

Review of CSI approaches for Lustre on Kubernetes

Matt Raso-Barnett – The Lustre Collective

Outline

- Many scientific, HPC and AI workloads increasingly target Kubernetes as the expected runtime environment
- Such applications still need high-performance, network-based storage
- Lustre gains from elevating CSI as a first-class supported runtime environment

What Kubernetes asks of Lustre

Kubernetes Storage Abstractions

What a CSI driver does



What exists today

What is a *Volume* for Lustre?

Comparison of existing drivers

Weaknesses of the 'directory' drivers



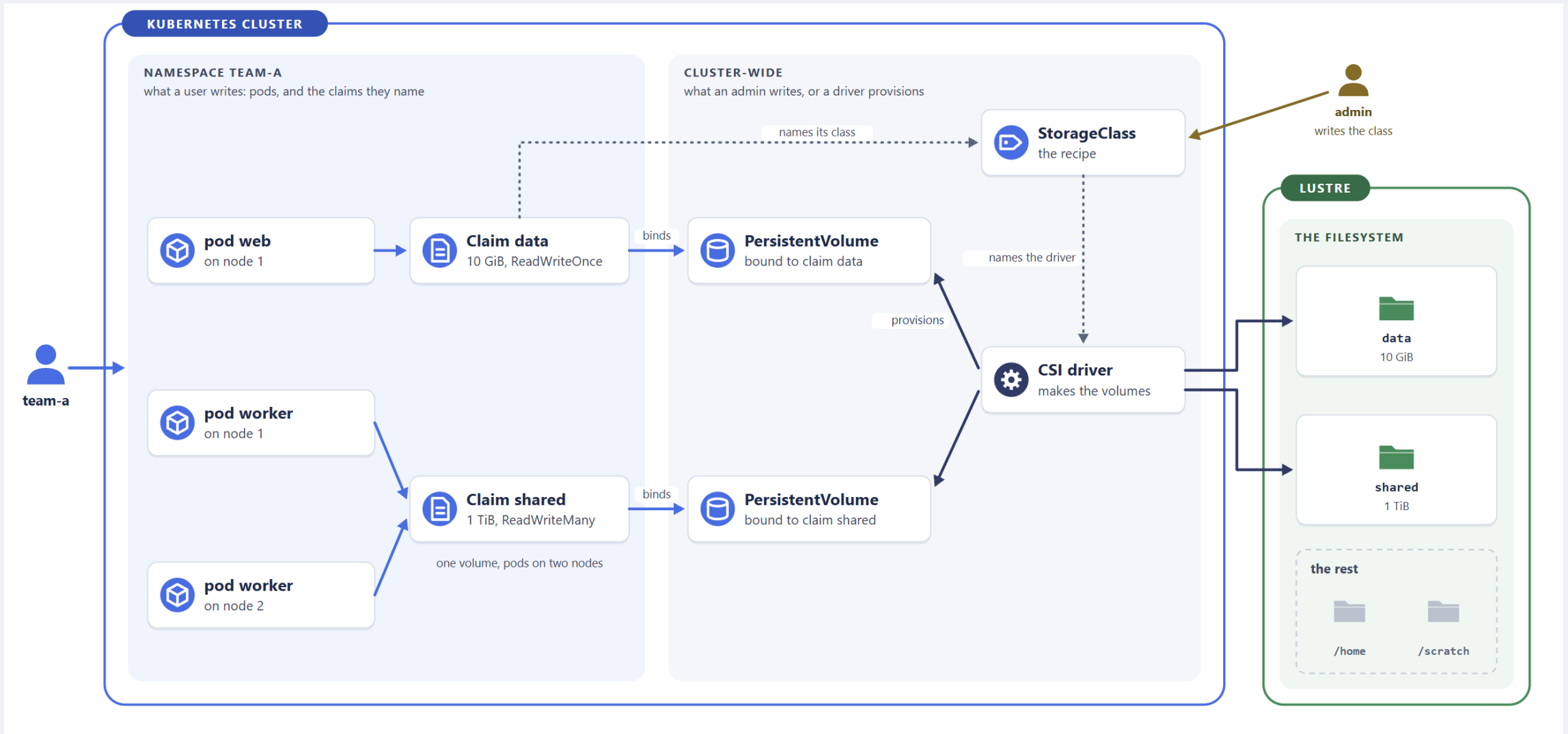
A Lustre Volume Driver

Can we make a Lustre *Volume* ?

