Distributed Filesystems on Kubernetes Clusters

Federico Fornari, INFN-CNAF

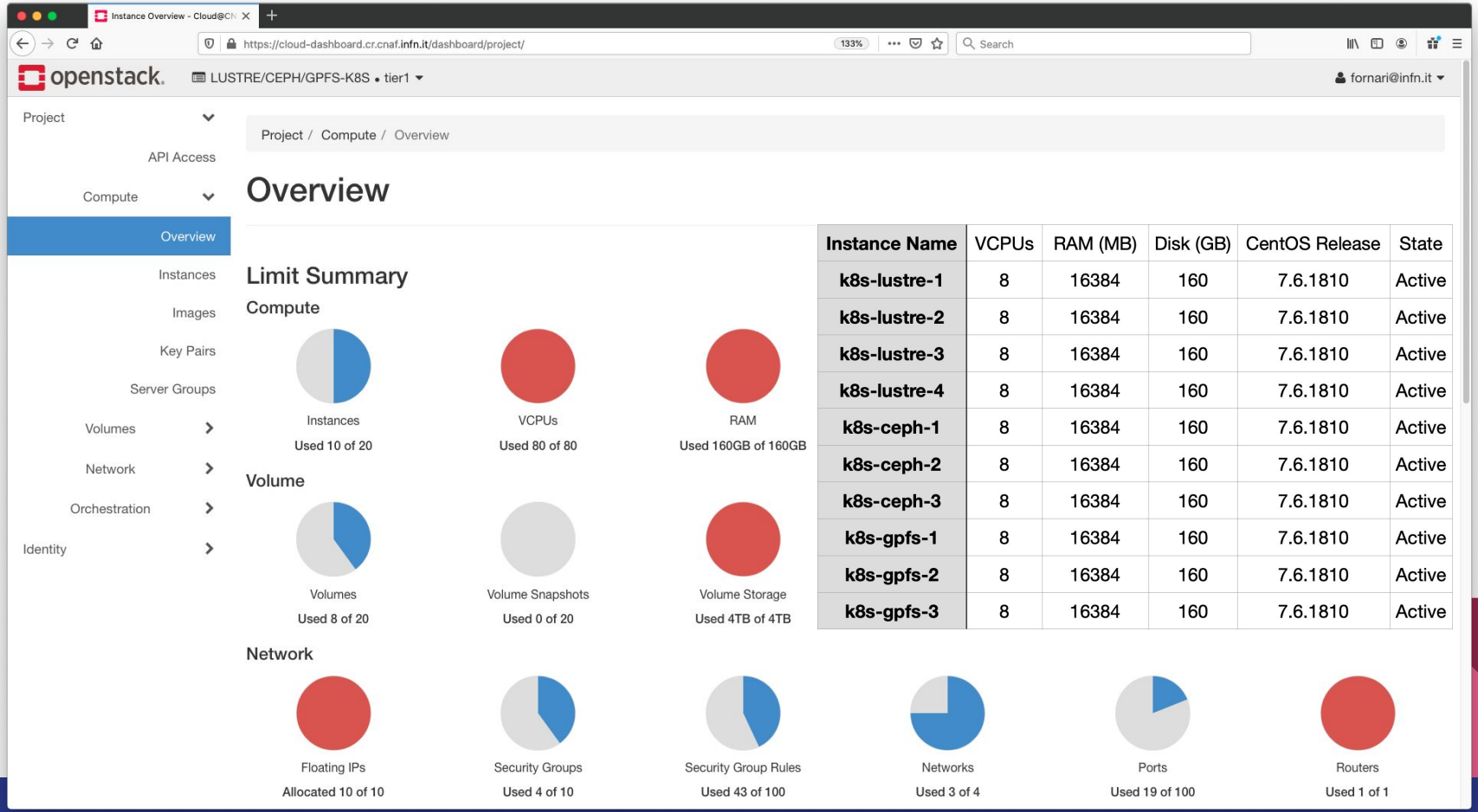
Introduction

- Goal I: demonstrate the feasibility of using Kubernetes and Docker to setup clients capable to access a distributed filesystem from existing clusters and to create clusters based on containerized servers (Part I - proof of concept).
- Three distributed filesystems have been considered:
  - IBM Spectrum Scale (GPFS)
  - Ceph (CephFS)
  - Lustre (on ZFS)
- Goal II: compare performances and stability between different distributed filesystems (Part II - perfomance tests).
- Performance tests have been done with containerized servers setups.

PART I - Proof Of Concept

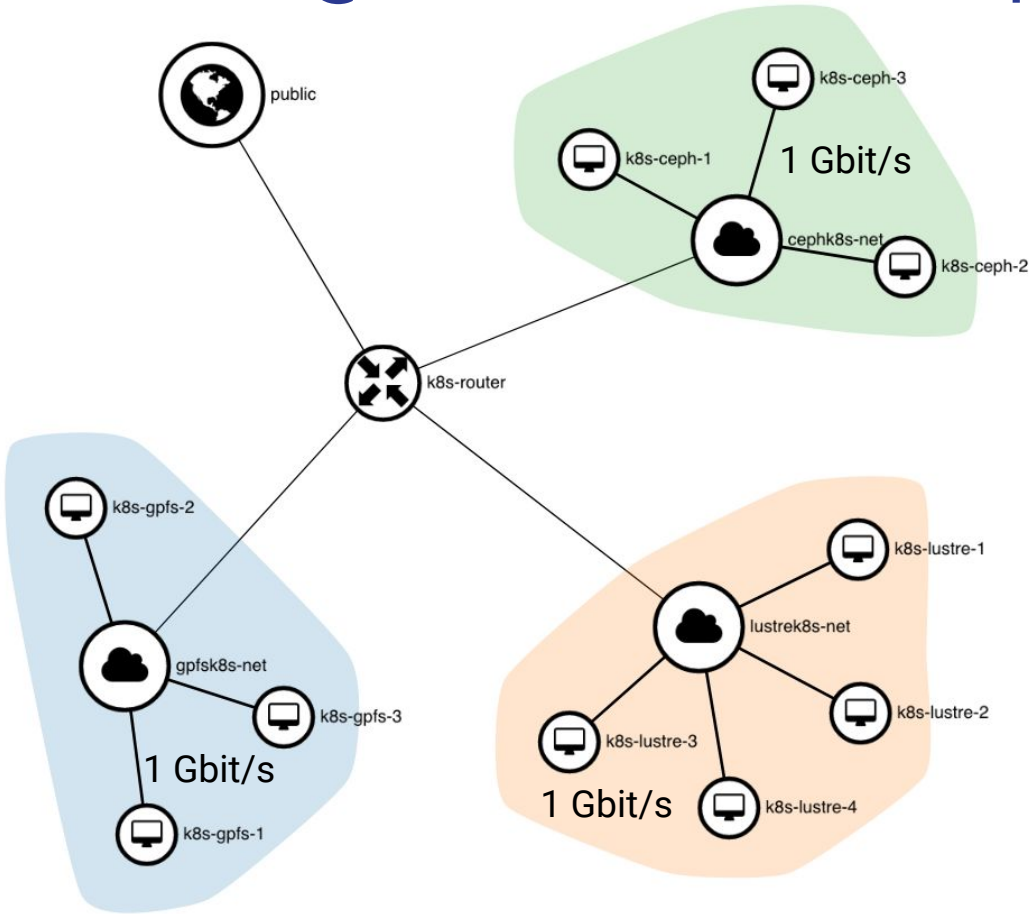
K8s Distributed Filesystems on Openstack Nodes

Openstack Cloud@CNAF Project Overview



Instance Name	VCPUs	RAM (MB)	Disk (GB)	CentOS Release	State
k8s-lustre-1	8	16384	160	7.6.1810	Active
k8s-lustre-2	8	16384	160	7.6.1810	Active
k8s-lustre-3	8	16384	160	7.6.1810	Active
k8s-lustre-4	8	16384	160	7.6.1810	Active
k8s-ceph-1	8	16384	160	7.6.1810	Active
k8s-ceph-2	8	16384	160	7.6.1810	Active
k8s-ceph-3	8	16384	160	7.6.1810	Active
k8s-gpfs-1	8	16384	160	7.6.1810	Active
k8s-gpfs-2	8	16384	160	7.6.1810	Active
k8s-gpfs-3	8	16384	160	7.6.1810	Active

Openstack Cloud@CNAF Network Topology



K8s GPFS Cluster Non-Containerized Server Setup

Replica 2 GPFS Filesystem:

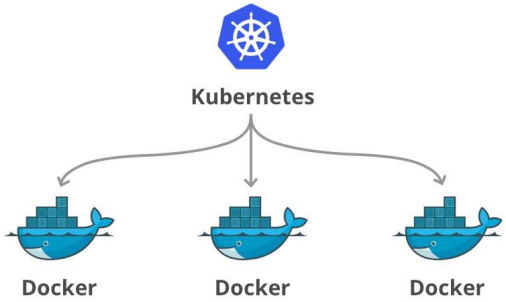
mmcrfs /gpfs/gpfskube gpfskube -F nsd -m 2 -r 2

GPFS Cluster with 2 NSDs:

File system	Disk name	NSD servers
gpfskube	vdb1	k8s-gpfs-1.novalocal
gpfskube	vdb2	k8s-gpfs-2.novalocal

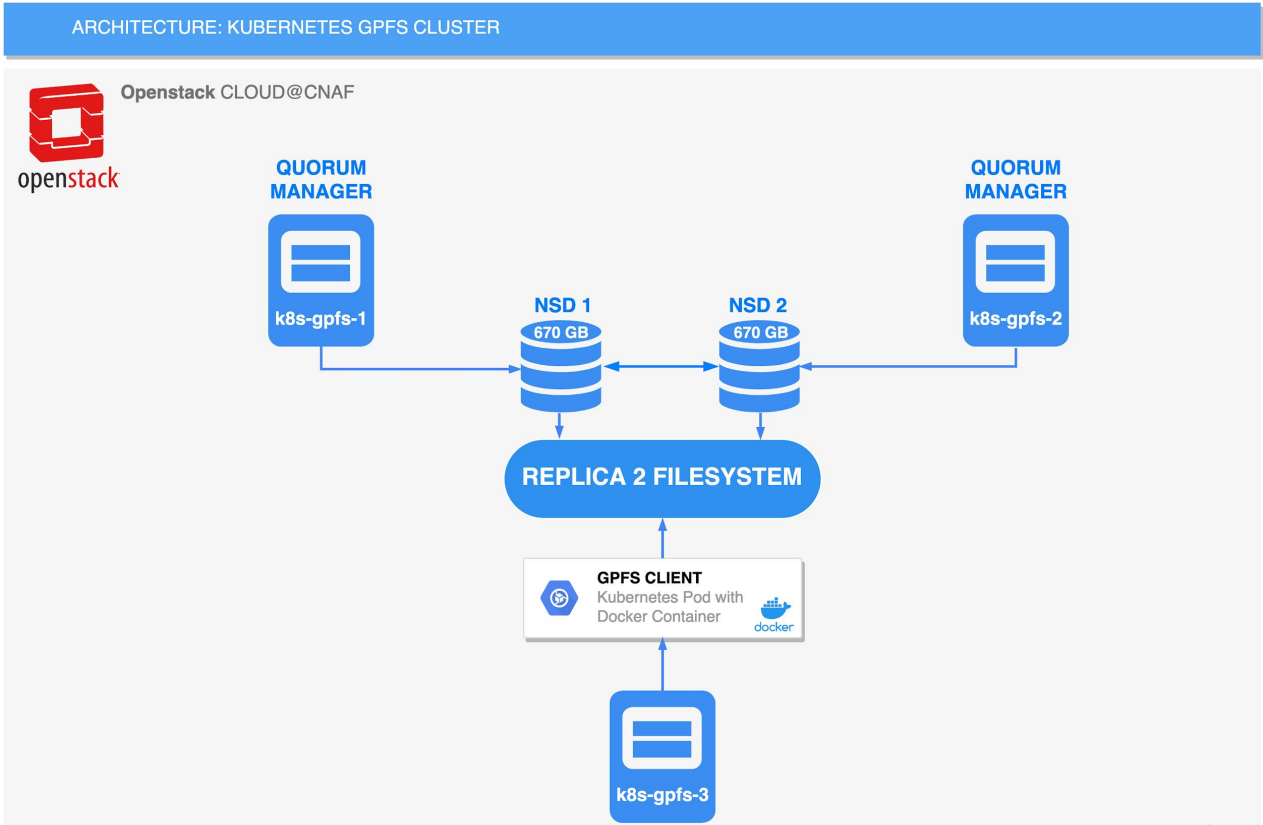
Orchestrator: **Kubernetes 1.20**

Virtualizer: **Docker 20.10**



Filesystem: **IBM GPFS 5.0.5-2**

IBM Spectrum Scale



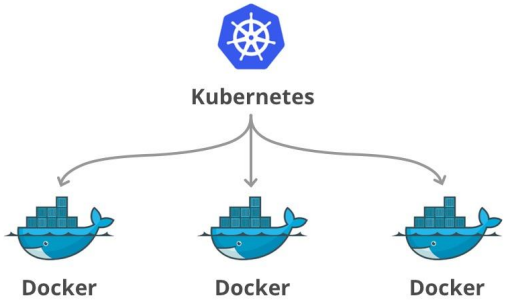
K8s GPFS Cluster Containerized Server Setup

Replica 2 GPFS Filesystem:
mmcrfs /gpfs/gpfskube gpfskube -F nsd -m 2 -r 2

GPFS Cluster with 2 NSDs:

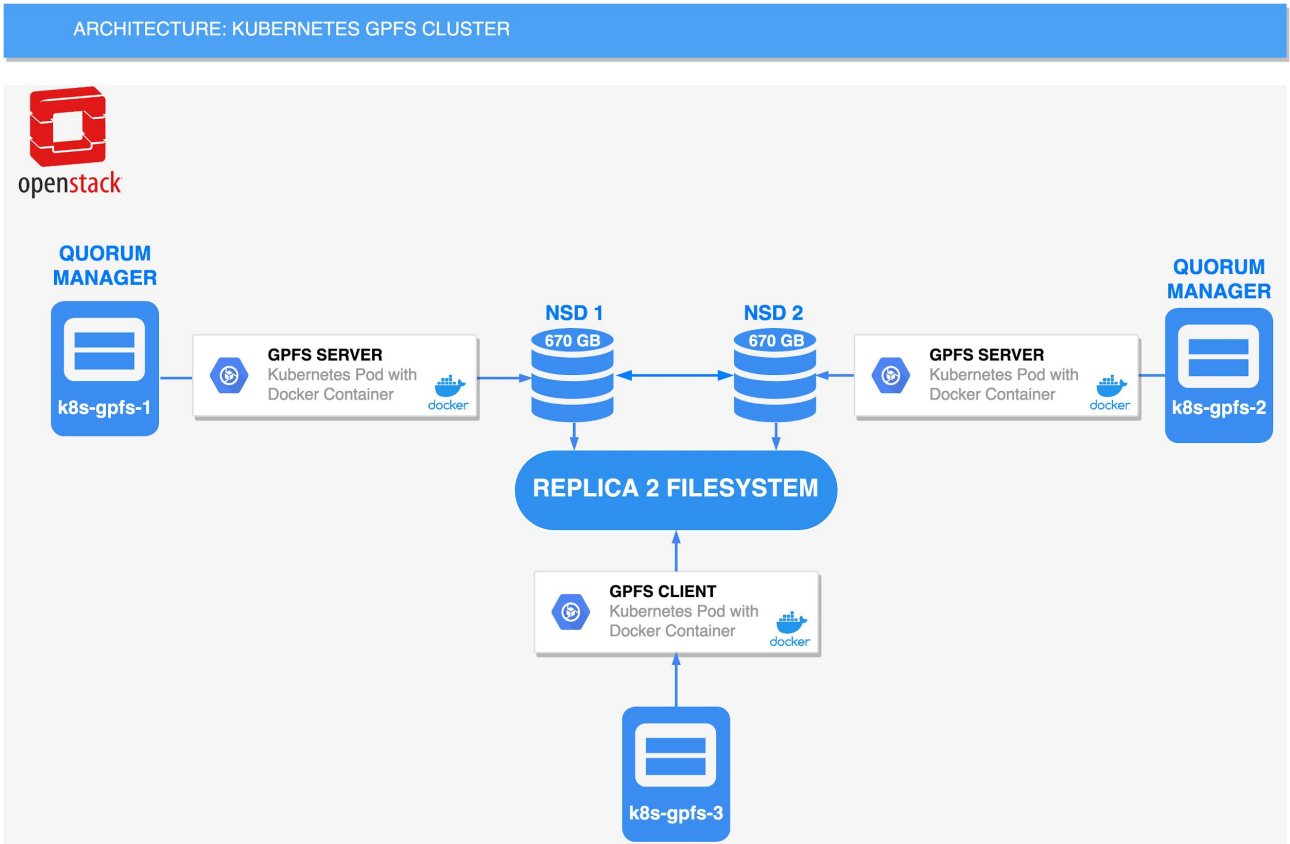
File system	Disk name	NSD servers
gpfskube	vdb1	k8s-gpfs-1.novalocal
gpfskube	vdb2	k8s-gpfs-2.novalocal

Orchestrator: **Kubernetes 1.20**
Virtualizer: **Docker 20.10**



Filesystem: **IBM GPFS 5.0.5-2**

IBM Spectrum Scale



K8s GPFS Pod Configuration File

```
[fornari@farm-ops k8s-gpfs]$ cat k8s/server/pod.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: gpfs-server-1
  namespace: gpfs
  labels:
    app: gpfs-server
spec:
  selector:
    matchLabels:
      app: gpfs-server
  serviceName: gpfs-server-1
  replicas: 1
  template:
    metadata:
      name: gpfs-server-1
      labels:
        app: gpfs-server
    spec:
      hostNetwork: true
      nodeName: k8s-gpfs-1.novalocal
      containers:
      - name: gpfs-server-1
        image: centos/systemd
        command: [ "/bin/bash", "-c", "--" ]
        args: [ "./scripts/start-container.sh" ]
        securityContext:
          privileged: true
        volumeMounts:
          - name: start-container
            mountPath: /scripts/start-container.sh
            subPath: start-container.sh
          - name: init-container
            mountPath: /scripts/init-container.sh
            subPath: init-container.sh
          - name: gendir
            mountPath: /var/mmfs/gen
      volumes:
      - name: start-container
        configMap:
          name: start-container
          defaultMode: 0755
      - name: init-container
        configMap:
          name: init-container
          defaultMode: 0755
      - name: gendir
        hostPath:
          path: /var/mmfs/gen
          type: Directory
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: start-container
data:
  start-container.sh: |
    #!/bin/bash
    # script set in background
    setsid /scripts/init-container.sh &
    # run systemd
    exec /usr/sbin/init
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: init-container
data:
  init-container.sh: |
    #!/bin/bash
    [install kernel, kernel-headers, kernel-devel packages]
    [install openssh server and client packages]
    [install iozone]
    [install ibm-spectrum-scale packages]
    [configure and start openssh server]
```

K8s CEPH Cluster Non-Containerized Server Setup

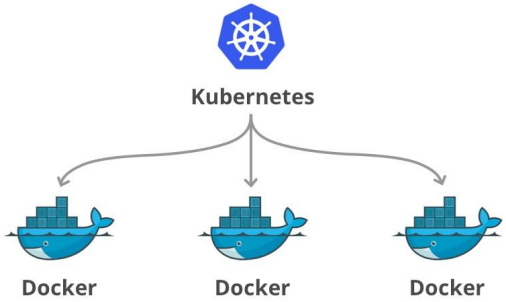
CephFS cluster:

- 3 Monitors, 3 Managers
- 2 OSDs
- 1 Metadata Server

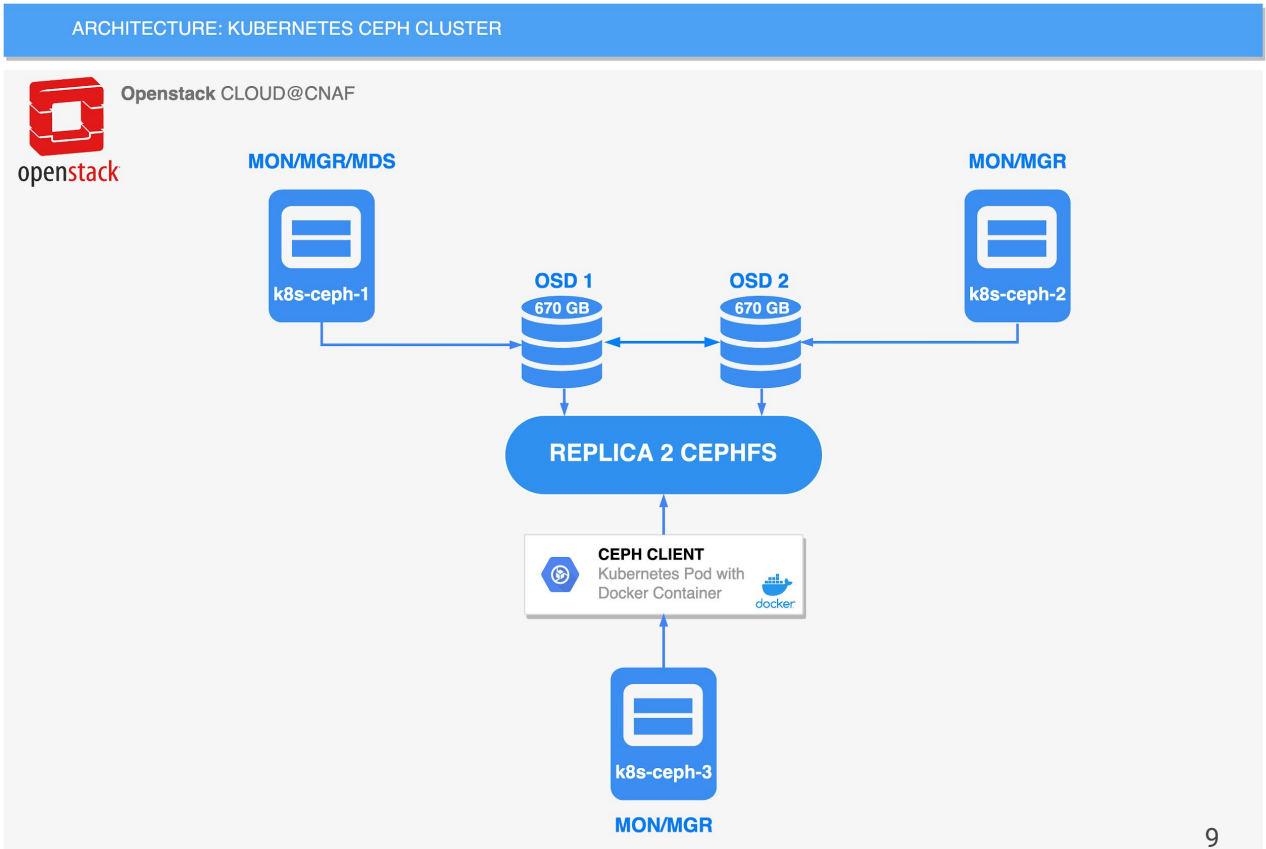
Installed with **Ansible 2.8**

Orchestrator: **Kubernetes 1.20**

Virtualizer: **Docker 20.10**



Filesystem: **Ceph 14.2.10 (nautilus)**



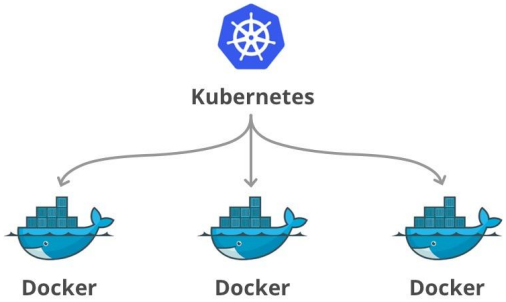
K8s CEPH Cluster Containerized Server Setup

CephFS cluster:

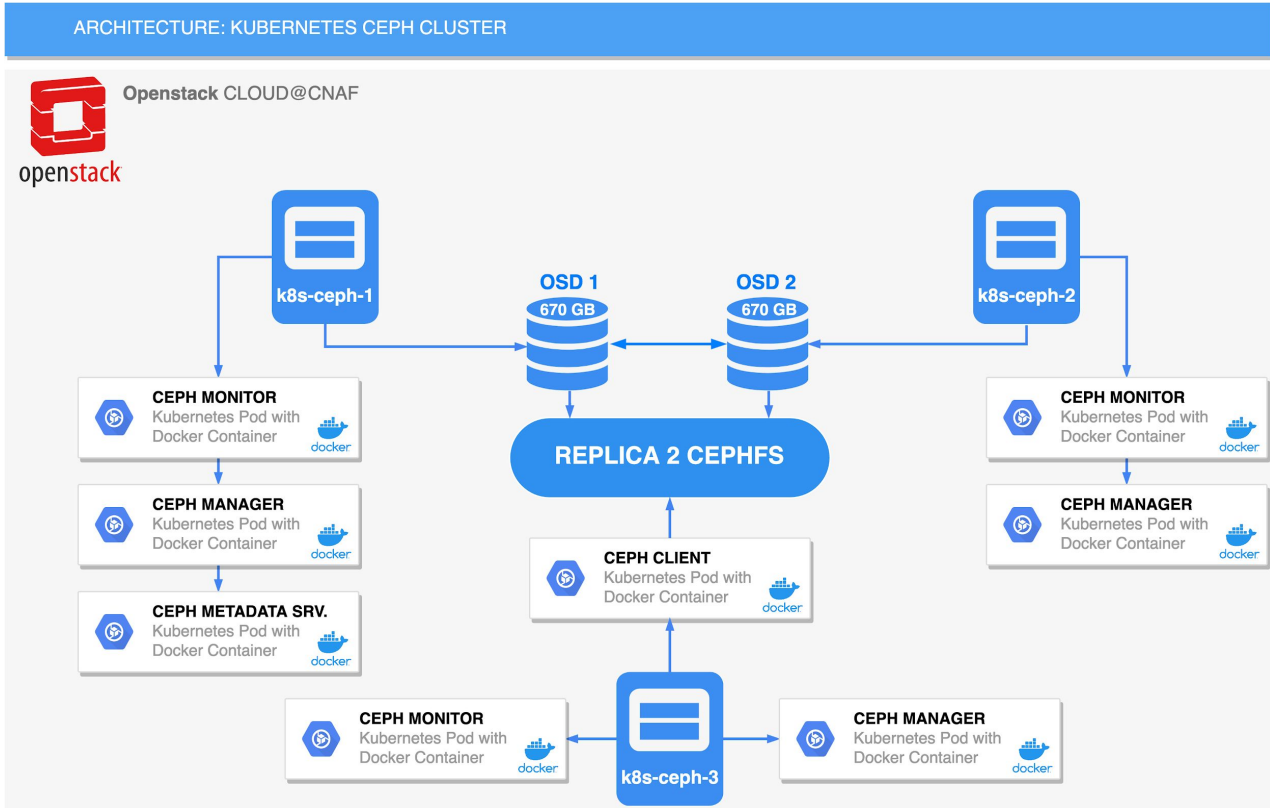
- 3 Monitors, 3 Managers
- 2 OSDs
- 1 Metadata Server

Containerized with **Docker 20.10.**

Orchestrated with specific **Kubernetes (1.20) Helm Charts.**



Filesystem: **Ceph 14.2.10 (nautilus)**



K8s Helm Charts for CEPH - GitHub Project

- Link to GitHub Project: https://github.com/ffornari90/ceph-helm
- Based on Ceph Nautilus
- Uses a Kubernetes operator for the OSDs
- Uses ceph-volume and only supports BlueStore
- Uses CentOS 7 based images from the ceph-container project
- ceph-config-helper image is based on ceph/daemon-base
- Use of Ceph's new centralized configuration management
- Update of CephFS and RBD provisioners (RBD provisioner supports additional RBD image features)
- Unified values.yaml

K8s CEPH Pod Configuration File

```
[fornari@farm-ops kube_ceph]$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: cephfs
spec:
  nodeName: k8s-ceph-3.novalocal
  containers:
  - name: cephfs-rw
    image: centos:7
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "./scripts/init-container.sh" ]
    volumeMounts:
      - mountPath: /scripts/init-container.sh
        name: init-container
        subPath: init-container.sh
      - mountPath: "/mnt/cephfs"
        name: cephfs
  volumes:
    - name: init-container
      configMap:
        name: init-container
        defaultMode: 0755
```

```
  - name: cephfs
    cephfs:
      monitors:
      - k8s-ceph-1.novalocal:6789
      user: admin
      secretFile: "/root/asecret"
      readOnly: false
```

Secret key generated with the command:
ceph auth get-key client.admin | sudo tee /root/asecret

```
[fornari@farm-ops k8s-ceph]$ cat configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: init-container
data:
  init-container.sh: |
   #!/bin/bash
   [install iozone]
   [install openssh server and client]
   [configure and start openssh server]
```

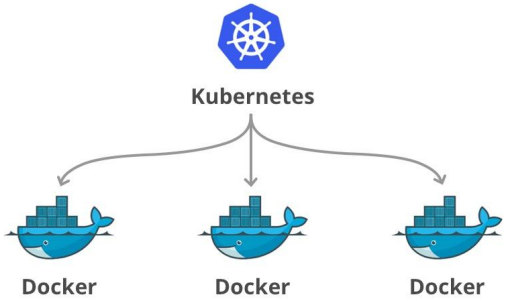
CephFS to be mounted in the container, the equivalent on the host is done with:
sudo mount -t ceph k8s-ceph-1:/ /mnt/cephfs \
-o name=admin,secretfile=/root/asecret

