckets to a separate IBM Storage Ceph cluster. This function is analogous to RGW multi-site, but with the added capability of replicating at the bucket level.

Figure 1-4 shows the multi-site replication with bucket granularity feature.



Multi-site replication with bucket granularity

Flexible sync policy

Configure sync policies per zonegroup or bucket, ability for fine grained replication control

Per bucket sync

Enable/disable synchronization per bucket.

Enable full zone replication and opt-out on specific bucket replication

One-to-many

Replicate from one source bucket to many destination buckets.

Source and destination bucket names can be different.

Configure flows

Configure different data flows between zones. Symmetrical or directional

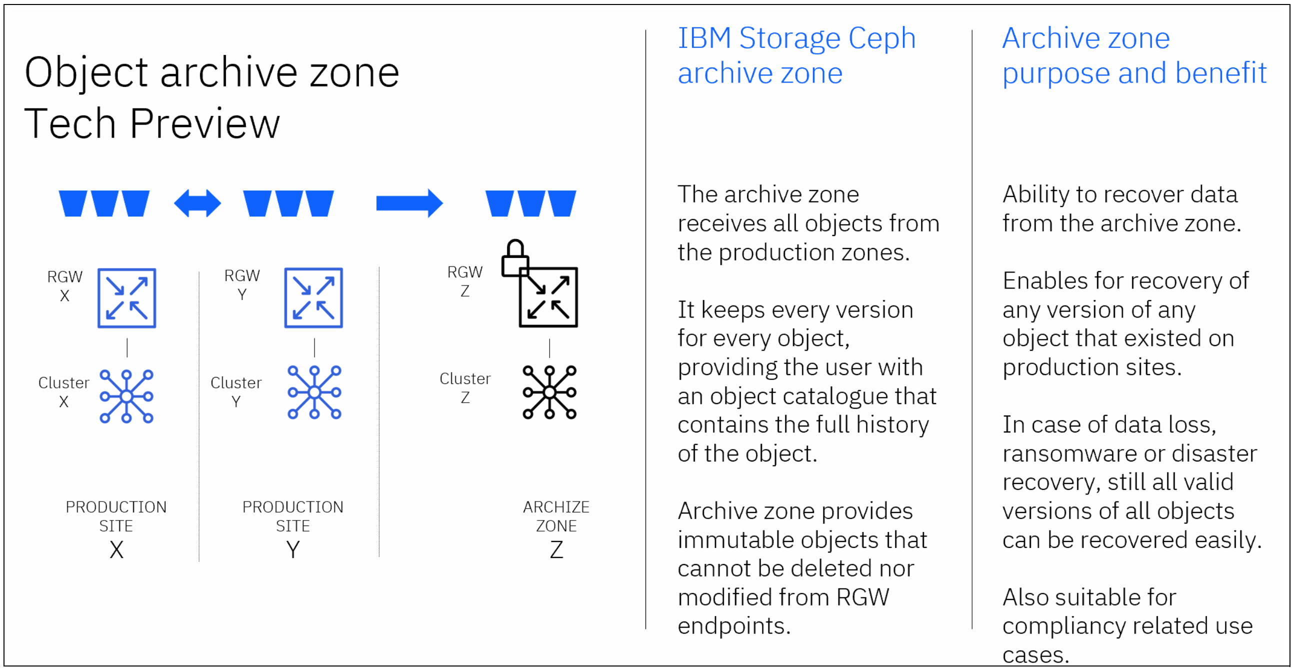
Figure 1-4 The multi-site replication with bucket granularity feature

Previously, replication was limited to full zone replication. This new feature grants clients enhanced flexibility by enabling the replication of individual buckets with or against different IBM Storage Ceph clusters. This granular approach enables selective replication, which can be beneficial for edge computing, colocations, or branch offices. Bidirectional replication is also supported.

1.3.3 Object archive zone (technology preview)

The archive zone serves as a repository for all object versions from the production zones. It maintains a complete history of each object, providing users with a comprehensive object catalog. The data that is stored in the archive zone is rendered immutable through the implementation of Write Once, Read Many (WORM) with Ceph Object Lock. This immutable safeguard renders data in the archive zone impervious to modifications or deletions that might be caused by ransomware, computer viruses, or other external threats.

Figure 1-5 on page 9 shows the object archive zone feature.



Object archive zone
Tech Preview

IBM Storage Ceph archive zone

The archive zone receives all objects from the production zones.

It keeps every version for every object, providing the user with an object catalogue that contains the full history of the object.

Archive zone provides immutable objects that cannot be deleted nor modified from RGW endpoints.

Archive zone purpose and benefit

Ability to recover data from the archive zone.

Enables for recovery of any version of any object that existed on production sites.

In case of data loss, ransomware or disaster recovery, still all valid versions of all objects can be recovered easily.

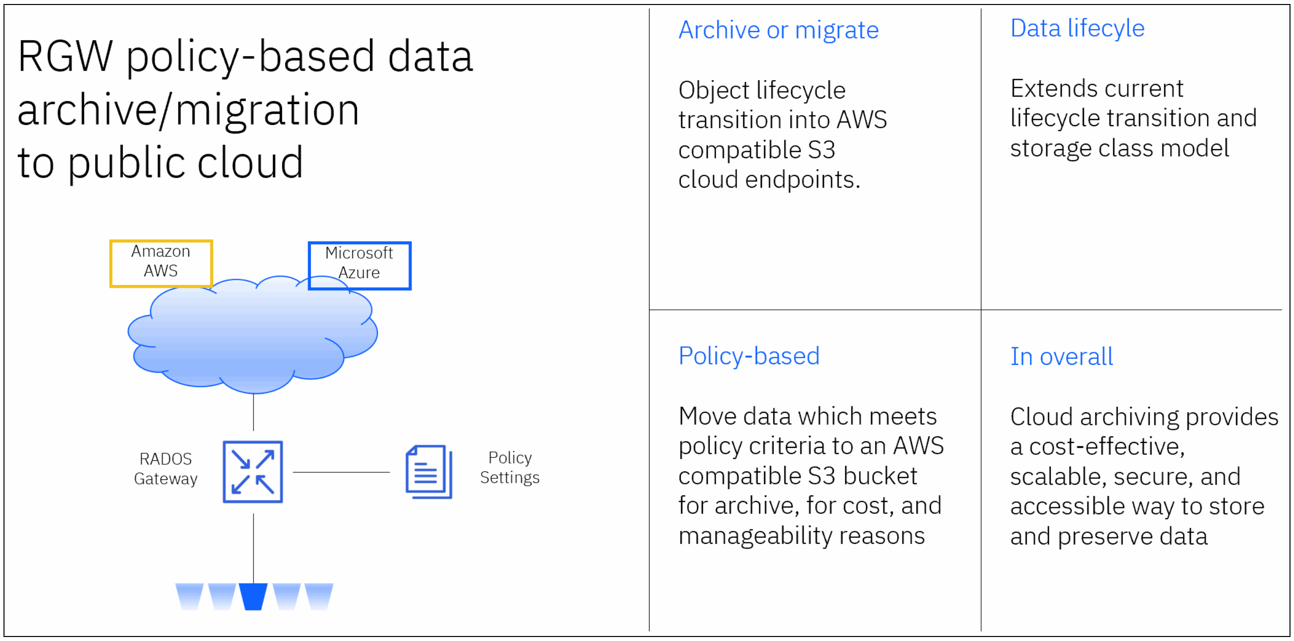
Also suitable for compliancy related use cases.

Figure 1-5 Object archive zone

The archive zone selectively replicates data from designated buckets within the production zones. System administrators can control which buckets undergo replication to the archive zone, enabling them to optimize storage capacity usage and prevent the accumulation of irrelevant content.

1.3.4 RGW policy-based data archive and migration capability

Figure 1-6 highlights the policy-driven data archive and migration capability.



RGW policy-based data
archive/migration
to public cloud

Archive or migrate

Object lifecycle transition into AWS compatible S3 cloud endpoints.

Data lifecyle

Extends current lifecycle transition and storage class model

Policy-based

Move data which meets policy criteria to an AWS compatible S3 bucket for archive, for cost, and manageability reasons

In overall

Cloud archiving provides a cost-effective, scalable, secure, and accessible way to store and preserve data

Figure 1-6 RGW policy-based data archive and migration capability

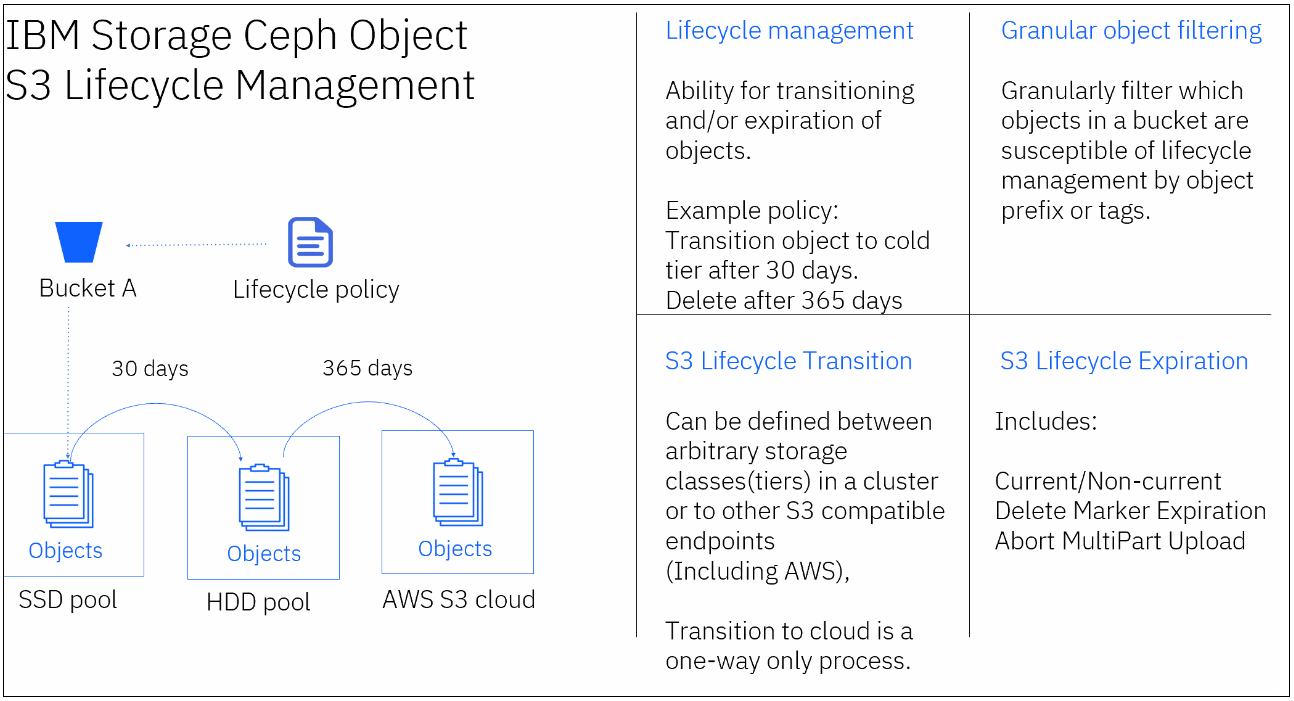
This feature enables clients to seamlessly migrate data that adheres to policy criteria to the public cloud archive. This function is available for both Amazon Web Services (AWS) and Microsoft Azure public cloud users.

The primary benefit of this feature is its ability to liberate on-premises storage space that is occupied by inactive, rarely accessed data. This reclaimed storage can be repurposed for active datasets, enhancing overall storage efficiency.

1.3.5 IBM Storage Ceph Object S3 Lifecycle Management

IBM Storage Ceph Lifecycle Management provides clients with the flexibility to combine object transition and expiration within a single policy. For example, you can configure a policy to migrate objects to the cold tier after 30 days and then delete them after 365 days.

Figure 1-7 shows the IBM Storage Ceph Object S3 Lifecycle Management feature.



IBM Storage Ceph Object S3 Lifecycle Management

Lifecycle management

Ability for transitioning and/or expiration of objects.

Example policy:
Transition object to cold tier after 30 days.
Delete after 365 days

Granular object filtering

Granularly filter which objects in a bucket are susceptible of lifecycle management by object prefix or tags.

S3 Lifecycle Transition

Can be defined between arbitrary storage classes(tiers) in a cluster or to other S3 compatible endpoints (Including AWS),

Transition to cloud is a one-way only process.

S3 Lifecycle Expiration

Includes:

Current/Non-current
Delete Marker Expiration
Abort MultiPart Upload

Figure 1-7 IBM Storage Ceph Object S3 Lifecycle Management

1.3.6 Dashboard UI enhancements

With these new UI functions, a Ceph cluster administrator can manage the whole lifecycle of CephFS file systems, volumes, and subvolumes by using the graphical dashboard UI:

- CephFS UI interaction

  Creation, listing, changing options, and deletion of CephFS file systems, volumes, subvolume groups, and subvolumes.

- CephFS access and encryption:
  - Access management for CephFS resources.
  - Encryption management options for CephFS.

- CephFS snapshot management:
  - List all snapshots for a file system, volume subvolume, or directory.
  - Create or delete a one-time snapshot.
  - Display occupied capacity by a snapshot.
- CephFS monitoring

  Shows the health status, basic read/write throughput and capacity usage for file systems, volumes and subvolumes. including historical graphs.
- RGW multi-site UI configuration:
  - RGW multi-site setup and configuration from the dashboard UI.
  - Simple setup without manual setup steps.
- RGW bucket level view and management:
  - ACL status (private and public).
  - Bucket tags (add or remove tags).
  - On a per-bucket basis.
- Labeled performance counters per user or bucket:
  - Reporting into the Prometheus monitoring stack.
  - Bucket operations and statistics.
  - User operations and statistics.
- Multisite synchronization status:
  - Synchronization status.
  - RGW operation metrics.
  - Client and replica traffic.
  - Dashboard visibility of an RGW's latest sync status.

1.3.7 NFS support for CephFS for non-native Ceph clients

After configuring the CephFS, users can easily manage NFS exports on the Ceph dashboard, creating, editing, and deleting them as needed.

IBM Storage Ceph Linux clients can seamlessly mount CephFS without additional driver installations because CephFS is embedded in the Linux kernel by default. This capability extends CephFS accessibility to non-Linux clients through the NFS protocol. In IBM Storage Ceph 7, the NFS Ganesha service expands compatibility by supporting NFS v4, empowering a broader spectrum of clients to seamlessly access CephFS resources.

1.3.8 NVMe over Fabrics (technology preview)

IBM Storage Ceph leverages NVMe over Fabrics (NVMe-oF) to deliver block storage access to non-Linux clients. Although Linux clients can directly connect to CephFS by using the RBD protocol due to its native integration with the Linux kernel, non-Linux clients require an extra gateway.

The newly introduced IBM Storage Ceph NVMe-oF gateway bridges the gap for non-Linux clients, enabling them to seamlessly interact with NVMe-oF initiators. These initiators establish connections with the gateway, which connects to the RADOS block storage system.

The performance of NVMe-oF block storage through the gateway is comparable to native RBD block storage, ensuring a consistent and efficient data access experience for both Linux and non-Linux clients.

1.3.9 Object storage for machine learning and analytics: S3 Select

S3 Select is a feature of Amazon S3 that enables you to filter the contents of an S3 object without having to download the entire object. IBM Storage Ceph 7.0 adds support for S3 Select, which can reduce the amount of data that needs to be transferred for certain workloads.

This feature empowers clients to employ straightforward SQL statements to filter the contents of S3 objects and retrieve only the specific data that they require. By leveraging S3 Select for data filtering, clients can minimize the amount of data that is transferred by S3, which reduces both retrieval costs and latency. The following data formats are supported:

- Comma-separated value (CSV)
- JSON
- Parquet

1.3.10 RGW multi-site performance improvements

IBM Storage Ceph 7 boosts performance for data replication and metadata operations by enabling parallel processing of operations as the number of RGW daemons increases. This capability enables horizontal scalability, empowering you to efficiently handle growing workloads.

1.3.11 Erasure code EC2+2 with four nodes

Erasure code is a data protection technique that uses multiple copies of data to protect against data loss. Erasure code EC2+2 is a new erasure code that provides better data protection and reduced storage costs than previous erasure codes.

With IBM Storage Ceph 7, erasure code C2+2 can be used with just four nodes, making it more efficient and cost-effective to deploy erasure coding (EC) for data protection.

Previously, a larger cluster of nodes was required to establish a supported configuration. However, this latest release reduces the minimum configuration requirement to four nodes, enabling users to leverage the benefits of erasure code C2+2 with a smaller and more cost-effective setup.

This enhancement is beneficial for smaller deployments or organizations seeking to optimize their storage infrastructure costs. By reducing the minimum node requirement, IBM Storage Ceph 7 makes erasure code C2+2 more accessible to a wider range of users and use cases.

IBM Storage Ready Nodes can be deployed with a minimum of four nodes and use EC for the RADOS back end in this basic configuration. This scalable solution can be expanded to accommodate up to 400 nodes.

2

IBM Storage Ceph architecture

This chapter introduces the architecture of IBM Storage Ceph. It details the various components of a Ceph cluster, such as the components that are used to build the Reliable Autonomic Object Store (RADOS) cluster that is built around the Controlled Replication Under Scalable Hashing (CRUSH) concept.

This chapter has the following sections:

- Architecture
- Access methods
- Deployment

2.1 Architecture

Figure 2-1 represents the structure and layout of a Ceph cluster, starting from RADOS at the bottom, up to the various access methods that are provided by the cluster.

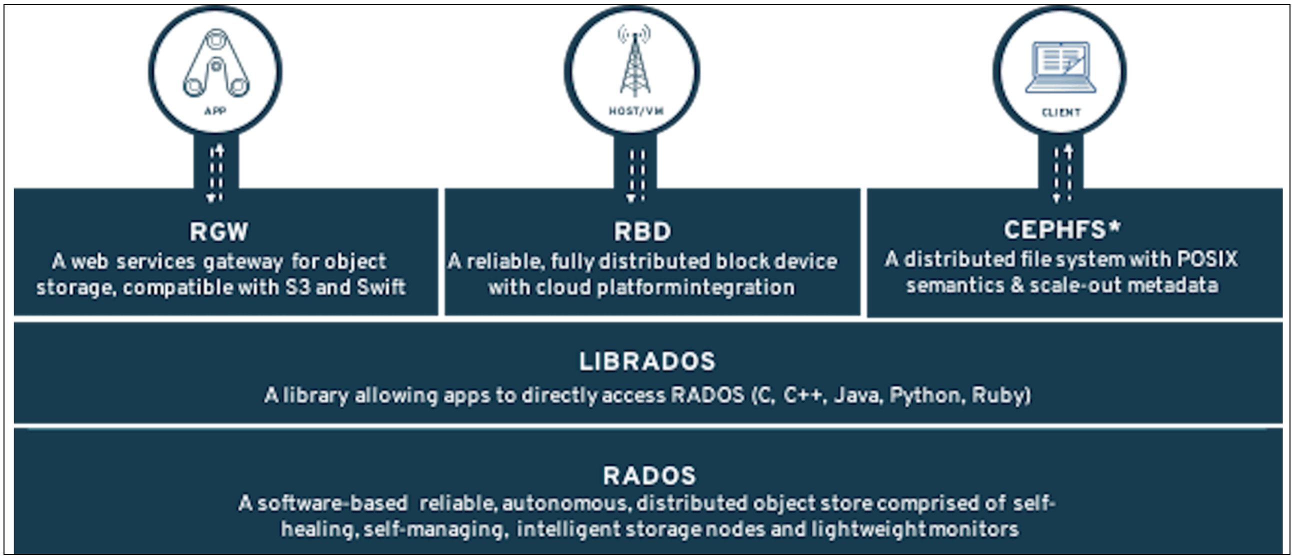

APP
HOST/VM
CLIENT
RGW
A web services gateway for object storage, compatible with S3 and Swift
RBD
A reliable, fully distributed block device with cloud platformintegration
CEPHFS*
A distributed file system with POSIX semantics & scale-out metadata
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby)
RADOS
A software-based reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors

Figure 2-1 Ceph general architecture

2.1.1 RADOS components

RADOS is built by using the following components:

- Monitors (MONs)
- Object storage daemons (OSDs)
- Managers (MGRs)
- Metadata Servers (MDSs)

Monitors

MONs are responsible for managing the state of the cluster. Like with any distributed storage system, the challenge is tracking the status of each cluster component (MONs, MGRs, OSDs, and others).

Ceph maintains its cluster state through a set of specialized maps, collectively referred to as the *cluster map*. Each map is assigned a unique version number (*epoch*), which starts at 1 and increments by 1 on every state change for the corresponding set of components.

The MONs maintain the following maps:

- MON map
- MGR map
- OSD map
- MDS map
- Placement group (PG) map
- CRUSH map

To ensure the integrity and consistency of map updates, the MONs employ the PAXOS algorithm, enabling them to reach a consensus among multiple MONs before validating and implementing any map changes.

To prevent split-brain scenarios, the number of MONs that are deployed in a Ceph cluster must always be an odd number greater than two to ensure that most MONs can validate map updates. More than half of the MONs that are present in the Monitor Map (MONMap) must agree on the change that is proposed by the PAXOS quorum leader for the map to be updated.

Note: The MONs are not part of the data path, meaning that they do not directly handle data storage or retrieval requests. They exist primarily to maintain cluster metadata and keep all components synchronized.

Managers

The MGRs are integrated with the MONs, and collect the statistics within the cluster. The MGRs provide a pluggable Python framework to extend the capabilities of the cluster. As such, the developer or the user can leverage or create MGR modules that are loaded into the MGR framework.

The following list provides some of the existing MGR modules that are available:

- Balancer module (dynamically reassign PGs to OSDs for better data distribution).
- Auto-scaler module (dynamically adjust the number of PGs that are assigned to a pool).
- Dashboard module (provide a UI to monitor and manage the Ceph cluster).
- RESTful module (provide a RESTful API for cluster management).
- Prometheus module (provide metrics support for the Ceph cluster).

Object storage daemons

Sometimes referred to as object storage daemons but always as OSDs, this component of the RADOS cluster is responsible for storing the data in RADOS and for serving the I/O requests that originate from the Ceph cluster clients.

OSDs are responsible for the following tasks:

- Serve I/O requests.
- Protect the data (replication or erasure coding (EC) model).
- Recover the data after failure.
- Rebalance the data on cluster expansion or reduction.
- Check the consistency of the data (scrubbing).
- Check for bit rot detection (deep scrubbing).

Each OSD can be assigned one role for a PG:

- Primary OSD
- Secondary OSD

The primary OSD performs all these functions, and a secondary OSD always acts under the control of a primary OSD. For example, if a write operation lands on the primary OSD for a PG, the primary OSD sends a copy of the I/O to one or more secondary OSDs, and the secondary OSD is solely responsible for writing the data onto the physical media and, when down, send acknowledgment to the primary OSD.

Note: A cluster node where OSDs are deployed is called an OSD node.

Usually, you deploy one OSD per physical drive.

OSD Object Store

This section talks about the two object store formats:

- FileStore
- BlueStore

FileStore

In its early days, the Ceph OSD used an object store implementation that was known as FileStore (Figure 2-2). This object store solution leverages an XFS formatted partition to store the actual data and a raw partition to store the object store journal. The journal is written as a wrap-around raw device that is written sequentially.

Here are the roles of the journal:

- Guarantees transaction atomicity across multiple OSDs.
- Leverages the sequential performance of hard disk drives (HDDs).

When flash-based drives arrived on the market, it became a best practice to use a solid-state drive (SSD) to host the journal to enhance the performance of write operations in the Ceph cluster.

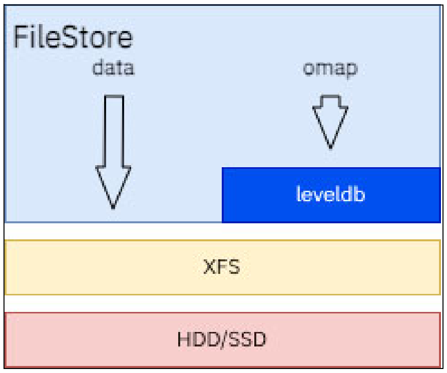


Figure 2-2 FileStore architecture

However, the complexity of the solution and the write amplification due to two writes for each write operation led the Ceph project to consider an improved solution for the future.

BlueStore

BlueStore is the new default OSD object store format since the launch of upstream Luminous (Red Hat Ceph Storage 3.x). With BlueStore, data is written directly to the disk device, and a separate RocksDB key-value store contains all the metadata.

When the data is written to the raw data block device, the RocksDB is updated with the metadata that is related to the new data blobs that were written.

RocksDB is a high-performance, key-value store that was developed in 2012 at Facebook that performs well with flash-based devices. Because RocksDB cannot write directly to a raw block device, a virtual file system (VFS) layer was developed for RocksDB to store its `.sst` files on the raw block device. The VFS layer is named BlueFS.

RocksDB uses a DB portion and a write-ahead log (WAL) portion. Depending on the size of the I/O, RocksDB writes the data directly to the raw block device through BlueFS or to the WAL so that the data can be later committed to the raw block device. The latter process is known as a *deferred write*.

As BlueStore is introduced, more functions are added to the Ceph cluster:

- Data compression at the OSD level, independent of the access method that is used.
- Checksums are verified on each read request.

Figure 2-3 shows the BlueStore architecture.

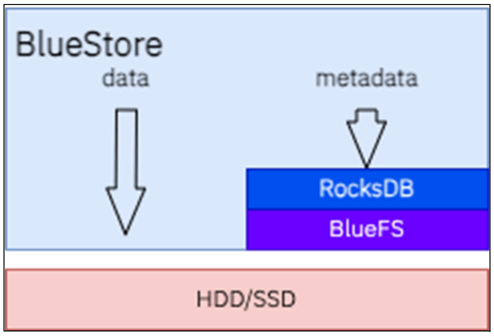


Figure 2-3 BlueStore architecture

BlueStore provides the following improvements over FileStore:

- Removal of the extra file system layer for direct access to the raw device to store data.
- RocksDB performance to store and retrieve metadata for faster physical object location.
- Full data and metadata checksum.
- Snapshot and clones benefit a more efficient copy-on-write (COW) on the raw block device.
- Minimizes write amplification for many workloads.
- Deferred write threshold configurable per device type.

BlueStore operates with the following layout:

- One device for the data (also known as *block*).
- One optional RocksDB device for the metadata (*block.db*).
- One optional RocksDB device for the WAL (*block.wal*).

> **Note:** A best practice is to use a device faster than the data device for the RocksDB metadata device and a device faster than the RocksDB metadata device for the WAL device.

When a separate device is configured for the metadata, it might overflow to the data device if the metadata device becomes full. Although this situation is not a problem if both devices are of the same type, it leads to performance degradation if the data device is slower than the metadata device. This situation is known as *BlueStore spillover*.

When deploying BlueStore OSDs, follow the following best practices:

- If you use the same device type, for example, all rotational drives, specify the block device only and do not separate `block.db` or `block.wal`.
- If you use a mix of fast and slow devices (SSD / NVMe and rotational), choose a fast device for `block.db` while `block` uses the slower rotational drive.

BlueStore uses a cache mechanism that dynamically adjusts (`bluestore_cache_autotune = 1`). When cache autotuning is enabled, the OSD tries to keep the OSD heap memory usage below the value that is assigned to the **`osd_memory_target`** parameter and not under the **`osd_memory_cache_min`** value.

The cache sizing can be adjusted manually by setting the parameter **`bluestore_cache_autotune`** to `0`, and the following parameters can be adjusted to allocate specific portions of the BlueStore cache:

- **`cache_meta`**: BlueStore node and associated data
- **`cache_kv`**: RocksDB block cache, including indexes and filters
- **`data_cache`**: BlueStore cache for data buffers

These parameters are expressed as a percentage of the cache size that is assigned to the OSD.

With BlueStore, you can configure more features to align best with your workload:

- `block.db` sharding
- Minimum allocation size on the data device

2.1.2 Ceph cluster partitioning and data distribution

This section describes Ceph cluster partitioning and data distribution.

Pools

The cluster is divided into logical storage partitions that are called *pools*. The pools have the following characteristics:

- Group data of a specific type.
- Group data that is to be protected by using the same mechanism (replication or EC).
- Group data to control access from Ceph clients.
- Only one CRUSH rule that is assigned to determine PG mapping to OSDs.

Note: At the time of writing, pools support compression, but do *not* support deduplication. The compression can be activated on a per pool basis.

Data protection

The data protection scheme is assigned individually to each pool. The data protection that IBM Storage Ceph supports are as follows:

- Replicated, which makes a full copy of each byte that is stored in the pool (the default is three copies):
  - Two replicas or more are supported by underlying flash devices.
  - Three replicas or more are supported by HDDs.

- EC, which functions in a similar way as the parity RAID mechanism:
  - 4+2 EC is supported by the `jerasure` plug-in.
  - 8+3 EC is supported by the `jerasure` plug-in.
  - 8+4 EC is supported by the `jerasure` plug-in.

Note: EC, although supported for all types of storage (block, file, and object), is not recommended for block and file because it delivers lower performance.

The difference between replicated and EC pools in data protection is summarized in Figure 2-4.

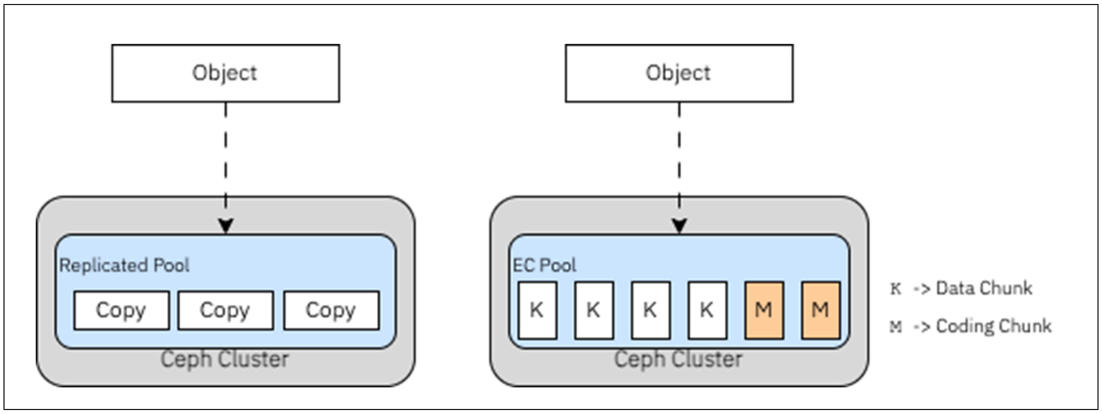


Figure 2-4 Replicated data protection versus erasure coding data protection

EC provides a more cost-efficient data protection mechanism and greater resiliency and durability as you increase the number of coding chunks, which enables the pool to survive the loss of many OSDs or servers before the data becomes unrecoverable, but offers lower performance because of the computation and network traffic that is required to split the data and calculate the coding chunks.

Here are the benefits of the replicated model:

- High durability with three copies
- Quicker recovery
- Performance optimized

Here are the benefits of the EC model:

- Cost-effective durability with the multiple coding chunks
- More expensive recovery
- Capacity optimized

Each pool is assigned a set of parameters that can be changed dynamically. Table 2-1 lists the main parameters that are used for pools, and details whether the parameter can be dynamically modified.

Table 2-1 Main parameters for pools and details whether the parameter can be dynamically modified

Name	Function	Dynamic update
`id`	Unique ID for the pool.	No
`pg_num`	Total number of PGs.	Yes
`pgp_num`	Effective number of PGs.	Yes
`size`	Number of replicas or chunks for the pool.	Yes (replicated) No (EC)
`min_size`	Minimum number of replicas for a PG to be active.	Yes
`crush_rule`	CRUSH rule to use for the PG placement.	Yes
`nodelete`	Prevents the pool from being deleted.	Yes
`nopgchange`	Prevents the number of PGs to be modified.	Yes
`nosizechange`	Prevents `size` and `min_size` to be modified.	Yes
`noscrub`	Prevents scrubbing for the PGs of the pool.	Yes
`nodeep-scrub`	Prevents deep-scrubbing for the PGs of this pool.	Yes
`pg_autoscale_mode`	Enables or disables PG autoscaling for this pool.	Yes
`target_size_ratio`	Expected capacity for the pool that is used by PG auto-scaler.	Yes

Placement groups

The pool is divided into hash buckets that are called placement groups. The PG does the following tasks:

- Stores the objects as calculated by the CRUSH algorithm.
- Ensures that the storage of the data is abstracted from the physical devices.

As such, each PG is assigned to a set of OSDs with the following rules:

- A RADOS object is assigned to only one PG.
- A PG is assigned to multiple objects.
- Each PG may occupy a different capacity on a disk.
- Each PG exists on multiple OSDs (assigned by CRUSH).
- Ensures that the storage of the data is abstracted from the physical devices.

Note: The mapping of a PG is always the same for a cluster state.

Placement group states

Each PG has a state that represents the status of the PG and helps you diagnose whether the PG is healthy or can serve I/O requests. Table 2-2 lists the PG states.

Table 2-2 Placement group states

State	Description
creating	The PG is being created, and it is not ready for use.
active+clean	The PG is functional, and all data is replicated and available.
undersized	The PG is functional, but has fewer copies than configured.
active+stale	The PG is functional, but one or more replicas are stale and must be updated.
stale	The PG is in an unknown state because the MONs did not receive an update after the PG placement changed.
down	A replica with necessary data is down, so the PG is offline.
peering	The PG is adjusting its placement and replication settings.
activating	The PG is peered but not yet active.
recovering	The PG is migrating or synchronizing objects and their replicas.
recovery_wait	The PG is waiting to start recovery.
recovery_toofull	The PG is waiting for recovery because the target OSD is full.
backfilling	The PG is adding new replicas to its data.
backfill_wait	The PG is waiting to start backfilling.
backfill_toofull	The PG is waiting for backfill because the target OSD is full.
incomplete	The PG is missing information about some writes that might have occurred or do not have healthy copies.
scrubbing	The PG is checking group metadata inconsistencies.
deep	The PG is checking the stored checksum.
degraded	The PG has not replicated some objects to the correct number of OSDs.
repair	The PG is being checked and repaired for any inconsistencies that it finds.
inconsistent	The PG has inconsistencies between its different copies that are stored on different OSDs.
snaptrim	The PG is trimming snapshot data.
snaptrim_wait	The PG is waiting for snap trimming to start.
snaptrim_error	The PG encountered errors during the snap trimming process.

Cluster status

Although PGs have their distinct states, the Ceph cluster has its own global status that can be checked with the **`ceph status`**, **`ceph health`**, or **`ceph health detail`** commands.

Table 2-3 Cluster status

Cluster status	Description
`HEALTH_OK`	The cluster is fully operational, and all components are operating as expected.
`HEALTH_WARN`	Some issues exist in the cluster, but the data is available to clients.
`HEALTH_ERR`	Serious issues exist in the cluster, and some data is unavailable to clients.
`HEALTH_FAILED`	The cluster is not operational, and data integrity might be at risk.
`HEALTH_UNKNOWN`	The status of the cluster is unknown.

Figure 2-5 illustrates the general component layout within a RADOS cluster.

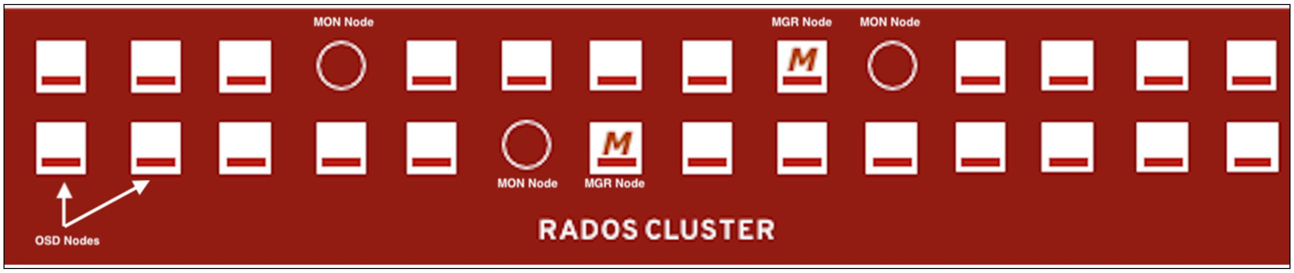


Figure 2-5 Ceph cluster layout

Note: With the latest version of IBM Storage Ceph, multiple components can be deployed on one node by referencing the support matrix. Log in with your IBMid to access this link.

2.1.3 Controlled Replication Under Scalable Hashing

To find the data in the cluster, Ceph implements a unique algorithm that is known as *CRUSH*, which enables the clients and the members of a Ceph cluster to find the data through a pseudo-random method that provides the same result for a cluster state.

Because the placement remains the same for a cluster state, imagine the following scenario:

- A RADOS object is hosted in PG 3.23.
- A copy of the object is written on OSDs 24, 3, and 12 (state A of the cluster).
- OSD 24 is stopped (state B of the cluster).
- An hour later, OSD 24 is restarted.

When OSD 24 stops, the PG that contains the RADOS object is re-created to another OSD so that the cluster satisfies the number of copies that must be maintained for the PGs that belong to the specific pool.

Now, state B of the cluster becomes different from the original state A of the cluster. PG 3.23 is now protected by OSD 3, 12, and 20.

When OSD 24 restarts, assuming that no changes were made to the cluster in the meantime (such as cluster expansion, reduction, or CRUSH customization or another OSD failing), the copy of the PG 3.23 resynchronizes on OSD 24 to provide the same mapping for the PG because the cluster state is identical to state A.

From an architectural point of view, CRUSH provides the mechanism that is shown in Figure 2-6.

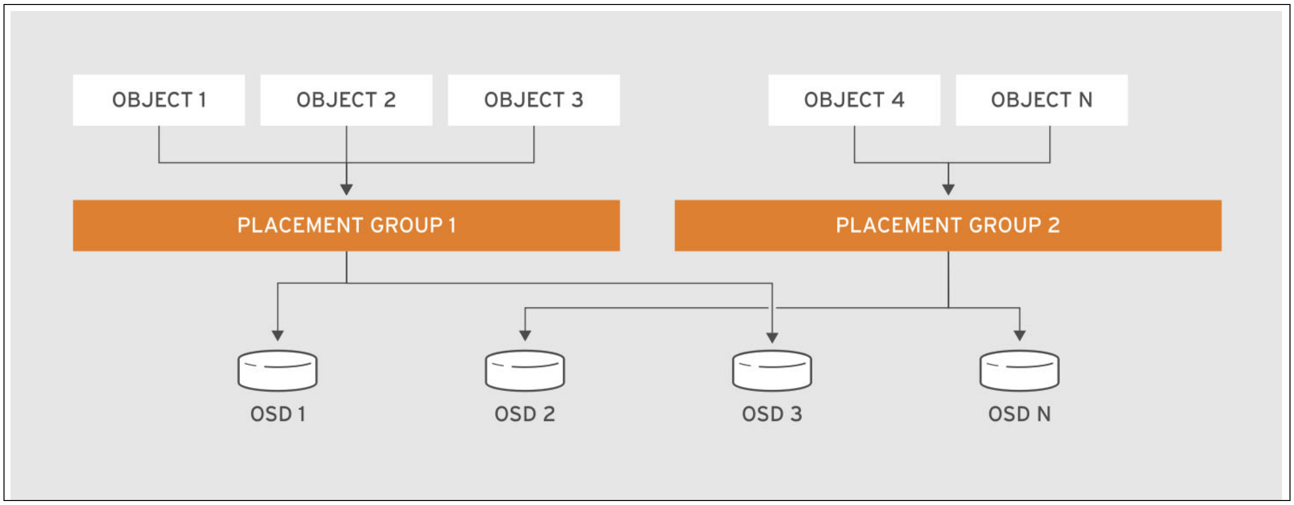


Figure 2-6 CRUSH from object to OSD architecture

To determine the location of a specific object, the mechanism that is shown in Figure 2-7 is used.

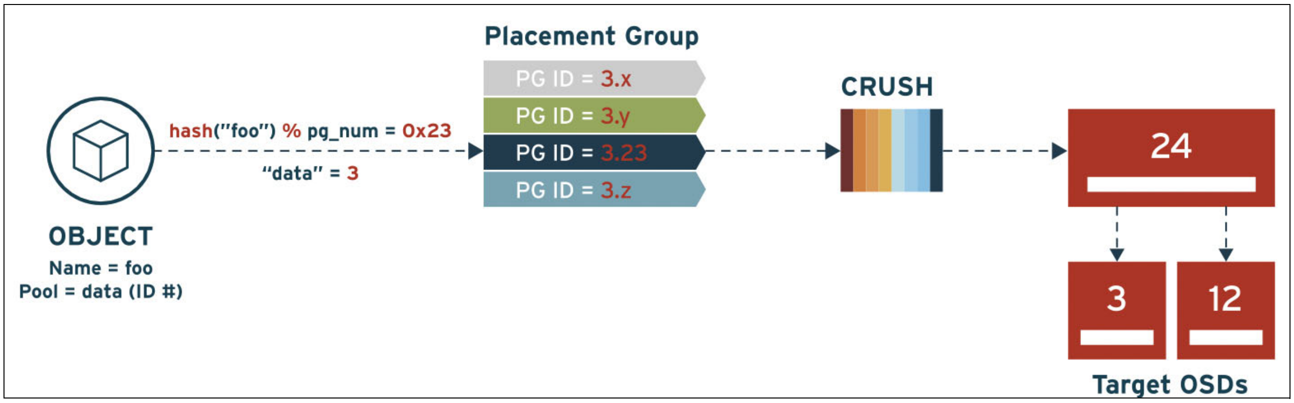


Figure 2-7 CRUSH calculation detail

To make sure that a client or a Ceph cluster component finds the correct location of an object that enables the client-to-cluster communication model, all maps that are maintained by the MONs have versions, and the version of the map that is used to find an object or a PG is checked by the recipient of a request.

The specific exchange determines whether the MONs or one of the OSDs updates the map version that is used by the sender. Usually, these updates are differentials, meaning that only the changes to the map are transmitted after the initial connection to the cluster.

If you look at the whole Ceph cluster with its many pools, where each pool has its own PGs, the cluster looks like Figure 2-8.

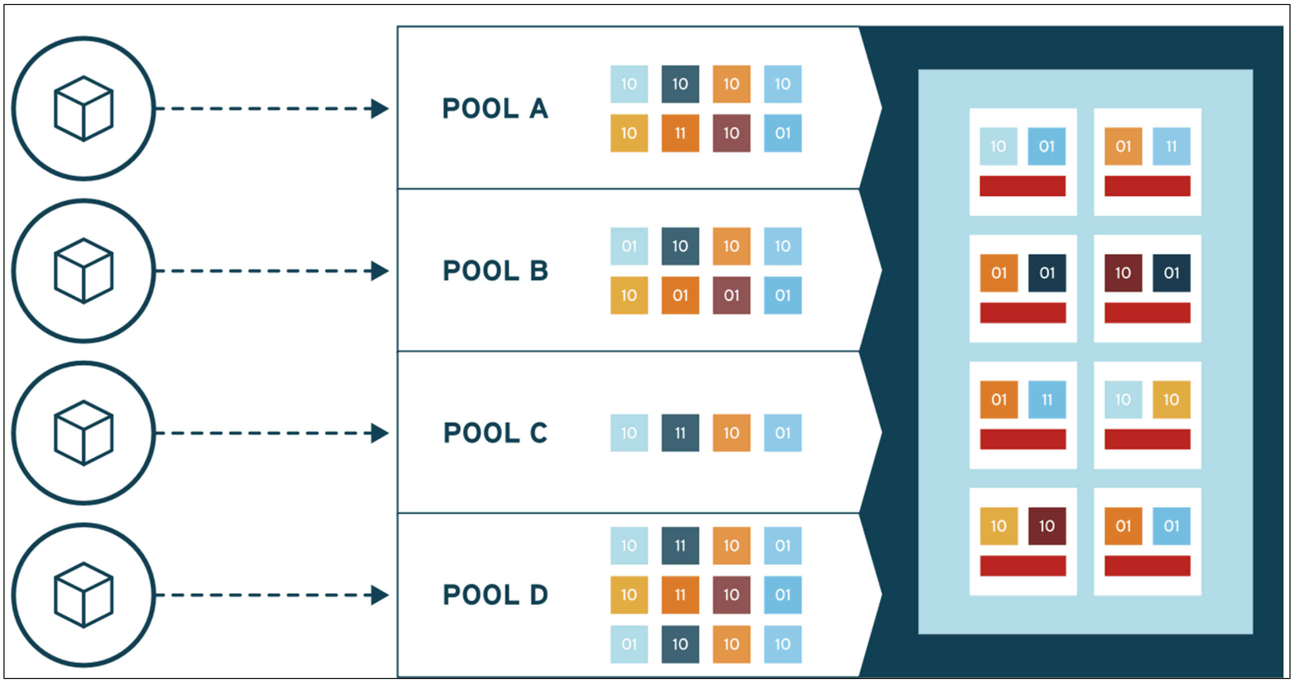


Figure 2-8 Full data placement picture (objects on the left and OSDs on the right)

2.1.4 OSD failure and recovery

OSDs operate with two separate sets of statuses:

- Up / Down: Tracks whether the OSD is running and responsive.
- In / Out: Tracks whether the OSD participates in data placement.

In the Ceph clusters, the following mechanisms exist to track the status of the different components of the cluster:

- MON-to-MON heartbeat
- OSD-to-MON heartbeat
- OSD-to-OSD heartbeat

The OSD-to-OSD heartbeat mechanism enables each OSD to operate independently, ensuring data availability even if communication with other OSDs is disrupted. Also, the OSD-to-MON heartbeat ensures that the OSDMap remains up-to-date even if all OSDs become unavailable.

On detecting an unavailable peer OSD (because it works with other OSDs to protect PGs), an OSD relays this information to the MONs, which enables the MONs to update the OSDMap to reflect the status of the unavailable OSD.

OSD failures

When an OSD becomes unavailable, the following statements become true:

- The total capacity of the cluster is reduced.
- The total throughput that can be delivered by the cluster is reduced.
- The cluster enters a recovery process that generates disk I/Os (to read the data that must be recovered) and network traffic (to send a copy of the data to another OSD and re-create the missing data).

When an OSD fails or becomes unresponsive, it is marked as down. If after a configurable amount of time (set by the **`mon_osd_down_out_interval`** parameter (the default is 600 seconds) the OSD does not return to an up state, it automatically is marked as out of the cluster, and the cluster starts recovering the data for all the PGs that the failed OSD was hosting.

OSD recovery and backfill

Although the recovery and backfill processes can be customized by Ceph administrators, the default parameters are selected to minimize performance impact on the client traffic. Modifying these parameters requires expertise and a thorough understanding of the potential performance implications.

Recovery is the process of moving or synchronizing a PG after the failure of an OSD.

Backfill is the process of moving a PG after the addition or the removal of an OSD to or from the Ceph cluster.

Client impact on failures

The cluster provides different maps so that every client or component of the Ceph cluster can be aware of the status of every component within the Ceph cluster.

When a Ceph client connects to the Ceph cluster, it must contact the MONs of the cluster so that the client can be authenticated. Once authenticated, the Ceph client is provided with a copy of the different maps that are maintained by the MONs.

In Figure 2-9, the different steps that occur when a Ceph client connects to the cluster and then accesses data is this high-level sequence:

1. On successful authentication, the client is provided with the cluster map.
2. The data placement for object name xxxxx is calculated.
3. The Ceph client initiates a connection with the primary OSD that protects the PG.

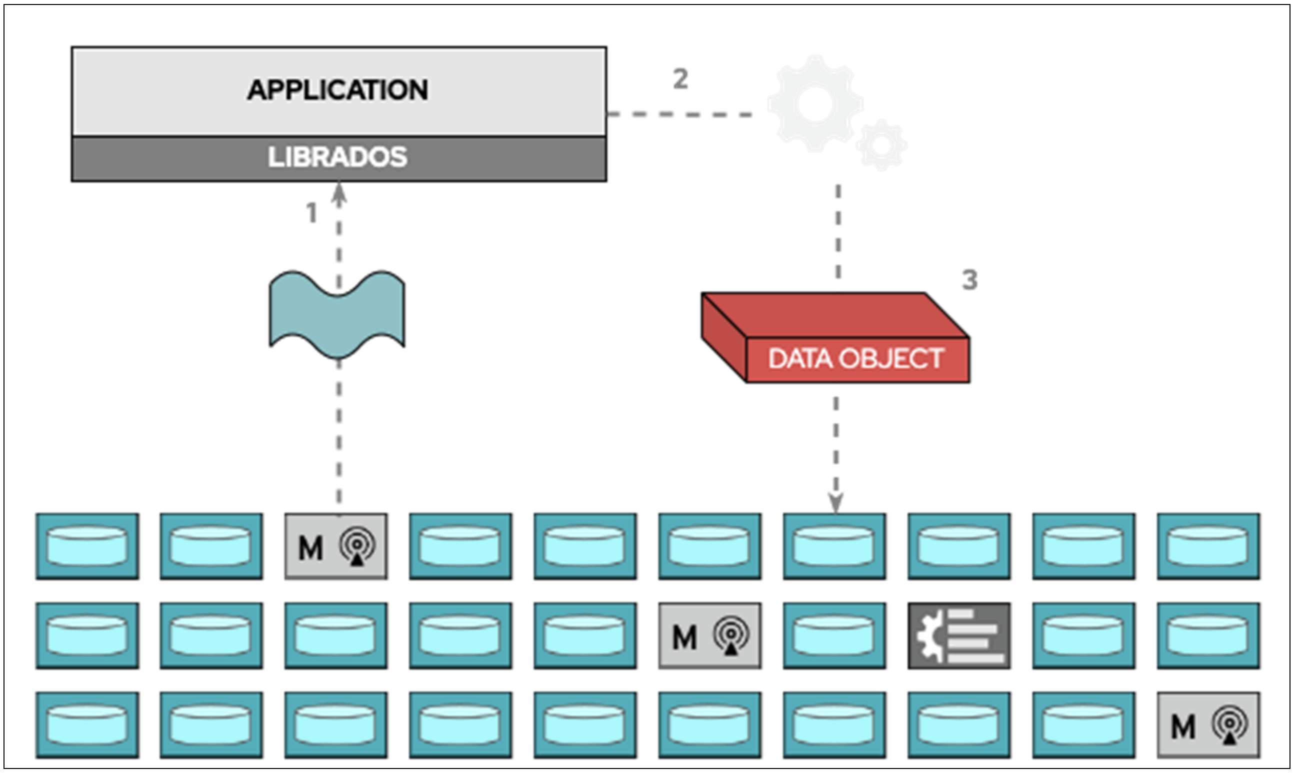


Figure 2-9 Client cluster interaction

When a failure occurs within the cluster as the Ceph client tries to access a specific OSD, the following cases can occur:

- The OSD that it is trying to contact is unavailable.
- The OSD that it is trying to contact is available.

