



Whamcloud



Lustre Multi-Tenancy

LUG 2025

sbuisson@whamcloud.com



Lustre Multi-Tenancy

- ▶ The base feature behind multi-tenancy: Nodemap
- ▶ How to implement isolation/multi-tenancy with Nodemap?
 - RBAC
 - Idmap ranges
 - Mapping offsets
- ▶ Upcoming Nodemap features
 - Dynamic nodemaps
 - Multiple filesets

What is Nodemap?

- ▶ Nodemap is a long-established feature...
 - Supported from Lustre 2.9 (2015)
 - But introduced in Lustre 2.7 as a technology preview!
- ▶ ... initially developed for ID mapping.
 - Multiple sites with conflicting user and group ids can operate on a single Lustre file system without collisions in UID or GID space.

How does Nodemap work?

- ▶ Lustre distinguishes clients based on their Network Identifier (NID).
- ▶ Then file system access is filtered through the Nodemap identity mapping policy engine.
- ▶ When a connection is made from a NID:
 - Lustre decides if that NID is part of a *nodemap*: a policy group made of NID ranges.
 - NID mapping is done only once at filesystem connection time
 - A collection of identity maps or idmaps is kept for each policy group.
 - idmaps translate client UIDs, GIDs, and PROJIDs into canonical filesystem IDs.
 - Each policy group also has properties, governing access conditions.

Basic Nodemap commands

```
(1) mgs# lctl nodemap_add Tenant1  
(2) mgs# lctl nodemap_add_range --name Tenant1 --range 192.168.1.[100-200]@tcp  
(3) mgs# lctl nodemap_add_range --name Tenant1 --range 192.168.2.[0-50]@tcp
```

Nodemap does not allow overlapping ranges, **globally** for all nodemaps.

```
(4) mgs# lctl nodemap_add_idmap --name Tenant1 --idtype uid --idmap 530:11000  
(5) mgs# lctl nodemap_add_idmap --name Tenant1 --idtype gid --idmap 530:11000  
(6) mgs# lctl nodemap_add_idmap --name Tenant1 --idtype projid --idmap 101:1001
```

projid mapping added in Lustre 2.15.

How does Nodemap work?

▶ Nodemap properties govern access conditions

- Privileged access:
 - admin: root remains root
 - trusted: no id mappings, client ids are kept as-is
 - ⇒ admin+trusted: for servers, administrative clients
- Unmapped ids:
 - squash_uid, squash_gid, squash_projid:
 - ids will be squashed to **if unmapped**
 - deny_unknown: denies all access to unmapped ids
- map_mode: which kind of ids are mapped: UID, GID, PROJID

Modify nodemap properties

```
(1) mgs# lctl nodemap_add TrustedSystems
(2) mgs# lctl nodemap_add_range --name TrustedSystems --range 192.168.0.[0-255]@tcp
(3) mgs# lctl nodemap_modify --name TrustedSystems --property admin --value 1
(4) mgs# lctl nodemap_modify --name TrustedSystems --property trusted --value 1
```

TrustedSystems nodemap required for proper Lustre operations.

Must include all Lustre server nodes.

```
(5) mgs# lctl nodemap_modify --name Tenant1 --property squash_uid --value 65534
```

OR

```
(6) mgs# lctl nodemap_modify --name Tenant1 --property deny_unknown --value 1
```

The default nodemap

▶ A default nodemap is needed

- Fallback nodemap, setting the behavior for Lustre clients that do not match any other nodemap
 - Cannot be removed
- No id mapping can be defined on the default nodemap!

