Cloudera Runtime 7.1.8

# Apache Hadoop Ozone Overview

**Date published: 2020-08-11**
**Date modified: 2022-07-26**

**https://docs.cloudera.com/**

# Legal Notice

# Contents

**Introduction to Ozone ..... 4**
Ozone architecture ..... 5
Ozone security architecture ..... 7
Authentication in Ozone ..... 7
Authorization in Ozone ..... 11
Ozone containers ..... 12
How Ozone manages read operations ..... 13
How Ozone manages write operations ..... 13
How Ozone manages delete operations ..... 13

# Introduction to Ozone

Apache Ozone is a scalable, redundant, and distributed object store optimized for big data workloads. Apart from scaling to billions of objects of varying sizes, applications that use frameworks like Apache Spark, Apache YARN and Apache Hive work natively on Ozone object store without any modifications.

HDFS works best when most of the files are large but HDFS suffers from small file limitations. Ozone is a distributed key-value object store that can manage both small and large files alike. Ozone natively supports the S3 API and provides a Hadoop-compatible file system interface. Ozone object store is available in a CDP Private Cloud Base deployment.

## Ozone storage elements

Ozone consists of the following important storage elements:

- Volumes

  Volumes are similar to accounts. Volumes can be created or deleted only by administrators. An administrator creates a volume for an organization or a team.
- Buckets

  A volume can contain zero or more buckets. Ozone buckets are similar to Amazon S3 buckets. Depending on their requirements, regular users can create buckets in volumes.
- Keys

  Each key is part of a bucket. Keys are unique within a given bucket and are similar to S3 objects. Ozone stores data as keys inside buckets.

When a client writes a key, Ozone stores the associated data on DataNodes in chunks called blocks. Therefore, each key is associated with one or more blocks. Within a DataNode, multiple unrelated blocks can reside in a storage container.

## Key features of Ozone

Key features of Ozone are:

- Consistency

  Strong consistency simplifies application design. Ozone object store is designed to provide strict serializability.
- Architectural simplicity

  A simple architecture is easy to use and easy to debug when things go wrong. The architecture of Ozone object store is simple and at the same time scalable. It is designed to store over 100 billion objects in a single cluster.
- Layered architecture

  The layered file system of Ozone object store helps to achieve the scale required for the modern storage systems, . It separates the namespace management from block and node management layer, which allows users to independently scale on both axes.
- Easy recovery

  A key strength of HDFS is that it can effectively recover from catastrophic events like cluster-wide power loss without losing data and without expensive recovery steps. Rack and node losses are relatively minor events. Ozone object store is similarly robust in the face of failures.
- Open source in Apache

  The Apache Open Source community is critical to the success of Ozone object store. All Ozone design and development are being done in the Apache Hadoop community.

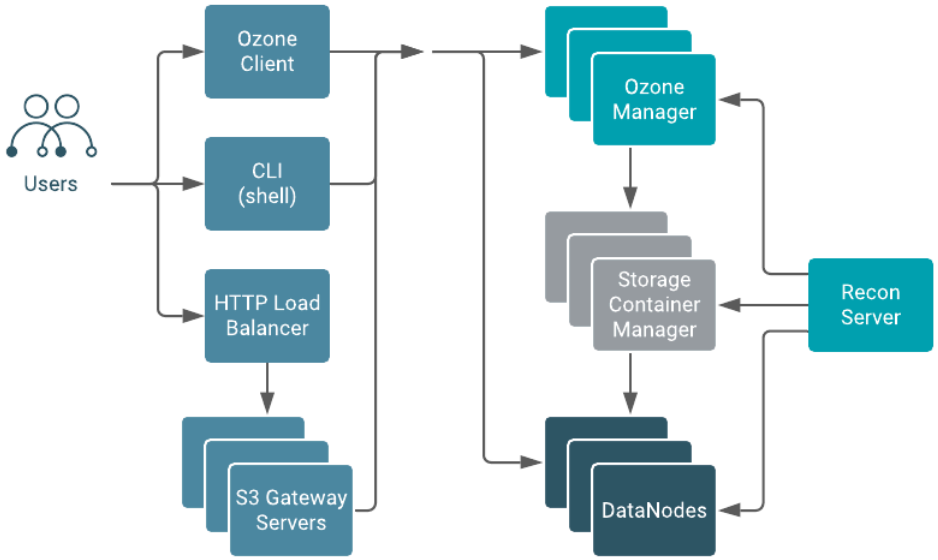
- Interoperability with Hadoop ecosystem

  Ozone object store is usable by the existing Apache Hadoop ecosystem and related applications like Apache Hive, Apache Spark and traditional MapReduce jobs.