K8s Lustre Cluster Setup

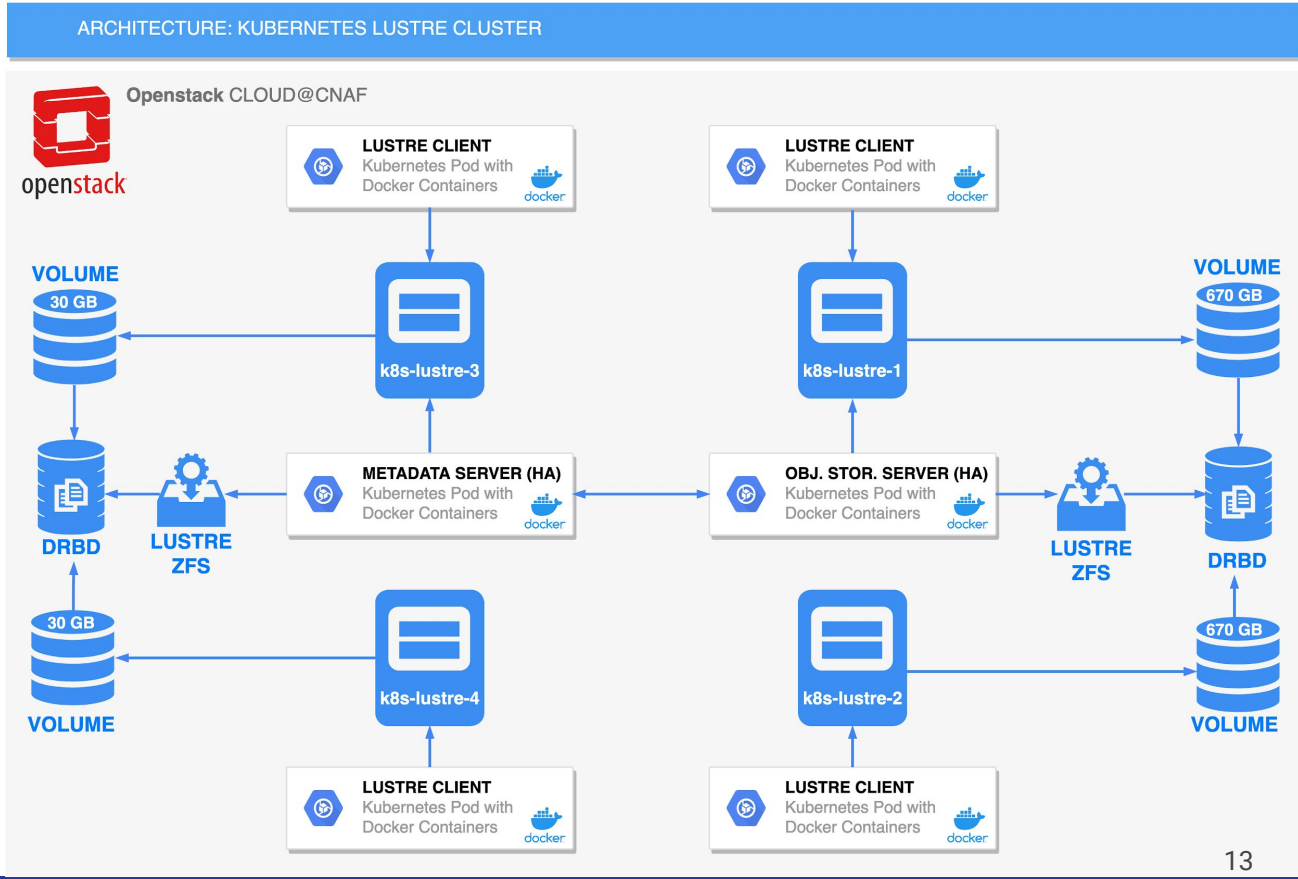
High Availability Lustre Cluster with **DRBD 8.4.11** over **ZFS 0.7.9**:

- 1 HA Metadata Server
- 1 HA Object Storage Server
- 4 Containerized Clients

Orchestrator: **Kubernetes 1.20**
Virtualizer: **Docker 20.10**



Filesystem: **Lustre 2.10.8 (ZFS)**



K8s Lustre GitLab Project

- Link to project: https://baltig.infn.it/fornari/kube-lustre
- K8s YAML files for services configuration (to be edited by end users):
  - kube-lustre-configurator.yaml to select which services to run (servers, clients)
  - kube-lustre-config.yaml to adjust servers/clients configurations, enable/disable HA
- Docker images for Docker containers (orchestrated by Kubernetes):
  - kube-lustre-configurator reads kube-lustre-config.yaml, then generates templates and assign resources to specific Kubernetes nodes
  - lustre makes lustre target, then imports ztool and mounts lustre target
  - lustre-client mounts lustre filesystem
  - lustre-install installs lustre and zfs packages and dkms modules
  - drbd makes and runs distributed replicated block device resource (for HA)
  - drbd-install installs drbd packages and dkms modules

K8s Lustre GitLab Project Development Branch

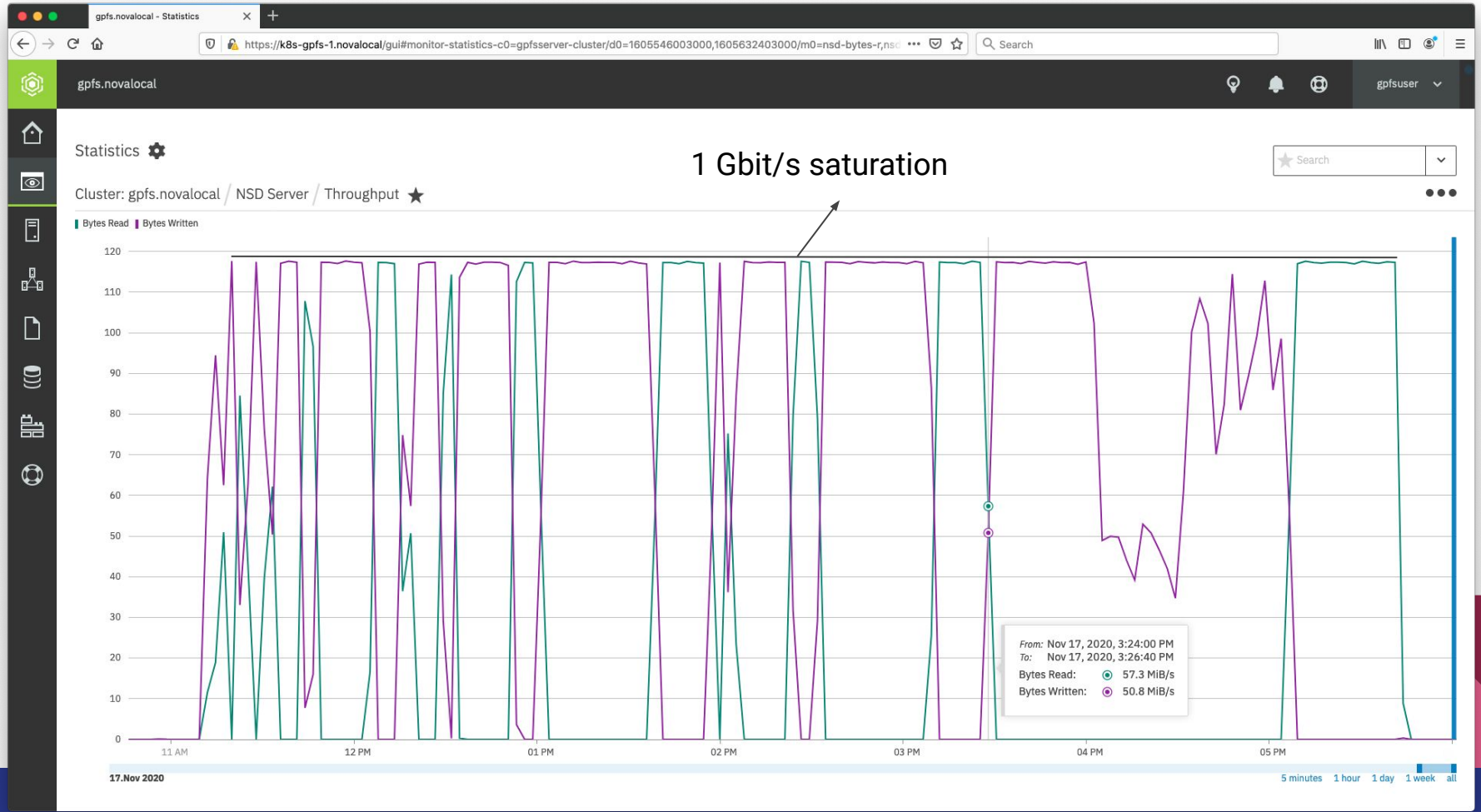
- Created Development Branches for different Lustre Releases with 2 areas:
  - “non-containerized”, contains Kubernetes and Docker configuration files for cluster deployment implying the start of Lustre services via systemd on the host machines
  - “containerized”, includes Kubernetes and Docker configuration files for cluster deployment implying the start of Lustre services via systemd within Docker containers in Kubernetes pods
- As in the GPFS containerized setup, Lustre containerized servers are deployed with Docker containers where the container initialization script is executed in background, after the start of systemd:
  - kernel packages have to be installed inside the containers
  - Lustre kernel modules for each specific kind of server have to be correctly loaded
  - the communication between servers has to be managed via Dynamic Lustre Net Configuration tool (i.e. lnetctl utility)

The Tests: Sequential Read and Write

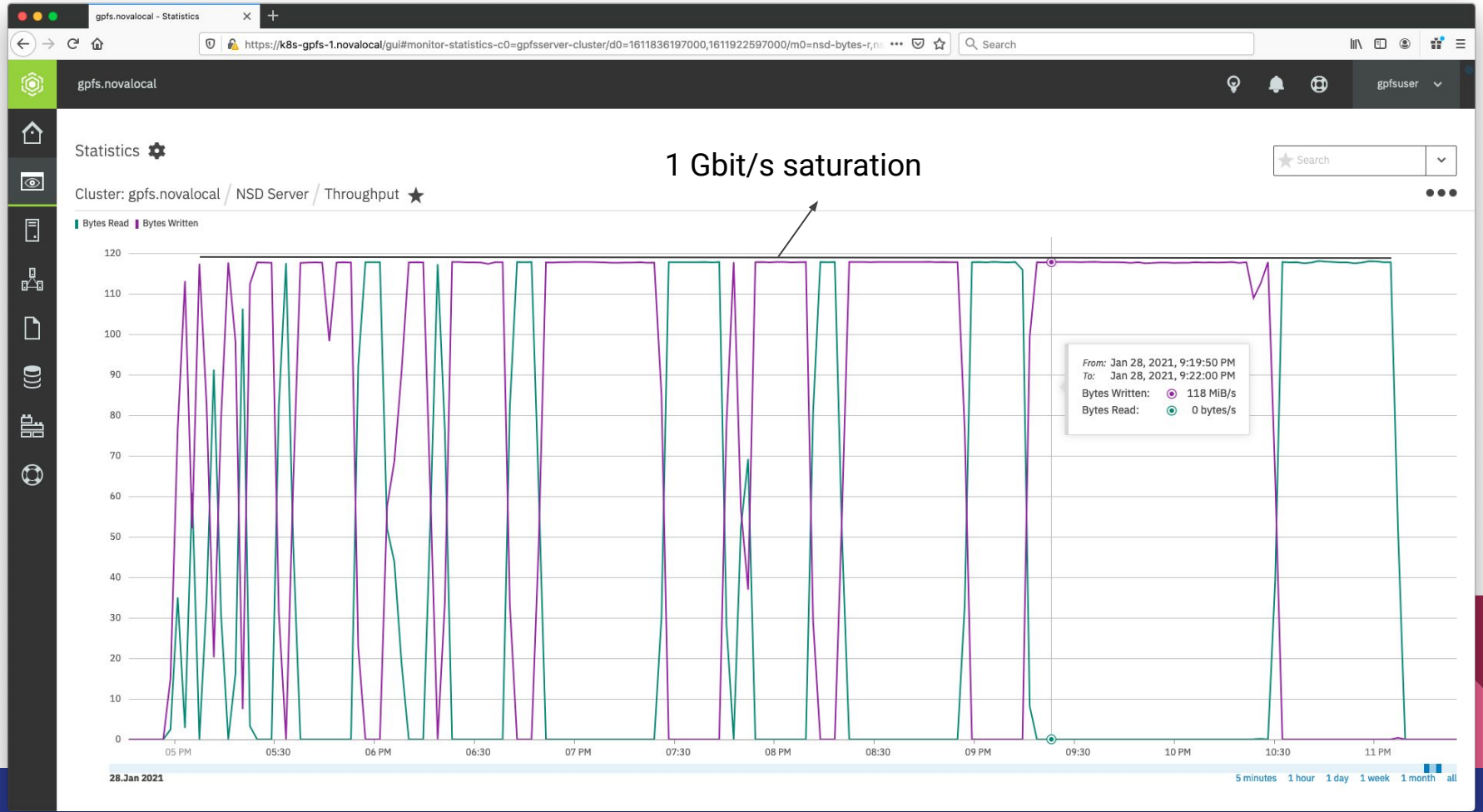
The test script (sequentially writing and reading files of increasing size, from 1 GB to 16 GB, with an increasing number of parallel threads, from 1 to 16, using **iozone**):

```
[fornari@farm-ops ]# cat test_rw.sh
#!/bin/bash
export RSH=ssh
n_th=(1 4 8 16)
size=(1g 4g 8g 16g)
for i in $(seq 0 `expr "${#n_th[@]}" - 1`)
do
   for j in $(seq 0 `expr "${#size[@]}" - 1`)
   do
      iozone -r 1024k -t "${n_th[$i]}" -s "${size[$j]}" -i0 -i1 -+n -w -+m $PWD/iozone_th_files/iozone_"${n_th[$i]}"th > \
          iozone_i0_i1_t"${n_th[$i]}"_s"${size[$j]}".out 2>&1
      for k in $(seq 1 "${n_th[$i]}"); do rm -f $FS_MOUNTPOINT/iozone/test"$k"/*; done
   done
done
```

