This document summarizes container discussions in the Mero architecture team. The goal is to provide a starting point for design level discussions and resource estimation. Note that this documents is not a substitute for a list of requirements or use-cases. Intended audience includes T1 team and Mero architects.

Overview.

Container is a low level abstraction that insulates higher Mero layers of the knowledge of storage device addresses (block numbers). The bulk of data and meta-data is stored in containers.

Items.

- A container is a storage for data or meta-data. There are several types of container: data container, meta-data container, possibly others.

- A container provides a very simple interface: it has an internal namespace consisting of *keys* and provides methods to fetch and update records stored at a given key. Nomenclature of keys and constraints on record structure depend on container type. For data container, keys are logical page offsets and records are simply pagesful of data without any internal structure. For meta-data container keys are opaque identifiers of meta-data records.

- A container stores its contents in a *backing store*---a storage device. A container allocates space from its backing store and returns no longer used space back as necessary.

- Local RAID is implemented by using a collection of containers to stripe data across. Note that local RAID is *not* implemented in the containers layer to avoid cyclic dependency between layers: the data structures (specifically layouts) that local RAID implementation is required to share with SNS are interpreted by Mero back-end that itself uses containers to store its data.

- Snapshots. Containers are used for data and meta-data snapshotting. When a local snapshot is made as a part of object or system level snapshotting, a container involved into the snapshot is COW-ed, so that all updates are re-directed to a new container.

- After a container is COW-ed no update should ever touch the blocks of the read-only master container. This is a necessary prerequisite of a scalable fsck implementation, that will achieve reasonable confidence in system consistency by background scan and check of periodically taken global snapshots.

- Migration. A container (either read-only or read-write) can migrate to another node. Typical scenarios of such migration are bulk re-integration of updates from a proxy-server to a set of master servers and moving snapshots.

- To make migration efficient, a container must be able to do a fast IO of its contents to another node. Specifically, it should be possible to send container contents over network without iterating through individual container records. This condition also implies that a container allocates space from its backing store in a relatively large contiguous chunks.

- Self-identification. A container has an identity that is stored in the container and migrated together with the container. Higher layers address container records by `(container-id, key)` pairs and such addresses remain valid across container migrations, including "migration" where a hard drive is pulled out of one server and inserted into another. In the latter case the system must be able to determine what containers were moved between nodes and update configuration respectively, also it should able determine whether any storage or layout invariants were violated, *e.g.*, whether multiple units of the same parity group are now located on the same server.

- Layouts. Higher layers address and locate data and meta-data through container identifiers and container keys. On the other hand, a layout produces a location of a given piece of data or meta-data. Together this means that lowest level layouts produce locations in the form of `(container-id, key)` pairs.

- A container can be merged into another container. Inclusion of one container into another can be done administrative reasons, to efficiently migrate a large number of smaller containers or for some other purpose. On inclusion, a container retains its identity (does it? Losing identity would require updating (and, hence, tracking) all references).

- Fids. A fid (file identifiers) is an immutable globally and temporally unique file identifier. As file meta-data record ("inode") is stored in some container, it's logical to use `(container-id, key)` address of this record as fid. Note that this is formally quite similar to the structure of Lustre fid, composed of a sequence identifier and offset within the sequence.

- CLDB. A method is required to resolve container addresses to node identifiers. This method should work while containers migrate between nodes and merge with each other. Container Location Data-Base (CLDB) is a distributed data-base tracking containers in the cluster. This data-base is updated transactionally on container migrations and merges (and splits? There must be splits for symmetry.). Note that CLDB supersedes Fid Location Data-Base (FLDB), see above on fids.

- Data integrity. A container is a possible place to deal with data integrity issues. Alternatively this can be relegated to lower levels (self-checking device pairs) or higher levels (DMU-like file system with check-sums in meta-data).