What is Ceph?

Ceph is an open-source, distributed storage platform that provides object storage, block storage, and file storage in a single unified system. It is designed to provide scalable and fault-tolerant storage for large-scale data environments such as cloud infrastructure and data centers. Ceph was developed by Sage Weil and is now part of the Linux Foundation.

Key Components of Ceph:

Monitor (MON):

- A **Ceph Monitor (MON)** is a critical component in a Ceph cluster that maintains the overall cluster state, including cluster maps, authentication, and quorum management. It is responsible for keeping track of the state of the entire cluster and ensuring data consistency and fault tolerance.

- **Cluster State Management:**

- MONs maintain the **current state of the cluster**, including information about OSDs, metadata servers (MDS), manager daemons (MGR), and placement groups (PGs).
- They store and manage the following cluster maps:
  - **OSD Map:** Tracks the state of all OSDs (up, down, in, out).
  - **MON Map:** Tracks the state and location of all monitors.
  - **PG Map:** Tracks the state of all placement groups.
  - **CRUSH Map:** Defines data distribution rules based on topology and capacity.
  - **MDS Map:** Manages the state of the metadata servers.

- **Consensus and Quorum Management:**

  - Ceph MONs use the **Paxos algorithm** to achieve consensus on cluster state changes.
  - Quorum is the minimum number of MONs that must agree to a state change to maintain data consistency.
  - Typically, Ceph deployments use an **odd number of MONs** (3, 5, 7) to avoid split-brain scenarios.

Example:

- In a 3-MON setup, at least 2 MONs must agree on a state change to reach quorum.

- **Authentication and Security:**

  - MONs handle **Cephx authentication**, a built-in authentication mechanism that provides secure access to cluster resources.
  - Each client and daemon must authenticate with the MONs to gain access using a Cephx keyring.

Example:

- When an OSD attempts to join the cluster, it must present its keyring to the MONs to be authenticated.

- Data Consistency and Recovery:

- MONs track the state of PGs and the placement of replicas to ensure data consistency.
- If an OSD goes down, the MONs initiate the recovery and rebalancing process based on the CRUSH map.

- How MONs Work Together:

- MONs work together as a cluster, sharing the same data and state information.
- They use **Paxos to agree on state changes**, ensuring that the entire cluster maintains a consistent view of the cluster state.
- Each MON maintains a copy of the cluster maps and updates them when state changes occur.

2. **Manager (MGR):**
    - Managers provide real-time monitoring and management information for the cluster.
    - They handle the dashboard, Prometheus metrics, and other plugins.
    - **Monitoring and Metrics Aggregation:**

- The MGR collects performance metrics from OSDs, MONs, MDS, and RGWs.
- It provides data on CPU usage, memory usage, IOPS, latency, throughput, and network traffic.
- Metrics are aggregated and made available through the Ceph dashboard, REST API, and third-party tools like **Prometheus and Grafana**.
    - Dashboard Service
    - **Failover and High Availability:**
- The MGR daemons are **not quorum-based** like MONs but can run multiple instances for redundancy.
- One MGR is always **active**, and the rest are **standby**.
- If the active MGR fails, a standby MGR is promoted to active.

    - **Failover and High Availability:**
- The MGR daemons are **not quorum-based** like MONs but can run multiple instances for redundancy.
- One MGR is always **active**, and the rest are **standby**.

- If the active MGR fails, a standby MGR is promoted to active.
    -

3. **Object Storage Daemon (OSD):**
    - OSDs are the primary data storage units in Ceph. Each OSD stores data, handles replication, recovery, and rebalancing.
    - Typically, each physical disk or SSD in a Ceph cluster is associated with a single OSD.
4. **Metadata Server (MDS):**
    - MDS handles metadata operations for CephFS, the POSIX-compliant file system.
    - It manages directories, file attributes, and access control lists.
5. **RADOS:(Reliable Autonomous Distributed Object Store)**
    - RGW provides an object storage interface that is compatible with Amazon S3 and OpenStack Swift.
    - It enables RESTful access to objects stored in the Ceph cluster.
    - RADOS is the underlying storage mechanism in Ceph. It provides a scalable object store that can handle massive amounts of data in a distributed fashion. It is the core of Ceph, offering storage at the object level, and it is designed to be highly reliable and fault-tolerant. Here's how RADOS functions:
    - **Object Storage**: RADOS stores data in objects, which are distributed across multiple storage nodes in a Ceph cluster. This is similar to how cloud storage works, where data is stored as objects instead of traditional files.

Data Distribution: RADOS uses a mechanism called the CRUSH algorithm (Controlled Replication Under Scalable Hashing) to intelligently distribute data across multiple nodes. CRUSH ensures that data is distributed evenly across all nodes in the cluster, with redundancy and fault tolerance.

- **Fault Tolerance**: If a node or disk in the Ceph cluster fails, RADOS will automatically redistribute the data to other nodes without any loss of data. It maintains multiple copies of each object (based on the replication settings), ensuring that the system can recover from failures without downtime.

Ceph OSDs (Object Storage Daemons): RADOS relies on OSDs to store and manage data on physical storage devices. These OSDs are responsible for storing objects, handling read/write requests, and replicating data to ensure redundancy and fault tolerance.

CRUSH Map: The CRUSH map is a configuration file that Ceph uses to map objects to specific locations in the cluster. The map defines how data is placed across the nodes and how replication works. The CRUSH algorithm uses the map to determine where each object is stored.

RGW (RADOS Gateway)

RGW is a gateway that allows you to interact with Ceph's RADOS object store using protocols such as Amazon S3 or OpenStack Swift. RGW provides a RESTful interface, enabling applications to access the object storage as they would with any cloud storage service (e.g., Amazon S3)

- **Object Storage Interface: RGW exposes a web-based API that supports object storage interfaces like S3 and Swift. This allows users to interact with Ceph storage using standard cloud protocols, making it compatible with many cloud-native applications.**
- **Ceph RADOS Gateway Daemons**: RGW runs as a daemon on Ceph nodes and provides services for managing objects. These services include uploading and downloading objects, as well as bucket management. Each RGW daemon communicates with the Ceph cluster to store and retrieve objects from the RADOS layer.
- **Multi-tenancy**: RGW supports multi-tenancy, meaning that it can be used by multiple users or applications, each with isolated storage and access control. Users can define their own buckets and objects within the system, while RGW ensures that access is controlled.
- **Load Balancing and Scaling**: RGW supports multiple instances running on different Ceph nodes. By scaling out RGW daemons, you can distribute the load of object storage traffic, providing high availability and scalability. The system can handle large amounts of object requests without overloading a single node.
- **Features of RGW**:
- **Bucket and Object Management**: It allows operations like creating buckets, uploading, and downloading objects.
- **Access Control**: RGW integrates with various authentication and authorization mechanisms, such as IAM (Identity and Access Management) for fine-grained control over access.
- **Data Durability**: Objects stored in Ceph via RGW are subject to the same fault tolerance and redundancy mechanisms as other Ceph data (replication, erasure coding).
- **Integration with Cloud Tools**: RGW enables integration with cloud-native tools that require object storage. For example, RGW’s compatibility with Amazon S3 allows users to configure their applications to use Ceph as the object storage backend while maintaining compatibility with S3 APIs.

