LUG 2026

Lustre on Kubernetes:
Leveraging Advanced Lustre Features via CSI Integration

Shuichi Ihara
DataDirect Networks

April 29, 2026

[ANY DATA] [ANY APPLICATION] [ANYWHERE]

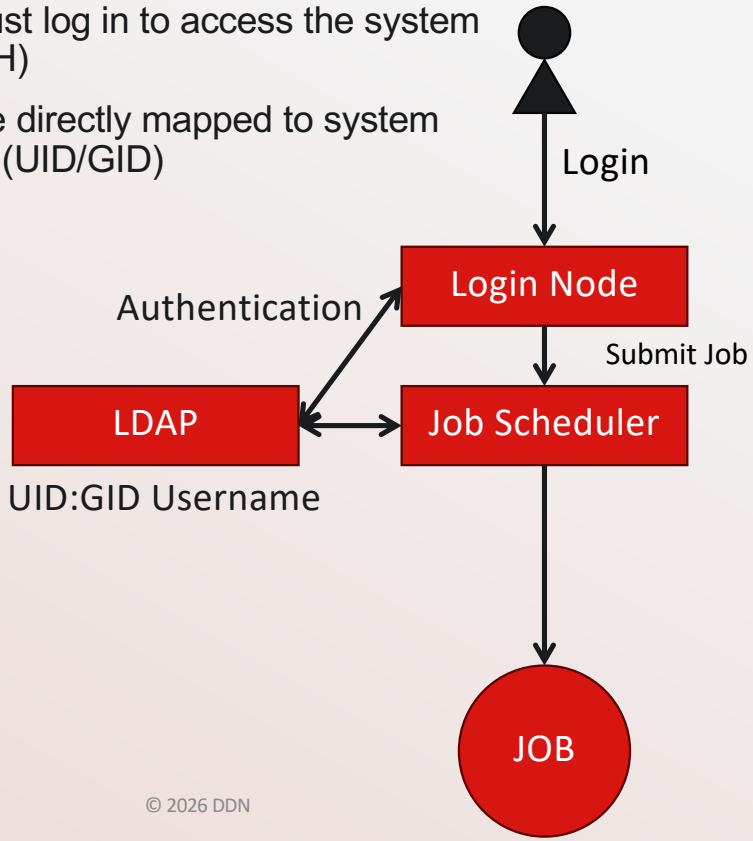
AI Infrastructure is Shifting Toward Kubernetes

- AI Infrastructure Trends
  - Large-scale GPU clusters and distributed training
  - High-performance networks (InfiniBand, RoCE)
  - Kubernetes is emerging as the standard platform
- Changing User Model
  - Users focus on applications, not infrastructure
  - Storage expertise is no longer required, as storage is abstracted by Kubernetes (CSI, PV)
- Implication for Storage
  - High-performance storage (e.g., Lustre) is still essential
  - But its advanced features are not easily exposed

User Interaction Models: Traditional vs Kubernetes

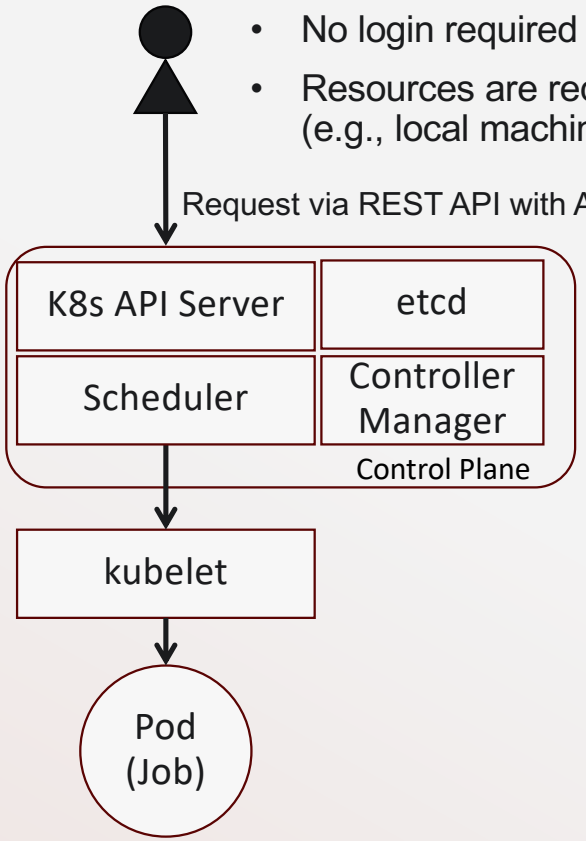
Traditional Model

- Users must log in to access the system (e.g., SSH)
- Users are directly mapped to system identities (UID/GID)

