Western Digital.

BeeGFS®

REFERENCE ARCHITECTURE JANUARY 2022

High-Performance BeeGFS Filesystem Reference Architecture & Deployment Guide on OpenFlex™ F3200

Puspanjali Panda | Saravana Kumar Pandian | Pavan Gururaj

Abstract

This document provides an overview of how to deploy BeeGFS parallel file systems on OpenFlex devices. This guide includes a basic building block that is composed of OpenFlex F3200 devices as the storage backends for the storage targets.

Contents

1. Executive Summary

For enterprises, managing on-premises HPC environments is a challenge. Organizations are constrained by data center space, available power, environmental considerations, time, and complexity to deploy new infrastructure, and the significant investments required. As the range of HPC workloads has expanded to include Big Data, analytics, and various AI applications, these challenges have become acute.

As in today's world, modern HPC workloads and data they operate on have changed. Big Data analytics, artificial intelligence (AI), and machine learning (ML) typically require a solution that provides very low latency and high IO throughput. These workloads demand that compute and storage be scaled independently.

Composable Disaggregated Infrastructure (CDI) represents the modern architectural approach to data center infrastructure, disaggregating compute, storage, and network resources into shared pools that can be composed for on-demand allocation.

The OpenFlex™ F3200 is a fabric device that leverages the Open Composable Infrastructure (OCI) approach in the form of disaggregated data storage using NVMe™ over Fabrics (NVMe-oF™). BeeGFS is an easily deployable parallel file system developed with a strong focus on performance and designed for ease of use, simple installation, and easy management. A parallel file system, also known as a clustered file system, is a type of storage system designed to store data across multiple networked servers and to facilitate high-performance access through simultaneous, coordinated input/output operations (IOPS) between clients and storage nodes.

The purpose of this document is to showcase a combined solution of BeeGFS cluster deployment on servers provisioned with OpenFlex F3200 fabric devices using an NVMe-oF protocol.

2. Problem Statement

In today's world, the volume of data generated by companies and individuals is increasing very rapidly. IDC believes that file-based storage (FBS) will continue to evolve to address the needs of traditional and next-generation workloads. Changes in workload and data impact the choice of a file system. Many applications now must accommodate both large blocks of data and many small files. For instance, training an ML model might use millions of small files, while a Big Data analytics workload runs on one massive dataset. There also is much more use of metadata (consisting of numerous small files) in many workloads today.

Many customers have performance complaints about these file system solutions but this is often not so much an issue with the file system itself but rather one of the storage media and the types of workloads that are running. Many distributed parallel file systems have evolved to support the latest compute and fast storage options like NVMe SSDs in HPC clusters. BeeGFS combines multiple storage servers to provide a highly scalable shared network file system with striped file contents. In such a system, high throughput demands of large numbers of clients can easily be satisfied.

As more and more business opportunities are depending on big data, ML, AI, and Internet of Things (IoT) workloads, they require data infrastructure designed to scale storage and compute independently to ensure both are provisioned efficiently and effectively. Composable Disaggregated Infrastructure (CDI) ensures compute, storage, and network resources are placed into shared pools that can be composed for on-demand allocation. This enables compute to become stateless, elastic, and scalable independent of storage.

In this document we demonstrate a solution to the growing demand of high-performance parallel file system for HPC cluster, by deploying a high-performance distributed parallel file system on a composable infrastructure.

3. Solution Highlights

Western Digital, a pioneer in reliable, high-density industry-standard hardware for software-defined storage projects, is partnering with BeeGFS, a leading hardware-independent POSIX parallel file system, providing fast access to storage systems of all kinds and sizes in all performance-oriented environments including and not limited to HPC, AI and deep learning, life sciences and oil and gas. The following sections of this paper provide an overview of high-performance parallel file systems – built on Western Digital, powered by BeeGFS.

The solution combines:

- **Western Digital OpenFlex:** OpenFlex F3200, which leverages an open composable infrastructure approach in the form of disaggregated data storage using NVMe-oF (NVMe over Fabrics), is a perfect fit for scaling big data and AI environments.
- **BeeGFS:** BeeGFS is a high-performance parallel file system designed for performance-oriented environments like HPC, AI, and deep learning workloads. BeeGFS includes a distributed metadata architecture for scalability and flexibility reasons. Its most important aspect is data throughput.

By combining BeeGFS with Western Digital OpenFlex F3200, organizations can benefit from parallel file systems:

- Robust performance
- Scalability
- High availability
- Data durability
- Data integrity
- Data throughput
- Easy to deploy and integrate with existing infrastructure
- Easy data management
- Optimized for highly concurrent access
- BeeGFS – open source software with enterprise features
- Easy cloud integration

4. Technology Overview

4.1 OpenFlex™ E3000 Enclosure with F3200 Fabric Device Overview

The OpenFlex E3000 is a 3U rack-mounted data storage enclosure built on the OpenFlex platform. OpenFlex is Western Digital's architecture that supports Open Composable Infrastructure (OCI) through storage disaggregation. The OpenFlex F3200 is a fabric device that leverages this OCI approach in the form of disaggregated data storage using NVMe-over-Fabrics (NVMe-oF). NVMe-oF is a networked storage protocol that allows storage to be disaggregated from compute to make that storage widely available to multiple applications and servers.

By enabling applications to share a common pool of storage capacity, data can be easily shared between applications, or needed capacity can be allocated to an application regardless of location. Exploiting NVMe device-level performance, NVMe-oF promises to deliver the lowest end-to-end latency from application to shared storage. NVMe-oF enables composable infrastructures to deliver the data locality benefits of NVMe DAS (low latency, high performance) while providing the agility and flexibility of sharing storage and compute.

The maximum data storage capacity of E3000 is 614TB when leveraging a full set of 10 F3200 fabric devices. F3200 is capable of scaling up to 2 million IOPs and cumulatively we can scale for each E3000 up to 20 million IOPs in a 3U solution.

Composable Infrastructure seeks to disaggregate compute, storage, and networking fabric resources into shared resource pools that can be available for on-demand allocation (i.e., "composable"). Composability occurs at the software level, disaggregation occurs at the hardware level using NVMe-over-Fabrics (NVMe-oF). NVMe-oF will vastly improve compute and storage utilization, performance, and agility in the data center.

Western Digital's vision for Open Composable Infrastructures (OCI) is based on four key pillars:

Open

- Open in both API and form factor.
- Designed for robust interoperability of multi-vendor solutions.

Scalable

- Ability to compose solutions at the width of the network.
- Enable self-organizing systems of composable elements that communicate horizontally.

Disaggregated

- Pools of resources available for any use case that is defined at run time.
- Independent scaling of compute and storage elements to maximize efficiency and agility.

Extensible

- Inclusive of both disk and flash.
- Entire ecosystem of composable elements managed and orchestrated using a common API framework.
- Prepared for yet-to-come composable elements – e.g., memory, accelerators.

Open Composable API – Western Digital's new Open Composable API is designed for data center composability. It builds upon existing industry standards utilizing the best features of those standards as well as practices from proprietary management protocols.

OpenFlex is Western Digital's architecture that supports OCI through storage disaggregation – both disk and flash natively attached to a scalable fabric. OpenFlex does not rule out multiple fabrics, but whenever possible, ethernet will be used as a unifying connect for both flash and disk because of its broad applicability and availability.

OpenFlex F3200 specification summary

Specification	Value
Max raw data storage capacity per device	61.4 TB
Data ingest capability	2 × 50Gb Ethernet
Data transfer rates	12 GBps*
Number per enclosure	Up to 10
Hot swappable	Yes

4.2 BeeGFS Overview

BeeGFS is a hardware-independent POSIX parallel file system (a.k.a., software-defined parallel storage) developed with a strong focus on performance and designed for ease of use, simple installation, and easy management. It is designed for all performance-oriented environments including HPC, AI, and deep learning, life sciences, and oil and gas.

One of the most fundamental concepts of BeeGFS is the strict avoidance of architectural bottle necks or locking situations in the cluster, through the user space architecture. BeeGFS is striping the file content on multiple storage nodes, plus it is distributing file system metadata across multiple metadata servers, which benefits small, large systems, and metadata-intensive applications.

BeeGFS is built on highly efficient and scalable multithreaded core components with native RDMA support. File system nodes can serve RDMA (InfiniBand, (Omni-Path), RoCE and TCP/IP) network connections at the same time and automatically switch to a redundant connection path in case any of them fail.

BeeGFS is available free of charge for end users. For production systems, professional commercial support is also available, typically in cooperation with international turn-key solution partners.

The BeeGFS architecture is composed of four main services.

Management service

- The management service can be figured as a "meeting point" for the BeeGFS metadata, storage, and client services.
- Very light-weight and typically not running on a dedicated machine, as it is not critical for performance and stores no user data.
- First service needs to be set up in a newly deployed environment.
- Watches all registered services and checks their state and maintains it.

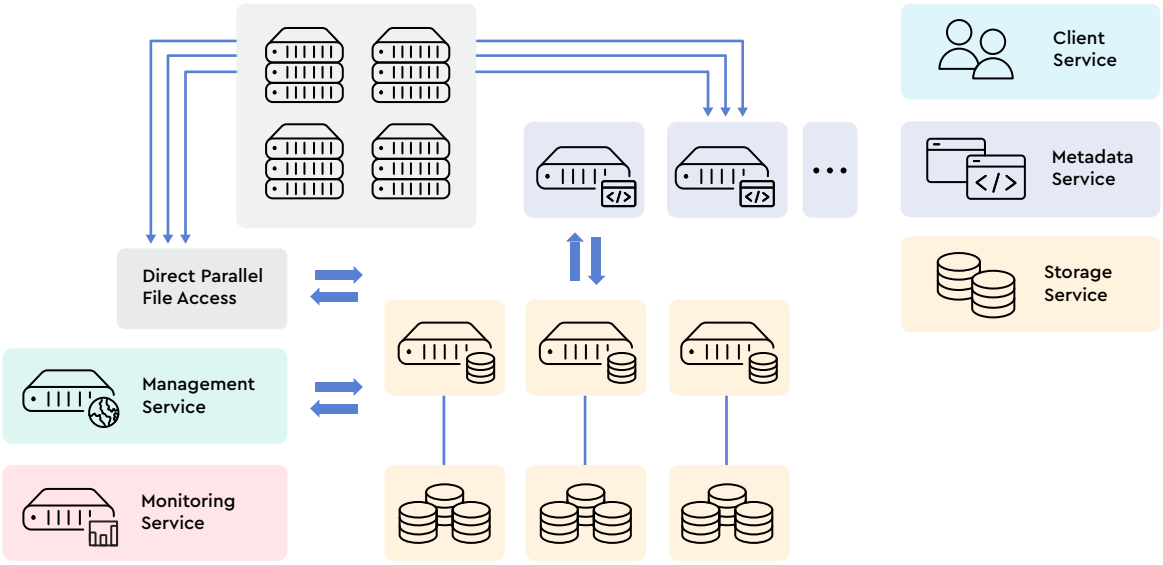


Figure 1. BeeGFS architecture, four main services

Metadata service

- The metadata service stores information about the data; e.g., directory information, file and directory ownership, and the location of user file contents on storage targets.
- It offers scale-out service, meaning there can be one or many metadata services in a BeeGFS file system.
- Each metadata service is responsible for its exclusive fraction of the global namespace, so that having more metadata servers improves the overall system performance.
- Usually, a metadata target is an ext4 file system based on a RAID1 or RAID10 of flash drives. 512GB of usable metadata capacity are typically good for about 150 million user files.
- Using faster CPU cores will result in low metadata access latency.

Storage service

- The storage service (sometimes also referred to as the "object storage service") is the main service to store striped user file contents, also known as data chunk files.
- It is based on a scale-out design. That means, it can have one or multiple storage services per BeeGFS file system instance, to add more capacity and performance to the file system.

- A storage service instance has one or multiple storage targets.
- The storage target can generally be any directory on a local file system (usually xfs); a storage target typically is a hardware RAID-6 (typically composed of 8+2 or 10+2) or zfs RAIDz2 volume, of either internal or externally attached drives.

Client service

- BeeGFS comes with a client that registers natively with the virtual file system interface of the Linux kernel for maximum performance.
- Client kernel module has to be compiled to match the used kernel.
- When the client module is loaded using the client service, it will mount the file systems defined in beegfsmounts.conf.
- Client kernel module uses an additional user space helper daemon for DNS lookups and to write the log file.

5. BeeGFS Solution Design on OpenFlex

Western Digital, a pioneer in reliable, high-density industry-standard hardware for software-defined storage projects, is partnering with BeeGFS to provide a verified, highly scalable shared network file system with striped file contents and high-performance computing and mirroring (HA) for on-premises and cloud systems. The following sections of this paper provide an overview of storage solution.

5.1 BeeGFS on OpenFlex Deployment Topology

OpenFlex and BeeGFS have created a unique reference architecture with industry-standard servers and ethernet network switches. It leverages the OCI approach in the form of disaggregated data storage using NVMe-over-Fabrics (NVMe-oF). F3200 uses NVMe-oF technology with NVMe-RoCEv2 protocol for all network transmissions. This end-to-end solution can be specifically designed to accelerate large-scale production while delivering the compute and storage performance needed. The figure below illustrates a verified deployment topology with OpenFlex F3200.