Data Storage Types in Ceph:

1. **Object Storage (RADOS):**
    - Ceph's base layer is the Reliable Autonomic Distributed Object Store (RADOS).
    - It distributes objects across the cluster and ensures data replication and consistency using CRUSH (Controlled Replication Under Scalable Hashing).
    - It supports data protection through replication and erasure coding.
    - In object storage, data is stored in the form of discrete units called objects

-- **Object**:

- An object is the basic unit of storage in an object storage system. It typically consists of three main components:
  - **Data**: The actual content or file being stored, such as an image, video, document, etc.
  - **Metadata**: Data that describes the object. This can include information like file size, creation date, file type, and custom-defined tags or attributes (e.g., security level or geographical location).
  - **Unique Identifier (Object ID)**: Every object in the storage system is assigned a unique ID, which allows it to be retrieved and accessed. This identifier can be based on the object's name or generated automatically by the system.
- **Buckets**:
- Objects are often grouped together into "buckets" or "containers" in object storage systems. A bucket is a logical container that holds objects. Buckets help organize and manage objects, and they can provide access control and other features.
- In cloud storage systems like Amazon S3, buckets are used to define namespaces for objects, and each object within a bucket must have a unique name.
- **Scalability**:
- Object storage is highly scalable. It is designed to grow seamlessly as data needs increase. As data is added, the system can distribute it across many storage devices or servers without disruption.
- This scalability is due to the distributed nature of object storage systems, which use algorithms like **CRUSH (Controlled Replication Under Scalable Hashing)** in Ceph or **Dynamo** in Amazon S3 to distribute data across many nodes in the cluster.
- **Flat Namespace**:
- Unlike traditional file storage, which uses a hierarchical file system (folders within folders), object storage uses a flat namespace where each object is stored with a unique identifier (often a key or name).
- This flat structure allows for greater scalability and easier management of large amounts of data, as the system doesn’t need to maintain complex directory structures.
- **Data Redundancy and Fault Tolerance**:
- Object storage systems often provide redundancy and data protection features, ensuring data durability and availability even in the event of hardware failures.
- Redundancy can be achieved through **replication** (storing multiple copies of an object across different locations) or **erasure coding** (splitting the object into fragments and storing them across different nodes with the ability to reconstruct it even if some fragments are lost).
- **Accessing Objects**:
- Objects in object storage systems are typically accessed via HTTP-based APIs (RESTful APIs), using protocols like **S3** or **OpenStack Swift**.

- Each object is accessed using its unique identifier (such as a URL or object key), allowing applications to retrieve or modify it directly.
- This is a major advantage over traditional file systems where the data is accessed through filesystem-specific protocols (e.g., NFS or SMB).
- **Metadata and Search**:
- The ability to store metadata along with objects allows for rich search capabilities. For example, an image stored in object storage might have metadata that includes the image's resolution, date taken, and location where it was taken. This metadata can be indexed, making it easier to search and organize large amounts of data.
- **Durability and Availability**:
- One of the defining features of object storage is its **durability**. Object storage systems often replicate data across multiple nodes or locations to ensure that data remains available even if a failure occurs.
- **Geographical distribution**: Object storage is often deployed in multiple geographic locations (data centers), ensuring that data is available even during regional outages.
- **Security**:
- Object storage systems often provide advanced security features such as:
    - **Encryption**: Both at-rest and in-transit encryption can be used to ensure that data is protected.
    - **Access Control**: Fine-grained access controls, such as role-based access control (RBAC) or policies, ensure that only authorized users or applications can access specific objects or buckets.
    - **Audit Logging**: Systems may also offer logging features to track access to data for security and compliance purposes.

2. **Block Storage (RBD - RADOS Block Device):**
    - Provides block storage volumes that can be attached to virtual machines or bare metal servers.
    - Supports snapshots, cloning, and thin provisioning.
    - Integrates with virtualization platforms like OpenStack, Kubernetes, and VMware.
3. **File System (CephFS):**
    - CephFS is a POSIX-compliant file system that uses RADOS as its backend.
    - It supports high throughput, large data sets, and concurrent file access.
    - It relies on the MDS for metadata operations.

Data Management in Ceph:

- **CRUSH Algorithm: (Controlled Replication Under Scalable Hashing)**
    - The CRUSH algorithm determines how data is distributed across OSDs.
    - It uses a pseudo-random algorithm to distribute data without a central index, eliminating bottlenecks.
- **Replication and Erasure Coding:**
    - Replication creates multiple copies of data to provide redundancy.
    - Erasure Coding is a more storage-efficient data protection method, splitting data into fragments and storing them across different OSDs.

Features and Benefits of Ceph:

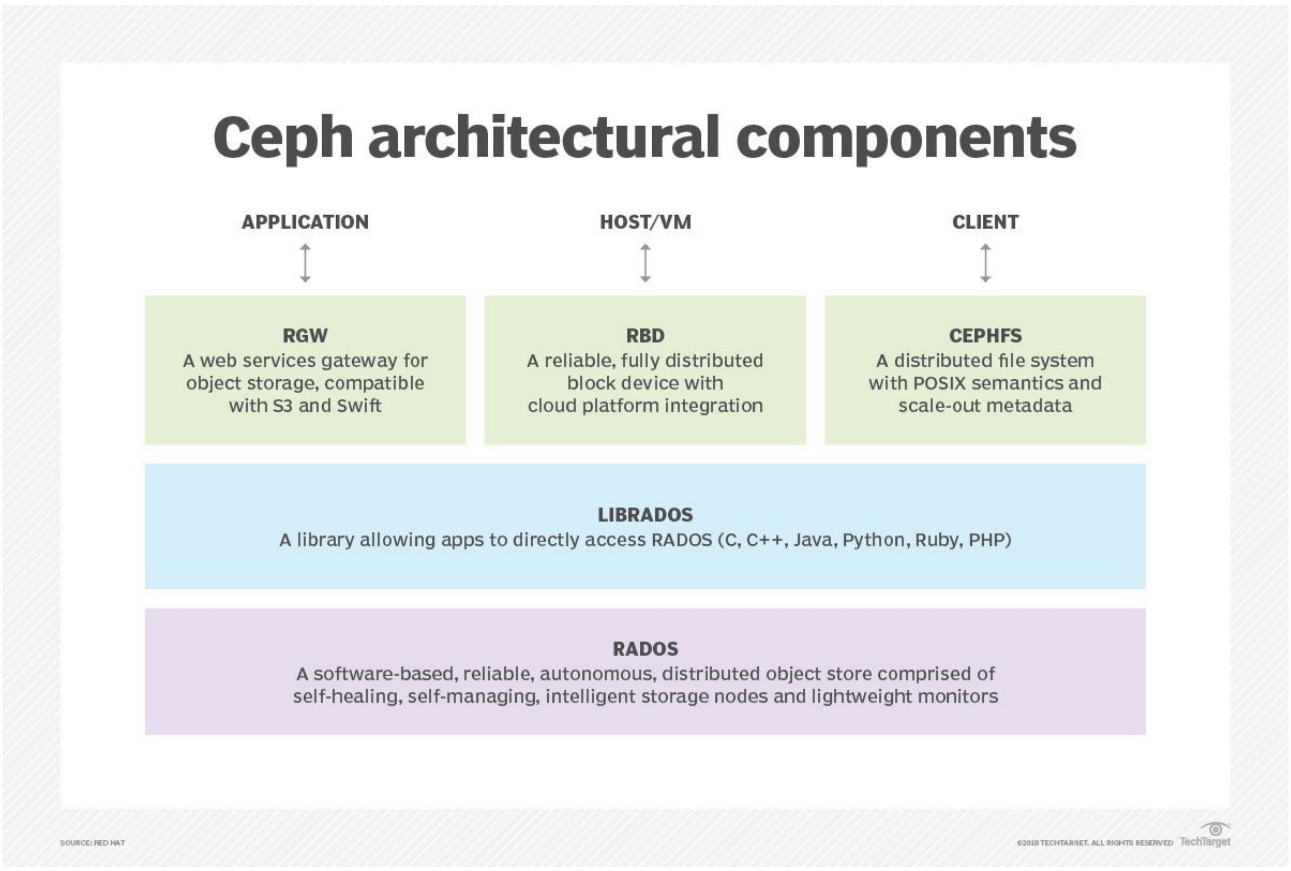
1. **Scalability:**
    - Ceph can scale to petabytes of data across thousands of nodes without a central bottleneck.
2. **High Availability and Fault Tolerance:**
    - Data is distributed and replicated, ensuring data availability even in the event of hardware failures.
3. **Self-Healing:**
    - Ceph automatically detects and repairs inconsistencies by comparing data replicas.
4. **Unified Storage Platform:**
    - Ceph provides object, block, and file storage within a single system, reducing management complexity.
5. **Open Source and Vendor-Neutral:**
    - Ceph is open-source and not tied to any specific hardware vendor, providing flexibility in deployment.

