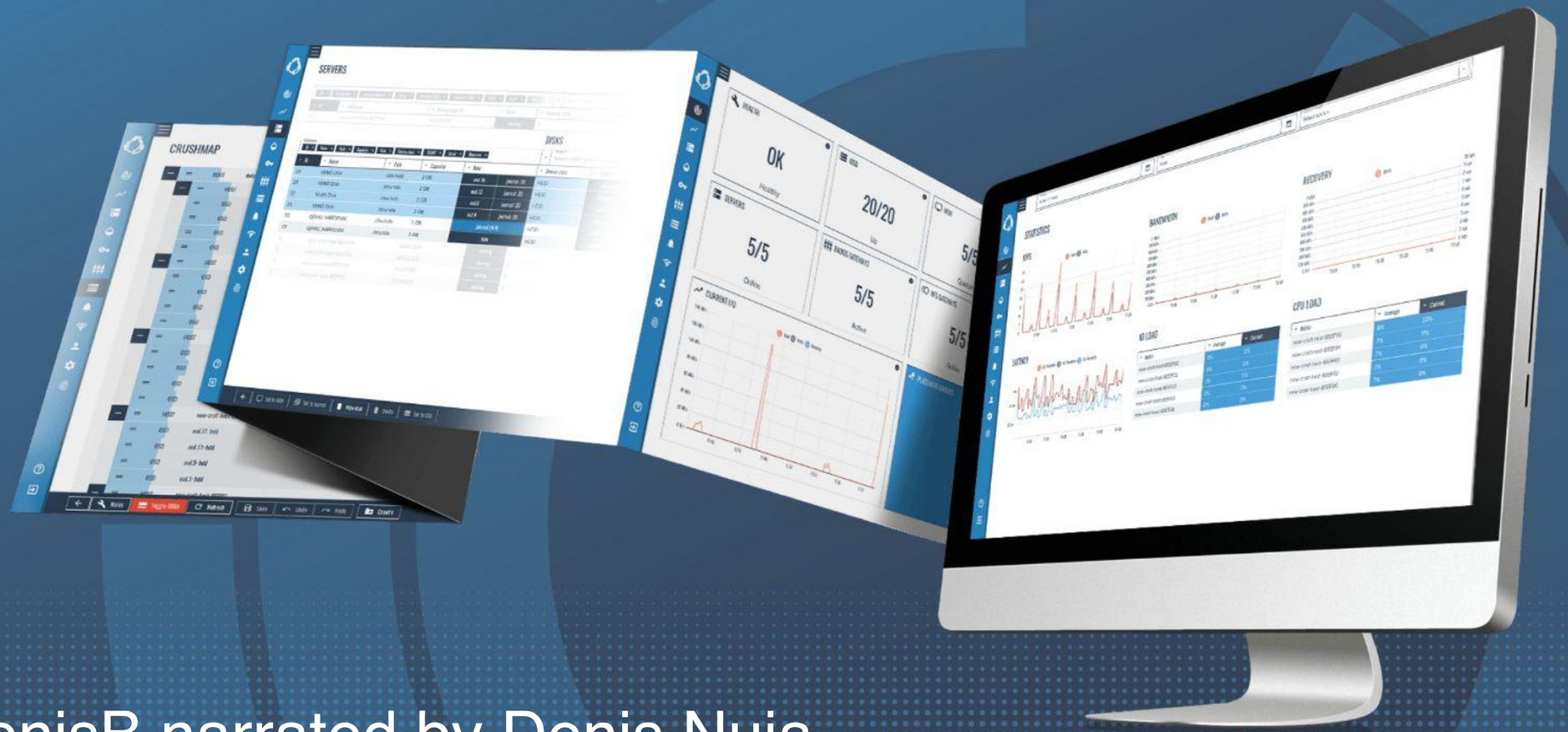
Implementing SPDK DAOS bdev module



A journey by DenisB narrated by Denis Nuja

Why

At the moment DAOS provides just a few connectivity options:

- Direct Key/Value and Array APIs - highly app specific.
- Direct DFS API (partially POSIX complaint) - also app level.
- DFuse (mounting like a volume) - system wide, but with it’s own limitations.

What if my application works only with block devices ?

NVMeOF client is already built in into many platforms.