Proof of Concept Design

Review of approach

Kubernetes Storage Abstractions



The user request path

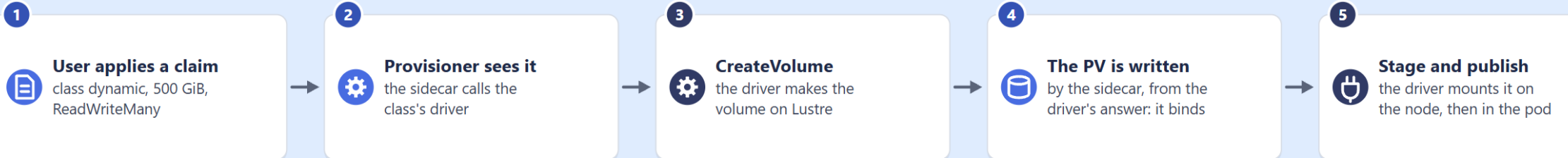
WHO ACTS

- the user, and Kubernetes
- the CSI driver
- an admin

THE CSI DRIVER DECIDES

what CreateVolume makes, and what a node mounts. Kubernetes does the rest.

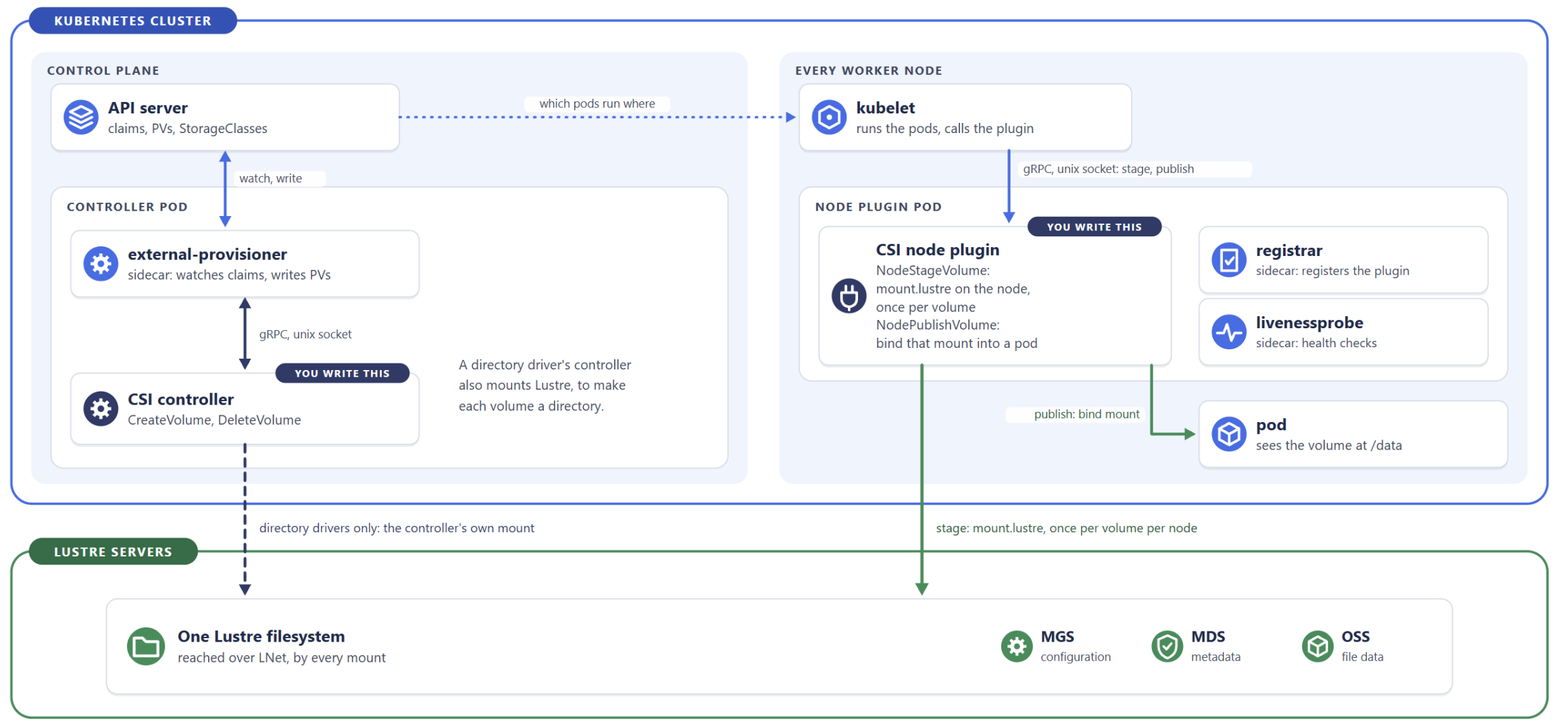
DYNAMIC: A VOLUME MADE ON DEMAND



STATIC: DATA THAT ALREADY EXISTS



What a CSI driver is

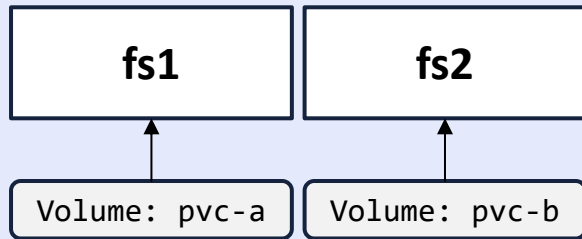




What is a *Volume* for Lustre?

How have existing CSI drivers implemented a *Volume*?

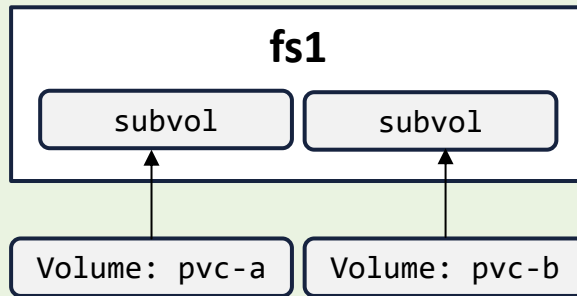
Filesystem per Volume



AWS FSx for Lustre