Ceph Use Cases:

- **Cloud Infrastructure (OpenStack, Kubernetes):**
    - Ceph is a preferred backend storage solution for OpenStack Cinder (block storage), Glance (image storage), and Swift (object storage).
- **Big Data and Analytics:**
    - Ceph can store massive datasets and provide high throughput for data-intensive applications.
- **Backup and Archiving:**

- With its erasure coding and replication, Ceph is well-suited for backup and disaster recovery scenarios.

- **Web Applications and Content Delivery Networks (CDNs):**
    - Ceph's RGW provides S3-compatible object storage, making it ideal for web-scale object storage.


Ceph architectural components
APPLICATION
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 platform integration
CEPHFS
A distributed file system with POSIX semantics and scale-out metadata
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)
RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors
SOURCE: RED HAT
©2018 TECHTARGET. ALL RIGHTS RESERVED
TechTarget

Ceph architecture and work

Ceph is built on top of Reliable Autonomic Distributed Object Store (RADOS) storage nodes described as intelligent, self-healing and self-managing. These nodes communicate with each other to dynamically replicate and redistribute data. RADOS uses intelligent nodes to secure stored data and provide data to clients, which are external programs that use a Ceph cluster to store and replicate data. A Ceph storage cluster based on RADOS might contain thousands of storage nodes.

This intelligent distributed solution provides a flexible, scalable and reliable way to store and access data. Enterprise users can distribute data throughout a cluster as needed using Ceph regardless of the data storage type since Ceph manipulates storage as objects within logical storage pools in RADOS.

How Data is Replicated and Distributed in a Ceph Cluster – Detailed Explanation

Ceph uses a combination of CRUSH (Controlled Replication Under Scalable Hashing) algorithm, Placement Groups (PGs), and Replication/Erasure Coding to distribute and replicate data across the cluster. This process ensures data redundancy, fault tolerance, and optimal data placement without a central directory or index.

1. Understanding CRUSH Algorithm

CRUSH is the core algorithm responsible for determining how and where data is stored in a Ceph cluster. It maps data to OSDs based on a set of rules defined in the `crushmap`.

Steps of the CRUSH Algorithm:

1. **Data Input (Object):**
    - The client (e.g., CephFS, RBD, RGW) requests to store an object in the cluster.
2. **Object Hashing:**
    - Ceph calculates a hash of the object to identify its **Placement Group (PG)**.
    - The hash is derived using the object ID and the number of PGs.
3. **Mapping to PG:**
    - Ceph uses the hash to determine which PG the object belongs to.
    - Example: Object `obj123` hashes to PG `12.34`.
4. **CRUSH Mapping:**
    - The CRUSH algorithm maps the PG to a set of OSDs based on the CRUSH ruleset.
    - Rules can define how many replicas to store, in which failure domains (e.g., host, rack, data center), and with what redundancy scheme (replication or erasure coding).
5. **OSD Selection:**
    - CRUSH identifies the OSDs responsible for storing the object based on PG mapping.
    - Example: PG `12.34` is mapped to OSDs `osd.1`, `osd.2`, and `osd.3`.

2. Placement Groups (PGs)

What are PGs?

- Placement Groups are logical containers that group objects together for replication and data placement.
- Instead of mapping objects directly to OSDs, Ceph maps objects to PGs, and then PGs to OSDs.

- **Purpose of PGs:**

    - They act as an abstraction layer that reduces the complexity of managing data placement.
    - They allow data to be evenly distributed across the cluster, ensuring balanced storage utilization.

PG Calculation Formula:

- The number of PGs is calculated using the formula:

  Total PGs=(Total OSDs × 100) / Max Replication Factor

  Example:

    - OSDs = 10
    - Replication Factor = 3
    - PGs = (10×100)/3=333

3. Data Replication Process in Ceph

Ceph implements replication to ensure data durability and high availability. The replication process involves the following steps:

Replication Steps:

1. **Primary OSD Selection:**
    - When an object is written to a PG, the CRUSH algorithm determines the **primary OSD**.
    - The primary OSD is responsible for initiating the replication process.
2. **Data Write to Primary OSD:**
    - The client writes the data to the primary OSD.
    - The primary OSD stores the data and then replicates it to secondary OSDs as per the replication factor.
3. **Replication to Secondary OSDs:**
    - The primary OSD sends copies of the data to the designated secondary OSDs.
    - If the replication factor is 3, there will be **1 primary OSD and 2 secondary OSDs**.
4. **Data Acknowledgment:**
    - Each secondary OSD acknowledges the receipt of data to the primary OSD.
    - Once all acknowledgments are received, the primary OSD confirms the write operation to the client.
5. **Handling OSD Failures:**
    - If an OSD fails, CRUSH recalculates the placement for the affected PGs and initiates recovery, replicating data to new OSDs.

4. Erasure Coding in Ceph

Erasure Coding (EC) is a data protection mechanism that provides data durability while consuming less storage compared to replication.

- **How EC Works:**
    - Data is divided into **k data chunks** and **m parity chunks**.
    - A total of `k + m` chunks are distributed across different OSDs.
    - Example: EC(2, 1) will split data into 2 data chunks and 1 parity chunk, resulting in 3 chunks.

Erasure Coding Steps:

1. **Data Encoding:**
    - The object is divided into data chunks and parity chunks using a specific algorithm (e.g., Reed-Solomon).
2. **Chunk Distribution:**
    - Chunks are distributed across OSDs based on the CRUSH algorithm.
3. **Data Recovery:**
    - If one or more chunks are lost, the missing data can be reconstructed using the remaining data and parity chunks.

5. Data Rebalancing in Ceph

When new OSDs are added or existing OSDs fail, Ceph rebalances data across the cluster.

- **Rebalancing Process:**
    1. The CRUSH map is updated to reflect the new OSD layout.
    2. The affected PGs are remapped to new OSDs based on the updated CRUSH ruleset.
    3. Data is moved to the new OSDs to maintain data distribution and redundancy.

6. Data Recovery Process in Ceph

Data recovery is initiated when an OSD fails or a PG becomes incomplete.

- **Recovery Steps:**
    1. The monitor detects the OSD failure and marks it as `down`.
    2. The primary OSD reassigns the affected PGs to healthy OSDs.
    3. Data replication or erasure coding is triggered to recreate the missing data.
    4. The new OSDs receive the data and acknowledge the completion of the recovery.

7. Monitoring Data Distribution in Ceph

Ceph CLI Commands:

- `ceph osd tree` – Displays the hierarchy of OSDs and their states.
- `ceph osd df` – Shows the data distribution across OSDs.
- `ceph pg dump` – Displays detailed information about each PG.
- `ceph pg ls-by-pool <pool_name>` – Lists PGs associated with a specific pool.

8. Practical Example of Data Distribution and Replication:

Scenario:

- Ceph Cluster with 5 OSDs (`osd.0` to `osd.4`), replication factor of 3.

Data Write Process:

1. A write request for object `obj123` is received.
2. Ceph hashes `obj123` to PG `12.34`.
3. CRUSH maps PG `12.34` to OSDs `osd.1`, `osd.2`, and `osd.4`.
4. The data is first written to the primary OSD (`osd.1`).
5. OSD `osd.1` replicates data to `osd.2` and `osd.4`.
6. Once all OSDs acknowledge the write, the operation is confirmed.

How Ceph Handles Data Redundancy and Failure – Detailed Explanation

Ceph employs robust mechanisms to ensure data redundancy and fault tolerance through replication, erasure coding, failure detection, recovery, and rebalancing. These mechanisms work together to maintain data availability and integrity even in the event of node, disk, or network failures.

1. Data Redundancy in Ceph

Data redundancy in Ceph is achieved through two primary methods:

- **Replication**
- **Erasure Coding (EC)**

1.1 Replication:

Replication involves creating multiple copies of data and distributing them across different OSDs to ensure data availability.

Replication Steps:

1. **Data Placement:**
    - When an object is written to Ceph, the CRUSH algorithm determines the **Primary OSD** responsible for storing the data.
    - The object is then replicated to `n` OSDs based on the specified replication factor.
    - Example: If the replication factor is `3`, there will be **1 primary OSD and 2 replica OSDs**.
2. **Data Write Process:**
    - The client writes data to the primary OSD.
    - The primary OSD stores the data and simultaneously replicates it to the secondary and tertiary OSDs.
3. **Data Acknowledgment:**
    - Each OSD acknowledges the receipt of data to the primary OSD.
    - Once all replicas have acknowledged, the primary OSD confirms the write operation to the client.