Why

Implementing SPDK bdev module allows to benefit from:

- NVMe-oF target
- iSCSI target
- vhost server
- NVMe-oF multipath
- BDEVs stacking (raid, lvols …)
- Dev and perf tooling
- Existing unit tests (who likes to write new ones !)
- Probably almost every upcoming features and improvements

How

SPDK has great documentation and a wide range of existing bdev implementations.

The idea was to grab **malloc** or **aio** bdev sources and replace ***open/read/write*** functions with DAOS API and see what's going to happen.

How

SPDK has great documentation and a wide range of existing bdev implementations.

The idea was to grab **malloc** or **aio** bdev sources and replace ***open/read/write*** functions with DAOS API and see what's going to happen.

… it's actually worked !

How

SPDK bdev's bare minimum operation list:

```
switch (bdev_io->type) {
    // Block IO
case SPDK_BDEV_IO_TYPE_READ:
case SPDK_BDEV_IO_TYPE_WRITE:
    // ...
case SPDK_BDEV_IO_TYPE_RESET:
    // Wait until in-flight requests are done
case SPDK_BDEV_IO_TYPE_FLUSH:
    // NOOP (DAOS persistent on write)
case SPDK_BDEV_IO_TYPE_UNMAP:
    // DAOS supports punching holes
}
```

How

SPDK_BDEV_IO_TYPE_RESET - there is no reset functionality in DAOS client (and probably could not be).

Current implementation just waits until all in-flight IO requests are completed.

Rest of the command handlers fit naturally in SPDK bdev model.

How

DAOS Array API vs DFS API, which one to use ?:

```
int daos_array_open(...)  vs int dfs_open(...);
int daos_array_read(...)  vs int dfs_read(...);
int daos_array_write(...) vs int dfs_write(...);
int daos_array_punch(...) vs int dfs_punch(...);
```


croit

How

DAOS Array API vs DFS API, which one to use ?

- The APIs are pretty similar
- Both support batch read and write
- Both provides async mode with DAOS event queue
- DFS API uses Array API underneath, should not be any performance penalties
- DFS API has a slight operational advantage - one could mount DAOS container with **dfuse** and copy the backing file to some other storage.

How

DAOS Array API vs DFS API, which one to use ?

- The APIs are pretty similar
- Both support batch read and write
- Both provides async mode with DAOS event queue
- DFS API uses Array API underneath, should not be any performance penalties
- DFS API has a slight operational advantage - one could mount DAOS container with **dfuse** and copy the backing file to some other storage.
- **The winner is DFS API as it looks familiar to POSIX API.**