Google Managed Lustre
Azure Managed Lustre

Driver creates a filesystem on-demand via cloud API

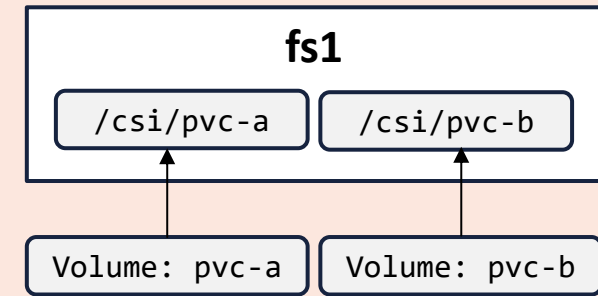
Server-side sub-namespace inside one filesystem



CephFS subvolume (ceph-csi)
IBM StorageScale fileset

- Driver creates through a management API on the server
- subvolume carries its own quota, identity or snapshot domain
- Driver does not access full filesystem

Sub-directory inside one filesystem



Lustre CSI Drivers: DDN, HPE, luscsci
BeeGFS
csi-driver-nfs

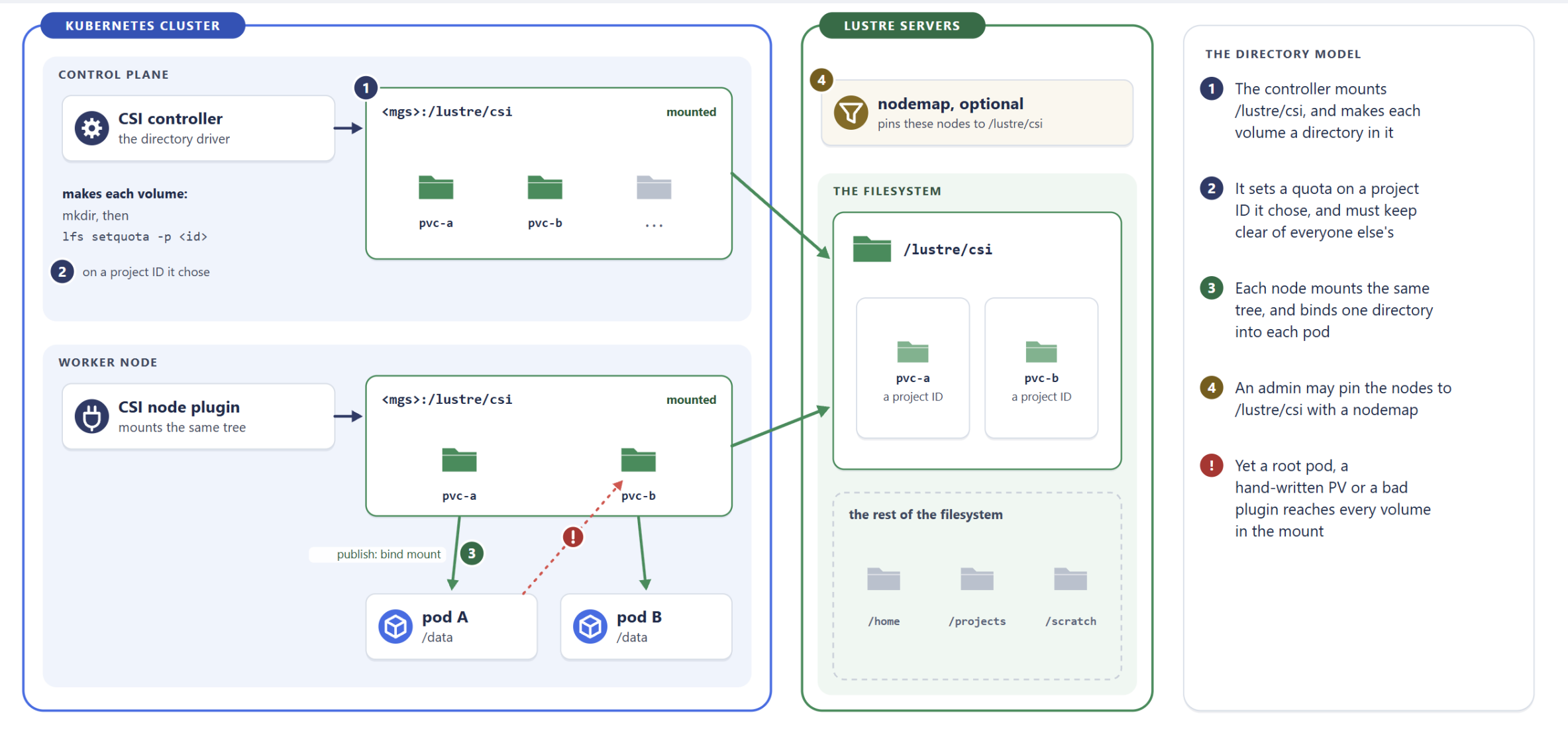
- Driver mounts filesystem or sub-directory mount, creates directory per-volume
- Driver composes everything itself

