# Application Interface and Tiering

## DAOS 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” to manage containers is under development and will be available and documented here for DAOS v1.0.

### Container Properties

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

* DAOS\_PROP\_CO\_LABEL is a string that a user can associate with a container. e.g. “Cat Pics” or “ResNet-50 training data”
* DAOS\_PROP\_CO\_LAYOUT\_TYPE is the container type (POSIX, MPI-IO, HDF5, …)
* DAOS\_PROP\_CO\_LAYOUT\_VER is a version of the layout that can be used by I/O middleware and application to handle interoperability.
* DAOS\_PROP\_CO\_CSUM defines whether checksums are enabled or disabled and the checksum type used.
* DAOS\_PROP\_CO\_REDUN\_FAC is 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 is 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\_SNAPSHOT\_MAX is the maximum number of snapshot to retain. When a new snapshot is taken and the threshold is reached, the oldest snapshot will be automatically deleted.
* DAOS\_PROP\_CO\_ACL is the list of ACL for the container.
* DAOS\_PROP\_CO\_COMPRESS and DAOS\_PROP\_CO\_ENCRYPT are reserved to configure respectively compression and encryption. Those features are currently not on the roadmap.

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 eventually provided.

### Container Snapshot

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 the ability 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.

This section will be updated once the “daos” tool is available.

### Container User Attributes

Similar to POSIX extended attributes, users can attach some metadata to each container through the daos\_cont\_{list/get/set}\_attr() API.

### Container ACLs

Support for per-container ACLs is scheduled for DAOS v1.2. Similar to pool ACLs, container ACLs will implement a subset of the NFSv4 ACL standard. This feature will be documented here once available.

## Native Programming Interface

### Building against the DAOS library

To build an applications or I/O middleware against the native DAOS API, include the daos.h header file in your program and link with -Ldaos. Examples are available under src/tests.

### DAOS API Reference

libdaos is written in C and uses Doxygen comments that are added to C header files.

[TODO] Generate Doxygen document and add a link here.

### Bindings to Different Languages

API bindings to both Python[[1]](#footnote-2) and Go[[2]](#footnote-3) languages are available.

## POSIX Filesystem

A regular POSIX namespace can be encapsulated into a DAOS container. This capability is provided by the libdfs library that implements the file and directory abstractions over the native libdaos library. The POSIX emulation can be exposed to applications or I/O frameworks either directly (e.g. for frameworks Spark or TensorFlow or benchmark like IOR or mdtest that support different storage backend plugin) or transparently via a FUSE daemon combined optionally with an interception library to address some of the FUSE performance bottleneck by delivering full OS bypass for POSIX read/write operations.

### libdfs

DFS stands for DAOS File System and is a library that allows a DAOS container to be accessed as a hierarchical POSIX namespace. It supports files, directories and symbolic links, but not hard links. Access permissions are inherited from the parent pool and not implemented on a per-file or per-directory basis. setuid() and setgid() programs as well as supplementary groups are currently not supported.

While libdfs can be tested from a single instance (i.e. single process or client node if used through dfuse), special care is required when the same POSIX container is mounted concurrently by multiple processes. Concurrent DFS mounts are not recommended. Support for concurrency control is under development and will be documented here once ready.

### dfuse

A simple high level fuse plugin (dfuse) is implemented to test the DFS API and functionality with existing POSIX tests and benchmarks (IOR, mdtest, etc.). The DFS fuse exposes one mountpoint as a single DFS namespace with a single pool and container. To test dfuse, the following steps need to be done:

1. Launch DAOS server(s) and create a pool as specified in the previous section. This will return a pool uuid "puuid" and service rank list "svcl"
2. Create an empty directory for the fuse mountpoint. For example let's use /tmp/dfs\_test
3. Mount dfuse with the following command:   
   orterun -np 1 --ompi-server file:~/uri.txt dfuse /tmp/dfs\_test -s -f -p puuid -l svcl   
   -p specifies the pool uuid and -l specifies the service rank list (from dmg).
4. Other arguments to dfuse: -r: option to destroy the container associated with the namespace when you umount. -d: prints debug messages at the fuse mount terminal
5. Now /tmp/dfs\_test can be used as a POSIX file system (i.e., can run things like IOR/mdtest on it)
6. When done, unmount the file system: fusermount -u /tmp/dfs\_test

Work is underway to rewrite the dfuse daemon against the low-level FUSE API.