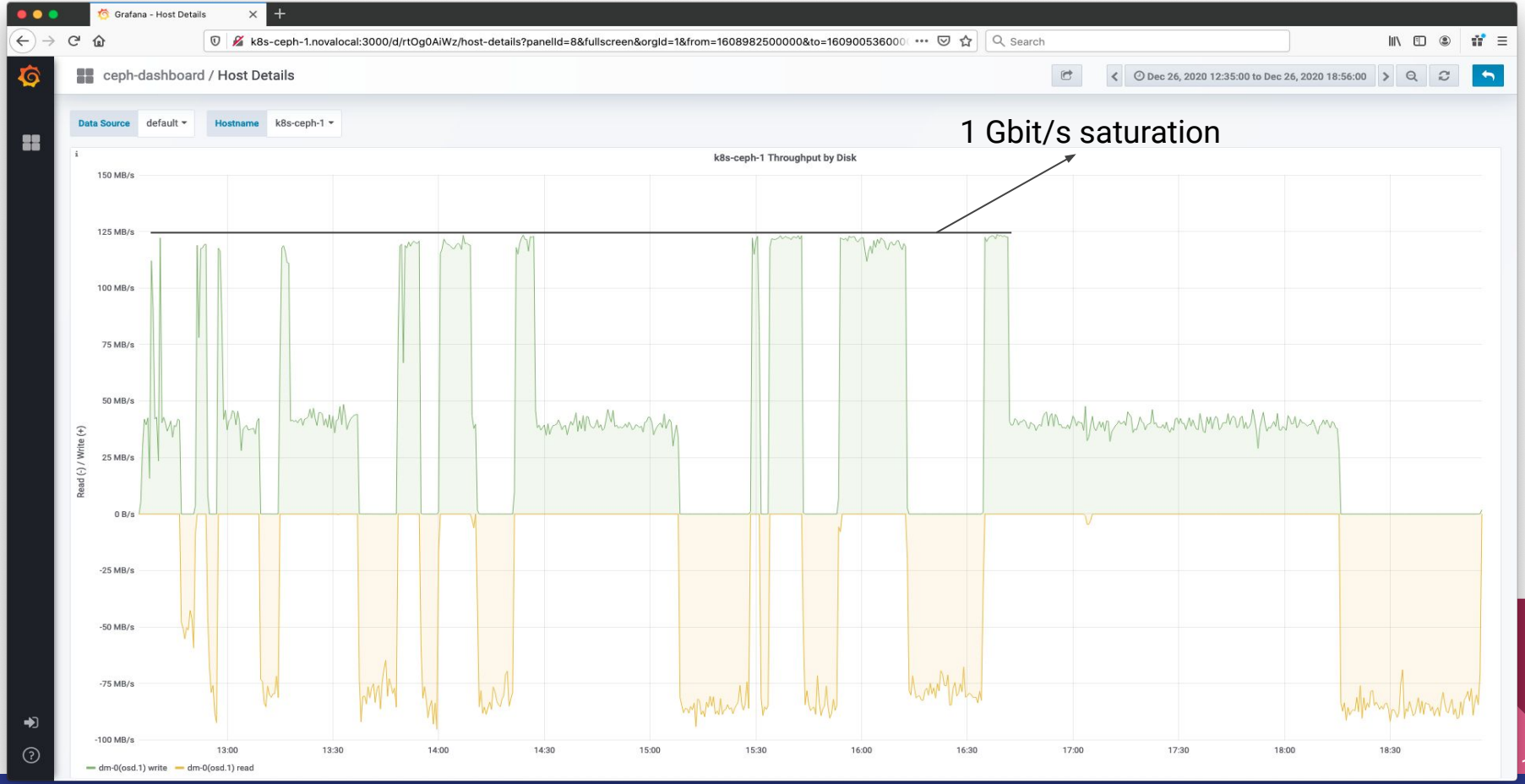
K8s GPFS Cluster Test -> Non-Containerized



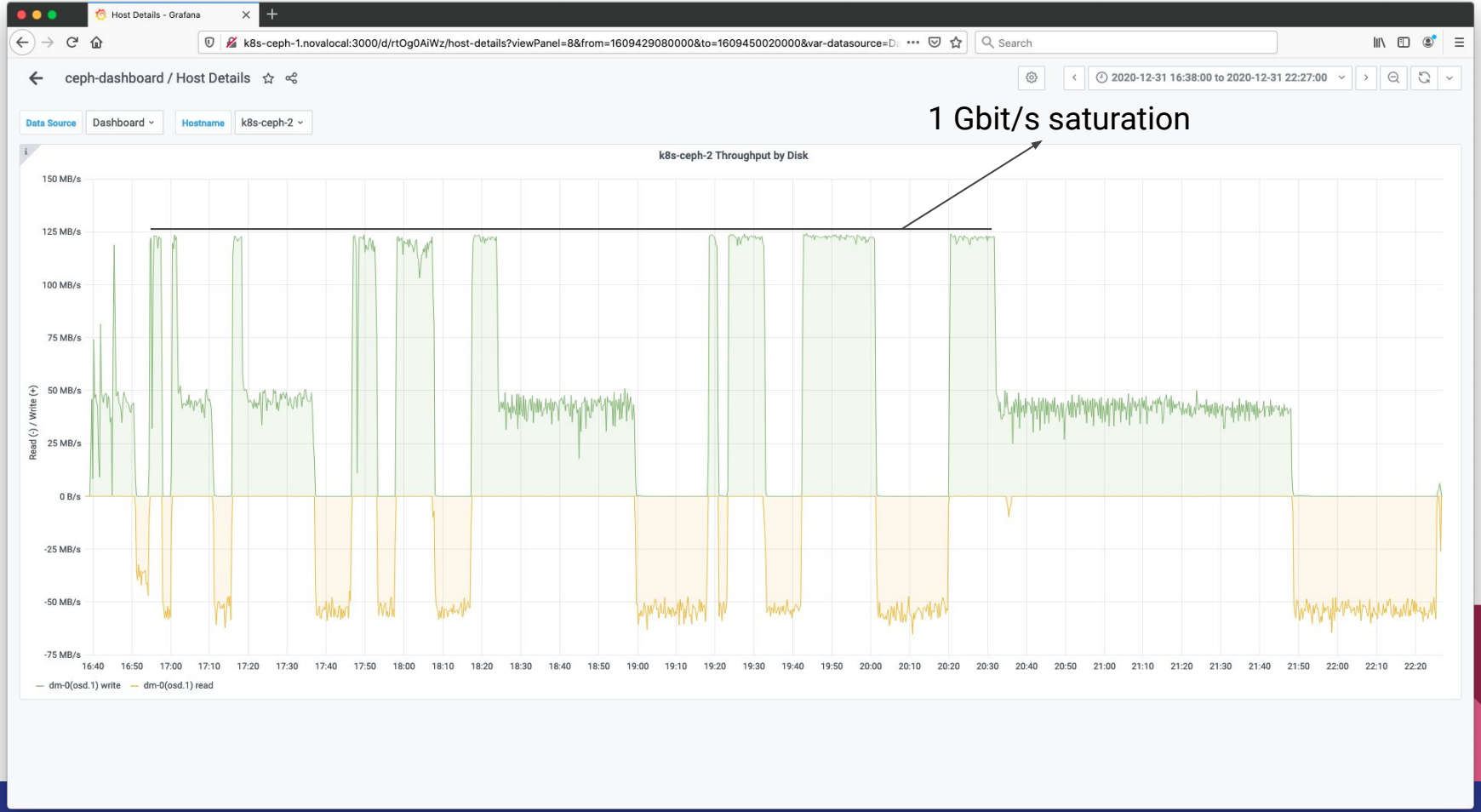
K8s GPFS Cluster Test -> Containerized



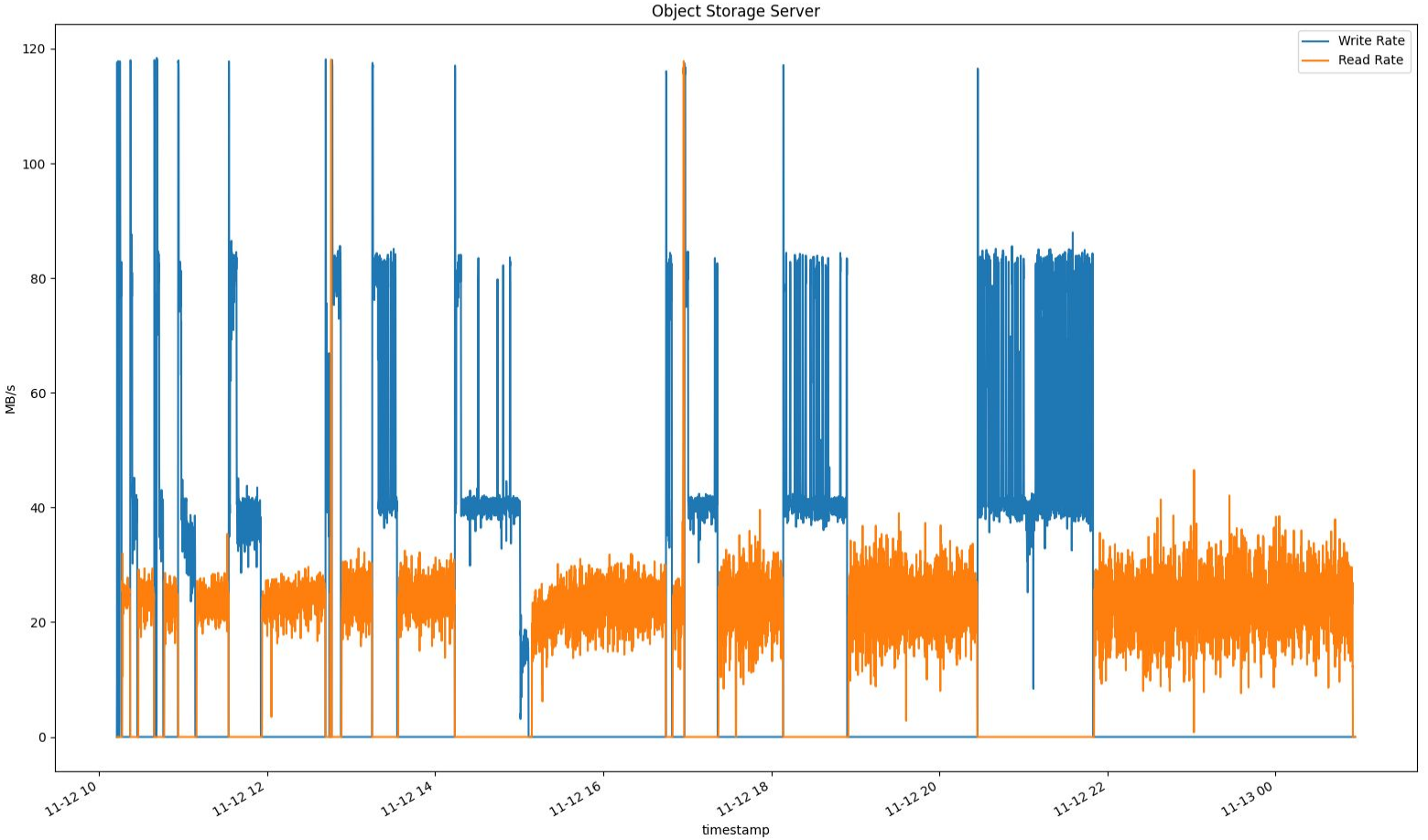
K8s CEPH Cluster Test -> Non-Containerized



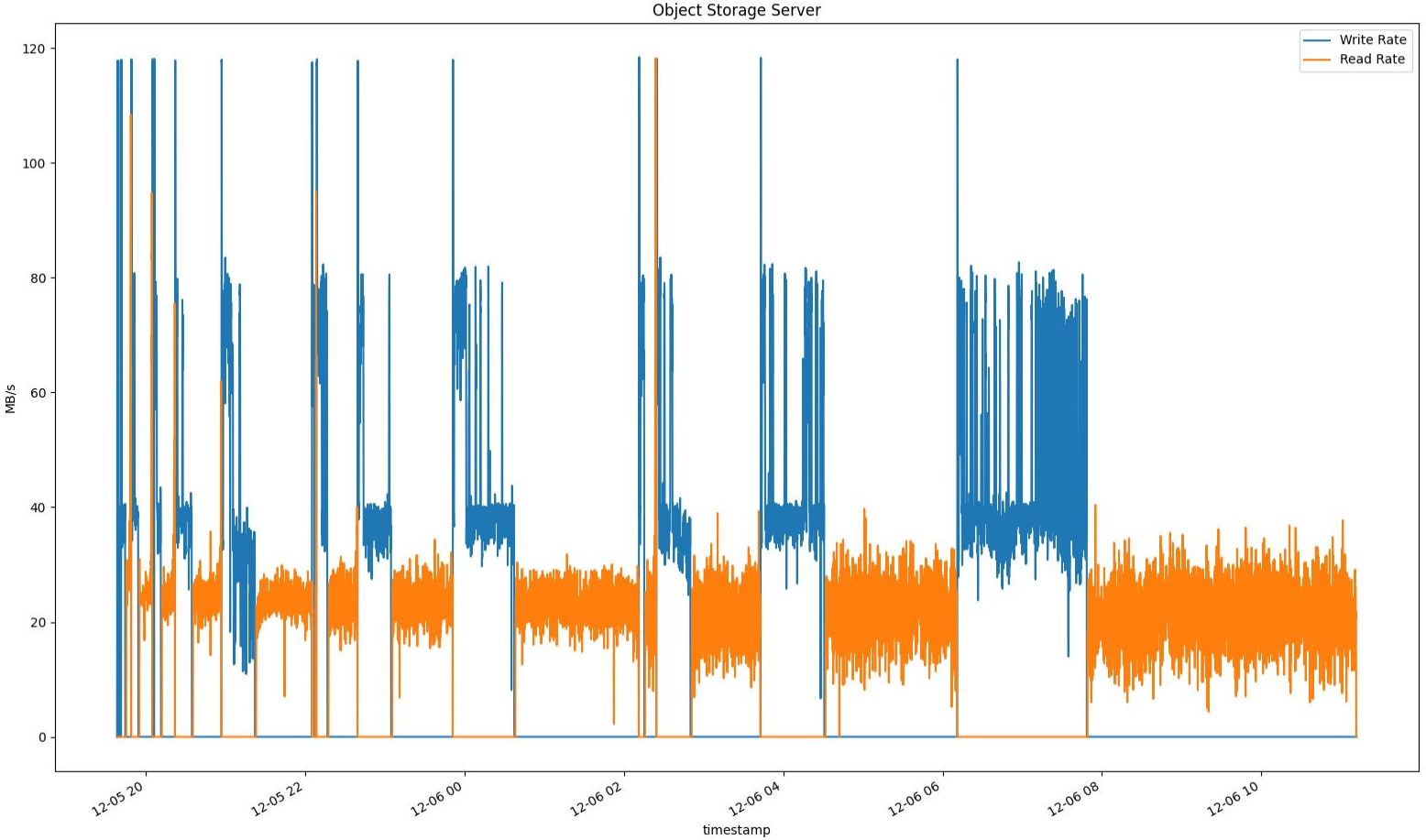
K8s CEPH Cluster Test -> Containerized



K8s Lustre Cluster Test -> Non-Containerized



K8s Lustre Cluster Test -> Containerized



K8s Lustre -> Lustre Fine Tuning

```
#ptlrpc.conf
options ptlrpc ptlrpcd_per_cpt_max=32
ptlrpcd_partner_group_size=32
```