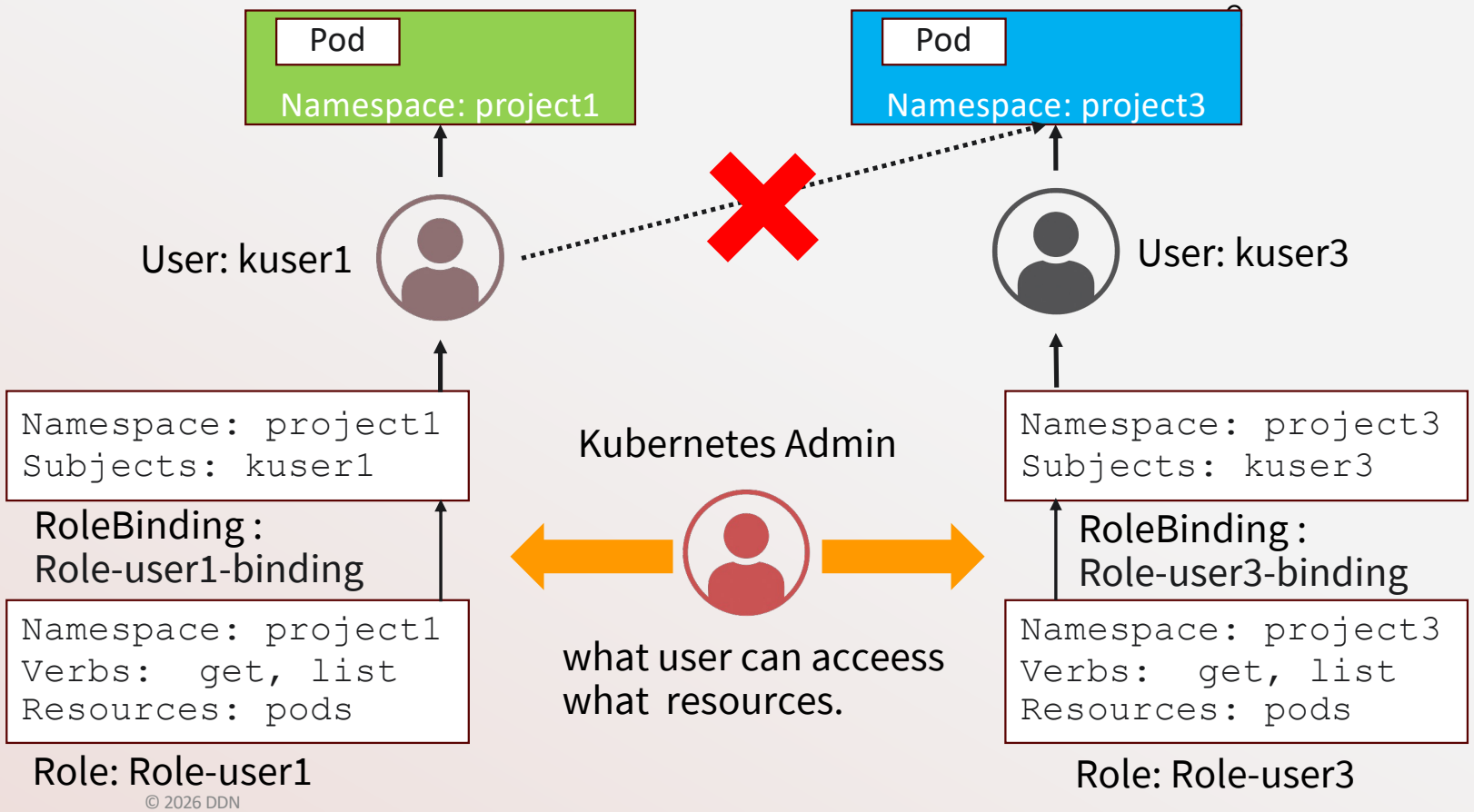


Kubernetes

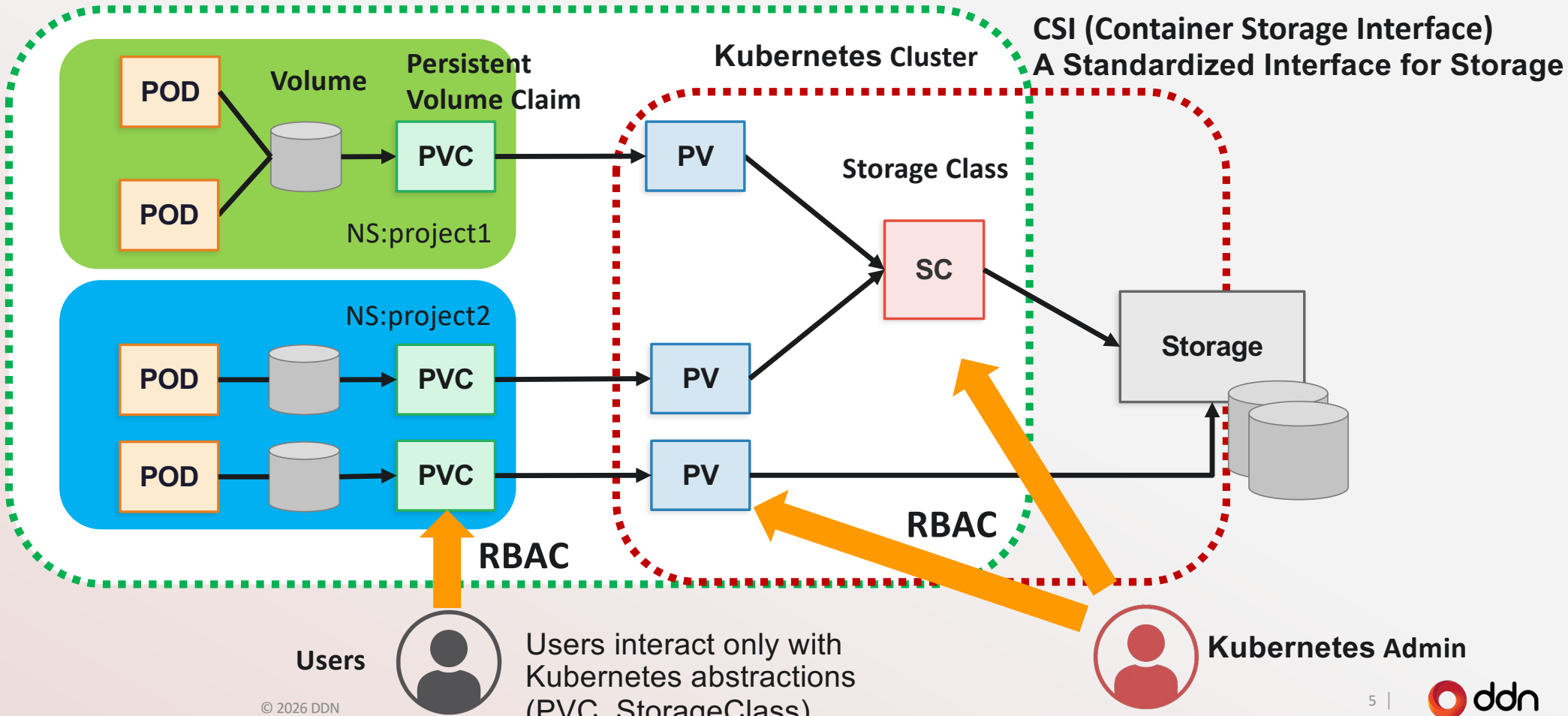
- No login required to access infrastructure
- Resources are requested via APIs from anywhere (e.g., local machine)



Namespace, Role, and RoleBinding in Kubernetes



Storage is Abstracted in Kubernetes



Lustre on Kubernetes - EXAScaler CSI Driver for Kubernetes

- Enables high-performance Lustre storage for Kubernetes workloads
  - Proven at scale in AI environments (Running on thousands of GPUs)
- Supports OpenShift and upstream Kubernetes (x86_64, ARM64)
- Native integration with Kubernetes
  - Works with Kubernetes storage abstractions (PVC, StorageClass)
- Extends advanced Lustre capabilities to Kubernetes
  - Multi-tenancy support (Nodemap + LQA integration)
  - Transparent Persistent Client Cache (Hot Nodes)
  - Encryption via KMS integration