## Ozone architecture

Ozone can be co-located with HDFS with single security and governance policies for easy data exchange or migration and also offers seamless application portability. Ozone has a scale-out architecture with minimal operational overheads. Ozone separates management of namespaces and storage, helping it to scale effectively. The Ozone Manager (OM) manages the namespaces while the Storage Container Manager (SCM) handles the containers. All the metadata stored on OM SCM and data nodes are required to be stored in low latency devices like NVME or SSD.

The following diagram shows the components that form the basic architecture of Ozone:




**Ozone Manager**

The Ozone Manager (OM) is a highly available namespace manager for Ozone. OM manages the metadata for volumes, buckets, and keys. OM maintains the mappings between keys and their corresponding Block IDs. When a client application requests for keys to perform read and write operations, OM interacts with SCM for information about blocks relevant to the read and write operations, and provides this information to the client.

OM uses Apache Ratis (Raft protocol) to replicate Ozone manager state. While RocksDB (embedded storage engine) persists the metadata or keyspace, the flush of the Ratis transaction to the local disk ensures data durability in Ozone. Therefore, a low-latency local storage device like NVMe SSD on each OM node for maximum throughput is required. A typical Ozone deployment has three OM nodes for redundancy.

**DataNodes**

Ozone DataNodes, not the same as HDFS DataNodes, store the actual user data in the Ozone cluster. DataNodes store the blocks of data that clients write. A collection of these blocks is a storage container. The client streams data in the form of fixed-size chunk files (4MB) for a block. The chunk files constitute what gets actually written to the disk. The DataNode sends heartbeats to SCM at fixed time intervals. With every heartbeat, the DataNode sends container reports.

Every storage container in the DataNode has its own RocksDB instance which stores the metadata for the blocks and individual chunk files. Every DataNode can be a part of one or more active pipelines. A pipeline of DataNodes is actually a Ratis quorum with an elected leader and follower

DataNodes accepting writes. Reads from the client go directly to the DataNode and not over Ratis. An SSD on a DataNode to persist the Ratis logs for the active pipelines and significantly boosts the write throughput is required.

**Storage Container Manager**

The Storage Container Manager (SCM) is a master service in Ozone.

A storage container is the replication unit in Ozone. Unlike HDFS, which manages block-level replication, Ozone manages the replication of a collection of storage blocks called storage containers. The default size of a container is 5 GB. SCM manages DataNode pipelines and placement of containers on the pipelines. A pipeline is a collection of DataNodes based on the replication factor. For example, given the default replication factor of three, each pipeline contains three DataNodes. SCM is responsible for creating and managing active write pipelines of DataNodes on which the block allocation happens.

The client directly writes blocks to open containers on the DataNode, and the SCM is not directly on the data path. A container is immutable after it is closed. SCM uses RocksDB to persist the pipeline metadata and the container metadata. The size of this metadata is much smaller compared to the keyspace managed by OM.

SCM is a highly available component which makes use of Apache Ratis. Cloudera recommends an SSD on SCM nodes for the Ratis write-ahead log and the RocksDB.

A typical Ozone deployment has three SCM nodes for redundancy. SCM service instances can be collocated with OM instances; therefore, you can use the same master nodes for both SCM and OM.

**Recon Server**

Recon is a centralized monitoring and management service within an Ozone cluster that provides information about the metadata maintained by different Ozone components such as OM and SCM.