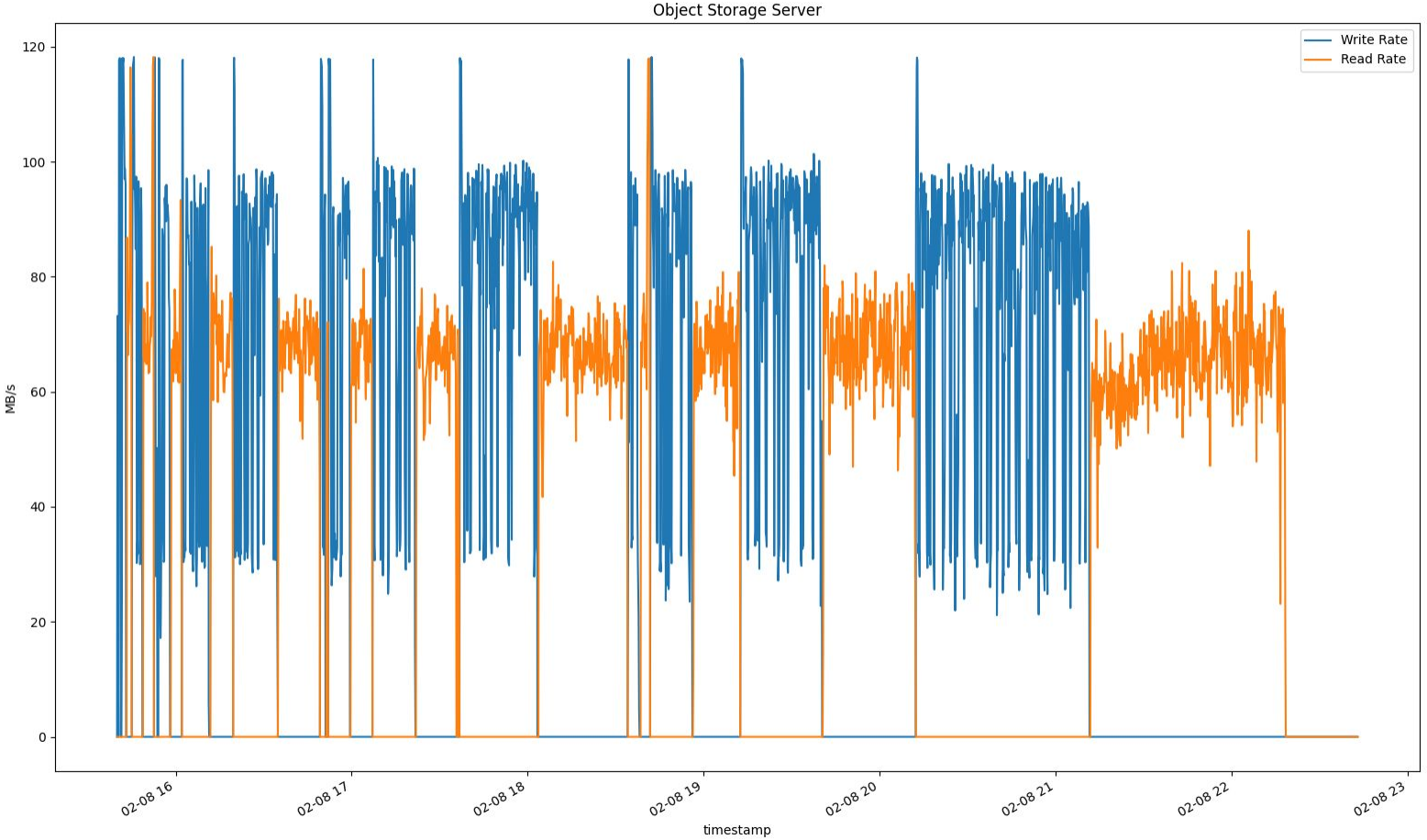
```
#ksocklnd.conf
options ksocklnd sock_timeout=100
credits=2560 peer_credits=63
```

```
lctl set_param -P osc.*.checksums=0
lctl set_param -P timeout=600
lctl set_param -P at_min=250
lctl set_param -P at_max=600
lctl set_param -P ldlm.namespaces.*.lru_size=2000
lctl set_param -P osc.*.max_rpcs_in_flight=64
lctl set_param -P osc.*.max_dirty_mb=1024
lctl set_param -P llite.*.max_read_ahead_mb=1024
lctl set_param -P llite.*.max_cached_mb=81920
lctl set_param -P llite.*.max_read_ahead_per_file_mb=1024
lctl set_param -P subsystem_debug=0
```

```
zfs set mountpoint=none fsname
zfs set sync=disabled fsname
zfs set atime=off fsname
zfs set redundant_metadata=most fsname
zfs set xattr=sa fsname
zfs set recordsize=1M fsname
```

```
#zfs.conf
options zfs zfs_prefetch_disable=1
options zfs zfs_txg_history=120
options zfs metaslab_debug_unload=1
options zfs zfs_vdev_scheduler=deadline
options zfs
zfs_vdev_async_write_active_min_dirty_percent=20
options zfs zfs_vdev_scrub_min_active=48
options zfs zfs_vdev_scrub_max_active=128
options zfs zfs_vdev_sync_write_min_active=8
options zfs zfs_vdev_sync_write_max_active=32
options zfs zfs_vdev_sync_read_min_active=8
options zfs zfs_vdev_sync_read_max_active=32
options zfs zfs_vdev_async_read_min_active=8
options zfs zfs_vdev_async_read_max_active=32
options zfs zfs_top_maxinflight=320
options zfs zfs_txg_timeout=30
options zfs zfs_dirty_data_max_percent=40
options zfs zfs_vdev_async_write_min_active=8
options zfs zfs_vdev_async_write_max_active=32
```

K8s Lustre Containerized Test after Tuning



Part I: Summary

- Distributed filesystem deployment using container orchestration has been proven to be feasible.
- GPFS and Ceph can saturate the Ethernet link bandwidth without particular configuration efforts.
- Lustre required configuration adjustments tuning up caches and metadata access in order to gain good performances.
- At a proof of concept level, no significant performance differences emerged between containerized and non-containerized server setups.

PART II - Perfomance Tests

K8s Distributed Filesystems on Bare Metal Servers

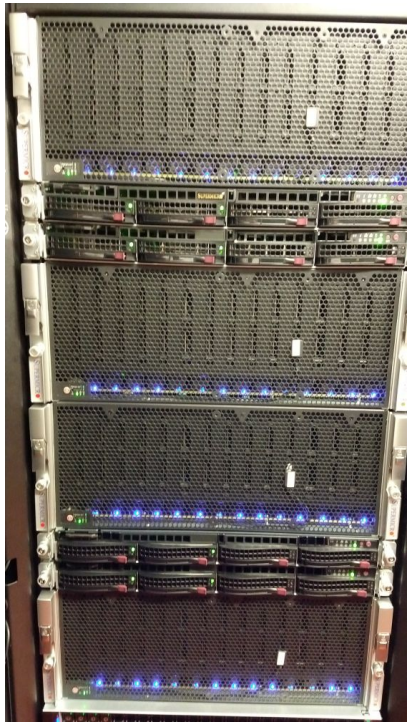
Testbed setup and components

- hosts: cs-00[1-8].cr.cnaf.infn.it
- 16 cores Intel(R) Xeon(R) CPU E5-2609 v4 @ 1.70GHz
- Memory not homogenous
  - cs-001 RAM: 144785 MB
  - cs-002 RAM: 144785 MB
  - cs-003 RAM: 128658 MB
  - cs-004 RAM: 128658 MB
  - cs-005 RAM: 257516 MB
  - cs-006 RAM: 257516 MB
  - cs-007 RAM: 257516 MB
  - cs-008 RAM: 257516 MB

used as clients

used as servers

- Network : Bonding 2 X 10Gbit/s
- four 60 disks JBODs (SAS) → Hardware hacked 8 X 30 disks
- 240 X 8 TB disks → 1920 TB RAW Space
- **3 X 8 TB disks per server → 1 GPFS, 1 Ceph, 1 Lustre**
- Operating System: CentOS 7.7.1908



K8s GPFS Cluster Containerized Server Setup

Replica 2 GPFS Filesystem:

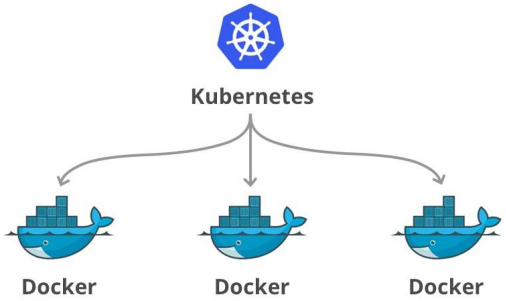
mmcrfs /gpfs/gpfskube gpfskube -F nsd -m 2 -r 2

GPFS Cluster with 4 NSDs:

File system	Disk name	NSD servers
gpfskube	cs005	cs-005
gpfskube	cs006	cs-006
gpfskube	cs007	cs-007
gpfskube	cs008	cs-008

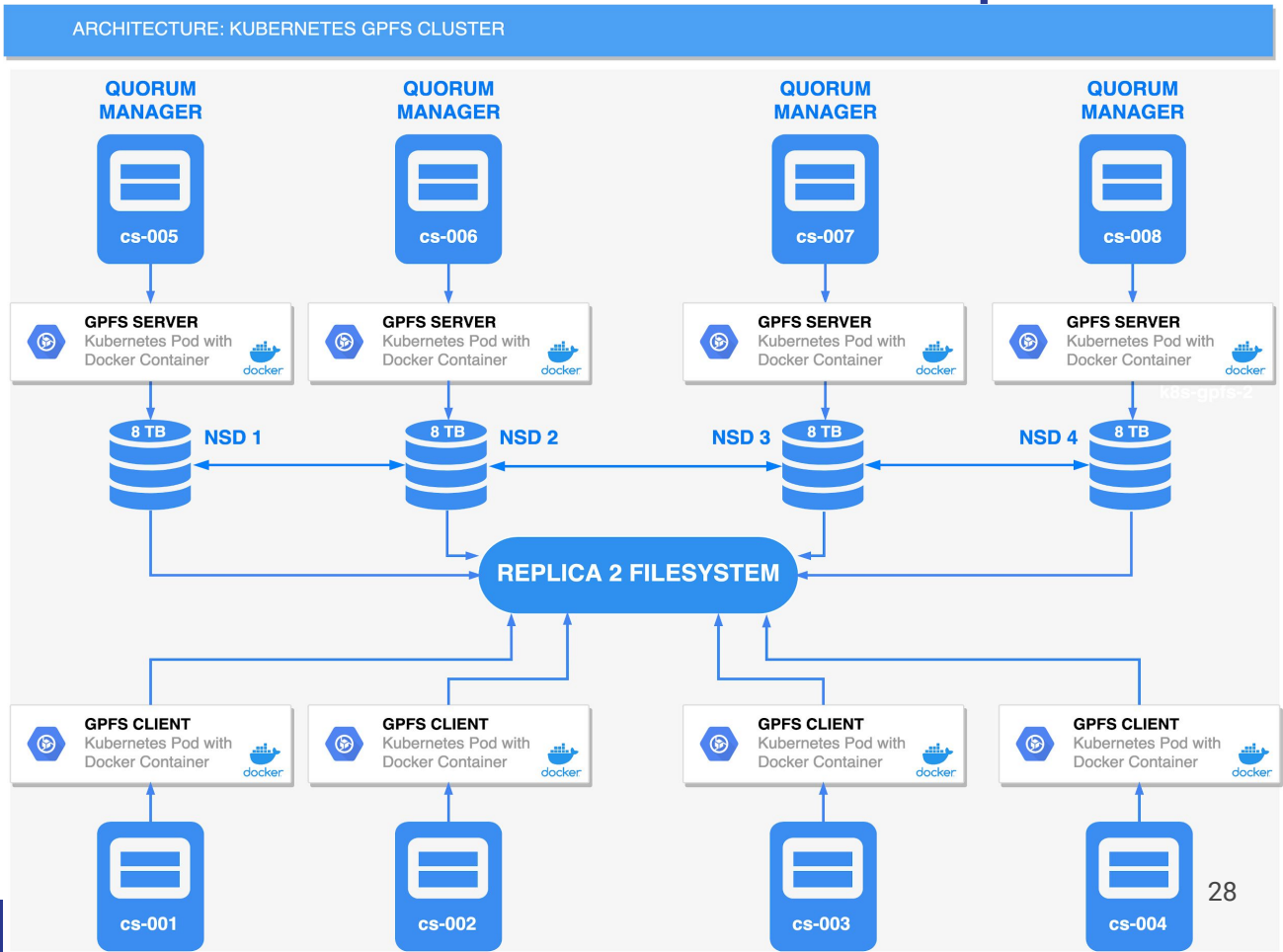
Orchestrator: **Kubernetes 1.20**

Virtualizer: **Docker 20.10**



Filesystem: **IBM GPFS 5.0.5-2**

IBM Spectrum Scale



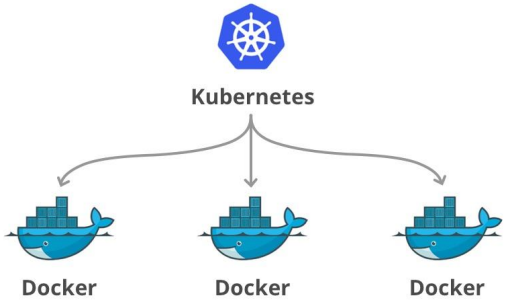
K8s CEPH Cluster Containerized Server Setup

CephFS cluster:

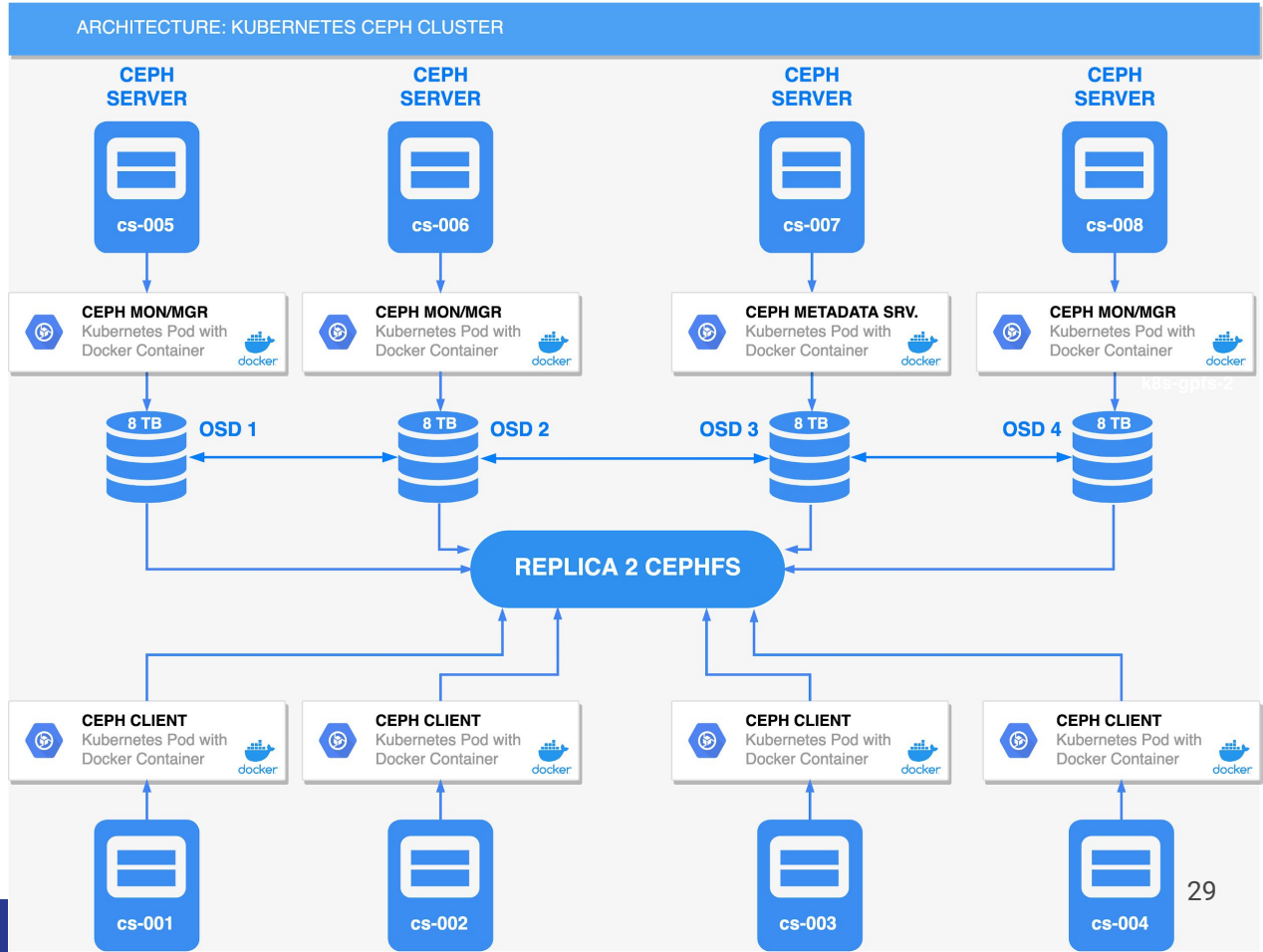
- 3 Monitors, 3 Managers
- 4 OSDs
- 1 Metadata Server