How Storage is Used in Kubernetes

- **Traditional Model:**
  - admin → manual mount → users directly access shared filesystem
- **Kubernetes:**
  - No direct mount — storage is dynamically provisioned for applications

1. StorageClass (Admin)

Define the storage

```
kind: StorageClass
metadata:
  name: exascaler
provisioner: exa.csi.ddn.com
parameters:
  exaFS: 10.0.19.10@o2ib:/exafs
  ..
```

$ kubectl apply –f sc.yaml

2. PVC(User)

User requests storage

```
kind: PersistentVolumeClaim
metadata:
  name: exa-pvc
spec:
  storageClassName: exascaler
  resources:
    requests:
      storage: 1Ti
..
```

$ kubectl apply –f pvc.yaml

3. Pod(App)

App consumes storage

```
apiVersion: v1
kind: Pod
metadata:
..
volumeMounts:
    - mountPath: /data
  volumes:
    - persistentVolumeClaim:
        claimName: exa-pvc
```

How the CSI driver works with Lustre

Your local node

```
mylaptop$ kubectl apply -f pvc.yaml && kubectl get pvc
NAME    STATUS    VOLUME           CAPACITY   ACCESS MODES
pvc1    Bound     pvc-exa-60xxxx   1Ti        RWO
```

The CSI driver automatically assigns project IDs and enforces quotas

No manual Lustre operations required

On worker node

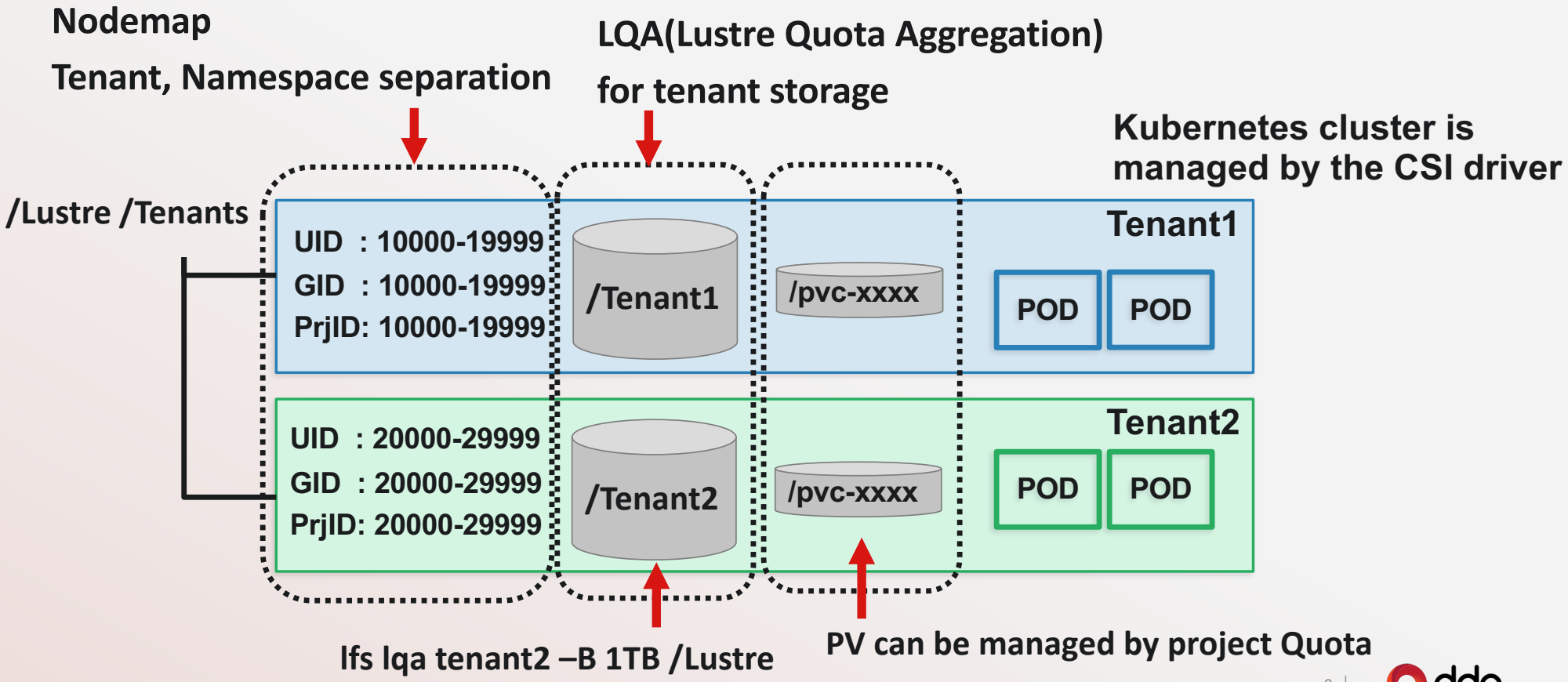
```
worker02:~$ sudo lfs project -d  /exafs/k8s/pvc-exa-60xxx
98939 P /exafs/pvc-exa-60xxx

worker02:~$ sudo lfs quota -p 98939 /exafs/ -h
Disk quotas for prj 98939 (pid 98939):
         Filesystem    used   bquota  blimit
         /exafs/       4k       0k      1T
```

```
mylaptop$ kubectl apply -f pod.yaml
mylaptop$ kubectl exec pod1 -- df -t lustre
Filesystem                                     1K-blocks  Used  Available Use% Mounted on
192.0.13.41@o2ib:192.0.13.42@o2ib:/exafs/k8s 1073741824     0 1073741824   0% /data
```