Recon takes a snapshot of the OM and SCM metadata while also receiving heartbeats from the Ozone DataNode. Recon asynchronously builds an offline copy of the full state of the cluster in an incremental manner depending on how busy the cluster is. Recon usually trails the OM by a few transactions in terms of updating its snapshot of the OM metadata. Cloudera recommends using an SSD for maintaining the snapshot information because an SSD would help Recon in staying updated with the OM.

**Ozone File System Interfaces**

Ozone is a multi-protocol storage system with support for the following interfaces:

- s3: Amazon’s Simple Storage Service (S3) protocol. You can use S3 clients and S3 SDK-based applications without any modifications on Ozone with S3 Gateway.
- o3: An object store interface that can be used from the Ozone shell.
- ofs: A Hadoop-compatible filesystem (HCFS) allowing any application that expects an HDFS-like interface to work against Ozone with no API changes. Frameworks like Apache Spark, YARN and Hive work against Ozone without the need of any change.
- o3fs: (Deprecated, not recommended) A bucket-rooted Hadoop-compatible filesystem (HCFS) interface.

**S3 Gateway**

S3 gateway is a stateless component that provides REST access to Ozone over HTTP and supports the AWS-compatible s3 API. S3 gateway supports multipart uploads and encryption zones.

In addition, S3 gateway transforms the s3 API calls over HTTP to rpc calls to other Ozone components.

To scale your S3 access, Cloudera recommends deploying multiple gateways behind a load balancer like haproxy that support Direct Server Return (DSR) so that the load balancer is not on the data path.

# Ozone security architecture

Apache Ozone is a scalable, distributed, and high performance object store optimized for big data workloads and can handle billions of objects of varying sizes. Applications that use frameworks like Apache Spark, Apache YARN and Apache Hive work natively on Ozone without any modifications. Therefore, it is essential to have robust security mechanisms to secure your cluster and to protect your data in Ozone.

There are various authentication and authorization mechanisms available in Apache Ozone.

## Authentication in Ozone

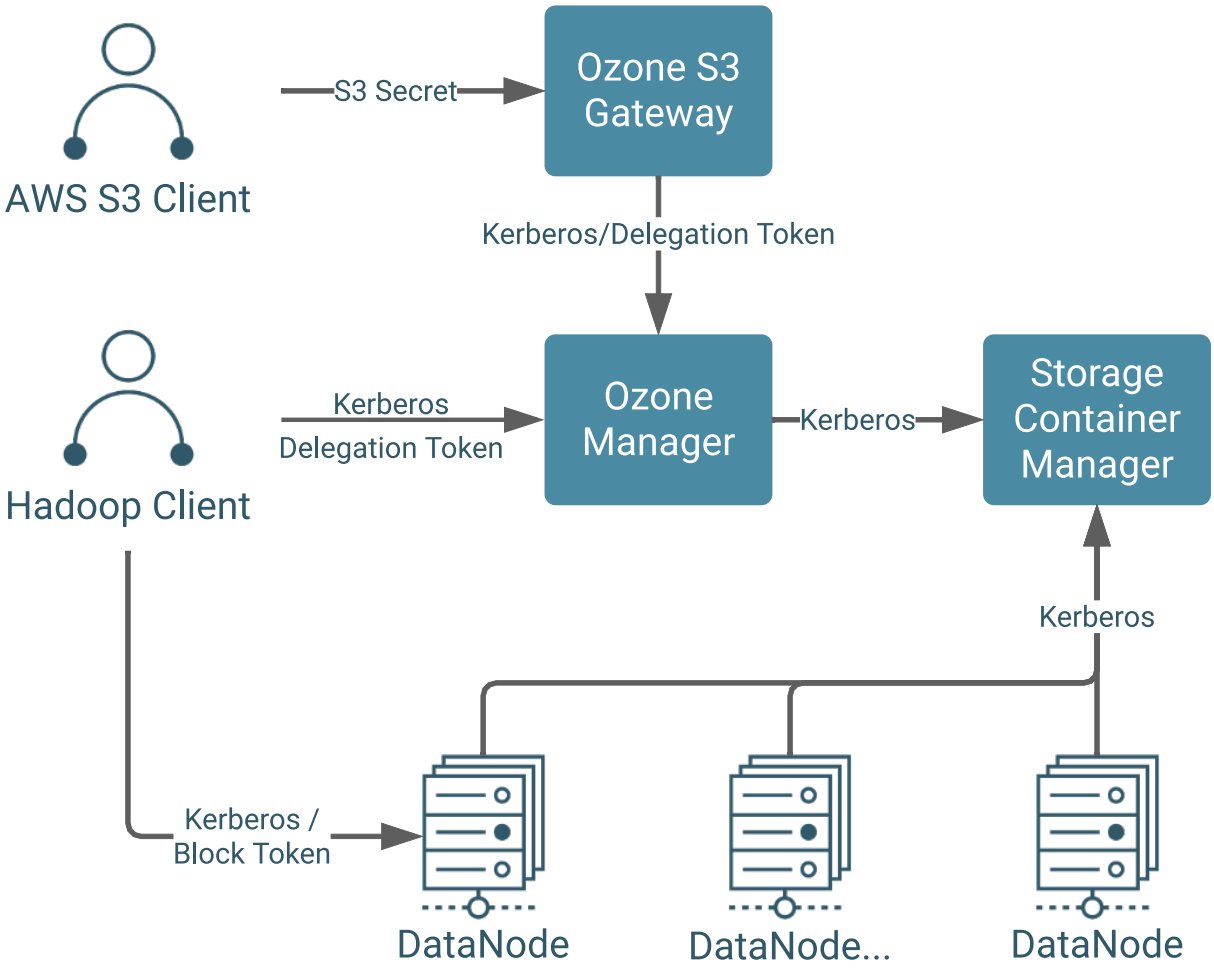
Authentication is the process of recognizing a user's identity for Ozone components. Apache Ozone supports strong authentication using Kerberos and security tokens.