Example of Replication:

- Cluster with 4 OSDs (`osd.0`, `osd.1`, `osd.2`, `osd.3`).
- Replication factor: 3
- Object `file.txt` is written to PG `12.34`.
- PG `12.34` is mapped to `osd.1`, `osd.2`, and `osd.3`.
- Data is stored as:
    - Primary: `osd.1`
    - Replica 1: `osd.2`
    - Replica 2: `osd.3`

1.2 Erasure Coding (EC):

Erasure Coding is a more storage-efficient method that breaks data into smaller fragments and distributes them across multiple OSDs.

- It uses a configuration of `k + m` chunks:
    - `k` = Data chunks
    - `m` = Parity chunks

Steps of Erasure Coding:

1. **Data Encoding:**
    - The object is divided into `k` data chunks and `m` parity chunks using algorithms like Reed-Solomon.
2. **Distribution Across OSDs:**
    - Chunks are distributed across multiple OSDs based on the CRUSH map.
    - Example: EC(2, 1) will create **2 data chunks and 1 parity chunk**, resulting in 3 chunks spread across 3 OSDs.
3. **Data Recovery:**
    - If one or more OSDs fail, the missing data is reconstructed using the remaining data and parity chunks.

Example of Erasure Coding:

- Cluster with 4 OSDs (`osd.0`, `osd.1`, `osd.2`, `osd.3`).
- EC configuration: 2 data chunks + 1 parity chunk.
- Object `video.mp4` is divided as follows:
    - Data Chunk 1 → `osd.0`
    - Data Chunk 2 → `osd.1`
    - Parity Chunk → `osd.2`

---

2. Failure Detection in Ceph

Ceph continuously monitors the health and status of OSDs, monitors, and other cluster components.

Failure Detection Mechanisms:

1. **Heartbeat Messages:**
    - OSDs regularly send heartbeat messages to neighboring OSDs.
    - If a heartbeat is not received within a specified timeout period, the OSD is marked as **down**.
2. **Monitor (MON) Quorum:**
    - Monitors maintain a consistent view of the cluster state.
    - If an OSD is marked down, the monitors update the CRUSH map to exclude the failed OSD.
3. **Failure Domains:**
    - Ceph can be configured to handle failures at different levels, such as:

        - OSD
        - Host
        - Rack
        - Data Center

Example of Failure Detection:

- `osd.2` stops responding.
- MONs detect the absence of heartbeats from `osd.2`.
- `osd.2` is marked **down and out**.
- Data previously stored on `osd.2` is now marked for recovery.

---

3. Data Recovery Process in Ceph

When an OSD fails or goes offline, Ceph initiates the recovery process to restore data redundancy.

Data Recovery Steps:

1. **Identify Affected PGs:**
    - The MONs identify which PGs were hosted by the failed OSD.
2. **Remapping PGs:**
    - The CRUSH map is adjusted to remap the affected PGs to new OSDs.
3. **Data Copying:**
    - The primary OSD (or the remaining replicas) replicates data to the newly assigned OSDs.
4. **Acknowledgment:**
    - Once all replicas have been restored, the new OSDs acknowledge the recovery.

Example of Data Recovery:

- `osd.2` was storing a replica of PG `12.34`.
- `osd.2` fails.
- The CRUSH map remaps `12.34` to `osd.3`.
- Data is replicated from the primary OSD to `osd.3`.

---

4. Data Rebalancing in Ceph

Data rebalancing occurs when:

- A new OSD is added,
- An existing OSD is removed, or
- The CRUSH map is modified.

Rebalancing Steps:

1. **Recalculate PG Mapping:**
    - The CRUSH algorithm recalculates PG mappings to accommodate the new OSD configuration.
2. **Data Migration:**
    - Data is redistributed to balance the data load across all OSDs.
    - Only the affected PGs are moved, minimizing data movement.
3. **Monitoring and Adjustments:**
    - The Ceph Manager continuously monitors data distribution and adjusts PG mappings as necessary.

Example of Rebalancing:

- A new OSD (`osd.4`) is added.
- PGs are redistributed to `osd.4`.
- Some data from `osd.1`, `osd.2`, and `osd.3` is moved to `osd.4`.

---

5. Data Scrubbing and Backfilling

- **Scrubbing:**
    - Ceph periodically checks for data consistency by comparing data replicas.
    - If a mismatch is detected, data is corrected.
- **Backfilling:**
    - Backfilling occurs when a failed OSD recovers or a new OSD is added.
    - Data is copied to the newly available OSD to restore redundancy.

---

6. Example of Failure and Recovery Workflow:

Scenario:

- 5 OSDs (`osd.0` to `osd.4`), replication factor `3`.
- PG `23.45` is assigned to `osd.0`, `osd.1`, and `osd.2`.

Failure Event:

- `osd.1` fails.

Recovery Workflow:

1. **Detection:**
    - The MONs detect the absence of heartbeats from `osd.1` and mark it as **down**.
2. **Recalculation:**
    - CRUSH remaps PG `23.45` to `osd.3`.
3. **Data Recovery:**
    - The primary OSD (`osd.0`) replicates data to `osd.3`.
4. **Acknowledgment:**
    - `osd.3` acknowledges data receipt and becomes part of the PG replication set.

How Ceph Handles Specific Failure Scenarios:

Ceph is designed to handle multiple failure scenarios using its distributed architecture, CRUSH algorithm, replication/erasure coding, and self-healing mechanisms. Here, we will discuss the following scenarios in detail:

1 OSD Failure (Disk Failure)

2 Network Partitioning (Network Split or Node Isolation)

3 Monitor (MON) Failure

4 Ceph Manager (MGR) Failure

5 PG Inconsistent State (Data Corruption)

6 Cluster Full State (Out of Capacity)

1. OSD Failure (Disk Failure)

When an OSD fails, Ceph automatically initiates data recovery and rebalancing to maintain data redundancy and prevent data loss.

Steps to Handle OSD Failure:

1. **Failure Detection:**
    - The OSD fails or becomes unresponsive.
    - Other OSDs and MONs detect the failure via missing heartbeat messages.
    - The OSD is marked as **down** and then **out** by the monitor.
2. **PG Remapping:**
    - The MON updates the CRUSH map to remove the failed OSD.

    - PGs previously assigned to the failed OSD are remapped to available OSDs.
3. **Data Recovery and Backfilling:**
    - The primary OSD holding the data initiates the replication to a newly selected OSD.
    - Data is replicated until the number of replicas specified by the replication factor is restored.
4. **Rebalancing:**
    - Once data recovery is complete, Ceph rebalances the cluster to evenly distribute data.

Example:

- Cluster with 3 OSDs: `osd.0`, `osd.1`, `osd.2`.
- Replication factor: `3`.
- `osd.1` fails.
- PG `12.34` was previously mapped to `osd.0`, `osd.1`, `osd.2`.
- Now, `osd.1` is marked as `out`, and PG `12.34` is remapped to `osd.0`, `osd.2`, and `osd.3`.

---

2. Network Partitioning (Network Split or Node Isolation)

Network partitioning can isolate certain nodes or entire racks, leading to data unavailability or split-brain scenarios.

Steps to Handle Network Partitioning:

1. **Quorum Check:**
    - MONs continuously communicate with each other to maintain quorum.
    - If a MON loses connectivity with the majority of MONs, it will be marked as `out of quorum`.
2. **Handling Isolated OSDs:**
    - Isolated OSDs are marked as `down` by the remaining MONs.
    - PGs are remapped to OSDs that are still reachable and in quorum.
3. **Data Recovery and Rebalancing:**
    - Once the network is restored, isolated OSDs reconnect and synchronize missing data.
4. **Split-Brain Mitigation:**
    - Ceph uses the MON quorum to decide which data is authoritative and discards out-of-date replicas.

Example:

- Network partition separates `osd.1` and `osd.2` from the rest of the cluster.
- MON quorum is maintained by `mon.0`, `mon.1`, `mon.2`.
- `osd.1` and `osd.2` are marked as `down`.
- Data is rebalanced to other OSDs until the network is restored.