Enabling Nodemap

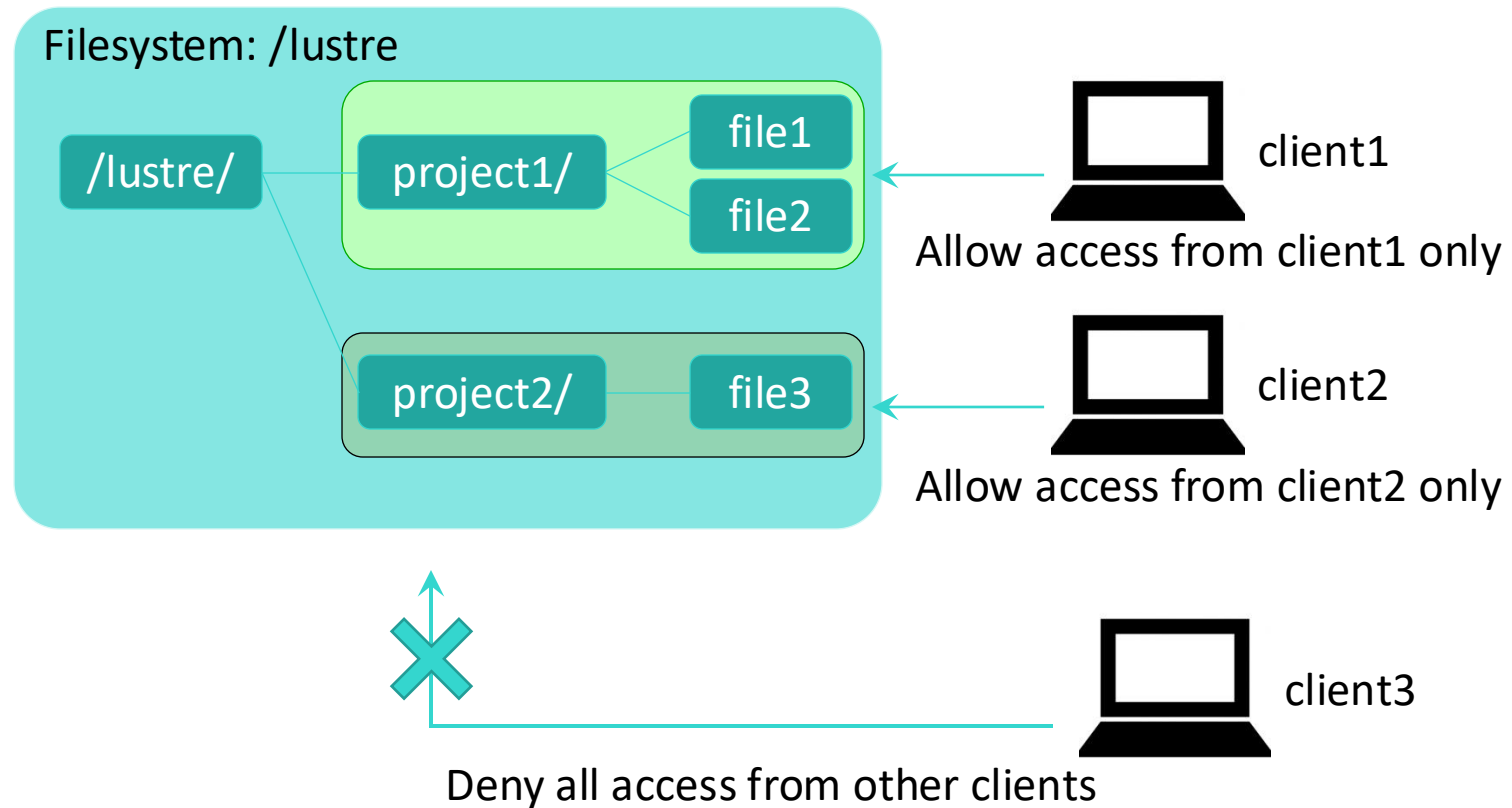
▶ Activate nodemap

```
mgs# lctl nodemap_activate 1
```

▶ Allow time for nodemap definitions to propagate.

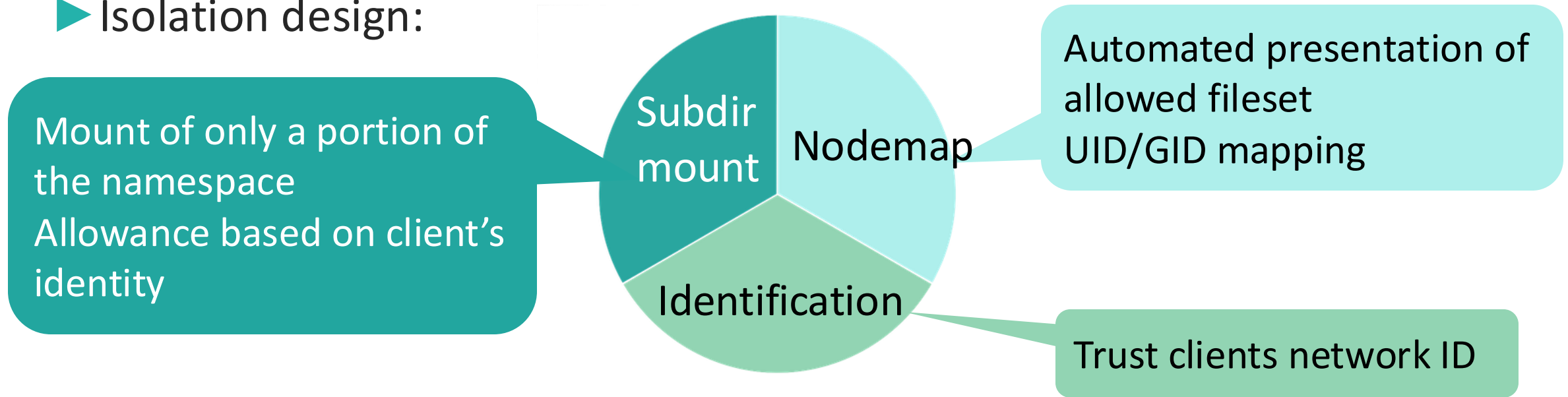
- Change events are queued and distributed across the cluster.
 - This can take tens of seconds.
- Nodemap changes should be implemented a minute before use, to ensure servers are up-to-date.