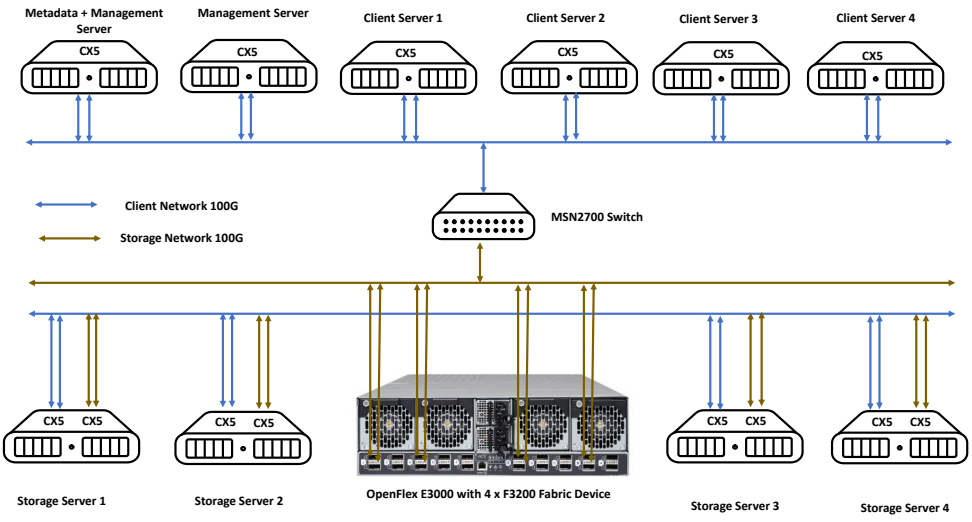


Figure 2. Verified deployment topology with OpenFlex F3200

BeeGFS deployment with OpenFlex requires the below steps.

- Set up the OpenFlex system.
- Set up the storage servers, metadata servers and client server's prerequisites.
- Set up DRBD and PCS to avail HA for BeeGFS management.
- Set up BeeGFS services (management, metadata, storage, and client).
- Apply tuning parameters to improve performance.
- Set up mirroring for storage and metadata to avail HA.

Hardware Requirements

In the following deployment procedures, 100Gb host channel adapters (HCAs) connect each server (metadata or storage) to the OpenFlex system. However, you can use any other block protocol that is supported. For simplicity in this use case, metadata and management services are running on the same server, and storage services are running on another four servers. There are also servers running the client services.

Hardware Specification

BeeGFS Storage Server Details

Storage product	OpenFlex E3000 with 4 x F3200 fabric devices
Storage interface	Dual QSFP28 (2 × 50Gb) per fabric device
Host OS	Red Hat Enterprise Linux release 8.2 (Ootpa)
Kernel	4.18.0–193.28.1.el8_2.x86_64
Host NIC	2 x CX5 – MCX516A-CCAT (100Gbps)
CX5 OFED package version	5.0–2.1.8
CPU	Intel® Xeon® Gold 6240R CPU @ 2.40GHz
CPU core details	Dual socket server with 24 core CPU each. 96 logical cores in total with HT enabled
Memory	128GB
No of volumes on each storage server	2

BeeGFS Management / Metadata Server Details

Host OS	Red Hat Enterprise Linux release 8.2(Ootpa)
Kernel	4.18.0–193.28.1.el8_2.x86_64
Host NIC	1 x CX5 – MCX516A-CCAT (100Gbps)
CX5 OFED package version	5.0–2.1.8
CPU	Intel® Xeon® Gold 5118 CPU @ 2.30GHz
CPU core details	Dual socket server with 16 core CPU each. 64 logical cores in total with HT enabled
NIC firmware version	16.28.2006 (MT_0000000012)
Memory	128GB
NVMe disk	2 (1.8TB each)

BeeGFS Client Server Details

Host OS	Red Hat Enterprise Linux release 8.2(Ootpa)
Kernel	4.18.0–193.28.1.el8_2.x86_64
Host NIC	1 x CX5 – MCX516A-CCAT (100Gbps)
CX5 OFED package version	5.0–2.1.8
CPU	Intel® Xeon® Gold 5118 CPU @ 2.30GHz
CPU core details	Dual socket server with 24 core CPU each. 96 logical cores in total with HT enabled
NIC firmware version	16.28.2006 (MT_0000000012)
Memory	128GB

Below are the detailed configuration used for this deployment:

- 1 x OpenFlex E3000 fabric enclosure.
- 4 x OpenFlex F3200 fabric devices. 2 x volumes on each F3200 device.
- 4 x storage servers and 2 x CX5 cards per server. Each storage server connected to 2 volumes of F3200 device running multimode configuration. Total 8 storage instances running on 8 volumes.

- 2 x management server and 1 x CX5 card per server. 1 x NVMe SSD each per management instances. Management server running DRBD and PCS HA.
- 1 x metadata server and 1 x CX5 card per server. 1 x NVMe SSD each per metadata instances. Total 2 metadata instances running on a single server.
- 4 x client servers and 1 x CX5 card per server.
- Mellanox MSN2700 – 32-port switch.
- 4 Buddy Mirror groups.
- Each F3200 device is 100G capable.

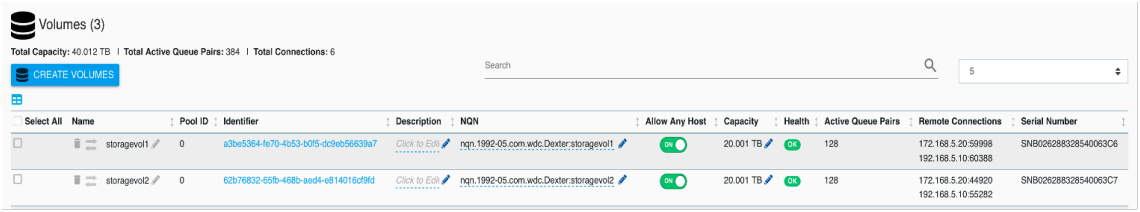
5.2 Set Up OpenFlex System

Refer to @openflex-composable-infrastructure for more details on OpenFlex. Follow the steps mentioned in @OpenFlex User Guide to set up the OpenFlex system. For more details regarding lossless configuration on OpenFlex contact @WD Support.

1. Create volumes for the storage services.

2. Assign the volumes to the appropriate servers.

The below screenshot shows the volume-to-host assignments that are used for this deployment on a single storage server.



Pre-requisites for all servers

1. Configure NTP on all servers and sync the servers.

2. Disable Selinux on all systems. Edit the below file and set the value to disabled.

```
$ vi /etc/selinux/config
$ SELINUX=disabled
```

3. Reboot the server after disabling Selinux.

4. Install Mellanox firmware. Check the firmware details after installation.

```
$ [root@storage1 ~] # ethtool -i ens1f0
driver: mlx5_core
version: 5.0-2.1.8
firmware-version: 16.27.2008 (MT_0000000012)
expansion-rom-version:
bus-info: 0000:3b:00.0
supports-statistics: yes
supports-test: yes
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: yes
```

5. Edit the file /etc/multipath.conf.

```
Defaults {
uid_attribute          ID_WWN
user_friendly_names    yes
path_grouping_policy   multibus
failback               immediate
path_selector          “round-robin 0”
no_path_retry          queue
}
blacklist_exceptions {
property “(ID_WWN|SCSI_IDENT_.*|ID_SERIAL|DEVTYPE)”
devnode “nvme*”
}
```

6. Restart the multipath service.

```
$ systemctl restart multipathd
```

7. Add the below details under the grub file to unload native multipath. Reboot the system.

```
$ grubby --update-kernel=ALL --args=”nvme_core.multipath=N
```

5.3 Setting Up Dm Multipath When Native Multipath is Already Configured

1. Disconnect all previously mapped volumes. Ensure the volume got disconnected successfully.

```
$ [root@ storage1~]# nvme disconnect-all
$ [root@ storage1~]# nvme list
```

2. Unload all the NVMe modules.

```
$ [root@ storage1 ~]# lsmod|grep nvme
$ [root@ storage1~]# rmmod nvme_rdma
$ [root@ storage1~]# rmmod nvme_fabrics
$ [root@ storage1 ~]# rmmod nvme
$ [root@ storage1 ~]# rmmod nvme_core
$ [root@ storage1 ~]# lsmod|grep nvme
```

3. Add the below details under the grub file to unload native multipath. Reboot the system.

```
$ grubby --update-kernel=ALL --args=”nvme_core.multipath=N”
```

4. Check the server after reboot and follow the steps mentioned in section 5.4 to set up lossless configuration, reload NVMe modules, and configure volume mapping. Verify the multipath configuration details once volume is mapped after boot.

```
$ [root@ storage1 ~]# multipath -ll
mpathe (uuid.863de30e-58ff-4b8d-ba8e-d65c4cb7edfd) dm-6 NVME,OpenFlex F3000
size=14T features=’1 queue_if_no_path’ hwhandler=’0’ wp=rw
`-+- policy=’round-robin 0’ prio=1 status=active
 |- 6:9:1:1  nvme6n1 259:5 active ready running
 `- 8:10:1:1 nvme8n1 259:7 active ready running
mpathd (uuid.8f0be9c4-4b18-4303-93ba-a40bf7396aa0) dm-5 NVME,OpenFlex F3000
```

5.4 Set Up Storage Server

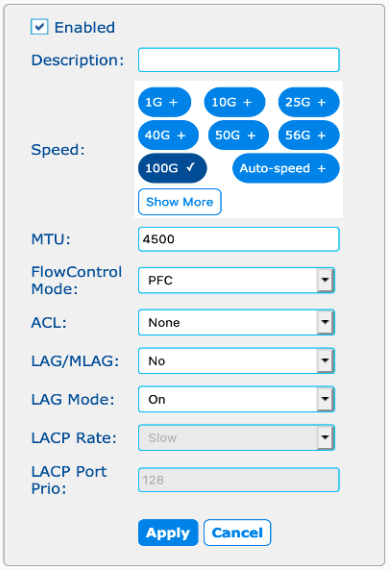
1. Set up lossless configuration on the host by running the below commands on all the storage nodes. For more details regarding lossless configuration on OpenFlex contact @WD Support.

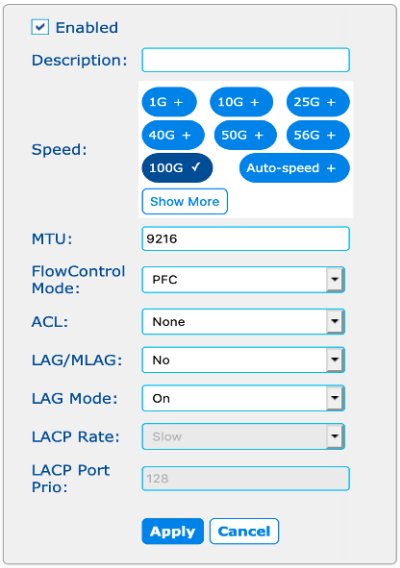
```
modprobe -v nvme-core multipath=no
modprobe -v nvme
modprobe -v nvme-rdma

mlnx_qos -i ens1f0
mlnx_qos -i ens1f1
mst start
mst status
yes | mlxconfig -d /dev/mst/mt4121_pciconf0 set LLDP_NB_DCBX_P1=FALSE LLDP_NB_TX_
MODE_P1=2 LLDP_NB_RX_MODE_P1=2 LLDP_NB_DCBX_P2=FALSE LLDP_NB_TX_MODE_P2=2 LLDP_NB_RX_
MODE_P2=2 PCI_WR_ORDERING=1
#yes | mlxconfig -d /dev/mst/mt4121_pciconf0.1 set LLDP_NB_DCBX_P1=FALSE LLDP_NB_TX_
MODE_P1=2 LLDP_NB_RX_MODE_P1=2 LLDP_NB_DCBX_P2=FALSE LLDP_NB_TX_MODE_P2=2 LLDP_NB_RX_
MODE_P2=2
yes | mlxconfig -d /dev/mst/mt4121_pciconf1 set LLDP_NB_DCBX_P1=FALSE LLDP_NB_TX_MODE_
P1=2 LLDP_NB_RX_MODE_P1=2 LLDP_NB_DCBX_P2=FALSE LLDP_NB_TX_MODE_P2=2 LLDP_NB_RX_MODE_
P2=2 PCI_WR_ORDERING=1
#yes | mlxconfig -d /dev/mst/mt4121_pciconf1.1 set LLDP_NB_DCBX_P1=FALSE LLDP_NB_TX_
MODE_P1=2 LLDP_NB_RX_MODE_P1=2 LLDP_NB_DCBX_P2=FALSE LLDP_NB_TX_MODE_P2=2 LLDP_NB_RX_
MODE_P2=2
mlnx_qos -i ens1f0 --trust dscp
mlnx_qos -i ens1f1 --trust dscp
mlnx_qos -i ens1f0 --pfc 0,0,0,1,0,0,0,0
mlnx_qos -i ens1f1 --pfc 0,0,0,1,0,0,0,0
mlnx_qos -i ens1f0 ----buffer_size 130944,130944,0,0,0,0,0,0
mlnx_qos -i ens1f1 --prio2buffer 0,0,0,1,0,0,0,0
mlnx_qos -i ens1f0 --tsa ets,ets,ets,ets,ets,ets,strict,ets --tcbw
14,15,14,15,14,14,0,14
mlnx_qos -i ens1f1 ----buffer_size 130944,130944,0,0,0,0,0,0
mlnx_qos -i ens1f0 --prio2buffer 0,0,0,1,0,0,0,0
mlnx_qos -i ens1f0 --tsa ets,ets,ets,ets,ets,ets,strict,ets --tcbw
14,15,14,15,14,14,0,14
cma_roce_mode -d mlx5_0 -m 2
#cma_roce_mode -d mlx5_1 -m 2
cma_roce_mode -d mlx5_2 -m 2
#cma_roce_mode -d mlx5_3 -m 2
cma_roce_tos -d mlx5_0 -t 96
#cma_roce_tos -d mlx5_1 -t 96
cma_roce_tos -d mlx5_2 -t 96

#cma_roce_tos -d mlx5_3 -t 96

sleep 3
```

```
echo 0 > /sys/kernel/debug/mlx5/0000:41:00.0/cc_params/np_cnp_prio_mode
echo 0 > /sys/kernel/debug/mlx5/0000:81:00.0/cc_params/np_cnp_prio_mode
cat /sys/kernel/debug/mlx5/0000:41:00.0/cc_params/np_cnp_prio_mode
cat /sys/kernel/debug/mlx5/0000:81:00.0/cc_params/np_cnp_prio_mode
echo 48 > /sys/kernel/debug/mlx5/0000:41:00.0/cc_params/np_cnp_dscp
echo 48 > /sys/kernel/debug/mlx5/0000:81:00.0/cc_params/np_cnp_dscp
echo 6 > /sys/kernel/debug/mlx5/0000:41:00.0/cc_params/np_cnp_prio
echo 6 > /sys/kernel/debug/mlx5/0000:81:00.0/cc_params/np_cnp_prio
cat /sys/kernel/debug/mlx5/0000:41:00.0/cc_params/np_cnp_prio
cat /sys/kernel/debug/mlx5/0000:81:00.0/cc_params/np_cnp_prio

sleep 5
```