In the first case, the Ceph client falls back to the MONs to obtain an updated copy of the cluster map.

In Figure 2-10 on page 27, the different steps that occur when a Ceph client connects to the cluster and then accesses data is this high-level sequence:

4. As the target OSD has become unavailable, the client contacts the MONs to obtain the latest version of the cluster map.
5. The data placement for object name xxxxx is recalculated.
6. The Ceph client initiates a connection with the new primary OSD that protects the PG.

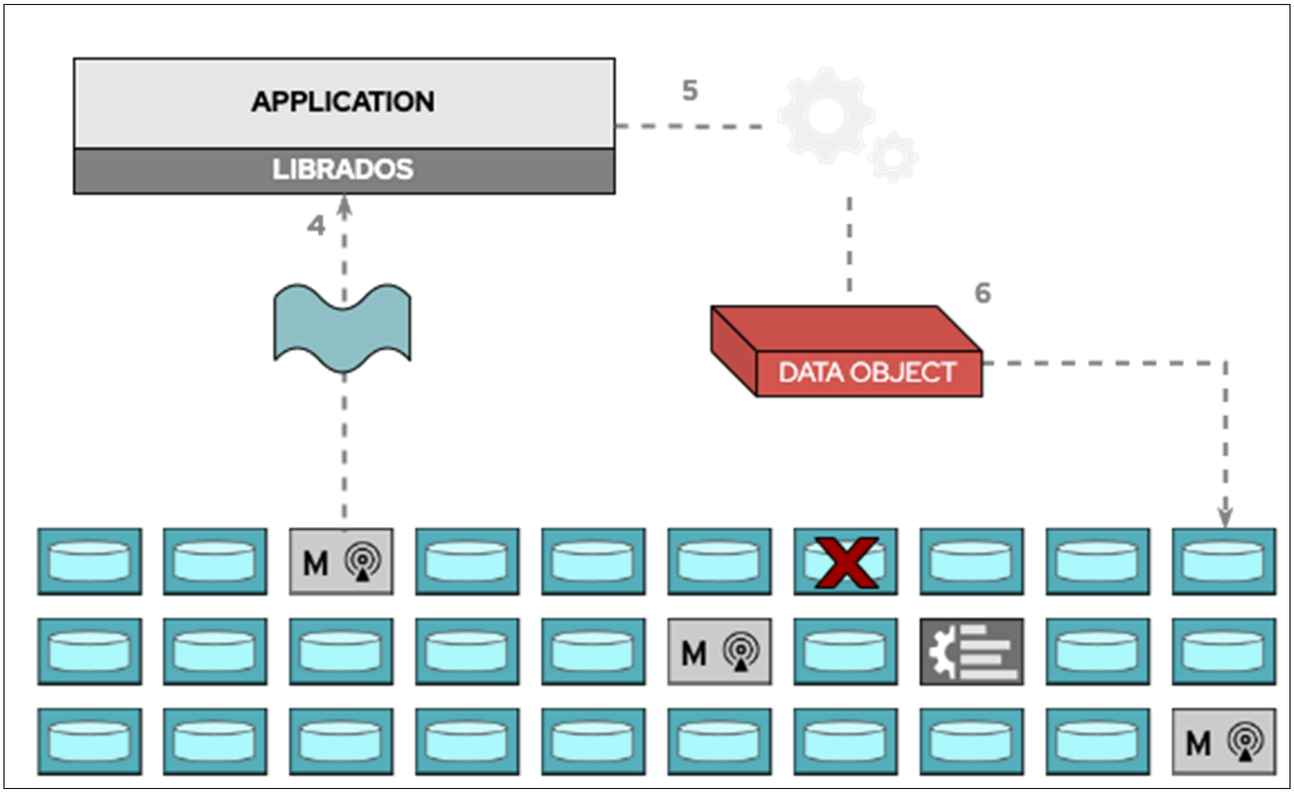


Figure 2-10 Client cluster interaction on OSD failure (case 1)

In the second case, the OSD detects that the client has an outdated map version and provides the necessary map updates that took place since the map version that was used by the Ceph client and the map version that was used by the OSD. On receiving these updates, the Ceph client recalculates data placement and retries the operation, ensuring that it is aware of the latest cluster configuration and can interact with the OSDs.

2.1.5 CephX authentication

Ceph uses the CephX protocol to manage the authorization and authentication between client applications and the Ceph cluster, and between Ceph cluster components. This protocol uses shared secret keys.

CephX is enabled by default during deployment. It is a best practice to keep it enabled for optimal performance. However, some benchmark results that are available online might show CephX disabled to eliminate protocol overhead during testing.

The installation process enables CephX by default, so that the cluster requires user authentication and authorization by all client applications.

Usernames and user IDs

The following user account types exist in a Ceph cluster:

- Accounts for communication between Ceph daemons.
- Accounts for Ceph client applications to access the Ceph cluster.
- An account for the Ceph cluster administrator.

The usernames that are used by the Ceph daemons are expressed as {type}.{id}. For example, mgr.servera, osd.0, or mds.0. They are created during the deployment process.

The usernames that are used by Ceph clients are expressed as client.{id}. For example, when connecting OpenStack Cinder to a Ceph cluster, it is common to create the client.cinder username.

The username that is used by the RADOS Gateways (RGWs) connecting to a Ceph cluster follow the client.rgw.hostname structure.

By default, if no argument is passed to the librados API by code or the Ceph CLI, the connection to the cluster is attempted with the client.admin username.

The default behavior can be configured by using the **CEPH_ARGS** environment variable. To specify a specific username, use **export CEPH_ARGS="--name client.myid"**. To specify a specific user ID, use **export CEPH_ARGS="--id myid"**.

Key rings

On creating a new username, a corresponding key ring file is generated in the Microsoft Windows INI file format. This file contains a section that is named [client.myid] that holds the username and its associated unique secret key. When a Ceph client application running on a different node must connect to the cluster by using this username, the key ring file must be copied to that node.

When the application starts, librados, which ends up being called whatever the access method that is used, searches for a valid key ring file in /etc/ceph. The key ring file name is generated as {clustername}.{username}.Key ring.

The default {clustername} is ceph. For example, if the username is client.testclient, librados searches for /etc/ceph/ceph.client.testclient.keyring.

You can override the location of the key ring file by inserting key ring={keyring_file_location} in the local Ceph configuration file (/etc/ceph/ceph.conf) in a section that is named after your username:

```
[client.testclient]
Key ring=/mnt/homedir/myceph.keyring
```

All Ceph MONs can authenticate users so that the cluster does not present any single point of failure in authentication. The process follows these steps:

1. The Ceph client contacts the MONs with a username and a key ring.
2. Ceph MONs return an authentication data structure like a Kerberos ticket that includes a session key.
3. The Ceph client uses the session key to request specific services.
4. A Ceph MON provides the client with a ticket so it can authenticate with the OSDs.
5. The ticket expires so it can be reused and prevent spoofing risks.

Capabilities

Each username is assigned a set of capabilities that enables specific actions against the cluster.

The cluster offers an equivalent of the Linux root user, which is known as client.admin. By default, a user has no capabilities and must be allowed specific rights. To accomplish this goal, use the **allow** keyword, which precedes the access type that is granted by the capability.

Here are the existing access types that are common to all Ceph daemons:

- `r:` Provides a read permission.
- `w:` Provides a write permission.
- `x:` Provides an run permission on class methods that exist for the daemon.
- `*:` Provides a rwx permission.

Profiles

To simplify capability creation, CephX profiles can be leveraged:

- `profile osd`: A user can connect as an OSD and communicate with OSDs and MONs.
- `profile mds`: A user can connect as an MDS and communicate with MDSs and MONs.
- `profile crash`: Read-only access to MONs for crash dump collection.
- `profile rbd`: A user can manipulate and access RADOS Block Device (RBD) images.
- `profile rbd-read-only`: A user can access RBD images in read-only mode.

Other deployment-reserved profiles exist and are not listed for clarity.

To create a username, run the following command:

```
ceph auth get-or-create client.forrbd mon 'profile rbd' osd 'profile rbd'
```

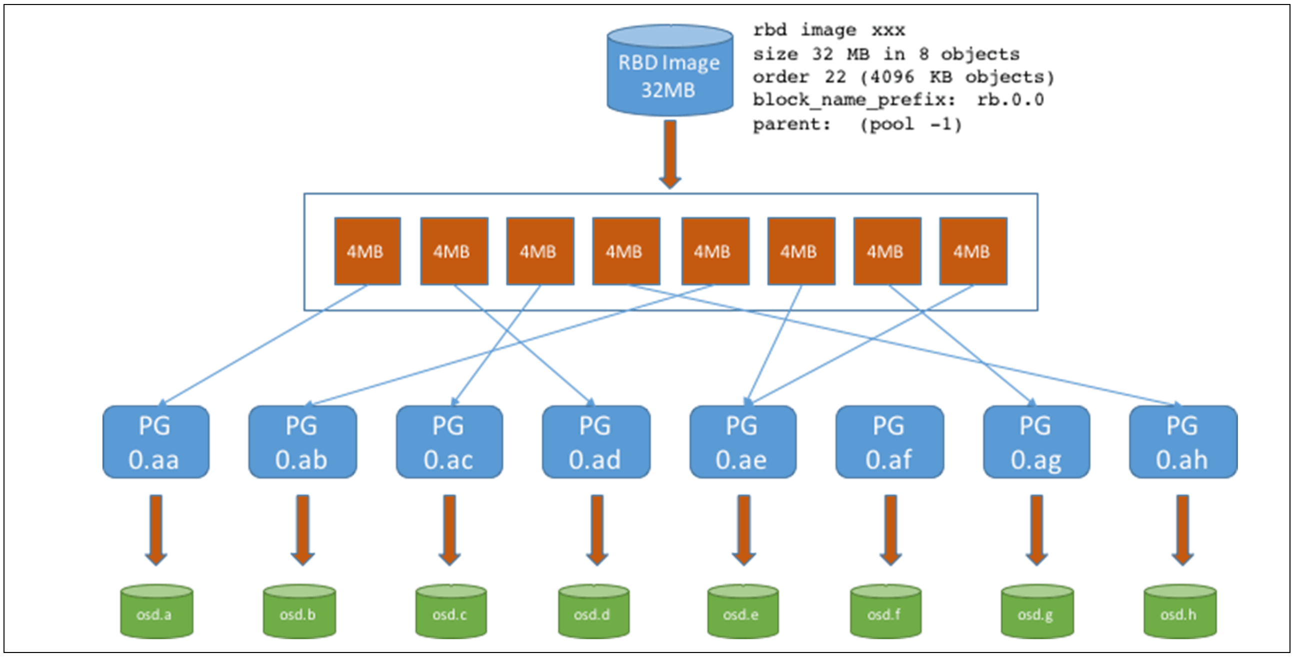
2.2 Access methods

Ceph supports the following access methods:

- `librados` is the native object protocol for direct RADOS access.
- Block through RBDs:
  - Attach support through a Linux kernel module (KRBD).
  - Attach support and boot for Quick Emulator (QEMU), KVM, and OpenStack Cinder through `librbd`.
- File through the Ceph File System (CephFS):
  - Mount support through a Linux kernel module (kceph).
  - Mount support through File System in Userspace (FUSE) for non-Linux environments (ceph-fuse).
  - NFS support through Ganesha.
- HTTS or object through the RGW:
  - S3 protocol support.
  - OpenStack Object Storage Swift support.
  - Static website support.
  - Bucket access through NFS through Ganesha.

All access methods except `librados` automatically stripe the data that is stored in RADOS to 4 MiB objects by default. This setting can be customized. For example, when using CephFS, storing a 1 GiB file from a client perspective results in 256* 4 MiB RADOS objects, each assigned to a PG with the pool that is used by CephFS.

As another example, Figure 2-11 shows the RADOS object layout of a 32 MiB RBD image that is created in a pool with ID 0 on top of OSDs.



```
rbd image xxx
size 32 MB in 8 objects
order 22 (4096 KB objects)
block_name_prefix: rb.0.0
parent:  (pool -1)
```

Figure 2-11 Ceph Virtual Block Device layout in a Ceph cluster

For more information about each access method, see Chapter 3, “IBM Storage Ceph main features and capabilities ” on page 31.

2.3 Deployment

There are many Ceph deployment tools, each of which were available at certain releases:

- **mkcephfs** was the first tool.
- **ceph-deploy** started with Cuttlefish.
- **ceph-asnible** started with Jewel.
- **cephadm** started with Octopus and later.

For more information about how to use `cephadm` to deploy your Ceph cluster, see IBM Storage Ceph documentation. A `cephadm`-based deployment follows these steps:

- For beginners:
  a. Bootstrap your Ceph cluster (create one initial MON and MGR).
  b. Add services to your cluster (OSDs, MDSs, RGWs, and others).
- For advanced users, bootstrap your cluster with a complete service file to deploy everything.

For more information about the guidelines and best practices for cluster deployment, see Chapter 6, “Day 1 and Day 2 operations” on page 125.

3

IBM Storage Ceph main features and capabilities

This chapter describes the main features and capabilities of IBM Storage Ceph.

This chapter has the following sections:

- IBM Storage Ceph access methods
- IBM Storage Ceph object storage
- IBM watsonx.data
- IBM Storage Ceph block storage
- IBM Storage Ceph file storage

3.1 IBM Storage Ceph access methods

An IBM Storage Ceph cluster is a distributed, data object store that provides excellent performance, reliability, and scalability. The key capability of IBM Storage Ceph is that it is a multiprotocol storage system that supports block storage operations, file systems and file sharing, and object storage APIs. Here are the attributes of these access methods:

- Block storage is implemented by the Reliable Autonomic Object Store (RADOS) Block Device (RBD) to support native block volume operations by Ceph clients.
- File services are enabled by the Ceph Metadata Service (`ceph-mds`) to support POSIX file operations and file sharing through standard protocols (for example, NFS and SMB/CIFS).
- Object services are provided by the RADOS Gateway (RGW) to support standard RESTful APIs for object storage operations (for example, S3).

Block storage devices are thin-provisioned, resizable volumes that store data striped over multiple object storage daemons (OSDs). Ceph block devices leverage RADOS capabilities, such as snapshots, replication, and data reduction. Ceph block storage clients communicate with Ceph clusters through kernel modules or the `librbd` library.

Ceph File System (CephFS) is a file system service that is compatible with POSIX standards and built on the Ceph distributed object store. CephFS provides file access to internal and external clients by using POSIX semantics wherever possible. CephFS maintains strong cache coherency across clients. The goal is for processes that use the file system to behave the same when they are on different hosts as when they are on the same host. It is simple to manage and yet meets the growing demands of enterprises for a broad set of applications and workloads. CephFS can be additionally accessed by using industry-standard file sharing protocols such as NFS and SMB/CIFS.

Object storage is a storage solution used in the cloud and by on-premises applications as a central storage platform for large quantities of unstructured data. Object storage continues to increase in popularity due to its ability to address the needs of the world’s largest data repositories. IBM Storage Ceph supports object storage operations through the S3 API with a high emphasis on what is commonly referred to as S3 API fidelity.

3.2 IBM Storage Ceph object storage

Object storage is a type of data storage that stores data as objects, with each object having a unique identifier and associated. These objects are stored in what is referred to as a bucket, and with a storage class / placement target that maps to a storage pool in the Ceph cluster and are accessed through a RESTful API. Object storage differs from traditional file and block storage that store data in a hierarchical file system or as blocks on a storage device. Object storage stores data in a single flat namespace that is addressed by metadata (object name).

Here are the key attributes of object storage:

- Object storage enables applications to store, retrieve, and delete large amounts of unstructured data. It is highly scalable and flexible, meaning that it can easily handle large amounts of data and accommodate growth as needed. Objects can be accessed over a routed IP network from anywhere by any client that supports the RESTful API.
- Object storage is also designed to be highly durable and fault-tolerant. It inherently leverages Ceph data protection features including data replication and erasure coding (EC) to ensure that data is stored redundantly and remains available when underlying hardware components fail. This feature makes object storage especially suitable for use cases such as archiving, backup, and disaster recovery (DR).
- Object storage is cost-effective because it uses commodity hardware, which is less expensive than specialized storage hardware that is used in proprietary storage systems.
- Object storage systems usually feature additional features including data versioning, data tiering, and data lifecycle management.

Object storage can serve many use cases, including archival, backup and DR, media and entertainment, and big data analytics.

3.2.1 IBM Storage Ceph object storage overview

IIBM Storage Ceph uses the Ceph Object Gateway daemon (radosgw to provide object storage client services. The Ceph Object Gateway is an HTTP Server that supports many client applications and use cases. Also referred to as the RGW, it provides interfaces that are compatible with both Amazon S3 and OpenStack Swift, and it has its own user management.

The Ceph Object Gateway provides interfaces that are compatible with OpenStack Swift and Amazon Web Services (AWS) S3. The Ceph Object Gateway has its own user management system and can also be interfaced with external systems including OpenStack Keystone. The Ceph Object Gateway can store data in the same Ceph storage cluster that is used to store data from Ceph Block Device and CephFS client, with separate pools and optionally different storage drive media or a different Controlled Replication Under Scalable Hashing (CRUSH) hierarchy.

The Ceph Object Gateway is a service that runs containerized on a Ceph node and provides object storage access to clients. In a production environment, it is best practice to run multiple instances of the object gateway service on multiple Ceph nodes to provide high availability (HA), which the clients access through an IP Load Balancer (Figure 3-1).

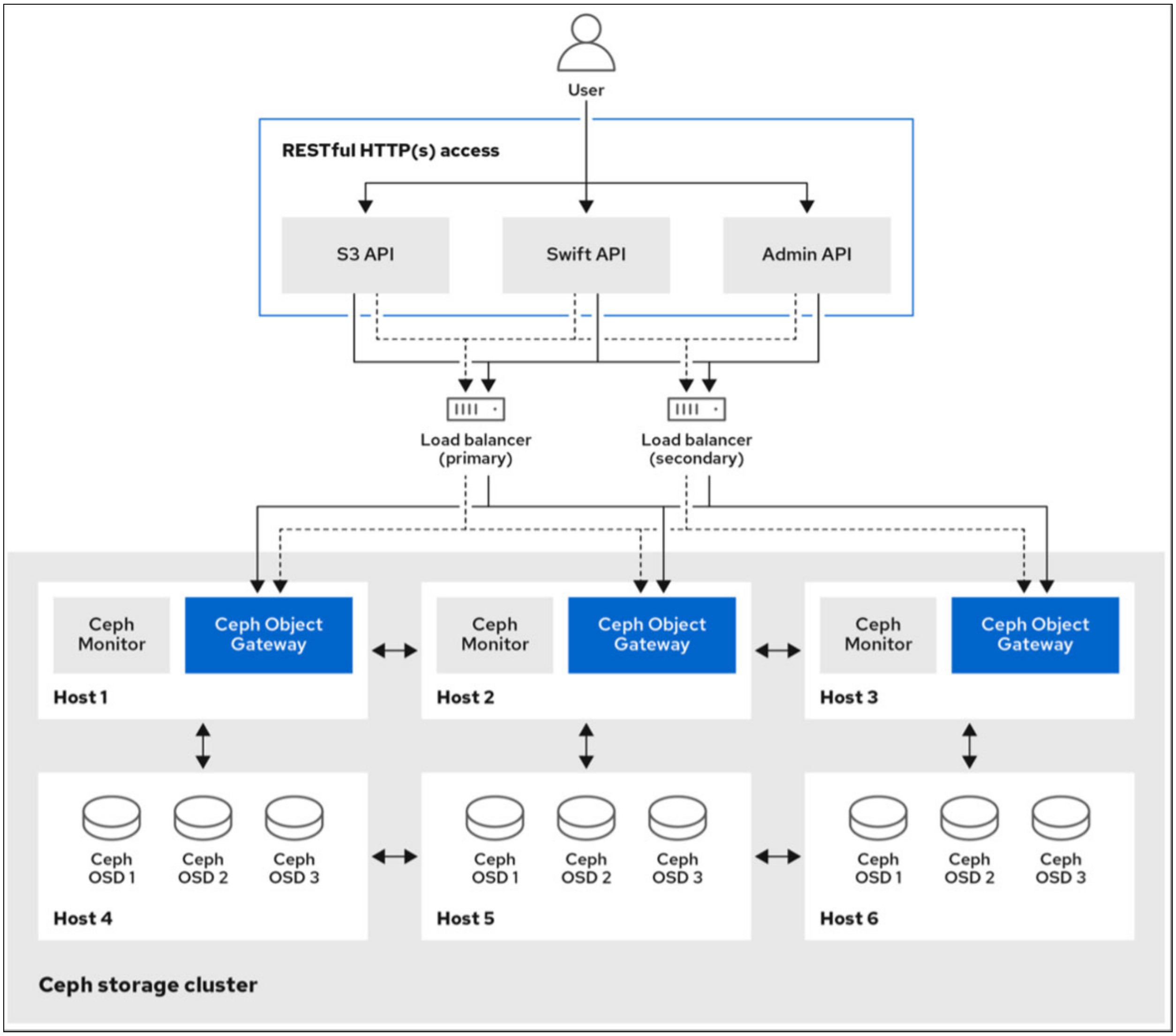


Figure 3-1 Conceptual view of the Ceph Object Gateway

3.2.2 Use cases for IBM Storage Ceph object storage

Typical application use cases for the Ceph Object Gateway across industries and use cases include the following examples:

- Analytics, artificial intelligence, and machine learning data repository. For example, Hadoop, Spark, and an IBM watsonx.data lakehouse. For more information, see *Unlocking Data Insights and AI: IBM Storage Ceph as a Data Lakehouse Platform for IBM watsonx.data and Beyond*, SG24-8563 (in draft at the time of publication).
- IoT data repository. For example, sensor data collection for autonomous driving.

- Auxiliary storage. For example:
  - Active archive: Tiering of inactive data from primary NAS file servers.
  - Storage for backup data: Leading backup applications have native integration with object storage for long-term retention purposes.
- Storage for cloud native applications. Object storage is the *de facto* standard for cloud native applications.
- Event-driven data pipelines.

Industry-specific use cases for the Ceph Object Gateway include the following examples:

- Healthcare and Life Sciences:
  - Medical imaging, such as picture archiving and communication system (PACS) and MRI.
  - Genomics research data.
  - Health Insurance Portability and Accountability Act (HIPAA) of 1996 regulated data.
- Media and entertainment (for example, audio, video, images, and rich media content).
- Financial services (for example, regulated data that requires long-term retention or immutability).
- Object Storage as a Service (SaaS) as a catalog offering (cloud or on-premises).

3.2.3 RADOS Gateway architecture within the Ceph cluster

The IBM Storage Ceph RGW is a client of the RADOS system. It interfaces with RADOS through the librados API. The relationship can be implemented by multiple topology strategies, which are described in 3.2.4, “Deployment options for IBM Storage Ceph object storage” on page 37.

Here are the components of the RGW architecture:

- The RGW serves as an internal client for Ceph Storage, and facilitates object access for external client applications. Client applications interact with the RGW through standardized S3 or Swift APIs, and the RGW uses `librados` module calls to communicate with the Ceph cluster.

  Figure 3-2 shows the RGW within the Ceph cluster.

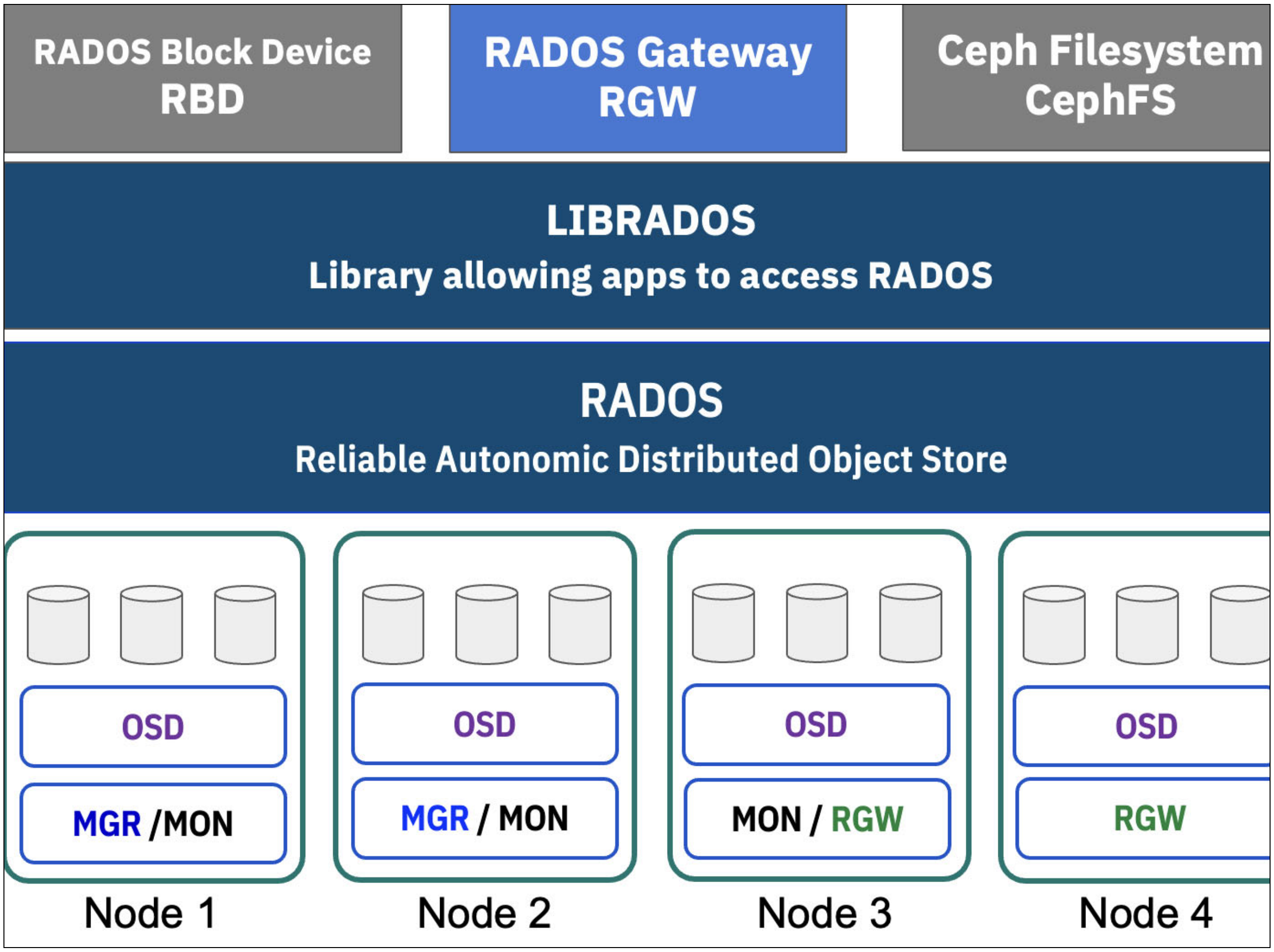


Figure 3-2 RADOS Gateway within the Ceph cluster

- The RGW stores its data, including user data and metadata, in a dedicated set of Ceph Storage pools. This ensures efficient data management and organization within the Ceph cluster.
- RGW Data pools are often built on hard disk drives (HDDs) with EC for cost-effective and high-capacity configurations.
- Data pools can also use solid-state drives (SSDs) with replication or EC to cater to performance-sensitive object storage workloads. QLC SSDs with lower cost and dense capacities are especially suited to object storage.
- The bucket index pool, which stores bucket index key/value entries (one per object), is a critical component of Ceph Object Gateway performance. Due to its performance-sensitive nature and the potential for millions of object entries, the bucket index pool should exclusively use SSDs to ensure optimal performance and responsiveness.

Figure 3-3 on page 37 shows the interaction of these components.

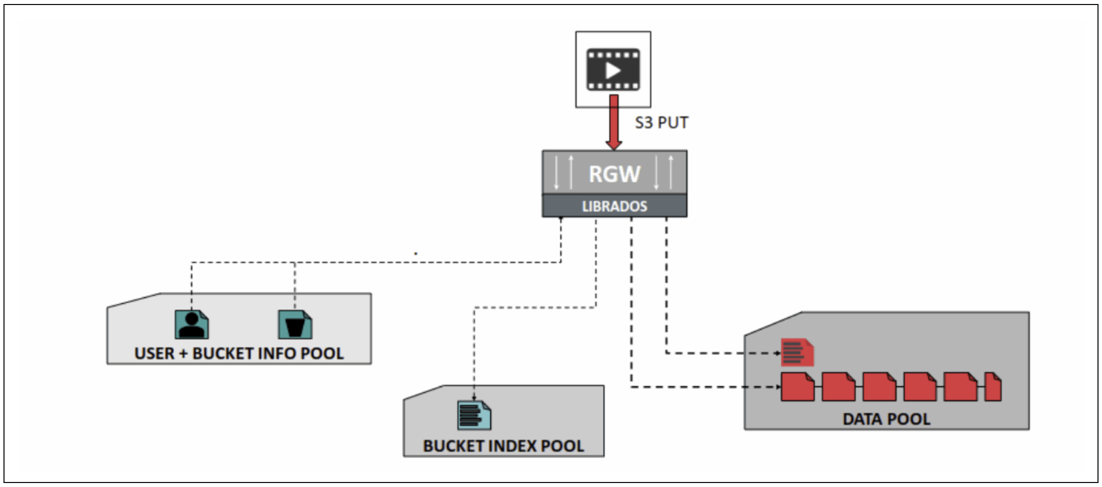


Figure 3-3 Interaction of the Ceph Object Gateway components

3.2.4 Deployment options for IBM Storage Ceph object storage

The IBM Storage Ceph RGW can be deployed by using several strategies. For example:

- Instances of collocated RGW daemons on nodes that are shared with other Ceph services (Figure 3-4).

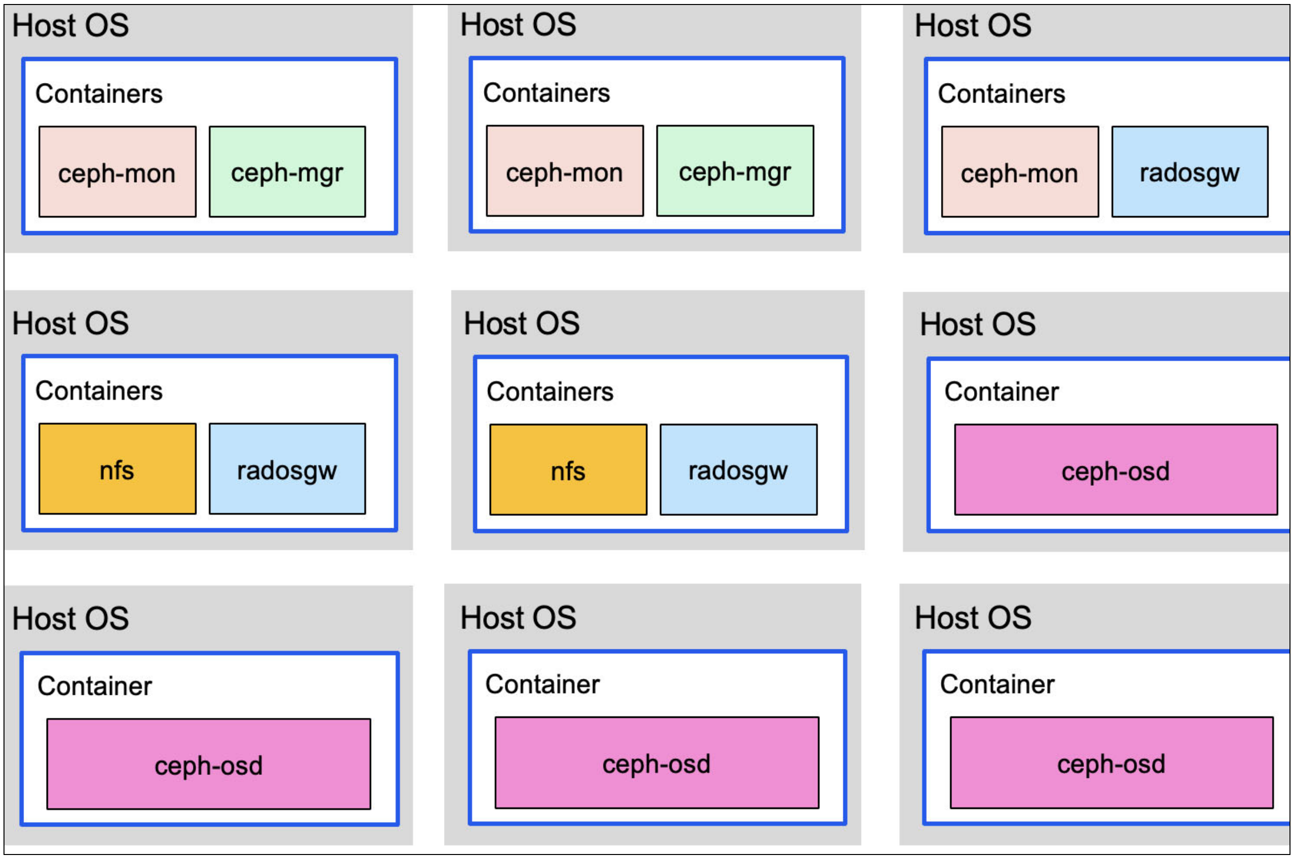


Figure 3-4 RADOS Gateway collocated daemons

- Instances of non-collocated RGW daemons on nodes that are dedicated to the RGW service (Figure 3-5).

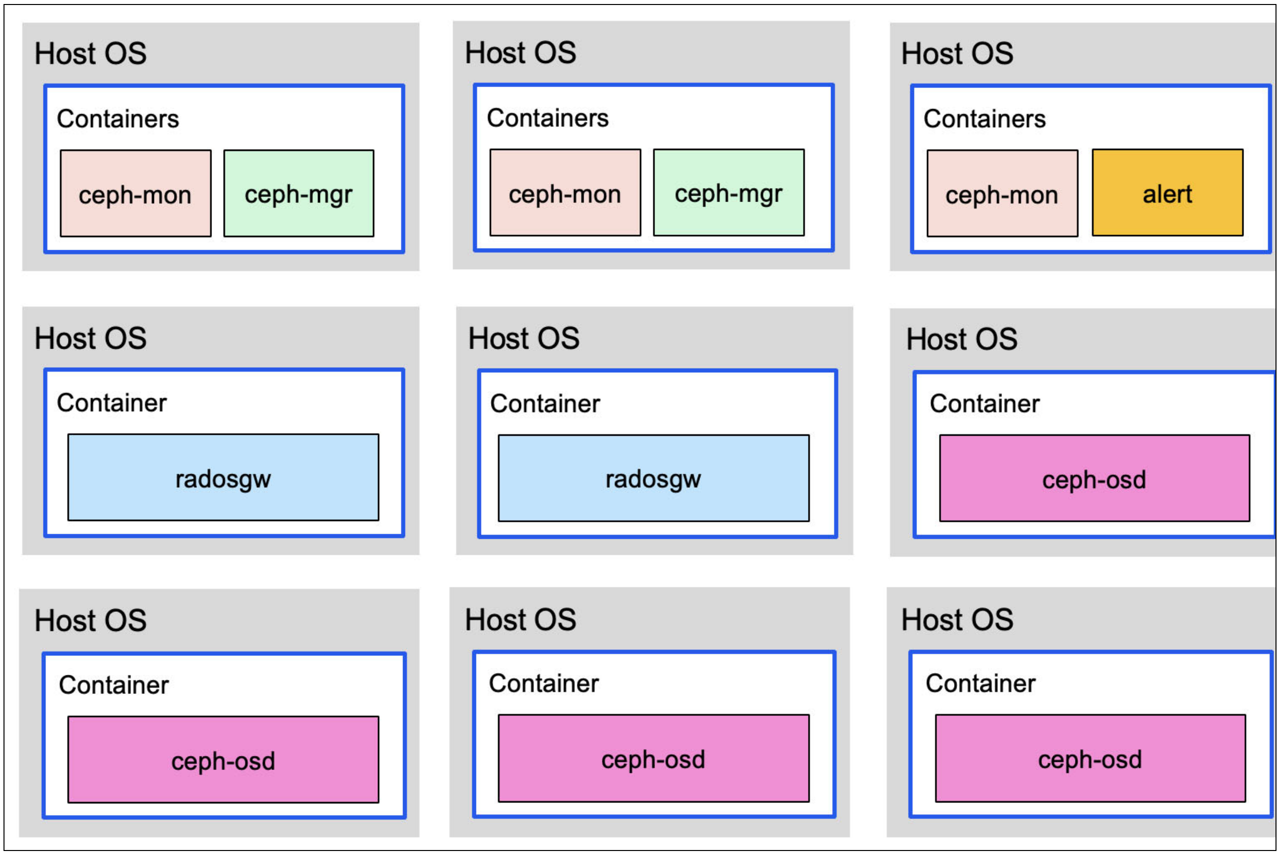


Figure 3-5 RADOS Gateway non-collocated daemons

- Multiple instances of the RGW daemons on a node that is either collocated or non-collocated with other Ceph services (Figure 3-6 on page 39).

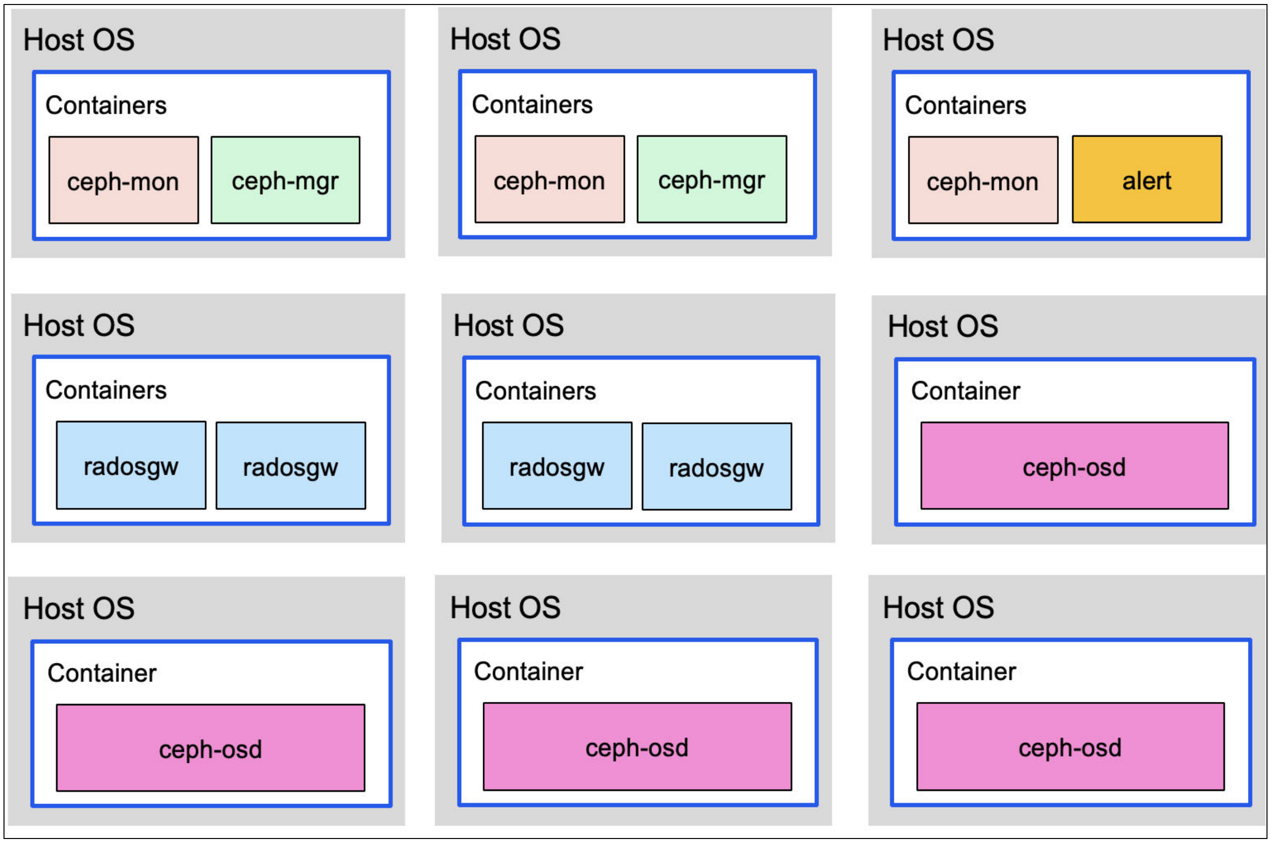


Figure 3-6 Multiple RADOS Gateway daemons on the same node

3.2.5 IBM Storage Ceph object storage topology

IBM Storage Ceph leverages the Ceph Object Gateway daemon (**radosgw**) to provide object storage client access to the cluster. This HTTP Server, also known as the RGW, caters to a diverse range of client applications and use cases.

The RGW offers interfaces that are compatible with AWS S3 and OpenStack Swift, which provide seamless integration with existing cloud environments. Also, it features an admin operations RESTful API for automating day-to-day operations, which streamlines management tasks.

The Ceph Object Gateway constructs of Realm, Zone Groups, and Zones are used to define the organization of a storage network for purposes of replication and site protection.

A deployment in a single data center is simple to install and manage. In recent feature updates, the Dashboard UI was enhanced to provide a single point of control for the deployment and ongoing management of the Ceph Object Gateway in a single data center. In this simple case, there is no need to define Zones and Zone Groups, and a minimized topology is automatically created.

For deployments that involve multiple data centers and multiple IBM Storage Ceph clusters, a more detailed configuration is required with granular control by using the **cephadm** CLI or the dashboard UI. In these scenarios, the Realm, Zone Group, and Zones must be defined by the storage administrator and configured. For more information about these constructs, see the IBM Storage Ceph documentation.

Figure 3-7 shows a design supporting a multiple data center implementation for replication and DR.

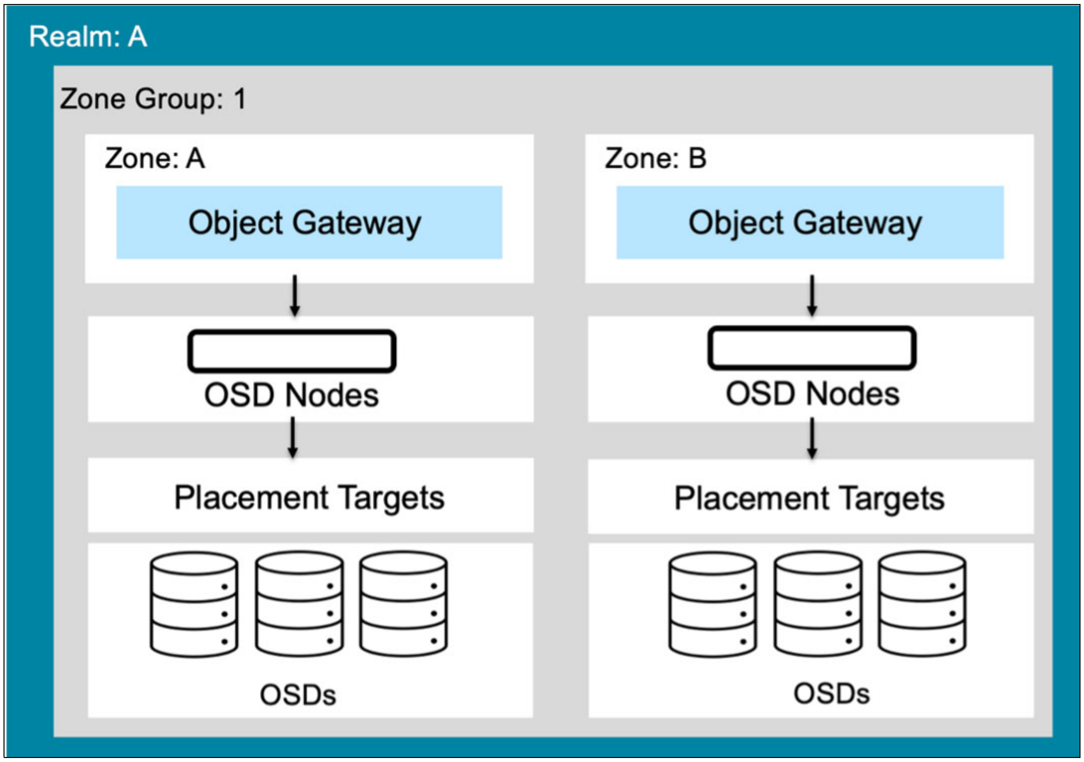


Figure 3-7 Ceph Object Gateway multi-site constructs

To avoid single points of failure in a Ceph RGW deployment, one can present to clients an S3/RGW endpoint that can tolerate the failure of one or more back end RGW services. RGW is a restful HTTP endpoint that can be load-balanced for HA and increase performance. There are some great examples of different RGW load-balancing mechanisms in this repository.

Starting with IBM Storage Ceph 5.3, Ceph provides an HA and load-balancing named ingress, which is based on keepalived and haproxy. With the ingress service, one may easily deploy an HA endpoint for RGW with a minimum set of configuration options.

The orchestrator deploys and manages a combination of haproxy and keepalived to balance the load through a floating virtual IP address. If SSL is used, then SSL must be configured and terminated by the ingress service, not RGW itself.

3.2.6 IBM Storage Ceph object storage key features

The RGW supports many advanced features that are specific to object storage and the S3 API by leveraging the capabilities of an IBM Storage Ceph cluster.

Ceph Object Gateway client interfaces

The Ceph Object Gateway supports three interfaces:

- S3 compatibility
- Swift compatibility
- Administrative API

S3 compatibility

S3 compatibility provides object storage functions with an interface that is compatible with a large subset of the AWS S3 RESTful API.

Swift compatibility

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

The S3 and Swift APIs share a common namespace, so you can write data with one API and retrieve it with the other. The S3 namespace can also be shared with NFS clients to offer a true multiprotocol experience for unstructured data use cases.

Administrative API

Provides an administrative restful API interface for managing the Ceph Object Gateways.

Administrative API requests are sent via a URI that starts with the admin resource endpoint. Authorization for the administrative API mimics the S3 authorization convention. Some operations require the user to have special administrative capabilities. XML or JSON format responses can specified by the format option in the request. JSON format is the default.

Ceph Object Gateway features

The Ceph Object Gateway supports the following key features. This is not an exhaustive list, but rather addresses interest expressed by customers, IBM Business Partners, and Ceph champions.

Management

The Ceph Object Gateway can be managed by using the Ceph Dashboard UI, the Ceph CLI (`cephadm`), the Administrative API, and through service specification files.

Authentication and authorization

IBM Storage Ceph seamlessly integrates with the Security Token Service (STS) to enable authentication of object storage users against your enterprise identity provider (IDP), such as LDAP or Active Directory, through an OpenID Connect (OIDC) provider. This integration leverages the STS to issue temporary security credentials, ensuring secure access to your object storage resources.

IBM Storage Ceph object storage further enhances security by incorporating IAM compatibility for authorization. This feature introduces IAM Role policies, empowering users to request and assume specific roles during STS authentication. By assuming a role, users inherit the S3 permissions that are configured for that role by an RGW administrator. This role-based access control (RBAC) or attribute-based access control (ABAC) approach enables granular control over user access, ensuring that users access only the resources that they need.

Data at rest encryption

The Ceph Object Gateway supports HTTPS, data at rest encryption, FIPS 140-2 compliance, and over the wire encryption with Messenger 2.1. Data at rest encryption is further supported by SSE, including the variants SSE-C, SSE-KMS, and SSE-S3.

Immutability

IBM Storage Ceph Object storage also supports the S3 Object Lock API in both Compliance and Governance modes. Ceph has been certified or passed the compliance assessments for SEC 17a-4(f), SEC 18a-6(e), Financial Industry Regulatory Authority (FINRA) 4511(c), and CFTC 1.31(c)-(d). Certification documents are available on publication.

Archive zone

The archive zone uses multi-site replication and S3 object versioning features. The archive zone retains all versions of all objects available even when they are deleted from a production zone.

An archive zone provides S3 object version history that can be eliminated only through the RGWs associated with the archive zone. Including an archive zone in your multisite zone replication setup provides the convenience of S3 object history while saving space that replicas of the versioned S3 objects would consume in production zones.

One can manage the storage space usage of the archive zone through bucket lifecycle policies, where one can define the number of versions to keep for each object.

An archive zone helps protect your data against logical or physical errors. It can save users from logical failures, such as accidentally deleting a bucket in the production zone. It can also save your data from massive hardware failures, like a complete production site failure. Also, it provides an immutable copy, which can serve as a key component of a ransomware protection strategy.

Security

IBM Storage Ceph provides the following security features for data access and management:

- Comprehensive auditing: Track user activity and access attempts with granular detail by enabling RGW operations per second (OPS) logs, which provides a valuable audit trail for data access accountability.
- Multi-factor authentication (MFA) for deletes: Prevent accidental or unauthorized data deletion with an extra layer of security. MFA requires an additional verification step beyond a traditional password.
- Secure Token Service (STS) integration: Enhance security by using short-lived, limited-privilege credentials that are obtained through STS. This approach eliminates the need for long-lived S3 credentials, reducing the risk that is associated with compromised keys.
- Secure communication with TLS/SSL: Protect data in transit by securing the S3 HTTP endpoint that is provided by RGW services with TLS/SSL encryption. Both external and self-signed SSL certificates are supported for flexibility.

Replication

BM Ceph Object Storage provides enterprise-grade, highly mature, geo-replication capabilities. The RGW multi-site replication feature facilitates asynchronous object replication across single or multi-zone deployments. Leveraging asynchronous replication with eventual consistency, Ceph Object Storage operates efficiently over LAN or WAN connections between replicating sites by leveraging asynchronous replication with eventual consistency.

With the 6.1 release, Ceph Object Storage introduces granular bucket-level replication, unlocking a plethora of valuable features. Users can now enable or disable syncing of individual buckets, enabling precise control over replication workflows. This approach enables full-zone replication while opting out specific buckets, replicating a single source bucket to multi-destination buckets, and implementing both symmetrical and directional data flow configurations. .

Figure 3-8 shows the IBM Ceph Object Storage replication feature.

