# Container Management

DAOS containers are the unit of data management for users.

## Container Creation/Destroy

Containers can be created and destroyed through the daos\_cont\_create/destroy() functions exported by the DAOS API. A user tool called daos is also provided to manage containers.

!!! note In DAOS 1.0, in order to use the daos command the following environment variables need to be set (this is no longer needed in later versions of DAOS): \* For Omni-Path: export OFI\_INTERFACE="ib0"; export CRT\_PHY\_ADDR\_STR="ofi+psm2" \* For InfiniBand: export OFI\_INTERFACE="ib0"; export CRT\_PHY\_ADDR\_STR="ofi+verbs;ofi\_rxm"; export OFI\_DOMAIN="mlx5\_0"

To create a container:

$ daos cont create --pool=a171434a-05a5-4671-8fe2-615aa0d05094  
Successfully created container 008123fc-6b6c-4768-a88a-a2a5ef34a1a2

The container type (i.e., POSIX or HDF5) can be passed via the –type option. As shown below, the pool UUID, container UUID, and container attributes can be stored in the extended attributes of a POSIX file or directory for convenience. Then subsequent invocations of the daos tools need to reference the path to the POSIX file or directory.

$ daos cont create --pool=a171434a-05a5-4671-8fe2-615aa0d05094 \  
 --path=/tmp/mycontainer --type=POSIX --oclass=SX \  
 --chunk\_size=4K  
Successfully created container 419b7562-5bb8-453f-bd52-917c8f5d80d1 type POSIX  
  
$ daos container query --path=/tmp/mycontainer  
Pool UUID: a171434a-05a5-4671-8fe2-615aa0d05094  
Container UUID: 419b7562-5bb8-453f-bd52-917c8f5d80d1  
Number of snapshots: 0  
Latest Persistent Snapshot: 0  
DAOS Unified Namespace Attributes on path /tmp/mycontainer:  
Container Type: POSIX  
Object Class: large  
Chunk Size: 4096

## Container Properties

At creation time, a list of container properties can be specified:

|  |  |
| --- | --- |
| **Container Property** | **Description** |
| DAOS\_PROP\_CO\_LABEL | A string that a user can associate with a container. e.g., “Cat Pics” or “ResNet-50 training data” |
| DAOS\_PROP\_CO\_LAYOUT\_TYPE | The container type (POSIX, HDF5, …) |
| DAOS\_PROP\_CO\_LAYOUT\_VER | A version of the layout that can be used by I/O middleware and application to handle interoperability. |
| DAOS\_PROP\_CO\_REDUN\_FAC | The redundancy factor that drives the minimal data protection required for objects stored in the container. e.g., RF1 means no data protection, RF3 only allows 3-way replication or erasure code N+2. |
| DAOS\_PROP\_CO\_REDUN\_LVL | The fault domain level that should be used to place data redundancy information (e.g., storage nodes, racks…). This information will be eventually consumed to determine object placement. |
| DAOS\_PROP\_CO\_CSUM | Checksum off, or algorithm to use. |
| DAOS\_PROP\_CO\_CSUM\_CHUNK\_SIZE | Checksum chunk size. |
| DAOS\_PROP\_CO\_CSUM\_SERVER\_VERIFY | Perform additional checksum verification on server (default: off). |
| DAOS\_PROP\_CO\_SNAPSHOT\_MAX | Impose a limit on number of snapshots to retain (default: 0, no limitation). |
| DAOS\_PROP\_CO\_ACL | Container access control list.| |DAOS\_PROP\_CO\_OWNER| User acting as the owner of the container.| |DAOS\_PROP\_CO\_OWNER\_GROUP` | Group acting as the owner of the container. |

Refer to the Data Integrity and Access Control Lists sections for more details on the checksum and access-related properties.

Refer to the Inline Deduplication section for details about additional properties for that preview feature that are not listed here.

While those properties are currently stored persistently with container metadata, many of them are still under development. The ability to modify some of these properties on an existing container will also be provided in a future release.

### Querying a container’s properties

The user-level administration daos utility may be used to query a container’s properties. Refer to the manual page for full usage details.

# daos cont get-prop --pool=<UUID> --cont=<UUID>  
# -OR- --path interface shown below  
  
$ daos cont get-prop --path=/tmp/mycontainer  
Container properties for 419b7562-5bb8-453f-bd52-917c8f5d80d1 :  
label: container label not set  
layout type: POSIX (1)  
layout version: 1  
checksum type: off  
checksum chunk-size: 32768  
cksum verif. on server: off  
deduplication: off  
dedup threshold: 4096  
redundancy factor: rf0  
redundancy level: rack  
max snapshots: 0  
compression type: off  
encryption type: off  
Allocated OID: 0  
owner: username@  
owner-group: groupname@  
status: HEALTHY  
acl:  
# Entries:  
A::OWNER@:rwdtTaAo  
A:G:GROUP@:rwtT

Additionally, a container’s properties may be retrieved using the libdaos API daos\_cont\_query() function. Refer to the file src/include/daos\_cont.h Doxygen comments and the online documentation available [here](https://daos-stack.github.io/html/).