---

3. Monitor (MON) Failure

Ceph monitors maintain the cluster map and critical state information. Loss of MON quorum can lead to a cluster-wide outage.

Steps to Handle MON Failure:

1. **Quorum Management:**
   - MONs maintain quorum using a majority voting system.
   - If one MON fails but the remaining MONs maintain quorum, the cluster continues to operate normally.
2. **Automatic Re-election:**
   - If a MON fails, the remaining MONs adjust the quorum and continue monitoring the cluster.
3. **Recovery Process:**
   - The failed MON is either restarted or replaced with a new MON instance.
   - The cluster map is synchronized, and the new MON receives the latest state.

Example:

- 3 MONs: `mon.0`, `mon.1`, `mon.2`.
- `mon.1` fails.
- The remaining MONs (`mon.0`, `mon.2`) retain quorum and continue to operate.
- `mon.1` is recovered or replaced and re-synced with the latest state.

4. Ceph Manager (MGR) Failure

The MGR provides monitoring, dashboard, and other metrics services. It does not handle data but provides operational insight.

Steps to Handle MGR Failure:

1. **Failover to Standby MGR:**
   - Ceph typically deploys one active MGR and one or more standby MGRs.

    - If the active MGR fails, a standby MGR takes over automatically.

2. **Recovery and Re-Synchronization:**
    - The failed MGR is restarted or replaced.
    - It synchronizes with the active MGR to receive the latest cluster state.

Example:

- Active MGR: `mgr1`.
- Standby MGR: `mgr2`.
- `mgr1` fails, and `mgr2` becomes the new active MGR.

5. PG Inconsistent State (Data Corruption)

Data corruption can occur due to disk errors, network issues, or bugs. Ceph detects and corrects data inconsistencies.

Steps to Handle PG Inconsistency:

1. **Data Scrubbing:**
    - Ceph performs regular scrubbing (deep and shallow) to verify data integrity across replicas.
2. **PG Marked as Inconsistent:**
    - If a data mismatch is detected, the PG is marked as `inconsistent`.
3. **Data Recovery:**
    - The primary OSD requests the correct data from other replicas and overwrites the corrupted copy.
4. **Data Rebuild:**
    - If the corruption cannot be resolved using replicas, data is rebuilt using erasure coding.

Example:

- PG `12.34` is found to be inconsistent during a scrub operation.
- The primary OSD contacts other replicas to resolve the inconsistency.

6. Cluster Full State (Out of Capacity)

When the cluster runs out of capacity, data writes are blocked, but reads remain functional.

Steps to Handle Cluster Full State:

1. **Write Operations Blocked:**
    - When the cluster reaches 95% capacity, Ceph marks the cluster as `full`.

- Write operations are blocked to prevent data corruption.

2. **Data Deletion and Rebalancing:**
    - Data must be deleted or additional storage must be added to reduce the cluster utilization.
3. **Rebalancing:**
    - Once space is freed up, data is rebalanced to ensure even distribution.

Example:

- Cluster reaches 95% capacity.
- Admin deletes unnecessary data or adds more OSDs.
- Ceph rebalances data to utilize the new storage.

Commands:
 Identify missing or down OSDs:
ceph osd ls --down
ceph osd ls –out
ceph osd log osd.<OSD_ID> --limit 100

Deploying a Ceph Cluster Using Cephadm – Detailed Step-by-Step Guide

Cephadm is the official deployment tool for Ceph, introduced in Ceph Octopus (v15). It uses containers to deploy and manage Ceph daemons, making it easier to manage large-scale clusters. Here, we will cover:

1. Environment Preparation
2. Install Cephadm
3. Bootstrap the Ceph Cluster
4. Deploy MONs and MGRs

```
ceph orch host add ceph2 192.168.1.132
ceph orch host add ceph3 192.168.1.133

ceph orch apply mon --placement "ceph1,ceph2,ceph3"

ceph orch apply mgr --placement "ceph1,ceph2"
```

5. Deploy OSDs
6. Configure Networking

7. Deploy CephFS and RBD Pools
8. Deploy RGW (Optional for Object Storage)
9. Verify Cluster Status and Monitoring

Different Ways to Configure a Ceph Cluster

0 **Manual Deployment and Configuration**

1. **Cephadm (Containerized Deployment and Management)**
2. **Ceph-Ansible (Ansible-Based Deployment)**
3. **Rook (Kubernetes Native Deployment)**
4. **Ceph-Docker (Legacy Containerized Deployment)**
5. **Ceph-Orchestrator (CLI-Based Management for Cephadm)**
6. **Ceph Deploy (Legacy Method, Deprecated)**

Summary and Recommendations

Method	Use Case	Pros	Cons
Manual	Learning, Testing	Full control	Time-consuming
Cephadm	Production, Scalable	Automated, Containerized	Requires container runtime
Ceph-Ansible	Production, Large Deployments	Highly customizable	Requires Ansible
Rook	Kubernetes Native	Kubernetes-native, Auto-scaling	Complex to manage
Ceph-Docker	Legacy, Small Setups	Easy to set up	Deprecated
Ceph-Orchestrator	Cephadm Management	Dynamic, CLI-based	Requires Cephadm
Ceph Deploy	Legacy, Deprecated	Simple CLI	Deprecated

Ensuring the Security of a Ceph Cluster – Detailed Guide

Ceph is a highly scalable, distributed storage system. However, its default configurations are not fully secure. To secure a Ceph cluster, we need to consider the following key areas:

1. **Network Security**
2. **Authentication and Access Control**
3. **Data Encryption**

1. Network Security

Network security is crucial in a distributed storage system like Ceph. Ensuring secure communication between nodes can prevent unauthorized access and data breaches.

1.1. Isolate Networks:

- Use separate networks for different Ceph traffic types:
  - **Public Network:** For client-facing services (e.g., RBD, CephFS, RGW).
  - **Cluster Network:** For internal Ceph communications (MON, OSD, MGR).

Example Configuration in `ceph.conf`:

```
[global]
public_network = 192.168.1.0/24
cluster_network = 10.0.0.0/24
```

1.2. Network Encryption:

- Use `cephx` authentication to secure network communication.
- Implement TLS/SSL for client-to-cluster and inter-daemon communication.

✅ 2. Authentication and Access Control

Ceph uses `cephx` for authentication and authorization. It is enabled by default but must be configured properly.

2.1. Enable cephx Authentication:

- Verify that `auth cluster`, `auth service`, and `auth client` are set to `cephx`:

```
[global]
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
```

- Restart Ceph services to apply changes:

```
sudo systemctl restart ceph-mon.target ceph-osd.target ceph-mgr.target
```

2.2. Generate and Manage Keys:

- Generate a new key for a specific client (e.g., admin user):

```
ceph auth get-or-create client.admin mon 'allow *' osd 'allow *' mgr 'allow *'
```

- List all keys:

```
ceph auth list
```

- Revoke a key:

```
ceph auth del client.admin
```

✅ 3. Data Encryption

Data encryption ensures that data is protected both **in transit** and **at rest**.

3.1. Encryption in Transit:

- Enable TLS for Ceph daemons using `cephadm`:

```
ceph config set global mgr/cephadm/tls true
ceph config set global mgr/cephadm/tls_cert /etc/ceph/ceph-cert.pem
ceph config set global mgr/cephadm/tls_key /etc/ceph/ceph-key.pem
```

- Apply to all daemons:

```
ceph orch restart mon
ceph orch restart osd
ceph orch restart mgr
```

3.2. Encryption at Rest:

- Enable encryption for OSDs during creation:

```
cephadm shell
ceph-volume lvm create --data /dev/sdb --dmcrypt
```

- Verify encryption status:

```
lsblk -o NAME,TYPE,SIZE,FSTYPE,UUID
```

✅ 4. TLS/SSL Configuration

TLS/SSL provides secure communication between Ceph clients and daemons.