2. Set up lossless configuration on the switch and on the host running the storage node services. For more details regarding lossless configuration refer to the lossless configuration guide.
3. Set up the IPs. Set up MTU value to 4500 for storage ports and MTU value to 9216 for other ports on the host as well as on the switch. Verify all the ports are up and configured properly.

```
ens1f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 4500
inet 172.168.6.20  netmask 255.255.255.0  broadcast 172.168.6.255
ether 0c:42:a1:a4:4d:cd  txqueuelen 1000  (Ethernet)
RX packets 402  bytes 35147 (34.3 KiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 22  bytes 2217 (2.1 KiB)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens2f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9216
inet 192.168.10.13  netmask 255.255.255.128  broadcast 192.168.10.127
ether 98:03:9b:97:20:f6  txqueuelen 1000  (Ethernet)
RX packets 106316  bytes 46681007 (44.5 MiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 199838  bytes 20128327 (19.1 MiB)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```





4. Install NVMe-CLI and check the NVMe device listing.

```
$ [root@ storage1 ~]# nvme --list
```

Node	SN	Model	Namespace Usage	Format	FW Rev
/dev/nvme0n1	SNB02628831AA 8001809	OpenFlex F3200	1	20.00 TB / 2.0.0	4 KiB + 0 B 20.00 TB
/dev/nvme1n1	SNB02628831AA 8001809	OpenFlex F3200	1	20.00 TB / 2.0.0	4 KiB + 0 B 20.00 TB
/dev/nvme2n1	SNB02628831AA 800180A	OpenFlex F3200	1	20.00 TB / 2.0.0	4 KiB + 0 B 20.00 TB
/dev/nvme3n1	SNB02628831AA 800180A	OpenFlex F3200	1	20.00 TB / 2.0.0	4 KiB + 0 B 20.00 TB

5. After you have followed the above procedures on all the storage servers, map the volumes and check the device mapper multipathing details.

```
$ [root@ storage1 ~]# multipath -ll
mpathc (uuid.62237347-1f7a-43d9-b957-a3e758135260) dm-4 NVME,OpenFlex F3200
size=18T features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
|- 2:6:1:1 nvme2n1 259:2 active ready running
`- 3:5:1:1 nvme3n1 259:3 active ready running
mpathb (uuid.da081584-8630-4dbd-9df5-82910952f7df) dm-3 NVME,OpenFlex F3200
size=18T features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
|- 0:5:1:1 nvme0n1 259:0 active ready running
`- 1:6:1:1 nvme1n1 259:1 active ready running
```

6. Repeat the above steps in all storage servers.

7. Verify that the following conditions are true:

- The hosts can ping each other's IPs and host names.
- Volumes are assigned to the appropriate hosts.

5.5 Set Up Management and Metadata Server

1. Set up the IP. Set up MTU value to 9216 for the ports on host and switch.

2. Install NVMe-CLI. Check the newly added disks for metadata are getting listed.

```
$ [root@meta1 ~]# nvme list
```

Node	SN	Model	Namespace Usage	Format	FW Rev
/dev/nvme1n1	A0597109	WUS4BB019 D7P3E3	1	1.92 TB / 1.92 TB	4 KiB + 0 B R1110007
/dev/nvme0n1	A0597123	WUS4BB019 D7P3E3	1	1.92 TB / 1.92 TB	4 KiB + 0 B R1110007

```
$ [root@meta1 ~]# lsblk
NAME                                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                                 8:0  0 894.3G  0 disk
|-sda1                              8:1  0   600M  0 part /boot/efi
|-sda2                              8:2  0     1G  0 part /boot
`-sda3                              8:3  0 892.7G  0 part
|-rhel_dhcp--10--206--137--62-root 253:0    0    50G  0 lvm  /
|-rhel_dhcp--10--206--137--62-swap 253:1    0     4G  0 lvm  [SWAP]
`-rhel_dhcp--10--206--137--62-home 253:2    0 838.7G  0 lvm  /home
nvme0n1                             259:0    0   1.8T  0 disk /data/beegfs/beegfs_meta1
nvme1n1                             259:1    0   1.8T  0 disk /data/beegfs/beegfs_meta2
```

3. Repeat the above steps in all management and metadata servers.

5.6 Set Up Client Server

1. Set up the IPs. Set up MTU value to 9216 for the ports on host and switch.

2. Repeat the above steps in all client servers.

Note: If the configuration meets all the requirements, you are ready to set up BeeGFS.

6. BeeGFS Multimode Deployment Details

This section provides detailed configuration steps and settings for all the BeeGFS components using multimode configuration. Find more details regarding multimode configuration here @Multimode configuration steps. For single instance configuration refer to @BeeGFS Manual Install Walk Through.

BeeGFS Configuration

The BeeGFS test configuration includes the following components.

- Management server
- Metadata server
- Storage server
- Client server

Based on BeeGFS recommendations, the metadata server is configured to use ext4 file system, and the storage servers are configured to use XFS file systems.

6.1 BeeGFS Management HA Configuration Details

To utilize the high-availability feature for BeeGFS management you need to configure DRBD and Pacemaker cluster software on the management nodes.

6.2 Setting Up DRBD

DRBD is a software-based, shared-nothing, replicated storage solution mirroring the content of block devices (hard disks, partitions, logical volumes etc.) between hosts. DRBD mirrors data.

- Replication occurs continuously while applications modify the data on the device.
- Applications need not be aware that the data is stored on multiple hosts.
- With synchronous mirroring, applications are notified of write completions after the writes have been carried out on all (connected) hosts. With asynchronous mirroring, applications are notified of write completions when the writes have completed locally, which usually is before they have propagated to the other hosts. For more details refer to @DRBD.

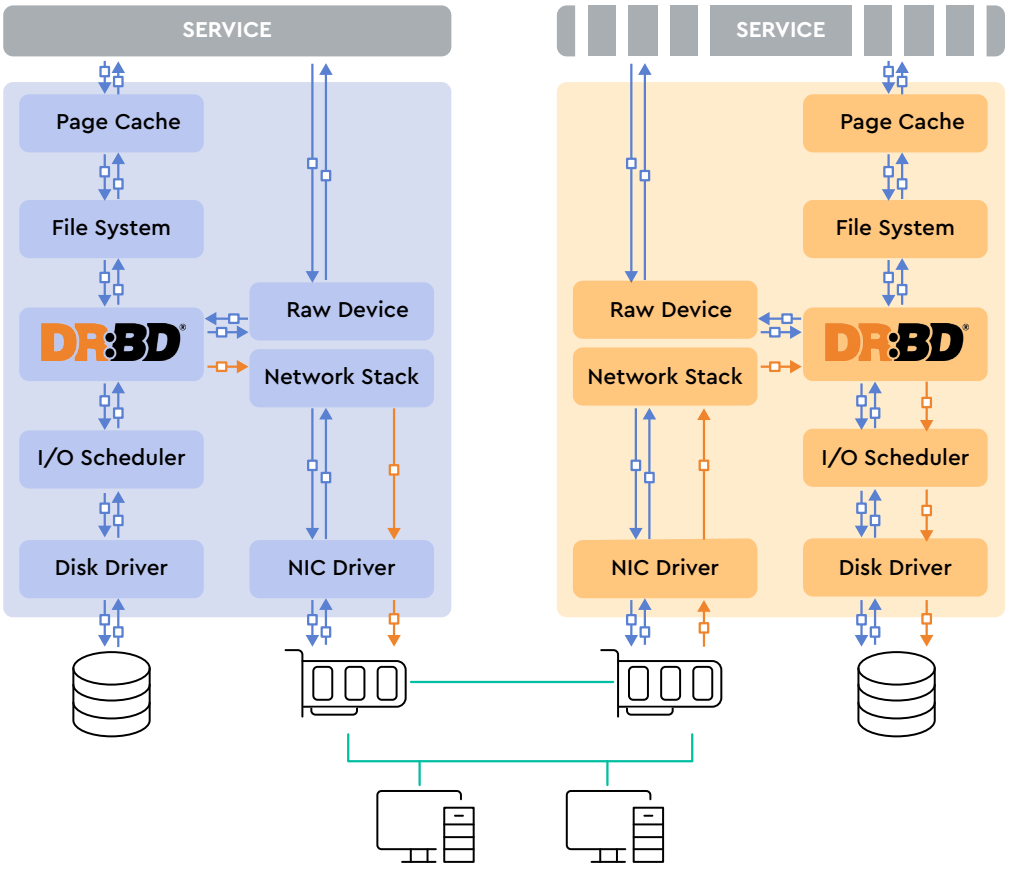


Figure 3: DRBD Design implementation on Linux Stack

Before you install BeeGFS, DRBD, or RHEL HA packages on each node, you must add the package repository for your Linux distribution.

1. The packages for DRBD needs to be installed on an RHEL8 system. For the installation use epel repository which contains extra packages for Enterprise Linux.

```
$ sudo dnf -y install https://www.elrepo.org/elrepo-release-8.el8.elrepo.noarch.rpm
```

2. With the repository added to the system you can import the public key.

```
$ sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
```

3. Confirm configuration by searching for DRBD packages.

```
$ sudo dnf search drbd
```

4. Now install DRBD with kernel module on RHEL8 system.

```
$ sudo dnf install vim drbd90-utils kmod-drbd90
```

5. Install the Redhat high-availability add-on software packages from the high-availability channel and then install Pacemaker agents.

```
$ yum-config-manager --enable rhel-8-for-x86_64-highavailability-rpms
$ yum install pcs pacemaker fence-agents-all
```

6. Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo https://www.beegfs.io/release/latest-stable/dists/beegfs-rhel8.repo
```

7. Install BeeGFS management packages.

```
$ yum install beegfs-mgmtd
```

Pre-requisites for DRBD configuration

1. RHEL8 – 64bit x 2 nodes (node1 and node2).

2. Dedicated local disk on each node (/dev/dm-3 and preferably same size).

Note: Disk size and naming convention should be the same on both nodes.

3. Dedicated network device and IP address on each node for replication.

4. Iptables ports (6996–7800) allowed.

By installing DRBD you'll get a set of administration tools which communicate with the kernel module in order to configure and administer DRBD resources.

drbdadm: This is the high-level administration tool of the DRBD program suite.

drbdsetup: This tool is used to configure the DRBD module loaded into the kernel.

drbdmeta: Used to create, dump, restore, and modify DRBD metadata structures.

6.3 Configure DRBD

Step 1: Configure policies:

1. If Selinux service is not disabled, then under the default Selinux security policies DRBD may fail to run and you may need to exempt DRBD processes from Selinux control.

```
$ sudo dnf -y install policycoreutils-python-utils
$ sudo semanage permissive -a drbd_t
```

2. Enable DRBD service ports on the firewall.

```
$ sudo firewall-cmd --add-port=6996-7800/tcp --permanent
$ sudo firewall-cmd -reload
```

Step 2: Preparing your lower-level storage.