Comparison of Lustre ‘Directory’ Drivers

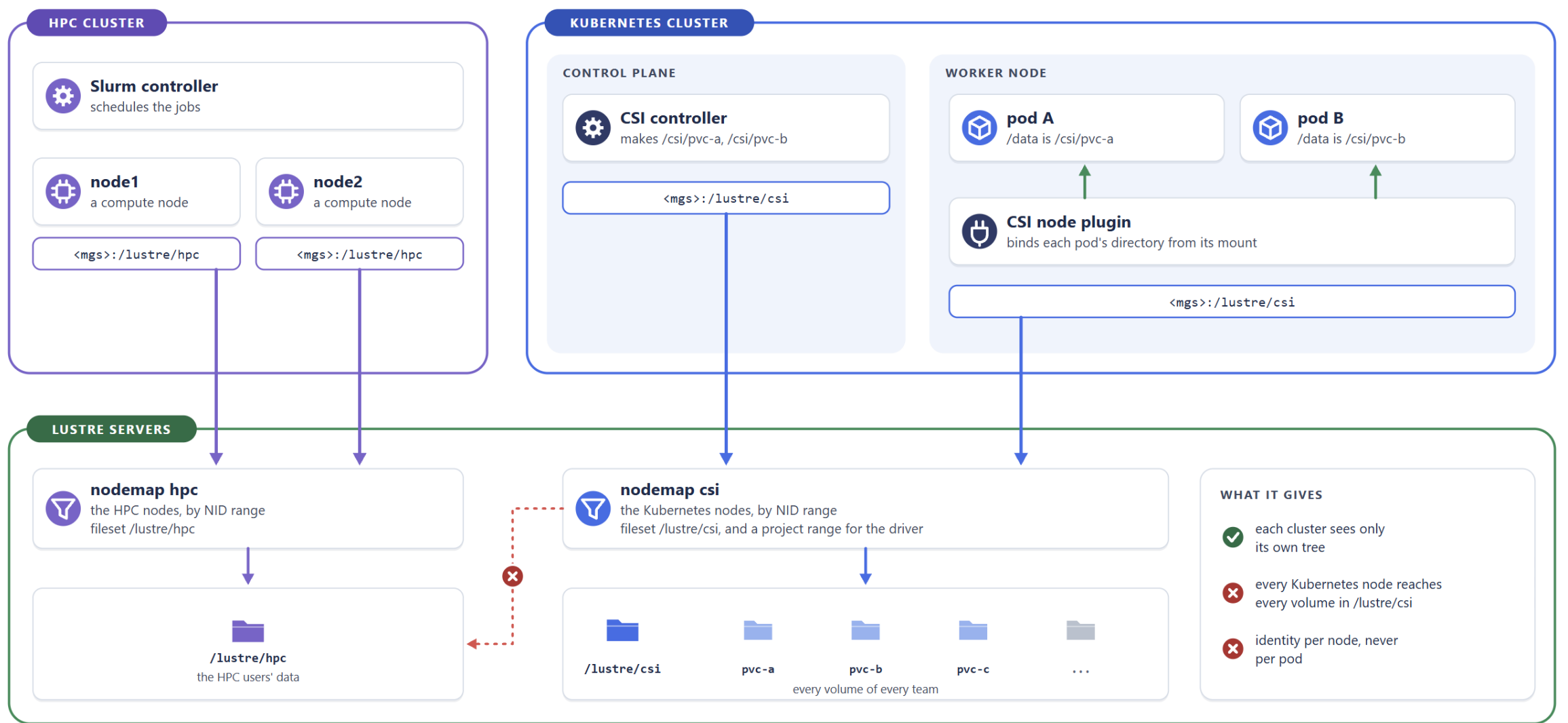
	DDN EXA CSI	HPE CSI	luscsi
Open source			
Static provisioning			
Dynamic provisioning			
Enforced capacity / Quota			
Snapshots			
Encrypted volumes			
Namespace to storage identity			
Mount authentication			