Containerized with **Docker 20.10.**

Orchestrated with specific **Kubernetes (1.20) Helm Charts.**



Filesystem: **Ceph 14.2.10 (nautilus)**

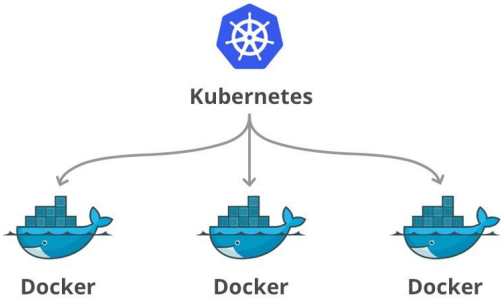


K8s Lustre Cluster Setup

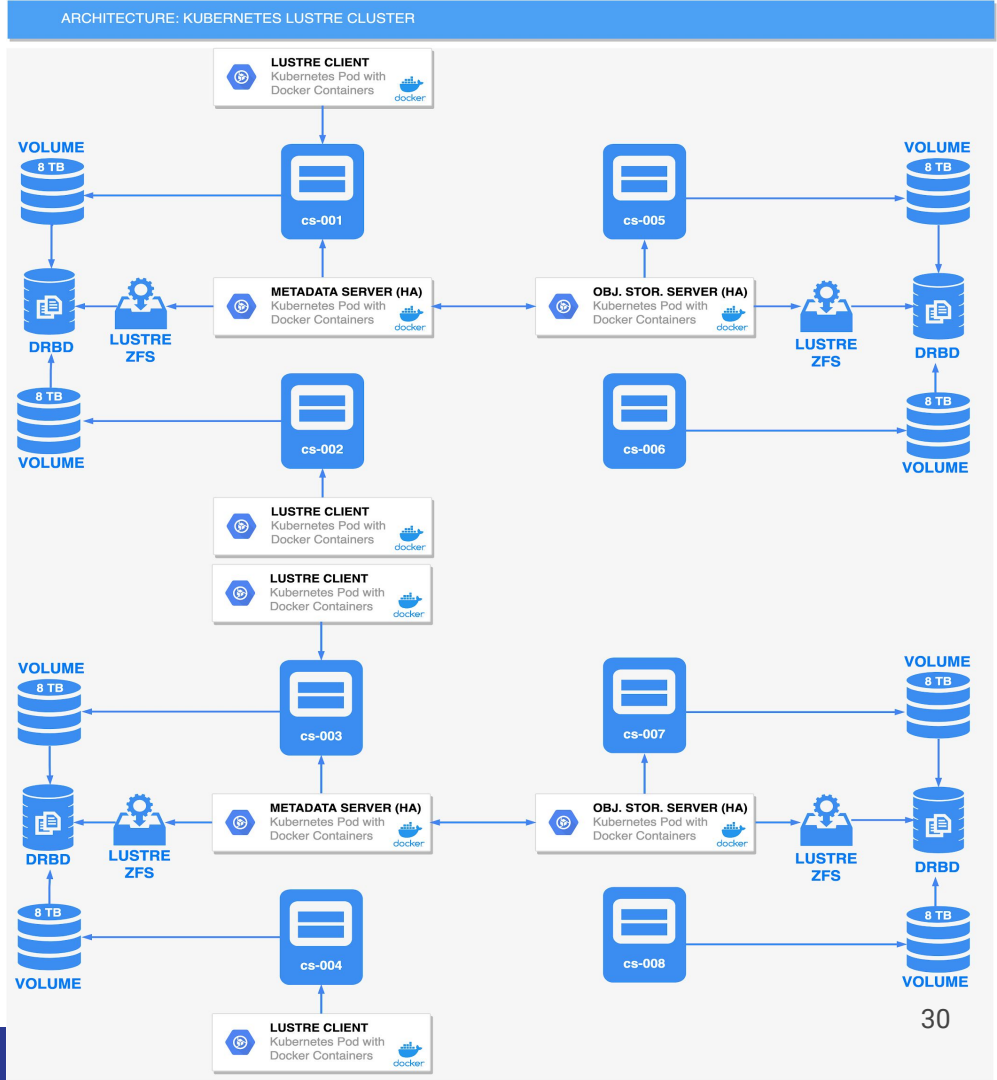
High Availability Lustre Cluster over **ZFS 0.7.13** with **DRBD 8.4.11**:

- 2 HA Metadata Server
- 2 HA Object Storage Server
- 4 Containerized Clients

Orchestrator: **Kubernetes 1.20**
Virtualizer: **Docker 20.10**



Filesystem: **Lustre 2.12.4 (ZFS)**



The Tests: Sequential Read and Write

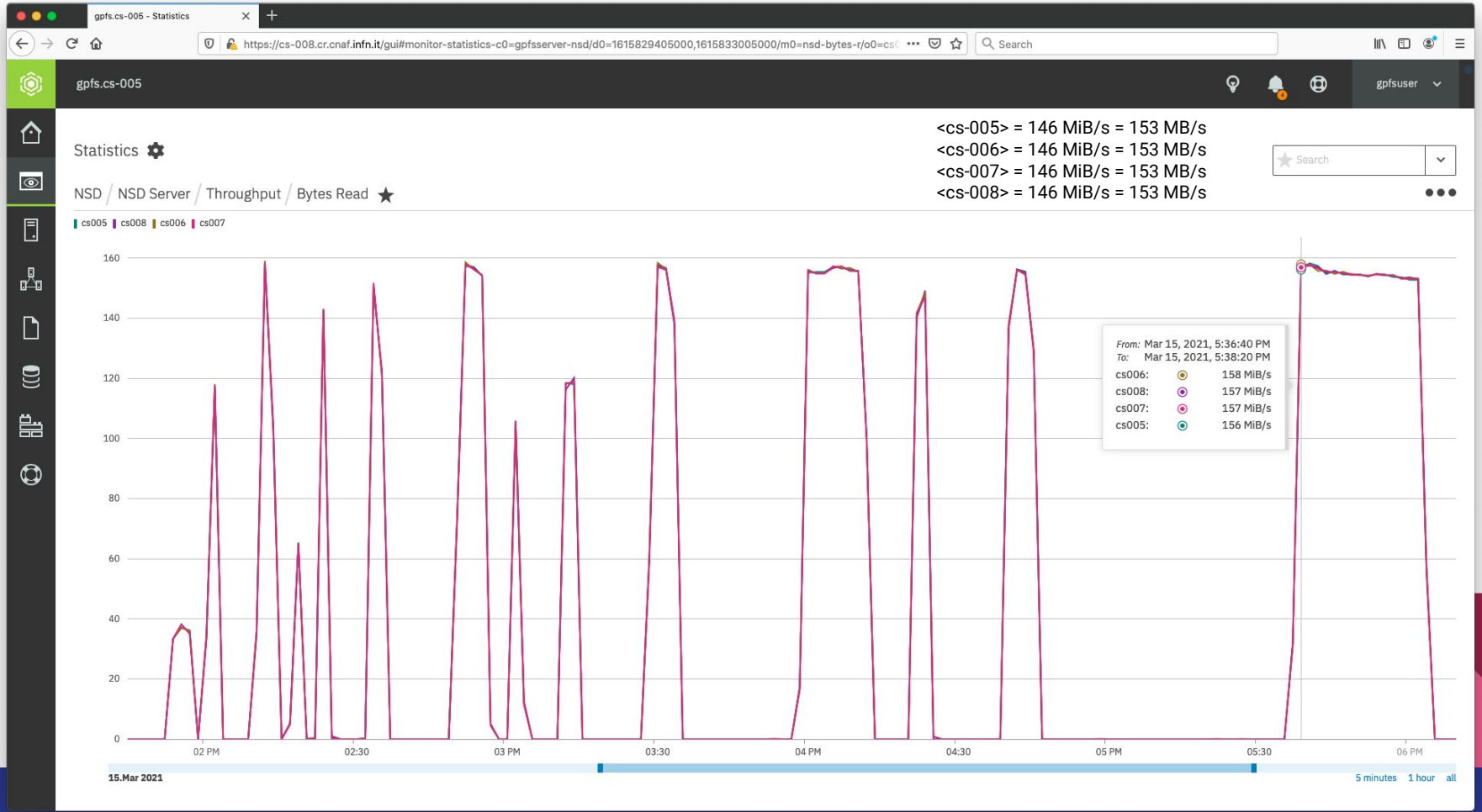
The test script (sequentially writing and reading files of increasing size, from 4 GB to 32 GB, with an increasing number of parallel threads, from 4 to 32, using **iozone**):

```
[fornari@farm-ops ]# cat test_rw.sh
#!/bin/bash
export RSH=ssh
n_th=(4 8 16 32)
size=(4g 8g 16g 32g)
for i in $(seq 0 `expr "${#n_th[@]}" - 1`)
do
   for j in $(seq 0 `expr "${#size[@]}" - 1`)
   do
      iozone -r 1024k -t "${n_th[$i]}" -s "${size[$j]}" -i0 -i1 -+n -w -+m $PWD/iozone_th_files/iozone_"${n_th[$i]}"th > \
          iozone_i0_i1_t"${n_th[$i]}"_s"${size[$j]}".out 2>&1
      for k in $(seq 1 "${n_th[$i]}"); do rm -f $FS_MOUNTPOINT/iozone/test"$k"/*; done
   done
done
```

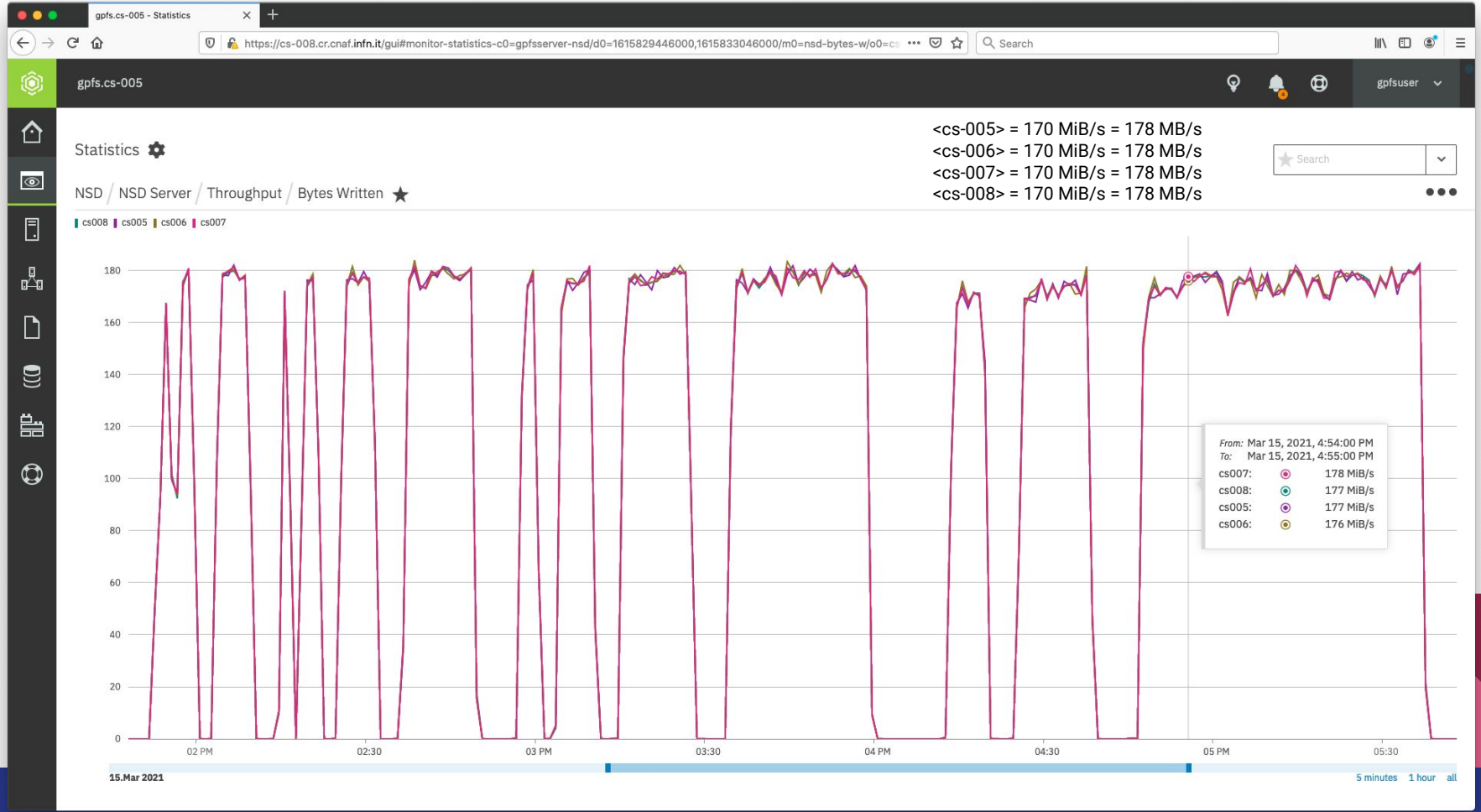
```
[fornari@farm-ops iozone_th_files]$ cat iozone_4th
cs-001 $FS_MOUNTPOINT/iozone/test1 /usr/bin/iozone
cs-002 $FS_MOUNTPOINT/iozone/test2 /usr/bin/iozone
cs-003 $FS_MOUNTPOINT/iozone/test3 /usr/bin/iozone
cs-004 $FS_MOUNTPOINT/iozone/test4 /usr/bin/iozone
[fornari@farm-ops iozone_th_files]$
```