1. With the installation of DRBD done on the RHEL8 system proceed to configure replicated storage using DRBD across two servers. We need to set aside a roughly identically sized storage area on both cluster nodes which are used as lower-level device for your DRBD resource. Use a dedicated separate disk to configure the DRBD resource on both the systems. Ensure the disk size and naming convention are same across both the systems.

```
$ [root@node1-mgmt ~]# lsblk
NAME                MAJ:MIN RM        SIZE RO         TYPE          MOUNTPOINT
sda                 8:0                0 447.1G        0             disk
`-md126             9:126              0 424.8G        0             raid1
|-md126p1           259:0              0 600M          0 md          /boot/efi
|-md126p2           259:1              0 1G            0 md          /boot
`-md126p3           259:2              0 423.2G        0 md
|-cl-root           253:0              0 50G           0 lvm         /
|-cl-swap           253:1              0 4G            0 lvm         [SWAP]
```

```
`-cl-home           253:2            0 369.2G        0 lvm         /home
nvme0n1             259:3            0               1.8T          0 disk
`-mpathb            253:3            0               1.8T          0 mpath
```

```
[root@node1-mgmt ~]# nvme list
                                                  Namespace
Node           SN         Model       Usage       Format           FW Rev
/dev/nvme0n1   A059710E   WUS4BB019   1           1.92TB/1.92TB    4KiB+0B R1110007
                          D7P3E3
```

2. Create a partition table on the disk on both management servers. You can configure ext3, ext4 or XFS partition as per your preference.

```
$ [root@ node1-mgmt ~]# multipath -ll
mpathb (eui.0100000000000000014ee830026dc00) dm-3 NVME,WUS4BB019D7P3E3
size=1.7T features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
`- 0:0:1:1 nvme0n1 259:3 active ready running
```

```
$ [root@ node1-mgmt]# mkfs.ext3 /dev/dm-3
mke2fs 1.44.6 (5-Mar-2019)
Discarding device blocks: done
Creating filesystem with 468829289 4k blocks and 117211136 inodes
Filesystem UUID: b31fe41e-b742-468e-979f-1ba3bcf03c00
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

Step 3: Configure DRBD resource.

1. Add the IP address and host name of all the management servers for DNS resolution under /etc/hosts file in all the management nodes.
2. Create a DRBD resource file with the contents below on both the nodes under /etc/drbd.d. The file contents should be the same across both nodes.
   - The resource file "beegfs.res" is used to define the resource name and details.
   - Protocol C defines the replication protocol. Resources are configured to use fully synchronous replication using (Protocol C).
   - "node1-mgmt"- defines the first node name and its options within the statement.
   - "device /dev/drbd1;" – defines the logical device name used for the block device. (Should be common on both nodes.)
   - "disk /dev/dm-3;" – defines the dedicated local disk used for replication.

- "address 192.168.10.20:7001" – defines the IP address assigned to the dedicated network used for the replication resource and the TCP port (7001/7002) details used for communication.
- meta-disk internal – This is used to define the internal metadata disk details.

```
$ [root@ node1-mgmt ~]# cat /etc/drbd.d/beegfs.res
resource beegfs {
protocol C;
on node1-mgmt {
                device /dev/drbd1;
                disk /dev/dm-3;
                address 192.168.10.20:7001;
                meta-disk internal;
                node-id 0;
        }
        on node2-mgmt {
                device /dev/drbd1;
                disk /dev/dm-3;
                address 192.168.10.10:7002;
                meta-disk internal;
                node-id 1;
        }
        connection {
        host node1-mgmt port 7001;
        host node1-mgmt port 7002;
        }
        }
```

Step 4: Initialize DRDB resource.

1. After you have completed the initial resource configuration you can bring up your resource. This is only done once.
2. Ensure the DRBD kernel module is loaded or not using the below command.

```
$ [root@ node1-mgmt]# lsmod | grep -i drbd
```

3. If it is not loaded, find the drbd.ko module using the find command and install it.

```
$ [root@ node1-mgmt]# find / -name drbd.ko
/usr/lib/modules/4.18.0-240.10.1.el8_3.x86_64/weak-updates/drbd90/drbd.ko
/usr/lib/modules/4.18.0-240.el8.x86_64/extra/drbd90/drbd.ko
```

4. Load the DRBD kernel module using the below command.

```
$ [root@ node1-mgmt]# insmod /usr/lib/modules/4.18.0-240.10.1.el8_3.x86_64/weak-
updates/drbd90/drbd.ko
```

5. Ensure the kernel module got loaded using the below command.

```
$ [root@ node1-mgmt]# lsmod | grep -i drbd
drbd                  643072  0
libcrc32c              16384  4 nf_conntrack,nf_nat,xfs,drb
```

6. By convention, /etc/drbd.d/global_common.conf contains the global and common sections of the DRBD configuration, whereas the ".res" files contain one resource section each.

7. Ensure the DRBD volumes are empty. Then initialize the metadata on all the DRBD nodes.

```
$ root@ node1-mgmt]# drbdadm create-md beegfs
initializing activity log
initializing bitmap (57232 KB) to all zero
Writing meta data...
New drbd meta data block successfully created.
```

8. DRBD should be started after sync is achieved. The DRBD initialization process can take several hours.

Wait for the initialization process to finish. Once disks are initialized, on both nodes start the DRBD service simultaneously one after another.

```
$ root@ node1-mgmt]# drbdadm up beegfs
or
$ [root@ node1-mgmt]# systemctl start drbd
● drbd.service - DRBD -- please disable. Unless you are NOT using a cluster manager.
  Loaded: loaded (/usr/lib/systemd/system/drbd.service; enabled; vendor preset:
  disabled)
  Active: active (exited) since Fri 2021-02-19 16:10:05 IST; 1min 37s ago
Main PID: 24456 (code=exited, status=0/SUCCESS)
  Tasks: 0 (limit: 39320)
Memory: 0B
CGroup: /system.slice/drbd.service

Feb 19 16:07:16 node1-mgmt drbd[24456]:      create res: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]:    prepare disk: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]:     prepare net: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]:     prepare net: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]:     adjust disk: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]: attempt to connect: beegfs
Feb 19 16:07:16 node1-mgmt drbd[24456]: ]
Feb 19 16:10:05 node1-mgmt drbd[24456]: WARN: stdin/stdout is not a TTY; using /dev/
console

Feb 19 16:10:05 node1-mgmt drbd[24456]: .

Feb 19 16:10:05 node1-mgmt systemd[1]: Started DRBD -- please disable. Unless you are
NOT using a cluster manager..
```

9. Check the DRBD status on both nodes using the below commands.

```
$ sudo drbdadm status
beegfs role:Secondary
disk:Inconsistent
node2-mgmt role:Secondary
peer-disk:Inconsistent
```

The Inconsistent disk state is expected at this point.

10. Then, on one node execute the following command.

```
$ drbdadm new-current-uuid --clear-bitmap <resource>/<volume>
or
$ drbdsetup new-current-uuid --clear-bitmap <minor>
```

Check DRBDADM status. Now it shows the disks as *UpToDate* (even though the backing devices might be out of sync). You can now create a file system on the disk and start using it.

11. Make one node as primary after the DRBD service is started on both the nodes using the below command.

```
$ [root@ node1-mgmt]# drbdadm primary beegfs
```

12. Check the DRBD status on both nodes using the below commands.

```
$ [root@ node1-mgmt]# drbdadm status
        beegfs role:Primary
        disk:UpToDate
        node2-mgmt role:Secondary
        replication:SyncSource peer-disk: UpToDate
```

Check that both primary and secondary disks appear as Up to Date.

```
$ [root@ node1-mgmt]# systemctl enable drbd
```

13. Create a file system. Mount the volume and write some data on the first node, "node1." You can create ext4/xfs any of the file systems as per your preference.

```
$ [root@ node1-mgmt ~]# mkfs.ext3 /dev/drbd1
$ [root@ node1-mgmt ~]# mount /dev/drbd1 /mnt/beegfsmgmt
$ [root@ node1-mgmt ~]# touch /mnt/beegfsmgmt/testfile
```

14. Let's test the configuration by changing primary mode "node1" to secondary node "node2" to check if the data replication works or not.

15. Unmount the volume drbd1 on the first node, "node1."

```
$ [root@ node1-mgmt ~]# umount /mnt/beegfsmgmt
```

16. Change the secondary mode to primary mode on the second node, "node2."

```
$ [root@ node2-mgmt ~] drbdadm primary beegfs
```

17. Mount the volume and check if the data are available or not.

```
$ [root@ node2-mgmt ~] sudo mount /dev/drbd1 /mnt/beegfsmgmt
$ [root@ node2-mgmt ~]ll /mnt/beegfsmgmt
total 16 drwx------ 2
root root 16384 Feb 24 20:29
lost+found -rw-r--r-- 1 root root 0 Feb 24 20:31 testfile
```

6.4 Setting Up Pacemaker (PCS) Cluster

Pacemaker is a cluster resource manager. It achieves maximum availability for your cluster services and resources by making use of the cluster infrastructure's messaging and membership capabilities to deter and recover from node and resource-level failure. A cluster configured with Pacemaker comprises separate component daemons that monitor cluster membership, scripts that manage the services, and resource management subsystems that monitor the disparate resources. For more details regarding PCS refer to @PCS Overview.

1. If there are problems authenticating and the firewalld daemon is in use, enable the following ports on your firewall.

```
$ firewall-cmd --permanent --add-service=high-availability
$ firewall-cmd --reload
```

2. On all the nodes start the PCSD daemon and allow it to start on reboot. Check the status.

```
$ systemctl start pcsd; systemctl enable pcsd
$ [root@ node1-mgmt]# systemctl status pcsd
● pcsd.service - PCS GUI and remote configuration interface
Loaded: loaded (/usr/lib/systemd/system/pcsd.service; disabled; vendor preset:
disabled)
Active: active (running) since Fri 2021-02-19 17:44:28 IST; 15s ago
Docs: man:pcsd(8)
man:pcs(8)
Main PID: 25750 (pcsd)
Tasks: 1 (limit: 820040)
Memory: 38.0M
CGroup: /system.slice/pcsd.service
└─25750 /usr/libexec/platform-python -Es /usr/sbin/pcsd

Feb 19 17:44:27 node1-mgmt systemd[1]: Starting PCS GUI and remote configuration
interface...
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Starting server...
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Binding socket for address '*'
and port '2224'
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Server is listening
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Notifying systemd we are
running (socket '/run/systemd/notify')
Feb 19 17:44:28 node1-mgmt systemd[1]: Started PCS GUI and remote configuration
interface.
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Config files sync started
Feb 19 17:44:28 node1-mgmt pcsd[25750]: INFO:pcs.daemon:Config files sync skipped, this
host does not seem to be in a cluster of at least 2 nodes
```

3. Set up the password for the Pacemaker cluster. You must provide a password for hacluster. Check that the Pacemaker service is running on both the nodes. If it's not running, then start the service.

```
$ [root@ node1-mgmt ~]# systemctl start pacemaker.service
$ [root@ node1-mgmt ~]# systemctl status pacemaker.service
$ passwd hacluster
```

4. Set up the node names in the host file to reflect the cluster node names. If you prefer, you can use IP addresses instead.

5. Set up communication between machines. This communication can be set from any node, and the argument is the other nodes in the cluster, including the node that you are using.

When prompted, enter the user hacluster and the corresponding password.

Note: This command is being performed:

```
$ [@node1-mgmt] pcs cluster auth node1-mgmt node2-mgmt
Or
$ [@node1-mgmt] pcs host auth node1-mgmt node2-mgmt
```

6. Create the empty cluster and provide the password for hacluster when prompted. This command should also contain the node that the command-line console is currently being used on.

Note: This command is being performed on node1-mgmt, and that node1-mgmt is being included in the given command.

```
$ [@node1-mgmt] pcs cluster setup --name mgmt_cluster node1-mgmt node2-mgmt
```

7. Enable the cluster service to start after reboot.

```
$ [@node1-mgmt] pcs cluster enable --all
```

8. Start the cluster services on all the nodes.

```
$ [@node1-mgmt] pcs cluster start --all
```

9. Display the status of the cluster and make sure that all the nodes are online.

```
$ [@node1-mgmt] pcs cluster status
Cluster name: mgmtcluster
Cluster Summary:
* Stack: corosync
* Current DC: node1-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) - partition with quorum
* Last updated: Thu Feb 23 20:12:00 2021
* Last change:  Thu Feb 22 20:11:21 2021 by root via cibadmin on node1-mgmt
* 2 nodes configured
* 0 resource instances configured
Node List:
* Node node1-mgmt: Online
* Node node2-mgmt: Online
PCSD Status:
node1-mgmt: Online
node2-mgmt: Online
```

6.5 Cluster Resource Creation

Let's create cluster resources and set the priority.

1. Create the floating IP resource for Pacemaker. The IP address must be on the same interface as the rest of the BeeGFS connections. (The max number of characters for the IP resource is 15.)

```
$ pcs resource create <mgmt_IP_resource_name> ocf:heartbeat:IPaddr ip=<mgmt_floating_
IP> iflabel=<IP_alias_name> nic=<network_interface> cidr_netmask=<netmask> flush_
routes=1
$ pcs resource create mgmtd_ip ocf:heartbeat:IPaddr2 ip=192.168.10.30 cidr_netmask=25
nic=enp0s3 iflabel=fl-mgm op monitor interval=30s
```

Check that the IP got added to the network.