- <https://github.com/DDNStorage/exa-csi-driver>
- <https://github.com/HewlettPackard/lustre-csi-driver>
- <https://github.com/luskits/luscsi>

Security Model of a 'Directory' CSI Driver



Security Model of Shared HPC/Kubernetes Cluster



Weaknesses of the ‘directory’ approach

Containment

CSI Drivers have complete unrestricted access to the namespace

Can limit their blast-radius via Nodemap + fileset subdirectory mount but all user volumes within namespace are accessible

Authentication

Server-side Nodemaps must be manually configured, with NID-based membership

Moving to GSS-Identity for nodemaps can somewhat improve this, but keys must be managed between client and server

Identity

All volumes share a singular identity, quota and QoS domain with entire filesystem

CSI Driver can be configured with a ‘private’ range, by convention of Quota IDs to provision into

QoS

All volumes share server resources, with no QoS or fairshare

Volumes are just directories on a shared namespace



Can we make a Lustre *Volume* ?

Proposal for a Lustre ‘Volume’

Lustre Volume

Nodemap

GSS Identification – SSK key

ID Range + offset mapping (UID/GID/ProjID)

LQA – Aggregate Quota over ID range

Fileset

Lustre 2.18

The containing ‘object’ and policy definition

Ties all primitives together. Customise behaviour through rbac, identity policy...

Key-based Authentication ([LU-19079](#))

Key embeds nodemap name – clients can only mount volume with correct key

Unique ID domain ([LU-18109](#))

Volume’s ID range is private and not shared with other volumes

Volume Quota ([LU-18222](#))

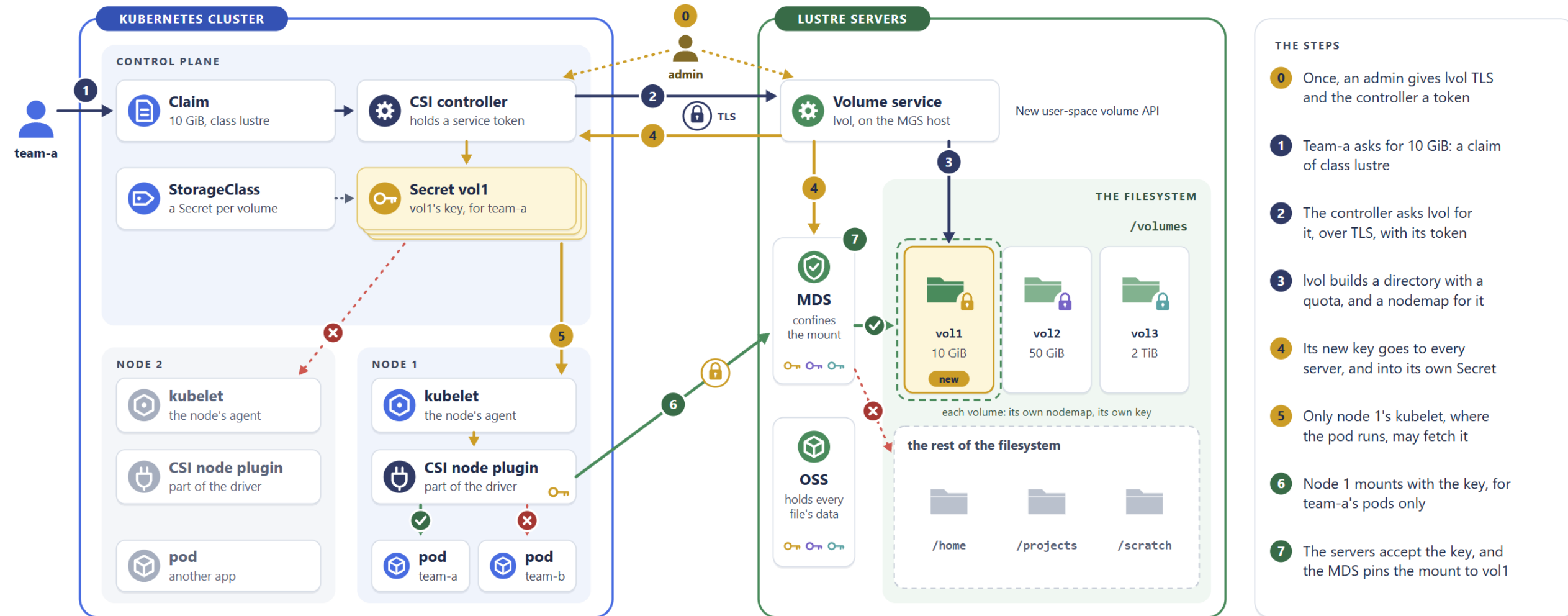
Volume capacity is bound by LQA quota over volumes ID range