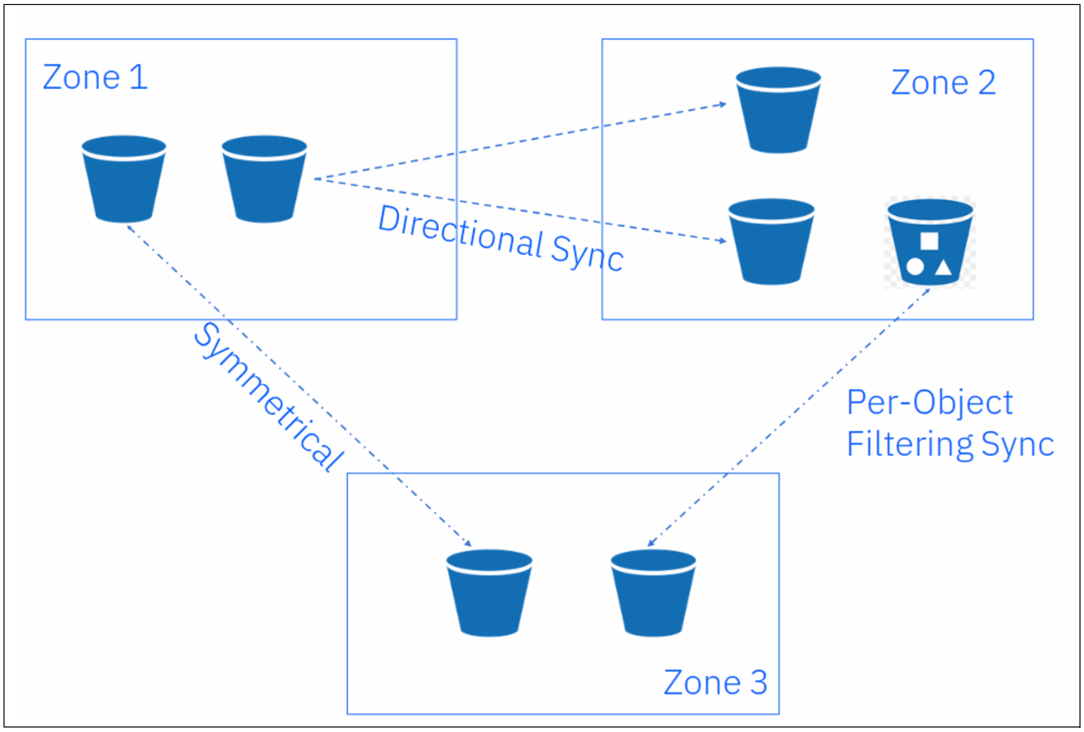


Figure 3-8 IBM Ceph Object Storage replication feature

Storage policies

Ceph Object Gateway supports multiple storage classes for the placement of RGW data. For example, SSD storage pool devices are a best practice for index data and a default storage class to house small client objects and HEAD RADOS objects, and erasure-coded HDD or QLC storage pool devices may be targeted for high-capacity bucket data. Ceph Object Gateway also supports storage lifecycle (LC) policies and migrations for data placement across tiers of storage as content ages.

Migrations across storage classes and protection policies (for example, replicated and EC pools) are supported. Ceph Object Gateway also supports policy-based data archiving to AWS S3, Azure or IBM Cloud Object Storage.

Figure 3-9 shows the Ceph Object Gateway storage policies.

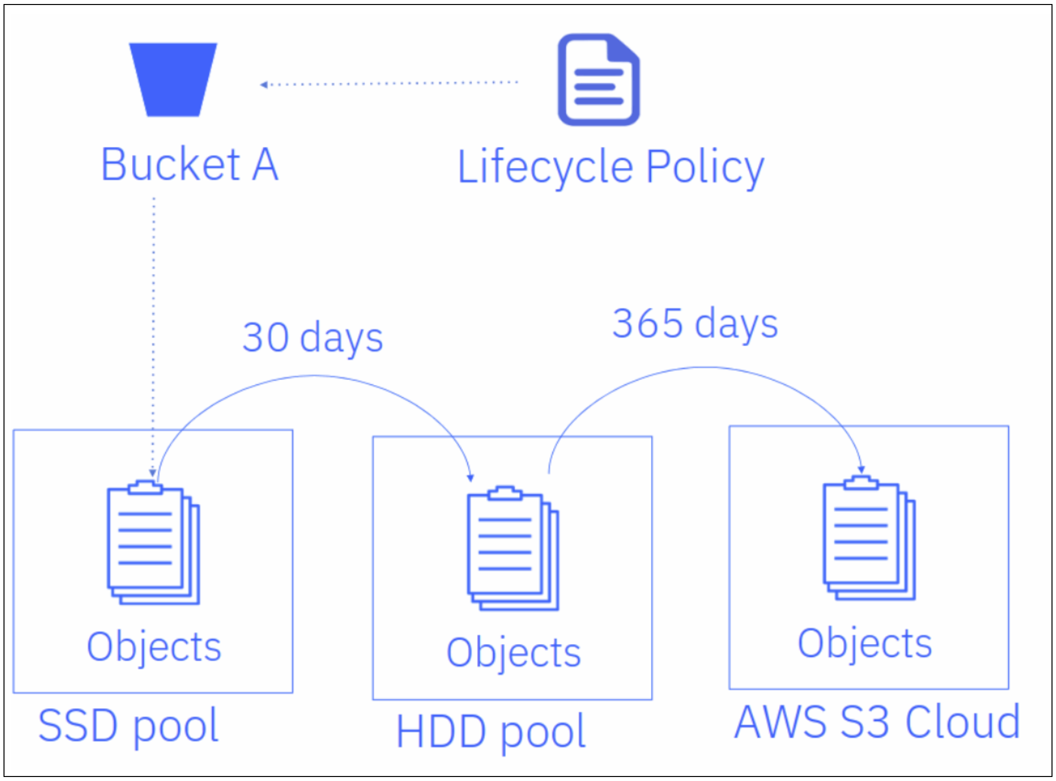


Figure 3-9 Storage policies

Multiprotocol

Ceph Object Gateway supports a single unified namespace for S3 client operations (S3 API) and NFS client operations (NFS Export) to the same bucket. This support provides a true multiprotocol experience for many use cases, particularly in situations where applications are being modernized from traditional file sharing access to native object storage access. As a best practice, limit the usage of S3 and NFS to the same namespace in use cases such as data archives, rich content repositories, or document management stores, that is, use cases where the files and objects are unchanging by design. Multiprotocol is not recommended for live collaboration use cases where multiuser modifications to the same content are required.

IBM watsonx.data

IBM Storage Ceph is the perfect candidate for a data lake or data lakehouse. For example, watsonx.data includes an IBM Storage Ceph license. The integration and level of testing between watsonx.data and IBM Storage Ceph is first rate. Some of the features that make Ceph a great solution for watsonx.data are S3 Select, Table Encryption, IDP authentication with STS, and Datacenter Caching with Datacenter-Data-Delivery Network (D3N).

S3 Select is a recent innovation that extends object storage to semi-structured use cases. An example of a semi-structured object is one that contains comma-separated values (CSV), JSON, or Parquet data formats. S3 Select enables a client to retrieve a subset of an object’s content by using SQL-like arguments to filter the resulting payload. At the time of writing, Ceph Object Gateway supports S3 Select for alignment with data lakehouse storage by IBM watsonx.data clients, and S3 Select supports the CSV, JSON, and Parquet formats.

D3N uses high-speed storage such as NVMe SSDs or DRAM to cache data sets near clients. D3N improves the performance of big data jobs running in analysis clusters by accelerating recurring reads from the data lake.

Bucket features

Ceph supports advanced bucket features like S3 bucket policy, S3 object versioning, S3 object lock, rate limiting, bucket object count quotas, and bucket capacity quotas. In addition to these advanced bucket features, Ceph Object Gateway boasts impressive scalability that empowers organizations to efficiently store and access massive amounts of data.

> **Note:** In a 6-server and 3-enclosure configuration (see Evaluator Group tests performance for 10 billion objects with Red Hat Ceph Storage), Ceph Object Gateway has supported 250 million objects in a single bucket and 10 billion objects overall.

Bucket notifications

Ceph Object Gateway supports bucket event notifications, which is a crucial feature for event-driven architectures that is widely used when integrating with Red Hat OpenShift Data Foundation. Notifications enable real-time event monitoring and triggering of downstream actions that include data replication, alerting, and workflow automation.

> **Note:** For more information about event-driven architectures, see Chapter 4, “S3 bucket notifications for event-driven architectures”, in *IBM Storage Ceph Solutions Guide*, REDP-5715.

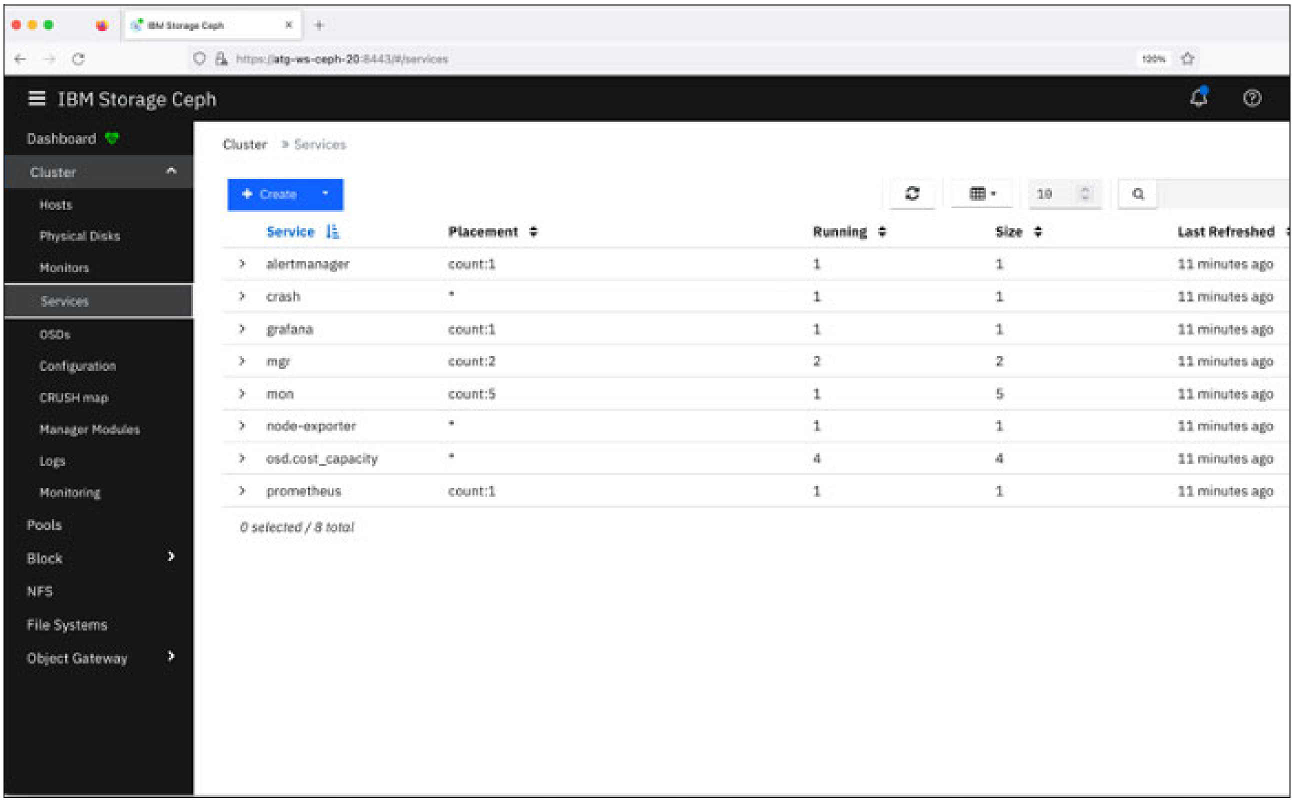
3.2.7 Ceph Object Gateway deployment step-by-step

This section demonstrates how to deploy the Ceph Object Gateway, also known as the RGW, and explore the S3 object storage client experience.

Starting the RGW service

To start the RGW service, complete the following steps:

1. From the Ceph Dashboard in the navigation pane, expand the **Cluster** group and select **Services**. The Services page opens along with the running Ceph services (daemons). Click **Create** to configure and start an instance of the RGW service (Figure 3-10).

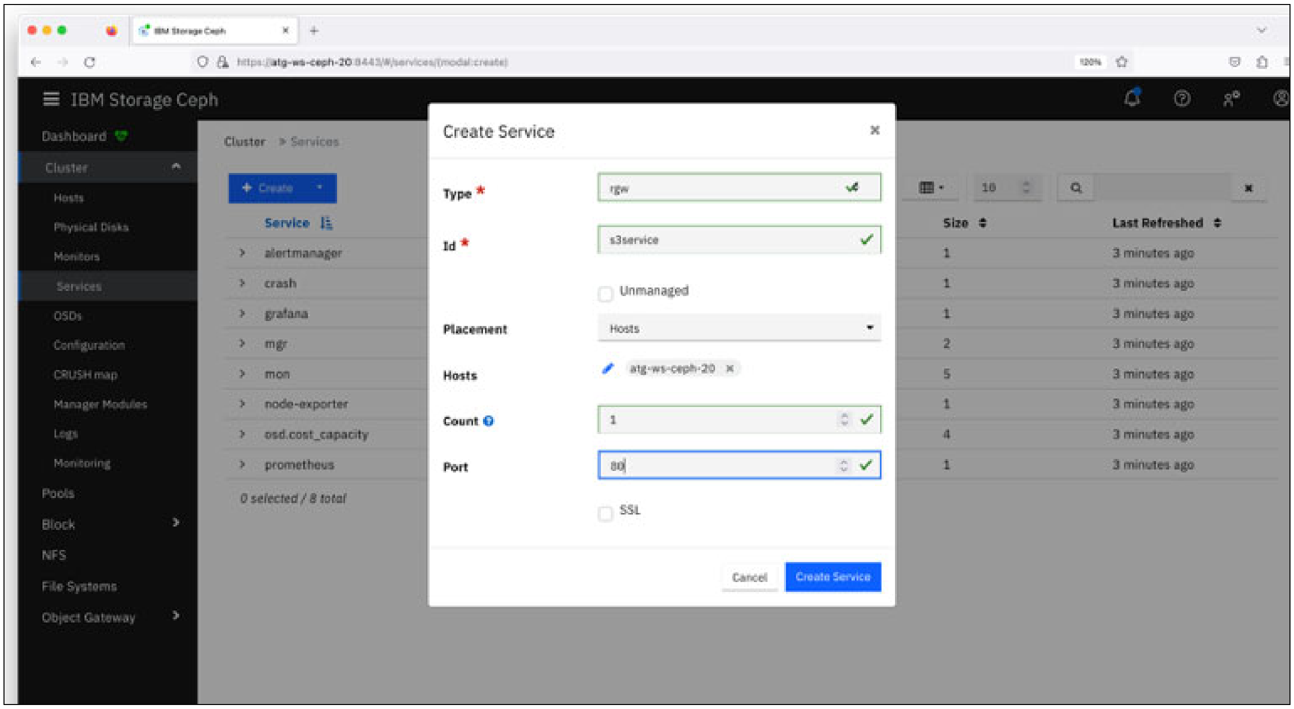


Service	Placement	Running	Size	Last Refreshed
alertmanager	count:1	1	1	11 minutes ago
crash	*	1	1	11 minutes ago
grafana	count:1	1	1	11 minutes ago
mgr	count:2	2	2	11 minutes ago
mon	count:5	1	5	11 minutes ago
node-exporter	*	1	1	11 minutes ago
osd.cost_capacity	*	4	4	11 minutes ago
prometheus	count:1	1	1	11 minutes ago

0 selected / 8 total

Figure 3-10 Ceph cluster services window

2. In the Create Service dialog box, enter values similar to the ones that are shown in Figure 3-11 on page 47. Then, click **Create Service**.



Create Service

Type * rgw

Id * s3service

☐ Unmanaged

Placement Hosts

Hosts atg-ws-ceph-20

Count 1

Port 80

☐ SSL

Cancel Create Service

Service	Size	Last Refreshed
alertmanager	1	3 minutes ago
crash	1	3 minutes ago
grafana	1	3 minutes ago
mgr	2	3 minutes ago
mon	5	3 minutes ago
node-exporter	1	3 minutes ago
osd.cost_capacity	4	3 minutes ago
prometheus	1	3 minutes ago

0 selected / 8 total

Figure 3-11 Ceph Object gateway service startup window

For this example, use the following RGW configuration values:

- Type: RGW
- Id: s3service
- Placement: Hosts
- Host: <ceph node hostname>
- Count: 1
- Port: 80

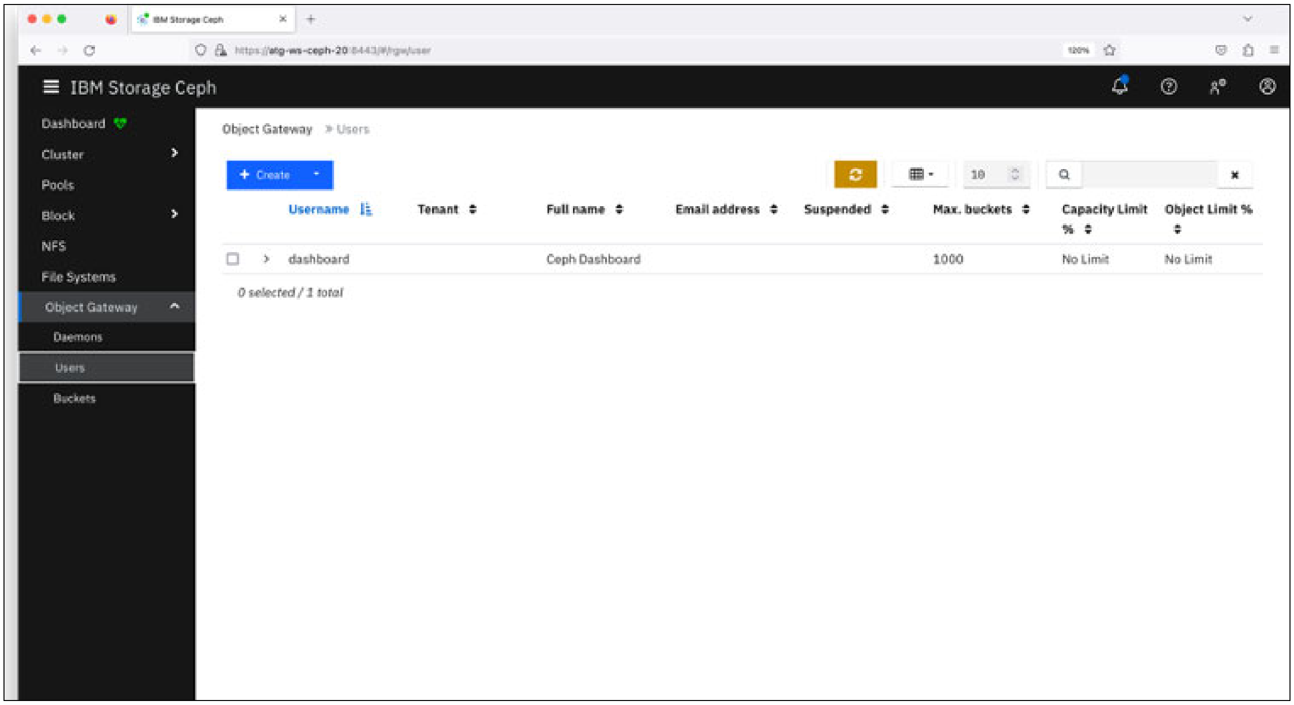
To observe the running RGW service, use any of the following dashboard locations:

- **Ceph Dashboard home page → Object Gateways section**
- **Ceph Dashboard → Cluster → Services**
- **Ceph Dashboard → Object Gateway - Daemons**

Creating an Object Gateway user

To create an Object Gateway user, complete the following steps:

1. From the Ceph Dashboard in the navigation pane, expand the **Object Gateway** group and select **Users**. The Object Gateway Users page opens (Figure 3-12). Click **Create** and proceed to the Create User window.

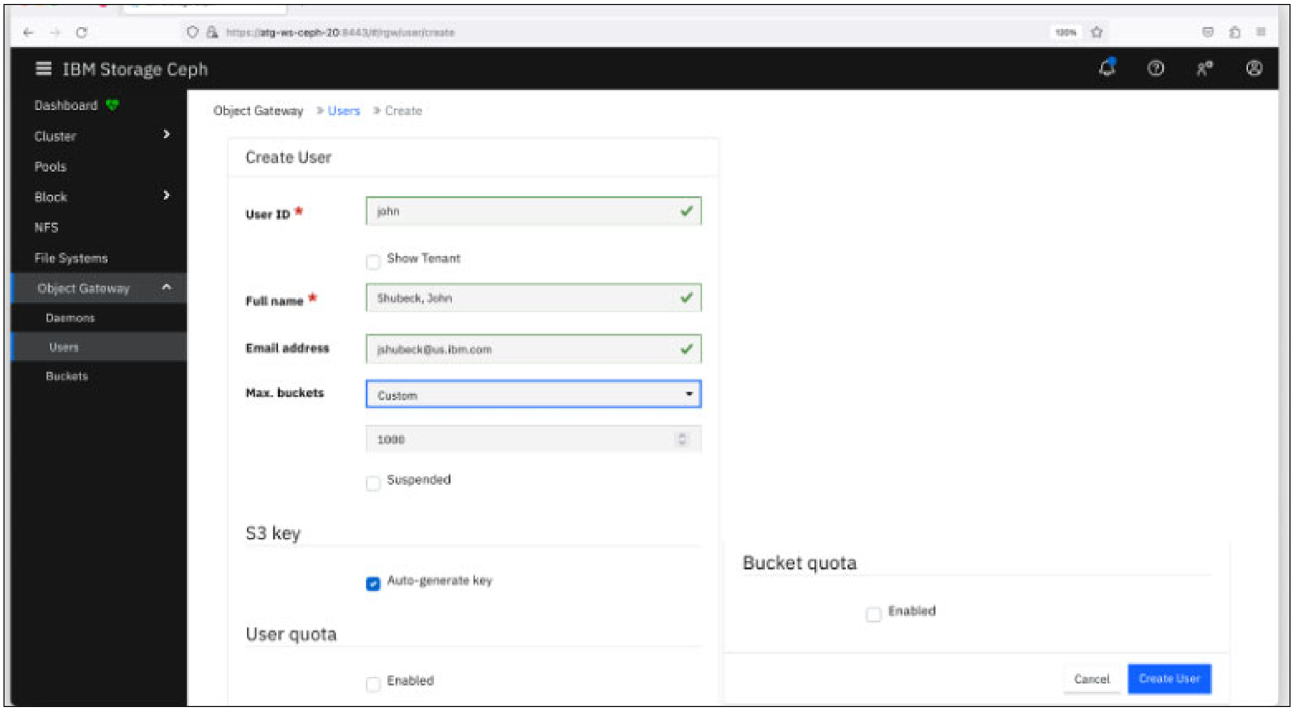


		Username	Tenant	Full name	Email address	Suspended	Max. buckets	Capacity Limit %	Object Limit %
□	›	dashboard		Ceph Dashboard			1000	No Limit	No Limit

0 selected / 1 total

Figure 3-12 Ceph Object Gateway user listing

2. In the Create User window, enter the required values that are shown in Figure 3-13. When finished, click **Create User**.



Create User

User ID *: john

□ Show Tenant

Full name *: Shubeck, John

Email address: jshubeck@us.ibm.com

Max. buckets: Custom

1000

□ Suspended

S3 key

☑ Auto-generate key

User quota

□ Enabled

Bucket quota

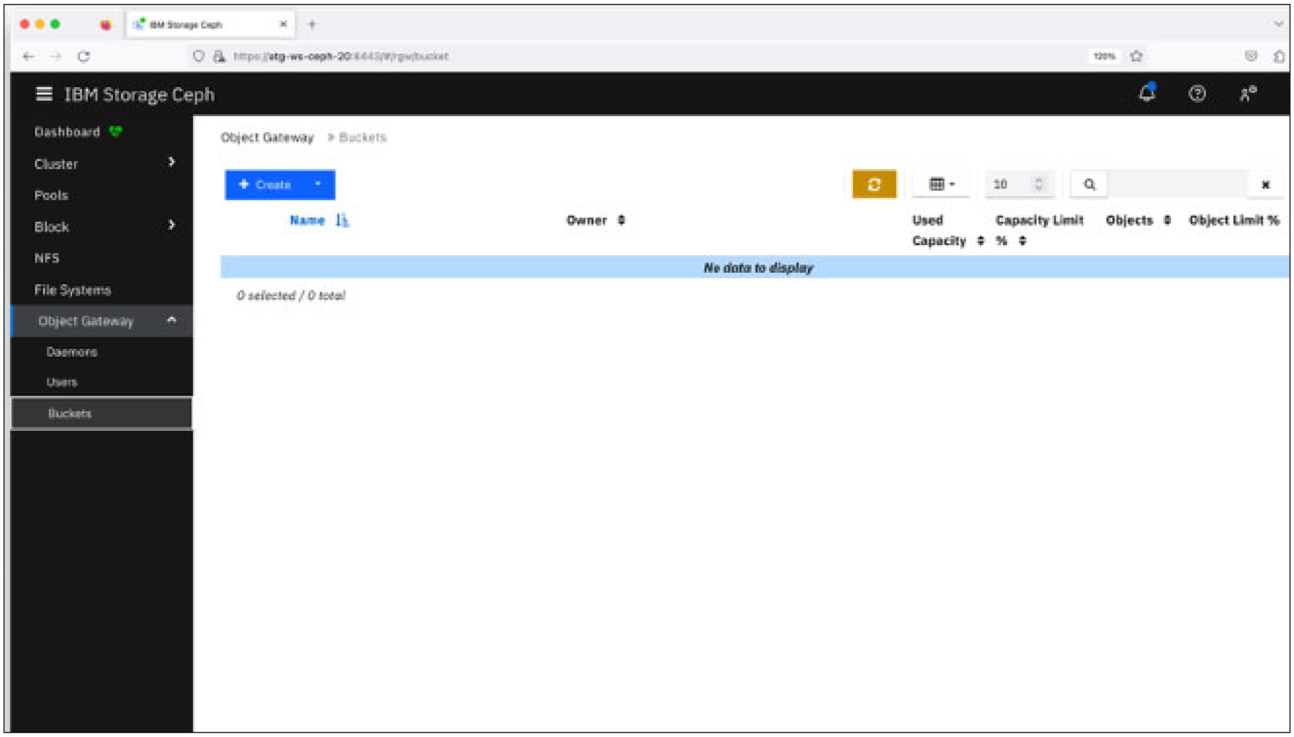
□ Enabled

Figure 3-13 Ceph Object Gateway: Create User window

Creating an Object Gateway S3 bucket

To create an Object Gateway S3 bucket, complete the following steps:

1. From the Ceph Dashboard in the navigation pane, expand the **Object Gateway** group and select **Buckets**. The Object Gateway Buckets window opens (Figure 3-14). Click **Create** and proceed to the Create Bucket window.

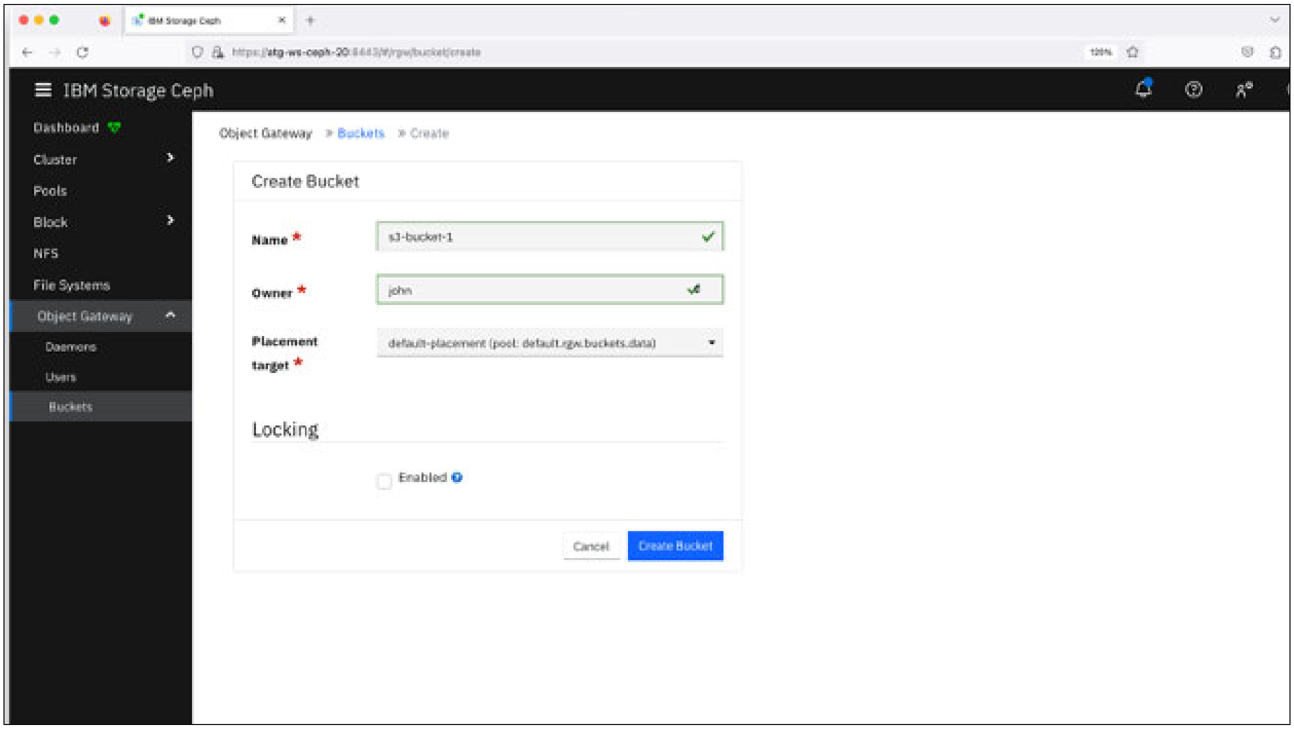


Name	Owner	Used Capacity	Capacity Limit %	Objects	Object Limit %
No data to display					

0 selected / 0 total

Figure 3-14 Ceph Object Gateway bucket listing

2. In the Create Bucket window, enter the required values that are shown in Figure 3-15 on page 51. When finished, click **Create Bucket**.



Create Bucket

Name * s3-bucket-1

Owner * john

Placement target * default-placement (pool: default.rgw.buckets.data)

Locking

□ Enabled

Figure 3-15 Ceph cluster services window

For this example, use the following Ceph Object gateway bucket configuration values:

- Name: Any bucket name that you prefer.
- Owner: The user that you created in “Creating an Object Gateway user” on page 48.
- Placement: Accept the default value.

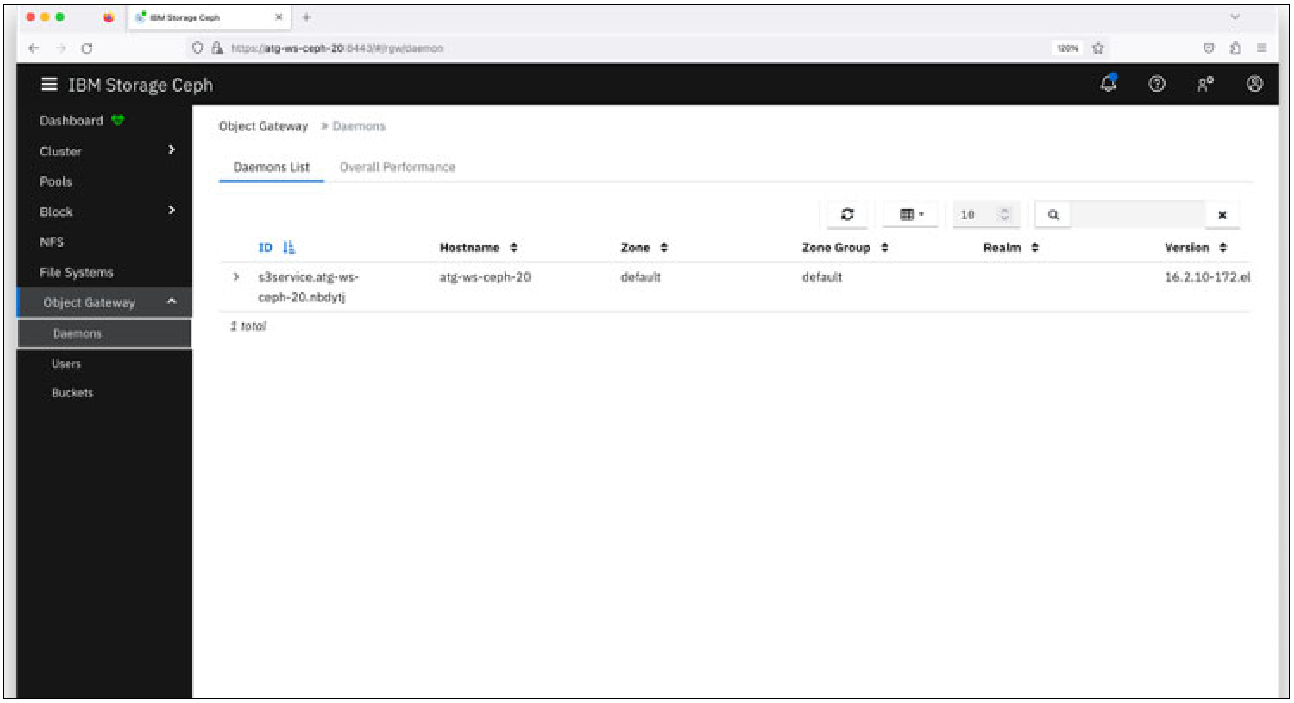
3.2.8 Ceph Object Gateway client properties

This section describes the S3 API client experience in using **PUT** and **GET** on a file to place it into the Object Gateway bucket that was created in “Creating an Object Gateway S3 bucket” on page 50.

Verifying the Ceph Object Gateway configuration

In this example, we proceed through three Object Gateway configuration windows. If any of these settings are not configured, then return to 3.2.7, “Ceph Object Gateway deployment step-by-step” on page 45 to finish the RGW configuration tasks.

Figure 3-16 shows the Ceph Object gateway services listing.

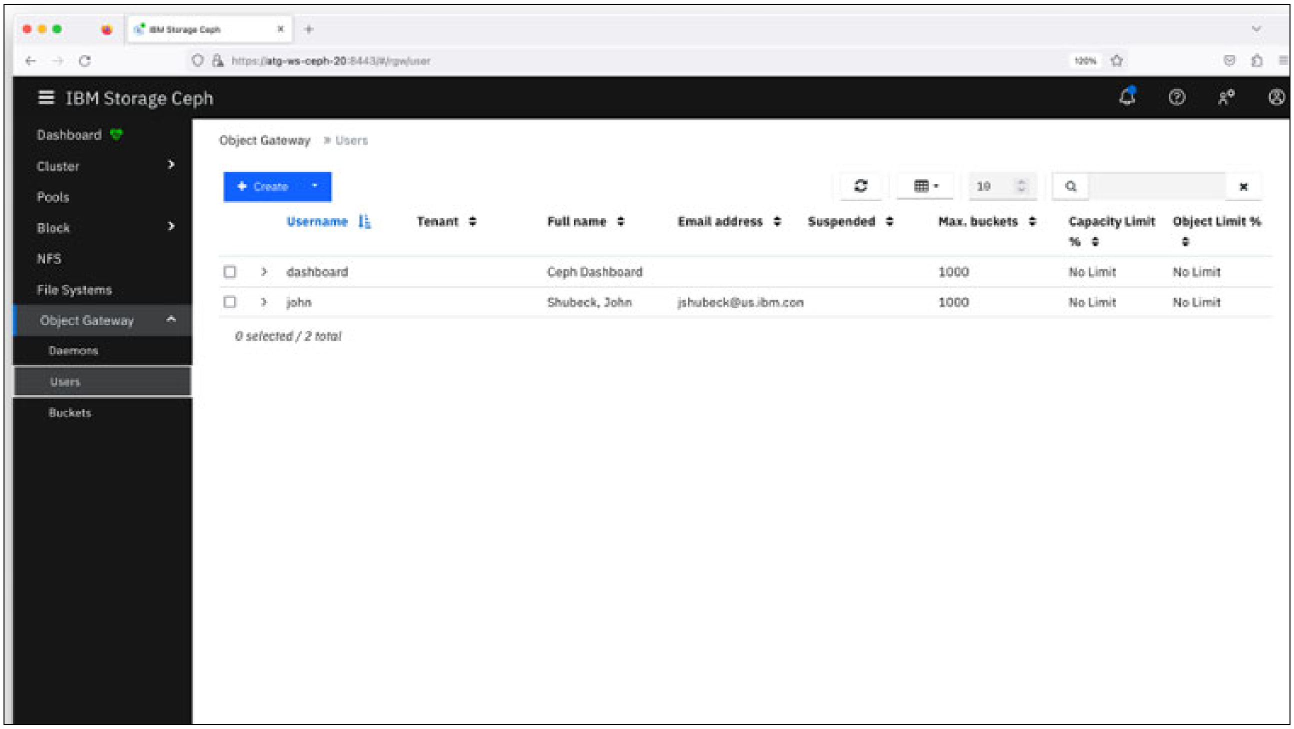


ID	Hostname	Zone	Zone Group	Realm	Version
s3service.atg-ws-ceph-20.nbdytj	atg-ws-ceph-20	default	default		16.2.10-172.el

1 total

Figure 3-16 Ceph Object Gateway services listing

Figure 3-17 shows the Ceph Object Gateway user listing.

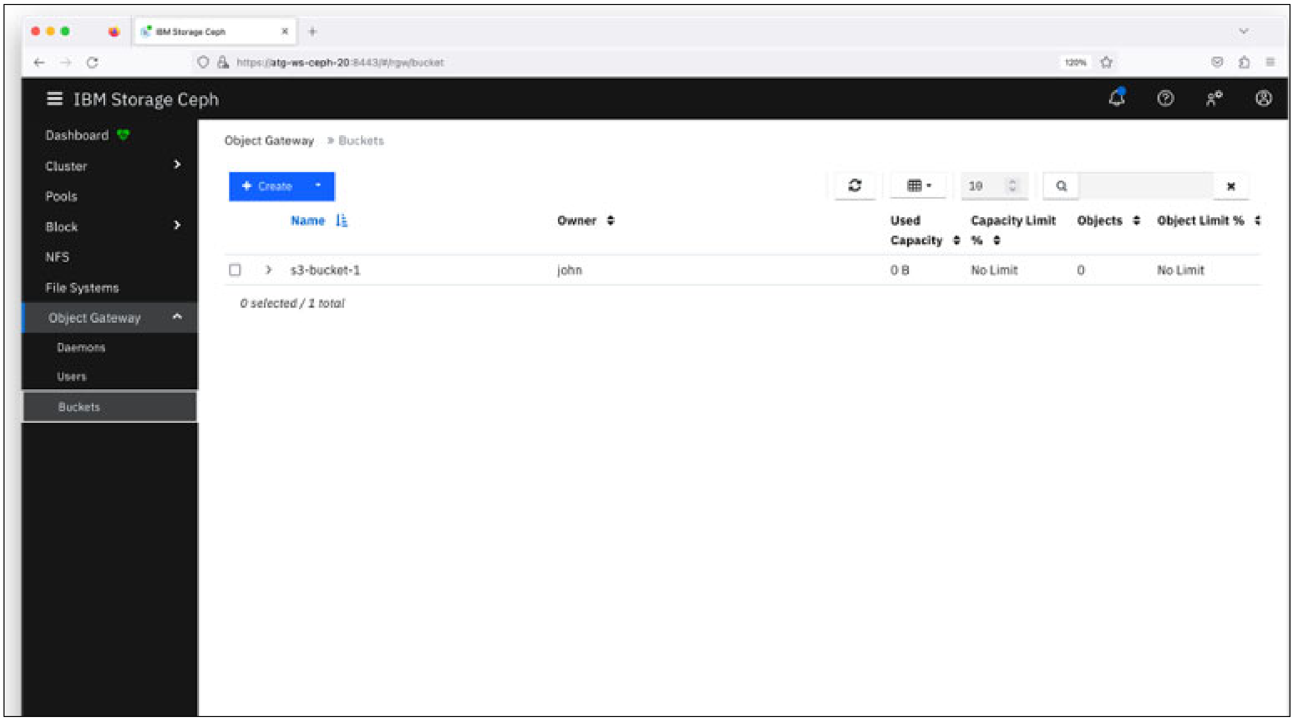


	Username	Tenant	Full name	Email address	Suspended	Max. buckets	Capacity Limit %	Object Limit %
□	dashboard		Ceph Dashboard			1000	No Limit	No Limit
□	john		Shubeck, John	jshubeck@us.ibm.con		1000	No Limit	No Limit

0 selected / 2 total

Figure 3-17 Ceph Object Gateway user listing

Figure 3-18 shows the Ceph Object Gateway bucket listing.



Name	Owner	Used Capacity	Capacity Limit %	Objects	Object Limit %
s3-bucket-1	john	0 B	No Limit	0	No Limit

0 selected / 1 total

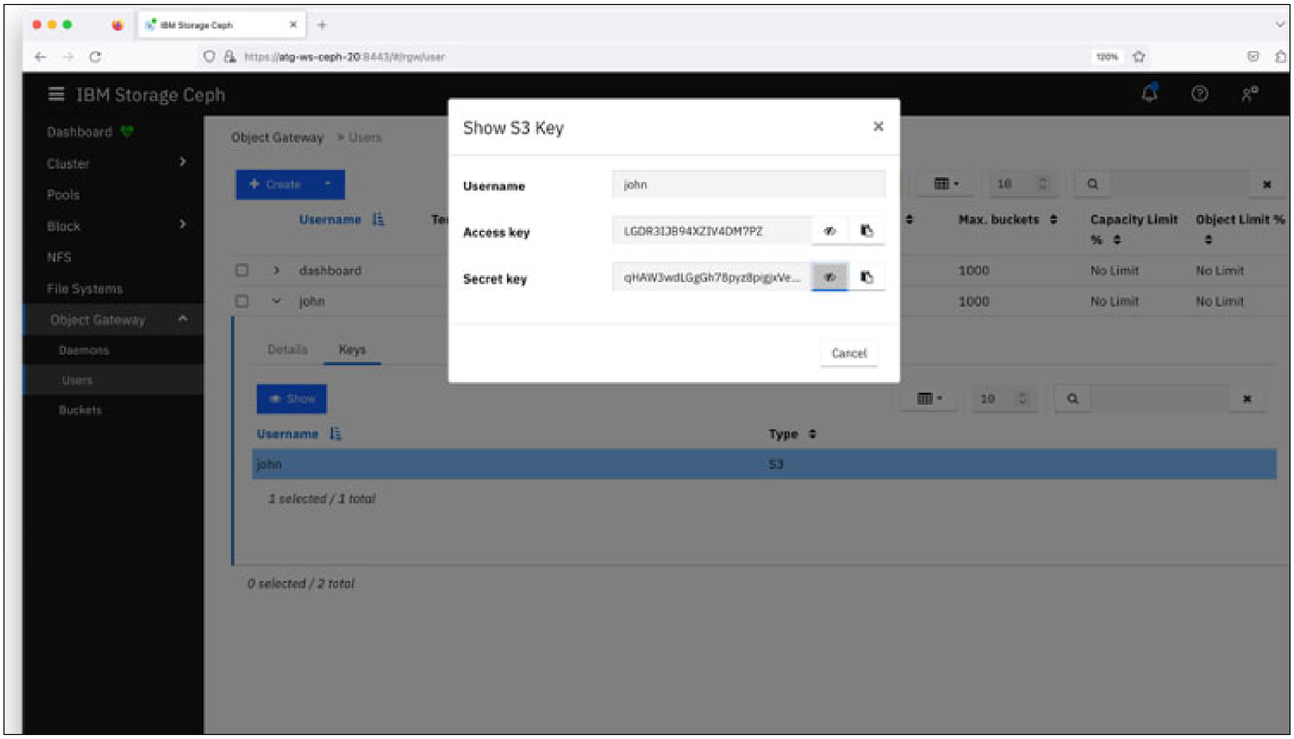
Figure 3-18 Ceph Object Gateway bucket listing

Obtaining S3 API access keys

SS3 client access requires a pair of credentials: the access key and secret access key. There are two ways in Ceph to view or generate a pair of access keys.

We show each of them below:

- From the Ceph Dashboard, select **Object Gateway** → **Users**. Select a user to display and select **Show Keys**. If multiple keys exist, select one key to show. The access key and secret key can be hidden or shown as readable (Figure 3-19).



Show S3 Key

Username: john

Access key: LGDR3IJB94XZIV4DM7PZ

Secret key: qHAW3wdLGgGh78pyz8pigjxVe...

Username	Max. buckets	Capacity Limit %	Object Limit %
dashboard	1000	No Limit	No Limit
john	1000	No Limit	No Limit

Username	Type
john	S3

1 selected / 1 total

0 selected / 2 total

Figure 3-19 Displaying the S3 access key and secret key for a selected user

- Alternatively, retrieve the S3 access keys by using the Ceph CLI. The **`radosgw-admin`** command can be run from the shell, or within the **`cephadm`** CLI (Example 3-1).

> **Note:** Substitute the RGW username that you created for “john” in “Creating an Object Gateway user” on page 48. The access key and secret key will differ and be unique for each Ceph cluster.

Example 3-1 Ceph Object Gateway S3 access key and secret key

```
[root@node1 ~]# radosgw-admin user info --uid="john"
{
    "user_id": "john",
    "display_name": "Shubeck, John",
    "email": "jshubeck@us.ibm.com",
    "suspended": 0,
    "max_buckets": 1000,
    "subusers": [],
    "keys": [
        {
            "user": "john",
            "access_key": "LGDR3IJB94XZIV4DM7PZ",
            "secret_key": "qHAW3wdLGgGh78pyz8pigjxVeoM1sz1HT6lIdYD3"
        }
    ],
. . . output omitted . . .
    },
    "user_quota": {
        "enabled": false,
        "check_on_raw": false,
        "max_size": -1,
        "max_size_kb": 0,
        "max_objects": -1
    },
    "temp_url_keys": [],
    "type": "rgw",
    "mfa_ids": []
}
```

3.2.9 Configuring the AWS CLI client

S3 client access requires a set of credentials that are called the access key and secret access key. In this example, we copy/paste the keys that we obtained in “Obtaining S3 API access keys” on page 53 to the AWS CLI client configuration tool.

There is nothing remarkable about an *object*. If a unit of data, for example, a JPEG/JFIF image, is stored in a file system, then we refer to it as a *file*. If that file is uploaded or **PUT** into an object storage bucket, we are likely to refer to it is an *object*. Regardless of where it is stored, there is nothing that changes the nature of that image. The binary data within the file or object, and the image that can be rendered from it, are unchanged.

Complete the following steps:

1. Configure the AWS CLI tool to use the client credentials. Enter the access key and secret key that the system generated for you in “Obtaining S3 API access keys” on page 53 (Example 3-2).

Note: The access key and secret key will differ and be specific to each Ceph cluster.

Example 3-2 AWS CLI configuration

```
[root@client ~]#

[root@client ~]# aws configure --profile=ceph
AWS Access Key ID [None]: LGDR3IJB94XZIV4DM7PZ
AWS Secret Access Key [None]: qHAW3wdLGgGh78pyz8pigjxVeHT6lIdYD3
Default region name [None]: <enter>
Default output format [None]: <enter>
[root@client ~]#
```

2. The AWS CLI client uses an S3 bucket as the repository to write and read objects. The action of uploading a local file to an S3 bucket is called a **PUT**, and the action of downloading an object from an S3 bucket to a local file is called a **GET**. Various S3 API clients use different commands, but at the underlying S3 API layer they always issue a **PUT** and a **GET** (Example 3-3).

Example 3-3 AWS CLI LIST buckets

```
[root@client ~]# aws --endpoint-url=http://ceph-node1:80 \
--profile=ceph s3 ls
2023-07-05 09:09:45 s3-bucket-1
[root@client ~]#
```

3. Create an S3 bucket from the AWS CLI client (optional) (Example 3-4).

> **Note:** The endpoint value should follow the local hostname of the Ceph Object Gateway daemon host.

Example 3-4 AWS CLI PUT bucket and LIST buckets

```
root@client ~]# aws --endpoint-url=http://ceph-node3:80 \
--profile=ceph s3api create-bucket --bucket s3-bucket-2

2023-07-05 09:09:45 s3-bucket-1

[root@client ~]# aws --endpoint=http://ceph-node3:80 \
--profile=ceph s3 ls
2023-07-05 09:09:45 s3-bucket-1
2023-07-05 11:47:26 s3-bucket-2
[root@client ~]#
```

4. Create a 10 MB file named `10MB.bin`. Upload the file to one of the S3 buckets (Example 3-5).

Example 3-5 AWS CLI PUT object from file

```
[root@client ~]# dd if=/dev/zero of=/tmp/10MB.bin bs=1024K count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.0115086 s, 911 MBps

[root@client ~]# aws --endpoint-url=http://ceph-node3:80 \
--profile=ceph s3 cp /tmp/10MB.bin s3://s3-bucket-1/10MB.bin
upload: ../tmp/10MB.bin to s3://s3-bucket-1/10MB.bin
```

5. Get a bucket listing to view the test object. Download the object to a local file (Example 3-6).

Example 3-6 AWS CLI LIST object and GET object to file

```
[root@client ~]# aws --endpoint-url=http://ceph-node3:80 \
--profile=ceph s3 ls s3://s3-bucket-1
2023-07-05 16:55:39   10485760 10MB.bin

[root@client ~]# aws --endpoint-url=http://ceph-node3:80 \
--profile=ceph s3 cp s3://s3-bucket-1/10MB.bin /tmp/GET-10MB.bin
download: s3://s3-bucket-1/10MB.bin to ../tmp/GET-10MB.bin
```

6. Verify the data integrity of the uploaded and downloaded files (Example 3-7).

Example 3-7 Verifying the file against the object MD5 checksum

```
[root@client ~]# diff /tmp/10MB.bin /tmp/GET-10MB.bin
[root@client ~]#
[root@client ~]# openssl dgst -md5 /tmp/10MB.bin
MD5(/tmp/10MB.bin)= f1c9645dbc14efddc7d8a322685f26eb
[root@client ~]# openssl dgst -md5 /tmp/GET-10MB.bin
MD5(/tmp/GET-10MB.bin)= f1c9645dbc14efddc7d8a322685f26eb
[root@client ~]#
```

3.2.10 The S3 API interface

IBM Storage Ceph Object Storage S3 compatibility provides object storage operations with an interface that is compatible with the Amazon S3 RESTful API.