## Data Integrity

DAOS allows to detect and fix (when data protection is enabled) silent data corruptions. This is done by calculating checksums for both data and metadata in the DAOS library on the client side and storing those checksums persistently in SCM. The checksums will then be validated on access and on update/write as well on the server side if server verify option is enabled.

Corrupted data will never be returned to the application. When a corruption is detected, DAOS will try to read from a different replica, if any. If the original data cannot be recovered, then an error will be reported to the application.

To enable and configure checksums, the following container properties are used during container create.

* DAOS\_PROP\_CO\_CSUM: Type of checksum algorithm to use. Supported values are

DAOS\_PROP\_CO\_CSUM\_OFF, // default  
 DAOS\_PROP\_CO\_CSUM\_CRC16,  
 DAOS\_PROP\_CO\_CSUM\_CRC32,  
 DAOS\_PROP\_CO\_CSUM\_CRC64,  
 DAOS\_PROP\_CO\_CSUM\_SHA1,  
 DAOS\_PROP\_CO\_CSUM\_SHA256,  
 DAOS\_PROP\_CO\_CSUM\_SHA512,  
 DAOS\_PROP\_CO\_CSUM\_ADLER32

* DAOS\_PROP\_CO\_CSUM\_CHUNK\_SIZE: defines the chunk size used for creating checksums of array types. (default is 32K).
* DAOS\_PROP\_CO\_CSUM\_SERVER\_VERIFY: Because of the probable decrease to IOPS, in most cases, it is not desired to verify checksums on an object update on the server side. It is sufficient for the client to verify on a fetch because any data corruption, whether on the object update, storage, or fetch, will be caught. However, there is an advantage to knowing if corruption happens on an update. The update would fail right away, indicating to the client to retry the RPC or report an error to upper levels.

!!! note Note that currently, once a container is created, its checksum configuration cannot be changed.

!!! warning The checksum feature is only supported in DAOS 1.2 and later.

## Inline Deduplication (Preview)

Data deduplication (dedup) is a process that allows to eliminate duplicated data copies in order to decrease capacity requirements. DAOS has some initial support of inline dedup.

When dedup is enabled, each DAOS server maintains a per-pool table indexing extents by their hash (i.e. checksum). Any new I/Os bigger than the deduplication threshold will thus be looked up in this table to find out whether an existing extent with the same signature has already been stored. If an extent is found, then two options are provided:

* Transferring the data from the client to the server and doing a memory compare (i.e. memcmp) of the two extents to verify that they are indeed identical.
* Trusting the hash function and skipping the data transfer. To minimize issue with hash collision, a cryptographic hash function (i.e. SHA256) is used in this case. The benefit of this approarch is that the data to be written does not need to be transferred to the server. Data processing is thus greatly accelerated.

The inline dedup feature can be enabled on a per-container basis. To enable and configure dedup, the following container properties are used:

* DAOS\_PROP\_CO\_DEDUP: Type of dedup mechanism to use. Supported values are

DAOS\_PROP\_CO\_DEDUP\_OFF, // default  
 DAOS\_PROP\_CO\_DEDUP\_MEMCMP, // memory compare  
 DAOS\_PROP\_CO\_CSUM\_HASH // hash-based using SHA256

* DAOS\_PROP\_CO\_DEDUP\_THRESHOLD: defines the minimal I/O size to consider the I/O for dedup (default is 4K).

!!! warning Dedup is a feature preview in 1.2 and has some known limitations. Aggregation of deduplicated extents isn’t supported and the checksum tree isn’t persistent yet. This means that aggregation is disabled for a container with dedplication enabled and duplicated extents won’t be matched after a server restart.

## Compression & Encryption

The DAOS\_PROP\_CO\_COMPRESS and DAOS\_PROP\_CO\_ENCRYPT properties are reserved for configuring respectively online compression and encryption. These features are currently not on the roadmap.

## Snapshot & Rollback

Similar to container create/destroy, a container can be snapshotted through the DAOS API by calling daos\_cont\_create\_snap(). Additional functions are provided to destroy and list container snapshots.

The API also provides interfaces to subscribe to container snapshot events and to rollback the content of a container to a previous snapshot, but those operations are not yet fully implemented.

The daos tool provides container {create/destroy}-snap and list-snaps commands. It provides access to basic snapshot create/destroy according to the constraints of the implementation (e.g., no named snapshots). It provides interfaces (commands and arguments) for the broader set of snapshot features, for when they are implemented later (e.g., specified epoch/epoch range, rollback).

The DAOS\_PROP\_CO\_SNAPSHOT\_MAX property is used to limit the maximum number of snapshots to retain. When a new snapshot is taken, and the threshold is reached, the oldest snapshot will be automatically deleted.

Rolling back the content of a container to a snapshot is planned for future DAOS versions.