```
$ [root@ node1-mgmt ~]# ifconfig -a
        ens31f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9216
        inet 192.168.10.10  netmask 255.255.255.128  broadcast 192.168.10.127
        ether ec:0d:9a:83:25:f2  txqueuelen 1000  (Ethernet)
        RX packets 6881815  bytes 35928736307 (33.4 GiB)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4531133  bytes 766561149 (731.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

        ens31f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9216
        inet 172.168.20.20  netmask 255.255.255.128  broadcast 172.168.20.127
        ether ec:0d:9a:83:25:f3  txqueuelen 1000  (Ethernet)
        RX packets 2549  bytes 294405 (287.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2494  bytes 313075 (305.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

        ens31f0:fl-mum: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9216
        inet 192.168.10.30  netmask 255.255.255.0  broadcast 0.0.0.0
        ether ec:0d:9a:83:25:f2  txqueuelen 1000  (Ethernet)
```

2. Configure DRBD resource in Pacemaker.

```
$ pcs resource create drbd_res0 ocf:linbit:drbd drbd_resource=beegfs drbdconf=/etc/
drbd.conf promotable notify=true op monitor interval=30s
```

3. For better handling of the resources it is sensible to group them in a resource group.

```
$ pcs resource group add gr_beegfs_mgmtd mgmtd-ip

$ pcs constraint colocation add master gr_beegfs_mgmtd with master drbd_res0-clone
INFINITY
```

4. Create the resource for the file system.

```
$ pcs resource create <mgmt_FS_resource_name> ocf:heartbeat:Filesystem
device=<mgmt_block_device> directory=<mgmt_mount_point> fstype=ext3
options=defaults,sync,noatime,nodiratime,journal_checksum op monitor interval=17s

$ pcs resource create fs_rs1 ocf:heartbeat:Filesystem device=/dev/drbd1 directory=/
mnt/beegfsmgmt fstype=ext3 --group=gr_beegfs_mgmtd op monitor interval=17s
```

Note: Execute below steps from 5 to 9 after setting up the BeeGFS management service using the steps provided in section 6.6.

5. Then create the resource for the BeeGFS management service. Ensure BeeGFS management packages are installed on both the management node and the BeeGFS service is running fine on the master node before creating the BeeGFS management resource.

```
$ pcs resource create <mgmt_service_resource_name> systemd:beegfs-mgmtd op monitor
interval=13s

$ pcs resource create beegfs_mgmtd systemd:beegfs-mgmtd --group=gr_beegfs_mgmtd op
monitor interval=13s
```

6. Create a constraint order for all the resources for failover.

```
$ pcs constraint order set drbd_res0-clone mgmtd_ip fs_rs1 beegfs_mgmtd
```

7. Check the PCS cluster status.

```
$ [root@ node1-mgmt]# pcs cluster status
        Cluster Status:
        Cluster Summary:
        * Stack: corosync
```

```
* Current DC: node1-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) - partition
with quorum
* Last updated: Thu Feb 25 15:10:15 2021
* Last change:  Tue Feb 23 14:27:38 2021 by root via cibadmin on node1-mgmt
 * 2 nodes configured
* 5 resource instances configured
Node List:
* Online: [node1-mgmt node2-mgmt]
PCSD Status:
node1-mgmt: Online
node2-mgmt: Online
```

8. Check the PCS resource status.

```
$ [root@ node1-mgmt]# pcs status
        Cluster name: mgmtcluster
        Cluster Summary:
        * Stack: corosync
        * Current DC: node1-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) - partition
        with quorum
        * Last updated: Thu Feb 25 15:14:07 2021
        * Last change:  Tue Feb 23 14:27:38 2021 by root via cibadmin on node1-mgmt
        * 2 nodes configured
        * 5 resource instances configured

        Node List:
        * Online: [node1-mgmt node2-mgmt]

        Full List of Resources:
        * Clone Set: drbd_res0-clone [drbd_res0] (promotable):
        * Masters: [node1-mgmt]
        * Slaves: [node1-mgmt]
        * Resource Group: gr_beegfs_mgmtd:
        * mgmtd_ip      (ocf::heartbeat:IPaddr2):        Started node1-mgmt
        * fs_rs1        (ocf::heartbeat:Filesystem):     Started node1-mgmt
        * beegfs_mgmtd (systemd:beegfs-mgmtd): Started node1-mgmt

        Daemon Status:
        corosync: active/disabled
        pacemaker: active/disabled
        pcsd: active/disabled
```

9. Check the cluster resource constraint ordering.

```
$ [root@ node1-mgmt]# pcs constraint show
        Location Constraints:
        Ordering Constraints:
```

```
        Resource Sets:
        set drbd_res0-clone mgmtd_ip fs_rs1 beegfs_mgmtd
        Colocation Constraints:
                gr_beegfs_mgmtd with drbd_res0-clone (score:INFINITY) (rsc-
                role:Master) (with-rsc-role:Master)
Ticket Constraints.
```

6.6 BeeGFS Management Server Setup

You will set up node1-mgmt and node2-mgmt to run a singular instance of the management service. After you have completed these steps, the services can run on either host. The steps presented here function only in a two-node building block; however, these steps will likely be useful if you set up building blocks that are larger than two nodes.

1. Use the DRBD-mounted partition to store the management data.

```
$ /mnt/beegfsmgmt
```

2. Set up the management location to the newly created directory.

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p <management_data_location>
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p /mnt/beegfsmgmt
```

3. Create the file connInterfacesFile.conf in the config directory.

Note: This file will hold the network interfaces to be used with BeeGFS services. Provide the VIP details in connInterfacesFile.conf file which you created during section 6.4 step 1 (Floating IP creation) cluster resource creation.

```
$ [root@ node1-mgmt]# cat /etc/beegfs/connInterfacesFile.conf
ens31f0:fl-mum
```

4. Edit the BeeGFS configuration file and set the location for connInterfacesFile. You specify which network interface can be used to contact the management service. Create a text file under /etc/beegfs, with the interface name to be used listed one per line. In the following example, ens31f0:fl-mum is used. The absolute path of the file must be specified in the configuration file /etc/beegfs/beegfs-mgmt.conf by using the connInterfacesFile parameter.

```
$ [root@mgmt1 ~]# cat /etc/beegfs/connInterfacesFile1.conf
ens31f0:fl-mum
```

5. Configure the location of the management data.

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p <mgmt_data_location> -c <mgmt_config_
location> -S <beegfs_mgmt_string_nodeID>
```

Example:

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p /mnt/beegfsmgmt/ -c /etc/beegfs/beegfs-
mgmtd.conf -S mgmt-server
```

6. Start the management service and enable it to start at boot.

```
$ systemctl start beegfs-mgmtd.service
$ systemctl enable beegfs-mgmtd.service
$ systemctl status beegfs-mgmtd.service
```

7. Ensure BeeGFS management packages are installed on the second management node. Copy all the management configuration files from the first management node to the second management node.

8. Configure the location of the management data on the second node.

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p <mgmt_data_location> -c <mgmt_config_
location> -S <beegfs_mgmt_string_nodeID>
```

Example:

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p /mnt/beegfsmgmt/ -c /mnt/beegfsmgmt/
beegfs-mgmtd.conf -S mgmt-server
```

9. Initiate PCS cluster failover to the second node and check all the services are getting migrated successfully to the second node and BeeGFS management service is getting started automatically in the nodes as part of the failover process.

To initiate cluster failover from management node1 to node2, put the management node1 in standby mode.

```
$ [root@ node1-mgmt ~]# pcs node standby node1-mgmt
$ [root@ node1-mgmt ~]# pcs cluster status
        Cluster Status:
        Cluster Summary:
        * Stack: corosync
                * Current DC: node2-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) -
                partition with quorum
        * Last updated: Fri Feb 26 07:24:15 2021
        * Last change:  Fri Feb 26 07:24:06 2021 by root via cibadmin on node1-mgmt
        * 2 nodes configured
        * 5 resource instances configured
        Node List:
        * Node node1-mgmt: standby
        * Online: [node2-mgmt]
        PCSD Status:
        node1-mgmt: Online
        node2-mgmt: Online
$ [root@ node1-mgmt ~]# pcs status
        Cluster name: mgmtcluster
        Cluster Summary:
        * Stack: corosync
                * Current DC: node2-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) -
                partition with quorum
        * Last updated: Fri Feb 26 07:24:25 2021
        * Last change:  Fri Feb 26 07:24:06 2021 by root via cibadmin on node1-mgmt
        * 2 nodes configured
        * 5 resource instances configured
        Node List:
        * Node node1-mgmt: standby
        * Online: [node2-mgmt]
        Full List of Resources:
        * Clone Set: drbd_res0-clone [drbd_res0] (promotable):
        * Masters: [node2-mgmt]
```

```
        * Stopped: [node1-mgmt]
        * Resource Group: gr_beegfs_mgmtd:
        * mgmtd_ip      (ocf::heartbeat:IPaddr2):       Started node2-mgmt
        * fs_rs1        (ocf::heartbeat:Filesystem):    Started node2-mgmt
        * beegfs_mgmtd (systemd:beegfs-mgmtd): Started node2-mgmt

        Daemon Status:
        corosync: active/disabled
        pacemaker: active/disabled
        pcsd: active/disabled
```

```
$ [root@ node2-mgmt ~]# drbdadm status
        beegfs role:Primary
        disk:UpToDate
        node1-mgmt connection:Connecting
```

Note: Ensure /mnt/beegfsmgmt/ and connInterfaces files are present in the second node and the node contains the same details as the first management node.

10. Make the first management node online again.

```
$ [root@ node1-mgmt ~]# pcs node unstandby node1-mgmt
$ [root@ node2-mgmt ~]# pcs cluster status
        Cluster Status:
        Cluster Summary:
        * Stack: corosync
                * Current DC: node2-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) -
                partition with quorum
        * Last updated: Fri Feb 26 07:27:12 2021
        * Last change:  Fri Feb 26 07:27:05 2021 by root via cibadmin on node1-mgmt
        * 2 nodes configured
        * 5 resource instances configured
        Node List:
        * Online: [node1-mgmt node2-mgmt]

        PCSD Status:
        Node1-mgmt: Online
        node2-mgmt: Online
```

```
$ [root@ node1-mgmt ~]# pcs status
        Cluster name: mgmtcluster

        Cluster Summary:
        * Stack: corosync
                * Current DC: node2-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) -
                partition with quorum
        * Last updated: Fri Feb 26 07:27:19 2021
        * Last change:  Fri Feb 26 07:27:05 2021 by root via cibadmin on node1-mgmt
```

```
        * 2 nodes configured
        * 5 resource instances configured

        Node List:
        * Online: [node1-mgmt node2-mgmt]

        Full List of Resources:
        * Clone Set: drbd_res0-clone [drbd_res0] (promotable):
        * Masters: [node2-mgmt]
        * Slaves: [node1-mgmt]
        * Resource Group: gr_beegfs_mgmtd:
        * mgmtd_ip     (ocf::heartbeat:IPaddr2):        Started node2-mgmt
        * fs_rs1       (ocf::heartbeat:Filesystem):     Started node2-mgmt
        * beegfs_mgmtd (systemd:beegfs-mgmtd): Started node2-mgmt
        Daemon Status:
        corosync: active/disabled
        pacemaker: active/disabled
        pcsd: active/disabled
```

```
$ [root@ node1-mgmt ~]# drbdadm status
        beegfs role:Secondary
        disk:UpToDate
        node2-mgmt role:Primary
        peer-disk:UpToDate
```

6.7 Fencing Configuration

Fencing or STONITH is the act of confirming that after a node or host fails, it is down. For more details refer to @Stonith Fencing Configuration Details. Imagine that you have a storage building block on node-mgmt1 and node-mgmt2. If node-mgmt1 fails for any undetermined reasons, Pacemaker is set up to failover to the alternate node, node-mgmt2. If node-mgmt1 comes back online and cannot communicate with the cluster, it might assume that node-mgmt2 is down and place the resource onto itself and mount to the same device. If both node-mgmt1 and node-mgmt2 mount to the same device at the same time, it causes data corruption. Pacemaker does a good job of avoiding these situations and they should not happen, but for additional data safety, you must confirm that the node is down for good.

To resolve the preceding issue, use the APC-branded power distribution unit (PDU) fencing agent. In the preceding scenario, node-mgmt2 gets an interrupted signal from node-mgmt1 and then fails the resources (file system, service, and so on) over to itself. Before taking this action, it also triggers the APC PDU and turns off the given port to the bad node or host. With the APC port turned off, it is unlikely that the server will power on. You are much more certain that this machine is in the off state.

Redhat supports multiple STONITH agents. Based on your requirement you can choose any of the configuration agents.

Note: You should always use fencing for production systems.

Test Your System for Fencing Configuration

If you want to test your system before you configure fencing.

1. Turn off the following setting. By default, STONITH is expected to be set up; the resources will not start without it.

```
$ pcs property set stonith-enabled=false
```

2. Verify the Pacemaker resources and groups.

Note: All resources at this point should be started and running on the correct nodes.

```
$ [root@ node1-mgmt]# pcs status
      Cluster name: mgmtcluster
      Cluster Summary:
       * Stack: corosync
      * Current DC: node1-mgmt (version 2.0.4-6.el8_3.1-2deceaa3ae) - partition
      with quorum
      * Last updated: Thu Feb 25 15:14:07 2021
      * Last change:  Tue Feb 23 14:27:38 2021 by root via cibadmin on node1-mgmt
      * 2 nodes configured
      * 5 resource instances configured

      Node List:
      * Online: [node1-mgmt node2-mgmt]

      Full List of Resources:
      * Clone Set: drbd_res0-clone [drbd_res0] (promotable):
      * Masters: [node1-mgmt]
      * Slaves: [node1-mgmt]
      * Resource Group: gr_beegfs_mgmtd:
       * mgmtd_ip     (ocf::heartbeat:IPaddr2):        Started node1-mgmt
      * fs_rs1       (ocf::heartbeat:Filesystem):     Started node1-mgmt
      * beegfs_mgmtd (systemd:beegfs-mgmtd): Started node1-mgmt

      Daemon Status:
      corosync: active/disabled
      pacemaker: active/disabled
      pcsd: active/disabled
```

Note: All servers should have the fencing agent installed.