S3 API compatibility

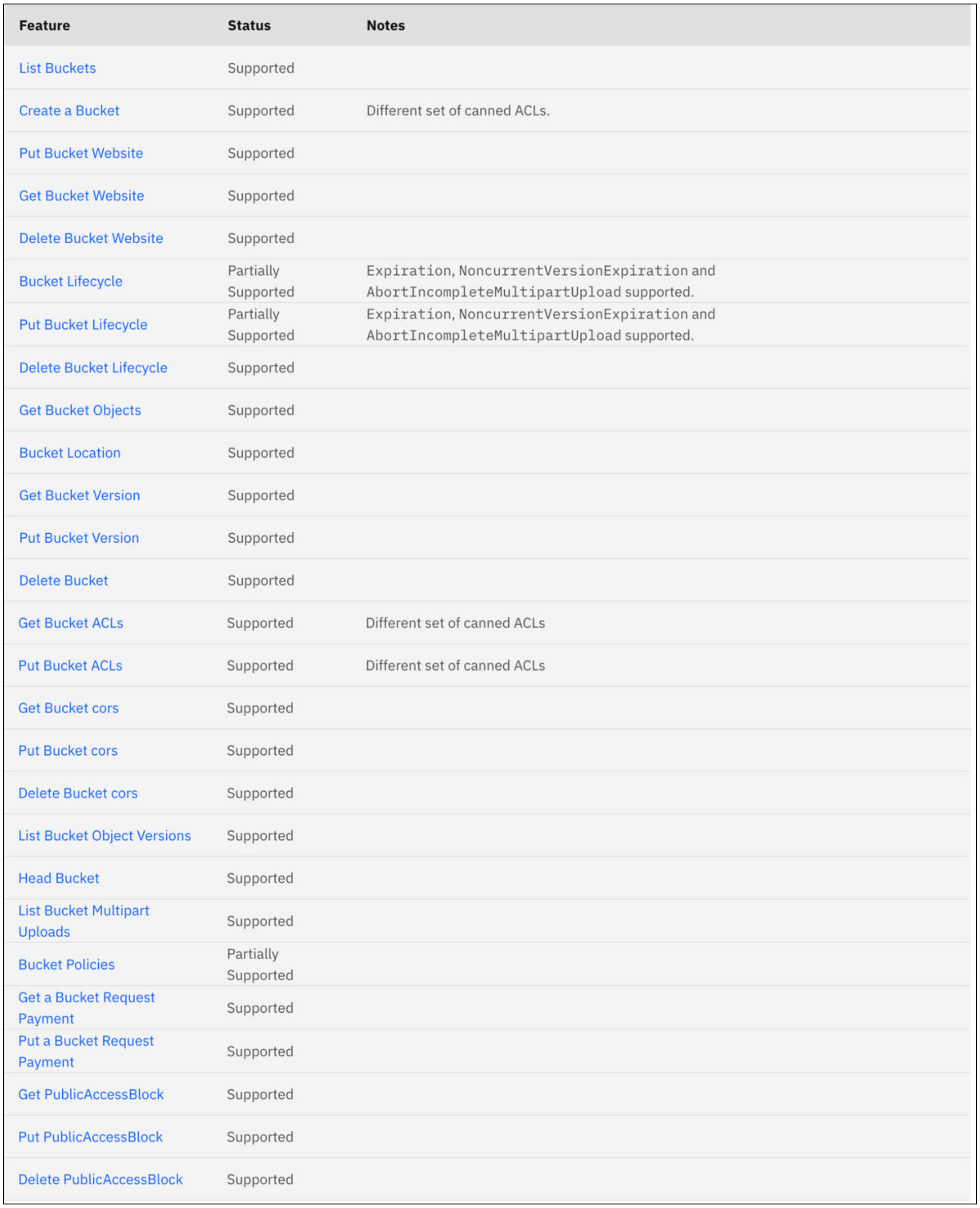
Developers can use a RESTful API that is compatible with the Amazon S3 data access model. It is through the S3 API that object storage clients and applications store, retrieve, and manage buckets and objects stored by an IBM Storage Ceph cluster. IBM Storage Ceph and the Ceph community are invested heavily in a design goal known as *S3 Fidelity*, which means clients and independent software vendors (ISVs) can realize independence and transportability for their applications across S3 vendors in a hybrid multi-cloud.

At a high level, here are the supported S3 API features in IBM Storage Ceph Object Storage:

- Basic bucket operations (**PUT**, **GET**, **LIST**, **HEAD**, and **DELETE**).
- Advanced bucket operations (Bucket policies, website, lifecycle, ACLs, and versions).
- Basic object operations (**PUT**, **GET**, **LIST**, and **POST**).
- Advanced object operations (object lock, legal hold, tagging, multipart, and retention).
- S3 select operations (CSV, JSON, and Parquet formats).
- Support for both virtual hostname and path name bucket addressing formats.

For more information, see IBM Documentation.

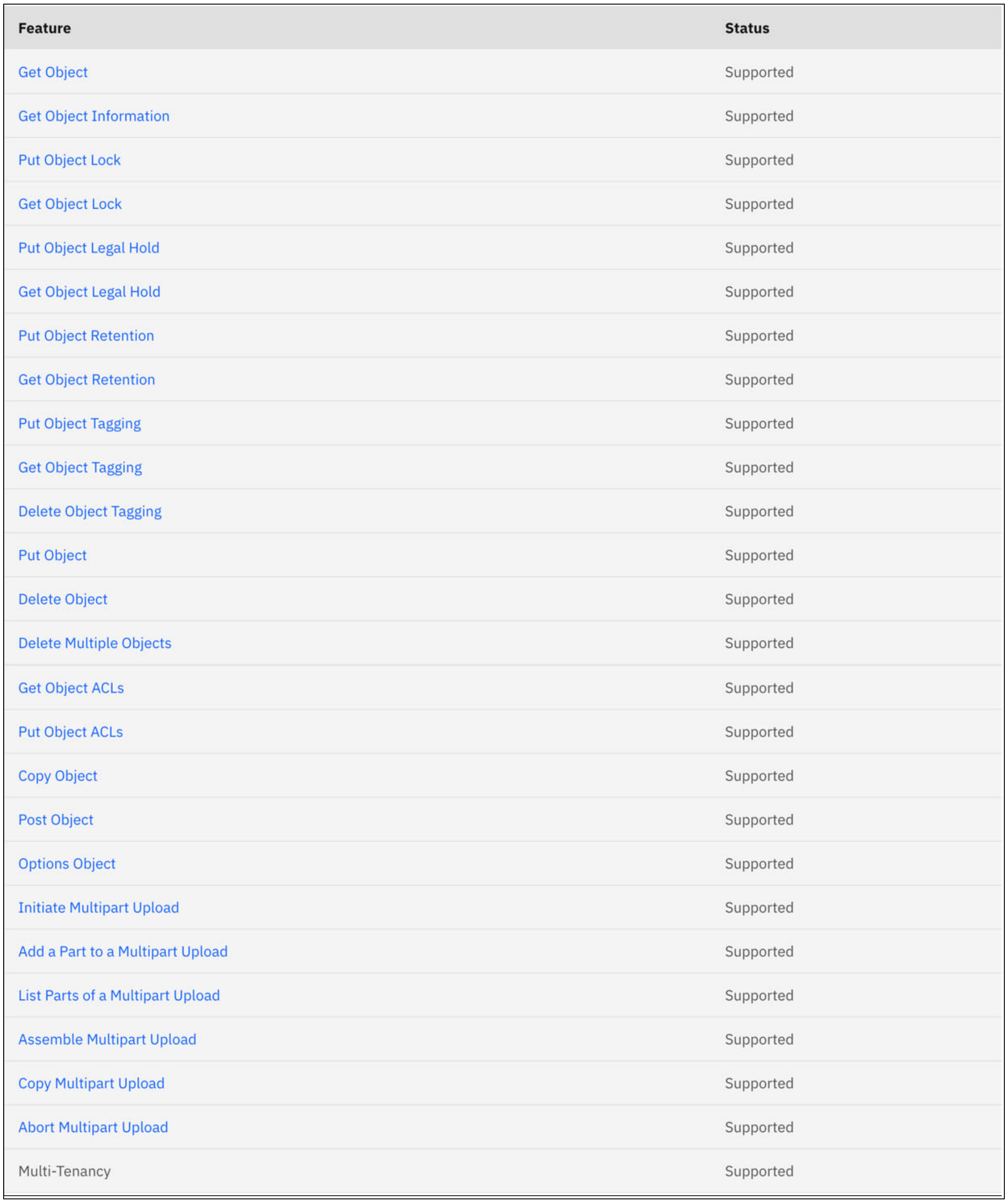
At the time of writing, Figure 3-20 shows the S3 bucket operations.



Feature	Status	Notes
List Buckets	Supported	
Create a Bucket	Supported	Different set of canned ACLs.
Put Bucket Website	Supported	
Get Bucket Website	Supported	
Delete Bucket Website	Supported	
Bucket Lifecycle	Partially Supported	`Expiration`, `NoncurrentVersionExpiration` and `AbortIncompleteMultipartUpload` supported.
Put Bucket Lifecycle	Partially Supported	`Expiration`, `NoncurrentVersionExpiration` and `AbortIncompleteMultipartUpload` supported.
Delete Bucket Lifecycle	Supported	
Get Bucket Objects	Supported	
Bucket Location	Supported	
Get Bucket Version	Supported	
Put Bucket Version	Supported	
Delete Bucket	Supported	
Get Bucket ACLs	Supported	Different set of canned ACLs
Put Bucket ACLs	Supported	Different set of canned ACLs
Get Bucket cors	Supported	
Put Bucket cors	Supported	
Delete Bucket cors	Supported	
List Bucket Object Versions	Supported	
Head Bucket	Supported	
List Bucket Multipart Uploads	Supported	
Bucket Policies	Partially Supported	
Get a Bucket Request Payment	Supported	
Put a Bucket Request Payment	Supported	
Get PublicAccessBlock	Supported	
Put PublicAccessBlock	Supported	
Delete PublicAccessBlock	Supported	

Figure 3-20 S3 API bucket operations

At the time of writing, Figure 3-21 on page 59 shows the S3 object operations.



Feature	Status
Get Object	Supported
Get Object Information	Supported
Put Object Lock	Supported
Get Object Lock	Supported
Put Object Legal Hold	Supported
Get Object Legal Hold	Supported
Put Object Retention	Supported
Get Object Retention	Supported
Put Object Tagging	Supported
Get Object Tagging	Supported
Delete Object Tagging	Supported
Put Object	Supported
Delete Object	Supported
Delete Multiple Objects	Supported
Get Object ACLs	Supported
Put Object ACLs	Supported
Copy Object	Supported
Post Object	Supported
Options Object	Supported
Initiate Multipart Upload	Supported
Add a Part to a Multipart Upload	Supported
List Parts of a Multipart Upload	Supported
Assemble Multipart Upload	Supported
Copy Multipart Upload	Supported
Abort Multipart Upload	Supported
Multi-Tenancy	Supported

Figure 3-21 S3 API object operations

3.2.11 Conclusion

IBM Storage Ceph Object storage provides a scale-out, high-capacity object store for S3 API and Swift API client operations.

The Ceph service at the core of object storage is the RGW. The Ceph Object Gateway services its clients through *S3 endpoints*, which are Ceph nodes where instances of RGW operate and service S3 API requests on well-known TCP ports through HTTP and HTTPS.

Because Ceph OSD daemons and Ceph Object Gateway daemons can be deployed independently on the same or different nodes, the cluster can effortlessly scale bandwidth and capacity across a broad range of object storage workloads and use cases.

3.2.12 References on the World Wide Web

For more information about Ceph, see the following resources:

- IBM Storage Ceph documentation
- Community Ceph documentation
- AWS CLI documentation

3.3 IBM Storage Ceph block storage

Note: This section provides a brief overview of the block storage feature in Ceph. All the following points and many others are documented in the IBM Storage Ceph documentation.

IBM Storage Ceph block storage, also commonly referred to as RBD, is a distributed block storage system that allows for the management and provisioning of block storage volumes (aka images), akin to traditional storage area networks (SANs) or direct-attach storage (DAS) but with increased flexibility and scalability.

RBD volumes can be accessed either through a kernel module (for Linux and Kubernetes) or through the librbd API (for OpenStack and Proxmox). In the Kubernetes world, RBD volumes are well suited for read/write once (RWO) persistent volume claims (PVCs). IBM Storage Ceph RBD by default employs Thin Provisioning, which means that underlying storage is allocated when client data is actually written, not when the volume itself is created. For example, one might create a 1 TiB RBD volume but initially write only 10 GiB of data to it. In this case, IBM Storage Ceph will only consume 10 GiB of physical storage media capacity (modulo replication or erasure coding for data protection)).

The RBD access architecture is shown in Figure 3-22 on page 61.

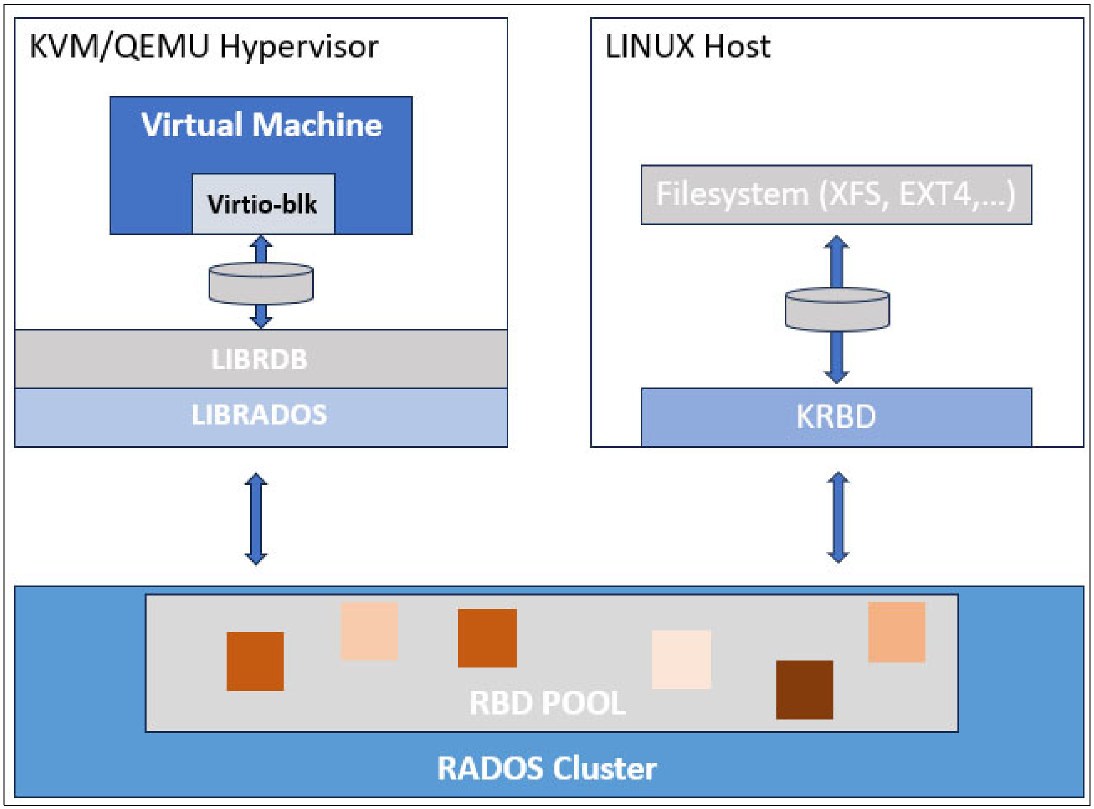


Figure 3-22 RBD access

Thhe virtual machine (VM), through the virtio-blk driver and Ceph library, accesses RBD images as though they are physical drives directly attached to a VM.

Many virtualization solutions are supported:

- Libvirt
- OpenStack (Cinder, Nova, and Glance)
- Proxmox, CloudStack, OpenNebula, and others

Librbd access method

The `librbd` access method runs in user space and can leverage all existing RBD features, including RBD mirroring. However, running in user space means that the Linux page cache cannot be used; instead, `librbd` uses its own in-memory caching, which is known as RBD caching. `librbd` acknowledges **`o_sync`** and **`o_dsync`** flags on I/O requests and flush requests that are initiated from RBD clients. Therefore, using write-back caching is as safe as using HDD caching with a VM that sends flushes (for example, Linux kernel >= 2.6.32). The cache uses a Least Recently Used (LRU) algorithm, and in write-back mode can coalesce contiguous requests for better throughput.

RBD cache is enabled by default in write-back mode on a Ceph client machine, but it can be set to write-through mode.

The following parameters can be used to configure each `librbd` client caching:

`rbd_cache`	True or false (defaults to true)
`rbd_cache_size`	Cache size in bytes (defaults to 32 MiB per RBD image)
`rbd_cache_max_dirty`	Maximum dirty bytes (defaults to 24 MiB. Set to 0 for write-through mode.)
`rbd_cache_target_dirty`	Dirty bytes to start a preemptive flush (defaults to 16 MiB)

Kernel RADOS Block Device access method

On a Linux client machine, the kernel module maps the RBD block device to a kernel block device, typically represented by the device file /dev/rbdX, where X is an integer assigned per RBD volume attachment. The kernel block device appears as a regular block device in the Linux system.The kernel RADOS Block Device (KRBD) driver may offer somewhat superior performance, but less functionality than librbd attachments. For example, it does not support RBD mirroring in journaling mode. Client KRBD feature support is expanded with successive kernel releases: a 5.4 or later kernel is required, for example, when using pg-upmap balancing within a Ceph cluster serving RBD volumes.

3.3.1 Managing RBD images

Ceph RBD images can be managed either through the Ceph Dashboard or by using the powerful **`rbd`** CLI tool. Although the Dashboard simplifies basic operations including image creation, striping settings, QoS, and namespace or trash management, the **`rbd`** command provides extensive control and flexibility for in-depth RBD image management.

The following list provides a summary of the different RBD commands:

`rbd create`	Create an RBD image.
`rbd rm`	Delete an RBD image.
`rbd ls`	List the RBD images in a pool.
`rbd info`	View the RBD image parameters.
`rbd du`	View the space that is used in an RBD image.
`rbd snap`	Create a snapshot of an RBD image.
`rbd clone`	Create a clone that is based on an RBD image snapshot.

3.3.2 Snapshots and clones

This section describes snapshots and clones.

Snapshots

RBD images, like many storage modalities, support snapshots, which are convenient for data protection, testing and development, and VM templating. RBD snapshots capture the state of an RBD image at a specific point in time by using copy-on-write (COW) technology.

IBM Storage Ceph supports up to 512 snapshots per RBD image (the number is technically unlimited, but the volume's performance is negatively affected beyond a handful).

Snapshots are read-only and track changes that are made to the original RBD image so that it is possible to access or roll back to the state of the RBD image at snapshot creation time. This means that snapshots cannot be modified. To make modifications, snapshots must be used in conjunction with clones.

Clones

Snapshots support COW clones (snapshot layering), which are new RBD images that share data with the original image or snapshot (parent). Using this feature, many writable (child) clones can be created from a snapshot, enabling rapid provisioning of new block devices that are initially identical to the source volume. For example, you might create a block device image with a Linux VM's filesystems written to it. Then, you snapshot the image, protect the snapshot, and create as many clones as you like. A snapshot is read-only, so cloning a snapshot simplifies semantics and makes it possible to create clones rapidly (Figure 3-23).

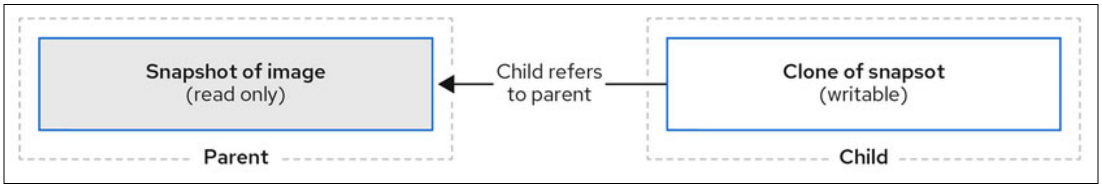


Figure 3-23 Ceph Block Device layering

Because clones rely on a parent snapshot, losing the parent snapshot causes all child clones to be lost. Therefore, the parent snapshot must be protected before creating clones (Figure 3-24).

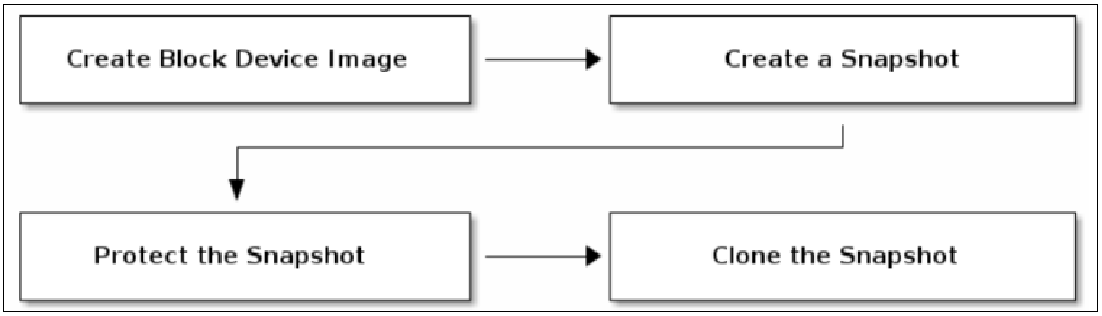


Figure 3-24 Creating a clone from a protected snapshot

Clones are new images, so they can be snapshotted, resized, or renamed. They can also be created on a separate pool for performance or cost reasons. Finally, clones are storage-efficient because only modifications made relative to the parent's data are stored on the cluster.

3.3.3 RBD write modes

RBD volumes are implemented using two types of underlying RADOS objects:

- A *header*, which is a small object containing RBD image metadata:
  - Image Name: A character string that supports Unicode.
  - Image Order: Also known as Image Object Size (defaults to 22, which corresponds to 4 Mib objects in size).
  - Image Size: The size of the block device.
  - Stripe Unit: Configures `librbd` striping (defaults to 4 MiB).
  - Stripe Count: Configures `librbd` striping (defaults to 1).
  - Format: Controls the image format (defaults to 2).
  - Image features: See “Features of RBD images” on page 66”.
- The *content* of an RBD block image is an array of blocks that are striped across RADOS objects in 4 MB chunks by default, and distributed throughout the Ceph cluster. These objects are created only when data is written, so no storage drive space is consumed until the block image is written to. Either replicated or erasure-coded pools can be used to store these objects, though replicated pools are best for non-archival use cases

Journaling mode can also be enabled to store data. In this mode, all writes are sent to a journal before being stored as an object. The journal contains all recent writes and metadata changes (device resizing, snapshots, clones, and others). Journaling mode is used for efficient RBD mirroring.

3.3.4 RBD mirroring

RBD mirroring is an asynchronous process of replicating Ceph Block Device images between two or more Ceph storage clusters. Journal-based mirroring leverages the journaling write mode. It is managed by an RBD mirroring daemon that runs on each Ceph cluster. The daemon on the mirrored Ceph cluster monitors the changes in the journal and sends them to the daemon on the mirror Ceph cluster, which replicates the changes to the copy image that is stored on that cluster.

The destination maintains an asynchronous, point-in-time, crash-consistent copy, meaning that if the mirrored cluster crashes, the destination copy lacks at most only the last few operations that were not yet replicated. The copy behaves like a clone of the original RBD image, including the data and all snapshots and clones.

RBD mirroring also supports a full lifecycle, meaning that after a failure, the synchronization can be reversed and the original site can be made the primary site again when it is back online. Mirroring can be set at the image level, so not all cluster images must be replicated.

RBD mirroring can be one-way or two-way. With one-way replication, multiple secondary sites can be configured. Figure 3-25 on page 65 shows one-way RDB mirroring.

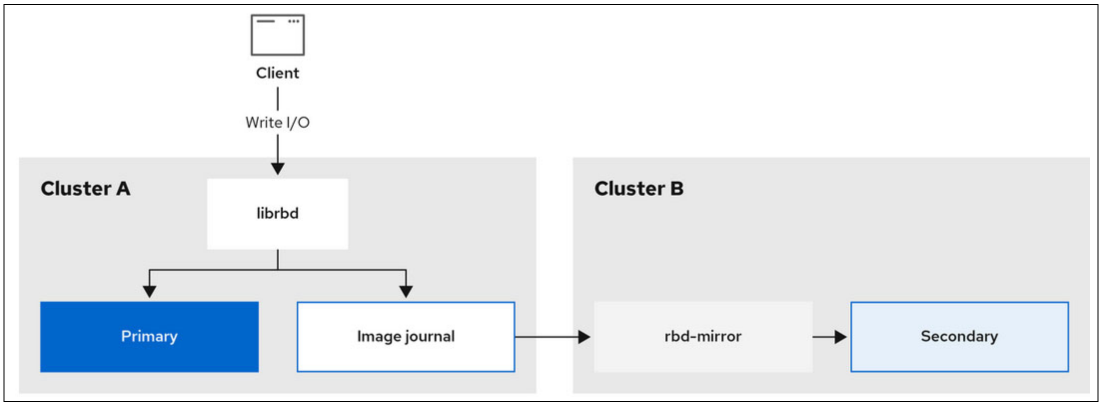


Figure 3-25 One-way RBD mirroring

Two-way mirroring limits replication to two storage clusters. Figure 3-26 shows two-way RBD mirroring.

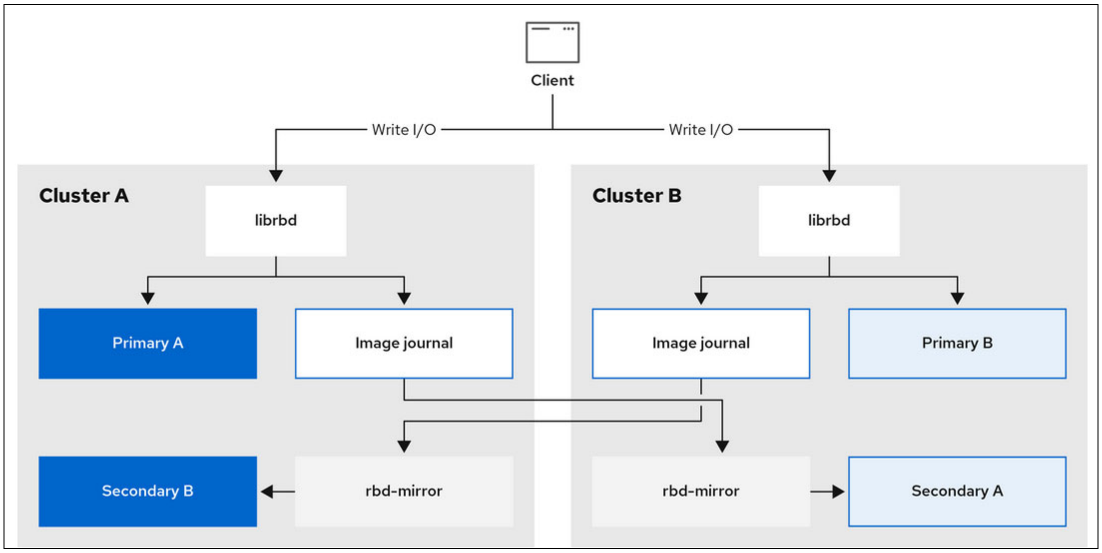


Figure 3-26 Two-way RBD mirroring

Another strategy is to use snapshot-based mirroring. With this method, the remote cluster site monitors the data and metadata differences between two snapshots before copying the changes locally. Unlike the journal-based method, the snapshot-based method must be scheduled or launched manually, so it is less incremental and current, but might also be faster.

Figure 3-27 shows snapshot RBD mirroring.

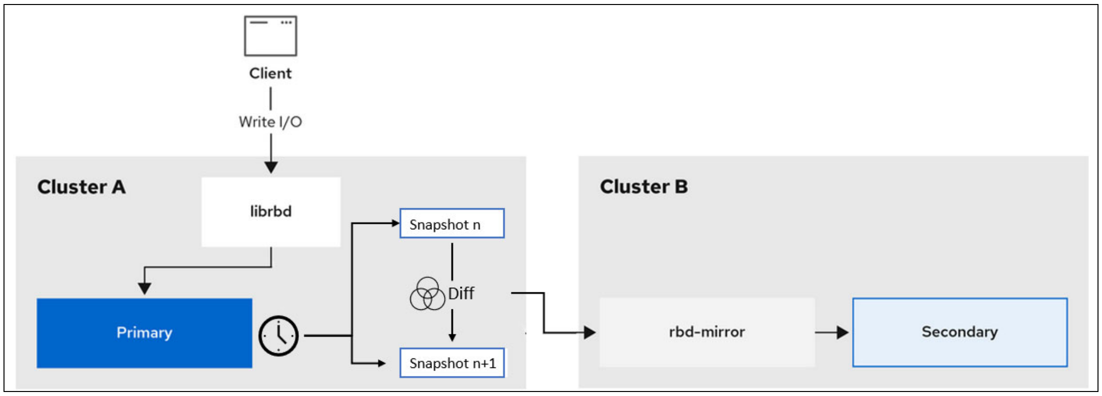


Figure 3-27 Snapshot RBD mirroring

> **Note:** For more information about mirroring, see the IBM Storage Ceph documentation.

3.3.5 Other RBD features

RBD offers many other notable and useful features.

Features of RBD images

Here are the features of RBD images:

- *Layering* refers copy on write (COW) clones of block devices.
- *Striping* spreads data across multiple RADOS objects. Striping improves parallelism for sequential workloads.
- *Exclusive locks* prevent multiple processes from accessing RBD images concurrently in an uncoordinated fashion. These locks address the write conflict situation that can occur when multiple clients try to write to the same object, which can be the case in virtualization environments or when using RBD mirroring to avoid simultaneous access to the journal.

  This feature is enabled by default on new RBD devices. It can be disabled, but other features that rely on it might be affected.

  This feature is mostly transparent to the user. When a client attempts to access an RBD device, it requests an exclusive lock on the device. An exclusive lock prevents data corruption by writers that do not co-ordinate with each other..
- *Object map support* depends exclusive lock support. RBD block volumes are thin-provisioned, meaning that they store only the data that exists. Object map support tracks which underlying RADOS objects exist (have data that is stored on a physical drive). Enabling object map support speeds up I/O operations for cloning, importing, and exporting sparsely populated images, and deleting.
- *Fast-diff support* depends on object map support and exclusive lock support. It adds a property to the object map, making it much faster to generate differentials between snapshots of an image and determine the data usage of volumes and their snapshots.

- *Deep-flatten* enables RBD flattening to work on all snapshots of an image, in addition to the image itself. Without deep-flatten, snapshots of an image rely on the parent, so the parent cannot be deleted until the snapshots are deleted. Deep-flatten makes a parent independent of its clones, even if they have snapshots, by copying those portions of data that is common between parent and children. Note that this process necessarily consumes additional underlying storage capacity.
- *Journaling support* depends on exclusive lock support. Journaling records data modifications to facilitate replication of a crash-consistent image to a remote cluster.

Encryption

Using the Linux Unified Key Setup (LUKS) 1 or 2 format, RBD images can be encrypted if they are used with `librbd` (at the time of writing, KRBD is not supported). The format operation persists the encryption metadata to the RBD image. The encryption key is secured by a passphrase that is provided by the user to create and access the device when it is encrypted. Data at rest, which means the data that Ceph writes to physical storage drives, may alternately be encrypted at the underlying RADOS layer by specifying the dmcrypt option when OSDs are deployed.

Note: For more information about encryption, see the IBM Storage Ceph documentation.

Quality of service

By using `librbd`, it is possible to limit per-image I/O by using parameters, which are disabled by default, that operate independently of each other. For example write input/output per second (IOPS) can be limited while read IOPS is not. Throughput and IOPS may be similarly throttled independently from each other. Here are these parameters:

`IOPS`	The number of IOPS (read + write)
`read IOPS`	The number of read IOPS
`write IOPS`	The number of write IOPS
`bps`	The bytes per second (read + write)
`read bps`	The bytes per second read
`write bps`	The bytes per second written

These settings can be configured at the image creation time or any time after by using either the **`rbd`** CLI tool or the Dashboard.

Namespace isolation

Namespace isolation can restrict client access to private namespaces by using authentication keys. In a private namespace, clients can see only their own RBD images and cannot access the images of other clients in the same RADOS pool but in a different namespace.

You can create namespaces and configure images at image creation by using the **`rbd`** CLI tool or the Dashboard.

Live migration of images

You can live-migrate RBD images between different pools, or even within the same pool, on the same storage cluster. You can also migrate between different image formats and layouts, and even from external data sources.

When live migration is initiated, the source image is deep-copied to the destination image, preserving all snapshot history and sparse allocation of data where possible. Live migration requires creating a target image that maintains read/write access to data while the linked source image is marked read-only. When background migration completes, you then commit the migration, removing the cross-link and deleting the source (unless the import-only mode is used). You can also cancel the migration, removing the cross-link and deleting the target image.

Importing and exporting images

Using the **rbd export** and **rbd import** commands, it is possible to copy images from a cluster to another through an SSH connection. Import and export operations can also leverage snapshots by using the **export-diff**, **import-diff**, and **merge-diff** commands. The **export-diff** command generates a new snapshot file containing the changes between two snapshots. This snapshot file can be used with the **import-diff** command to apply the changes to an image on a remote Ceph cluster, or with the **merge-diff** command to merge two continuous snapshots.

Trash

When you move an RBD image to trash, you may keep it for a specified time before it is permanently deleted and the underlying storage capacity freed. If the retention time has not been exceeded and the trash has not been purged, a trashed image can be restored.

Trash management is available from both the CLI tool and the Dashboard.

The rbdmap service

The systemd unit file, `rbdmap.service`, is installed by the `ceph-common` package. The `rbdmap.service` unit runs the **rbdmap** shell script.

This script is useful to automatically mount at boot time and umount at shutdown RBD images from a Ceph client by adding one path per line to RBD devices (`/dev/rbdX`) and the associated credentials to manage.

RBD Network Block Device

RBD Network Block Device (RBD-NBD) is an alternative to KRBD for mapping Ceph RBD images. Unlike KRBD, which works in kernel space, RBD-NBD relies on the NBD kernel module and works in user space. This situation enables access to `librbd` features such as exclusive lock and **fast-diff**. NBD exposes mapped devices as local devices in paths like `/dev/nbdX`.

In summary, RBD image features make them well suited for various workloads, including both virtualization (VMs and containers) and high-performance applications (such as databases and applications that require high IOPS and use small I/O size). RBD images provide capabilities including snapshots and layering, and stripe data across multiple servers in the cluster to improve performance and availability.

3.4 IBM Storage Ceph file storage

IBM Storage Ceph provides a distributed network file system that is compatible with POSIX standards. Because data within Ceph is internally stored as objects in RADOS, an additional component is required to maintain ACLs, ownership, and permissions for files and directories while mapping POSIX inode numbers to RADOS object names. This specific CephFS component is the *Metadata Server* (MDS).

Although the shared file system feature was the original use case for IBM Storage Ceph, due to the demand for block and object storage by cloud providers and companies implementing

OpenStack, this feature was put on the back burner. CephFS was the last Ceph core feature to become generally available, preceded by virtual block devices (RBD) and the OpenStack Swift and Amazon S3 gateway (RGW). The first line of code of a prototype file system was written in 2004 during Sage Weil's internship at Lawrence Livermore National Laboratory. Sage continued working on the project during another summer project at the University of California Santa Cruz in 2005 to create a fully functional file system (Ceph), and presented it at Super Compute and USENIX in 2006.

3.4.1 Metadata Server

CephFS stores both data and metadata (inodes and dentries) for the file system in RADOS pools. Using RADOS as a storage layer enables Ceph to leverage built-in RADOS object features such as watch and notify to easily implement an active-active or active-passive mechanism and a highly efficient online file system check mechanism.

The MDS also maintains a cache to improve metadata access performance while managing the cache of CephFS clients to ensure that the clients have the proper cache data and to prevent deadlocks on metadata access.

An MDS can have two roles:

- Active
- Standby

Active Metadata Server

An active MDS responds to CephFS client requests to provide the client with the exact RADOS object name and its metadata for an i-node while maintaining a cache of all accesses to the metadata.

When a new inode or dentry is created, updated, or deleted, the cache is updated and the inode or dentry is recorded, updated, or removed in a RADOS pool that is dedicated to CephFS metadata.

Standby Metadata Server

A standby MDS is assigned a rank if it enters the `Failed` state. When the standby MDS becomes active, it replays the journal to reach a consistent state.

If the active MDS becomes inactive, a standby MDS becomes the active MDS.

An MDS can be marked as standby-replay for one or more ranks so that journal changes are continuously applied to its cache. This feature enables faster failover between two MDS daemons. To use this feature, each CephFS file system rank must be assigned to a standby-replay MDS.

Metadata Server journaling

Every action on CephFS metadata is streamed into a journal before the file system operation is committed. This journal is shared among MDSs and located in a dedicated metadata RADOS pool.

This journal is used to maintain file system consistency during an MDS failover operation. Events can be replayed by the standby MDS to reach a file system state that is consistent with the state last reached by the now defunct, previously active MDS.

The benefit of using a journal to record the changes is that most journal updates are sequential. They are handled more quickly by most physical storage drives, and sequential consecutive writes can be coalesced for even better performance.

The performance and availability of metadata is of vital importance to CephFS performance, which is why it is a best practice to use SSD OSDs to host the placement groups (PGs) of the metadata pool. The metadata pool must employ the replication data protection strategy: erasure coding cannot be used for metadata. It is also recommended to configure the metadata pool to maintain four replicas.

The journal comprises multiple RADOS objects in the metadata pool, and journals are striped across multiple objects for performance. Each active MDS maintains its own journal for performance and resiliency reasons. Old journal entries are automatically trimmed by the active MDS.

3.4.2 Ceph File System configuration

Ceph supports one or more file systems hosted by a common storage cluster. Administrators can adapt shared file system capabilities to multiple use cases. Creating multiple file systems is aligned with CRUSH customization, mapping specific hardware capabilities or layout:

- Lightning-fast file system
- Fast file system
- Home directory file system
- Archival file system

The file system configuration can be paired with an MDS deployment and configuration to serve each file system with the required level of metadata performance:

- A hot directory requires a dedicated set of MDSs.
- An archival file system is read-only, and metadata updates occur in large batches.
- Increasing the number of MDSs can remediate a metadata access bottleneck.

Each file system is assigned a parameter named **max_mds** that indicates the maximum number of MDSs that can be active. By default, this parameter is set to 1 for each file system that is created in the Ceph storage cluster.

A best practice when creating file systems is to deploy **max_mds+1** MDSs for each file system, with one MDS in tje standby role to ensure file system accessibility and resilience.

The number of active MDSs can be dynamically increased or decreased for a file system without client disruption.

Figure 3-28 on page 71 provides a visual representation of a multi-MDS configuration within a Ceph cluster. The shaded nodes represent directories, and the unshaded nodes represent files. The MDSs at the bottom of the picture illustrate which specific MDS rank is in charge of a specific subtree.

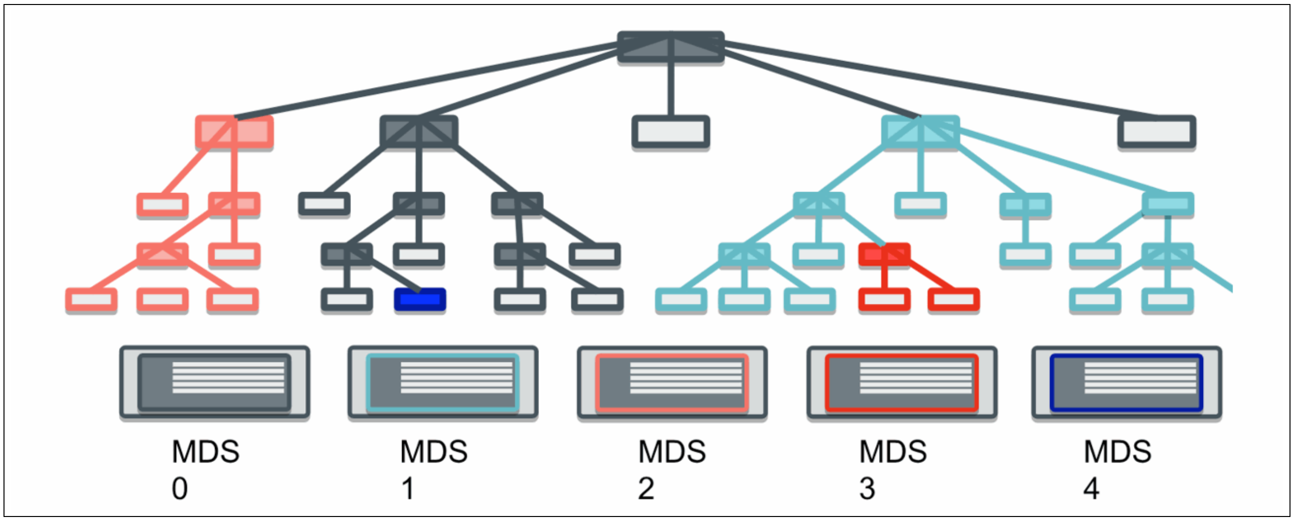


Figure 3-28 CephFS MFS subtree partitioning

Metadata Server rank

Each active MDS is assigned a file system rank. This rank is used when creating subtree partitioning policies or when pinning the directory to a specific MDS. The first rank that is assigned to an active MDS for a file system is `0`.

Each rank has a state:

`Up`	The rank is assigned to an MDS,.
`Failed`	The rank is not assigned to any MDS.
`Damaged`	`The` metadata for the rank is damaged or corrupted.

> **Note:** A Damaged rank will not be assigned to any MDS until it is fixed and after the Ceph administrator issues a **`ceph mds repaired`** command.

Regarding the number of MDSs that are assigned to a file system, the Ceph Manager (MGR) provides the MDS Autoscaler module that monitors the **`max_mds`** and the **`standby_count_wanted`** parameters for a file system.

Metadata Server caching

To control the performance of the MDS, the Ceph administrator can modify the amount of memory that is used by an MDS for caching or the number of inodes that are maintained in cache by an MDS:

`mds_cache_memory_limit`	Controls the amount of memory that is used (8 - 64 GiB).
`mds_cache_size`	Controls the inode count in the cache (disabled by default).

> **Note:** As a best practice, use the **`mds_cache_memory_limit`** parameter.

Because a client may malfunction and cause some metadata to be held in the cache longer than expected, the Ceph cluster has a built-in warning mechanism, which uses the **`mds_health_cache_threshold`** parameter. A warning is raised when the actual cache usage is 150% of the **`mds_cache_memory_limit`** parameter.

Metadata Server file system affinity

If your Ceph cluster has heterogeneous hardware that includes fast and slow CPUs and/or storage drives, you can assign file system affinity to favor specific MDSs running on specific hardware according to your individual needs. Very active MDS instances, for example, benefit from a server CPU with a high clock rate (frequency), while an MDS for an archival file system may do fine on a server with a modest CPU. Note that uniquely among Ceph components, an MDS daemon does not benefit from a large number of CPU cores. Thus it is a common strategy to place high-traffic MDS daemons on nodes with high clocks but as few as 4 cores.

This affinity is achieved by assigning the mds_join_fs parameter for a specific MDS instance. When a rank enters the Failed state, the Monitors (mons) in the cluster assign to the Failed rank a standby MDS for which the mds_join_fs is set to the name of the associated file system.

If no standby MDS has this parameter set, a non-dedicated standby MDS is assigned to the Failed rank.

Dynamic tree partitioning

When multiple active MDSs are assigned to a file system, the workload is automatically distributed among them to maximize file system performance.

Ephemeral pinning

Dynamic tree partitioning of directory subtree can be configured by using policies assigned to directories within a file system, which influences the distribution of the MDS workload. The following extended attributes of a file system directory can be used to control the balancing strategy across active MDSs:

`ceph.dir.pin.distributed`	All children will be pinned to a rank.
`ceph.dir.pin.random`	Percentage of children that will be pinned to a rank.

Ephemera pinning does not persist after the inode is dropped from the MDS cache.

Manual pinning

If necessary, when Dynamic Tree Partitioning is not sufficient, with or without subtree partitioning policies (for example, one directory is hot), the Ceph administrator can pin a directory to a particular MDS. This goal is achieved by setting an extended attribute (`ceph.dir.pin`) of a specific directory to indicate which MDS rank oversees metadata requests for it.

Pinning a directory to a specific MDS rank does not dedicate that rank to this directory because Dynamic Tree Partitioning never stops between active MDSs.

3.4.3 Ceph File System layout

This section describes the CephFS data layout.

RADOS pools

Each file system requires a RADOS metadata pool to store metadata the and the file system journal, and at least one data pool to store file data.

The overall CephFS architecture is shown in Figure 3-29 on page 73.

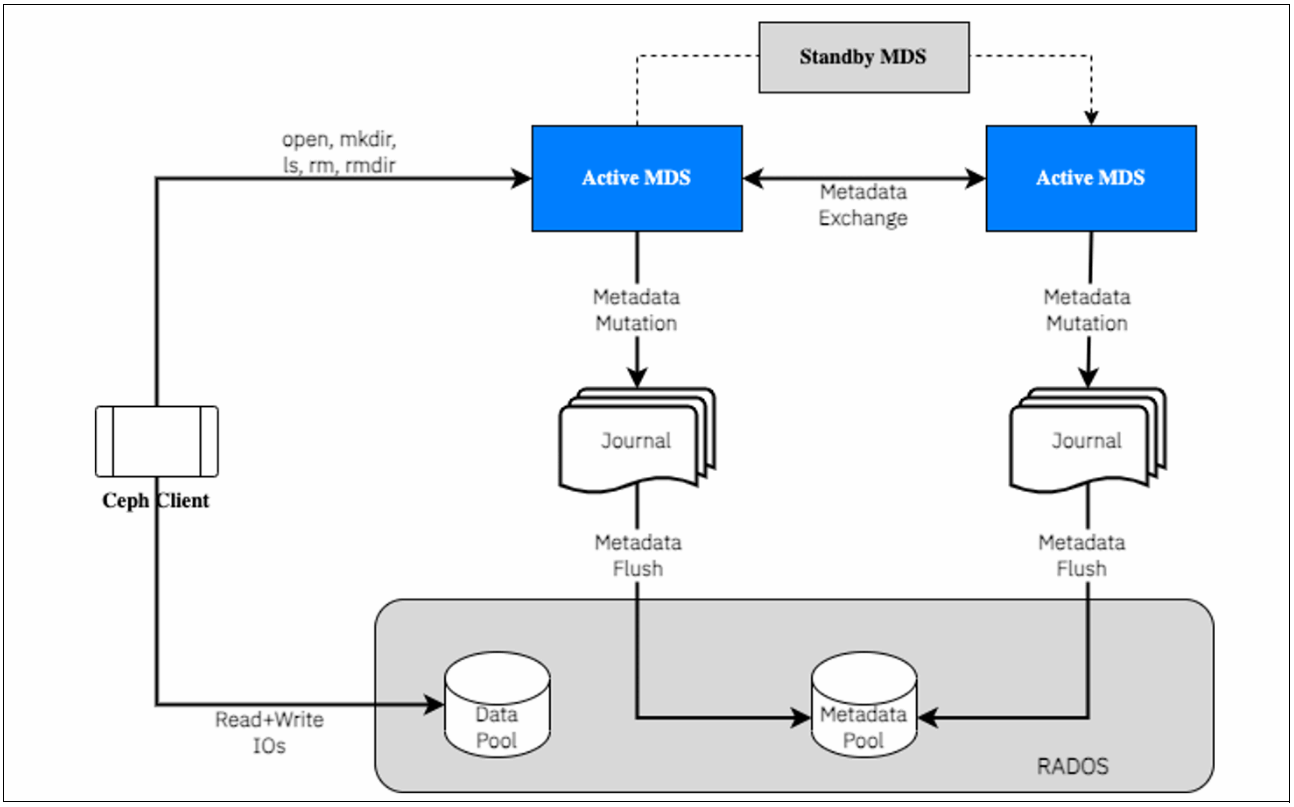


Figure 3-29 Ceph File System pools and components

Data layout

This section describes metadata and data pools.

Metadata pool

The journaling mechanism uses dedicated RADOS objects, and the journal is striped across multiple RADOS objects for performance.

Each inode in the file system is stored in a set of RADOS objects that are named as follows:

```
{inode_number}.{inode_extent}
```

The `{inode_extent}` starts as `00000000` and is incremented as additional data extents are added.

Datapool

The organization of the data pool is driven by a set of extended attributes assigned to files and directories. By default, a CephFS file system is created with one metadata pool and one data pool. An extra data pool can be attached to an existing file system through the set of extended attributes that is managed by the MDS. To add an additional data pool, run a command of the following form:

```
$ ceph fs add_data_pool {filesysten_name} {data_pool_name}
```

The placement and the organization data follow the following rules:

- A file, when created, inherits the attributes of the parent directory.
- A file attribute can be changed only if it contains no data.
- Existing files are affected by attribute changes on the parent directory.

Table 3-1 shows the directory attributes.

Table 3-1 Directory attributes

Name	Controls
`ceph.dir.layout.pool`	The pool to which data is written.
`ceph.dir.layout.stripe_unit`	The size of a stripe unit.
`ceph.dir.layout.stripe_count`	The number of stripe units in a full stripe.
`ceph.dir.layout.object_size`	The RADOS object size to use when striping.
`ceph.dir.layout.pool_namespace`	The RADOS namespace to use.

The default values for these parameters are:

`ceph.dir.layout.pool`	CephFS filesystem datapool
`ceph.dir.layout.stripe.unit`	`1` MiB
`ceph.dir.layout.stripe.count`	`1`
`ceph.dir.layout.object.size`	`4` MiB
`ceph.dir.layout.pool.namespace`	Default namespace name=""

All attributes at the file level have similar names prefixed with `ceph.file.layout`.

These attributes can be displayed by using the following commands when the file system is mounted:

- **`getfattr -n ceph.dir.layout {directory_path}`**
- **`getfattr -n ceph.file.layout {file_path}`**

These attributes can be set by using the following commands when the file system is mounted:

- **`setfattr -n ceph.dir.layout.attribute -v {value} {dir_path}`**
- **`setfattr -n ceph.file.layout.attribute -v {value} {file_path}`**

For example, let us visualize the RADOS physical layout of a CephFS file with the following attributes, set explicitly or inherited from its parent directory (Figure 3-30 on page 75):

- The file size is 8 MiB.
- The object size is 4 MiB.
- The stripe unit is 1 MiB.
- The stripe count is 4.