### Kerberos-based authentication

Ozone depends on Kerberos to make the clusters secure. To enable security in an Ozone cluster, you must set the following parameters:

| Property | Value |
|---|---|
| ozone.security.enabled | true |
| hadoop.security.authentication | kerberos |

The following diagram illustrates how service components, such as Ozone Manager (OM), Storage Container Manager (SCM), DataNodes, and Ozone Clients are authenticated with each other through Kerberos:




Each service must be configured with a valid Kerberos Principal Name and a corresponding keytab file, which is used by the service to login at the start of the service in secure mode. Correspondingly, Ozone clients must provide either a valid Kerberos ticket or security tokens to access Ozone services, such as OM for metadata and DataNode for read/write blocks.

### Security tokens

With Ozone serving thousands of requests every second, authenticating through Kerberos each time can be overburdening and is ineffective. Therefore, once authentication is done, Ozone issues delegation and block tokens to users or client applications authenticated with the help of Kerberos, so that they can perform specified operations against the cluster, as if they have valid kerberos tickets.

Ozone security token has a token identifier along with a signed signature from the issuer. The signature of the token can be validated by token validators to verify the identity of the issuer and a valid token holder can use the token to perform operations against the cluster.

Delegation token

Delegation tokens allow a user or client application to impersonate a user's kerberos credentials. The client initially authenticates with OM through Kerberos and obtains a delegation token from OM. Delegation token issued by OM allows token holders to access metadata services provided by OM, such as creating volumes or listing objects in a bucket.

When OM receives a request from a client with a delegation token, it validates the token by checking the signature using its public key. A delegation token can be transferred to other client processes. When a token expires, the original client must request a new delegation token and then pass it to the other client processes.

Delegation token operations, such as get, renew, and cancel can only be performed over a Kerberos authenticated connection.

Block token

Block tokens are similar to delegation tokens because they are issued/signed by OM. Block tokens allow a user or client application to read or write a block in DataNodes. Unlike delegation tokens, which are requested through get, renew, or cancel APIs, block tokens are transparently provided to clients with information about the key or block location. When DataNodes receive read/write requests from clients, block tokens are validated by DataNodes using the certificate or public key of the issuer (OM).

Block tokens cannot be renewed by the client. When a block token expires, the client must retrieve the key/block locations to get new block tokens.