Unique namespace

Client mount is forcibly bound only to the Volume’s namespace

Building a Lustre 'Volume' Service

Proof of concept design – volume creation and key-management via API



PoC - Creating a PVC via 'lvol'

Kubernetes

head1

```
$ kubectl get pvc walk
NAME      STATUS  VOLUME                                     CAPACITY  ACCESS MODES  STORAGECLASS
walk      Bound   pvc-37fc96f6-c868-4e9b-bb5d-ae1ee80f921a  1Gi       RWX           lustre-volumes-offset
$ kubectl get pv pvc-37fc96f6-... -o yaml
volumeHandle: 10.0.0.1@tcp0,...:/lustre/volumes/v576e1b44052e
```

lvol

server1

```
# lvol name pvc-37fc96f6-c868-4e9b-bb5d-ae1ee80f921a
v576e1b44052e
# lvol list
[ { ..., "name": "pvc-37fc96f6-...", ..., "size_bytes": 1073741824, ..., "vol": "v576e1b44052e" } ]
# lvol get v576e1b44052e
{ ..., "gid": 1000, ..., "limit": 10000, ..., "offset": 1000000, "projid": 1000000, ..., "uid": 1000, ... }
# lvol pool
{"ready": 10, "target": 10, "dir": "/etc/lustre/ssk-lvol-pool"}
```

Lustre servers

server1

```
# ls -lnd /mnt/lustre/volumes/v576e1b44052e
drwxrws---. 2 1001000 1001000 4096 Sep 25 19:50 /mnt/lustre/volumes/v576e1b44052e
# lfs project -d /mnt/lustre/volumes/v576e1b44052e
1000000 P /mnt/lustre/volumes/v576e1b44052e
# lctl lqa list --fsname lustre --name v576e1b44052e
name: v576e1b44052e
ranges: 1000000-1009999
# lfs quota -h -P --lqa v576e1b44052e /mnt/lustre
      Filesystem   used  bquota  blimit  bgrace   files  iquota  ilimit  igrace
/mnt/lustre       4k      0k      1G      -         0        0        0      -
# lctl get_param nodemap.v576e1b44052e.exports
[ { nid: 10.0.0.103@tcp, uuid: c42e8d3b-..., dev: lustre-MDT0000 },
  { nid: 10.0.0.102@tcp, uuid: 24e2d274-..., dev: lustre-MDT0000 }, ... ]
```

PoC – How the Volume appears on the nodes

Driver

node plugin, client2

```
$ kubectl -n kube-system logs lustre-csi-node-l9m8p -c plugin
INFO node_stage_volume{.../volumes/v576e1b44052e}: staged mount=.../globalmount/mount
INFO node_publish_volume{.../volumes/v576e1b44052e}: published target=/var/lib/kubelet/pods/a93f40d8-.../mount
$ kubectl -n kube-system exec lustre-csi-node-l9m8p -c plugin -- ls -l /etc/lustre/ssk/lvol
-r----- 1 root root 2407 Sep 25 18:51 v576e1b44052e.key
$ kubectl -n kube-system exec lustre-csi-node-l9m8p -c plugin -- grep ' /etc/lustre/ssk/lvol ' /proc/mounts
tmpfs /etc/lustre/ssk/lvol tmpfs rw,seclabel,relatime,size=1748048k,inode64 0 0
```