Multi-Tenancy: Rough Idea



Multi-Tenancy: Concept

► Isolation design:



► Isolation enables Multi-tenancy:

- Different populations of users on the same file system
- Isolation of these different populations of users

Nodemap for Multi-Tenancy

- ▶ As long as clients NIDs can be trusted, *fileset* property guarantees isolation.

```
mgs# lctl nodemap_set_fileset --name <nodemap_name> --fileset=/<directory>
```

- ▶ Groups of clients assigned to each tenant can change over time.
 - Needs to update tenant definitions in nodemaps
- ▶ Can also force read-only for all mounts in the nodemap
 - Property *readonly_mount*

Nodemap for Multi-Tenancy

▶ How can client NIDs be trusted?

1. Users do not have root access on clients.

⇒ Cannot modify Lustre configuration

2. Make use of strong authentication to prevent address spoofing.

- Kerberos

- Shared-Secret Key

- How does it work?

- Maliciously modified client NID will not match client's key.

- Lustre servers will refuse connection.

Nodemap for Multi-Tenancy

▶ How can client NIDs be trusted?

3. Enforce network level constraints.

- VLANs or IB partitions
- LNet routers
- How does it work?
 - Maliciously modified client NID will not match VLAN/IB pkey.
 - Network requests will not reach Lustre servers.

Recently added Nodemap feature

- ▶ New property to refine admin capabilities: [LU-16524](#)
 - Role-Based Admin Control, *rbac*
 - Available roles are (multiple choices are possible):
 - *byfid_ops*, to allow operations by FID (e.g. 'lfs rmfid').
 - *chlg_ops*, to allow access to Lustre Changelogs.
 - *dne_ops*, to allow operations related to DNE (e.g. 'lfs mkdir').
 - *file_perms*, to allow modifications of file permissions and owners.
 - *quota_ops*, to allow quota modifications.
 - *fscrypt_admin*, to allow fscrypt admin actions
 - lock/unlock encrypted dir. always possible

Recently added Nodemap feature

- ▶ New property to refine admin capabilities: [LU-16524](#) (cont)
 - Role-Based Admin Control, *rbac*
 - Available roles are (multiple choices are possible):
 - *ignore_root_prjquota*, to control if project quota is enforced for root.
 - *hsm_ops*, to allow HSM actions (archive, release, restore, ...).
 - *local_admin*, to keep root capabilities even if mapped.

Recently added Nodemap feature

► New possibility to ease mapping definitions:

- Ability to declare an idmap range: [LU-17922](#)
- Syntax is:
 - `<clientid_start> - <clientid_end> : <fsid_start> [- <fsid_end>]`
 - `fsid_end` optional

```
mgs# lctl nodemap_add_idmap --name nm --idtype uid --idmap 500-510:10000
```

- Works also to delete idmap ranges:

```
mgs# lctl nodemap_del_idmap --name nm --idtype uid --idmap 500-510:10000
```

Recently added Nodemap feature

► Mapping offsets: [LU-18109](#)

- Defining and maintaining individual mappings for each ID used by a tenant can be super heavy for admins.
 - Having millions of mappings in memory can consume a lot of resources.
- Storage admins might not have access or knowledge of tenant IDs (Cloud Service Provider).
- But still need to isolate IDs from different tenants.
- New option to specify an ID mapping offset:
 - Every tenant local ID automatically mapped to "ID+OFFSET" on storage.
 - No need to add mapping rules for new users in the tenant.