Your application accesses Lustre from the pod

Multi-tenancy support (Nodemap + LQA integration)

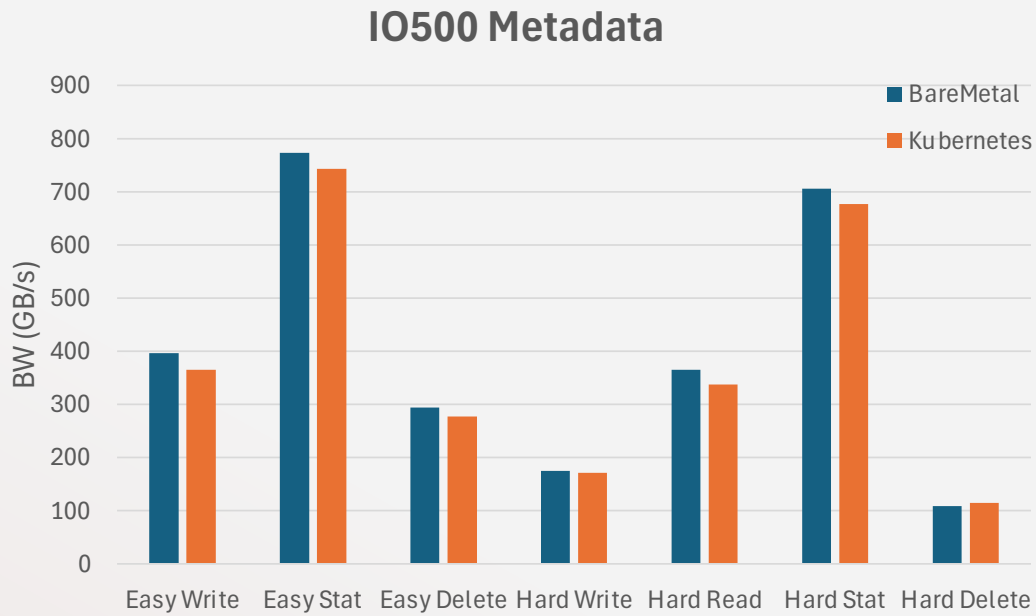


Use Case: IO500 on Kubernetes with EXAScaler and EXA-CSI Driver

- Run IO500 on a Kubernetes Cluster based on:
  - MPI Operators need to be installed
  - DDN EXAScaler CSI Driver
- Containerized IO500
  - Generate container image from Dockerfile
  - Integrate IO500 binaries into the container image
  - Step by Step Procedures https://github.com/sihara/io500-k8s
- Good demonstration for IO500 on Kubernetes
  - You also can manage your HPC MPI application in Kubernetes