iozone execution distributed on client nodes

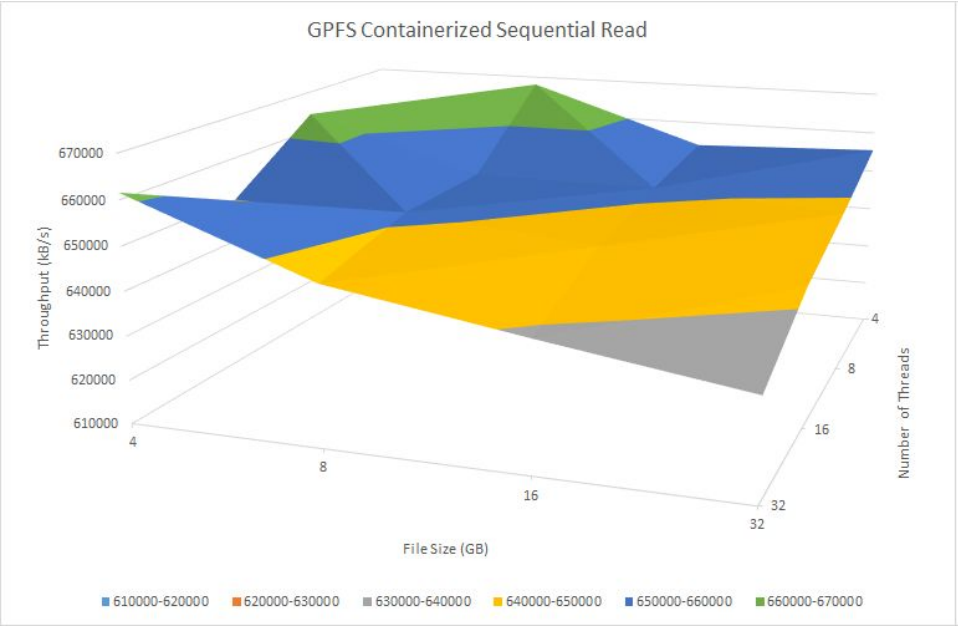
K8s GPFS Cluster Test -> Read Throughput

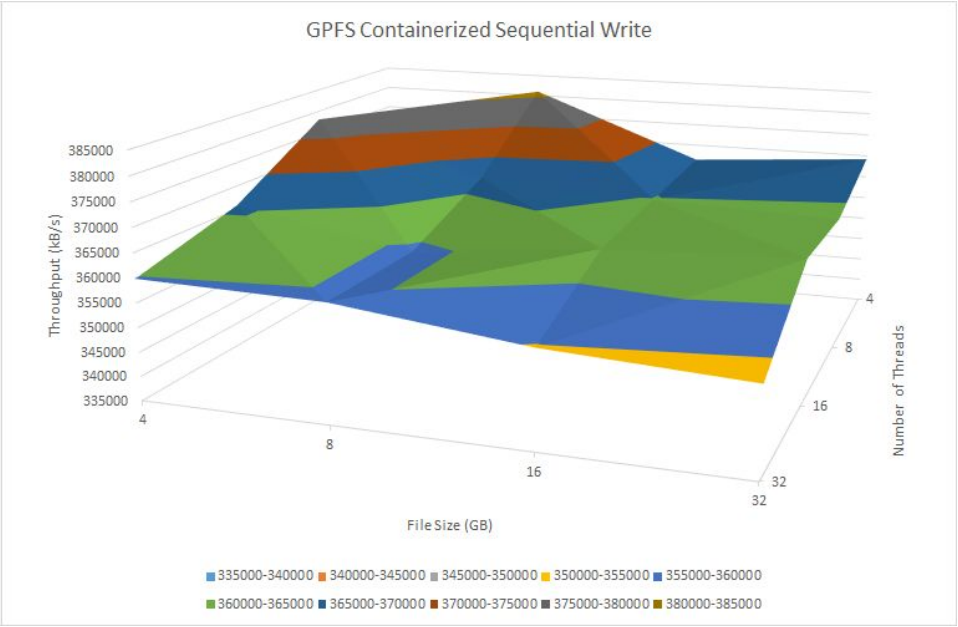


K8s GPFS Cluster Test -> Write Throughput



K8s GPFS Test -> Iozone

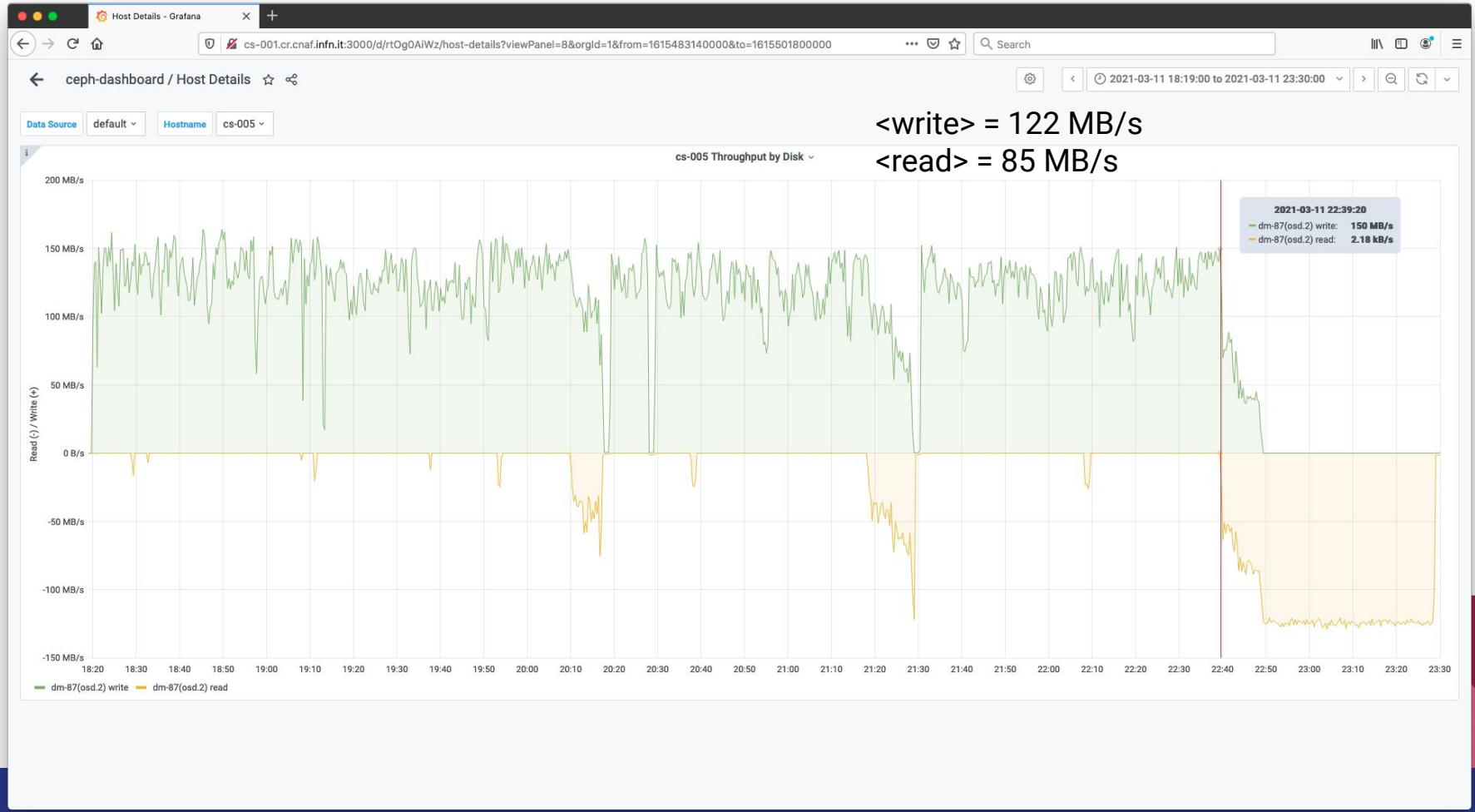




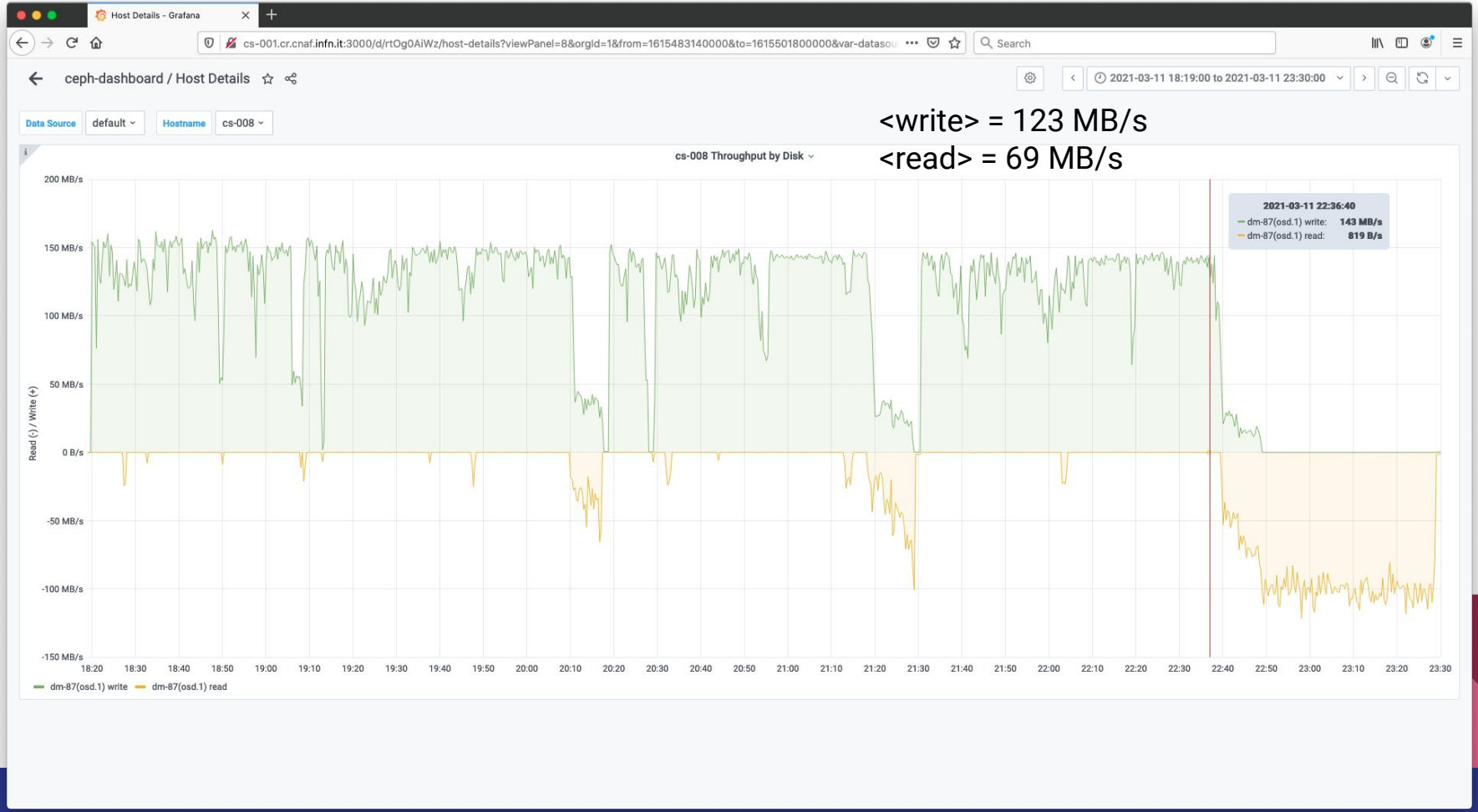
READ	File Size (GB)			
N Threads	4	8	16	32
32	661563.6	645842.5	638884.9	632416.9
16	648580.8	651233.7	647064.3	642291.3
8	664257.5	651393.5	651190.7	648351.1
4	659721	667985.2	653978.2	655476.4

WRITE	File Size (GB)			
N Threads	4	8	16	32
32	359817.8	359082.8	354704.5	352726.3
16	365822.8	359061.1	363184	364777.7
8	378062.3	366796.3	364850.2	363591.5
4	373992.7	380990.9	366411.8	369185.6