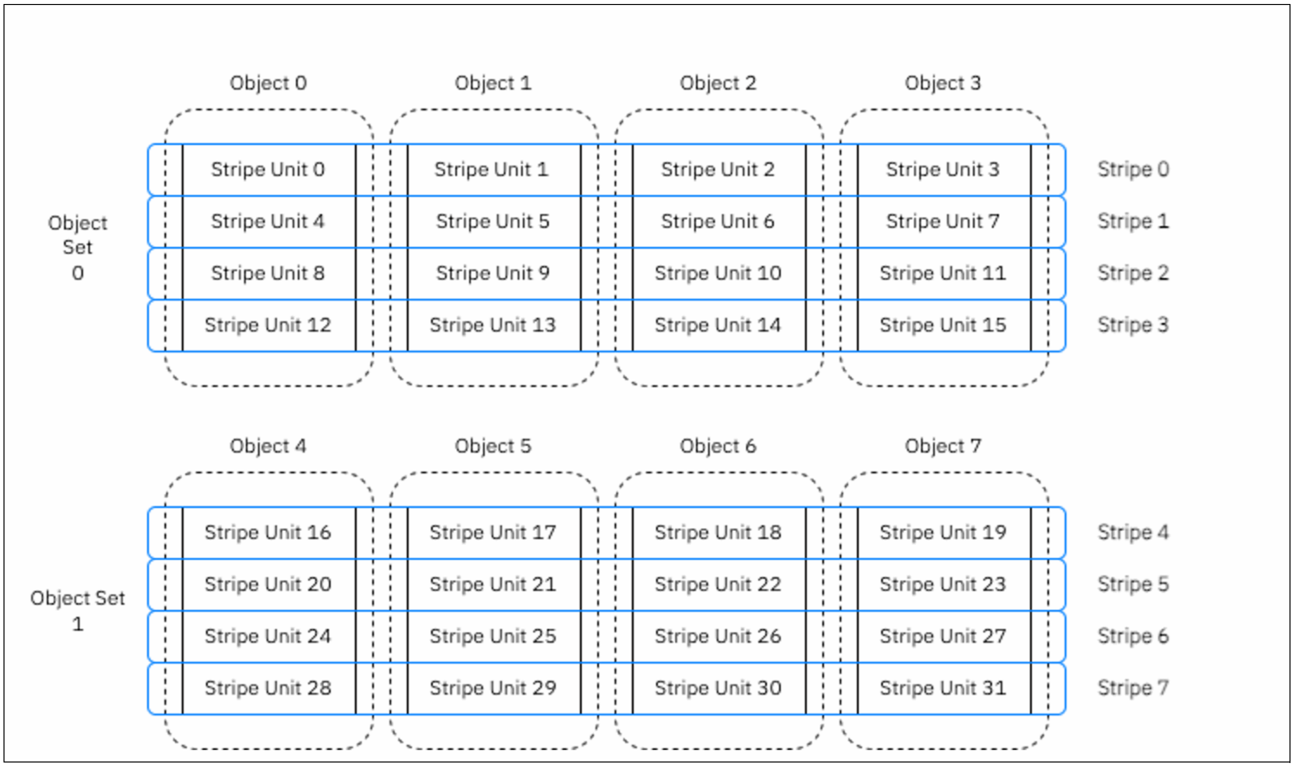


Figure 3-30 Ceph File System file data layout across RADOS objects

3.4.4 Data layout in action

As an example, we create the structure that is shown in Example 3-8 on an empty CephFS file system.

Example 3-8 Creating the following structure on an empty file system

```
/mnt/myfs/testdir
/mnt/myfs/emptyfile (100MB)
/mnt/myfs/testdir/emptyfileindir (100MB)
```

We can display the names of the RADOS objects that store the file's data (Example 3-9).

Example 3-9 RADOS objects that store file data

```
$ mount.ceph 10.0.1.100:/ /mnt/myfs -o name=admin
$ df | grep myfs
10.0.1.100:/        181141504  204800 180936704   1% /mnt/myf
$ mkdir /mnt/myfs/testdir
$ dd if=/dev/zero of=/mnt/myfs/emptyfile bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.194878 s, 538 MBps
$ dd if=/dev/zero of=/mnt/myfs/testdir/emptyfileindir bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.0782966 s, 1.3 GBps
$ rados -p cephfs_data ls | cut -f1 -d. | sort -u
10000000001
```

```
100000001f6
$ for inode in $(rados -p cephfs_data ls | cut -f1 -d. | sort -u); do echo
"Processing INODE=${inode}";echo "----------------------------";rados -p
cephfs_data ls | grep $inode; done
Processing INODE=10000000001
----------------------------
10000000001.00000009
10000000001.0000000f
10000000001.00000012
10000000001.00000018
10000000001.00000015
10000000001.00000003
10000000001.0000000e
10000000001.00000002
10000000001.00000004
10000000001.0000000b
10000000001.00000010
10000000001.00000008
10000000001.0000000a
10000000001.00000005
10000000001.00000007
10000000001.00000014
10000000001.00000006
10000000001.00000000
10000000001.00000013
10000000001.0000000d
10000000001.00000017
10000000001.00000001
10000000001.00000011
10000000001.0000000c
10000000001.00000016
Processing INODE=100000001f6
----------------------------
100000001f6.00000017
100000001f6.00000018
100000001f6.00000016
100000001f6.0000000e
100000001f6.0000000d
100000001f6.0000000b
100000001f6.0000000c
100000001f6.00000004
100000001f6.00000015
100000001f6.00000005
100000001f6.00000001
100000001f6.00000007
100000001f6.00000012
100000001f6.00000000
100000001f6.00000003
100000001f6.00000008
100000001f6.00000006
100000001f6.00000013
100000001f6.00000011
100000001f6.00000009
100000001f6.0000000f
100000001f6.00000014
```

```
100000001f6.0000000a
100000001f6.00000002
100000001f6.00000010
```

Two inodes were created (10000000001 and 100000001f6), and each one counts as 25 objects. Because the file system is empty and not pre-configured or customized, we use `stripe_unit=4 MiB`, `stripe_count=1`, and `object_size=4 MiB`, that is, 25*4=100 MiB.

Example 3-10 Two inodes are created

```
$ for inode in $(rados -p cephfs_data ls | cut -f1 -d. | sort -u); do echo
"Processing INODE=${inode}";echo "----------------------------";echo "Found
$(rados -p cephfs_data ls | grep $inode | wc -l) RADOS objects"; done
Processing INODE=10000000001
----------------------------
Found 25 RADOS objects
Processing INODE=100000001f6
----------------------------
Found 25 RADOS objects
```

How do we know which file is which inode number? To answer this question, see Example 3-11.

Example 3-11 Running the printf command to map files to inodes

```
$ printf '%x\n' $(stat -c %i mnt/myfs/emptyfile)
10000000001
$ printf '%x\n' $(stat -c %i /mnt/myfs/testdir/emptyfileindir)
100000001f6
```

Volumes and subvolumes

The IBM Storage Ceph 4.x (Nautilus) release added the concepts of volumes and subvolumes.

Volumes are used to manage exports through a Ceph Manager module that provides shared file system capabilities via OpenStack Manila and Ceph CSI in Kubernetes / Red Hat OpenShift environments:

- Volumes represent an abstraction for CephFS filesystems.
- Subvolumes represent an abstraction for directory trees.
- Subvolume groups aggregate subvolumes to apply specific common policies across multiple subvolumes.

This feature streamlines creating a CephFS file system with a single command that automatically creates the underlying pools, then deploys the MDSs. Here is the command:

```
ceph fs volume create {filesystem_name} [options]
```

Quotas

IBM Storage Ceph CephFS supports quotas to restrict the number of bytes or the number of files within a directory. This quota mechanism is enforced by both Linux kernel and userspace FUSE clients.

Quotas are managed through the extended attributes of a directory, and can be set by running the **`setfattr`** command for a specific directory:

- **`ceph.quota.max_bytes`**
- **`ceph.quota.max_files`**

> **Note:** These commands are managed directly by the OpenStack Ceph Manila driver and the Ceph CSI driver at the subvolume level.

3.4.5 Ceph File System clients

The CephFS kernel client was merged into the Linux 2.6.34 kernel in October 2010. It was the first Ceph kernel client, and remains at the time of writing the most performant way to access a CephFS file system.

The Ceph FUSE client was subsequently created to allow non-Linux clients to access a CephFS filesystem (Figure 3-31).

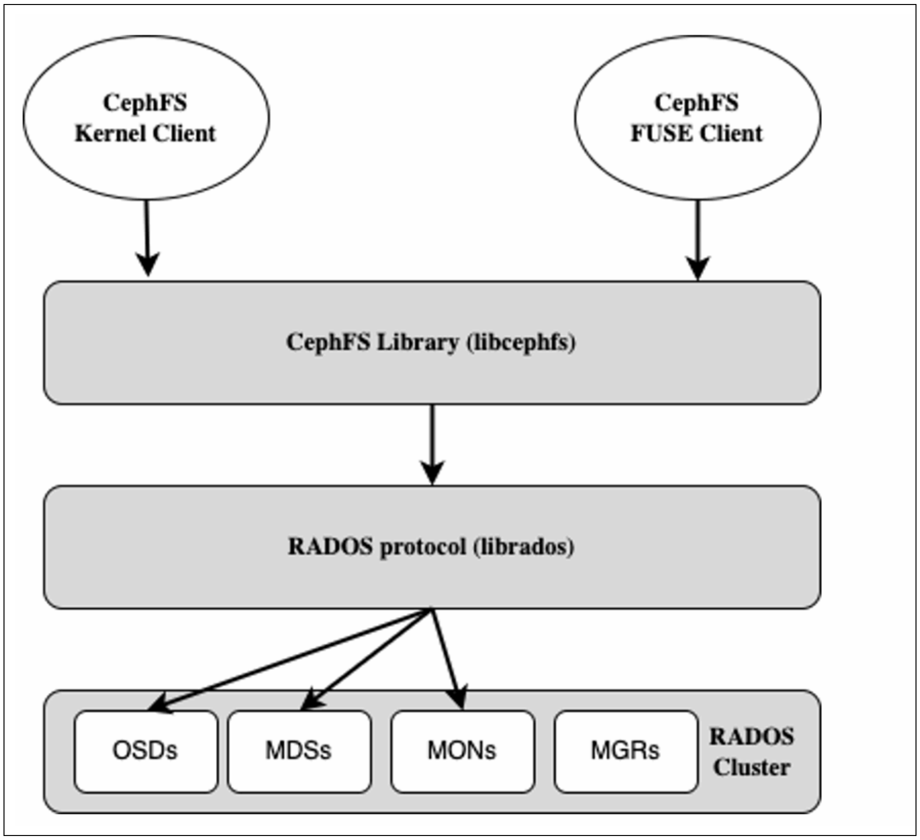


Figure 3-31 CephFS File System client stack

3.4.6 File System NFS Gateway

IBM Storage Ceph 5 added the ability to export a CephFS directory through NFS by using a gateway that leverages the Ganesha open-source project. This feature is based on FUSE.

Ganesha supports multiple protocols including NFS v3 and NFS v4 and does so through the File System Abstraction Layer (FSAL).

IBM Storage Ceph supports NFS v3, and NFS v4 with Version 6.x and later.

The NFS implementation requires enabling the Ceph Manager nfs module . This can be accomplished by running the following command:

```
ceph mgr module enable nfs
```

3.4.7 Ceph File System permissions

CephFS manages granular capabilities that control which cluster components and clients are allowed to perform specific operations. These Ceph capabilities are designed to control permissions when accessing a CephFS file system. For more information about these capabilities (client ID, client name, CephX, and others), see Chapter 2, “IBM Storage Ceph architecture” on page 13.

You can assign specific permissions to a directory for a specific Ceph client username or user ID by running the following command:

```
$ ceph fs authorize {filesystem_name} {client_id|client_name} {path} {permissions}
```

The `{client_name}` is `client.admin`, for example.

The {permissions} can be one or more of the following:

- **r** for read permission
- **w** for write permission
- **p** to let the user manipulate layout and quota attributes
- **s** to let the user manipulate snapshots

For example, imagine a Ceph client CephX definition like Example 3-12.

Example 3-12 Ceph client CephX definition

```
# ceph auth get client.0
[client.0]
   key = AQA+KrxjWUovChAACWGjOYUbUEZHSKmNtYxriw==
   caps mds = "allow rw fsname=myfs"
   caps mon = "allow r fsname=myfs"
   caps osd = "allow rw tag cephfs data=myfs"
```

If we try to modify an attribute of the root directory (/) of a CephFS file system mounted at `/mnt` mount point, the request is denied because the set of capabilities do not include **'p'**:

```
# setfattr -n ceph.dir.layout.stripe_count -v 2 /mnt
setfattr: /mnt: Permission denied
```

If we create another Ceph client user with the appropriate capabilities, then remount the CephFS on the same /mnt mount point, the request succeeds (Example 3-13).

Example 3-13 Creating another user with the correct capabilities and remounting the CephFS file system

```
# mkdir /mnt/dir4
# ceph fs authorize myfs client.4 / rw /dir4 rwp
[client.4]
   key = AQBmK71jOFcKERAAJqwhXOHoucR+iYOnzGV9BQ==
# umount /mnt
# mount -t ceph ceph-node01.example.com,ceph-node02.example.com:/ /mnt -o
name=4,secret="AQBmK71jOFcKERAAJqwhXOHoucR+iYOnzGV9BQ=="
# touch /mnt/dir4/file1
# setfattr -n ceph.file.layout.stripe_count -v 2 /mnt/dir4/file1
# getfattr -n ceph.file.layout /mnt/dir4/file1
# file: mnt/dir4/file1
ceph.file.layout="stripe_unit=4194304 stripe_count=2 object_size=4194304
pool=cephfs.fs_name.data"
```

3.4.8 IBM Storage Ceph File System snapshots

CephFS provides an asynchronous snapshot capability. These snapshots are accessible through a virtual directory named .snap.

Snapshots can be created at any level in the directory tree structure, including the root level of a CephFS file system.

Snapshot capabilities for specific users can be enabled at the individual client level by granting the 's' flag.

Snapshot capabilities can also be enabled or disabled for an entire file system by running a command of the following form:

```
# ceph fs set {filesystem_name} allow_new_snaps true|false
```

To create a snapshot, the user that mounted the CephFS creates a unique subdirectory within the '.snap' directory with the desired name by running a command of the following form:

```
# mkdir /mnt/.snap/mynewsnapshot
```

A user with sufficient privileges can create regular snapshots of a specific directory by running the **snap-schedule** command (Example 3-14).

Example 3-14 The snap-schedule command

```
# ceph fs snap-schedule add / 1h
Schedule set for path /
# ceph fs snap-schedule list /
/ 1h
# ceph fs snap-schedule status / | jq .
{
  "fs": "fs_name",
  "subvol": null,
  "path": "/",
  "rel_path": "/",
  "schedule": "1h",
  "retention": {},
  "start": "2023-01-11T00:00:00",
  "created": "2023-01-11T09:38:07",
  "first": null,
  "last": null,
  "last_pruned": null,
  "created_count": 0,
  "pruned_count": 0,
  "active": true
}
```

Tip: As a best practice, keep each snapshot at least 1 hour apart.

To set the retention of snapshots, run the **snap-schedule** command with a retention argument:

```
# ceph fs snap-schedule retention add / h 24
Retention added to path /
```

3.4.9 IBM Storage Ceph File System asynchronous replication

CephFS asynchronous replication between two clusters enables geo-replication for DR purposes.

This feature requires both clusters to run an identical Ceph release, IBM Storage Ceph 5.3 or later.

The feature is based on CephFS snapshots taken at regular intervals. The first snapshot requires a full transfer of the data, and later snapshots require transferring only the data that was updated (delta) since the last snapshot was applied on the remote cluster.

CephFS mirroring is disabled by default, and must be enabled separately by making the following changes:

- Enable the MGR `mirroring` module.
- Deploy a `cephfs-mirror` component with **cephadm**.
- Authorize the mirroring daemons in both clusters (source and target).
- Set up peering among the source and the target clusters.
- Configure the CephFS file system path to mirror.

Example 3-15 highlights the sequence of changes required.

Example 3-15 Sequence of changes that are required

```
# ceph mgr module enable mirroring
# ceph orch apply cephfs-mirror [node-name]
# ceph fs authorize fs-name client_ / rwps
# ceph fs snapshot mirror enable fs-name
# ceph fs snapshot mirror peer_bootstrap create fs-name peer-name site-name
# ceph fs snapshot mirror peer_bootstrap import fs-name bootstrap-token
# ceph fs snapshot mirror add fs-name path
```

4

Sizing IBM Storage Ceph

An IBM Storage Ceph cluster can serve different types of capacity and workload requirements. There is no *one size fits all* Ceph deployment, but each cluster must be carefully designed. This chapter provides guidance about designing and sizing an IBM Storage Ceph cluster.

This chapter has the following sections:

- Workload considerations
- Performance domains and storage pools
- Network considerations
- Colocation versus non-colocation
- Minimum hardware requirements for daemons
- OSD node CPU and memory requirements
- Scaling RGWs
- Recovery calculator
- IBM Storage Ready Nodes for Ceph
- Performance guidelines
- Sizing examples

> **Note:** For more information about sizing an IBM Storage Ceph environment, see the Planning section of the IBM Storage Ceph documentation.

4.1 Workload considerations

One of the key benefits of IBM Storage Ceph is its ability to serve multiple workloads from a single cluster by using performance domains. Different hardware configurations can be associated with each performance domain. For example, these performance domains might coexist in the same IBM Storage Ceph cluster:

- Latency-sensitive, input/output per second (IOPS)-intensive workloads, such as MySQL and MariaDB, are best served by solid-state drives (SSDs.
- Throughput-sensitive workloads may use HDDs with Ceph metadata on SSDs.
- Hard disk drives (HDDs)or QLC SSDs may be appropriate for cost and capacity-focused workloads.

Figure 4-1 shows an example of performance domains.

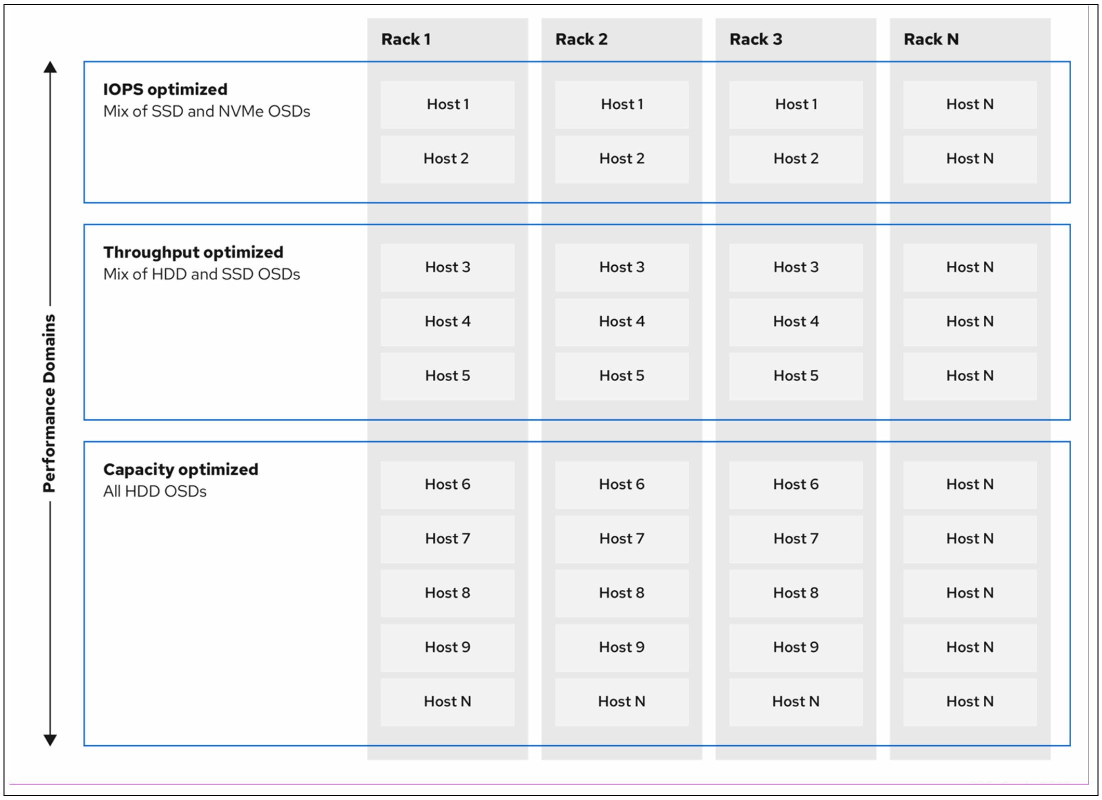


Figure 4-1 A cluster with multiple performance domains

4.1.1 IOPS-optimized

IOPS optimization deployments are suitable for cloud computing operations, such as running MySQL or MariaDB instances as virtual machines (VMs) on OpenStack or as containers on Red Hat OpenShift. IOPS-optimized deployments require higher performance SSD storage devices to achieve high IOPS and throughput.

An IOPS-optimized storage cluster has the following properties:

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

Here are use cases for an IOPS-optimized storage cluster or pool within diverse cluster:

- Block storage
- 2x replication or 3x replication using SSDs, for example for RGW index and CephFS metadata
- MySQL on OpenStack clouds

4.1.2 Throughput-optimized

Throughput-optimized deployments are ideal for serving large amounts of data, such as graphics, audio, and video content. They require high-bandwidth networking hardware, controllers, and HDDs or SDDs with fast sequential read/write performance. If fast data access is required, use a throughput-optimized storage strategy. If fast write performance is required, consider using SSDs for metadata and optionally payload data. A throughput-optimized storage cluster has the following properties:

- Lowest cost per MB/s (throughput)
- Highest MB/s per TB
- 97th percentile latency consistency

Here are use cases for a throughput-optimized storage cluster:

- Block or object storage
- 3x replication for block data or erasure coding (EC) for object data
- Archives and backups
- Active storage for video, audio, and graphics

4.1.3 Capacity-optimized

Capacity-optimized deployments are ideal for storing large amounts of data at the lowest possible cost. They typically trade performance for a more attractive price point. For example, capacity-optimized deployments often use slower and less expensive large capacity SATA or NL-SAS HDDs or QLC SSDs, and can host many object storage daemon (OSD) drives per server. A cost and capacity-optimized storage cluster has the lowest cost per TB. Note that IBM Storage Ceph RGW object storage and CephFS deployments ideally use fast SSD media for their metadata pools.

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

- Often object storage
- EC for maximizing usable capacity
- Artifact archival
- Video, audio, and image object repositories

4.2 Performance domains and storage pools

Multiple performance domains are implemented by creating multiple storage pools, each tailored to the needs of the use-case and access method.

There are two data protection strategies at the pool level:

- Replicated
- Erasure coded

By default, Ceph uses replicated pools, which means that data is copied from a primary OSD node to one or more secondary OSDs. Erasure-coded pools reduce the disk space required to ensure data durability but are computationally more expensive than replication and often have lower write performance.

EC is a method of durably storing a RADOS object in the Ceph storage cluster by breaking it into data chunks (k) and coding chunks (m). These chunks are stored on different OSDs in unique hosts, racks, or other failure domain. When an OSD fails, Ceph retrieves the remaining data (k) and coding (m) chunks from the surviving OSDs and uses the erasure code algorithm to restore the object from those chunks.

EC uses storage capacity more efficiently than replication. The n-replication approach maintains n copies of an object (3x by default in Ceph), but EC maintains k + m chunks. For example, four data and two coding chunks protect data while consuming only 1.5x the physical storage space of the original object's size.

Figure 4-2 shows the comparison between replication and EC.

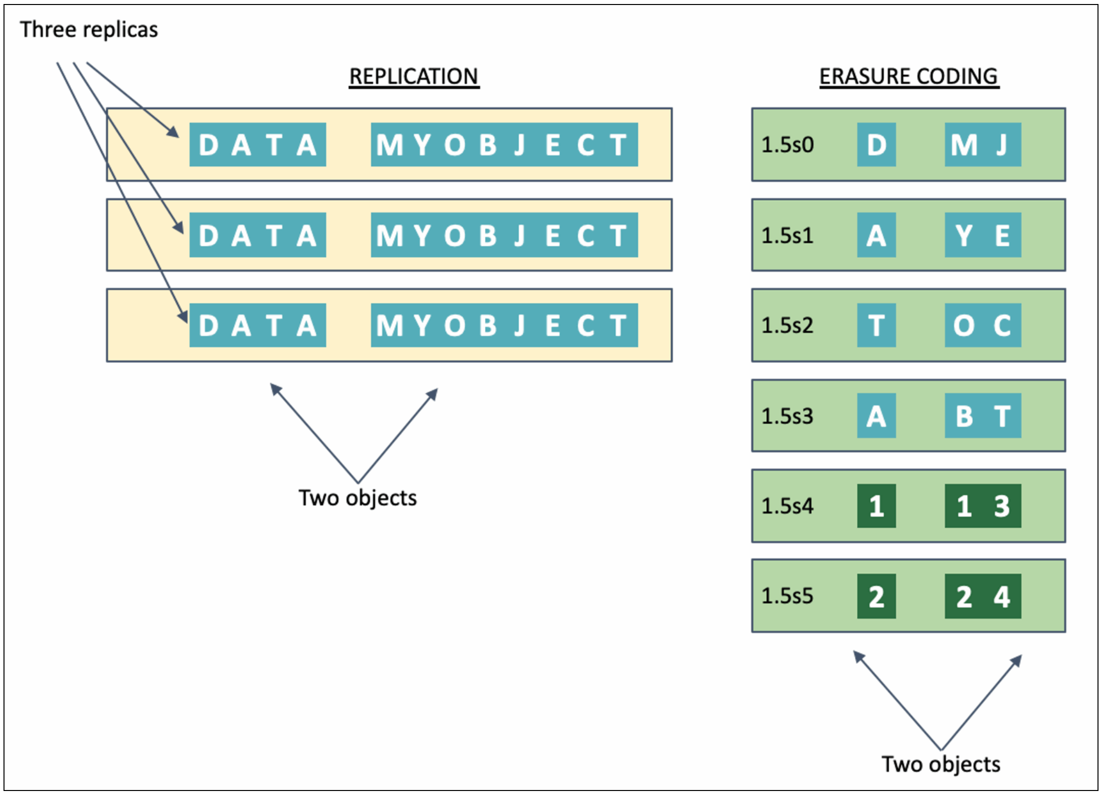


Figure 4-2 Comparison between replication and erasure coding

Although common EC profiles use less physical storage capacity than replication (lower space amplification), it requires more memory and CPU than replication to access or recover objects. EC is advantageous when data storage must be durable and fault-tolerant, but does not require fast write performance (for example, cold storage, historical records, and others).

4.3 Network considerations

Cloud storage solutions are susceptible to IOPS exhaustion due to network latency, bandwidth constraints, and other factors, long before the storage clusters reach their space capacity limits. Therefore, network hardware configurations must be chosen carefully to support the intended workloads and meet price/performance requirements.

Ceph supports two networks: a public network and a storage cluster network. The public network handles client traffic and communication with Ceph Monitors (mons). The cluster (aka replication, back end, or private) network if deployed carries Ceph OSD heartbeats, replication and recovery traffic. For production clusters, it is a best practice to use at least two bonded 10 Gbps links for each logical network interface.

Link Aggregation Control Protocol (LACP) mode 4 (`xmit_hash_policy 3+4 or 2+3` according to local topology) can be used to bond network interfaces. Use jumbo frames with a maximum transmission unit (MTU) of 9000, especially on the back-end or cluster network.

4.4 Colocation versus non-colocation

Every Ceph cluster includes Monitor daemons (ceph-mon), Manager (mgr) daemons (ceph-mgr), and Object Storage Daemons (ceph-osd).

Clusters that provide object storage also deploy RADOS Gateway (RGW) daemons (radosgw). If CephFS shared file services are required, Ceph Metadata Servers (MDSs) (ceph-mds) will be configured.

Colocating multiple services within a single cluster node has the following benefits:

- Significant improvement in total cost of ownership (TCO) versus smaller clusters
- Reduction from six hosts to four for the minimum configuration
- Simpler upgrade
- Better resource isolation
- Potential to share underlying physical storage device capacity

Modern servers have 16 or more CPU cores and large amounts of RAM, which supports colocation of services. Running each service type in dedicated cluster nodes that are separated from other types (non-collocated) would require many, smaller servers or underutilize resources.

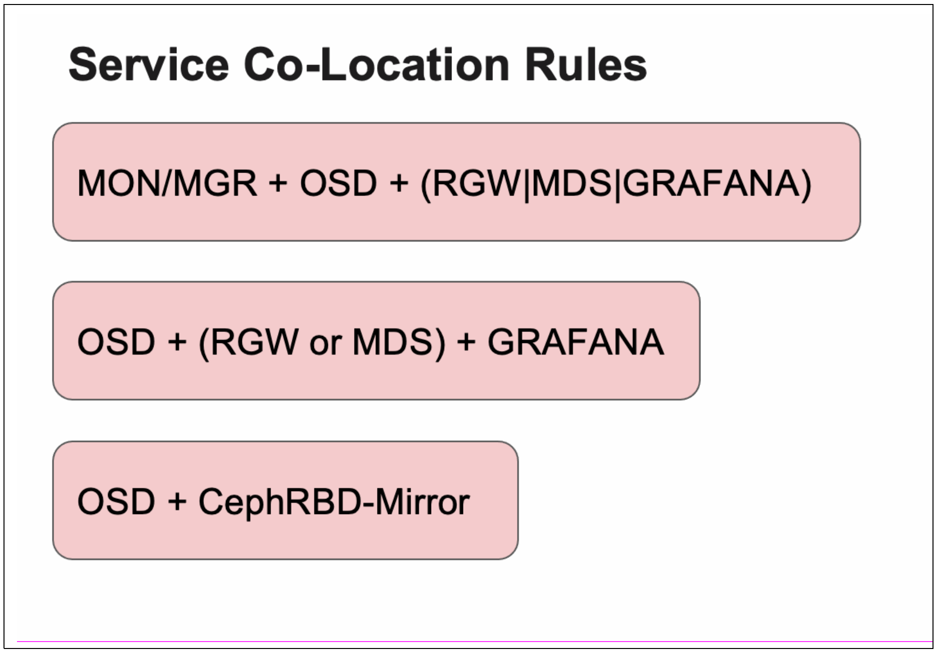
There are considerations that constrain deploying every type of service on a single node.

For any host that has ceph-mon/ceph-mgr and OSD, only one of the following should be added to the same host:

- RGW
- Ceph MDSs
- Grafana

RGW and Ceph Metadata services are best deployed on different nodes.

Grafana is typically deployed on a node that does not host Monitor or Manager services, as shown in Figure 4-3, to minimize resource contention.



Service Co-Location Rules

MON/MGR + OSD + (RGW|MDS|GRAFANA)

OSD + (RGW or MDS) + GRAFANA

OSD + CephRBD-Mirror

Figure 4-3 Grafana service colocation rules

The smallest supported cluster size for IBM Storage Ceph is four nodes. An example of service colocation for a four-node cluster is shown in Figure 4-4.

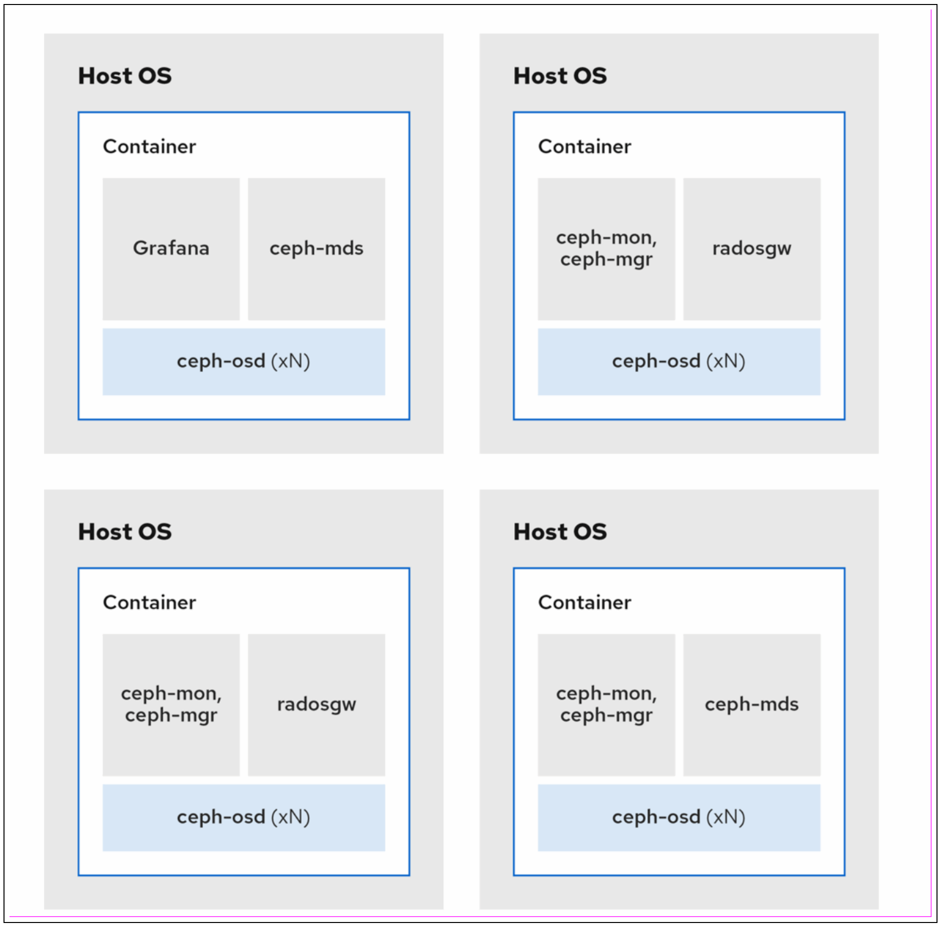
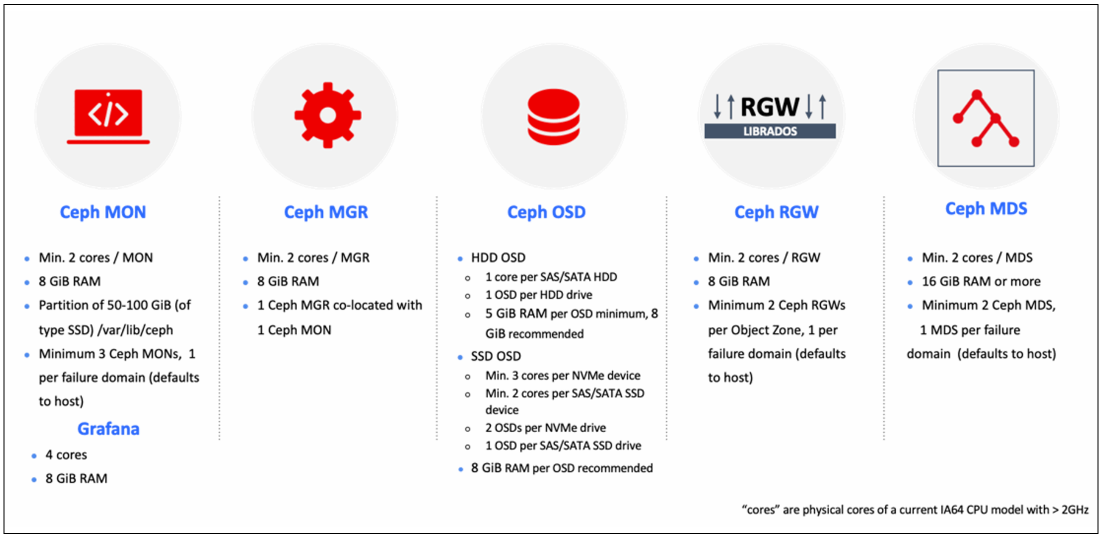


Figure 4-4 Example of service colocation for a four-node cluster

4.5 Minimum hardware requirements for daemons

Each service daemon requires CPU and memory resources, and some daemons need disk capacity as well. Best practice minimums for each daemon are shown in Figure 4-5.



Ceph MON

- Min. 2 cores / MON
- 8 GiB RAM
- Partition of 50-100 GiB (of type SSD) /var/lib/ceph
- Minimum 3 Ceph MONs, 1 per failure domain (defaults to host)

Grafana

- 4 cores
- 8 GiB RAM

Ceph MGR

- Min. 2 cores / MGR
- 8 GiB RAM
- 1 Ceph MGR co-located with 1 Ceph MON

Ceph OSD

- HDD OSD
  - 1 core per SAS/SATA HDD
  - 1 OSD per HDD drive
  - 5 GiB RAM per OSD minimum, 8 GiB recommended
- SSD OSD
  - Min. 3 cores per NVMe device
  - Min. 2 cores per SAS/SATA SSD device
  - 2 OSDs per NVMe drive
  - 1 OSD per SAS/SATA SSD drive
- 8 GiB RAM per OSD recommended

Ceph RGW

- Min. 2 cores / RGW
- 8 GiB RAM
- Minimum 2 Ceph RGWs per Object Zone, 1 per failure domain (defaults to host)

Ceph MDS

- Min. 2 cores / MDS
- 16 GiB RAM or more
- Minimum 2 Ceph MDS, 1 MDS per failure domain (defaults to host)

“cores” are physical cores of a current IA64 CPU model with > 2GHz

Figure 4-5 Best practice minimums for each daemon

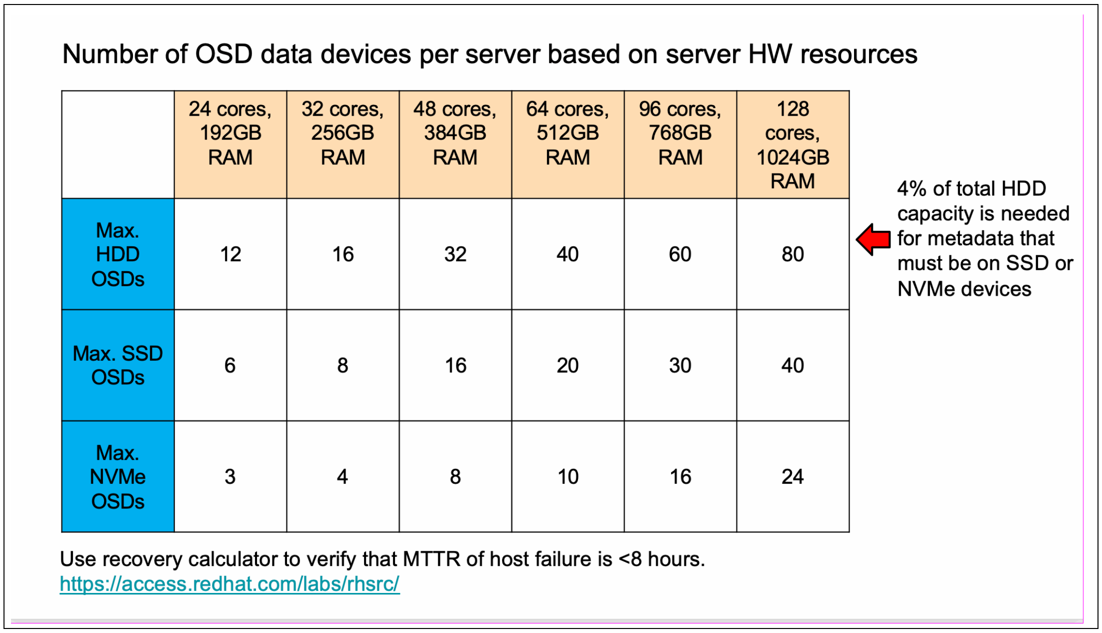
Daemon resource requirements might change between releases. Check the latest (or the release that you want) requirements from IBM Documentation.

Supported configurations: For more information about supported configurations, see Supported configurations (you must have an IBMid to access this link).

4.6 OSD node CPU and memory requirements

Adding up individual daemon CPU and memory requirements for each server can be time-consuming. Alternatively, you can use Figure 4-6 on page 91 as a guideline for server sizing. This table calculates sufficient CPU and memory per server to deploy common services including`ceph-mon`, `ceph-mgr`, `radosgw`, and Grafana, in addition to OSDs.

Figure 4-6 on page 91 shows the aggregate server CPU and memory resources that are required for an OSD node with various numbers of HDD, SATA SSD OSDs, or NVMe SSD OSDs.



Number of OSD data devices per server based on server HW resources

	24 cores, 192GB RAM	32 cores, 256GB RAM	48 cores, 384GB RAM	64 cores, 512GB RAM	96 cores, 768GB RAM	128 cores, 1024GB RAM
Max. HDD OSDs	12	16	32	40	60	80
Max. SSD OSDs	6	8	16	20	30	40
Max. NVMe OSDs	3	4	8	10	16	24

4% of total HDD capacity is needed for metadata that must be on SSD or NVMe devices

Use recovery calculator to verify that MTTR of host failure is <8 hours.
https://access.redhat.com/labs/rhsrc/

Figure 4-6 Aggregate server CPU and memory resources that are required for an OSD node

It is best practice for many workloads to add SSD devices with capacity at least 4% of aggregate HDD capacity to each server for metadata when deploying IBM Storage Ceph RGW for object storage service. RGW index pools experience a high rate of small random operations and do not perform well on slower media.

4.7 Scaling RGWs

To reach high throughput and availability for object storage service, multiple RGWs are required. Consider these general principles:

- RGW daemons can provide 1 - 2 GBps throughput each for large S3/Swift objects.
- One to eight RGW daemons can be placed on a single node.
- The minimum configuration is two RGW daemons that are deployed on separate nodes.
- Up to 80 RGW daemons in a single cluster have been tested.
- For optimal performance, allocate four CPU cores and 16 - 32 GB memory per RGW daemon.
- One RGW daemon per node in a 7-node cluster performs better (and provides higher availability) than seven RGW daemons on a single node. Spread RGW daemons across nodes.
- Multi-colocated RGWs on OSD hosts can improve read performance for read-intensive workloads, enabling near-zero-latency reads.
- Although not optimal for performance, running RGW daemons on dedicated VMs might be necessary for networking or security reasons, especially when providing service to the Internet at large. IBM Storage Ceph deployments can be designed with distinct cohorts of RGWs for internal and external clients.

Figure 4-7 shows performance scaling when serving large objects as additional RGW instances are deployed in a seven-node cluster.

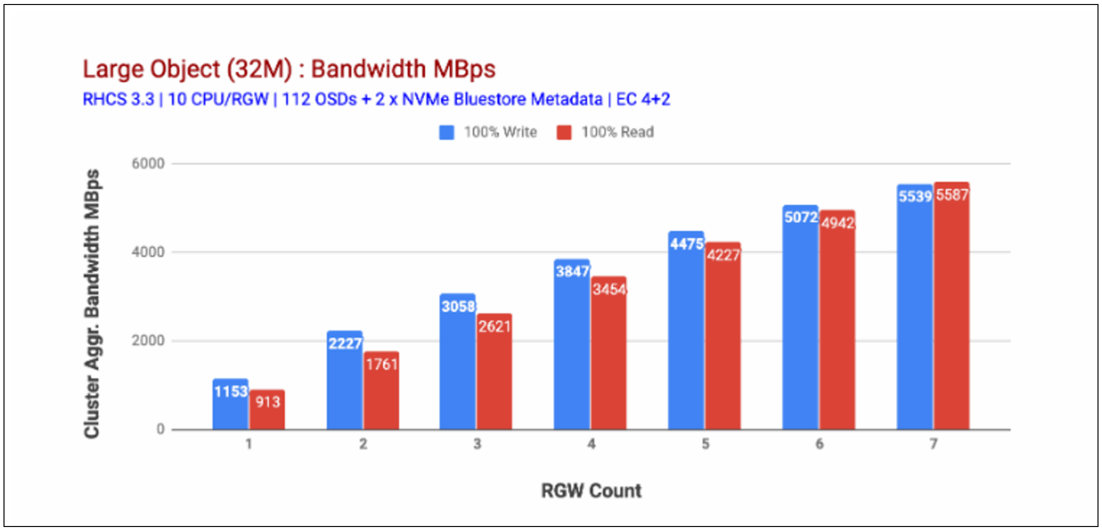


Figure 4-7 Performance scaling serving large objects as more RGW instances are deployed

Scaling from to one to seven RGW daemons in a seven-node cluster by spreading daemons across all nodes improves performance 5 - 6x. Adding additional RGW daemons to nodes that already host an RGW daemon does not scale performance as much.

Figure 4-8 shows the performance impact of adding a second RGW daemon to each of the seven cluster nodes.

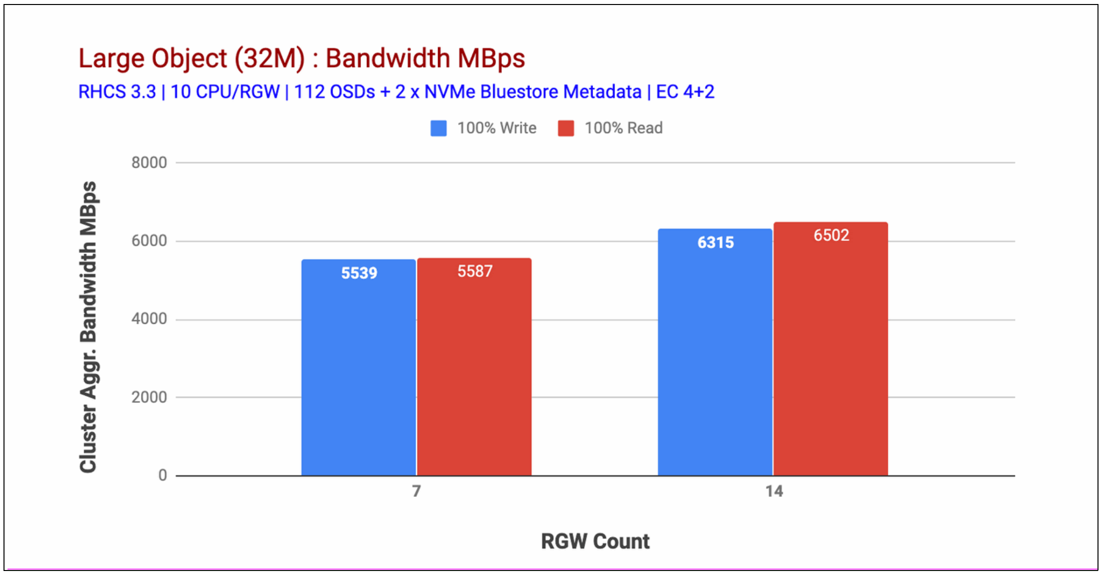


Figure 4-8 Performance effect of adding a second RGW daemon in each of the seven cluster nodes

Small objects also scale well in IBM Storage Ceph RGW deployments, and performance is typically measured in operations per second (OPS).. Figure 4-9 on page 93 shows the small object performance of the same cluster as above.

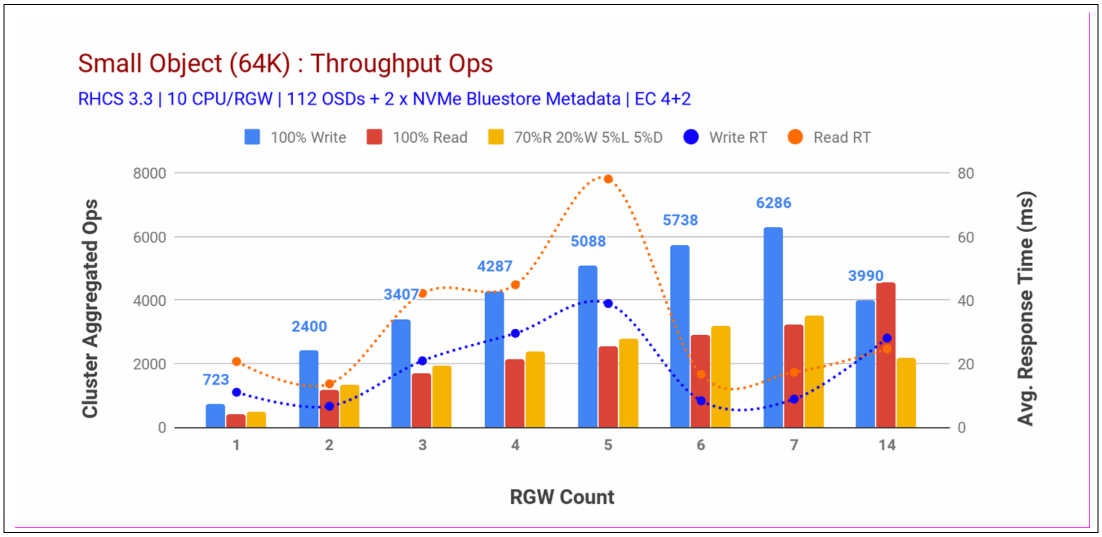


Figure 4-9 Small object performance of the same cluster as above

IBM Storage Ceph includes an ingress service for RGW that deploys haproxy and keepalived daemons for convenient provisioning of a single virtual IP service endpoint.

4.8 Recovery calculator

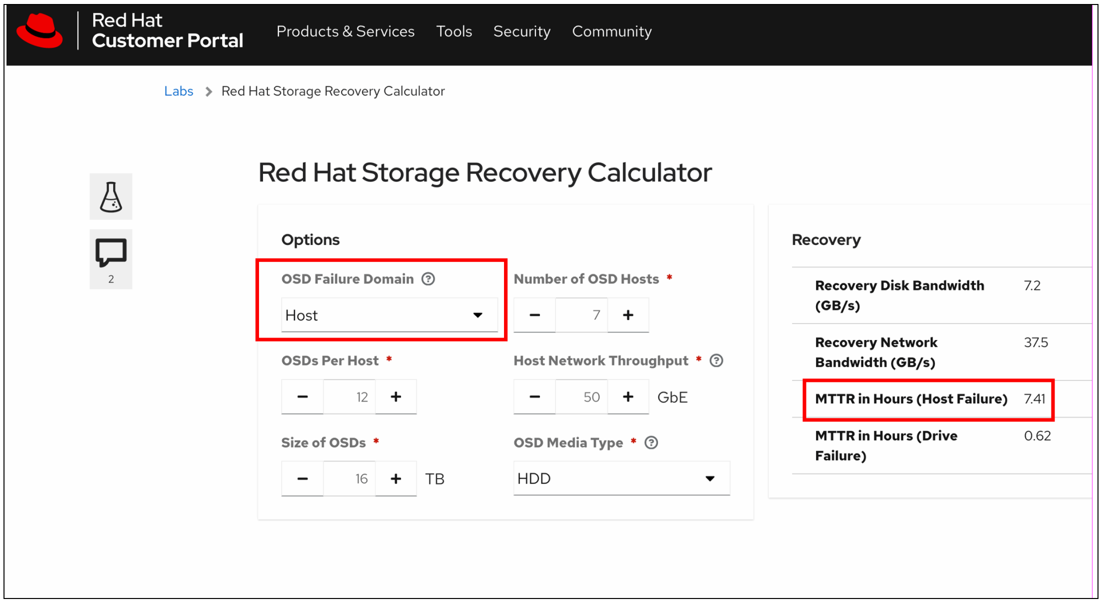
IBM Storage Ceph does not restrict OSD drive sizes that are used or the number of OSD drives per node. However, the cluster must be able to recover from a node failure in less than 8 hours.