S3 token

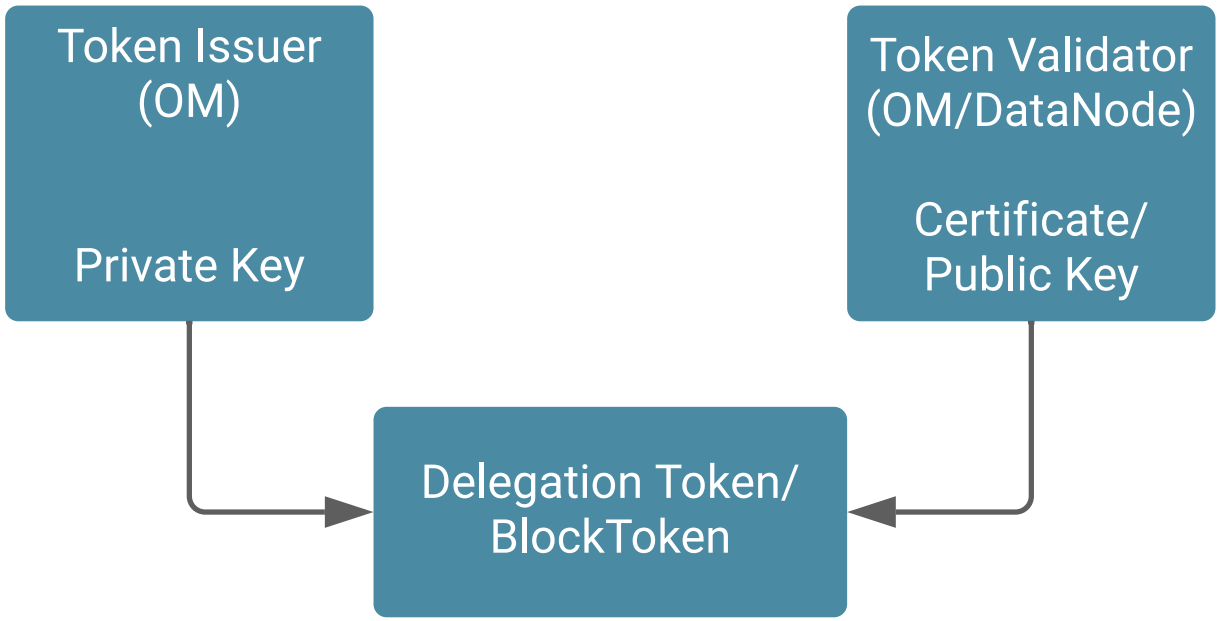
Ozone supports Amazon S3 protocol through Ozone S3 Gateway. In secure mode, OM issues an S3 secret key to Kerberos-authenticated users or client applications that are accessing Ozone using S3 APIs. The access key ID secret access key can be added in the AWS configuration file for Ozone to ensure that a particular user or client application can access Ozone buckets.

S3 tokens are signed by S3 secret keys that are created by the Amazon S3 client. Ozone S3 gateway creates the token for every S3 client request. Users must have an S3 secret key to create the S3 token and similar to the block token, S3 tokens are handled transparently for clients.

### How does Ozone security token work?

Ozone security uses a certificate-based approach to validate security tokens, which make the tokens more secure as shared secret keys are never transported over the network.

The following diagram illustrates the working of an Ozone security token:




In secure mode, SCM bootstraps itself as a certifying authority (CA) and creates a self-signed CA certificate. OM and DataNode must register with SCM CA through a Certificate Signing Request (CSR). SCM validates the identity of OM and DataNode through Kerberos and signs the component's certificate. The signed certificates are then used

by OM and DataNode to prove their identity. This is especially useful for signing and validating delegation or block tokens.