## User Attributes

Similar to POSIX extended attributes, users can attach some metadata to each container through the daos\_cont\_{list/get/set}\_attr() API, and through the utility commands: daos cont {get/set}-attr and list-attrs.

## Access Control Lists

Client user and group access for containers is controlled by [Access Control Lists (ACLs)](https://daos-stack.github.io/overview/security/#access-control-lists).

Access-controlled container accesses include:

* Opening the container for access.
* Reading and writing data in the container.
  + Reading and writing objects.
  + Getting, setting, and listing user attributes.
  + Getting, setting, and listing snapshots.
* Deleting the container (if the pool does not grant the user permission).
* Getting and setting container properties.
* Getting and modifying the container ACL.
* Modifying the container’s owner.

This is reflected in the set of supported [container permissions](https://daos-stack.github.io/overview/security/#permissions).

### Pool vs. Container Permissions

In general, pool permissions are separate from container permissions, and access to one does not guarantee access to the other. However, a user must have permission to connect to a container’s pool before they can access the container in any way, regardless of their permissions on that container. Once the user has connected to a pool, container access decisions are based on the individual container ACL. A user need not have read/write access to a pool in order to open a container with read/write access, for example.

There is one situation in which the pool can grant a container-level permission: Container deletion. If a user has Delete permission on a pool, this grants them the ability to delete *any* container in the pool, regardless of their permissions on that container.

If the user does not have Delete permission on the pool, they will only be able to delete containers for which they have been explicitly granted Delete permission in the container’s ACL.

!!! note In DAOS version 1.0, permissions are set on the *pool* level and all containers in the pool inherit the permissions of the pool. Starting with DAOS version 1.2, pool and container permissions are controlled individually.

### Creating Containers with Custom ACL

To create a container with a custom ACL:

$ daos cont create --pool=<UUID> --acl-file=<path>

The ACL file format is detailed in the [ACL section](https://daos-stack.github.io/overview/security/#acl-file).

### Displaying a Container’s ACL

To view a container’s ACL:

$ daos cont get-acl --pool=<UUID> --cont=<UUID>

The output is in the same string format used in the ACL file during creation, with one ACE per line.

### Modifying a Container’s ACL

For all of these commands using an ACL file, the ACL file must be in the format noted above for container creation.

#### Overwriting the ACL

To replace a container’s ACL with a new ACL:

$ daos cont overwrite-acl --pool=<UUID> --cont=<UUID> \  
 --acl-file=<path>

#### Adding and Updating ACEs

To add or update multiple entries in an existing container ACL:

$ daos cont update-acl --pool=<UUID> --cont=<UUID> \  
 --acl-file=<path>

To add or update a single entry in an existing container ACL:

$ daos cont update-acl --pool=<UUID> --cont=<UUID> --entry <ACE>

If there is no existing entry for the principal in the ACL, the new entry is added to the ACL. If there is already an entry for the principal, that entry is replaced with the new one.

#### Removing an ACE

To delete an entry for a given principal in an existing container ACL:

$ daos cont delete-acl --pool=<UUID> --cont=<UUID> \  
 --principal=<principal>

The principal corresponds to the principal portion of an ACE that was set during container creation or a previous container ACL operation. For the delete operation, the principal argument must be formatted as follows:

* Named user: u:username@
* Named group: g:groupname@
* Special principals:
  + OWNER@
  + GROUP@
  + EVERYONE@

The entry for that principal will be completely removed. This does not always mean that the principal will have no access. Rather, their access to the container will be decided based on the remaining ACL rules.

### Ownership

The ownership of the container corresponds to the special principals OWNER@ and GROUP@ in the ACL. These values are a part of the container properties. They may be set on container creation and changed later.

#### Privileges

The owner-user (OWNER@) has implicit privileges on their container. The owner-user can always open the container, and has set-ACL (A) and get-ACL (a) permissions. These permissions are included alongside any permissions that the user was explicitly granted by entries in the ACL.

Because the owner’s special permissions are implicit, they apply to access control decisions even if they do not appear in the OWNER@ entry, and even if the OWNER@ entry is deleted.

The owner-group (GROUP@) has no special permissions outside what they are granted by the ACL.

#### Creating Containers with Specific Ownership

The default owner user and group are the effective user and group of the user creating the container. However, a specific user and/or group may be specified at container creation time.

$ daos cont create --pool=<UUID> --user=<owner-user> \  
 --group=<owner-group>

The user and group names are case sensitive and must be formatted as [DAOS ACL user/group principals](https://daos-stack.github.io/overview/security/#principal).

#### Changing Ownership

To change the owner user:

$ daos cont set-owner --pool=<UUID> --cont=<UUID> \  
 --user=<owner-user>

To change the owner group:

$ daos cont set-owner --pool=<UUID> --cont=<UUID> \  
 --group=<owner-group>

The user and group names are case sensitive and must be formatted as [DAOS ACL user/group principals](https://daos-stack.github.io/overview/security/#principal).