IO500 Performance Comparison: Bare Metal vs Kubernetes

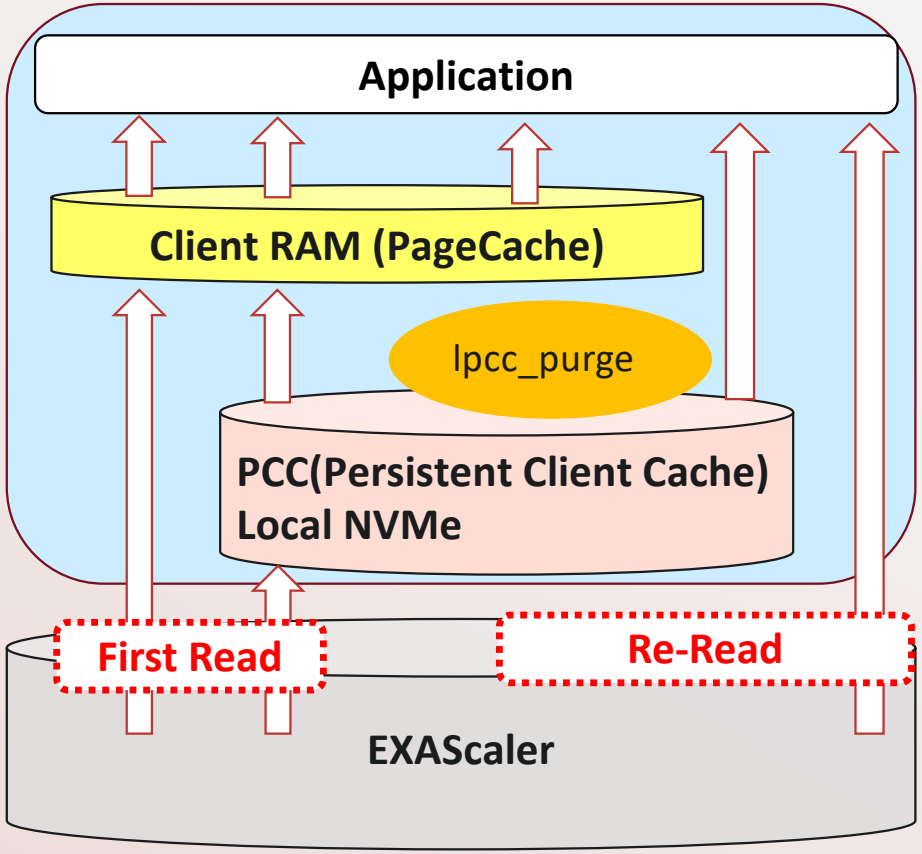
Running IO500 on 10 Bare Metal clients and Kubernetes pods using PVCs through the CSI driver



Bandwidth and metadata performance are identical in both cases

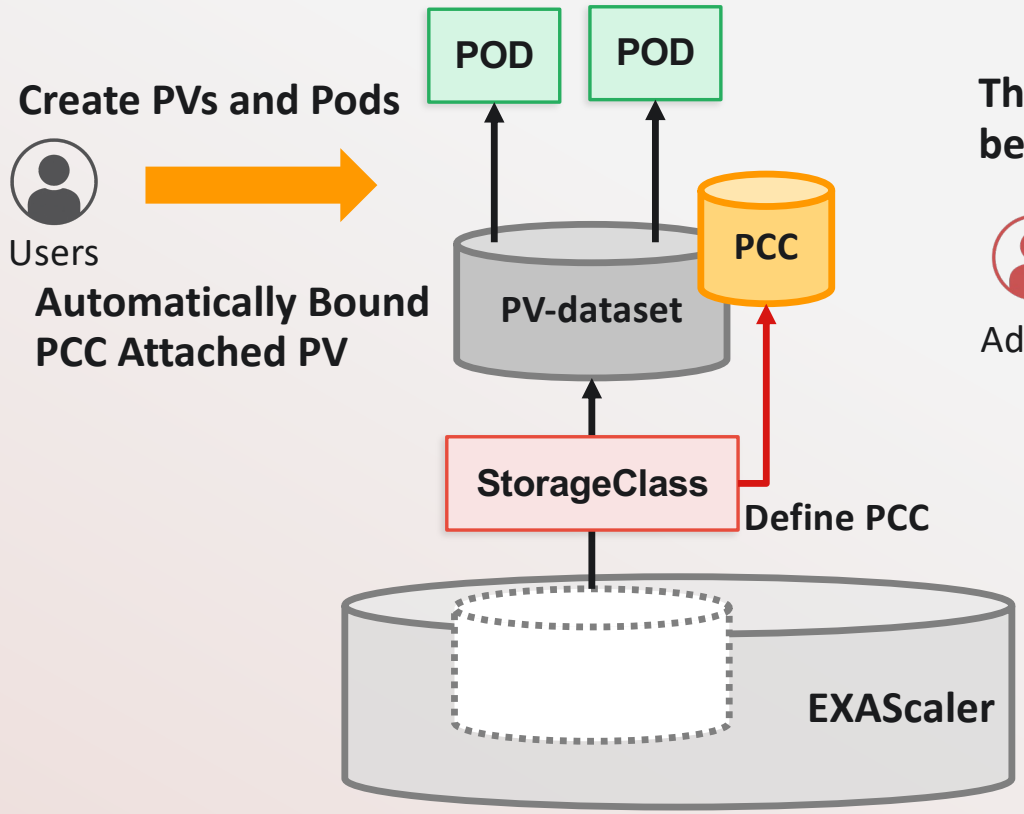
Submitted Results (10 Nodes, Research)
https://io500.org/submissions/view/751
https://io500.org/submissions/view/752