Install Fencing

Many fencing agents have already been created for use with Pacemaker. You can also create your own. You can find the existing fencing agents with the following commands. For more information, see the RedHat documentation on fencing @Stonith Fencing Configuration Details.

1. Find and install the fencing agent.

```
$ yum search fence-
$ yum install <fencing_agent>
#Example
$ yum install fence-agents-apc
```

2. Review the list of fencing agents that have been installed and are ready for use.

```
$ pcs stonith list
```

3. Read the descriptions of the fencing agents.

```
$ pcs stonith describe [fencing-agent name]
#Example
$ pcs stonith describe fence_apc
```

Set Up Fencing with APC

Most of the recent articles seem to suggest the use of APC-branded PDUs if they are available. The reason is that they are an entirely separate component, away from any hosts. Some other fencing agents, such as integrated Dell Remote Access Controller (iDRAC)/Intelligent Platform Management Interface (IPMI), and XVM, are also available. You also have the option to create your own custom fencing agent.

Note: Some of these confirmation efforts and tests can cause corruption; do not test with real and/or valuable information.

Easy and simple success with APC-branded PDUs has been well documented, and many Pacemaker fencing articles and guides recommend the use of these PDUs.

1. Create the fencing object.

```
$ pcs stonith create <fencing object name> <fence object type> <fields>
```

Example: Single APC fencing agent to a single cluster node.

```
$ pcs stonith create myapc fence_apc pcmk_host_list=”nodeMM1” pcmk_host_map=”node-mgmt1:12” ipaddr=”10.10.10.10” login=”apc” passwd=”apc_pass”
```

```
# Create a fencing agent called 'myapc'
# Type fence_apc
# Host nodes 'node-mgmt1.
# Map nodeMM1to port 12
# IP address of the APC is 10.10.10.10
# Login username is apc
# Password of apc is apc_pass
```

Example: Multiple nodes to multiple ports.

```
$ pcs stonith create myapc fence_apc pcmk_host_list=” node-mgmt1,node-mgmt2” pcmk_host_map=” node-mgmt1:15; node-mgmt2:17” ipaddr=”myAPCAddress” login=”apc” passwd=”apc_pass”
```

```
# Create a fencing agent called 'myapc'
# Type fence_apc
# Host nodes 'node-mgmt1, node-mgmt2'
# Create a map to the nodes:
nodeMM1 is on port 15
nodeMM2 is on port 17
# IP address of the APC is 10.10.10.10
# Login username is apc
# Password of apc is apc_pass
```

Example: Redundant power.

This setup is the same as in the second example, but with two APC agents.

```
$ pcs stonith create myapc fence_apc pcmk_host_list=” node-mgmt1,node-mgmt2” pcmk_host_map=” node-mgmt1:15; node-mgmt2:17” ipaddr=”myAPCAddress” login=”apc” passwd=”apc_pass”
$ pcs stonith create myapc2 fence_apc pcmk_host_list=” node-mgmt1,node-mgmt2” pcmk_host_map=” node-mgmt1:1; node-mgmt2:2” ipaddr=”myOtherAPCaddress” login=”apc” passwd=”apc_pass”
```

2. Connect the cluster node with the two fencing agents and put them on the same Pacemaker fencing topology. If you want to connect multiple fencing agents to each cluster node, Pacemaker's fencing topology is a hierarchy of fencing agents. In this example, both were added to the same level, which makes Pacemaker use both at the same time.

```
$ pcs stonith level add 1 nodeMM1 myapc1,myapc2
```

3. List to see whether the fencing agents are working.

```
$ pcs stonith
```

4. Confirm that the fencing covers the nodes.

```
$ pcs stonith confirm <nodeId>
#Example
$ pcs stonith confirm node-mgmt1
```

Enter y to accept.

5. Start the cluster on all the nodes.

```
$ pcs cluster start --all
```

6. Test the node with the built-in fencing agent tester. If the --off flag is triggered, the fencing shuts down completely and must be manually turned back on. The default behavior is to reboot rather than shut down.

Configure Fencing Settings

1. If you are using opt-in location constraints, you must confirm that the fencing agents can be run on the other nodes in the cluster.

```
$ pcs property show symmetric-cluster -> false
```

2. For opt-in location constraints only, allow fencing agents to be run on all the nodes.

```
$ pcs constraint location myApcFence prefers node-mgmt1=0 node-mgmt2=0
```

3. Enable STONITH for the cluster.

```
$ pcs property set stonith-enabled=true
```

4. Based on your need you can set up quorum rules on the cluster nodes to handle failure. Its primary use is to allow a cluster to sustain more node failures than standard quorum rules allow. For more details refer to @PCS_quorum_rules. The quorum rules provides an auto tie-breaking setting, which automatically breaks a tie if there is a split vote on which resource must go to which system. For example, if you have a two-node cluster, Pacemaker cannot decide which of the two nodes is given the resource or resource group. This situation is referred to as "split brain."

By default, the resource is not loaded on either node until further instructions are given.

```
$ pcs cluster stop --all
$ pcs quorum update auto_tie_breaker=1
$ pcs cluster start --all
$ pcs quorum
```

Hence using quorum rules settings, you can define rules for your cluster to handle the resource migration effectively in case of failure using the "auto_tie_breaker", "ffsplit", and "lms" policies.

5. Set the STONITH action to off. For production systems, Western Digital recommends that you manually verify that the system is off. Rebooting might resolve the issue, but you should diagnose why the fencing event was triggered. If the system reboots without knowing whether the node is truly operational, it could be potentially dangerous for the data.

By default, when fencing occurs the node reboots.

```
$ pcs property set stonith-action=off
```

6. Synchronize the settings to all nodes in the cluster.

```
$ [root@node-mgmt1 ~]# pcs cluster sync
        node-mgmt1: Succeeded
        node-mgmt2: Succeeded
```

Test APC Fencing

Not every event triggers a fencing response. Here are some gathered results about events that activate or do not activate fencing.

- Pulling the power cable
  - Fencing is activated.
- Pulling the Ethernet cable
  - Fencing is activated.
- Stopping the service
  - Fencing is not activated.
- Kernel panic
  - Fencing is activated.

Test the node with real situations, such as pulling the power cable or disconnecting the network. Western Digital recommends that you try the following tests physically and that you not rely on any SSH or other remote mechanisms.

1. Pull the power cable.

2. Pull the network cable.

3. Trigger a kernel panic on the system that you want to be turned off.

```
$ echo c > /proc/sysrq-trigger
```

4. Clean fencing failures from PCS status.

```
$ stonith_admin --cleanup --history=node-mgmt1
```

6.8 Metadata Configuration Details

1. Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo https://www.beegfs.io/release/beegfs_7.2/dists/beegfs-rhel8.repo
```

2. Install metadata packages.

```
$ yum install beegfs-meta libbeegfs-ib
```

3. Create directories which you want to use as mount points for the metadata.

```
$ mkdir -p /data/beegfs/beegfs_meta1 /data/beegfs/beegfs_meta2
```

4. Format each volume where you want to store the metadata.

```
$ mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype <mapped volume>
$ mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype /dev/nvme0n1
$ mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype /dev/nvme1n1
```

5. Mount each mapped volume.

```
$ mount -onoatime,nodiratime,nobarrier <mapped volume> <mount point>.
$ mount -onoatime,nodiratime,nobarrier /dev/nvme0n1 /data/beegfs/beegfs_meta1
$ mount -onoatime,nodiratime,nobarrier /dev/nvme1n1 /data/beegfs/beegfs_meta2
```

6. Add each mount point to /etc/fstab to automatically mount when the server boots.

7. As you are configuring two metadata instances here on the same server for multimode configuration, you need to run the following steps.

```
$ mkdir /etc/beegfs/inst1.d
$ mkdir /etc/beegfs/inst2.d
$ cp /etc/beegfs/beegfs-meta.conf /etc/beegfs/inst1.d/
$ cp /etc/beegfs/beegfs-meta.conf /etc/beegfs/inst2.d/
$ vi /etc/beegfs/inst1.d/beegfs-meta.conf # Adapt “connMetaPort*”, “logStdFile”, etc.
$ vi /etc/beegfs/inst2.d/beegfs-meta.conf # Adapt “connMetaPort*”, “logStdFile”, etc.
```

8. You specify which network interface or interfaces other services can use to contact the metadata services. Create a text file under /etc/beegfs, with the interface names to be used listed one per line. In the following example, ens31f0, and ens31f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/inst1.d/beegfs-meta.conf and /etc/beegfs/inst2.d/beegfs-meta.conf by using the connInterfacesFile parameter.

```
$ [root@meta1 ~]# cat /etc/beegfs/connInterfacesFile1.conf
ens31f0
ens31f1
$ [root@meta1 ~]# cat /etc/beegfs/inst1.d/beegfs-meta.conf | grep connInterfacesFile
connInterfacesFile           = /etc/beegfs/connInterfacesFile1.conf

$ [root@meta1 ~]# cat /etc/beegfs/inst2.d/beegfs-meta.conf | grep connInterfacesFile
connInterfacesFile           = /etc/beegfs/connInterfacesFile1.conf
```

9. Configure metadata service by running the following steps.

```
$ /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst1.d/beegfs-meta.conf -p
/data/beegfs/beegfs_meta1 -s 11 -S meta1-inst1 -m node01
```

To add a second metadata target on the same machine change the instance path, id, and name and run as the below command.

```
$ /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst1.d/beegfs-meta.conf -p /
data/beegfs/beegfs_meta1 -s 11 -S meta1-inst1 -m 192.168.10.30
$ /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst2.d/beegfs-meta.conf -p /
data/beegfs/beegfs_meta2 -s 12 -S meta1-inst2 -m 192.168.10.30
```

10. Start the metadata service and enable it to start at boot.

```
$ [root@meta1 ~]# systemctl start beegfs-meta@inst1
$ [root@meta1 ~]# systemctl enable beegfs-meta@inst1
$ [root@meta1 ~]# systemctl status beegfs-meta@inst1
● beegfs-meta@inst1.service - BeeGFS Metadata Server (multimode)
        Loaded: loaded (/usr/lib/systemd/system/beegfs-meta@.service; enabled; vendor
        preset: disabled)
Active: active (running) since Mon 2021-01-25 16:22:32 IST; 2 days ago
Docs: http://www.beegfs.com/content/documentation/
Main PID: 21504 (beegfs-meta/Mai)
Tasks: 203 (limit: 820084)
Memory: 754.2M
```

```
CGroup: /system.slice/system-beegfs\x2dmeta.slice/beegfs-meta@inst1.service
        └─21504 /opt/beegfs/sbin/beegfs-meta cfgFile=/etc/beegfs/inst1.d/beegfs-meta.
        conf runDaemonized=false

Jan 25 16:22:32 meta1 systemd[1]: Started BeeGFS Metadata Server (multimode).
$ [root@meta1 ~]# systemctl start beegfs-meta@inst2

$ [root@meta1 ~]# systemctl enable beegfs-meta@inst2

$ [root@meta1 ~]# systemctl status beegfs-meta@inst2
● beegfs-meta@inst2.service - BeeGFS Metadata Server (multimode)
Loaded: loaded (/usr/lib/systemd/system/beegfs-meta@.service; disabled; vend>
Active: active (running) since Wed 2021-01-27 18:34:53 IST; 20h ago
Docs: http://www.beegfs.com/content/documentation/
Main PID: 135374 (beegfs-meta/Mai)
Tasks: 83 (limit: 820084)
Memory: 29.9M
CGroup: /system.slice/system-beegfs\x2dmeta.slice/beegfs-meta@inst2.service
└─135374 /opt/beegfs/sbin/beegfs-meta cfgFile=/etc/beegfs/inst2.d/be>

Jan 27 18:34:53 meta1 systemd[1]: Started BeeGFS Metadata Server
```

11. Repeat above steps on the node which you want to configure as a metadata node.

6.9 Storage Server Configuration Details

In BeeGFS, multi-mode is a way to run multiple instances of one type of service on one server. This is useful if more than one BeeGFS file system/storage target instance or more than one metadata target instance is configured on a single physical server. Please take into account that, when using multi-mode within a single file system, BeeGFS will treat them as individual machines.

For multi-mode, you will need to create a separate configuration file for the other daemon instance, using different network ports, a different storage directory, a different log file, and so on. If the second daemon instance on a machine should become part of the same file system instance (i.e., it registers at the same management daemon as the first daemon instance on this machine), then you would also need to set a different NodeID manually for the second daemon instance. Follow the below steps to set up multi-mode storage servers.

1. Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo https://www.beegfs.io/release/beegfs_7.2/
dists/beegfs-rhel8.repo
```

2. Install storage packages.

```
$ yum install beegfs-storage libbeegfs-ib # storage service; libbeegfs-ib is only
required for RDMA
```

3. Create directories which you want to use as mount points for the storage servers.

```
$ mkdir -p /mnt/myraid21/beegfs_storage
$ mkdir -p /mnt/myraid22/beegfs_storage
```

4. Format each mapped volume: XFS SSD fs creation.

```
$ mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k <mapped volume>
```

Note: "su" is the segment size of the volume.

```
$ mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k /dev/dm-3
$ mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k /dev/dm-4
```

5. Mount each mapped volume.

```
$ mount
-onoatime,nodiratime,logbufs=8,logbsize=256k,largeio,inode64,swalloc,allocsize=131072k,
<mapped volume> <mount point>.
$ mount
-onoatime,nodiratime,logbufs=8,logbsize=256k,largeio,inode64,swalloc,allocsize=131072k
/dev/dm-3 /mnt/myraid21/beegfs_storage
$ mount
-onoatime,nodiratime,logbufs=8,logbsize=256k,largeio,inode64,swalloc,allocsize=131072k
/dev/dm-3 /mnt/myraid22/beegfs_storage
```

6. Add each mount point to /etc/fstab to automatically mount when the server boots.

7. As you are configuring two storage instances here on the same server for multimode configuration, you need to run the following steps.

```
$ mkdir /etc/beegfs/inst1.d
$ mkdir /etc/beegfs/inst2.d
$ cp /etc/beegfs/beegfs-storage.conf /etc/beegfs/inst1.d/
$ cp /etc/beegfs/beegfs-storage.conf /etc/beegfs/inst2.d/
$ vi /etc/beegfs/inst1.d/beegfs-storage.conf # Adapt “connStoragePort*”,
“logStdFile”, etc.
$ vi /etc/beegfs/inst2.d/beegfs-storage.conf # Adapt “connStoragePort*”,
“logStdFile”, etc.
```

Edit the network port details, storage directory path, and log file path details for the second storage instance in the above files.