4.1. Generate TLS Certificates:

- Install `openssl` and generate certificates:

```
openssl genrsa -out /etc/ceph/ceph-key.pem 2048
openssl req -new -key /etc/ceph/ceph-key.pem -out /etc/ceph/ceph.csr
openssl x509 -req -days 365 -in /etc/ceph/ceph.csr -signkey /etc/ceph/ceph-
key.pem -out /etc/ceph/ceph-cert.pem
```

4.2. Configure Daemons to Use TLS:

- Add TLS configuration to `ceph.conf`:

```
[global]
mon_host = 192.168.1.131
mgr/cephadm/tls = true
mgr/cephadm/tls_cert = /etc/ceph/ceph-cert.pem
mgr/cephadm/tls_key = /etc/ceph/ceph-key.pem
```

✅ 5. Monitoring and Auditing

Monitoring and auditing are essential to detect unauthorized access and potential security breaches.

5.1. Enable Audit Logging:

- Enable logging for all daemons:

```
[global]
log_to_file = true
log_file = /var/log/ceph/ceph.log
log_max_files = 5
log_max_file_size = 1000000
```

5.2. Enable Ceph Telemetry (Optional):

- Enable telemetry to receive security advisories:

```
ceph telemetry on
```

5.3. Monitor with Ceph Dashboard:

- Ceph Dashboard provides real-time monitoring and alerts.
- Access via:
- `https://<admin-node-ip>:8443`

✅ 6. Hardening Ceph Nodes

Hardening the OS and securing the network is crucial.

6.1. Disable Unnecessary Services:

```
sudo systemctl disable avahi-daemon
sudo systemctl disable rpcbind
```

6.2. Apply Kernel Hardening:

- Modify `/etc/sysctl.conf`:

```
net.ipv4.ip_forward = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
```

- Apply changes:

```
sudo sysctl -p
```

✅ 7. Secure Configuration of Ceph Daemons

7.1. Restrict Client Capabilities:

- Assign minimal privileges to users:

```
ceph auth get-or-create client.rgw rgw 'allow rwx' osd 'allow rwx pool=rgw_pool'
```

7.2. Remove Unused Keys and Capabilities:

- List all capabilities:

```
ceph auth list
```

- Remove unnecessary capabilities:

```
ceph auth del client.unused
```

✅ 8. Implementing Network Isolation and Firewalls

8.1. Isolate Ceph Networks:

- Use VLANs or dedicated interfaces for Ceph traffic.
- Separate public and cluster networks using distinct subnets.

8.2. Configure Firewalls:

- Open specific Ceph ports:

```
sudo firewall-cmd --add-port=6789/tcp --permanent   # MON
sudo firewall-cmd --add-port=6800-7300/tcp --permanent   # OSD
sudo firewall-cmd --reload
```

✅ 9. Backup and Disaster Recovery

9.1. Backup Ceph Configuration:

```
sudo tar -czvf ceph-config-backup.tar.gz /etc/ceph/
```

9.2. Backup MON Map:

```
ceph mon getmap -o /etc/ceph/monmap-backup
```

9.3. Backup Keyrings:

```
sudo cp /etc/ceph/ceph.client.admin.keyring /backup/
```

Monitoring the Health and Performance of a Ceph Cluster

Monitoring a Ceph cluster is crucial to ensure its stability, performance, and data integrity. Ceph provides built-in monitoring tools, but we can also integrate external monitoring systems to gain more detailed insights. Here’s a comprehensive approach to monitoring a Ceph cluster:

Health Status Codes:

- `HEALTH_OK`: Cluster is healthy.
- `HEALTH_WARN`: Potential issues, such as near-full OSDs.
- `HEALTH_ERR`: Critical issues, such as OSD down or data unavailability.

# **Troubleshooting:**

Troubleshooting Slow Performance in a Ceph Cluster – Detailed Guide

Slow performance in a Ceph cluster can be caused by various factors such as network issues, disk bottlenecks, misconfigured parameters, or resource contention. Here’s a structured, step-by-step approach to diagnose and resolve slow performance in a Ceph cluster:

Check Cluster Health

Look for warnings or errors such as:

`HEALTH_WARN` or `HEALTH_ERR`

- Slow requests
- Inconsistent PGs
- OSDs down

Check PG State and Distribution

Check placement group (PG) states to ensure they are not stuck, undersized, or degraded:

Check OSD and Disk Status

Analyze disk health using `smartctl`:

```
sudo smartctl -a /dev/sdb
```

Check for high I/O wait on disks:

```
iostat -xd 1
```

Analyze Performance Metrics

Ceph provides several commands to analyze latency and throughput.

ceph osd perf

Check Pool Utilization:

Check Network Latency:

Adjust PG Count:

Check Ceph Logs:

##Data Recovery in Ceph Cluster – Detailed Step-by-Step Guide

Data recovery in a Ceph cluster involves multiple steps depending on the type of failure, such as OSD failure, PG inconsistency, data corruption, or MON failure. Here’s a comprehensive approach to data recovery in a Ceph cluster:

1. Identify the Type of Failure

First, assess the failure type to determine the appropriate recovery approach.

1.1. Check Cluster Health

Run the `ceph status` or `ceph -s` command:

```
ceph -s
```

Example output:

```
  cluster:
    id: 12345678-1234-1234-1234-123456789abc
    health: HEALTH_ERR
    Reduced data availability: 3 pgs inactive
```

- Look for critical warnings:
    - `HEALTH_WARN`: Potential issues detected.
    - `HEALTH_ERR`: Data unavailability or critical failures.
    - `PG DEGRADED`: Indicates data loss or corruption.
    - `PG INCOMPLETE`: PGs are missing or incomplete.

1.2. Check Placement Group (PG) State

Identify problematic PGs:

```
ceph pg dump | grep -E "active|degraded|incomplete"
```

- `degraded`: PGs with missing objects.
- `incomplete`: PGs that are not fully replicated.

Check the specific PG state:

```
ceph pg <pg-id> query
```

2. Recover from OSD Failures

If an OSD fails, Ceph automatically attempts to recover data. If recovery is slow or stuck, take manual steps.

2.1. Check OSD Status

Verify OSD status:

```
ceph osd tree
ceph osd stat
```

Example output:

```
ID  WEIGHT  STATE  TYPE NAME
-1        10.00000  up   root default
-3        5.00000   down  host ceph1
 0   1.00000  down  osd.0
 1   1.00000  up    osd.1
```

- `down`: OSD is offline.
- `out`: OSD is marked as out of the cluster.

2.2. Restart the OSD

If the OSD is down, try to restart it:

```
systemctl restart ceph-osd@<osd-id>.service
```

- Check OSD logs:

```
journalctl -u ceph-osd@<osd-id>.service -f
```

2.3. Mark OSD as `in` or `out`

If the OSD is down but the hardware is functional, reintroduce it:

```
ceph osd in <osd-id>
```

If the OSD is dead and needs to be removed:

```
ceph osd out <osd-id>
```

- Reweight the OSD to 0 to prevent data allocation:

```
ceph osd reweight <osd-id> 0
```

2.4. Rebalance Data After OSD Removal

If an OSD is permanently removed, Ceph rebalances data. Monitor rebalancing:

```
ceph -s
ceph osd df tree

#If the rebalancing is too slow, increase backfill settings:

ceph config set osd osd_max_backfills 4
ceph config set osd osd_recovery_max_active 4
```

---

3. Recover from PG Incomplete State

If a PG is incomplete, data recovery becomes more complex.

3.1. Inspect the PG:

Identify missing OSDs:

```
ceph pg <pg-id> query
```

Example output:

```
{
  "state": "incomplete",
  "acting": [1, 2],
  "up": [1, 2],
  "missing": [0]
}
```

- Missing OSDs are potential sources of data loss.

3.2. Mark Missing OSDs as Lost

If the missing OSDs are irrecoverable, mark them as lost:

```
ceph pg <pg-id> mark_unfound_lost revert
```

- `revert`: Attempt to recover as much data as possible.
- `delete`: Permanently remove lost objects.

---

4. Recover Data from Pool Snapshots

If data loss occurred due to accidental deletion, restore data from a snapshot.

4.1. List Snapshots:

```
rados -p <pool-name> ls
```

- Identify the snapshot ID.