Host

client2

```
# findmnt -t lustre -o TARGET,SOURCE
/var/lib/kubelet/plugins/.../330607fd.../globalmount/mount 10.0.0.1@tcp,...:/lustre/volumes/v576e1b44052e
/var/lib/kubelet/pods/a93f40d8-.../volumes/.../pvc-37fc96f6-.../mount 10.0.0.1@tcp,...:/lustre/volumes/v576e1b44052e
# lctl dl
1 UP lov lustre-clilov-ffff8ac34778e000 24e2d274-388e-4a04-87a8-3826542c3ae9 4
3 UP mdc lustre-MDT0000-mdc-ffff8ac34778e000 24e2d274-388e-4a04-87a8-3826542c3ae9 5
# awk '/lustre/ {print $1, $8, $9}' /proc/keys
1f2f63bf user lustre:lustre:24e2d274-388e-4a04-87a8-3826542c3ae9:
# lctl get_param mdc.*.srpc_info
rpc flavor: skn bulk flavor: skn
```

Pod

client2 and client3

```
$ kubectl exec walk-59dd56749c-8mmhf -- sh -c 'id; grep " /data " /proc/mounts; df -h /data'
uid=2000 gid=2000 groups=1000,2000
10.0.0.1@tcp,...:/lustre/volumes/v576e1b44052e /data lustre rw,seclabel,flock,...,statfs_project,...
... 1.0G 4.0K 1024.0M 0% /data
$ kubectl exec walk-59dd56749c-8mmhf -- sh -c '...; ls -ln /data'
-rw-r--r-- 1 2000 1000 27 Sep 25 18:51 from-client2
$ kubectl exec walk-59dd56749c-fwqcd -- sh -c '...; cat /data/from-client2'
from walk-59dd56749c-8mmhf
```

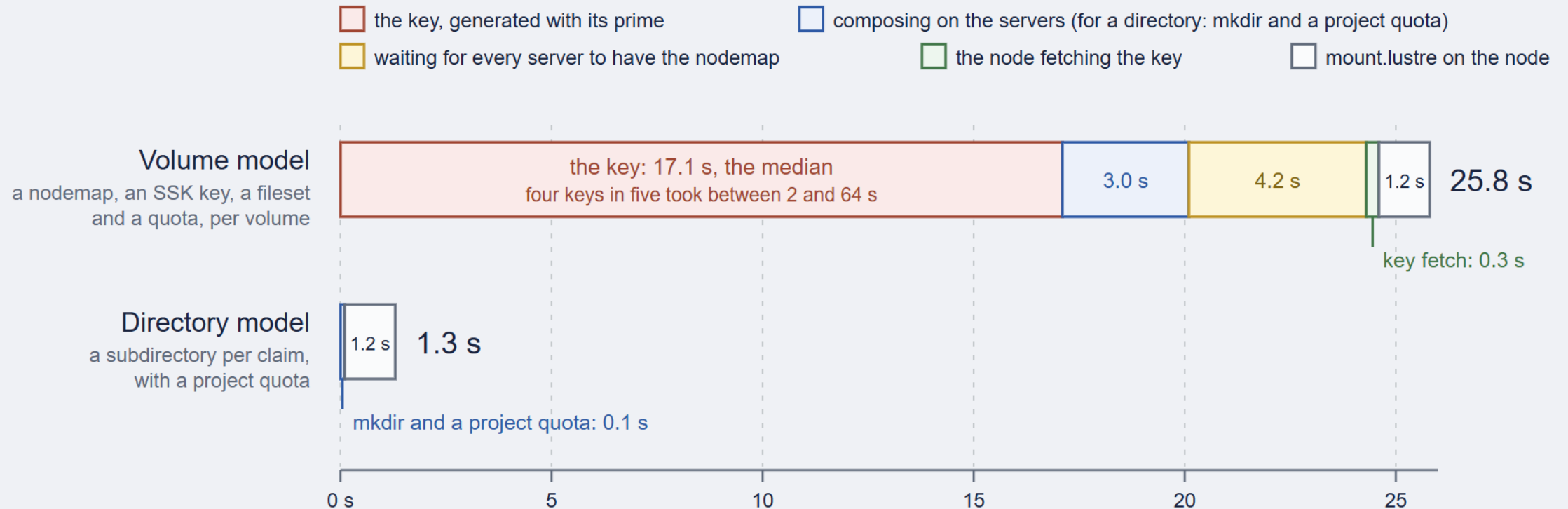
Lustre servers

server1

```
# ls -ln /mnt/lustre/volumes/v576e1b44052e
-rw-r--r-- 1 1002000 1001000 27 Sep 25 19:51 from-client2
-rw-r--r-- 1 1002000 1001000 27 Sep 25 19:51 from-client3
```