8. You specify which network interface or interfaces other services can use to contact the storage services. Create a text file under /etc/beegfs, with the interface names to be used listed one per line. In the following example, ens2f0 and ens2f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/inst1.d/beegfs-storage.conf and /etc/beegfs/inst2.d/beegfs-storage.conf by using the connInterfacesFile parameter.

```
$ [root@ storage1 ~]# cat /etc/beegfs/connInterfacesFile.conf
ens2f0
ens2f1
$ [root@ storage1 ~]# cat /etc/beegfs/inst1.d/beegfs-storage.conf | grep
connInterfacesFile
connInterfacesFile = /etc/beegfs/connInterfacesFile.conf
$ [root@ storage1 ~]# cat /etc/beegfs/inst2.d/beegfs-storage.conf | grep
connInterfacesFile
connInterfacesFile = /etc/beegfs/connInterfacesFile.conf
```

9. Configure the storage service by running the following steps.

```
$ /opt/beegfs/sbin/beegfs-setup-storage -c /etc/beegfs/inst1.d/beegfs-storage.conf
-p /mnt/myraid11/beegfs_storage -s 31 -S stor1-inst1 -i 311 -m node01
```

To add a second storage target on this same machine, change the instance path, id, and name and run the below command.

```
$ /opt/beegfs/sbin/beegfs-setup-storage -c /etc/beegfs/inst1.d/beegfs-storage.conf
-p /mnt/myraid21/beegfs_storage -s 21 -S stor1-inst1 -i 211 -m 192.168.10.30

$ /opt/beegfs/sbin/beegfs-setup-storage -c /etc/beegfs/inst2.d/beegfs-storage.conf
-p /mnt/myraid22/beegfs_storage -s 22 -S stor1-inst2 -i 221 -m 192.168.10.30
```

10. Start the storage service and enable it to start at boot.

```
$ root@storage1 ~]# systemctl start beegfs-storage@inst1

$ root@ storage1 ~]# systemctl enable beegfs-storage@inst1

$ [root@ storage1 ~]# systemctl status beegfs-storage@inst1
● beegfs-storage@inst1.service - BeeGFS Storage Server (multimode)
Loaded: loaded (/usr/lib/systemd/system/beegfs-storage@.service; disabled; vendor
preset: disabled)

Active: active (running) since Mon 2021-01-25 16:22:34 IST; 2 days ago
Docs: http://www.beegfs.com/content/documentation/
Main PID: 13414 (beegfs-storage/)

Tasks: 89 (limit: 818894)
Memory: 467.2M
CGroup: /system.slice/system-beegfs\x2dstorage.slice/beegfs-storage@inst1.service
        └─13414 /opt/beegfs/sbin/beegfs-storage cfgFile=/etc/beegfs/inst1.d/beegfs-
        storage.conf runDaemonized=false

Jan 25 16:22:34 storage1 systemd[1]: Started BeeGFS Storage Server (multimode).
[root@ storage1 ~]#

$ root@ storage1 ~]# systemctl start beegfs-storage@inst2

$ root@ storage1 ~]# systemctl enable beegfs-storage@inst2

# [root@ storage1 ~]# systemctl status beegfs-storage@inst2
● beegfs-storage@inst2.service - BeeGFS Storage Server (multimode)
Loaded: loaded (/usr/lib/systemd/system/beegfs-storage@.service; enabled; vendor
preset: disabled)
Active: active (running) since Mon 2021-01-25 16:22:34 IST; 2 days ago
Docs: http://www.beegfs.com/content/documentation/
Main PID: 13592 (beegfs-storage/)
Tasks: 89 (limit: 818894)
Memory: 376.3M
CGroup: /system.slice/system-beegfs\x2dstorage.slice/beegfs-storage@inst2.service
        └─13592 /opt/beegfs/sbin/beegfs-storage cfgFile=/etc/beegfs/inst2.d/beegfs-
        storage.conf runDaemonized=false

Jan 25 16:22:34 storage1 systemd[1]: Started BeeGFS Storage Server (multimode).
[root@ storage1 ~]#
```

11. Repeat the above steps on all the storage nodes.

6.10 Client Server Configuration Details

1. Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo https://www.beegfs.io/release/beegfs_7.2/dists/beegfs-rhel8.repo
```

2. Install client packages.

```
$ yum install beegfs-client beegfs-helperd beegfs-utils
```

3. Client kernel module auto build: To enable support for remote direct memory access (RDMA) based on the OFED ibverbs API, the corresponding BeeGFS client kernel module build parameters need to be added. If you installed separate OFED kernel modules, add the OFED_INCLUDE_PATH:

```
$ ssh clientnode
$ vi /etc/beegfs/beegfs-client-autobuild.conf
$ buildArgs=-j8  OFED_INCLUDE_PATH=/usr/src/ofa_kernel/default/include
```

4. Afterwards, force a rebuild of the client kernel module.

```
$ /etc/init.d/beegfs-client rebuild
```

5. The client mount directory is defined in a separate configuration file. This file will be used by the beegfs-client service start-up script. By default, BeeGFS will be mounted to /mnt/beegfs. Thus, you need to perform this step only if you want to mount the file system to a different location.

```
$ ssh root@clientnode
$ vi /etc/beegfs/beegfs-mounts.conf
```

The first entry defines the mount directory. The second entry refers to the corresponding config file for this mountpoint.

Else create a mount point as below where BeeGFS data will get mounted.

```
$ mkdir -p /mnt/beegfs
```

6. You need to specify which network interface or interfaces other services can use to contact the storage services. Create a text file under /etc/beegfs, with the interface names to be used listed one per line. In the following example, ens31f0, and ens31f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/ and /etc/beegfs/beegfs-client.conf by using the connInterfacesFile parameter.

```
$ [root@client1 ~]# cat /etc/beegfs/connInterfacesFile.conf
ens31f0
ens31f1
```

7. The client needs to know where the management service is running.

```
$ /opt/beegfs/sbin/beegfs-setup-client -m node01
$ /opt/beegfs/sbin/beegfs-setup-client -m 192.168.10.30
```

8. Start the client services.

```
$ [root@client1 ~]# systemctl start beegfs-helperd.service
$ [root@client1 ~]# systemctl enable beegfs-helperd.service
$ [root@client1 ~]# systemctl status beegfs-helperd.service
● beegfs-helperd.service - BeeGFS Helperd
Loaded: loaded (/usr/lib/systemd/system/beegfs-helperd.service; enabled; vendor preset: disabled)
Active: active (running) since Wed 2021-01-27 11:34:02 IST; 1 day 4h ago
```

```
Docs: http://www.beegfs.com/content/documentation/
Main PID: 23064 (beegfs-helperd/)
Tasks: 5 (limit: 39320)
Memory: 2.8M
CGroup: /system.slice/beegfs-helperd.service
└─23064 /opt/beegfs/sbin/beegfs-helperd cfgFile=/etc/beegfs/beegfs-helperd.conf
runDaemonized=false

Jan 27 11:34:02 client1 systemd[1]: Started BeeGFS Helperd.
[root@client1 ~]#

$ [root@client1 ~]# systemctl start beegfs-client.service
$ [root@ client1 ~]# systemctl enable beegfs-client.service

$ [root@ client1 ~]# systemctl status beegfs-client.service
● beegfs-client.service - Start BeeGFS Client
Loaded: loaded (/usr/lib/systemd/system/beegfs-client.service; enabled; vendor
preset: disabled)
Active: active (exited) since Wed 2021-01-27 11:34:28 IST; 1 day 4h ago
Process: 23123 ExecStart=/etc/init.d/beegfs-client start (code=exited, status=0/
SUCCESS)
Main PID: 23123 (code=exited, status=0/SUCCESS)

Jan 27 11:34:28 client1 systemd[1]: Starting Start BeeGFS Client...
Jan 27 11:34:28 client1 beegfs-client[23123]: Starting BeeGFS Client:
Jan 27 11:34:28 client1 beegfs-client[23123]: - Loading BeeGFS modules
Jan 27 11:34:28 client1 beegfs-client[23123]: - Mounting directories from /   etc/
beegfs/beegfs mounts.conf
Jan 27 11:34:28 client1 systemd[1]: Started Start BeeGFS Client.
[root@ client1 ~]#
```

9. Repeat the above steps in all the client nodes.

6.11 BeeGFS Configuration Status Details

Check BeeGFS system configuration details using the below commands.

```
$ [root@client1 ~]# beegfs-df
METADATA SERVERS:
TargetID  Cap. Pool    Total       Free        %       ITotal      IFree     %
========  ==========  =========  =========   ====    ======    ======    ====
11        normal      1340.7GiB   1340.4GiB   100%    937.7M    937.6M    100%
12        normal      1340.7GiB   1340.4GiB   100%    937.7M    937.6M    100%
```

```
STORAGE TARGETS:
TargetID  Cap. Pool   Total       Free        %      ITotal    IFree     %
========  ==========  ==========  ==========  ====   ======    ======    ====
 211      normal      18625.0GiB  14146.4GiB  76%    1953.2M   1953.2M   100%
 221      normal      18625.0GiB  14081.1GiB  76%    1953.2M   1953.2M   100%
 311      normal      18625.0GiB  14102.3GiB  76%    1953.2M   1953.2M   100%
 321      normal      18625.0GiB  14042.8GiB  75%    1953.2M   1953.2M   100%
 411      normal      18625.0GiB  14081.0GiB  76%    1953.2M   1953.2M   100%
 421      normal      18625.0GiB  13924.2GiB  75%    1953.2M   1953.2M   100%
 511      normal      18625.0GiB  14084.3GiB  76%    1953.2M   1953.2M   100%
 521      normal      18625.0GiB  14078.1GiB  76%    1953.2M   1953.2M   100%
```

```
$ [root@ client1 ~]# beegfs-net
mgmt_nodes
=============
mgmt [ID: 1]
Connections: TCP: 1 (192.168.10.30:8008);

meta_nodes
=============
meta1-inst1 [ID: 11]
Connections: RDMA: 1 (192.168.10.10:8005);
meta1-inst2 [ID: 12]
Connections: <none>

storage_nodes
=============
stor1-inst1 [ID: 21]
Connections: RDMA: 1 (192.168.10.11:8003);
stor1-inst2 [ID: 22]
Connections: RDMA: 1 (192.168.10.11:8005);
stor2-inst1 [ID: 31]
Connections: RDMA: 1 (192.168.10.12:8003);
stor2-inst2 [ID: 32]
Connections: RDMA: 1 (192.168.10.12:8005);
stor3-inst1 [ID: 41]
Connections: RDMA: 1 (192.168.10.13:8003);
stor3-inst2 [ID: 42]
Connections: RDMA: 1 (192.168.10.13:8005);
stor4-inst1 [ID: 51]
Connections: RDMA: 1 (192.168.10.14:8003);
stor4-inst2 [ID: 52]
 Connections: RDMA: 1 (192.168.10.14:8005);

$ [root@ client1 ~]# beegfs-check-servers
```

```
Management
==========
mgmt [ID: 1]: reachable at 192.168.10.30:8008 (protocol: TCP)

Metadata
==========
meta1-inst1 [ID: 11]: reachable at 192.168.10.10:8005 (protocol: RDMA)
meta1-inst2 [ID: 12]: reachable at 192.168.10.10:8003 (protocol: RDMA)