4.2. Restore Snapshot:

```
rados rollback -p <pool-name> <object-id> <snapshot-id>
```

- Example:

```
rados rollback -p mypool myobject mysnapshot
```

---

5. Recover Data from CephFS

If a CephFS filesystem is corrupted or inaccessible:

5.1. Check CephFS Health:

```
ceph fs status
```

5.2. Mount the CephFS:

```
mkdir /mnt/cephfs
mount -t ceph <mon-ip>:6789:/ /mnt/cephfs
```

- Verify mount point:

```
ls /mnt/cephfs
```

5.3. Use cephfs-data-scan for Recovery

If data is missing, run:

```
cephfs-data-scan start /mnt/cephfs
```

- After scanning, replay logs:

```
cephfs-journal-tool journal inspect
cephfs-journal-tool journal replay
```

---

6. Recover Data from RBD Images

If RBD images are corrupted or missing:

6.1. List RBD Images:

```
rbd ls <pool-name>
```

6.2. Recover Deleted RBD Image:

- If a snapshot exists:

```
rbd snap ls <pool-name>/<image-name>
rbd snap rollback <pool-name>/<image-name>@<snapshot-name>
```

---

7. Recover from MON Failure

If a monitor node fails, the cluster may lose quorum.

7.1. Verify MON Status:

```
ceph mon stat
```

7.2. Restart the MON Service:

```
systemctl restart ceph-mon@<mon-id>.service
```

7.3. Rebuild a Failed MON:

If the MON is unrecoverable:

1. Remove the old MON:

```
ceph mon remove <mon-id>
```

2. Deploy a new MON:

```
ceph orch apply mon --placement <node>
```

---

8. Monitor Recovery Progress

During recovery, monitor cluster health and recovery speed:

```
ceph -s
ceph health detail
ceph osd pool stats
```

- Watch for slow requests:

```
ceph health detail | grep "slow request"
```

##Data Migration in Ceph Cluster – Detailed Step-by-Step Guide

Data migration within a Ceph cluster typically involves moving data between OSDs, pools, or storage devices. This can be necessary when rebalancing data, upgrading hardware, or adjusting pool configurations. Here’s a detailed approach to managing data migration in a Ceph cluster:

1. Understanding Data Placement and Migration in Ceph

Ceph uses the **CRUSH algorithm** to determine data placement across OSDs. Data migration may be required in the following scenarios:

- Adding or removing OSDs.
- Reweighting OSDs to balance data distribution.
- Adjusting the number of Placement Groups (PGs).
- Migrating data to a new pool or device class (e.g., HDD to SSD).

2. Data Migration Scenarios and Strategies

Scenario	Migration Strategy	Tool/Command
Adding a New OSD	Rebalance data across OSDs	`ceph osd reweight`
Removing an OSD	Migrate data off the OSD	`ceph osd out`
Adjusting PG Count	Redistribute PGs	`ceph osd pool set`
Migrating Pool Data	Copy data to a new pool	`rados cppool`
Changing Pool Configuration	Rebalance using CRUSH map	`ceph osd crush reweight`
Migrating RBD Images	Export and import RBD data	`rbd export/import`

3. Rebalancing Data Across OSDs

4. Migrating Data Between Pools

4.1. Copy Data Using `rados cppool`

Syntax:

```
rados cppool <source-pool> <destination-pool>
```

- Example:

```
rados cppool pool1 pool2
```

- Verify data migration:

```
rados ls -p pool2
```

4.2. Changing Pool Replication Settings

If you adjust replication settings, data may need to be redistributed.

- Change the replication size:

```
ceph osd pool set <pool-name> size 3
```

- Adjust the minimum replication size:

```
ceph osd pool set <pool-name> min_size 2
```

- Monitor data movement:

```
ceph -s
```

---

5. Migrating Data Between Device Classes (SSD/HDD)

If you have different device classes (e.g., HDD, SSD), you can use the CRUSH map to migrate data.

5.1. Verify Device Classes:

```
ceph osd crush tree
```

- Example output:

```
ID  CLASS WEIGHT  TYPE NAME
-1       10.00000 root default
-2   ssd 4.00000  host node1
-3   hdd 6.00000  host node2
```

5.2. Create a New Pool with a Specific Device Class:

```
ceph osd pool create ssd_pool 128
ceph osd pool set ssd_pool crush_rule ssd_rule
```

- Assign the SSD class rule to the pool:

```
ceph osd pool application enable ssd_pool rados
```

5.3. Migrate Data to the New Pool:

- Copy data:

```
rados cppool old_pool ssd_pool
```

- Verify data migration:

```
rados ls -p ssd_pool
```

---

6. Migrating RBD Images

RBD images can be migrated by exporting and importing them.

6.1. Export RBD Image:

```
rbd export <pool>/<image> /tmp/image.img
```

- Example:

```
rbd export pool1/myimage /tmp/myimage.img
```

6.2. Import RBD Image to a New Pool:

```
rbd import /tmp/myimage.img <pool>/<new-image>
```

- Verify the imported image:

```
rbd ls <pool>
```

7. Migrating Data Between Ceph Clusters

Data migration between clusters can be achieved using `radosgw` or by exporting and importing data.

7.1. Using `radosgw` for Object Data:

- Sync data using `radosgw-admin`:

```
radosgw-admin zonegroup update --rgw-zonegroup=<zonegroup> --rgw-zone=<zone> --
rgw-remote-zone=<remote-zone>
```

8. Monitoring Data Migration Progress

During data migration, monitoring is crucial to ensure data integrity and availability.

- Monitor data movement:

```
ceph -s
```

- Check the PG states:

```
ceph pg dump | grep -i "active|degraded"
```

- Monitor OSD usage:

```
ceph osd df tree
```

##Openstack
Do you have experience with Ceph in cloud environments (e.g., OpenStack, Kubernetes)

Yes, I have extensive experience with Ceph in cloud environments, particularly in the context of **OpenStack** and **Kubernetes**.

✅ Ceph in OpenStack:

- **Ceph as a Backend for OpenStack Services:**
    - **Glance (Image Service):** Ceph provides a backend for storing images using RBD (RADOS Block Device).
    - **Cinder (Block Storage):** Ceph RBD is used as the backend for persistent block storage.
    - **Nova (Compute):** Ceph is integrated with Nova to provide ephemeral storage using RBD.
    - **Manila (File Share Service):** CephFS can be used to provide shared file storage.

  - **Swift (Object Storage):** Ceph RGW (RADOS Gateway) can serve as a drop-in replacement for OpenStack Swift.
- **Deployment and Integration:**
  - Ceph can be deployed using **Kolla-Ansible** or manually using **Cephadm** or **Ceph-Ansible**.
  - Integration involves configuring `/etc/kolla/globals.yml`, setting up Ceph keyrings, and adjusting the OpenStack services to use Ceph endpoints.

Ceph Integration with OpenStack Services – Deployment Scenarios

Which specific service would you like to focus on? Here are some common deployment scenarios:

1. **Glance (Image Service):** Store images in Ceph RBD pools.
2. **Cinder (Block Storage):** Use Ceph RBD for volume storage.
3. **Nova (Compute):** Configure ephemeral storage on Ceph RBD.
4. **Manila (File Shares):** Use CephFS for shared file storage.
5. **Swift (Object Storage):** Replace Swift with Ceph RGW.
6. **Gnocchi (Telemetry):** Store metrics data in Ceph pools.

##**Differences Between Ceph Quincy and Ceph Reef Releases**

2. Performance Enhancements:

- **Reef:**
  - Optimized BlueStore for faster write throughput and lower latency.
  - Enhanced multi-site replication to reduce latency in geo-distributed clusters.
  - Improved RADOS namespace sharding, reducing contention and increasing concurrency.
- **Quincy:**
  - BlueStore optimizations for small I/O workloads.
  - RBD journaling improvements for better data consistency.
  - Optimized cache tiering logic for lower latency.

✅ 3. Cephadm and Orchestration:

- **Quincy:**
  - First release to fully adopt `cephadm` as the primary deployment and management tool.
  - Basic monitoring and container orchestration support.
- **Reef:**
  - Enhanced `cephadm` with advanced scheduling logic.
  - Support for external services (e.g., Prometheus, Grafana) directly via Cephadm.
  - Improved rollback and recovery mechanisms for `cephadm` managed clusters.