A recovery calculator tool to estimate node recovery time can be found at Red Hat Storage Recovery Calculator.

You must be registered as a Red Hat customer to access the tool. Go to this site to create a Red Hat login.

Select **Host** as the OSD Failure Domain. The number to observe is MTTR in Hours (Host Failure), which must be less than 8 hours.

Figure 4-10 shows the recovery calculator user interface.



Labs > Red Hat Storage Recovery Calculator

Red Hat Storage Recovery Calculator

Options

OSD Failure Domain: Host

Number of OSD Hosts *: 7

OSDs Per Host *: 12

Host Network Throughput *: 50 GbE

Size of OSDs *: 16 TB

OSD Media Type *: HDD

Recovery

Recovery Disk Bandwidth (GB/s)	7.2
Recovery Network Bandwidth (GB/s)	37.5
MTTR in Hours (Host Failure)	7.41
MTTR in Hours (Drive Failure)	0.62

Figure 4-10 Ceph Recovery Calculator GUI

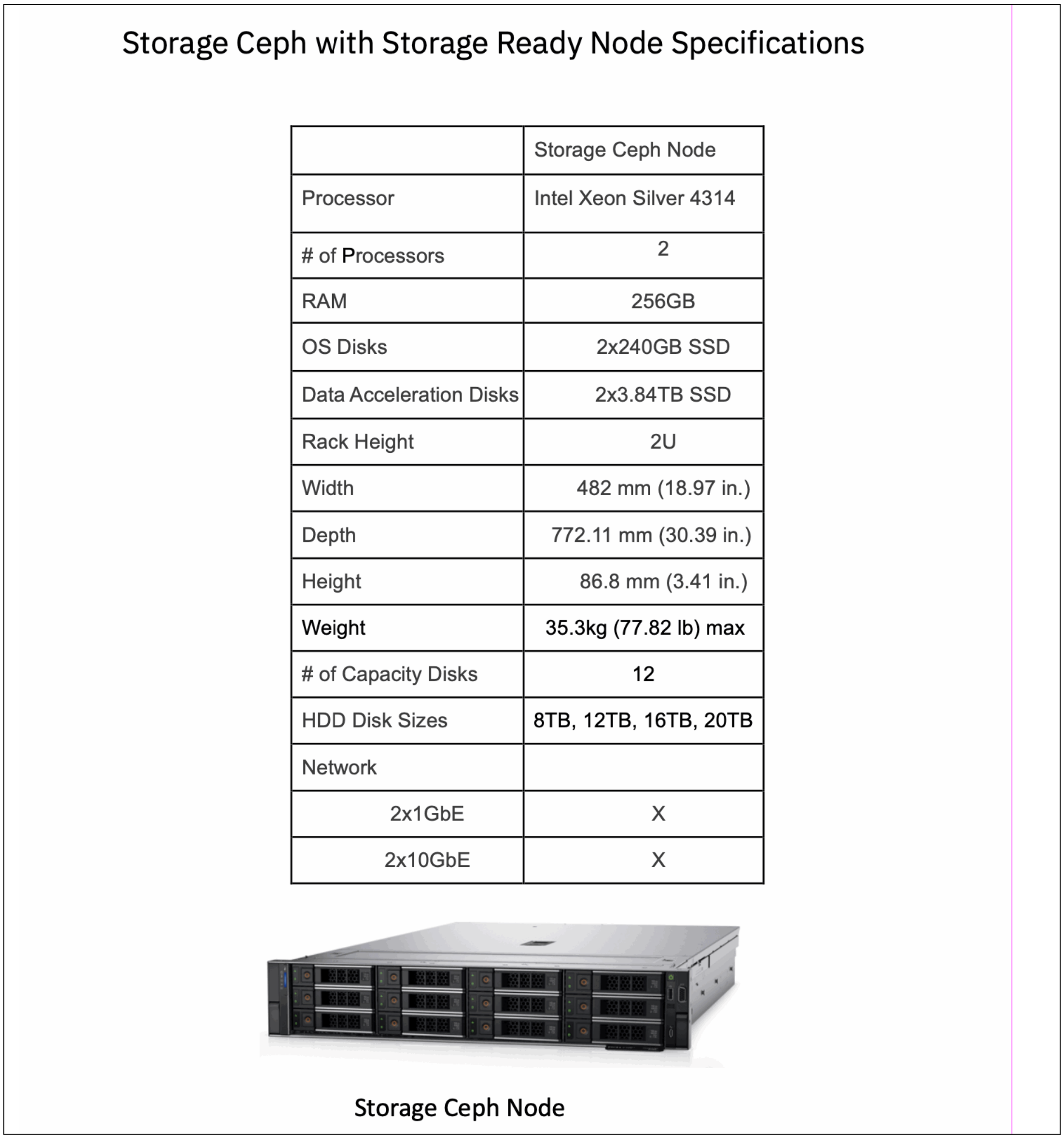
The usage of the recovery calculator is important for small clusters where recovery times with large drives are often more than 8 hours.

4.9 IBM Storage Ready Nodes for Ceph

IBM offers IBM Storage Ready Nodes for Ceph, which are pre-configured and valided configurations designed for use with IBM Storage Ceph software.

IBM Storage Ready Nodes for Ceph enable customers to receive both hardware maintenance and software support from IBM.

At the time of writing, there is one server model available with various drive configurations, as shown in Figure 4-11 on page 95.



Storage Ceph with Storage Ready Node Specifications

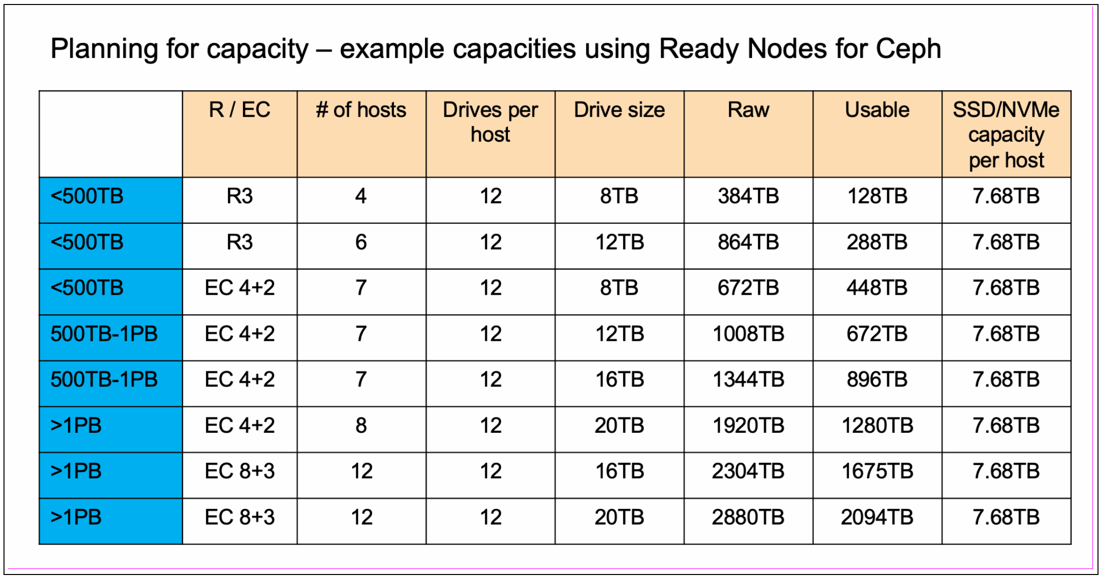
	Storage Ceph Node
Processor	Intel Xeon Silver 4314
# of Processors	2
RAM	256GB
OS Disks	2x240GB SSD
Data Acceleration Disks	2x3.84TB SSD
Rack Height	2U
Width	482 mm (18.97 in.)
Depth	772.11 mm (30.39 in.)
Height	86.8 mm (3.41 in.)
Weight	35.3kg (77.82 lb) max
# of Capacity Disks	12
HDD Disk Sizes	8TB, 12TB, 16TB, 20TB
Network	
2x1GbE	X
2x10GbE	X

Storage Ceph Node

Figure 4-11 IBM Storage Ceph with Storage Ready Node specifications

Each node comes with two 3.84 TB SSD acceleration drives, which are used as metadata drives. Available data drives are 8 TB, 12 TB, 16 TB, and 20 TB SATA HDDs. Every node must be fully populated with 12 drives of the same size.

Possible capacity examples for IBM Storage Ready Nodes for Ceph are shown in Figure 4-12.



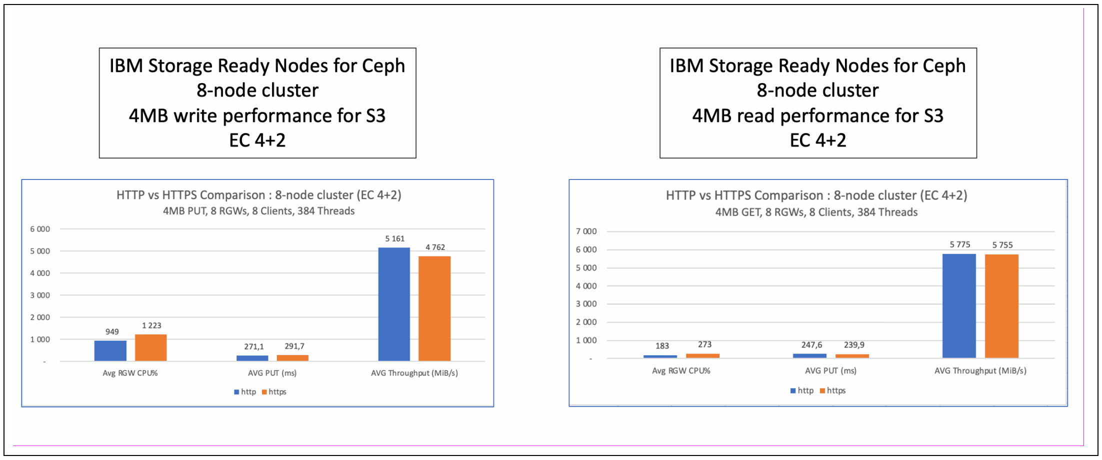
Planning for capacity – example capacities using Ready Nodes for Ceph

	R / EC	# of hosts	Drives per host	Drive size	Raw	Usable	SSD/NVMe capacity per host
<500TB	R3	4	12	8TB	384TB	128TB	7.68TB
<500TB	R3	6	12	12TB	864TB	288TB	7.68TB
<500TB	EC 4+2	7	12	8TB	672TB	448TB	7.68TB
500TB-1PB	EC 4+2	7	12	12TB	1008TB	672TB	7.68TB
500TB-1PB	EC 4+2	7	12	16TB	1344TB	896TB	7.68TB
>1PB	EC 4+2	8	12	20TB	1920TB	1280TB	7.68TB
>1PB	EC 8+3	12	12	16TB	2304TB	1675TB	7.68TB
>1PB	EC 8+3	12	12	20TB	2880TB	2094TB	7.68TB

Figure 4-12   Capacity examples for IBM Storage Ready Nodes for Ceph

IBM Storage Ready Nodes for Ceph are well suited for throughput-optimized workloads, such as backup storage.

Examples of an 8-node IBM Storage Ceph cluster's performance using Ready Nodes are shown in Figure 4-13.



IBM Storage Ready Nodes for Ceph
8-node cluster
4MB write performance for S3
EC 4+2

IBM Storage Ready Nodes for Ceph
8-node cluster
4MB read performance for S3
EC 4+2

Figure 4-13   Examples of an 8-node IBM Storage Ceph cluster's performance using Ready Nodes

4.10 Performance guidelines

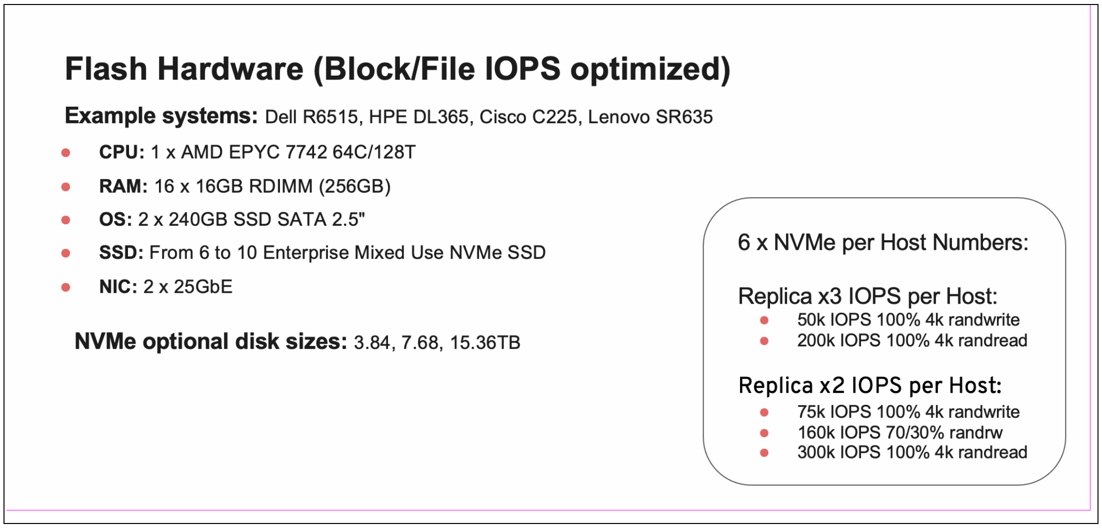
Daemon deployment among cluster nodes must be considered, and best practices should be followed to ensure that servers prvide sufficient CPU and memory resources to support the subsets of daemons, both during normal operation and when components are added or fail.

When servers are correctly sized we can estimate cluster performance

4.10.1 IOPS-optimized

IOPS-optimized Ceph clusters employ SSDs for data. The design goal of an IOPS-optimized pool or cluster is to provide ample IOPS capacity for RBD block devices, RGW metadata, or CephFS metadata These workloads are often a database application that requires high IOPS and low latency for small random reads and writes.

Example server models and estimated IOPS per host using NVMe SSDs are shown in Figure 4-14.



Flash Hardware (Block/File IOPS optimized)

Example systems: Dell R6515, HPE DL365, Cisco C225, Lenovo SR635

- **CPU:** 1 x AMD EPYC 7742 64C/128T
- **RAM:** 16 x 16GB RDIMM (256GB)
- **OS:** 2 x 240GB SSD SATA 2.5"
- **SSD:** From 6 to 10 Enterprise Mixed Use NVMe SSD
- **NIC:** 2 x 25GbE

NVMe optional disk sizes: 3.84, 7.68, 15.36TB

6 x NVMe per Host Numbers:

Replica x3 IOPS per Host:

- 50k IOPS 100% 4k randwrite
- 200k IOPS 100% 4k randread

Replica x2 IOPS per Host:

- 75k IOPS 100% 4k randwrite
- 160k IOPS 70/30% randrw
- 300k IOPS 100% 4k randread

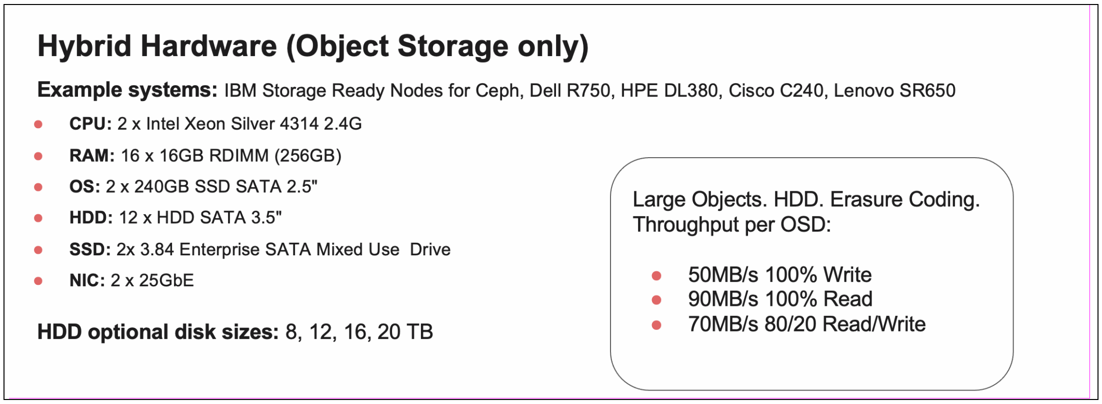
Figure 4-14 Example server models and estimated IOPS per host using NVMe SSDs

It is a best practice to have at least 10 Gbps of network bandwidth for every 12 HDDs in an OSD node for both cluster network and client network. This approach ensures that the network does not become a bottleneck.

4.10.2 Throughput-optimized

A throughput-optimized Ceph cluster can be designed with SSDs or hybrid using both SSDs and HDDs. The objective of a throughput-optimized cluster is to provide required throughput performance. The workload is typically large files, and the most common use cases are backups, artifactories, and content distribution via RGW object access.

Example server BOMs and estimated MB/s per OSD drive with HDD data drives are shown in Figure 4-15.



Hybrid Hardware (Object Storage only)

Example systems: IBM Storage Ready Nodes for Ceph, Dell R750, HPE DL380, Cisco C240, Lenovo SR650

- **CPU:** 2 x Intel Xeon Silver 4314 2.4G
- **RAM:** 16 x 16GB RDIMM (256GB)
- **OS:** 2 x 240GB SSD SATA 2.5"
- **HDD:** 12 x HDD SATA 3.5"
- **SSD:** 2x 3.84 Enterprise SATA Mixed Use Drive
- **NIC:** 2 x 25GbE

HDD optional disk sizes: 8, 12, 16, 20 TB

Large Objects. HDD. Erasure Coding.
Throughput per OSD:

- 50MB/s 100% Write
- 90MB/s 100% Read
- 70MB/s 80/20 Read/Write

Figure 4-15 Example server models and estimated MBps per OSD drive by HDDs as data drives

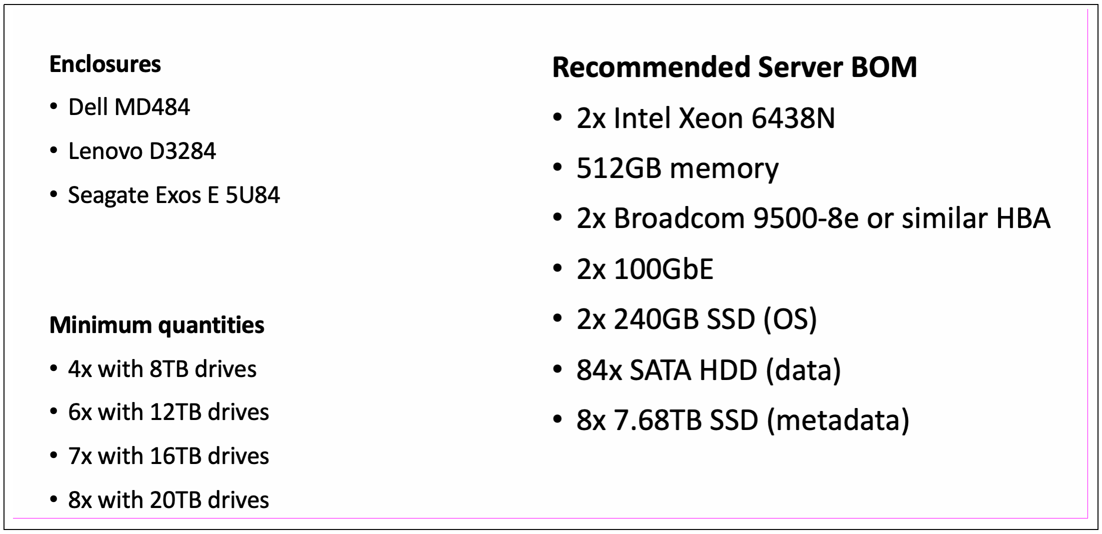
4.10.3 Capacity-optimized

Capacity-optimized Ceph clusters use large-capacity HDDs, and are often designed to be narrower (fewer servers) but deeper (more HDDs per server) than throughput-optimized clusters. The primary objective is to achieve the lowest cost per GB.

Capacity-optimized Ceph clusters are typically used for archive storage of large files, with RGW object access as the most common use case. Although it is possible to colocate metadata on the same HDDs as data, it is still a best practice to separate metadata and place it on SSDs, especially when using EC protection for the data. This approach can provide a performance benefit.

Many of the typical server models connect to an external Just a Bunch of Disks (JBOD) drive enclosure where data disks are. External JBODs can range from a 12-drive group to more than a 100 drive models.

Example server models for capacity-optimized clusters are shown in Figure 4-16.



Enclosures

- Dell MD484
- Lenovo D3284
- Seagate Exos E 5U84

Minimum quantities

- 4x with 8TB drives
- 6x with 12TB drives
- 7x with 16TB drives
- 8x with 20TB drives

Recommended Server BOM

- 2x Intel Xeon 6438N
- 512GB memory
- 2x Broadcom 9500-8e or similar HBA
- 2x 100GbE
- 2x 240GB SSD (OS)
- 84x SATA HDD (data)
- 8x 7.68TB SSD (metadata)

Figure 4-16 Example server models to build a capacity-optimized Ceph cluster

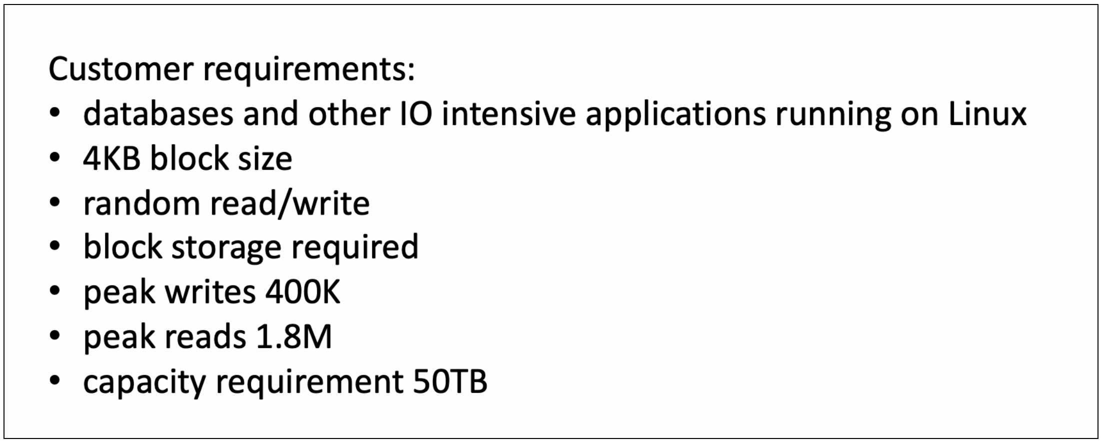
Clusters that are too narrow cannot be used with the largest drives because the calculated recovery time would be greater than 8 hours. For the cold archive type of use case, it is possible to have less than one CPU core per OSD drive. Note that it is still important to the blast radius, which is to say how much of a cluster's resources are impacted when an entire node fails or is taken offline for maintenance. A cluster with four 80-OSD servers has a much larger blast radius than one with twenty 12-OSD servers. An important nuance is considering what happens when a node experiences, say, a kernel panic or motherboard failure. Ceph will in most cases valiantly endeavor to restore configured data protection redundancy by creating replacement replicas of data on surviving servers. In a four-node cluster, this means that one should plan for each node to be less than two-thirds full in order to reserve capacity for recovery. With a larger number of nodes, the percentage of prudent reserve required naturally decreases.

4.11 Sizing examples

This section covers scenarios for IBM Storage Ceph sizing.

4.11.1 IOPS-optimized scenario

An example IOPS-optimized use case scenario is described in Figure 4-17.



Customer requirements:

- databases and other IO intensive applications running on Linux
- 4KB block size
- random read/write
- block storage required
- peak writes 400K
- peak reads 1.8M
- capacity requirement 50TB

Figure 4-17   Scenario: IOPS-optimized

The IOPS requirement is high for a small capacity requirement. Therefore, plan to use NVMe SSDs, which provide the highest IOPS per drive. Refer to the performance chart in Figure 4-14 on page 97.

Performance planning

Figure 4-14 on page 97 shows that a single server with six NVMe drives can serve 50 K write IOPS and 200 K read IOPS with a 4 KB block size.

Here are the customer requirements:

- 400 K writes
- 400 K / 50 K (single-server performance) = 8 servers required
- 1.8 M reads
- 1.8 M / 200 K (single-server performance) = 9 servers required

Plan for ten servers to accommodate growth and server failure and maintenance.

Capacity planning

Here are considerations for capacity planning:

- Use 3-way replication with block data.
- The total raw storage requirement is 3x 50 TB = 150 TB.
- The performance calculation is based on having six NVMe drives per server.
- Ten servers in the cluster means 60 OSD drives.
- The capacity requirement per NVMe drive is 150 TB (total raw capacity) / 60 (OSD drives) = 2.5 TB.
- The closest larger capacity NVMe is 3.84 TB.

CPU, memory, and networking requirements

It is critical to have enough CPU, memory, and networking bandwidth when using fast NVMe drives as data drives. Note that for IBM Storage Ceph purposes a "core" means a virtual core (aka thread). Hyperthreading (HT) is usually beneficial for Ceph workloads, so a nominal 64 core server usually offers 128 vcores (threads)

CPU requirement

A best practice is to have 4 - 10 CPU cores per NVMe OSD. Our example solution has six NVMe OSDs per server, so the CPU requirement for the OSD daemons is 24 - 60 CPU cores (threads).

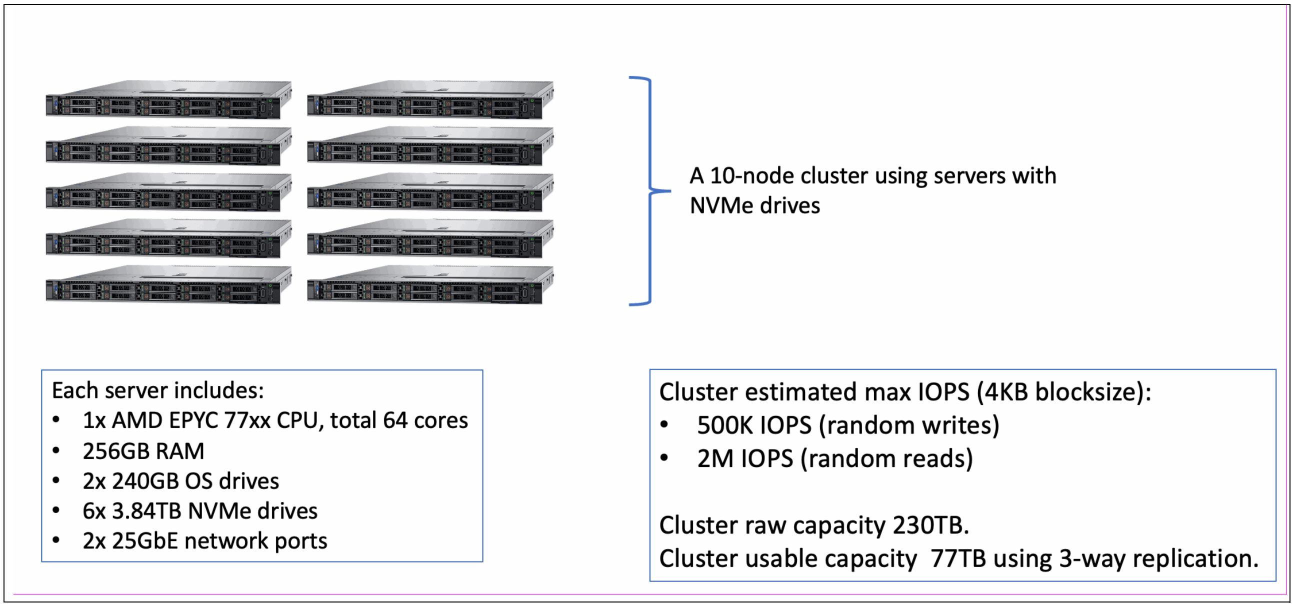
Memory requirement

A best practice is to have 12 - 15 GB of memory per NVMe drive. Our solution has six NVMe drives per server, so the memory requirement for the OSD daemons is 72 - 90 GB.

Networking requirement

We know that six NVMe drives in one host serve about 200 K read IOPS (the write IOPS is less). The block size is 4 KB. The network bandwidth that is required per server is 200000* 4 KBps =
800 MBps = 6.4 Gbps for both client and cluster networks.

Figure 4-18 shows our possible solution for an IOPS-optimized Ceph cluster.



A 10-node cluster using servers with NVMe drives

Each server includes:

- 1x AMD EPYC 77xx CPU, total 64 cores
- 256GB RAM
- 2x 240GB OS drives
- 6x 3.84TB NVMe drives
- 2x 25GbE network ports

Cluster estimated max IOPS (4KB blocksize):

- 500K IOPS (random writes)
- 2M IOPS (random reads)

Cluster raw capacity 230TB.
Cluster usable capacity 77TB using 3-way replication.

Figure 4-18   A possible solution for an IOPS-optimized Ceph cluster

4.11.2 Throughput-optimized scenario

A customer is looking for 1 PB of usable S3 capacity for their long-term backups using IBM Storage Defender Data Protect.

Sizing example

Throughput-optimized systems often combine low-cost drives for data and high-performance drives for metadata. IBM Storage Ready Nodes for Ceph offer this combination of drives.

The Ceph cluster must be designed in a way that allows data to be rebuilt on the remaining nodes if one of the servers fails. This headroom must be added to the requested 1 PB.

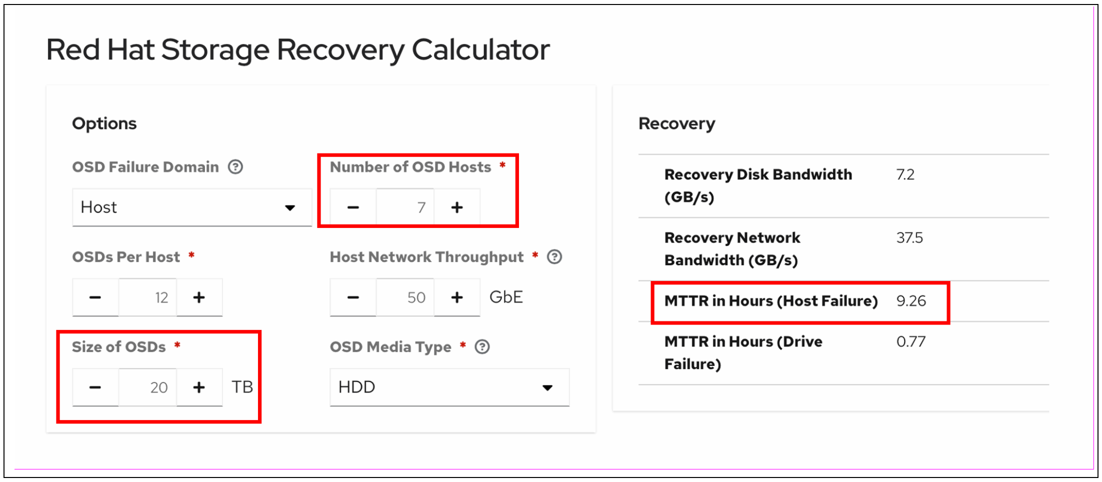
Also, a Ceph cluster goes into read-only mode when an OSD reaches 95% capacity utilization and OSDs that reach 90% (by default) cannot accept backfill/recovery .

Ceph S3 backup pools typically use EC to protect data. For a 1 PB usable capacity, EC 4+2 is often a better choice than EC 8+3 or EC 8+4 because erasure coding profiles with large values of (K+M) result in lower write performance and much longer elapsed time to recover from component loss.

The minimum number of nodes to support an EC pool is (K+M+1). In our 4+2 case, this means 7. Investigate whether a good solution for required capacity requirement can be found that has seven nodes, or if more are needed.

The largest available drive size in IBM Storage Ready Nodes for Ceph is 20 TB (7 nodes * 12 drives per node * 20 TB drives = 1680 TB raw).

This configuration seems to be a low-cost alternative for 1 PB of usable space. But, can we use 20 TB drives in a 7-node cluster, as shown in Figure 4-19? What does the recovery calculator show?



Red Hat Storage Recovery Calculator

Options

OSD Failure Domain: Host

Number of OSD Hosts *: 7

OSDs Per Host *: 12

Host Network Throughput *: 50 GbE

Size of OSDs *: 20 TB

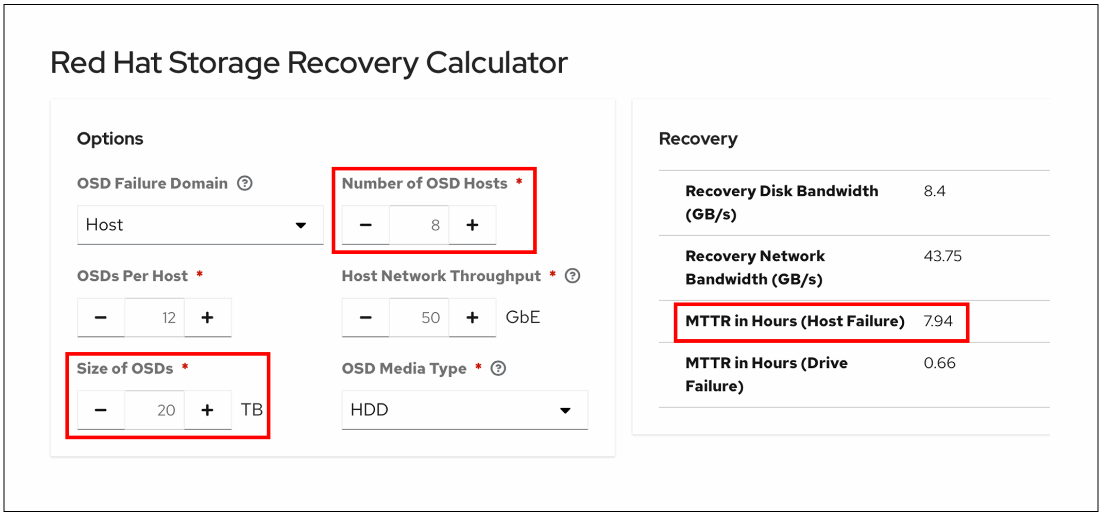
OSD Media Type *: HDD

Recovery

Recovery Disk Bandwidth (GB/s)	7.2
Recovery Network Bandwidth (GB/s)	37.5
MTTR in Hours (Host Failure)	9.26
MTTR in Hours (Drive Failure)	0.77

Figure 4-19 Seven-node 20 TB drives

Because the calculated recovery time for host failure is longer than 8 hours, this cluster would not be supported. So, increase the number of hosts in the recovery calculator from 7 to 8 (Figure 4-20).



Red Hat Storage Recovery Calculator

Options

OSD Failure Domain: Host

Number of OSD Hosts *: 8

OSDs Per Host *: 12

Host Network Throughput *: 50 GbE

Size of OSDs *: 20 TB

OSD Media Type *: HDD

Recovery

Recovery Disk Bandwidth (GB/s)	8.4
Recovery Network Bandwidth (GB/s)	43.75
MTTR in Hours (Host Failure)	7.94
MTTR in Hours (Drive Failure)	0.66

Figure 4-20   Eight-node 20 TB drives

Recovery time is just under 8 hours.

The capacity math for an 8-node cluster with 20 TB drives is shown in the following list:

1. Eight nodes, each with 12 pcs of 20 TB drives = 8*12*20 TB = 1920 TB raw capacity.
2. EC 4+2 has four data chunks and two EC chunks. Each chunk is the same size.
3. One chunk of 1920 TB is 1920 TB / 6 = 320 TB.
4. There are four data chunks in EC 4+2, so the usable capacity is 4*320 TB = 1280 TB.
5. This scenario would satisfy the customer's 1 PB capacity requirement, and the cluster has enough capacity to rebuild data if one node fails and still stay below 95% capacity utilization.
   a. Each server has 12x 20 TB = 240 TB raw capacity.
   b. Total cluster raw capacity is 1920 TB.
   c. The cluster raw capacity if one node breaks is 1920 TB - 240 TB = 1680 TB.
   d. Raw 1680 TB equals 1120 TB usable.
   e. A customer has 1 PB (1000 TB) of data.
   f. The capacity utilization if one host is broken is 1000 TB / 1120 TB = 0.89 = 89%.
6. The conclusion is that an 8-node cluster with 20 TB drives has enough capacity to stay below 95% capacity utilization even if one the servers fails and its data is rebuilt on the remaining servers.
7. Now that the raw capacity in each OSD node is calculated, add SSD drives for metadata with capacity at least 4% of the raw capacity on the host.
   – 4% of 240 TB is 9.6 TB.
   – A best practice is to have one metadata SSD for every four or six HDDs. Therefore, provision two or three 3.84 TB SSDs in each host.

- Two 3.84 TB SSDs (non-RAID) amounts to 7.68 TB, which is 3.2% of raw capacity in the host.
- Three 3.84 TB SSDs (non-RAID) amounts to 11.52 TB, which is 4.8% of raw capacity in the host.

8. Because the customer performance requirement (2 GBps for ingest) is not high compared to what an 8-node cluster can achieve (over 4 GBps), you can settle for the lower SSD metadata capacity.

CPU, memory, and networking requirements

This section discusses CPU, memory, and networking requirements.

CPU requirement

A best practice is to have at least one CPU core (thread) per HDD. This solution has 12 HDDs per server, so the CPU requirement for the OSD daemons is 12 CPU cores.

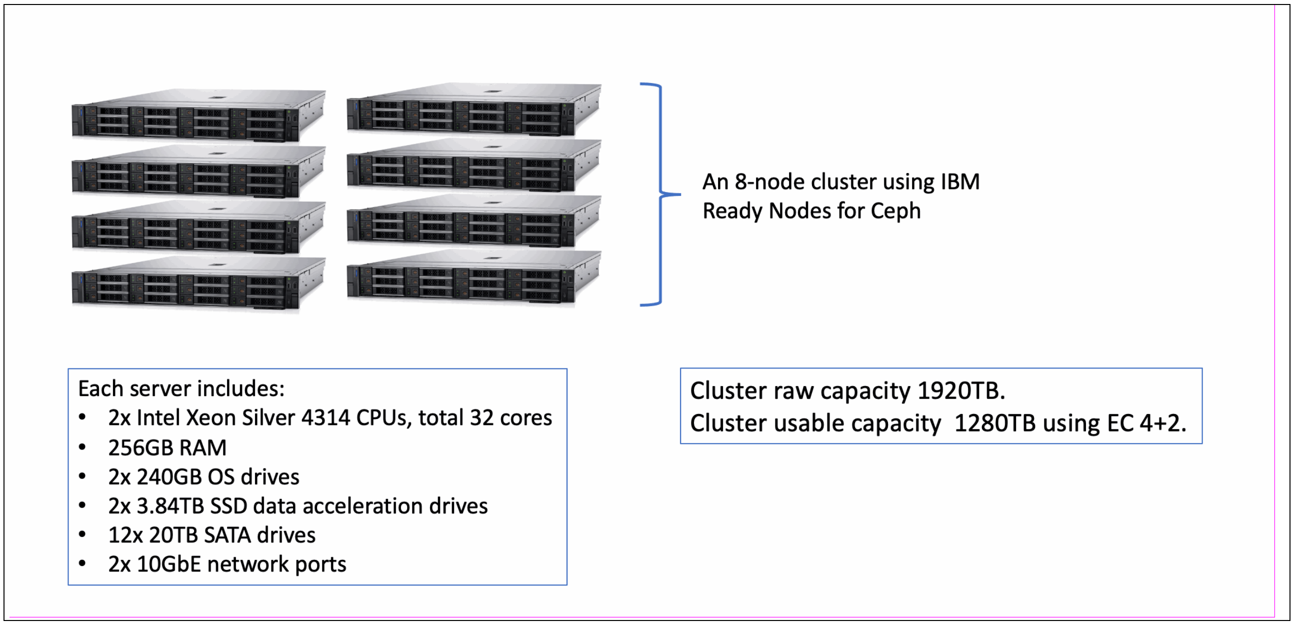
Memory requirement

A best practice is to have 10 GB of memory per HDD for a throughput-optimized use case. Our solution has 12 HDDs per server, so the memory requirement for the OSD daemons is 120 GB.

Networking requirement

One HDD delivers 90 MB/s read performance. Our solution has 12 HDDs per server, so the network bandwidth requirement per network is 12* 90 MBps = 1080 MBps = 8.64 Gbps.

Figure 4-21 shows a solution for a 1 PB throughput-optimized Ceph cluster.



Each server includes:
- 2x Intel Xeon Silver 4314 CPUs, total 32 cores
- 256GB RAM
- 2x 240GB OS drives
- 2x 3.84TB SSD data acceleration drives
- 12x 20TB SATA drives
- 2x 10GbE network ports

Cluster raw capacity 1920TB.
Cluster usable capacity 1280TB using EC 4+2.

Figure 4-21   A possible solution for a 1 PB throughput-optimized Ceph cluster

4.11.3 Capacity-optimized scenario

Here are the requirements for a capacity-optimized scenario:

- Low-cost storage for cold data archives
- S3 object storage (RGW) access method
- 12 PB usable capacity

A possible solution

An object storage archive for cold data is a use case where you can design a cluster that is narrow (fewer servers) and deep (many OSD drives per host). The result is a low-cost but not high-performing cluster. Dense servers benefit greatly from using SSDs for metadata, and it is a best practice to include them even for a cold archive use case.

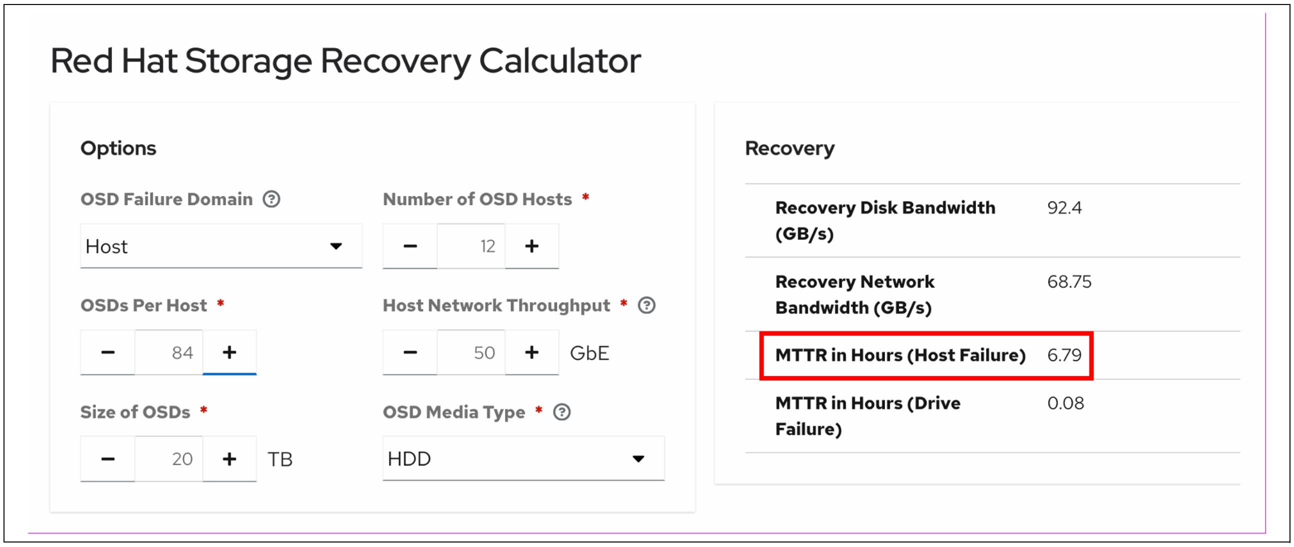
Because the capacity requirement is high, consider using EC 8+3, which has the lowest overhead (raw capacity versus usable capacity). An IBM Storage Ceph cluster that uses EC 8+3 requires a minimum of 12 nodes.

First, check whether a 12-node cluster with large drives and and dense chassis can recover from host failure in less than 8 hours.

Bring up the Recovery Calculator tool and enter the parameters of the cluster that you want to check:

- Twelve OSD hosts
- Eighty-four OSD drives at 20 TB each per host
- Two 25 GbE network ports

Figure 4-22 shows that the calculated recovery time for such a cluster is less than 8 hours, which means that this scenario is a supported design.



Red Hat Storage Recovery Calculator

Options

OSD Failure Domain: Host

Number of OSD Hosts *: 12

OSDs Per Host *: 84

Host Network Throughput *: 50 GbE

Size of OSDs *: 20 TB

OSD Media Type *: HDD

Recovery

Recovery Disk Bandwidth (GB/s)	92.4
Recovery Network Bandwidth (GB/s)	68.75
MTTR in Hours (Host Failure)	6.79
MTTR in Hours (Drive Failure)	0.08

Figure 4-22 Checking the calculated recovery time of a 12-node cluster

Calculate the capacity:

1. There are 12 nodes, each with eighty-four OSDs at 20 TB = 12*84*20 TB = 20160 TB raw capacity.
2. EC 8+3 has eight data chunks and three EC chunks. Each chunk is the same size.
3. One chunk of 20160 TB is 20160 TB / 11 = 1832 TB.

4. There are eight data chunks in EC 8+3, so the usable capacity is 8*1832 TB = 14656 TB.
5. This capacity would satisfy the customer's 12 PB capacity requirement, and the cluster has enough capacity to rebuild data if one node fails and still stay below 95% capacity utilization.
   - Each server has 84x 20 TB = 1680 TB raw capacity.
   - The total cluster raw capacity is 20160 TB.
   - The cluster raw capacity if one node breaks is 20160 TB - 1680 TB = 18480 TB.
   - Raw 18480 TB equals 13440 TB usable.
   - The customer has 12 PB (12000 TB) of data.
   - The capacity utilization if one host is broken = 12000 TB / 13440 TB = 0.89 = 89%.
6. The conclusion is that a 12-node cluster with 20 TB drives has enough capacity to stay below 95% capacity utilization even if one the servers breaks and its data is rebuilt on the remaining servers.

   Even for a cold archive, it is a best practice to have SSD metadata devices totaling 4% of the HDD capacity per server, which results in 4% of 84*20 TB = 67 TB. 8 or 10 large SSDs (7.68 TB) would be required for metadata, and more if they will be protected with server RAID.

CPU, memory, and networking requirements

The following lists the CPU, memory, and networking requirements for this use case.

CPU requirement

For a cold archive use case, you can provision as few as 0.5 CPU cores (threads) per HDD. Our solution has 84 HDDs per server, so the CPU requirement for OSD daemons is 42 CPU cores.

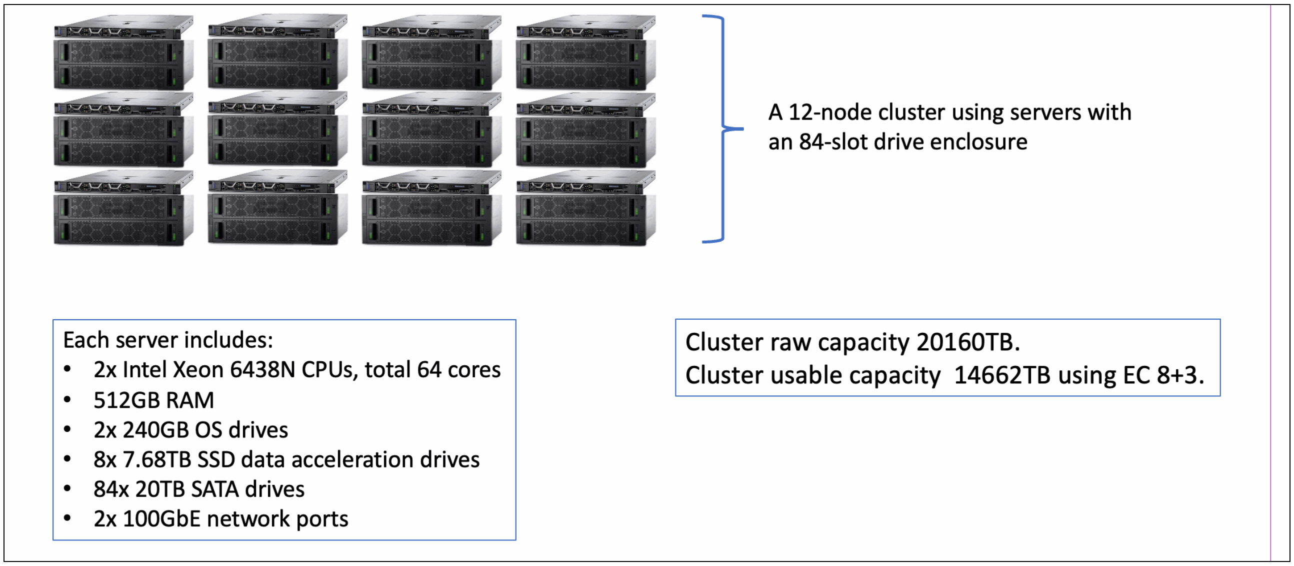
Memory requirement

A best practice is to have 5 GB of memory per HDD for capacity-optimized use case. Our solution has 84 HDDs per server, so the memory requirement for OSD daemons is 420 GB.

Networking requirement

One HDD has 90 MBps read performance. Our solution has 84 HDDs per server, so the network bandwidth requirement per network is 84* 90 MBps = 7560 MBps ~60 Gbps.

Figure 4-23 shows a possible solution for a 12 PB capacity-optimized Ceph cluster.



A 12-node cluster using servers with an 84-slot drive enclosure

Each server includes:

- 2x Intel Xeon 6438N CPUs, total 64 cores
- 512GB RAM
- 2x 240GB OS drives
- 8x 7.68TB SSD data acceleration drives
- 84x 20TB SATA drives
- 2x 100GbE network ports

Cluster raw capacity 20160TB.
Cluster usable capacity 14662TB using EC 8+3.

Figure 4-23 A possible solution for a 12 PB capacity optimized Ceph cluster

5

Monitoring your IBM Storage Ceph environment

The supporting hardware of an IBM Storage Ceph cluster is subject to failure over time. The Ceph administrator is responsible for monitoring and, if necessary, troubleshooting the cluster to keep it in a healthy state. This chapter provides an overview of the Ceph services, metrics, and tools that are in scope for end-to-end monitoring of cluster health and performance.

This chapter has the following sections:

- Monitoring overview
- IBM Storage Ceph monitoring examples
- Conclusion
- References on the World Wide Web

5.1 Monitoring overview

Managing and monitoring a Ceph storage deployment demands a diverse range of technical skills. Organizations devise various strategies for allocating administrative responsibilities in accordance with their unique requirements. In large Ceph deployments, organizations often assign technical specialists to specific technical domains. Conversely, organizations with smaller Ceph clusters typically employ IT generalists who oversee multiple disciplines.