Mapping offsets commands

```
(1) mgs# lctl nodemap_add_offset --name Tenant1 --offset 100000 --limit 70000
```

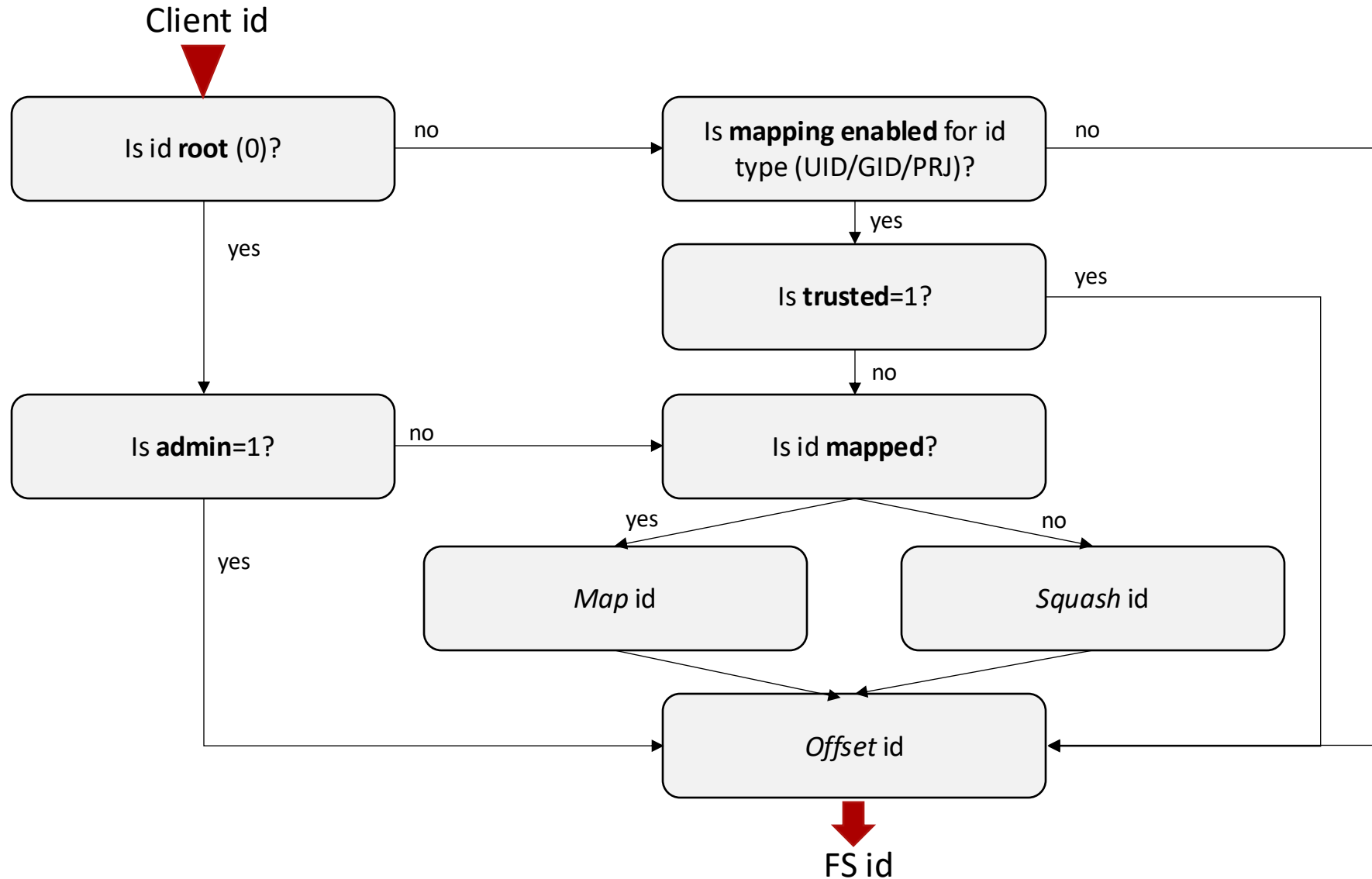
Map client IDs from range 0-69999 to filesystem IDs in range 100000-169999

--limit specifies the number of IDs mapped by the range.

An offset range cannot overlap into another's offset range.

⇒ Guarantees tenants have disjoint ID spaces.