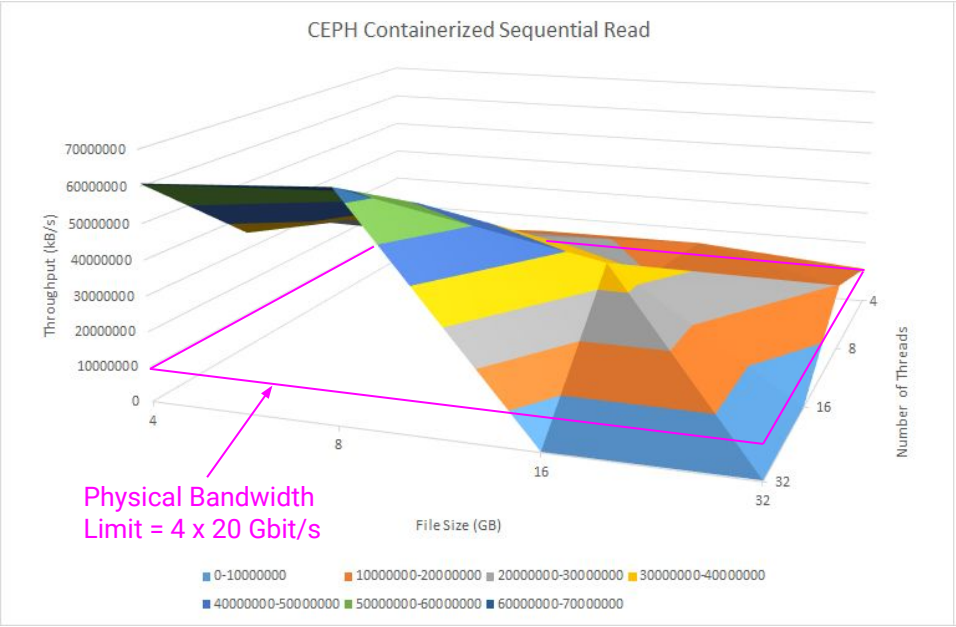
K8s CEPH Cluster Test -> cs-005

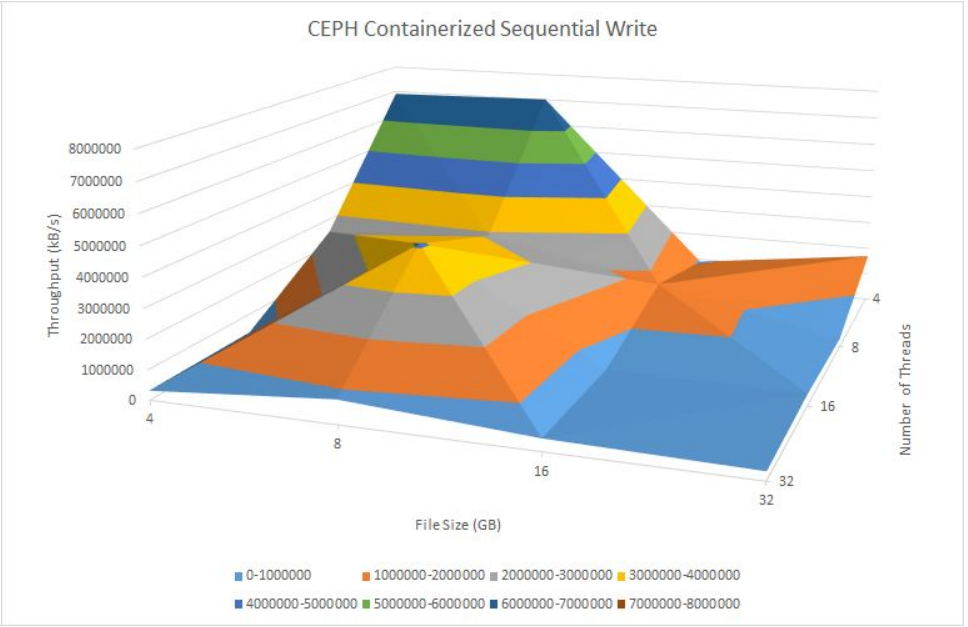


K8s CEPH Cluster Test -> cs-008



K8s CEPH Test -> Iozone

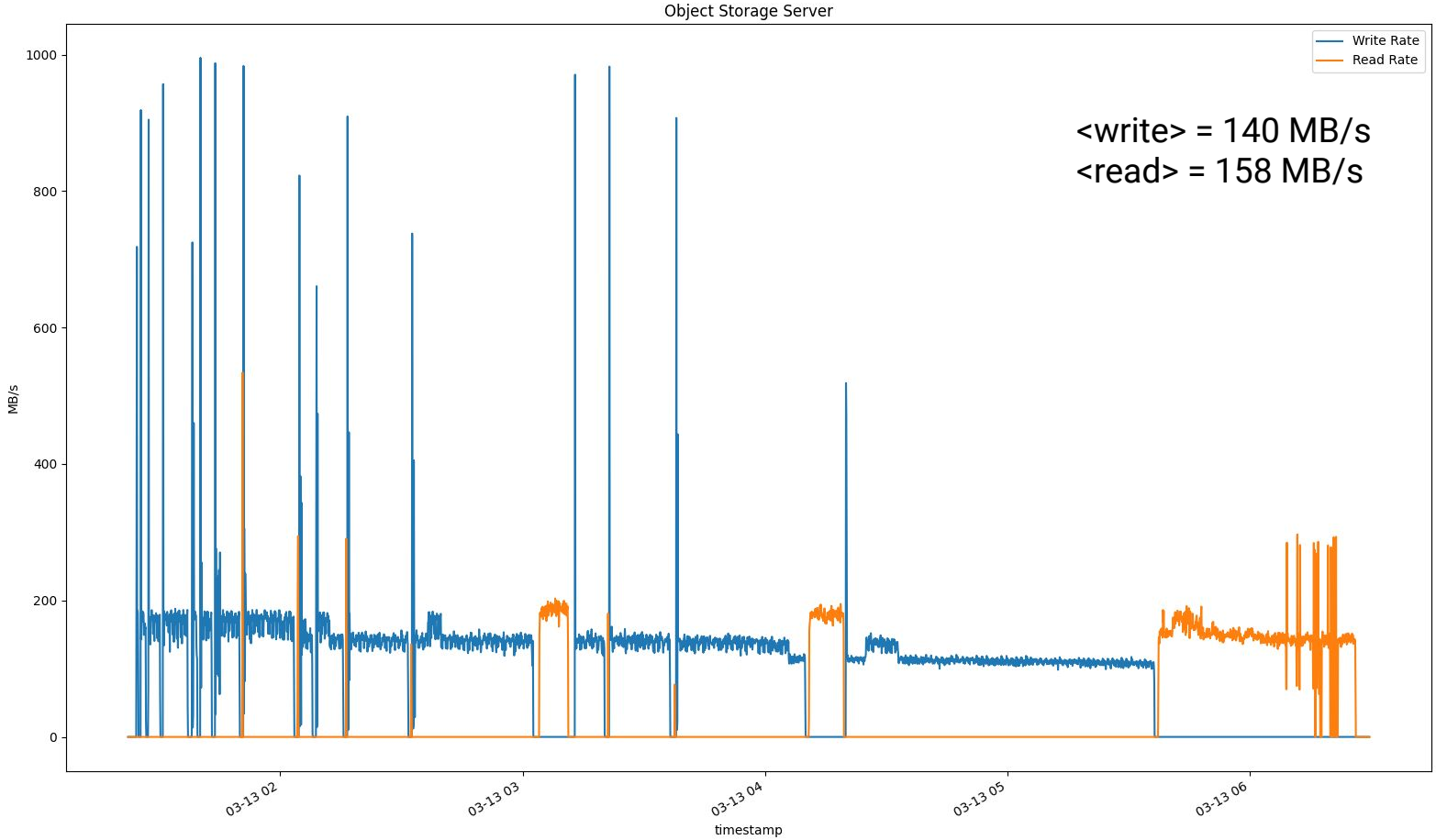




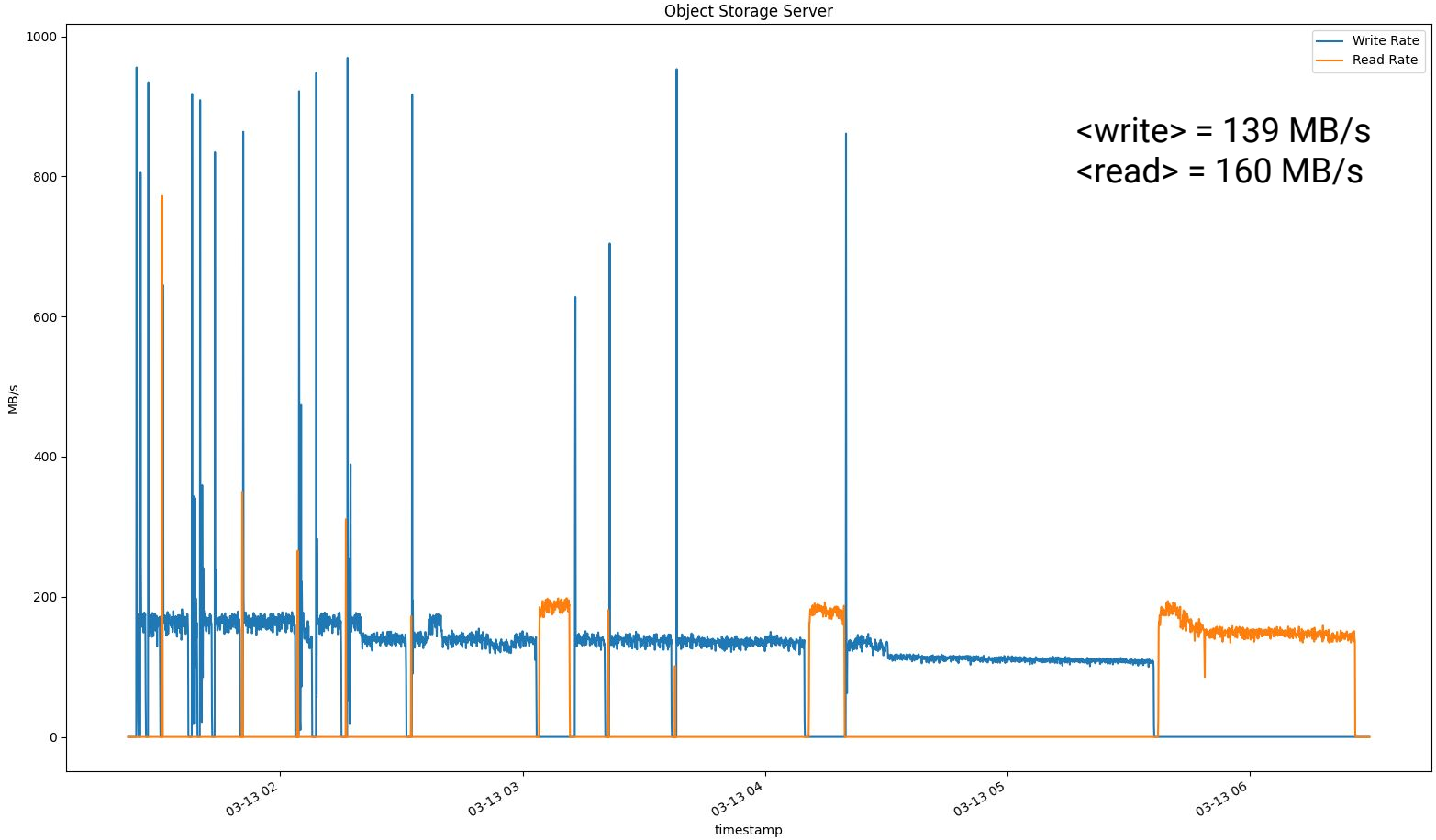
READ	File Size (GB)			
N Threads	4	8	16	32
32	60813065	63908128	182454.2	325401.4
16	34806781	48075323	35490314	191484
8	25596897	26442987	19380236	19739864
4	10706003	14833897	15162460	11005315

WRITE	File Size (GB)			
N Threads	4	8	16	32
32	312791.5	805753.6	404822.6	287302.4
16	354195.4	4097362	465916	399017.4
8	2516173	2891599	1646133	297694.3
4	6898321	7009735	902976.7	1693195

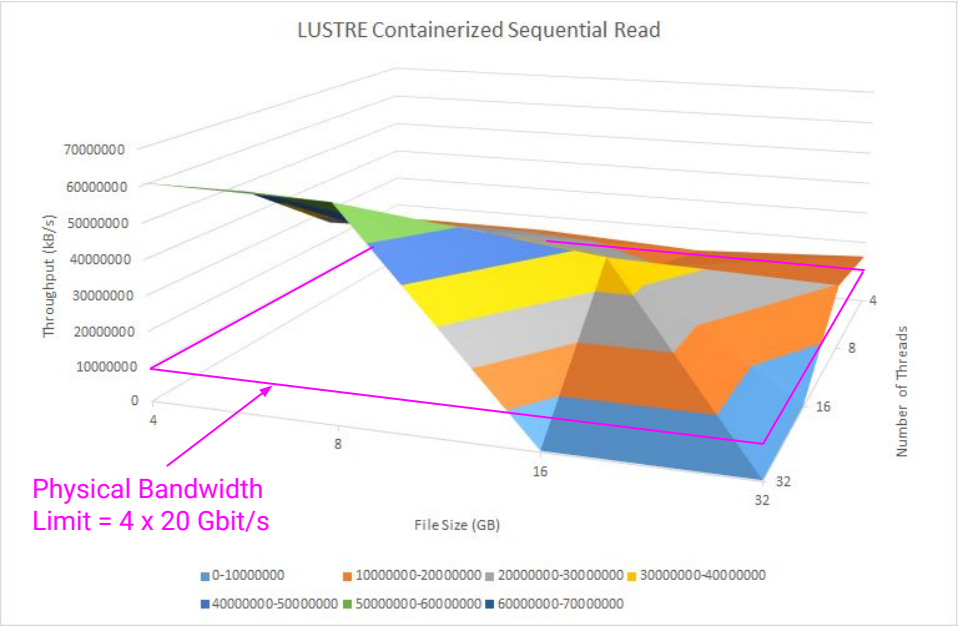
K8s Lustre Bare Metal Test -> OSS 1



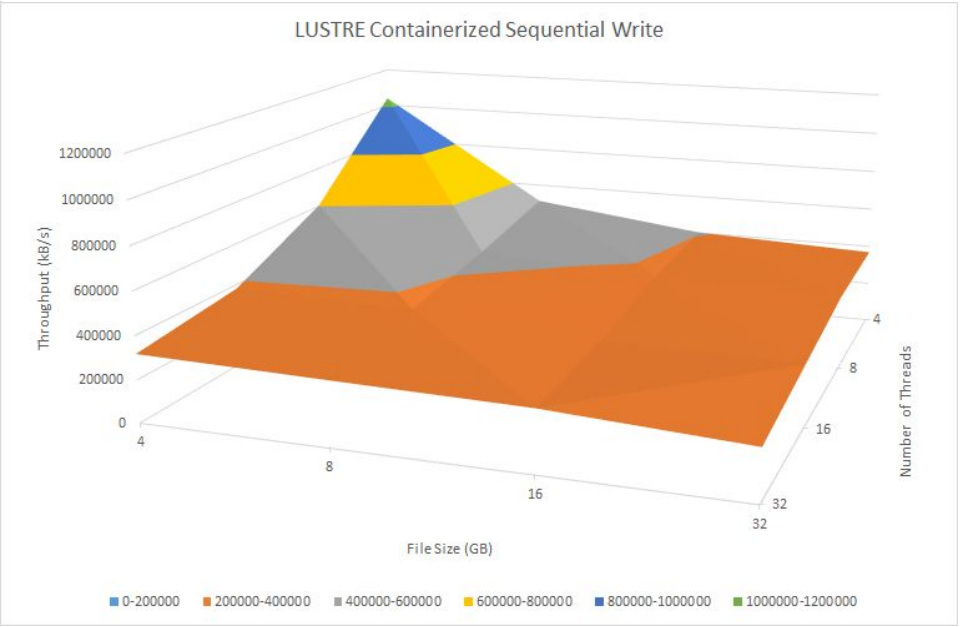
K8s Lustre Bare Metal Test -> OSS 2



K8s Lustre Test -> Iozone



Physical Bandwidth
Limit = 4 x 20 Gbit/s



READ	File Size (GB)			
N Threads	4	8	16	32
32	61058194	60027752	381984.3	308573.3
16	47487395	36190656	37021501	431107.9
8	25666642	27106206	20534411	19764381
4	15262967	15062812	12446332	15218170

WRITE	File Size (GB)			
N Threads	4	8	16	32
32	317410.8	300318.5	284665.7	236158.7
16	381405.3	365325.8	308683.1	297843.7
8	602870.7	427289.5	379880.6	358864.9
4	1038826	510253.6	402137.8	368413.1

Part II: Summary

- Good performance stability observed for all distributed filesystems.
- IBM Spectrum Scale (GPFS) showed best read/write throughput performances on server side and least cache effects on client side.
- CEPH and Lustre are strongly dominated by cache effects on client side especially for small files reading with high number of threads, but if the total size of read/written files gets equal or greater w.r.t. server RAM, performances become similar (however not better) to GPFS ones.
- Considering a single server, Lustre showed better read/write performances than CEPH; anyway Lustre has only half of CEPH/GPFS servers because of DRBD setup (not recommended in production, better to use Corosync).

Thank you very much for your attention!