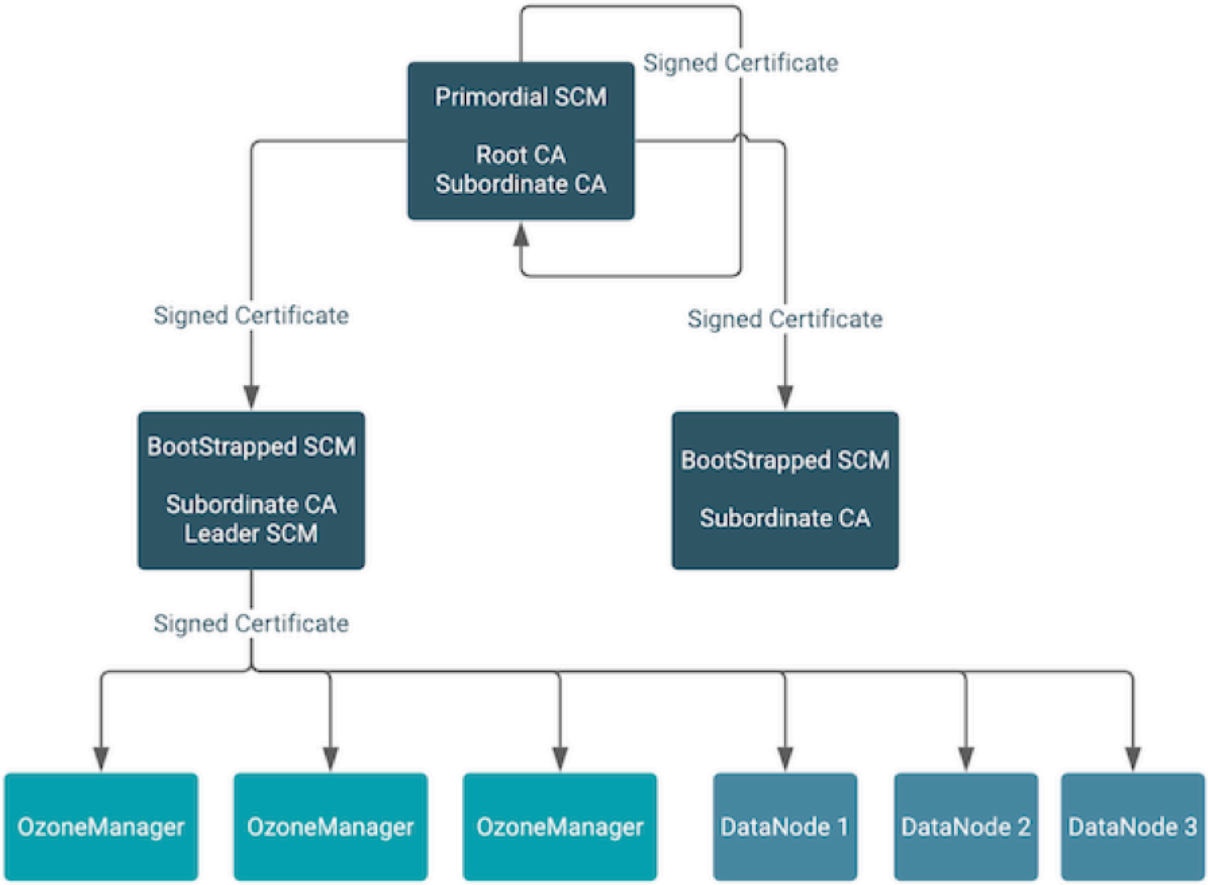
In the case of block tokens, OM (token issuer) signs the token with its private key and DataNodes (token validator) use OM's certificate to validate block tokens because OM and DataNode trust the SCM CA signed certificates.

In the case of delegation tokens when OM (is both a token issuer and token validator) is running in High Availability (HA) mode, there are several OM instances running simultaneously. A delegation token issued and signed by OM instance 1 can be validated by OM instance 2 when the leader OM instance changes. This is possible because both the instances trust the SCM CA signed certificates.

### Certificate-based authentication for SCMs in High Availability

Authentication between Ozone services, such as Storage Container Manager (SCM), Ozone Manager (OM), and DataNodes is achieved using certificates and ensures secured communication in the Ozone cluster. The certificates are issued by SCM to other services during installation.

The following diagram illustrates how SCM issues certificates to other Ozone services:




The primordial SCM in the HA configuration starts a root Certificate Authority (CA) with self-signed certificates. The primordial SCM issues signed certificates to itself and the other bootstrapped SCMs in the HA configuration. The primordial SCM also has a subordinate CA with a signed certificate from the root CA.