✅ 4. Object Storage (RGW) Enhancements:

- **Reef:**
  - S3 Object Lock and Legal Hold for compliance with regulatory standards.
  - Multi-site replication with EC support, reducing data storage costs.
- **Quincy:**
  - Basic multi-site synchronization with bucket versioning.
  - Support for S3 lifecycle policies and data expiration.

✅ 5. CephFS Improvements:

- **Reef:**
  - CephFS snapshot mirroring for cross-cluster failover.
  - Enhanced metadata cache for better read performance.
- **Quincy:**
  - Initial support for NFS v4 exports.
  - Directory-based quotas and subvolume management.

✅ 6. Security and Compliance:

- **Reef:**
  - S3 object encryption at rest using server-side encryption (SSE).
  - Data integrity checks and compliance audits.
- **Quincy:**

    - Basic RBAC policies for user and service management.
    - SSL/TLS support for RGW endpoints.

✅ 7. Upgrade Considerations:

- **Upgrading from Quincy to Reef:**
    - Requires BlueStore database format upgrade.
    - Potential changes in CRUSH map configuration due to multi-site EC support.
    - RBD mirroring upgrades may require additional configuration for async replication.

## What are OSDs in Ceph? How do they interact with other components?

OSDs are the heart of a Ceph cluster. Each OSD daemon is responsible for managing a disk or storage device. OSDs store data, provide access to it, and handle data replication and recovery.

- **Interactions with other components**:
    - OSDs work with **Monitors (MONs)** to maintain the cluster's state and ensure consistency.
    - OSDs interact with **Ceph Manager** for monitoring and data statistics.
    - In **CephFS**, OSDs store the actual file data, while **MDS** handles the file system metadata.
    - They replicate data across other OSDs for fault tolerance.

What happens if an OSD fails?

When an OSD fails, Ceph automatically detects the failure through **OSD heartbeats**. The failed OSD will be marked as "down" in the cluster.

- Ceph will then initiate **recovery** by using the replica or erasure-coded data from other OSDs in the cluster.
- Data that was previously stored on the failed OSD will be redistributed to healthy OSDs according to the **CRUSH map**. This process ensures data availability and redundancy without data loss.
- The **recovery process** might involve copying or rebalancing data to another OSD, depending on the replication or erasure coding scheme used.

How does Ceph handle data replication and fault tolerance?

Replication: Ceph replicates data across multiple OSDs. By default, it uses a **replication factor** of 3, meaning each piece of data is stored on 3 different OSDs. This ensures redundancy and protection against failures.

- **Fault Tolerance**: When a failure occurs (such as a disk, server, or network failure), Ceph ensures data availability by automatically creating new copies of the lost data on other healthy OSDs. The replication process is managed by the **CRUSH algorithm**, which determines the placement of data across the cluster.
- **Erasure Coding**: As an alternative to replication, Ceph can use **erasure coding** to store data in a more space-efficient manner. It splits data into fragments and adds redundancy in the form of coding fragments. This allows data recovery even if some fragments are lost.

What is the difference between BlueStore and FileStore

FileStore: An older backend that uses a filesystem (typically XFS or ext4) to store objects on disk. While it provides basic functionality, FileStore is slower than BlueStore.

- **BlueStore**: The default backend introduced in Ceph Kraken (v10.2). It provides a more efficient and higher-performing storage backend by using raw devices (instead of a filesystem). It avoids the overhead of a filesystem and integrates with Ceph’s internal storage, improving I/O performance, scalability, and durability.

How do you configure CephFS?

Create a CephFS pool to store data. This is typically done using `ceph osd pool create` or through a pre-configured pool if using Ceph's default pools.

1. **Create a CephFS filesystem** using the `ceph fs new` command. This command will configure both the data pool and metadata pool.
2. **Set up MDS daemons** (Metadata Servers). MDS daemons manage the metadata of the CephFS filesystem, which includes directories, files, and permissions.
3. **Mount CephFS** on client systems by using the `ceph-fuse` tool or a kernel mount if using Ceph’s kernel client support.

What is RBD in Ceph and how does it work?

RBD (RADOS Block Device) is a block storage interface in Ceph. It allows clients to use block storage devices from Ceph as if they were local disks. RBD is often used in environments like **OpenStack**, where it provides persistent storage for virtual machines.

- **How it works**:

- RBD images are created and stored in Ceph pools, which are distributed and replicated across the cluster.
- RBD clients access the storage through a network, typically using a Ceph client or the `radosgw` gateway.
- Features like **RBD snapshots** and **cloning** allow for efficient data management.

What is the RADOS Gateway (RGW) and how does it integrate with Ceph?

The **RADOS Gateway (RGW)** provides a RESTful interface to Ceph, making Ceph compatible with object storage APIs such as **S3** and **Swift**. This allows applications that rely on these APIs to store and retrieve data from a Ceph cluster.

- **RGW integrates with Ceph** by directly interacting with the underlying RADOS (Reliable Autonomic Distributed Object Store) layer to store objects.
- RGW provides features like **multi-site synchronization**, **bucket versioning**, and **object lifecycle management**

What are common performance bottlenecks in Ceph?

Disk I/O: Ceph is very sensitive to disk performance. Slow or over-utilized disks (especially in OSDs) can be a major bottleneck.

- **Network Latency**: Ceph relies heavily on network performance, especially for communication between OSDs and monitors. High latency can lead to slow performance.
- **CPU Overload**: Ceph's processes can consume significant CPU resources, especially during recovery or rebalancing operations.
- **Memory Usage**: Insufficient memory can cause the Ceph daemons to become slow or unresponsive, leading to performance issues.

How do you secure a Ceph cluster?

Authentication: Use **CephX** for authentication between clients and the cluster. CephX ensures that only authorized clients can access the data.

- **Encryption**: Enable **encryption at rest** to protect stored data, and **encryption in transit** to protect data during transmission. Ceph supports SSL/TLS for encryption.
- **RBAC (Role-Based Access Control)**: Ceph allows you to implement RBAC for controlling access to specific resources (e.g., pools or RBD images).
- **Key management**: Use **cephadm** or **Ceph Keyring** management to securely store and distribute credentials.

How would you perform backup and restore operations in Ceph?

Backup:

  - **RBD snapshots**: Use `rbd snap create` to create point-in-time backups of RBD images.
  - **CephFS snapshots**: Use the `ceph fs snapshot` command to take backups of CephFS data.
  - **Export objects**: You can use RGW's S3 API to export objects or sync them to another storage solution.

- **Restore**:
  - **RBD restore**: Use `rbd snap rollback` to restore a previous snapshot.
  - **CephFS restore**: Use the `ceph fs restore` command to restore data from snapshots.
  - **Multi-site replication**: In case of RGW multi-site replication, you can synchronize objects between Ceph clusters.

How the CRUSH Algorithm Uses `crush_weight`

The **CRUSH algorithm (Controlled Replication Under Scalable Hashing)** is a data placement algorithm used by Ceph to distribute data across OSDs in a balanced manner. The `crush_weight` plays a crucial role in determining how much data each OSD stores relative to its capacity.

1. **Data Placement:**
   - When a placement group (PG) is mapped to OSDs, CRUSH considers the `crush_weight` of each OSD.
   - OSDs with higher weights will receive more PGs and thus more data.
   - The algorithm ensures that data distribution is proportional to the weight of each OSD.
2. **Balancing Data:**
   - The `crush_weight` ensures that data is balanced according to the capacity of each OSD.
   - If OSD A has a weight of `2.0` and OSD B has a weight of `1.0`, OSD A will receive twice as much data as OSD B.
3. **Data Rebalancing:**
   - When OSDs are added, removed, or their weights are adjusted, CRUSH recalculates data distribution to maintain the balance.

- A change in `crush_weight` triggers data rebalancing to align data placement with the new capacity distribution.

Example:

Assume a Ceph cluster with three OSDs:

OSD	Crush Weight	Capacity (TB)
osd.0	1.0	1.0
osd.1	2.0	2.0
osd.2	0.5	0.5

- Data distribution will be in the ratio `1 : 2 : 0.5`.
- `osd.1` will receive the most data, `osd.0` will receive a moderate amount, and `osd.2` will receive the least.