### libioil

An interception library called libioil is under development. This library will work in conjunction with dfuse and allow to intercept POSIX read(2) and write(2) and issue the I/O operations directly from the application context through libdaos and without any application changes.

Support for libioil is currently planned for DAOS v1.2.

## Unified Namespace

The DAOS tier can be tightly integrated with the Lustre parallel filesystem in which DAOS containers will be represented through the Lustre namespace. This capability is under development and is scheduled for DAOS v1.2.

## HPC I/O Middleware Support

Several HPC I/O middleware libraries have been ported to the native API.

### MPI-IO

DAOS has its own MPI-IO ROM ADIO driver located in a MPICH fork on GitHub:

<https://github.com/daos-stack/mpich>

This driver has been submitted upstream for integration.

To build the MPI-IO driver:

* export MPI\_LIB=""
* download the mpich repo from above and switch to daos\_adio branch
* ./autogen.sh
* mkdir build; cd build
* ../configure --prefix=dir --enable-fortran=all --enable-romio --enable-cxx --enable-g=all --enable-debuginfo --with-file-system=ufs+daos --with-daos=dir --with-cart=dir
* make -j8; make install

Switch PATH and LD\_LIBRARY\_PATH where you want to build your client apps or libs that use MPI to the above installed MPICH. Note that the DAOS server will still need to be launched with OMPI's orterun. This is a unique situation where the server uses OMPI and the clients will be launched with MPICH.

Build any client (HDF5, ior, mpi test suites) normally with the mpicc and mpich library installed above (see child pages).

To run an example:

1. Launch DAOS server(s) and create a pool as specified in the previous section. This will return a pool uuid "puuid" and service rank list "svcl"
2. At the client side, the following environment variables need to be set:  
   export PATH=/path/to/mpich/install/bin:$PATH  
   export LD\_LIBRARY\_PATH=/path/to/mpich/install/lib:$LD\_LIBRARY\_PATH  
   export MPI\_LIB=""  
   export CRT\_ATTACH\_INFO\_PATH=/path/ (whatever was passed to daos\_server -a)  
   export DAOS\_SINGLETON\_CLI=1
3. export DAOS\_POOL=puuid; export DAOS\_SVCL=svcl  
   This is just temporary till we have a better way of passing pool connect info to MPI-IO and other middleware over DAOS.
4. Run the client application or test.

Limitations to the current implementation include:

* Incorrect MPI\_File\_set\_size and MPI\_File\_get\_size - This will be fixed in the future when DAOS correctly supports records enumeration after punch or key query for max/min key and recx.
* Reading Holes does not return 0, but leaves the buffer untouched   
  (Not sure how to fix this - might need to wait for DAOS implementation of iov\_map\_t to determine holes vs written bytes in the Array extent).
* No support for MPI file atomicity, preallocate, shared file pointers.   
  (Those features were agreed upon as OK not to support.)

### HDF5

A prototype version of a HDF5 DAOS connector is available. Please refer to the DAOS VOL connector user guide[[3]](#footnote-4) for instructions on how to build and use it.

## Spark Support

Spark integration with libdfs is under development and is scheduled for DAOS v1.0 or v1.2.

## Data Migration

### Migration to/from a POSIX filesystem

A dataset mover tool is under consideration to move a snapshot of a POSIX, MPI-IO or HDF5 container to a POSIX filesystem and vice versa. The copy will be performed at the POSIX or HDF5 level. The resulting HDF5 file over the POSIX filesystem will be accessible through the native HDF5 connector with the POSIX VFD.

The first version of the mover tool is currently scheduled for DAOS v1.4.

### Container Parking

The mover tool will also eventually support the ability to serialize and deserialize a DAOS container to a set of POSIX files that can be stored or “parked” in an external POSIX filesystem. This transformation is agnostic to the data model and container type and will retain all DAOS internal metadata.

1. https://github.com/daos-stack/daos/blob/master/src/utils/py/README.md [↑](#footnote-ref-2)
2. https://godoc.org/github.com/daos-stack/go-daos/pkg/daos [↑](#footnote-ref-3)
3. https://bitbucket.hdfgroup.org/projects/HDF5VOL/repos/daos-vol/browse/README.md [↑](#footnote-ref-4)