Use Case : PCC(Persistent Client Cache) Integration



- PCC enables intelligent caching in Lustre
  - Leverages local NVMe
  - Transparent extension of client-side cache
  - Supports flexible caching rules
- DDN Hot Nodes integrates PCC for AI/HPC workloads
  - Accelerates repeated-read workloads
  - Eliminates network traffics
  - lpcc_purge manages local cache capacity

Use Case : PCC(Persistent Client Cache) Integration



The administrator defines PCC behavior via StorageClass.

Admin

$ kubectl apply –f sc-pcc-dataset.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
…
parameters:
…
hotNodes: "true"
  pccCache: "/csi-pcc"
  pccAutocache: "fname={*.h5}"
  pccPurgeHighUsage: "90"
  pccPurgeLowUsage: "70"
  pccPurgeInterval: "30"
```

Use Case: Persistent Client Cache (PCC) on Kubernetes

Your local node

```
mylaptop$ kubectl apply -f pvc-pcc.yaml && kubectl apply -f pod.yaml
```

```
mylaptop$ kubectl -n myns exec pod1 --
sh -c 'dd if=/data/10GBfile.h5 of=…'
```

First Read

On worker node

```
worker$ sudo lctl get_param llite.*.stats | grep read_bytes
read_bytes                10560 samples [bytes]

worker$ lfs pcc state
10GBfile.h5, type: readonly, PCC_file: xxx/0x3c00013af:0x2:0x0

worker$ sudo du -h /mnt/pcc/
11G /mnt/pcc/
```

```
worker$ sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"
```

Drop all caches

```
mylaptop$ kubectl -n myns exec pod1 --
sh -c 'dd if=/data/10GBfile.h5 of=…'
```

Second Read

Lustre read bytes do not increase on the second read

```
worker$ sudo lctl get_param llite.*.stats | grep read_bytes
read_bytes                10560 samples [bytes]
```

- On the second read, no data is read from Lustre at all.
- All data is served directly from PCC.

Use Case : Lustre Client Encryption

Alice

/exafs/users/alice/encrypted 