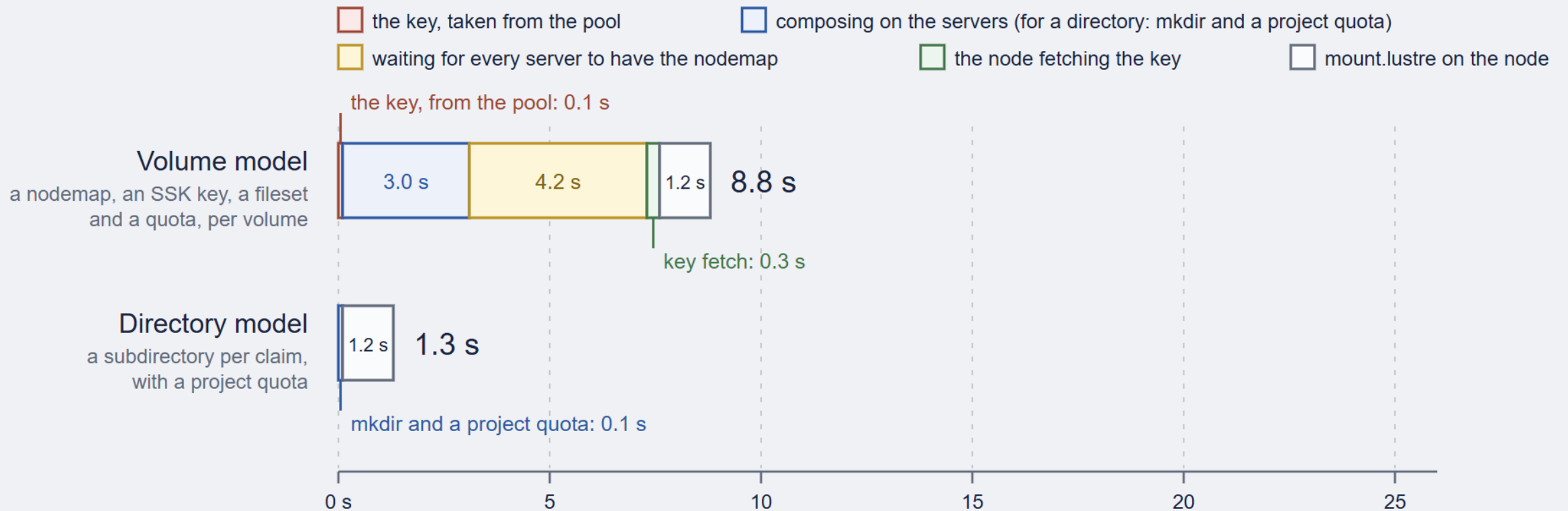
Challenges to the ‘volume’ approach

Creating a new volume takes significantly longer, dominated by the time for SSK key generation



SSK Pre-Generated Key Pool for volumes

lvol generates keys ahead on the MGS host, so new volume creations just take an allocated key



Advantages to the ‘volume’ approach

Containment

CSI Driver restricted to only create and mount its own volumes

Authentication

Every volume requires it's key to mount. Nodes only given key when running a pod

Identity

Every volume has a unique ID and Quota domain

QoS

In development - volume-level QoS/Fairshare – but Nodemap provides a handle for NRS TBF ([LU-17902](#))

Conclusion

Lustre has the technology primitives

Built for multitenancy, a ‘volume’ is identical

- Nodemap
- GSS Identification – SSK key
- ID Range + offset mapping (UID/GID/ProjID)
- LQA – Aggregate Quota over ID range
- Fileset

It lacks the abstraction and API

Nothing currently composes the primitives into a volume. Requires higher-level management service



Volume service
Ivol, on the MGS host

Open-source CSI driver would be of benefit to the community

- Drive further usage of Lustre in containerized environments
- Would benefit the community to have a ‘default’ driver, used in testing/development
- Excellent use-case for many of Lustre’s advanced capabilities to build secure ‘volumes’

	Potential Community Driver
Open source	
Static provisioning	
Dynamic provisioning	
Enforced capacity / Quota	
Snapshots	
Encrypted volumes	
Namespace to storage identity	
Mount authentication	