When an SCM is bootstrapped, it receives a signed certificate from the primordial SCM and starts a subordinate CA of its own. The subordinate CA of any SCM that becomes the leader in the HA configuration issues signed certificates to Ozone Managers and the DataNodes in the Ozone cluster.

### Related Information

Kerberos principal and keytab properties for Ozone service daemons

Kerberos principal and keytab properties for Ozone DataNodes
Considerations for enabling SCM HA security

## Authorization in Ozone

Authorization is the process of specifying access rights to Ozone resources. Once a user is authenticated, authorization enables you to specify what the user can do within an Ozone cluster. For example, you can allow users to read volumes, buckets, and keys while restricting them from creating volumes.

Ozone supports authorization through the Apache Ranger plugin or through native Access Control Lists (ACLs).

### Using Ozone native ACLs for authorization

Ozone's native ACLs can be used independently or with Ozone ACL plugins, such as Ranger. If Ranger authorization is enabled, the native ACLs are not evaluated.

Ozone native ACLs are a superset of POSIX and S3. The format of an ACL is object:who:rights.

**object**

In an ACL, an object can be the following:

- Volume - An Ozone volume. For example, /volume1.
- Bucket - An Ozone bucket. For example, /volume1/bucket1.
- Key - An object key or an object. For example, /volume1/bucket1/key1.
- Prefix - A path prefix for a specific key. For example, /volume1/bucket1/prefix1/prefix2.

**who**

In an ACL, a who can be the following:

- User - A user in the Kerberos domain. The user can be named or unnamed.
- Group - A group in the Kerberos domain. The group can be named or unnamed.
- World - All authenticated users in the Kerberos domain. This maps to others in the POSIX domain.
- Anonymous - Indicates that the user field should be completely ignored. This value is required for the S3 protocol to indicate anonymous users.

**Note:** Ozone Manager translates an S3 user accessing Ozone by using the AWS v4 signature protocol, to the appropriate Kerberos user.

**rights**

In an ACL, a right can be the following:

- Create - Allows a user to create buckets in a volume and keys in a bucket.

  Only administrators can create volumes.
- List - Allows a user to list buckets and keys. This ACL is attached to the volume and buckets which allow listing of the child objects.

  The user and administrator can list volumes owned by the user.
- Delete - Allows a user to delete a volume, bucket, or key.
- Read - Allows a user to read the metadata of a volume and bucket and data stream and metadata of a key.
- Write - Allows a user to write the metadata of a volume and bucket and allows the user to overwrite an existing ozone key.
- Read_ACL - Allows a user to read the ACL on a specific object.
- Write_ACL - Allows a user to write the ACL on a specific object.

APIs for working with an Ozone ACL

Ozone supports a set of APIs to modify or manipulate the ACLs. The supported APIs are as follows:

- SetAcl - Accepts the user principal, the name and type of an Ozone object, and a list of ACLs.

- GetAcl - Accepts the name and type of an Ozone object and returns the corresponding ACLs.
- AddAcl - Accepts the name and type of the Ozone object, and the ACL to add it to existing ACL entries of the Ozone object.
- RemoveAcl - Accepts the name and type of the Ozone object, and the ACL to remove.

### Using Ranger for authorization

Apache Ranger provides a centralized security framework to manage access control through an user interface that ensures consistent policy administration across Cloudera Data Platform (CDP) components. For Ozone to work with Ranger, you can use Cloudera Manager and enable the Ranger service instance with which you want to use Ozone.

If Ranger authorization is enabled, the native ACLs are ignored and ACLs specified through Ranger are considered.