```
alice $ fscrypt encrypt encrypted --source=custom_passphrase
```

```
alice $ fscrypt lock encrypted
```

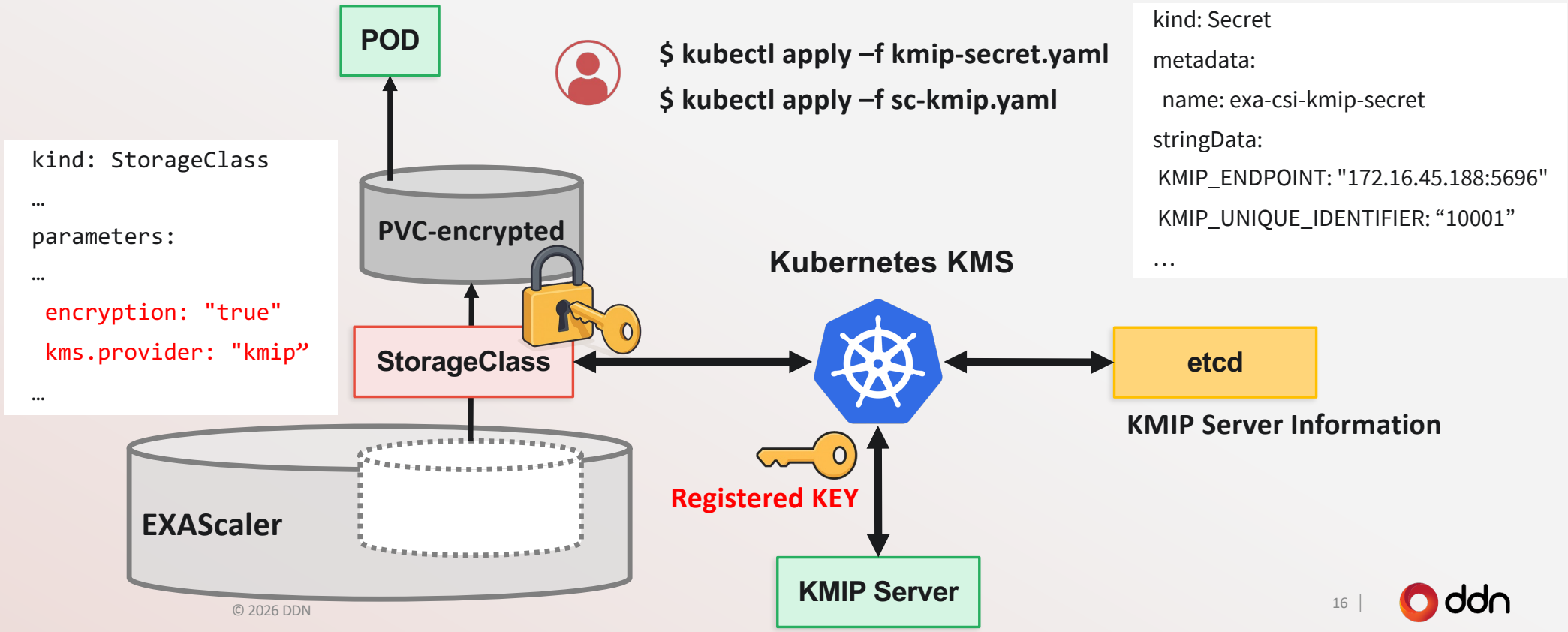
Bob

/exafs/users/bob/encrypted 

- Lustre supports Flexible Client Side Encryption Framework based on fscrypt
  - It’s scalable and high performance with the Client CPU offloading
  - Supports per-user, per-directory encryption
- Key management is challenging

Use Case :
Integrated Lustre Client Encryption in CSI Driver with KMIP

Admin configures KMIP and encryption via StorageClass and secrets



Use Case: Lustre Client Encryption with KMIP in Kubernetes

Encrypted PVC is attached to a pod

```
mylaptop$ kubectl get pvc
NAME         STATUS   VOLUME          CAPACITY
encrypted    Bound    pvc-exa-f3xxx   1Ti
```

```
mylaptop$ kubectl get pod -o wide
NAME       READY   STATUS    RESTARTS AGE.. NODE
secure1    1/1     Running   0        20s.. kube04
```

Clients automatically retrieve keys from KMIP via KMS and perform fscrypt unlock

```
mylaptop$ kubectl -n myns exec secure1 -- sh -c 'echo "encrypted file" > /data/secret.txt'
```

```
kube01:~$ sudo cat /exafs/k8s/pvc-exa-f3xx/secret.txt
cat: /exafs/k8s/pvc-exa-f3xx/secret.txt: Required key not available
```

File is not accessible without the encryption key

```
mylaptop$ kubectl get pod -o wide
NAME       READY   STATUS    RESTARTS   AGE..   NODE
secure1    1/1     Running   0          16m..   kube04
secure2    1/1     Running   0          119s.   kube02
```

Encrypted PVCs are seamlessly accessible across nodes without manual key handling

```
mylaptop$ kubectl -n myns exec secure2 -- cat /data/secret.txt
encrypted file
```

More Lustre feature integrations via CSI

Admin-defined features, automatically consumed via PVC

- Lustre Client-side Compression
  - Enabled via StorageClass parameters
- Block Device Access
  - ublk (user-space block driver) on Lustre
  - Improves performance compared to loopback
- Multiple Mount Options
  - Flexible mount configurations for optimal performance
- Tunable Lustre Client Parameters
  - For benchmarking and performance optimization

Conclusions

- Lustre remains a key foundation for HPC and AI infrastructure
- Kubernetes is already running at scale on DDN EXAScaler, powering large-scale GPU clusters
- EXAScaler CSI provides Kubernetes access to the EXAScaler/Lustre ecosystem
  - Seamless access to Lustre from Kubernetes workloads
  - Transparent integration for AI applications
- Unlocks advanced storage capabilities in Kubernetes
  - e.g., PVC recovery with Lustre TrashCan