Here are the three technical skills that are needed to manage and monitor a Ceph cluster:

- Operating system: The ability to install, configure, tune, and manage the operating system running on Ceph nodes (Red Hat Enterprise Linux).
- Networking: The ability to configure TCP/IP networking in support of Ceph cluster operations for both client access (public network) and Ceph node-to-node communications (cluster network).
- Storage: The ability to design and implement a Ceph storage architecture tailored to the performance, data availability, and data durability requirements of client applications and data.

Administrators must perform these roles in managing a Ceph cluster by using the correct tool at the correct time. This chapter provides an overview of Ceph cluster monitoring and the tools that are available to perform monitoring tasks.

5.1.1 IBM Storage Ceph monitoring components

This section describes the Ceph monitoring components.

Ceph Monitor

Ceph Monitors (mons) are the daemons that are responsible for maintaining the cluster map, which is a collection of data that provides comprehensive information about the cluster's topology, state, and configuration. Ceph proactively handles cluster events, updates the relevant map, and distributes the updated map to all Monitor daemons. A typical Ceph cluster includes three or five Monitors, each running on a separate for high availability. To ensure data integrity and consistency, Monitors employ a consensus mechanism that requires a majority of the configured Monitors to be available and agree on the map update before it is applied. This mechanism is why a Ceph cluster must be configured with an odd number of Monitors to establish a quorum and prevent potential conflicts.

Ceph Manager

The Ceph Manager (mgr) is a critical component of any Ceph cluster that is responsible for collecting and aggregating cluster-wide statistics. The first Manager daemon in a cluster becomes the active Manager and all others are on standby. If the active Manager does not send a beacon within the configured time interval, a standby takes over. Client I/O operations continue normally if no Manager is active, but queries for cluster statistics fail. A best practice is to deploy at least two Managers in a Ceph cluster to provide high availability (HA). Ceph Managers are typically run on the same hosts as the Monitor daemons, but this association is not required.

The existence of the first Ceph Manager and the first Ceph Monitor defines the existence of a Ceph cluster. When cephadm bootstraps the first node, a Monitor and a Manager are started on that node, and the cluster is considered operational. In the Ceph single-node experience, that first node also has one or several object storage daemons (OSDs) further extending the concept of an operational and fully functioning Ceph cluster that can store data. Note that a single node cluster is in no way a viable production architecture, but is useful for bootstrapping, testing, and development.

Ceph daemons

Each daemon within a Ceph cluster maintains a log of events, and the Monitors maintain a cluster log that records high-level events and status regarding the cluster as a whole. These events are logged to disk on Monitors servers (in the default location /var/log/ceph/ceph.log), and can be monitored with CLI commands or passed to log management solutions like Splunk or Grafana Loki.

5.1.2 IBM Storage Ceph monitoring categories

This section describes the Ceph monitoring categories.

Monitoring services

A Ceph cluster is a collection of daemons and services that perform their respective roles in the operation of a Ceph cluster. The first step in Ceph monitoring or troubleshooting is to inspect these Ceph components and discover whether they are running on the nodes they are designated to run, and if they report a healthy state. For example, if the cluster design specifies two Ceph Object Gateway (RGW) instances for handling S3 API client requests on distinct nodes, a single CLI command or a glance at the Ceph Dashboard provides insights into the operating status of the RGW daemons and their respective nodes.

Monitoring resources

Ceph resources encompass a cluster’s entities and constructs that define its characteristics. These resources include networking infrastructure, storage devices (SSDs or HDDs), logical storage pools and their capacities, and data protection mechanisms. As with monitoring other resources, understanding the health of Ceph storage resources is crucial for effective cluster management. At the core, administrators must ensure sufficient capacity and expansion capabilities to provide the appropriate data protection for the applications that they support.

Monitoring performance

nsuring that both physical and software-defined resources operate at required levels is the responsibility of the Ceph administrator. System alerts and notifications serve as valuable tools, alerting the administrator to anomalies without requiring constant hands-on surveillance. Key metrics include node utilization (CPU, memory, and disk), network utilization (interfaces, bandwidth, latency, and routing), storage device performance, and daemon workload.

5.1.3 IBM Storage Ceph monitoring tools

This section describes the Ceph monitoring tools.

Ceph Dashboard

The Ceph Dashboard UI provides an HTTP web browser interface that is accessible on port 8443. Dashboard navigation menus provide real-time health and basic statistics, which are context-sensitive, for cluster resources. For example, the Dashboard can display cluster read and write throughput and IOPS. When you bootstrap your cluster with the **`cephadm bootstrap`** command, then the Dashboard is enabled by default.

The Dashboard plug-in provides context-sensitive metrics for the following:

- Hosts
- OSDs
- Pools
- Block devices
- File systems
- Object storage gateways

The Dashboard also provides a convenient access point to observe the state and the health of resources and services in the Ceph cluster. The following Dashboard UI navigation provides at-a-glance views into the health of the cluster.

- **Cluster** → **Services** (View Ceph services and daemon node placement and instances.)
- **Cluster** → **Logs** (View and search within the cluster logs, audit logs, and daemon logs..)
- **Cluster** → **Monitoring** (Management of active alerts, alert history, and alert silences.)

Prometheus

The Prometheus plug-in to the Dashboard facilitates the collection and visualization of Ceph performance metrics by enabling the export of performance counters directly from `ceph-mgr`. `ceph-mgr` gathers MMgrReport messages from all MgrClient processes, including Monitors and OSDs, containing performance counter schema and data. These messages are stored in a circular buffer, maintaining a record of the last N samples for analysis.

This plug-in establishes an HTTP endpoint, akin to other Prometheus exporters, and retrieves the most recent sample of each counter on polling, or *scraping* in Prometheus parlance. The HTTP path and query parameters are disregarded, and all available counters for all reporting entities are returned in the Prometheus text exposition format (for more information, see the Prometheus documentation). By default, the module accepts HTTP requests on TCP port 9283 on all IPv4 and IPv6 addresses on the host.

The Prometheus metrics can be viewed in Grafana from an HTTP web browser on TCP port 3000 (for example, `https://ceph-mgr-node:3000`).

Ceph CLI tool

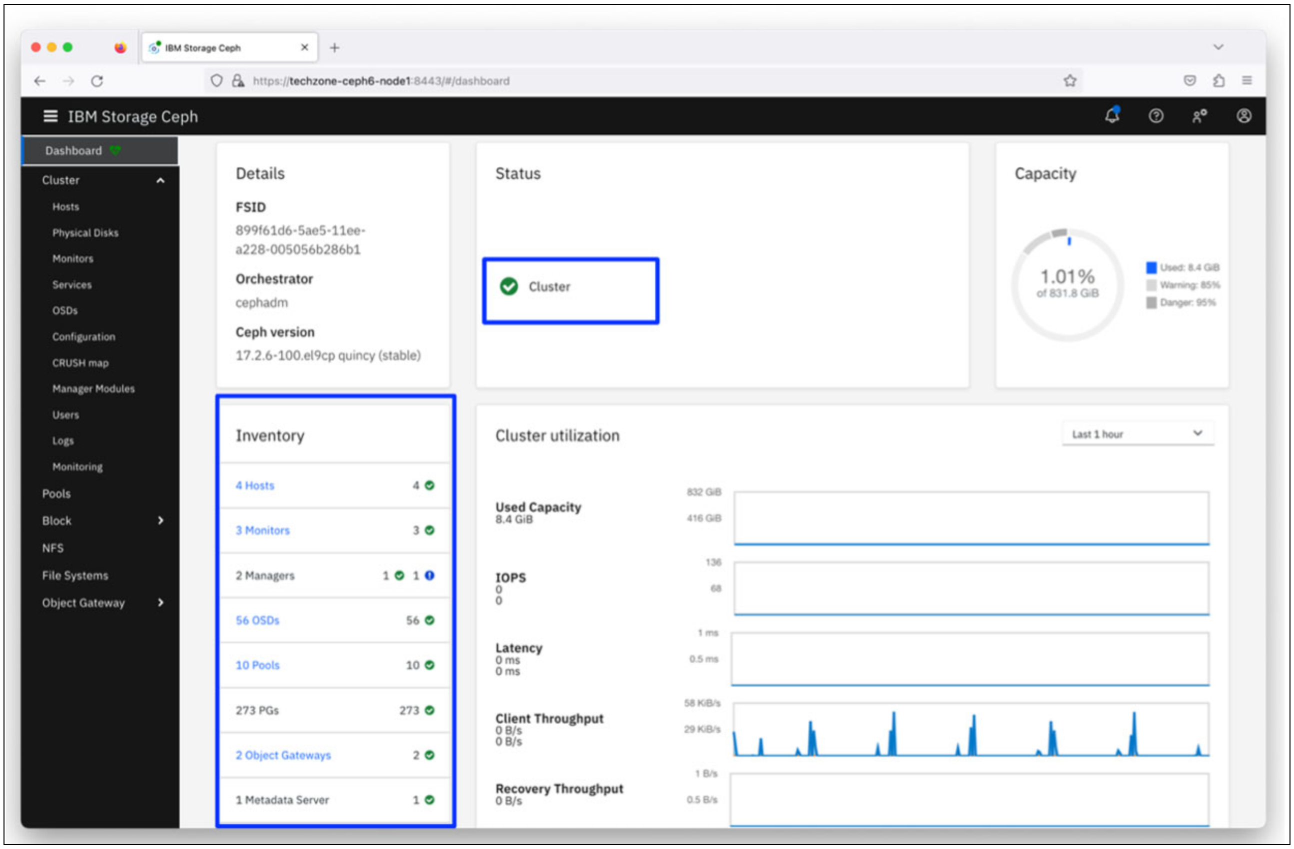
The Ceph CLI tool **`ceph`** offers the deepest levels of inspection and granularity to monitor all areas of the cluster. In addition to viewing performance metrics, it offers the ability to control daemons and adjust tuning parameters to optimize Ceph performance for client workloads.

5.2 IBM Storage Ceph monitoring examples

The following section offers specific examples of Ceph monitoring tools for the most common operational tasks.

5.2.1 Ceph Dashboard health

The Dashboard offers a convenient at-a-glance view into the configuration and the health of the cluster. In addition to the Dashboard home page, deeper inspection is available by browsing to relevant Cluster subcategories (for example, **Cluster** → **Monitoring**).



FSID
899f61d6-5ae5-11ee-a228-005056b286b1

Orchestrator
cephadm

Ceph version
17.2.6-100.el9cp quincy (stable)

✓ Cluster

1.01%
of 831.8 GiB

Used: 8.4 GiB
Warning: 85%
Danger: 95%

4 Hosts	4 ✓
3 Monitors	3 ✓
2 Managers	1 ✓ 1 ●
56 OSDs	56 ✓
10 Pools	10 ✓
273 PGs	273 ✓
2 Object Gateways	2 ✓
1 Metadata Server	1 ✓

Used Capacity
8.4 GiB

IOPS
0
0

Latency
0 ms
0 ms

Client Throughput
0 B/s
0 B/s

Recovery Throughput
0 B/s

Figure 5-1 Ceph Dashboard home page and health status

5.2.2 Ceph CLI health check

Although the Ceph Dashboard provides a convenient, at-a-glance overview of the cluster's configuration and health, the Ceph CLI (`cephadm`) empowers administrators to delve into the intricate details of all Ceph services and resources.

The **ceph status** and **ceph health** commands report the health of the cluster. If the cluster health status is HEALTH_WARN or HEALTH_ERR, the operator can use the **ceph health detail** command to view health check messages to troubleshoot the issue (Example 5-1).

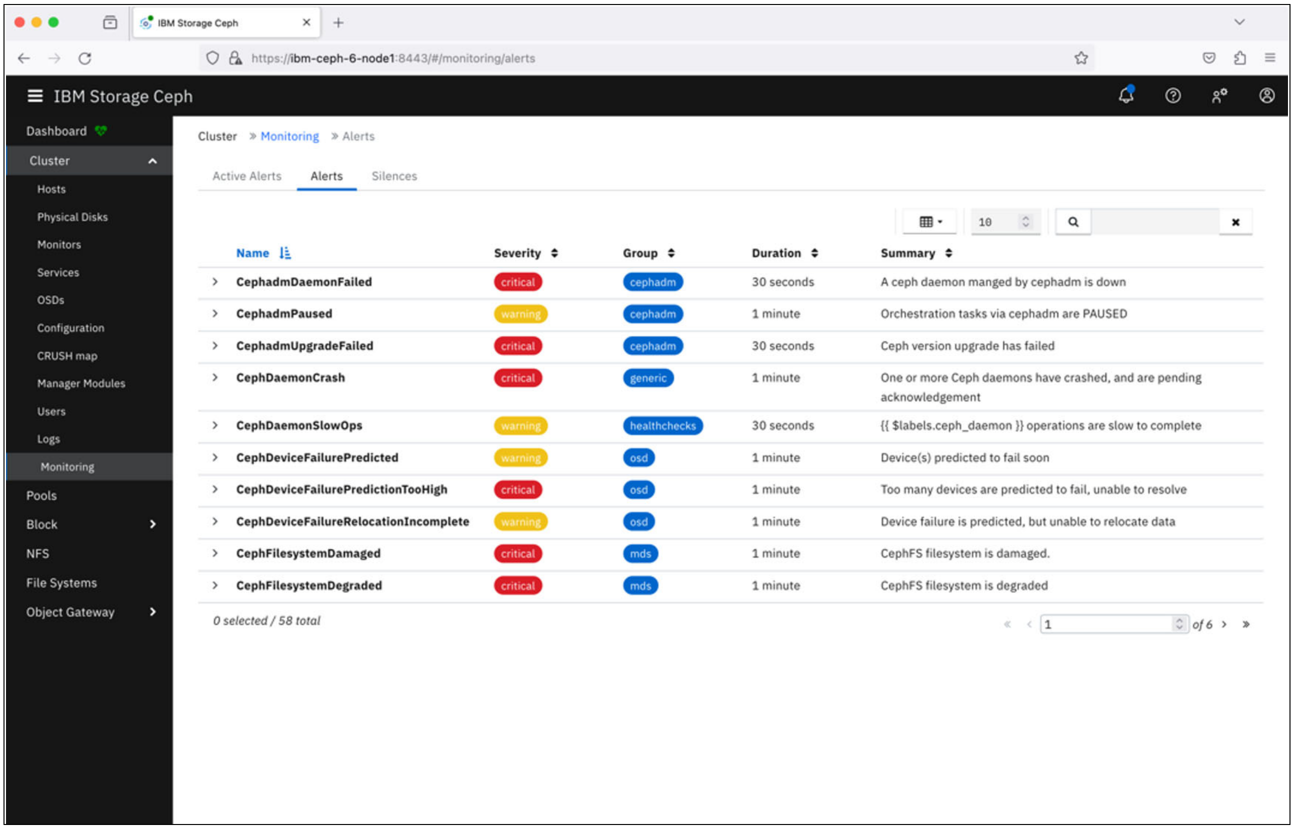
Example 5-1 Ceph command-line health and status examples

```
[root@node1 ~]# cephadm shell
[ceph: root@node1 /]# ceph health
HEALTH_OK

[ceph: root@node1 /]# ceph health detail
HEALTH_WARN failed to probe daemons or devices; 1/3 mons down
. . . output omitted . . .

[ceph: root@node1 /]# ceph status

  cluster:
    id:     899f61d6-5ae5-11ee-a228-005056b286b1
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum
techzone-ceph6-node1,techzone-ceph6-node2,techzone-ceph6-node3 (age 16h)
    mgr: techzone-ceph6-node1.jqdquv(active, since 2w), standbys:
techzone-ceph6-node2.akqefd
    mds: 1/1 daemons up
    osd: 56 osds: 56 up (since 2w), 56 in (since 4w)
    rgw: 2 daemons active (2 hosts, 1 zone)

  data:
    volumes: 1/1 healthy
    pools:   10 pools, 273 pgs
    objects: 283 objects, 1.7 MiB
    usage:   8.4 GiB used, 823 GiB / 832 GiB avail
    pgs:     273 active+clean
```

5.2.3 Ceph Dashboard alerts and logs

The Dashboard also offers views of cluster alerts and logs for a “point and click” inspection into the active alerts, alerts history, and alert silences. (For example, **Cluster** → **Monitoring** → **Alerts and Cluster** → **Logs** → **Audit Logs**).

Figure 5-2 shows the Ceph Dashboard alerts history.



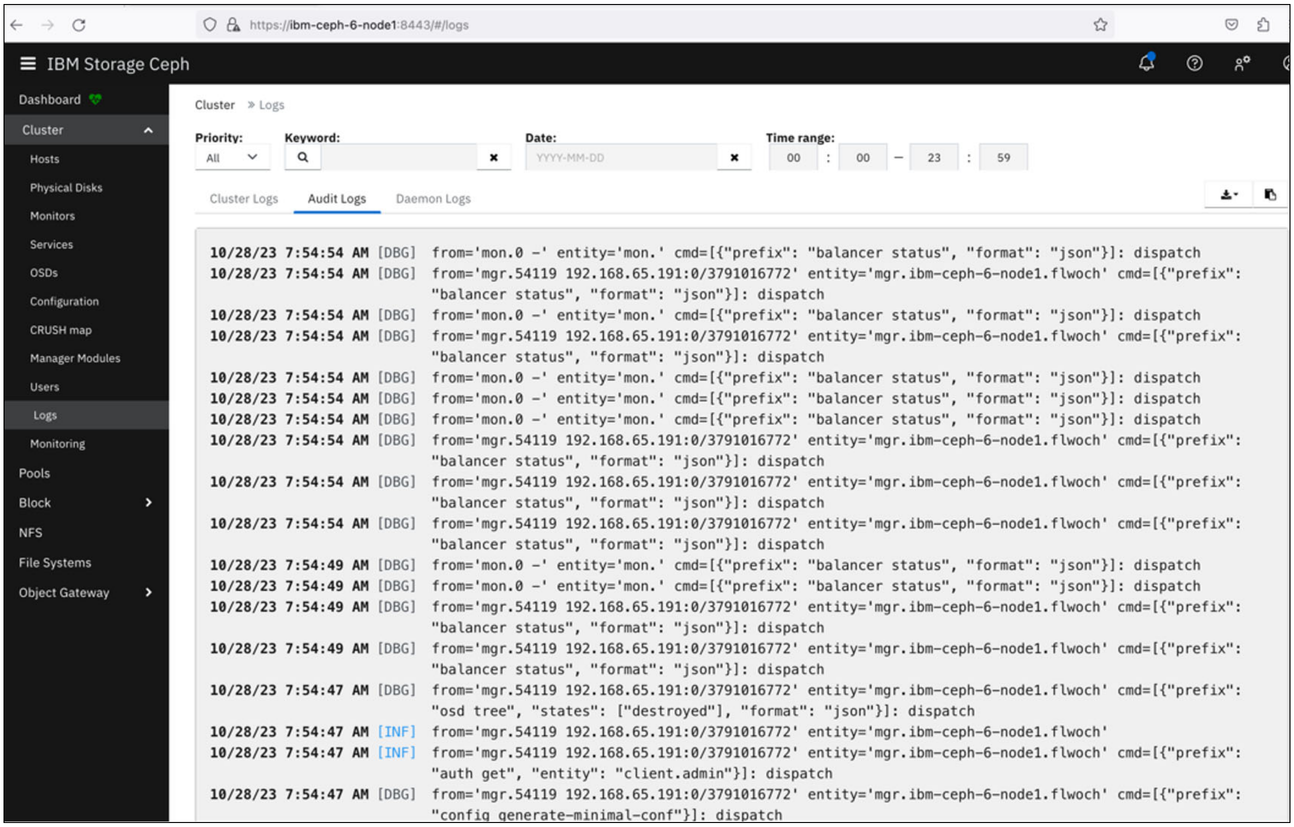
Name	Severity	Group	Duration	Summary
CephadmDaemonFailed	critical	cephadm	30 seconds	A ceph daemon manged by cephadm is down
CephadmPaused	warning	cephadm	1 minute	Orchestration tasks via cephadm are PAUSED
CephadmUpgradeFailed	critical	cephadm	30 seconds	Ceph version upgrade has failed
CephDaemonCrash	critical	generic	1 minute	One or more Ceph daemons have crashed, and are pending acknowledgement
CephDaemonSlowOps	warning	healthchecks	30 seconds	{{ $labels.ceph_daemon }} operations are slow to complete
CephDeviceFailurePredicted	warning	osd	1 minute	Device(s) predicted to fail soon
CephDeviceFailurePredictionTooHigh	critical	osd	1 minute	Too many devices are predicted to fail, unable to resolve
CephDeviceFailureRelocationIncomplete	warning	osd	1 minute	Device failure is predicted, but unable to relocate data
CephFilesystemDamaged	critical	mds	1 minute	CephFS filesystem is damaged.
CephFilesystemDegraded	critical	mds	1 minute	CephFS filesystem is degraded

0 selected / 58 total

1 of 6

Figure 5-2 Ceph Dashboard alerts history

Figure 5-3 shows the Ceph Dashboard audit logs.



Priority: All
Keyword:
Date: YYYY-MM-DD
Time range: 00 : 00 – 23 : 59

```
10/28/23 7:54:54 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:54 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:49 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:49 AM [DBG]  from='mon.0 -' entity='mon.' cmd=[{"prefix": "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:49 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:49 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "balancer status", "format": "json"}]: dispatch
10/28/23 7:54:47 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "osd tree", "states": ["destroyed"], "format": "json"}]: dispatch
10/28/23 7:54:47 AM [INF]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch'
10/28/23 7:54:47 AM [INF]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "auth get", "entity": "client.admin"}]: dispatch
10/28/23 7:54:47 AM [DBG]  from='mgr.54119 192.168.65.191:0/3791016772' entity='mgr.ibm-ceph-6-node1.flwoch' cmd=[{"prefix":
                           "config generate-minimal-conf"}]: dispatch
```

Figure 5-3 Ceph Dashboard audit logs

5.2.4 Ceph Dashboard plug-in

The Dashboard offers a convenient at-a-glance view into context-sensitive performance metrics. If a Grafana view is available for a property page, the Dashboard displays an **Overall Performance** tab on that selected page. Clicking this tab displays a context-sensitive view into the current performance for the chosen service or resource (Figure 5-4 on page 117).

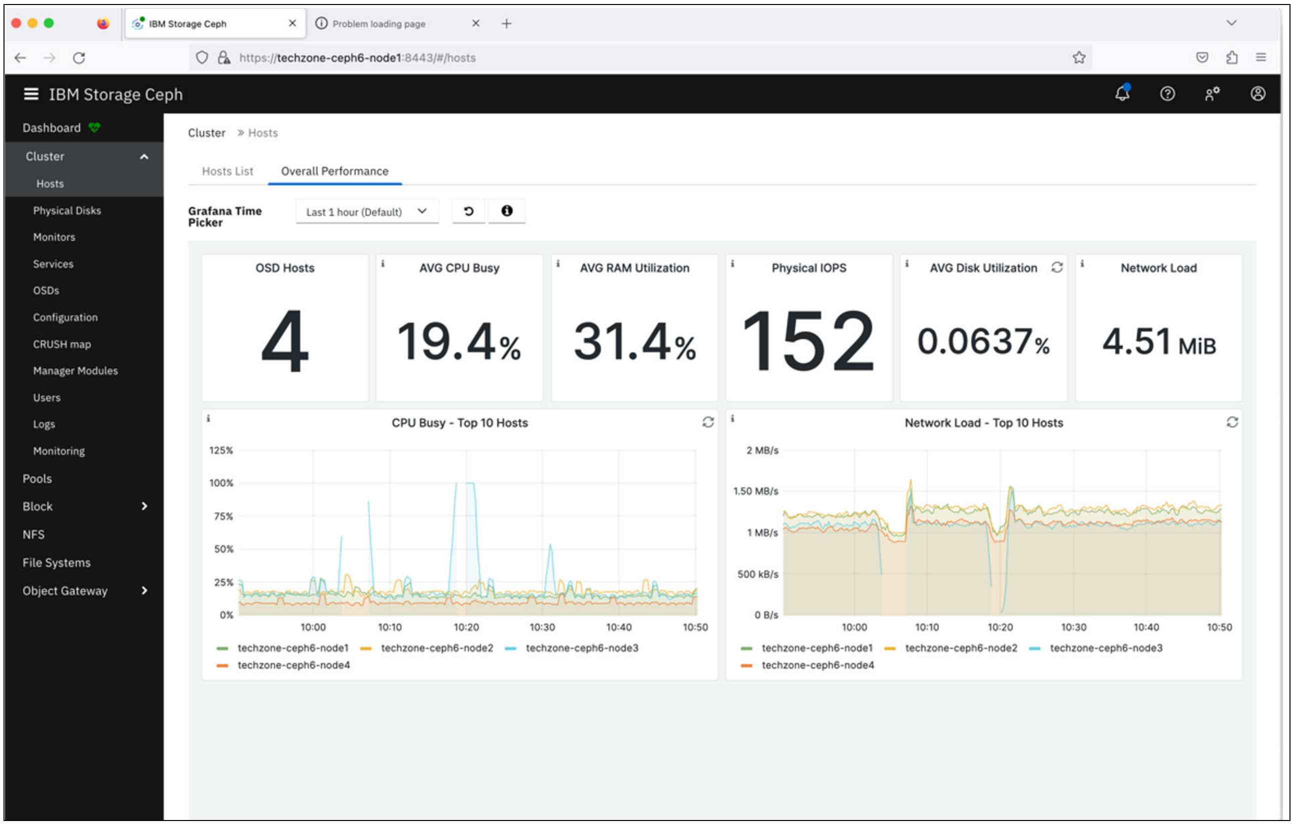


Figure 5-4 Ceph Dashboard plug-in for Grafana context-sensitive performance charts

5.2.5 Grafana stand-alone dashboard

The Grafana stand-alone Dashboard offers a more in-depth view into Ceph cluster performance metrics. The stand-alone Grafana deployment offers more than a dozen pre-defined dashboards that can be viewed in a dedicated, full-screen experience.

The Grafana stand-alone Dashboard can be accessed from a web browser that addresses TCP port 3000 on the Ceph MGR node (for example, `https://ceph-node1:3000`) (Figure 5-5).

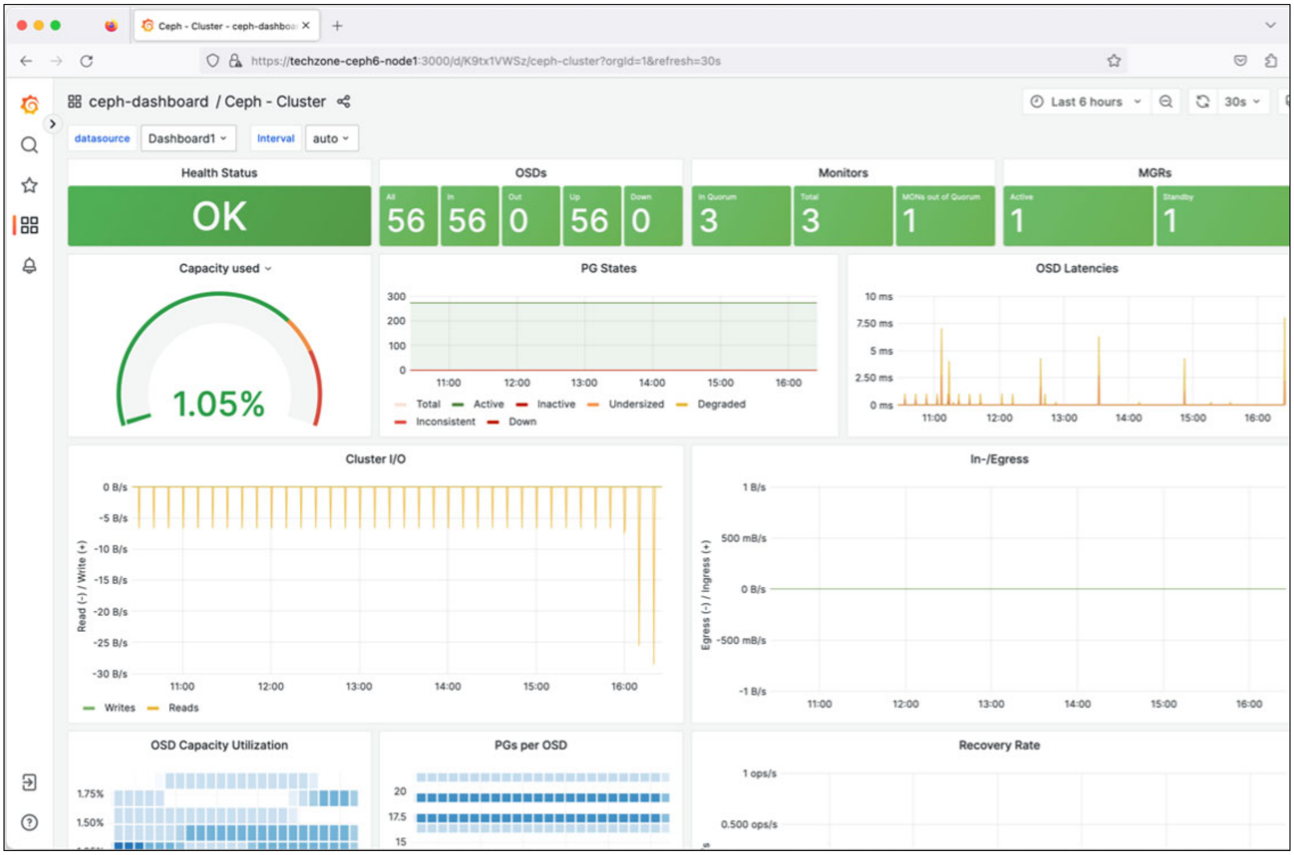


Figure 5-5 The Grafana stand-alone Dashboard cluster view

5.2.6 Ceph CLI deeper dive

The following examples illustrate some of the most frequently used **cephadm** commands for monitoring or querying the configuration and health of Ceph cluster resources.

Ceph service status

This command summarizes the services running in the Ceph. To view details about a particular service, use the command **ceph orch [service] ls**. Selected examples of **cephadm** commands follow.

Example 5-2 shows the **cephadm shell** command.

Example 5-2 Ceph service status

```
[root@node1 ~]# cephadm shell
[ceph: root@node1 /]#
NAME         PORTS     RUNNING REFRESHED AGE PLACEMENT
alertmanager   ?:9093,9094    1/1  8m ago    4w  count:1
ceph-exporter              4/4  8m ago    4w  *
crash                    4/4  8m ago    4w  *
grafana      ?:3000       1/1  8m ago    4w  count:1
mds.mycephfs                          1/1  6m ago     2w
techzone-ceph6-node3;count:1
```

```
mgr                   2/2  8m ago   4w  count:2
mon                                3/3  8m ago     3w
techzone-ceph6-node1;techzone-ceph6-node2;techzone-ceph6-node3
node-exporter   ?:9100      4/4  8m ago   4w  *
osd                     1  8m ago   -   <unmanaged>
osd.all-available-devices      55  8m ago   4w  *
prometheus     ?:9095      1/1  8m ago   4w  count:1
rgw.s3service     ?:80            2/2  7m ago     2w
techzone-ceph6-node2;techzone-ceph6-node3;count:2
```

Ceph host status

This command displays all notes in the cluster and their status (Example 5-3).

Example 5-3 Ceph host status

```
[ceph: root@node1 /]# ceph orch host ls
HOST                  ADDR            LABELS  STATUS
techzone-ceph6-node1  192.168.65.161  _admin
techzone-ceph6-node2  192.168.65.162
techzone-ceph6-node3  192.168.65.163
techzone-ceph6-node4  192.168.65.164
4 hosts in cluster
```

Ceph device status

This command lists all OSD devices on each host in the system, along with their size and availability status. In Example 5-4, the available status of the devices is "No," which indicates that these devices are normal ones that have already been integrated into the cluster.

Example 5-4 Ceph device status

```
[ceph: root@node1 /]# ceph orch device ls
HOST                  PATH      TYPE  DEVICE ID   SIZE  AVAILABLE  REFRESHED
REJECT REASONS
techzone-ceph6-node1  /dev/sdb  hdd             8589M  No         29m ago
Insufficient space (<10 extents) on vgs, LVM detected, locked
techzone-ceph6-node1  /dev/sdc  hdd             8589M  No         29m ago
Insufficient space (<10 extents) on vgs, LVM detected, locked
locked

. . . output omitted . . .

techzone-ceph6-node4  /dev/sdn  hdd             17.1G  No         104s ago
Insufficient space (<10 extents) on vgs, LVM detected, locked
techzone-ceph6-node4  /dev/sdo  hdd             17.1G  No         104s ago
Insufficient space (<10 extents) on vgs, LVM detected, locked
```

Ceph OSD status

This command displays all OSDs in the cluster with status, logical topology, and device (Example 5-5).

Example 5-5 Ceph OSD tree listing

```
[ceph: root@node1 /]# ceph osd tree
ID  CLASS  WEIGHT   TYPE  NAME          STATUS  REWEIGHT  PRI-AFF
-1         0.81091  root default
-3         0.20273      host techzone-ceph6-node1
 9    hdd  0.01559        osd.9          up      1.00000  1.00000
14    hdd  0.01559        osd.14         up      1.00000  1.00000
20    hdd  0.01559        osd.20         up      1.00000  1.00000
25    hdd  0.01559        osd.25         up      1.00000  1.00000

. . . output omitted . . .

-9         0.20273      host techzone-ceph6-node2
 1    ssd  0.00780        osd.1          up   1.00000  1.00000
 5    ssd  0.00780        osd.5          up   1.00000  1.00000
 2    ssd  0.00780        osd.2          up   1.00000  1.00000
 7    ssd  0.00780        osd.7          up   1.00000  1.00000
```

5.2.7 Ceph monitoring Day 2 operations

This section describes a few more commands that can be useful for Ceph Day 2 operations.

Ceph versions and status

Earlier versions of Ceph clients might not benefit from or be compatible with features provided by the version of Ceph running on cluster systems. When upgrading a Ceph cluster, it is a best practice to also update the clients. RGWs, kernel or FUSE mounts of CephFS file systems, librbd (virtual machines), and CLI tools are examples of Ceph clients.

Check the version of Ceph installed on client system (Example 5-6).

Example 5-6 Ceph client version report

```
[ceph: root@node1 /]# ceph -v
ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343) quincy
(stable)

[ceph: root@node1 /]# ceph version
ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343) quincy
(stable)
```

Check the running daemon versions on Ceph cluster nodes (Example 5-7). If the reported versions are not all identical, you may have encountered a snag during deployment or upgrade and should look and address the discrepancy to ensure smooth operation.

Example 5-7 Ceph cluster versions report

```
[ceph: root@node1 /]# ceph versions
{
    "mon": {
        "ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343)
quincy (stable)": 3
    },
    "mgr": {
        "ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343)
quincy (stable)": 2
    },
    "osd": {
        "ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343)
quincy (stable)": 56
    },
    "mds": {
        "ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343)
quincy (stable)": 1
    },
    "overall": {
        "ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343)
quincy (stable)": 64
    }
}
```

Ceph cluster safety check

Check the current OSD fullness ratios for your cluster (Example 5-8). Note that Ceph will not accept values that are not numerical order, for example, the full_ratio cannot be less than the nearfull_ratio. (Example 5-8).

Example 5-8 Checking the current OSD fullness ratios for your cluster

```
[ceph: root@ceph-node1 /]# ceph osd dump | grep ratio
full_ratio 0.95
backfillfull_ratio 0.9
nearfull_ratio 0.85
```

Change one of these settings (Example 5-9).

Example 5-9 Changing one of these ratios

```
[ceph: root@ceph-node1 /]# ceph osd set-full-ratio 0.9
osd set-full-ratio 0.9

[ceph: root@ceph-node1 /]# ceph osd dump | grep ratio
full_ratio 0.9
backfillfull_ratio 0.9
nearfull_ratio 0.85

[ceph: root@ceph-node1 /]# ceph osd set-full-ratio 0.95
osd set-full-ratio 0.9
```

```
[ceph: root@ceph-node1 /]# ceph osd dump | grep ratio
full_ratio 0.95
backfillfull_ratio 0.9
nearfull_ratio 0.85
```

Display current cluster space usage (Example 5-10).

Example 5-10 Displaying current cluster space usage

```
[ceph: root@ceph-node1 /]# ceph df
--- RAW STORAGE ---
CLASS     SIZE    AVAIL    USED  RAW USED  %RAW USED
hdd    384 GiB  384 GiB  15 MiB    15 MiB          0
TOTAL  384 GiB  384 GiB  15 MiB    15 MiB          0

--- POOLS ---
POOL                   ID  PGS  STORED  OBJECTS  USED  %USED  MAX AVAIL
device_health_metrics   1    1     0 B        0   0 B      0    173 GiB
```

Check each OSD's usage (Example 5-11).

Example 5-11 Checking each OSD's usage

```
[ceph: root@ceph01 /]# ceph osd df
ID  CLASS  WEIGHT   REWEIGHT  SIZE     RAW USE  DATA     OMAP  META     AVAIL
%USE  VAR   PGS  STATUS
 0    hdd  0.12500   1.00000  128 GiB  5.1 MiB  184 KiB   0 B  4.9 MiB  128 GiB
0.00  1.00    1      up
 1    hdd  0.12500   1.00000  128 GiB  5.1 MiB  184 KiB   0 B  4.9 MiB  128 GiB
0.00  1.00    1      up
 2    hdd  0.12500   1.00000  128 GiB    5 MiB  184 KiB   0 B  4.8 MiB  128 GiB
0.00  0.99    0      up
                       TOTAL  384 GiB   15 MiB  552 KiB   0 B   15 MiB  384 GiB
0.00
MIN/MAX VAR: 0.99/1.00  STDDEV: 0
```

Tip: When a CephOSD reaches the full ratio, it indicates that the device is nearly full and can no longer accommodate additional data. This condition can have several negative consequences for the Ceph cluster:

- Reduced performance: As an OSD fills up, its performance degrades. This situation can lead to slower read/write speeds, increased latency, and longer response times for client applications.
- Cluster rebalancing challenges: When an OSD reaches the full ratio, it can make it more difficult to rebalance the cluster. Rebalancing is the process of evenly distributing data across all OSDs to optimize performance and improve fault tolerance.

To avoid these consequences, monitor OSD usage and take proactive measures to prevent OSDs from reaching the full ratio. This task might involve adding OSDs to the cluster, increasing the capacity of existing OSDs, or deleting old or unused data.

5.3 Conclusion

The Ceph cluster maintains centralized cluster logs that capture high-level events and other information regarding the cluster. These events are stored on disk on Monitor nodes and can be accessed and monitored through various administrative tools.

Optional Ceph Manager modules include InfluxDB, Insights, Telegraph, Alerts, Disk Prediction, and iostat, and can help you integrate your IBM Storage Ceph cluster into existing monitoring and alerting systems. You can also use the SNMP Gateway service to enhance monitoring capabilities.

Administrators can leverage the Ceph Dashboard for a straightforward and intuitive view into the health of the Ceph cluster. The built-in Grafana dashboard presents detailed information, context-sensitive performance counters, and performance data for specific resources and services.

Administrators can also use cephadm, the stand-alone Grafana dashboards, and other third-party tools to visualize and record detailed metrics on cluster utilization and performance.

In conclusion, the overall performance of a software-defined storage solution like IBM Storage Ceph is heavily dependent on the network. Therefore, it is crucial to integrate the nodes and switches serving your IBM Storage Ceph cluster with existing network monitoring infrastructure. This includes tracking packet drops and other network errors and the health of network interfaces. The SNMP subsystem included in Red Hat Enterprise Linux can facilitate comprehensive monitoring.

5.4 References on the World Wide Web

You are invited to explore the wealth of Ceph documentation and resources that can be found on the World Wide Web. A concise selection of recommended reading is provided here:

- IBM Storage Ceph Documentation:

  https://www.ibm.com/docs/en/storage-ceph/6?topic=dashboard-monitoring-cluster

- Community Ceph Documentation

  https://docs.ceph.com/en/latest/monitoring/

6

Day 1 and Day 2 operations

This chapter describes the Day 1 and Day 2 operations of an IBM Storage Ceph environment.

This chapter has the following sections:

- Day 1 operations
- Day 2 operations

6.1 Day 1 operations

Initial operations, also known as Day 1 operations, include the following:

- Check deployment prerequisites.
- Bootstrap IBM Storage Ceph cluster.
- Deploy IBM Storage Ceph cluster services.
- Configure cluster (pools, placement groups (PGs), parameters, and other settings).

6.1.1 Prerequisites

This section describes IBM Ceph prerequisites.

Server node requirements

Although basic proofs-of-concept can be implemented by using a single node or virtual machine (VM), production environments require a more robust architecture. For production deployments that use **cephadm**, IBM Storage Ceph mandates at a minimum the following components:

- At least three Monitors.
- At least three object storage daemon (OSD) hosts that use directly attached storage (SAN LUNs are not supported as OSD devices).

In addition, you must configure the following components:

- At least two Manager daemons on separate nodes.
- At least as many OSDs as the number of data replicas that are configured.
- At least two, identically configured Metadata Server (MDS) daemons on separate nodes, if you are deploying CephFS file systems.
- At least two RGW daemons on separate nodes, if you are using Ceph Object Gateway. .

For more information about the server sizing and configuration, see Chapter 4, “Sizing IBM Storage Ceph” on page 83.

Servers must be configured with the following tools:

- Python 3
- Systemd
- Podman
- Time synchronization (Chrony)
- Supported operating system (RHEL)

cephadm requires password-less SSH capabilities. DNS resolution must be configured for all nodes. OSD nodes must have all the tools in the preceding list plus LVM2.

Firewall requirements

Table 6-1 on page 127 lists the TCP ports that IBM Storage Ceph uses.

Table 6-1 TCP ports that IBM Storage Ceph uses

Service	Ports	Description
MON	6789/TCP and 3300/TCP	N/A
MGR	8443/TCP SSL 8080/TCP SSL Disabled 8003/TCP 9283/TCP	Default Dashboard port Default RESTful port Default Prometheus port
OSD/MDS	6800 - 7300/TCP	OSD ports
RADOS Gateway (RGW)	7480/TCP Civetweb 80/TCP Beast	Default RGW ports

Miscellaneous requirements

If you configure SSL for your Dashboard access and your object storage endpoint (RGW), ensure that you obtain the correct certificate files from your security team and deploy them where needed.

Ensure that your server network and configuration are adequate for the workload served by your IBM Storage Ceph cluster:

- Size your network cards, interfaces, and switches for the throughput that you must deliver. Evaluate the network bandwidth required for data protection because it must be carried by the network for all write operation.
- Ensure that your network configuration does not present single points of failure.
- If your cluster spans multiple subnets, ensure that each server can communicate with the other servers.

6.1.2 Deployment

The IBM Storage Ceph documentation describes how to use `cephadm` to deploy your Ceph cluster. A `cephadm` based deployment has the following steps:

- For beginners:
  a. Bootstrap your Ceph cluster (create one initial Monitor and one Manager).
  b. Add services to your cluster (OSDs, MDSs, RGWs, and other services).
- For advanced users: Bootstrap your cluster with a complete service file to deploy everything.

Cephadm

The only supported tool for IBM Storage Ceph, `cephadm` has been available since the Ceph upstream Octopus release, and has been the default deployment tool since Pacific.

Service files

An entire cluster can be deployed by using a single service file. The service file follows the syntax in Example 6-1.

Example 6-1 Service file format

```
service_type: {type_value}
service_name: {name_value}
addr: {address_value}
hostname: {hostname}
{options}
```

The **`service_type`** accepts the following values:

- `host` to declare hosts.
- `crash` to place the Ceph crash collection daemon.
- `grafana` to place the Grafana component.
- `node-exporter` to place the exporter component.
- `prometheus` to place the Prometheus component.
- `mgr` to place the Manager daemons.
- `mon` to place the Monitor daemons.
- `osd` to place the Object Storage daemons.
- `rgw` to place the RGW daemons.
- `mds` to place the MDS daemons.
- `ingress` to place the load balancer (haproxy).

You can assign labels to hosts by using the `labels:` field of a `host` service file (Example 6-2).

Example 6-2 Service file with label

```
service_type: host
addr: {host_address}
hostname: {host_name}
labels:
- xxx
- yyy
...
```

You can assign specific placement for a service by using the `placement:` field in a service file. For more information, see “Placement” on page 130.

Example 6-3 Service file with placement

```
service_type: {service_type}
service_name: {service_name}
placement:
  host_pattern: '*'
```

An OSD service file offers many specific options to specify how the OSDs should be deployed:

- `block_db_size` to specify the RocksDB size on separate devices.
- `block_wal_size` to specify the RocksDB size on separate devices.
- `data_devices` to specify which devices receive file data.
- `db_devices` to specify which devices receive the RocksDB data.
- `wal_devices` to specify which devices receive the write-ahead log (WAL).

- db_slots to specify how many RocksDB DB partitions per db_device.
- wal_slots to specify how many RocksDB WAL partitions per wal_device.
- data_directories to specify a list of device paths to use.
- filter_logic to specify OR or AND relationships among filters. The default is AND.
- objectstore to specify the OSD back-end type (currently only bluestore or filestore).
- crush_device_class to specify the CRUSH device class.
- data_allocate_fraction to specify a portion of a drive for data devices.
- osds_per_device to specify how many OSDs to deploy per device (default is 1).
- osd_id_claims to specify how OSD IDs should be preserved per node (true or false).

The **data_devices**, **db_devices**, and **wal_devices** parameters accept the following arguments:

- **all** to specify that all devices are to be consumed (true or false).
- **limit** to specify how many OSD to deploy per node.
- **rotational** to specify the type of devices to select (0 for SSDs, 1 for HDDs).
- **size** to specify the size of the devices to select:
  - xTB to select a specific device size.
  - xTB:yTB to select devices between the two capacities.
  - :xTB to select any device up to this size.
  - xTB: to select any device at least this size.
  - **path** to specify the device path to use.
- **model** to specify the manufacturer name..
- **vendor** to specify the vendor model name.
- **encrypted** to specify whether the data will be encrypted at rest (**data_devices** only).

The RGW service file accepts the following specific arguments:

- **networks** to specify which CIDR the gateway binds to.
- **spec**:
  - **rgw_frontend_port** to specify the TCP port to which the gateway binds.
  - **rgw_realm** to specify the realm for this gateway.
  - **rgw_zone** to specify the zone for this gateway.
  - **ssl** to specify whether this gateway uses SSL (true or false).
  - **rgw_frontend_ssl_certificate** to specify the certificate to use.
  - **rgw_frontend_ssl_key** to specify the key to use.
  - **rgw_frontend_type** to specify the front end to use (the default is beast).
- **placement.count_per_host** to specify how many RGWs to deploy per node.

Container parameters

You can customize the parameters that are used by the Ceph containers by using a special section of your service file that is known as extra_container_args. To add extra parameters, use the template that is shown in Example 6-4 in the appropriate service files.

Example 6-4 Passing container extra parameters

```
service_type: {service_type}
service_name: {service_name}
placement:
  host_pattern: '*'
extra_container_args:
  - "--cpus=2"
```

Placement

Placement can be a simple count to indicate the number of daemons to deploy. In such a configuration, **cephadm** chooses where to deploy the daemons.

Placement can use explicit naming: `--placement="host1 host2 …"`. In such a configuration, the daemons are deployed on the nodes that are listed.

Placement can use labels: `--placement="label:mylabel"`. In such a configuration, the daemons are deployed on the nodes that match the provided label.

Placement can use expressions: `--placement="host[1-5]"`. In such a configuration, the daemons are deployed on the nodes that match the provided expression.

Using a service file, you encode the count as shown in Example 6-5.

Example 6-5 Service file with placement count

```
service_type: rgw
placement:
  count: 3
```

Using a service file, you encode the label as shown in Example 6-6.

Example 6-6 Service file with placement label

```
service_type: rgw
placement:
  label: "mylabel"
```

Using a service file, you encode the host list as shown in Example 6-7.

Example 6-7 Service file with placement host list

```
service_type: rgw
placement:
  hosts:
    - host1
    - host2
```

Using a service file, you encode the pattern as shown in Example 6-8.

Example 6-8 Service file with placement pattern

```
service_type: rgw
placement:
  host_pattern: "ceph-server-[0-5]"
```

Minimal cluster (bootstrapping)

Initial cephadm deployment always starts with what is known as cluster bootstrapping. To begin cluster bootstrapping, install the **cephadm** binary file on a node and run the following command:

```
$ cephadm bootstrap --mon-ip {monitor_ip_address}
```

This command performs the following actions:

- Creates an initial Monitor daemon.
- Creates an initial Manager daemon.
- Generates a **cephadm** SSH key.
- Adds the **cephadm** SSH key to `~/.ssh/authorized_keys`.
- Writes a copy of the public key to `/etc/ceph`.
- Generated a minimal `/etc/ceph/ceph.conf` file.
- Creates the `/etc/ceph/ceph.client.admin.keyring` file.

You can pass an initial Ceph configuration file to the **`bootstrap`** command by using the **`--config {path_to_config_file}`** option.

You can override the SSH user that is used by **`cephadm`** by using the **`--ssh-user {user_name}`** option.

You can pass a specific set of container registry parameters in a valid registry JSON file by using the `--registry-json {path_to_registry_json}` option.

You can choose the Ceph container image to deploy by using the `--image {registry}[:{port}]/{imagename}:{imagetag}` option.

You can specify the network configuration that the cluster uses. A Ceph cluster uses up to two networks:

- Public network,
  - UUsed by clients (including RGWs) to connect to Ceph daemons.
  - Used by Monitors to converse with other daemons.
  - Used by Managers and MDSs to communicate with other daemons.
- Cluster network: Used by OSDs to perform OSD operations, including heartbeats, replication, and recovery. If a cluster network is not defined, these operations use the public network.

Figure 6-1 shows the Ceph network layout.