croit

How

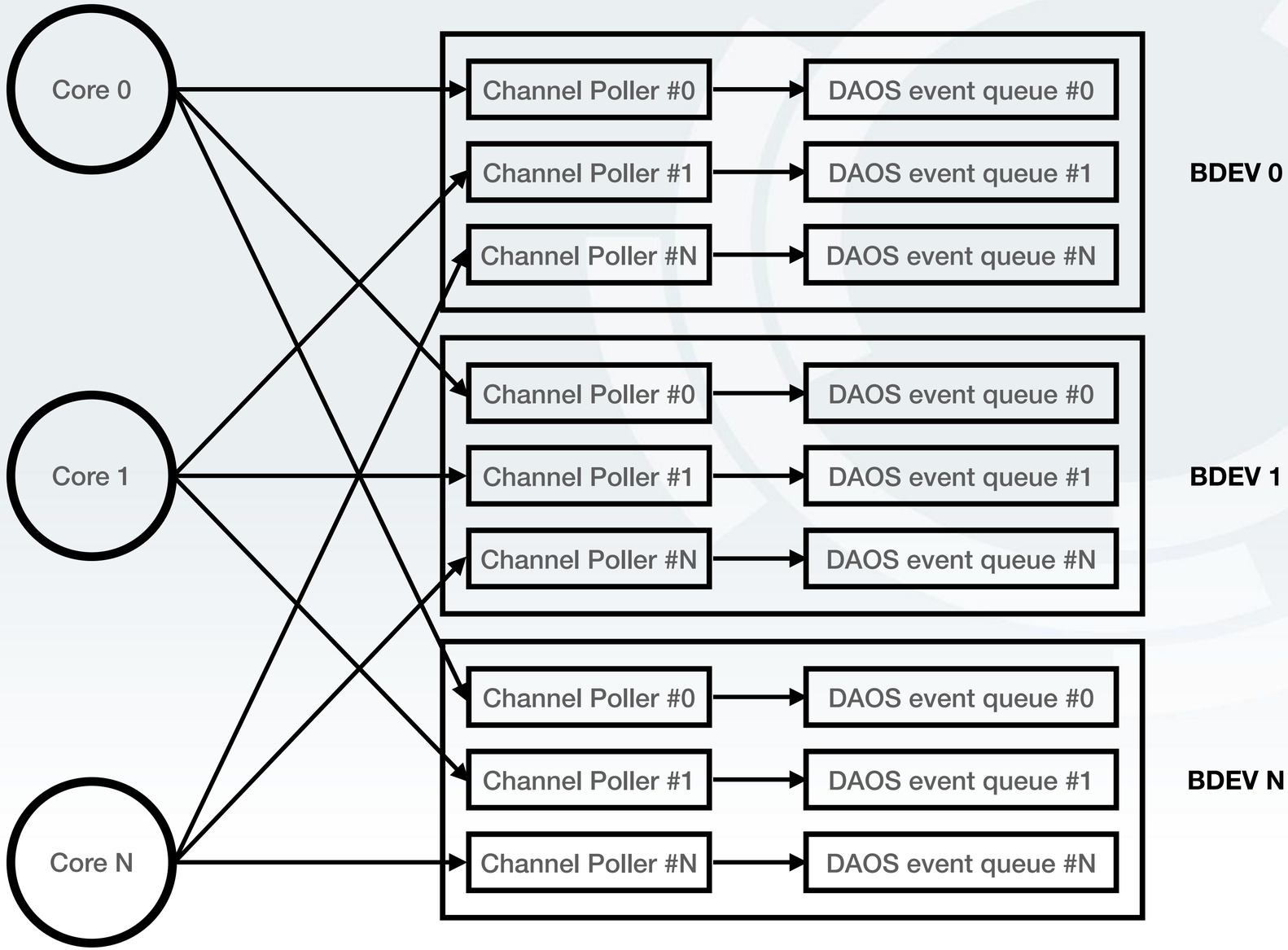
Initial version event handling approach

Every device’s channel has its own poller with DAOS event queue attached.

This approach allows almost linear performance scaling (up to certain point) with increasing number of cores allocated to the target.