Id mapping flow chart



Upcoming Nodemap features

- ▶ Make multi-tenancy configuration easier
 - Dynamic nodemaps
 - Multiple filesets

Upcoming Nodemap features

► Dynamic nodemaps: [LU-17431](#)

- For workflows that impose frequent change of nodemap definitions
 - Nodemaps created and removed on a per-job basis
 - To contain only the nodes in the job for a specific runtime environment
- Current mechanism is not adapted.
 - Nodemaps in the config would be slow to process.
 - Propagation time can be problematic.
- Proposal:
 - Have **in-memory** nodemaps, created directly on servers.
 - Requires some external orchestration to set nodemaps consistently across servers
 - » e.g. "clush -a lctl set_param ..."

Dynamic nodemaps: LU-17431 (cont.)

- ▶ See dynamic nodemaps as a refinement of a generic behavior enforced by a regular nodemap.
- ▶ Dynamic nodemaps are hierarchical.
 - They have a parent (possibly `default`).
 - NID ranges of the child nodemap **must** be included in parent's NID ranges.
 - Child nodemap inherits all the parent nodemap's properties...
 - ...and the id mappings.
 - Then properties/mappings can be refined.
 - Sub-nodemaps can only lower privileges, unless parent grants permission to raise.

Dynamic Nodemap commands



```
(1) mds# lctl nodemap_add -d -p Tenant1 subTen1
```

-d for dynamic, -p for parent

```
(2) mds# lctl nodemap_add_range --name subTen1 --range 192.168.1.[100-150]@tcp
```

Range for subTen1 must be **included** in parent Tenant1 ranges.

```
nodemap.subTen1.admin_nodemap=0
nodemap.subTen1.deny_unknown=0
nodemap.subTen1.fileset=/Tenant1_dir
nodemap.subTen1.idmap=
[
  { idtype: uid, client_id: 530, fs_id: 11000 },
  { idtype: gid, client_id: 530, fs_id: 11000 },
  { idtype: projid, client_id: 101, fs_id: 1001 }
]
nodemap.subTen1.readonly_mount=0
nodemap.subTen1.squash_gid=65534
nodemap.subTen1.squash_projid=65534
nodemap.subTen1.squash_uid=65534
nodemap.subTen1.trusted_nodemap=0
nodemap.subTen1.rbac=file_perms,dne_ops,quota_ops,byfid_ops,chlg_ops,fscrypt_admin
```

Properties inherited from parent Tenant1.

Dynamic Nodemap commands

```
(1) mds# lctl nodemap_modify --name subTen1 --property rbac --value  
dne_ops,byfid_ops,chlg_ops,fsencrypt_admin
```

Remove file_perms and quota_ops.

```
(2) mds# lctl nodemap_modify --name subTen1 --property deny_unknown --value 1
```

Deny any unmapped access.

Upcoming Nodemap features

► Multiple filesets: [LU-18357](#)

- Fileset is automatically presented to clients when mounting.
 - ⇒ Restricts namespace seen by clients
- But it can be too limiting to be restricted to only one subdirectory.
- Multiple filesets allow to choose between authorized subdirectories.
 - *Primary* fileset always used as default
 - *Alternate* filesets for other allowed subdirectories
 - Clients can ask for one subdirectory when mounting.
- Bonus: ability to force read-only each fileset individually

Multiple fileset commands

```
(1) mgs# lctl nodemap_fileset_add --nodemap Tenant2 --fileset /project
```

```
(2) mgs# lctl nodemap_fileset_add --alt --ro --nodemap Tenant2 --fileset /data_src
```

Add a primary fileset /project as rw, and alternate fileset /data_src as ro.

```
nodemap.Tenant2.fileset=  
[  
  { primary: /project },  
  { alternate: /data_src, mode: ro }  
]
```

```
(3) client# mount -t lustre mgs@tcp:/testfs /mnt/lustre
```

Client gets content of directory “project” in /mnt/lustre/.

```
(3) client# mount -t lustre mgs@tcp:/testfs/data_src /mnt/lustre
```

Client gets content of directory “data_src” in /mnt/lustre/,

but with read-only access.

Lustre Multi-Tenancy – wrap-up

- ▶ Nodemap is a mature feature.
- ▶ And still evolving constantly:
 - RBAC
 - Idmap ranges
 - Mapping offsets
- ▶ Nodemap is the feature of choice for Isolation and Multi-tenancy.
- ▶ And we will keep on adding new capabilities:
 - Dynamic nodemaps, multiple filesets, ...



Whamcloud

Thank you!

sbuisson@whamcloud.com