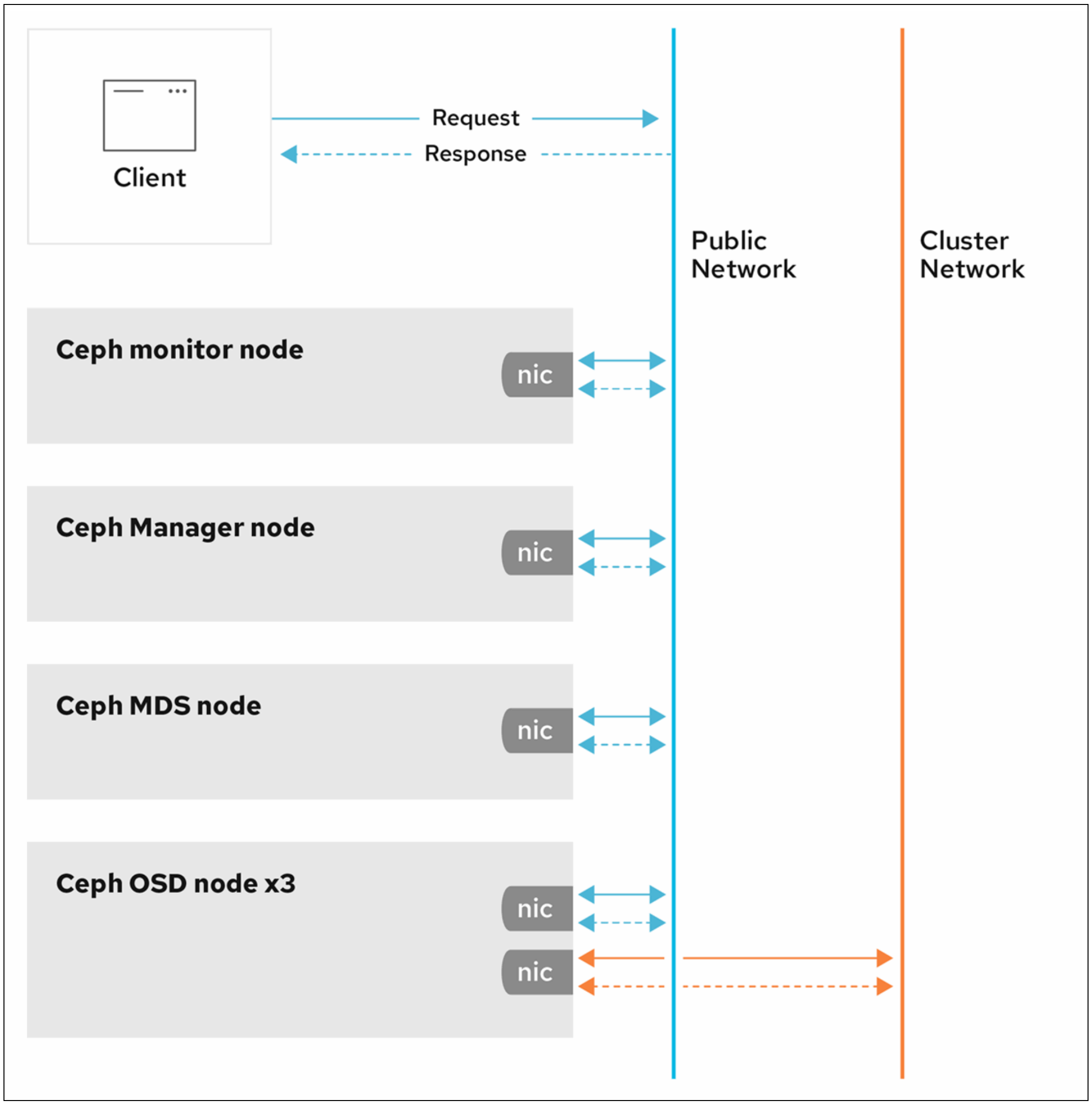


Figure 6-1 Ceph network layout

The public network is extrapolated from the **`--mon-ip`** parameter provided to the **`bootstrap`** command. The cluster network can be specified with the **`--cluster-network`** parameter. If the **`--cluster-network`** parameter is not specified, it is set to the same value as the public network.

Tip: Add the **`--single-host-defaults`** argument to your **`bootstrap`** command to bootstrap a test cluster that uses a single node.

To connect to your initial cluster, run the following command:

```
$ cephadm shell
```

To connect to the Dashboard GUI, look for the following lines in the **`cephadm`** bootstrap output and point your HTTP browser to the URL that is displayed. (Example 6-9 on page 133).

Example 6-9 Lines in the cephadm bootstrap output

```
Ceph Dashboard is now available at:
         URL: https://{initial_node_name}:8443/
        User: admin
    Password: {generated_password}
```

This document is not designed to provide extensive details about the deployment and configuration of your cluster. For more information, see IBM Storage Ceph documentation.

IBM Storage Ceph Solutions Guide, REDP-5715 provides GUI-based deployment methods for users who want to use a GUI instead of the CLI.

After the cluster is successfully bootstrapped you can deploy additional cluster services. A production cluster requires the following elements to be deployed in appropriate numbers for a reliable cluster that does not present any single point of failure:

- Monitors
- Managers
- OSDs

Adding nodes

When the cluster is bootstrapped, the Ceph administrator must add each node where services will be deployed. Copy the cluster public SSH key to each node that will be part of the cluster. When the nodes are prepared, add them to the cluster (Example 6-10).

Example 6-10 Adding nodes

```
# ssh-copy-id -f -I /etc/ceph/ceph.pub root@ceph02
# ssh-copy-id -f -I /etc/ceph/ceph.pub root@ceph03
# ceph orch host add ceph02 10.10.0.102
# ceph orch host add ceph02 10.10.0.103 --labels=label1,label2
```

Tip: To add multiple hosts, use a service file that describes all your nodes.

Removing nodes

If you must remove a node that was added by mistake, run the commands in Example 6-11. If the node does not have OSD already deployed, skip the second command of the example.

Example 6-11 Removing nodes

```
# ceph orch host drain {hostname}
# ceph orch osd rm status
# ceph orch ps {hostname}
# ceph orch rm {hostname}
# ssh root@{hostname} rm -R /etc/ceph
```

Tip: You can also clean up the SSH keys that are copied to the host.

Assigning labels

You can assign labels to hosts after they are added to the cluster (Example 6-12).

Example 6-12 Assigning a label after a node addition

```
# ceph orch host label add {hostname} {label}
```

Adding services

When the cluster is bootstrapped, the Ceph administrator must add the required services for the cluster to become fully operational.

After bootstrapping your cluster, you have only a single Monitor, a single Manager, no OSDs, no MDSs, and no RGWs. These services are deployed in the following order:

1. Deploy at least another two Monitors for a total of three or five.
2. Deploy at least one more Manager.
3. Deploy the OSDs.
4. If needed, deploy CephFS MDS daemons.
5. If needed, deploy RGWs.

Monitors

To deploy a total of three Monitors, deploy two more (Example 6-13).

Example 6-13 Deploying a full Monitor quorum

```
# ceph orch daemon add mon --placement="{ceph02},{ceph03}"
```

Tip: This task can also be achieved by using a service file and running the following command:

```
ceph orch apply -i {path_to_mon_service_file}
```

Tip: If you want your MON to bind to a specific IP address or subnet, use `{hostname}:{ip_addr}` or `{hostname}:{cidr}` to specify the host.

Managers

To deploy a total of two Managers, deploy one more (Example 6-14).

Example 6-14 Deploying highly available Managers

```
# ceph orch daemon add mgr --placement="{ceph02}"
```

Object storage daemons

The only additional requirement for a fully operational cluster is to deploy OSDs on at least three separate nodes. At least three nodes that you add to the cluster must have at least one available local device each that can be used to deploy OSDs.

List the devices that are available on all nodes by using the command in Example 6-15 after you add all the nodes to your cluster.

Example 6-15 Listing devices

```
# ceph orch device ls
```

`cephadm` scans all nodes for the available devices (free of partitions, formatting, LVM configuration) (Example 6-16).

Example 6-16 Deploying OSDs on all available device

```
# ceph orch apply osd --all-available-devices
```

Tip: For a production cluster with specific configuration and strict deployment scenarios, run the following command:

```
service ceph orch apply -i {path_to_osd_service_file}
```

To visualize what devices will be consumed by this command, run the following command:

```
ceph orch apply osd --all-available-devices --dry-run
```

An OSD service file, using the information that is provided in 6.1.2, “Deployment” on page 127, can be tailored to each node’s needs. As such, an OSD service file can contain multiple specifications (Example 6-17).

Example 6-17 Multiple specification OSD service file

```
service_type: osd
service_id: osd_spec_hdd
placement:
  host_pattern: '*'
spec:
  data_devices:
    rotational: 1
  db_devices:
    model: MC-55-44-XZ # This model is identified as an HDD
    limit: 2
---
service_type: osd
  service_id: osd_spec_ssd
placement:
  host_pattern: '*'
spec:
  data_devices:
      rotational: 0
    model: MC-55-44-XZ # This model is identified as an SDD
```

You can also add an OSD that uses a specific device (Example 6-18).

Example 6-18 Deploying an OSD by using a specific device

```
# ceph orch daemon add osd {hostname}:data_devices={dev1}
```

Tip: You can pass many parameters through the CLI, including **`data_devices`**, **`db_devices`**, and **`wal_devices`**. For example:

```
{hostname}:data_devices={dev1},{dev2},db_devices={db1}
```

If an OSD does not deploy on a given device it might be necessary to initialize it so that it can be consumed by an OSD (Example 6-19).

Example 6-19 Initializing disk devices

```
# ceph orch device zap {hostname} {device_path}
```

6.1.3 Initial cluster configuration

This section describes the initial cluster configuration.

Ceph RADOS Gateway

The deployment of an RGW service to support the Swift and S3 APIs follows the same model as for other components due to the **cephadm** standardized processes (Example 6-20).

Example 6-20 Deploying the RADOS Gateway service

```
# ceph orch apply rgw {service_name}
```

Arguments can be passed through CLI arguments or provided by a service file containing a detailed configuration.

> **Tip:** You can pass many parameters through the CLI, including **--realm={realm_name}**, **--zone={zone_name} --placement={placement_specs}**, **--rgw_frontend_port={port}**, or **count-per-host:{n}**.

Ingress service

If your cluster deploys an RGW serviced, you need a load balancer in front of the RGW service to ensure the distribution of the traffic between multiple RGWs and provide a highly available (HA) service endpoint with no single point of failure.

The ingress service provides the following parameters:

- **backend_service** to specify the RGW service name to the front end.
- **virtual_ip** to specify the CIDR that the ingress service binds to.
- **frontend_port** to specify the port that the ingress service binds to.
- **monitor_port** to specify the port where the LB status is maintained.
- **virtual_interface_networks** to specify a list of CIDRs.
- **ssl_cert** to specify the SSL certificate and key.

These parameters can be used in an ingress service file (Example 6-21).

Example 6-21 Creating an ingress service file for RGW

```
service_type: ingress
service_id: rgw.default
placement:
  hosts:
    - ceph01
    - ceph02
    - ceph03
spec:
  backend_service: rgw.default
  virtual_ip: 10.0.1.0/24
  frontend_port: 8080
  monitor_port: 1900
  ssl_cert: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    -----BEGIN PRIVATE KEY-----
    ...
    -----END PRIVATE KEY-----
```

Tip: For a production cluster with a specific configuration and strict deployment scenarios, run the following command:

```
service ceph orch apply -i {path_to_mds_service_file}
```

When the MDSs are active, manually create the file system by running the following command:

```
ceph fs new {fs_name} {meta_pool} {data_pool}
```

When your cluster is fully deployed, a best practice is to export the cluster configuration and back up the configuration. Use **`git`** to manage the cluster configuration (Example 6-22).

Example 6-22 Exporting the cluster configuration

```
# ceph orch ls --export
service_type: alertmanager
service_name: alertmanager
placement:
  count: 1
---
service_type: crash
service_name: crash
placement:
  host_pattern: '*'
---
service_type: grafana
service_name: grafana
placement:
  count: 1
---
service_type: mds
service_id: myfs
service_name: mds.myfs
placement:
  count: 1
  hosts:
  - ceph01
---
service_type: mgr
service_name: mgr
placement:
  count: 2
---
service_type: mon
service_name: mon
placement:
  count: 5
---
service_type: node-exporter
service_name: node-exporter
placement:
  host_pattern: '*'
---
service_type: osd
service_id: all-available-devices
```

```
service_name: osd.all-available-devices
placement:
  host_pattern: '*'
spec:
  data_devices:
    all: true
  filter_logic: AND
  objectstore: bluestore
---
service_type: prometheus
service_name: prometheus
placement:
  count: 1
---
service_type: rgw
service_id: myrgw
service_name: rgw.myrgw
placement:
  count: 1
  hosts:
  - ceph01
```

Tip: Redirect the command output to a file to create a full cluster deployment service file.

Pools and placement groups

When the cluster is deployed, you might need to create additional pools or customize the pools that were created automatically by services deployed by **cephadm**.

In a production cluster, it is best for each OSD to serve 100 - 200 PGs replicas.

The Placement Group Autoscaler automatically adjust the number of PGs within each pool based on the **target size ratio** that is assigned to each pool.

Controlled Replication Under Scalable Hashing

After the cluster is deployed, you might need to customize the CRUSH configuration to align it with your physical infrastructure. By default, the failure domain is the host, but you might prefer to set it to rack, row, or other datacenter failure domain.

6.2 Day 2 operations

After the deployment and configuration of your cluster, additional operations will be required. They are known as *Day 2 operations*, and include the following elements:

- Configuring logging to file.
- Upgrading your cluster.
- Expanding your cluster for more performance or capacity.
- Replacing a failed node.
- Replacing a failed disk.
- Redeploying a failed daemon.
- Monitoring the cluster.
- Monitoring the network.

This section provides insights about each topic, and highlights some of the best practices and necessary steps where applicable.

6.2.1 General considerations

Ceph is designed to automatically protect data so that a node or device failure is handled according to the data protection strategy that is specified for each pool: replication or erasure coding.

The failure of a node or the failure of multiple devices can endanger the resiliency of the data and expose it to a double failure scenario that can degrade the availability or durability of data.

Ceph clusters must be monitored and action taken swiftly to replace failed nodes and failed devices to limit the risk presented by multiple failure scenarios.

Data rebalancing or recovery within a Ceph cluster may impact client. As Ceph matured, the default recovery and backfill mechanisms and parameters were adjusted to minimize this negative impact.

6.2.2 Maintenance mode

cephadm can stop all daemons on a node. This feature is used for node maintenance, such as an operating system upgrade (Example 6-23).

Example 6-23 Node Maintenance mode

```
# ceph orch host maintenance enter {hostname}
# ceph orch host maintenance exit {hostname}
```

General Ceph commands

This section provides some basic Ceph commands that might prove helpful for Day 2 operations. Table 6-2 shows the general Ceph commands.

Table 6-2 General Ceph commands

Command	Description
ceph status|-s	Displays cluster status.
ceph health [detail]	Displays cluster health information.
ceph df	Displays cluster usage information.
ceph osd df	Displays OSD usage information.
ceph osd tree	Displays the CRUSH topology.
ceph osd pool ls [detail]	Displays Ceph pool information
ceph mon dump	Dumps the Monitor Map (MONMap).
ceph mgr dump	Dumps the MGRMap.
ceph osd dump	Dumps the OSDMap.
ceph fs dump	Dumps the MDSMap.
ceph osd pool set {pool} {parm} {value}	Changes a pool attribute.

Command	Description
`ceph osd pool get {pool} {parm}`	Displays a pool attribute.
`ceph mgr module ls`	Displays Manager module information.
`ceph mgr module enable {module}`	Enables an Manager module.
`ceph mgr module disable {module}`	Disables an Manager module.
`ceph version|-v`	Displays the Ceph release installed.
`ceph versions`	Displays the daemon versions across the cluster.
`ceph auth list`	Dumps CephX authentication and permission entries.
`ceph auth get-or-create`	Creates a CephX entry.
`ceph auth get {username}`	Displays a specific CephX entry.
`ceph auth del {username}`	Deletes a specific CephX entry.
`rados -p {poolname} ls`	Lists objects in a pool.
`rados list-inconsistent-pg {poolname}`	Lists inconsistent PGs within a pool.
`rados list-inconsistent-obj {pgid}`	Lists inconsistent objects within a PG.
`ceph pg dump_stuck {pgstate}`	Lists PGs that are stuck in stale, inactive, or unclean states.

6.2.3 Configuring logging

This section describes the configuration of logging.

Ceph cluster logging

Ceph daemons log to `journald` by default, and Ceph logs are captured by the container runtime environment. They are accessible by running **`journalctl`**.

For example, to view the daemon `mon.foo` logs for a cluster with ID `5c5a50ae-272a-455d-99e9-32c6a013e694`, run the command that is shown in Example 6-24.

Example 6-24 Checking the Ceph cluster logs

```
# journalctl -u ceph-5c5a50ae-272a-455d-99e9-32c6a013e694@mon.1
```

Ceph logging to file

You can also configure Ceph daemons to log to files instead of to journald if you prefer. When Ceph logs to files, the logs are written in `/var/log/ceph/<cluster-fsid>` (Example 6-25 on page 141).

Example 6-25 Configuring logging to files

```
# ceph config set global log_to_file true
# ceph config set global mon_cluster_log_to_file true
```

Tip: You can set a single daemon to log to a file by using `osd.0` instead of `global`.

By default, **`cephadm`** enables log rotation on each host. You can adjust the retention schedule and other specifics by modifying `/etc/logrotate.d/ceph.<CLUSTER FSID>`.

Because a few Ceph daemons (notably, Monitors and Prometheus) store a large amount of data in /var/lib/ceph, you may find it helpful to arrange for this directory to reside on a dedicated disk, partition, or logical volume so that it does not fill up the / or /var file system.

For more information, see Ceph daemon logs in IBM Documentation.

Disabling logging to journald

If you choose to log to files, we recommend that you disable logging to journald so that everything is not logged twice. To disable logging to `stderr`, run the commands in Example 6-26.

Example 6-26 Disabling logging to stderr

```
# ceph config set global log_to_stderr false
# ceph config set global mon_cluster_log_to_stderr false
# ceph config set global log_to_journald false
# ceph config set global mon_cluster_log_to_syslog false
```

Cephadm logs

`cephadm` writes logs to the **`cephadm`** cluster log channel. You can monitor the Ceph activity in real time by reading the logs as they fill up. To see the logs in real time, run the command in Example 6-27.

Example 6-27 The cephadm watch logs

```
# ceph -W cephadm
```

By default, this command shows info-level events and above. To see debug-level messages as well, run the commands in Example 6-28.

Example 6-28 Enabling the debug logs on cephadm

```
# ceph config set mgr mgr/cephadm/log_to_cluster_level debug
# ceph -W cephadm --watch-debug
```

You can see recent events by running the command in Example 6-29.

Example 6-29 Listing recent cephadm events

```
# ceph log last cephadm
```

If your Ceph cluster is configured to log events to files, these will be found in `ceph.cephadm.log` on all Monitor hosts (for more information, see the cephadm utility).

Cephadm per-service event logs

To simplify the debugging of failed daemon deployments, **cephadm** stores events on a per-service and per-daemon basis. To view the list of events for a specific service, run the command that is shown in Example 6-30.

Example 6-30 The cephadm per-service logs

```
# ceph orch ls --service_name=alertmanager --format yaml
  service_type: alertmanager
  service_name: alertmanager
  placement:
    hosts:
    - unknown_host
[...]
  events:
  - 2022-02-01T12:09:25.264584 service:alertmanager [ERROR] "Failed to apply: \
   Cannot place <AlertManagerSpec for service_name=alertmanager> on \
   unknown_host: Unknown hosts"'
```

For more information, see Monitor cephadm log messages in IBM Documentation.

Ceph subsystem logs

Each IBM Storage Ceph subsystem has a logging level setting for journald / file output logs and for logs in memory. You may set different values for each subsystem by setting a log and a memory level. Ceph logging levels operate on a scale of 1 - 20, where 1 is terse and 20 is verbose. The memory logs are generally not sent to the output log unless a unrecoverable signal is raised or an assertion in the source code is triggered on request.

A logging setting can take a single value for the log and memory levels, which sets them both to the same value. For example, if you specify debug ms = 5, Ceph treats it as a log and memory level of 5. You may also specify them separately. The first setting is the log level, and the second is the memory level. Separate them with a forward slash (/). For example, if you want to set the ms subsystem's debug logging level to 1 and its memory level to 5, you specify it as debug ms = 1/5. You can adjust log verbosity at runtime in different ways. Note that runtime adjustments will be reverted when each daemon is eventually restarted.

Example 6-31 shows how to configure the logging level for a specific subsystem.

Example 6-31 Configuring the logging level for a specific subsystem

```
# ceph tell osd.0 config set debug_osd 20
```

Example 6-32 how to use the admin socket from inside a daemon’s container.

Example 6-32 Configuring the logging level by using the admin socket

```
# ceph --admin-daemon /var/run/ceph/ceph-client.rgw.<name>.asok config set
debug_rgw 20
```

Example 6-33 shows how to make the verbose/debug change permanent so that it persists after a restart.

Example 6-33 Making the change log persistent

```
# ceph config set client.rgw debug_rgw 20
```

For more information, see Ceph subsystems in IBM Documentation.

Getting logs from the container startup

You might need to investigate why a **cephadm** command failed or why a certain service no longer runs properly.

To do so, use the **cephadm** logs to get logs from the containers running Ceph services (Example 6-34).

Example 6-34 Checking the startup logs from a Ceph service container

```
# cephadm ls | grep mgr
        "name": "mgr.ceph-mon01.ndicbs",
        "systemd_unit":
"ceph-3c6182ba-9b1d-11ed-87b3-2cc260754989@mgr.ceph-mon01.ndicbs",
        "service_name": "mgr",
# cephadm logs --name mgr.ceph-mon01.ndicbs
Inferring fsid 3c6182ba-9b1d-11ed-87b3-2cc260754989
-- Logs begin at Tue 2023-01-24 04:05:12 EST, end at Tue 2023-01-24 05:34:07 EST.
--
Jan 24 04:05:21 ceph-mon01 systemd[1]: Starting Ceph mgr.ceph-mon01.ndicbs for
3c6182ba-9b1d-11ed-87b3-2cc260754989...
Jan 24 04:05:25 ceph-mon01 podman[1637]:
Jan 24 04:05:26 ceph-mon01 bash[1637]:
36f6ae35866d0001688643b6332ba0c986645c7fba90d60062e6a4abcd6c8123
Jan 24 04:05:26 ceph-mon01 systemd[1]: Started Ceph mgr.ceph-mon01.ndicbs for
3c6182ba-9b1d-11ed-87b3-2cc260754989.
Jan 24 04:05:27 ceph-mon01
ceph-3c6182ba-9b1d-11ed-87b3-2cc260754989-mgr-ceph-mon01-ndicbs[1686]: debug
2023-01-24T09:05:27.272+0000 7fe90710d>
Jan 24 04:05:27 ceph-mon01
ceph-3c6182ba-9b1d-11ed-87b3-2cc260754989-mgr-ceph-mon01-ndicbs[1686]: debug
2023-01-24T09:05:27.272+0000 7fe90710d>
```

6.2.4 Cluster upgrade

As part of the lifecycle of an IBM Storage Ceph cluster, you must do periodic updates to gain new features and security fixes.

There are two types of upgrades: minor and major upgrades.

Major releases include more disruptive changes, such as Dashboard or Manager API deprecation, than minor releases. Major IBM Storage Ceph releases correlate with an upstream major release, for example from 5.X to 6.X or from 6.X to 7. Major upgrades may also require upgrading the operating system. To see the matrix of OS-supported versions depending on the IBM Storage Ceph release, see What are the Red Hat and IBM Storage Ceph releases and corresponding Ceph package Versions?

Minor upgrades generally correlate with the same upstream release and usually avoid disruptive changes. For IBM Storage Ceph, minor releases are IBM Storage Ceph 7.1, 7.2, and so forth.

For each minor release there are periodic maintenance releases. Maintenance releases bring security and bug fixes. New features are rarely introduced in maintenance releases. Maintenance releases are named 6.1z1, 6.1z2, and so on.

Ceph upgrade orchestration tools

For all versions of IBM Storage Ceph 5 and later, the only supported orchestration tool is **cephadm**.

If you are upgrading from Red Hat Ceph Storage to IBM Storage Ceph, and the Red Hat Ceph Storage cluster is at version 3.X or 4.X, the upgrade from 3.X to 4.X to 5.X is done with ceph-ansible. Before **cephadm**, the ceph-ansible tool found here was used for upgrading minor and major versions of Red Hat Ceph Storage.

Upgrading Ceph with cephadm: Prerequisites

With IBM Storage Ceph, you must use the **cephadm** orchestrator for upgrades. An example of an IBM Storage Ceph major upgrade is upgrading from Version 5.3 to 6.1. **cephadm** fully automates the process, making upgrades straightforward and reliable. The automated upgrade process follows Ceph upgrade best practices. For example:

- The upgrade begins with Ceph Managers, Ceph Monitors, and then other daemons.
- Each daemon is restarted only after Ceph validates that the cluster will remain available.

Before starting a Ceph upgrade, there are prerequisites:

1. Read the release notes of the version that you are upgrading to. The upgrade might have a known issue, for example, the depreciation of an API that you are using.
2. Open a proactive case with the IBM Support team.
3. Upgrade to the latest maintenance release of the latest minor version before doing a major upgrade. For example, before upgrading from 5.3 to 6.1, ensure that you are running the latest 5.3 maintenance release, which in this case is 5.3z5.
4. Ensure that at least two nodes have the admin label so that one can manage the cluster while the other is updated.
5. Check the podman package version. Podman and IBM Storage Ceph have independent end-of-life strategies that might make it challenging to find compatible versions. There is a matrix of supported podman versions for each release at Compatibility considerations between Ceph and podman versions.
6. Ensure that the currently running RHEL version is supported by the IBM Storage Ceph version to which you are upgrading. If the currently running version of RHEL is not supported by the new IBM Storage Ceph release, upgrade the OS before you upgrade IBM Storage Ceph. There are two ways to upgrade the OS:
   - RHEL Leapp upgrade. You can find the instructions at Upgrading from RHEL 8 to RHEL 9. The Leapp upgrade is an in-place upgrade of the OS. All OSDs and other Ceph data and services will be available after the upgrade of the OS is completed, so the recovery time will be shorter than a full reinstall that entails recreating OSDs and services.
   - RHEL full OS reinstall. With this approach, you do a scratch OS installation with the new version. The data on OSDs will be lost, so when you add the updated OS node to the Ceph cluster, it will need to recover all OSDs.

Both approaches are valid, and have pros and cons depending on the use case and infrastructure resources.

Upgrading Ceph with cephadm: Upgrade strategy

After you check and fulfill the prerequisites that are listed in “Upgrading Ceph with cephadm: Prerequisites” on page 144, you can start the Ceph upgrade with the help of **cephadm**.

There are two possible ways of updating Ceph with **cephadm**:

- All components are updated in a single pass; **cephadm** upgrades services individually. For more information, see IBM Storage Ceph.
- Staggered upgrade: This approach upgrades IBM Storage Ceph components in phases, rather than all at once. The Ceph Orch **upgrade** command offers options to limit which daemons are upgraded. This option offers fine-grained control of what is updated because you can limit per service, daemon, or host. **cephadm** strictly enforces an order for upgrading daemons, even in staggered upgrade scenarios.

  The current upgrade order is **Manager (mgrs)** → **Monitors (mons)** → **Crash daemons** → **OSDs** → **MDSs** → **RGWs** → **RBD mirrors** → **CephFS mirrors** → **NFS**. If you specify parameters that upgrade daemons out of this order, the **upgrade** command will block the action, and notify you which daemons will be missed if you proceed.

Upgrades can also be performed in disconnected (airgapped) environments with limited internet connectivity. For a step-by-step guide for performing disconnected upgrades, see Disconnected installation.

Data rebalancing during an upgrade

During an upgrade OSD services are restarted. The time that it takes to restart the OSD or OSD node determines whether the cluster declares the OSDs out of the cluster and starts a recovery process to ensure the data protection requirements of the pools/PGs on the affected OSDs. For example, if you have a replica count of 3 and lose an OSD containing one replica, the cluster declares the OSD out of the cluster after 10 minutes and starts a recovery process to create a new copy of the data on a different OSD, bringing the active replica count back to 3.

There are two strategies:

- The most common one is setting the parameters that are shown in Example 6-35 to prevent data movement, even if it takes longer than 10 minutes for an OS upgrade of an OSD node.

Example 6-35 Preventing OSDs from getting marked out during an upgrade and avoiding unnecessary load on the cluster

```
# ceph osd set noout
# ceph osd set noscrub
# ceph osd set nodeep-scrub
```

- The conservative approach is to recover all OSDs when upgrading the OS, especially if you are doing a clean OS installation to upgrade RHEL. This approach can take hours or days to complete in a large cluster because the upgrade blocks between hosts while the cluster is in a degraded state with only two valid copies of the data (assuming replica 3 is used).

To avoid this scenario, you can take a node out of the cluster by enabling maintenance mode in **cephadm**. When the Monitors' OSD timeout expires (the default is 10 minutes), recovery starts. When recovery completes, the cluster is in a fully protected state with three replicas. When the OS is updated, zap the OSD drives and add the node back to the cluster. This action triggers a data rebalance, which finishes when the cluster is in the HEALTH_OK status. Then, you can upgrade the next node and repeat the process.

For more information, see the Dashboard section in IBM Documentation.

Cluster upgrade monitoring

After running the **ceph orch upgrade start** command to upgrade the IBM Storage Ceph cluster, you can check the status, pause, resume, or stop the upgrade process. The cluster's health changes to HEALTH_WARNING during an upgrade.

Example 6-36 shows how to determine whether an upgrade is in process and the version to which the cluster is upgrading.

Example 6-36 Getting the upgrade status

```
# ceph orch upgrade status
```

To get detailed information about all the steps that the upgrade is taking, query the **cephadm** logs (Example 6-37).

Example 6-37 Getting the detailed upgrade logs

```
# ceph -W cephadm
```

You can pause, resume, or stop an upgrade while it is running (Example 6-38).

Example 6-38 Pausing, stopping, or resuming a running upgrade

```
# ceph orch upgrade [pause|resume|stop]
```

Cluster expansion

One of the outstanding features of Ceph is its ability to add or remove Ceph OSD nodes at run time. You can resize cluster capacity without taking it down, adding, removing, or replacing hardware during regular business hours. Note that adding or removing OSD nodes can impact performance.

Before adding Ceph OSD nodes, consider the effects on storage cluster performance. Adding or removing Ceph OSD nodes results in backfill or recovery as the storage cluster rebalances. During this rebalancing period, Ceph uses additional network and storage drive resources, which can impact client performance.

Because a Ceph OSD host is part of the CRUSH hierarchy adding or removing a node can affect the performance clients of pools that use OSDs placed on the affected node.

Adding a new IBM Storage Ceph node example

Before deploying any Ceph service into the cluster, fulfill the **cephadm** requirements. The following steps show an example of what you must do to add a new host to your cluster:

1. Install the RHEL operating system.
2. Register the node and subscribe to a valid entitlement pool (Example 6-39 on page 147).

Example 6-39 Registering the RHEL node to the Red Hat CDN

```
$ subscription-manager register
$ subscription-manager subscribe --pool=8a8XXXXXXXX9ff
```

3. Ensure that the appropriate RHEL repositories are enabled (Example 6-40).

Example 6-40 Enabling the RHEL repositories

```
$ subscription-manager repos --disable="*"
--enable="rhel-9-for-x86_64-appstream-rpms"
--enable="rhel-9-for-x86_64-baseos-rpms"
```

4. Add the IBM Storage Ceph repository for the release the cluster currently runs (Example 6-41).

Example 6-41 Adding the IBM Storage Ceph repository

```
$ curl
https://public.dhe.ibm.com/ibmdl/export/pub/storage/ceph/ibm-storage-ceph-6-rhel-9
.repo | sudo tee /etc/yum.repos.d/ibm-storage-ceph-6-rhel-9.repo
```

5. From the IBM Storage Ceph Cluster bootstrap node, enter the **`cephadm`** shell (Example 6-42).

Example 6-42 Entering the cephadm shell

```
$ cephadm shell
```

6. Extract the cluster public SSH key to a file (Example 6-43).

Example 6-43 Extracting the cluster public SSH key

```
$ ceph cephadm get-pub-key > ~/PATH
```

7. Copy the Ceph cluster public SSH keys to the root user's `authorized_keys` file on the new host (Example 6-44).

Example 6-44 Copying the SSH key to a new node

```
$ ssh-copy-id -f -i ~/PATH root@HOSTNAME
```

8. Add the new host to the Ansible inventory file. The default location for the file is `/usr/share/cephadm-ansible/hosts`. Example 6-45 shows the structure of a typical inventory file.

Example 6-45 Adding a new node to the Ansible inventory

```
$ cat /usr/share/cephadm-ansible/hosts
host01
host02
host03

[admin]
host00
```

9. Run the preflight playbook with the **--limit** option (Example 6-46).

Example 6-46 Copying the SSH key to the new node that you are adding

```
$ ansible-playbook -i INVENTORY_FILE cephadm-preflight.yml --extra-vars
"ceph_origin=ibm" --limit NEWHOST
```

10. From the Ceph administration node, log in to the **cephadm** shell (Example 6-47).

Example 6-47 Entering the cephadm shell

```
$ cephadm shell
```

11. Use the **cephadm** orchestrator to add hosts to the storage cluster (Example 6-48).

Example 6-48 Adding the new host to the cluster

```
$ ceph orch host add HOST_NAME IP_ADDRESS_OF_HOST
```

When the node is added to the cluster, you should see the node listed in the output of the **ceph orch host ls** command or **ceph orch** device list.

If any disks on the newly added host pass the filter that you configured in your **cephadm** OSD service spec, new OSDs are on them. The cluster rebalances, moving PGs from other OSDs to the new OSDs to distribute data evenly across the cluster.

Limiting backfill and recovery

Tuning the storage cluster for the fastest possible recovery time can impact Ceph client I/O performance. To maintain the highest Ceph client I/O performance, limit backfill and recovery operations by modifying the following Ceph configuration parameters (Example 6-49).

Example 6-49 Ceph parameters to favor client I/O during backfill

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

Increasing the number of placement groups

If you are expanding the size of your storage cluster, you might need to increase the number of PGs. As a best practice, make incremental increases because making large increases in the number of PGs can cause a noticeable degradation in performance.

6.2.5 Node replacement

A cluster likely will at some point experience a node failure. Handling a node failure is similar to handling a disk failure, but instead of recovering PGs using only one OSD, Ceph must recover all PGs on the OSDs served by that node. Ceph automatically detects that the OSDs are down and starts the recovery process to protect data by restoring the configured level of redundancy.

The best practices described in "Cluster expansion" on page 146 also apply to this section, so read them carefully before starting a node replacement.

Node replacement strategies

There are three host failure scenarios. Here is the high-level workflow for each scenario when replacing a host:

- Replacing the host by using the failed host's root and Ceph OSD disks:
  a. Disable backfilling.
  b. Replace the host , taking the disks from the old host and adding them to the new host .
  c. Enable backfilling.
- Replacing the host reinstalling the operating system, and using the Ceph OSD disks from the failed host:
  a. Disable backfilling.
  b. Create a backup of the Ceph configuration.
  c. Replace the node and add the Ceph OSD disks from the failed node.
  d. Configure disks as Just a Bunch of Disks (JBOD) aka passthrough or HBA mode.
  e. Install RHEL.
  f. Restore the Ceph configuration.
- Add the host to the storage cluster as an entirely new host. Ceph daemons are placed automatically on the respective host.
  a. Enable backfilling.
  b. Replacing the host, installing the operating system, and using all new Ceph OSD disks.
  c. Disable backfilling.
  d. Remove all OSDs on the failed node from the storage cluster.
  e. Replace the node and add the Ceph OSD disks from the failed node.
  f. Configuring disks as JBOD.
  g. Install the operating system.
  h. Add the new node to the storage cluster with **`ceph orch`** commands. Ceph daemons are placed automatically.
  i. Enable backfilling.

Node replacement steps

For more information about removing nodes, see Removing a Ceph OSD node. For more information about adding nodes, see Adding a Ceph OSD node.

6.2.6 Disk replacement

When a disk fails, you can replace the faulty storage device and reuse the same OSD ID so that you do not need to reconfigure the CRUSH map. You can remove the faulty OSD from the cluster while preserving the OSD ID with the **`ceph orch rm`** command. The OSD is not permanently removed from the CRUSH hierarchy, but it is flagged as destroyed. This flag is used to determine which OSD IDs can be used for future OSD deployments. By using the **`destroyed`** flag, you can easily identify which OSD ID can be assigned to the next OSD deployment. If you use OSD specification for deployment, your newly added OSD receives the OSD ID used by the failed disk.

For more information about disk replacement, see Replacing the OSDs.

All the best practices in this section on cluster expansion also apply to disk replacement, so read them carefully before starting a disk replacement. These best practices are used at a smaller scale because, in this case, it is a single OSD, not a full node.

6.2.7 Cluster monitoring

There are three complementary tools to monitor an IBM Storage Ceph cluster: the Dashboard RESTful API, the Ceph CLI commands, and the IBM Storage Ceph Observability Stack. Each tool provides different insights into cluster health.

The IBM Storage Ceph Observability Stack provides management and monitoring capabilities to administer and configure the cluster and visualize related information and performance statistics. The Dashboard uses a web server that is hosted by the `ceph-mgr` daemon.

Dashboard Observability Stack components

Multiple components provide the Dashboard's functions:

- The **`cephadm`** application for deployment.
- The Dashboard `ceph-mgr` module.
- The Prometheus `ceph-mgr` module.
- The Prometheus time-series database.
- The Prometheus node-exporter daemon runs on each storage cluster host.
- The Grafana platform to provide monitoring user interface and alerting.
- The Alert Manager daemon for alerting.

Figure 6-2 shows the Dashboard Observability Stack architecture.

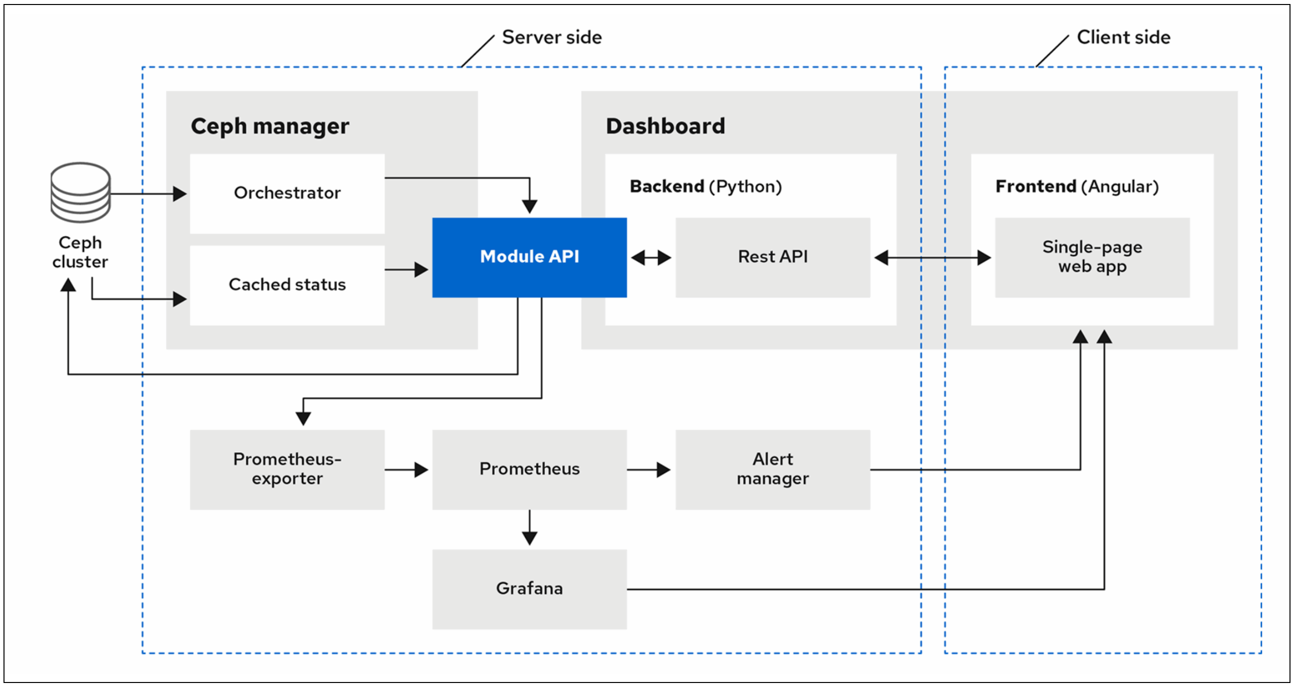


Figure 6-2 Dashboard Observability Stack architecture

Dashboard Observability Stack features

The Ceph Dashboard provides the following features:

- Multi-user and role management: The Dashboard supports multiple user accounts with individual permissions and roles. User accounts and roles can be managed by using the CLI or the web user interface. The Dashboard supports various strategies to enhance password security. Password complexity rules may be configured, requiring users to change their password after the first login or after a configurable period.

- Single Sign-On (SSO): The Dashboard supports authentication with an external identity provider (IDP) by using the SAML 2.0 protocol.
- Auditing: The Dashboard back end can be configured to log all `PUT`, `POST`, and `DELETE` API requests to the Ceph Manager log.

Management features

The Dashboard has the following management features:

- View the cluster hierarchy: You can view, for example, the CRUSH map to determine the host on which a specific OSD is located. This is helpful if there is an issue with an OSD.
- Configure Manager modules: You can view and change parameters for Ceph Manager modules.
- Embedded Grafana Dashboards: Grafana Dashboards may be embedded in external applications and web pages to surface information and performance metrics that are gathered by the Prometheus module.
- View and filter logs: You can view event and cluster logs and filter them based on priority, keyword, date, or time range.
- Toggle Dashboard components: You can enable and disable Dashboard components so only the features that you need are available.
- Embedded Grafana Dashboards: Grafana Dashboards may be embedded in external applications and web pages to surface information and performance metrics that are gathered by the Prometheus module.
- Viewing Alerts: The alerts page shows details of the current alerts.
- Quality of service for RBD volumes: You can set performance limits on volumes, limiting IOPS or throughput.

Monitoring features

The Dashboard has the following monitoring features:

- Username and password protection: You can access the Dashboard only by providing a configurable username and password.
- Overall cluster health: Displays overall cluster status as well as performance and capacity metrics. Storage utilization is also shown including number of RADOS objects, raw capacity, usage per pool, and a list of pools and their status and usage statistics.
- Hosts: Provides a list of all hosts in the cluster along with their running services and the installed Ceph version.
- Performance counters: Displays detailed statistics for each running service.
- Monitors: Lists Monitor daemons, their quorum status, and open sessions.
- Configuration editor: Displays available configuration options, their descriptions, types, defaults, and current values. These values are editable.
- Cluster logs: Displays and filters the cluster's event and audit log files by priority, date, or keyword.
- Device management: Lists hosts known by the Orchestrator. Lists all drives attached to each host and their properties. Displays drive health predictions, SMART data, and manages enclosure LEDs.
- View storage cluster capacity: You can view the raw storage capacity of the IBM Storage Ceph cluster in the Capacity pane of the Ceph Dashboard.
- Pools: Lists and manages all Ceph pools and their details. For example, application tags, PGs, replication size, EC profile, quotas, CRUSH ruleset, and others.

- OSDs: Lists and manages OSDs, their status and usage statistics, and detailed information including OSD map, metadata, and performance counters for read/write operations. Lists all drives associated with each OSD.
- Images: Lists RADOS Block Device (RBD) images and their properties including size, RADOS objects, and features. Creates, copies, modifies, and deletes RBD images. Creates, deletes, and rolls back snapshots of selected images, and protects or unprotects these snapshots against modification. Copies or clones snapshots, and flattens cloned images.
- RBD Mirroring: Enables and configures RBD mirroring to a remote Ceph cluster. Lists all active rbd-mirror daemons and their status, pools, and RBD images, including their synchronization state.
- CephFSs: Lists all active CephFS clients and associated pools, including their usage statistics. Evicts active CephFS clients, manages CephFS quotas and snapshots, and browses a CephFS directory structure.
- Object Gateway (RGW): Lists all active Object Gateways and their performance counters. Displays and manages Object Gateway users and their details including quotas and the users' buckets.

Security features: SSL and TLS support

All HTTP communication between the web browser and the Dashboard is secured by SSL. A self-signed certificate can be created with a built-in command, but it is also possible to import custom certificates that are signed and issued by a certificate authority (CA).

Dashboard access

You can access the Dashboard with the credentials that are provided on bootstrapping the cluster.

Cephadm installs the Dashboard by default. Example 6-50 is an example of the Dashboard URL.

Example 6-50 Dashboard credentials example during bootstrap

```
URL: https://ceph-mon01:8443/
User: admin
Password: XXXXXXXXX
```

To find the Ceph Dashboard credentials, search the `var/log/ceph/cephadm.log` file for the string `"Ceph Dashboard is now available at"`.

Change the password the first time that you log in to the Dashboard with the credentials that are provided by bootstrapping only if the **`--dashboard-password-noupdate`** option is not used while bootstrapping.

6.2.8 Network monitoring

Network configuration and health are critical for IBM Storage Ceph cluster operation and performance. As a distributed storage solution, Ceph is sensitive to network fluctuations, flapping, and other health issues. Because OSDs replicate client writes to other OSDs in the cluster over the network, network latency can hinder performance. Therefore, it is critical to have a network monitoring stack that alerts when an issue is identified.

IBM Storage Ceph issues built-in network warnings in CLI output and in the observability stack via Alertmanager.

Ceph OSDs send heartbeat liveness probes to each other to test daemon availability and network performance. If a single delayed response is detected, it might indicate nothing more than a busy OSD. But if recurring delays between OSDs are detected, it might indicate a failed network switch, a NIC failure, or a layer 1 issue.

In the output of the **ceph health detail** command, you can see which OSDs are experiencing delays and how long the delays are. The output of ceph health detail is limited to ten lines. Example 6-51shows an example of the output the **ceph health detail** command.

Example 6-51 Output example of the ceph health detail command

```
[WRN] OSD_SLOW_PING_TIME_BACK: Slow OSD heartbeats on back (longest 1118.001ms)
Slow OSD heartbeats on back from osd.0 [rack2,host1] to osd.1 [rack2,host2]
1118.001 msec possibly improving
Slow OSD heartbeats on back from osd.0 [rack2,host1] to osd.2 [rack2,host3]
1030.123 msec
```

To see more details and collect a complete dump of network performance information, use the **dump_osd_network** command.

The Alertmanager that is part of the Observability stack offers a comprehensive set of preconfigured alerts related to networking. Example 6-52 shown an example.

Example 6-52 Alertmanager pre-configured network alert example

```
- alert: "CephNodeNetworkPacketDrops"
        annotations:
          description: "Node {{ $labels.instance }} experiences packet drop > 0.5%
or > 10 packets/s on interface {{ $labels.device }}."
          summary: "One or more NICs reports packet drops"
        expr: |
          (
            rate(node_network_receive_drop_total{device!="lo"}[1m]) +
            rate(node_network_transmit_drop_total{device!="lo"}[1m])
          ) / (
            rate(node_network_receive_packets_total{device!="lo"}[1m]) +
            rate(node_network_transmit_packets_total{device!="lo"}[1m])
          ) >= 0.0050000000000000001 and (
            rate(node_network_receive_drop_total{device!="lo"}[1m]) +
            rate(node_network_transmit_drop_total{device!="lo"}[1m])
          ) >= 10
        labels:
          oid: "1.3.6.1.4.1.50495.1.2.1.8.2"
          severity: "warning"
          type: "ceph_default"
      - alert: "CephNodeNetworkPacketErrors"
        annotations:
          description: "Node {{ $labels.instance }} experiences packet errors >
0.01% or > 10 packets/s on interface {{ $labels.device }}."
          summary: "One or more NICs reports packet errors"
        expr: |
          (
            rate(node_network_receive_errs_total{device!="lo"}[1m]) +
            rate(node_network_transmit_errs_total{device!="lo"}[1m])
```

```
  ) / (
    rate(node_network_receive_packets_total{device!="lo"}[1m]) +
    rate(node_network_transmit_packets_total{device!="lo"}[1m])
  ) >= 0.0001 or (
    rate(node_network_receive_errs_total{device!="lo"}[1m]) +
    rate(node_network_transmit_errs_total{device!="lo"}[1m])
  ) >= 10
labels:
  oid: "1.3.6.1.4.1.50495.1.2.1.8.3"
  severity: "warning"
  type: "ceph_default"
```

For more information about the preconfigured alarms, see this GitHub repository.

Abbreviations and acronyms

ABAC	attribute-based access control
AWS	Amazon Web Services
CA	certificate authority
CephFS	Ceph File System
COW	copy-on-write
CRUSH	Controlled Replication Under Scalable Hashing
CSV	comma-separated values
D3N	Datacenter-Data-Delivery Network
DAS	direct-attach storage
DR	disaster recovery
EC	erasure coding
FINRA	Financial Industry Regulatory Authority
FSAL	File System Abstraction Layer
FUSE	File System in UserSpace
GA	general availability
HA	high availability or highly available
HDD	hard disk drive
HIPAA	Health Insurance Portability and Accountability Act
IBM	International Business Machines Corporation
IDP	identity provider
IOPS	input/output operations per second
ISV	independent software vendor
ITIL	IT Infrastructure Library
JBOD	Just a Bunch of Drives
KRBD	Kernel RBD
LACP	Link Aggregation Control Protocol
LRU	Least Recently Used
LUKS	Linux Unified Key Setup
MDS	MetaData Server
MFA	Multi-Factor Authentication
MGR	Manager
MON	Monitor
MONMap	Monitor Map
MTU	maximum transmission unit
NVMe-oF	NVMe over Fabrics
OIDC	OpenID Connect
OPS	operations per second
OSD	object storage daemon
PACS	picture archiving and communication system
PG	placement group
PMP	Project Management Professional
PVC	persistent volume claim
QEMU	Quick Emulator
RADOS	Reliable Autonomic Object Store
RBAC	role-based access control
RBD-NBD	RBD Network Block Device
RBD	RADOS Block Device
RGW	RADOS Gateway
RWO	Read Write Once
SaaS	Storage as a Service
SAN	storage area network
SEC	Securities and Exchange Commission
SSDs	solid-state drives
SSO	Single Sign-On
STS	Secure Token Service
STS	Security Token Service
TCO	total cost of ownership
VFS	virtual file system
VM	virtual machine
WAL	write-ahead log
WORM	Write Once, Read Many

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- *IBM Storage Ceph Solutions Guide*, REDP-5715
- *Unlocking Data Insights and AI: IBM Storage Ceph as a Data Lakehouse Platform for IBM watsonx.data and Beyond*, SG24-8563 (in draft at the time of editing)

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Amazon Web Services (AWS) CLI documentation:

  https://docs.aws.amazon.com/cli/index.html
- Community Ceph Documentation:

  https://docs.ceph.com/en/latest/monitoring/
- IBM Storage Ceph Documentation:

  https://www.ibm.com/docs/en/storage-ceph/6?topic=dashboard-monitoring-cluster
- IP load balancer documentation:

  https://github.ibm.com/dparkes/ceph-top-gun-enablement/blob/main/training/modules/ROOT/pages/radosgw_ha.adoc

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5721-00

ISBN 0738461520

Printed in U.S.A.