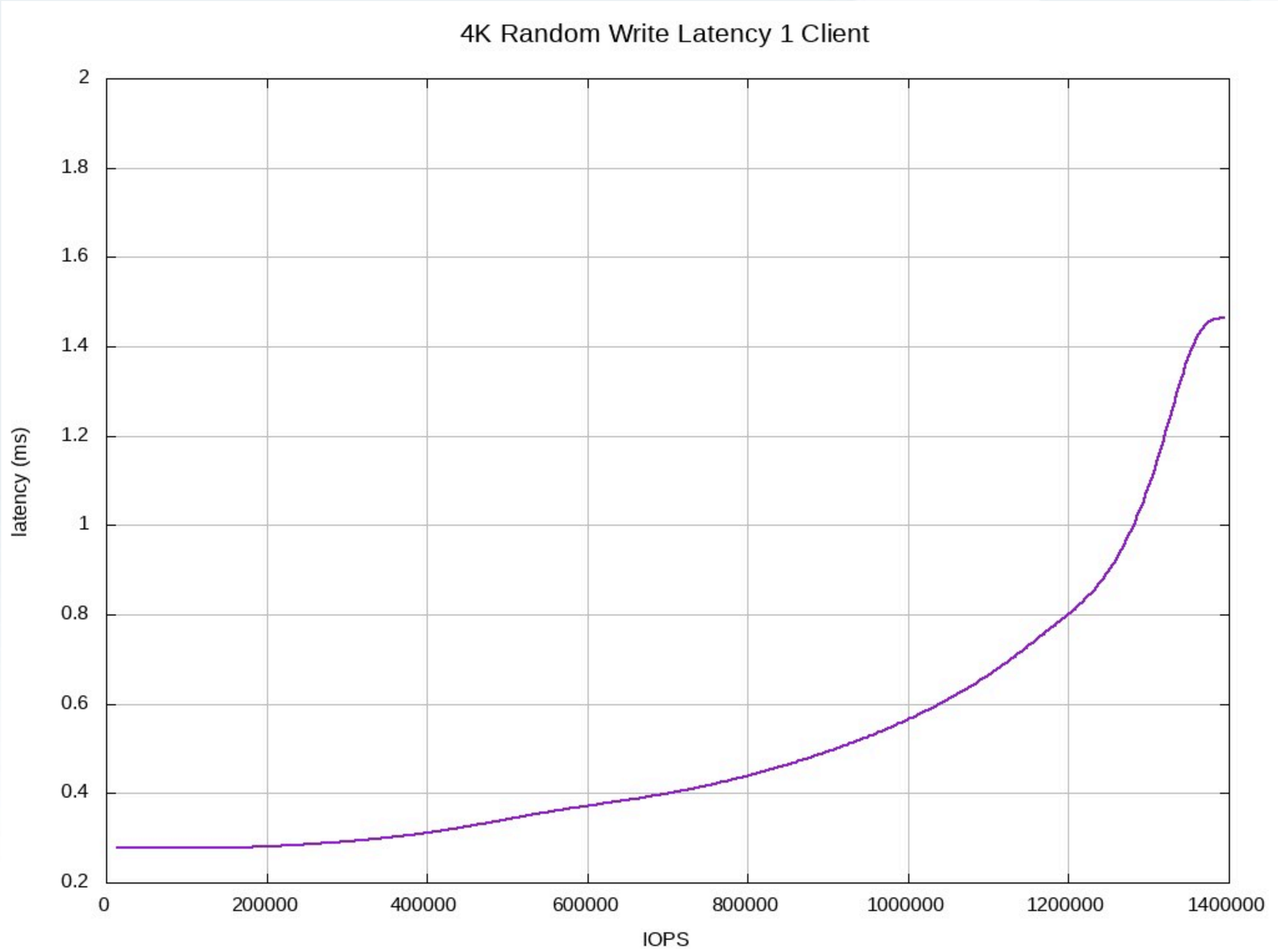
How

It looks like this:



Latency vs IOPS graph
4k random write
1 client (vdbench)

nonRDMA Ethernet TCP
on DAOS cluster and client



Pitfalls along the way

Prior DAOS v2.2 there wasn’t per thread pool/cont cache which significantly affected multithreaded performance (but not multiprocess one).

With v2.2 release cache and more granular locking were introduced resolving the issue and increasing the multithreaded performance.

More details: https://daosio.atlassian.net/browse/DAOS-11230

Pitfalls along the way

MxN problem: M devices create N channels/event queue each. As turned out, the number of DAOS event queue:

- fabric provider specific and limited

- does not scale after certain limit

For instance, for TCP provider the limit is 256 queues, which means 8 bdev with 32 cores enabled.

Pitfalls along the way

Over provisioning: there’s no way to reserve space inside the DAOS container. This might create an issue when someone creates more bdevs than there are free space and the error would happened on the writes.

Next iteration

Fixed pool of DAOS event queues and possible the fixed number of poller across all DAOS bdevs to distribute load evenly.

Better recovery procedure on errors: in the current version the module treats most of the errors as recoverable, which might not be always the case and proper self shutdown is preferable.

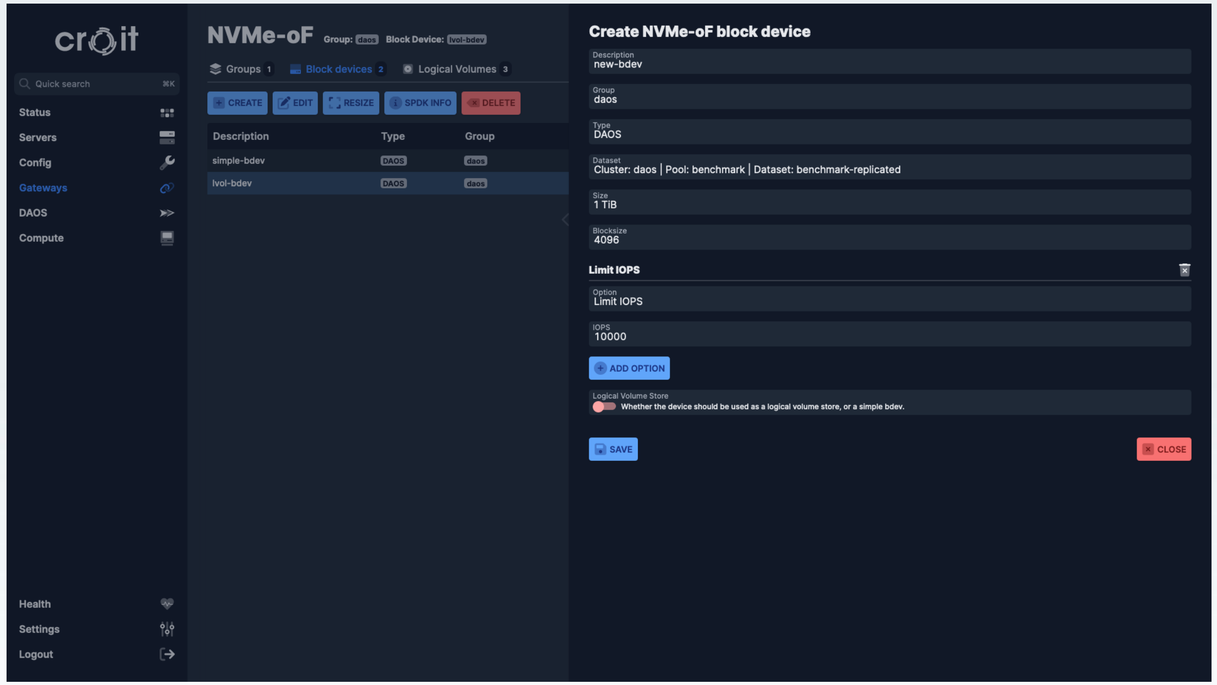
Various QoL improvements.