The following table lists the Ranger permissions corresponding to the various Ozone operations:

| Operation / Permission | Volume Permission | Bucket Permission | Key Permission |
|---|---|---|---|
| Create Volume | CREATE | | |
| List Volume | LIST | | |
| Get Volume Info | READ | | |
| Delete Volume | DELETE | | |
| Create Bucket | READ | CREATE | |
| List Bucket | LIST, READ | | |
| Get Bucket Info | READ | READ | |
| Delete Bucket | READ | DELETE | |
| List Key | READ | LIST, READ | |
| Write Key | READ | READ | CREATE, WRITE |
| Read Key | READ | READ | READ |

### Related Information

Enabling Ranger service instance for Ozone

## Ozone containers

Containers are the fundamental replication unit of Ozone and are managed by the Storage Container Manager (SCM) service. Containers are big binary units (5 Gb by default) that can contain multiple blocks.

Blocks are local information that are not managed by SCM. Therefore, even if billions of small files are created in the system (which means billions of blocks are created), only the status of the containers will be reported by the DataNodes and the containers will be replicated.

Whenever Ozone Manager requests a new block allocation from the SCM, SCM identifies the suitable container and generates a block id which contains a ContainerId and a LocalId. The client connects to the DataNode which stores the container, and the DataNode manages the separated block based on the LocalId.

### Open versus closed containers

| Open | Closed |
|---|---|
| mutable | immutable |
| replicated with RAFT (Ratis) | Replicated with async container copy |

| Open | Closed |
|---|---|
| Write: Allowed only on Raft Leader<br>Read: On all replicas<br>Deletes: Not allowed | Write: No writes are allowed<br>Read: On all replicas<br>Deletes: allowed |

## How Ozone manages read operations

The client requests the block locations corresponding to the key it wants to read. The Ozone Manager (OM) returns the block locations if the client has the required read privileges.

The following steps explain how Ozone manages read operations:

1. The client requests OM for block locations corresponding to the key to read.
2. OM checks the ACLs to confirm whether the client has the required privileges, and returns the block locations and the block token that allows the client to read data from the DataNodes.
3. The client connects to the DataNode associated with the returned Block ID and reads the data blocks.

## How Ozone manages write operations

The client requests blocks from the Ozone Manager (OM) to write a key. OM returns the Block ID and the corresponding DataNodes for the client to write data.

The following steps explain how Ozone manages write operations:

1. The client requests blocks from OM to write a key. The request includes the key, the pipeline type, and the replication count.
2. OM finds the blocks that match the request from SCM and returns them to the client.

   **Note:** If security is enabled on the cluster, OM also provides a block token along with the block location to the client. The client uses the block token to connect to the DataNodes and send the command to write chunks of data.
3. The client connects to the DataNodes associated with the returned block information and writes the data.
4. After writing the data, the client updates the block information on OM by sending a commit request.
5. OM records the associated key information.

**Note:**

- Keys in Ozone are not visible until OM commits the block information associated with the keys. The client is responsible for sending the key-block information to OM after it has written the blocks on the DNs via a commit request.
- If OM fails to commit block information for keys after they have been written, for example, client was unable to send the commit request OM because the write job failed, the keys would not be visible but the data would remain on disk.

  HDDS-4123: OpenKeyCleanup service in Ozone cleans up data on disk whose block information has not been committed on OM. It does by expiring open keys that have not been committed for a configurable time period (7days by default). OM checks for expired keys every 300s by default and issues deletes for these keys. Any data associated with these keys are deleted when the deletes reach DNs. You can contact Cloudera support and request a Hotfix for HDDS-4123 to get this feature.

## How Ozone manages delete operations

Ozone is a consistent object store and hence delete operations in Ozone are synchronous. When the delete request is complete, Ozone Manager deletes the key from the active namespace and marks the file for garbage collection.

Ozone also follows the principle of asynchronous deletion. Garbage collection and the mechanism to recover a storage space is managed asynchronously on most file systems to ensure that the delete operations do not compete with the read and write operations.

The files marked as deleted in Ozone Manager are aggregated by containers and the request is sent to SCM to delete the blocks. SCM then forwards the requests to the DataNode to recover the actual space from the disk. A block deletion happens only over closed containers. For example, if you delete objects in the namespace, the deletion reflects only for the corresponding blocks in the closed containers. For blocks present in containers with state other than CLOSED, the deletion happens only after the container reaches a CLOSED state.

### Related Information

Ozone containers