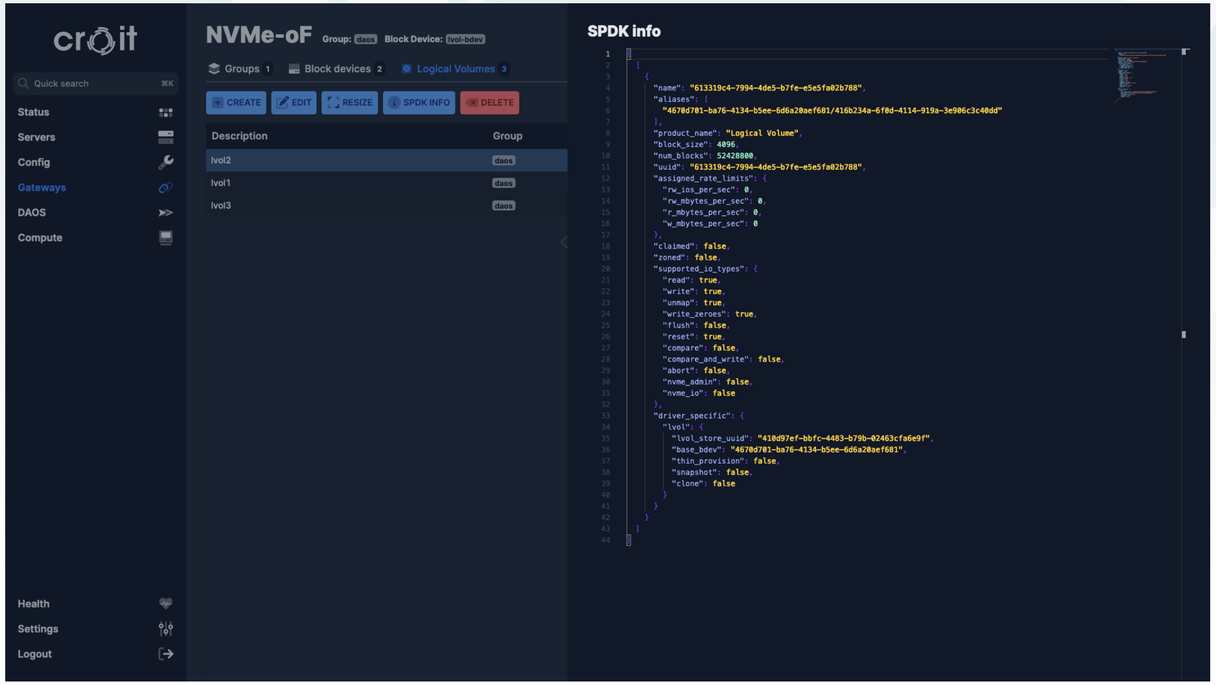
Acknowledgement

SPDK community in Slack, especially James Harris for patiently answering all silly questions, suggestion and review.

DAOS team, in particularly, Johann Lombardi and Mohamad Chaarawi for helping with never ending stream of questions!

How are actually use it in the Croit Platform





Thank you for your attention!

www.croit.io
contact@croit.io