Storage
==========
stor1-inst1 [ID: 21]: reachable at 192.168.10.11:8003 (protocol: RDMA)
stor1-inst2 [ID: 22]: reachable at 192.168.10.11:8005 (protocol: RDMA)
stor2-inst1 [ID: 31]: reachable at 192.168.10.12:8003 (protocol: RDMA)

stor2-inst2 [ID: 32]: reachable at 192.168.10.12:8005 (protocol: RDMA)
stor3-inst1 [ID: 41]: reachable at 192.168.10.13:8003 (protocol: RDMA)

stor3-inst2 [ID: 42]: reachable at 192.168.10.13:8005 (protocol: RDMA)

stor4-inst1 [ID: 51]: reachable at 192.168.10.14:8003 (protocol: RDMA)
stor4-inst2 [ID: 52]: reachable at 192.168.10.14:8005 (protocol: RDMA)

$ [root@client1 ~]# beegfs-ctl --listnodes --nodetype=storage --details
stor1-inst1 [ID: 21]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor1-inst2 [ID: 22]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor2-inst1 [ID: 31]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor2-inst2 [ID: 32]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor3-inst1 [ID: 41]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor3-inst2 [ID: 42]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor4-inst1 [ID: 51]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
```

```
stor4-inst2 [ID: 52]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)

Number of nodes: 8
$ [root@ client1 ~]# beegfs-ctl --listnodes --nodetype=meta --details
meta1-inst1 [ID: 11]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens31f0(RDMA) ens31f0(TCP) ens31f1(RDMA) ens31f1(TCP)
meta1-inst2 [ID: 12]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens31f0(RDMA) ens31f0(TCP) ens31f1(RDMA) ens31f1(TCP)

Number of nodes: 2
Root: 11

$ [root@client1 ~]# beegfs-ctl --listnodes --nodetype=client --details
3544-600EA3CA-client1 [ID: 5]
Ports: UDP: 8004; TCP: 0
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(TCP) ens2f1(RDMA)
330F-600EA415- client2 [ID: 6]
Ports: UDP: 8004; TCP: 0
Interfaces: ens7f0(RDMA) ens7f0(TCP) ens7f1(TCP) ens7f1(RDMA)
2F6E-600EA418- client3 [ID: 7]
Ports: UDP: 8004; TCP: 0
Interfaces: ens7f0(RDMA) ens7f0(TCP) ens7f1(TCP) ens7f1(RDMA)
8A61-600EA41B-client4 [ID: 8]
Ports: UDP: 8004; TCP: 0

Interfaces: ens1f0(RDMA) ens1f0(TCP) ens1f1(TCP) ens1f1(RDMA)

Number of nodes: 4

$ [root@client1 ~]# beegfs-ctl --liststoragepools
Pool ID   Pool Description                     Targets                Buddy
Groups
======= ================== ============================ ============================
1       Default 211,221,311,321,411,421,511, 100,200,300,400
        521
```

6.12 BeeGFS Log Details

Check if log files contain any error messages.

```
$ less /var/log/beegfs-mgmtd.log
$ less /var/log/beegfs-meta.log
$ less /var/log/beegfs-storage.log
$ less /var/log/beegfs-client.log
```

7. Tuning Parameters to Improve Performance

7.1 Storage Server Tuning: Refer to @StorageServerTuning for more details.

1. Add the corresponding commands to /etc/rc.local, as seen in the example below. Use /etc/sysctl.conf or create udev rules to reapply them automatically when the machine boots.

```
echo 5 > /proc/sys/vm/dirty_background_ratio
echo 20 > /proc/sys/vm/dirty_ratio
echo 50 > /proc/sys/vm/vfs_cache_pressure
echo 262144 > /proc/sys/vm/min_free_kbytes
echo 1 > /proc/sys/vm/zone_reclaim_mode

echo always > /sys/kernel/mm/transparent_hugepage/enabled
echo always > /sys/kernel/mm/transparent_hugepage/defrag

devices=(dm-3 dm-4)
for dev in "${devices[@]}"
do
echo deadline > /sys/block/${dev}/queue/scheduler
echo 2048 > /sys/block/${dev}/queue/nr_requests
echo 4096 > /sys/block/${dev}/queue/read_ahead_kb
echo 64 > /sys/block/${dev}/queue/max_sectors_kb
done
```

2. On the server in /etc/beegfs/inst1.d/beegfs-storage.conf and /etc/beegfs/inst2.d/beegfs-storage.conf.

```
$ tuneNumWorkers=80
$ tuneBindToNumaZone=0
```

Note: Change the tuneNumWorkers value as per the CPU core available in your system. You can change the value to 4x or 8x of the available CPU core and tune it to achieve better system performance.

7.2 Metadata Server Tuning: Refer to @Metadata_tuning for more details.

1. In order to keep them permanently, you could add the corresponding commands to /etc/rc.local, as seen in the example below. Use /etc/sysctl.conf or create udev rules to reapply them automatically when the machine boots.

```
#!/bin/bash
echo 5 > /proc/sys/vm/dirty_background_ratio
echo 20 > /proc/sys/vm/dirty_ratio
echo 50 > /proc/sys/vm/vfs_cache_pressure

echo 262144 > /proc/sys/vm/min_free_kbytes
echo 1 > /proc/sys/vm/zone_reclaim_mode

echo always > /sys/kernel/mm/transparent_hugepage/enabled
echo always > /sys/kernel/mm/transparent_hugepage/defrag
devices=(nvme0n1 nvme1n1)
for dev in "${devices[@]}"
```

```
do
echo deadline > /sys/block/${dev}/queue/scheduler
echo 128 > /sys/block/${dev}/queue/nr_requests
echo 128 > /sys/block/${dev}/queue/read_ahead_kb
echo 256 > /sys/block/${dev}/queue/max_sectors_kb
done
```

7.3 System BIOS Settings

```
$ echo performance | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor >/
dev/null
```

Each metadata server establishes multiple connections to each of the other servers to enable more parallelism on the network level by having multiple requests in flight to the same server. The tuning option tuneNumWorkers in /etc/beegfs/beegfs-meta.conf can be used to configure the number of simultaneous connections.

Note: Change the tuneNumWorkers value as per the CPU core available in your system.

7.4 Client Tuning

1. On the client in **/etc/beegfs/beegfs-client.conf**.

```
$ connRDMABufSize=32768
$ connRDMABufNum=70
$ connMaxInternodeNum=64
```

Note: Change the connMaxInternodeNum value as per the CPU core available in your system.

2. Increase the chunksize from standard 512k to 1M.

```
$ beegfs-ctl --setpattern --chunksize=1M --numtargets=4 /mnt/beegfs
New chunksize: 1048576
New number of storage targets: 4
Path:
Mount: /mnt/beegfs
```

Note: Based on the chunksize and targets you want to stripe the data and change the values.

3. On the client in **/etc/beegfs/beegfs-client.conf** increase the cache buffer size on all client nodes.

```
$ tuneFileCacheBufSize = 2097152
```

4. Restart all the BeeGFS services so that all the tuning parameter changes will get reflected across all servers.

5. **Communication Retries:** BeeGFS clients support a time-limited retry mode for communications, which will keep on trying to complete the I/O operation in case a server is temporarily unreachable. The connCommRetrySecs option in /etc/beegfs/beegfs-client.conf sets the time limit for communication retries.

8. BeeGFS Buddy Mirroring Configuration

BeeGFS supports mirroring of data across buddy groups. BeeGFS provides support for metadata and file contents mirroring. Mirroring capabilities are integrated into the normal BeeGFS services, so that no separate services or third-party tools are needed. Both types of mirroring (metadata mirroring and file contents mirroring) can be used independent of each other. Refer to @BuddyGroups and @MetaDataMirror for more details.

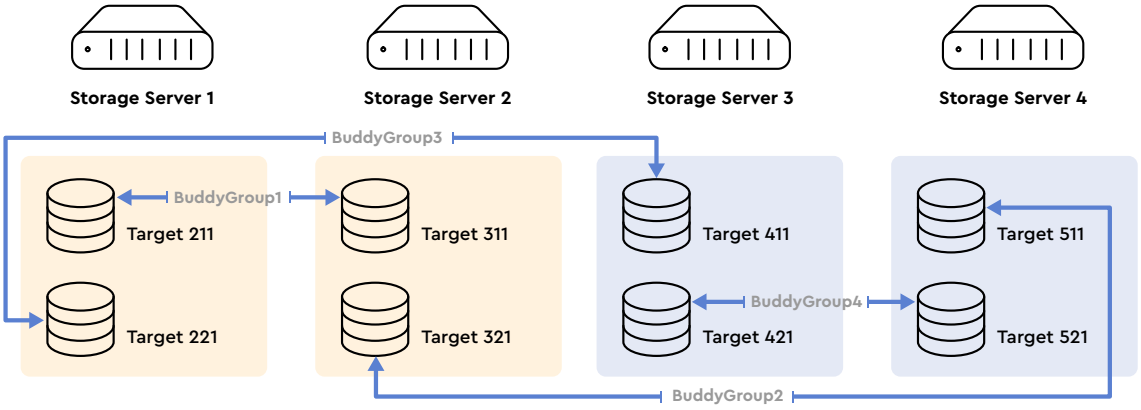


Figure 4: BeeGFS Storage Buddy Mirroring Configuration – 4 Storage Servers with 2 Targets per Server

8.1 BeeGFS Storage Buddy Group Configuration Details

Defining Storage Buddy Mirror Groups Manually

Manual definition of mirror buddy groups can be useful if you want to set custom group IDs or if you want to make sure that the buddies are in different failure domains. Manual definition of mirror buddy groups is done with the beegfs-ctl tool. By using the following command, you can create a buddy group with the ID 100, consisting of targets 1 and 2.

```
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=1 --secondary=2
--groupid=100
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=211 --secondary=311
--groupid=100
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=321 --secondary=511
--groupid=200
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=411 --secondary=221
--groupid=300
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=521 --secondary=421
--groupid=400
```

8.2 List-Defined Buddy Mirrored Groups

```
$ beegfs-ctl --listmirrorgroups --nodetype=storage
$ root@client1 ~]# beegfs-ctl --listmirrorgroups --nodetype=storage
BuddyGroupID         PrimaryTargetID         SecondaryTargetID
============         ===============         =================
100                  211                     311
200                  321                     511
300                  411                     221
400                  521                     421
[root@client1 ~]#
$ [root@client1 ~]# beegfs-ctl --liststoragepools
```

```
Pool ID    Pool Description    Targets                      Buddy Groups
=======    ==================  ===========================  ==============
1          Default             211,221,311,321,411,421,511, 100,200,300,400
```

```
$ [root@client1 ~]# beegfs-ctl --listtargets --mirrorgroups
MirrorGroupID   MGMemberType    TargetID        NodeID
=============    =============    =============   =============
100              primary         211             21
100              secondary       311             31
200              primary         511             51
200              secondary       321             32
300              primary         411             41
300              secondary       221             22
400              primary         521             52
400              secondary       421             42
```

8.3 Display Target States

```
$ beegfs-ctl --listmirrorgroups --nodetype=storage
$ [root@client1 ~]# beegfs-ctl --listmirrorgroups --nodetype=storage
BuddyGroupID    PrimaryTargetID    SecondaryTargetID
=============   =================  =============
100             211                311
200             321                511
300             411                221
400             521                42
$ [root@client1 ~]#  beegfs-ctl --listtargets --nodetype=storage --state
TargetID       Reachability       Consistency     NodeID
===========    ============       ===========     ===========
211            Online             Good            21
221            Online             Good            22
311            Online             Good            31
321            Online             Good            32
411            Online             Good            41
421            Online             Good            42
511            Online             Good            51
521            Online             Good            52
```

8.4 Define Stripe Pattern for Mirroring

BeeGFS supports folder level mirroring with different chunksize for striping. You can set the number of targets and chunksize based on your requirement for mirroring. Refer to @BeeGFS BuddyMirroring for more information.

1. Use the below commands to set the chunksize to stipe the data across specific targets.

```
$ beegfs-ctl --setpattern --buddymirror --help
$ [root@client1 ~]# beegfs-ctl --setpattern --chunksize=1m  --pattern=buddymirror
--numtargets=2 /mnt/beegfs/bdm
New chunksize: 1048576
New number of storage targets: 2
```

2. Validate the buddy mirroring configuration for the folder.

```
$ [root@client1 ~]#  beegfs-ctl --getentryinfo /mnt/beegfs/bdm
Entry type: directory
EntryID: 0-6008063F-B
Metadata node: meta1-inst2 [ID: 12]
Stripe pattern details:
+ Type: Buddy Mirror
+ Chunksize: 1M
+ Number of storage targets: desired: 2
+ Storage Pool: 1 (Default)
[root@client1 ~]#
```

8.5 Node Timeout Settings

BeeGFS support a time-limited retry mode for communications. Set the timeout value to avoid delay in case of failure when the storage/metadata servers are temporarily unreachable; the buddy mirrored secondary target will become the primary server and handle all the requests. You need to set the timeout value in the management and client servers as well to reinitiate connectivity in case of access failure.

The default settings is 180 seconds on the server. Edit the sysTargetOfflineTimeoutSecs value and decrease the timeout value to 30/60 seconds to avoid delay in case of node failure. Edit the timeout value for management, metadata, and client nodes as well to avoid access failure/delay in case of connectivity issues.

Storage Node file location.

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/inst1.d/beegfs-storage.conf
sysTargetOfflineTimeoutSecs = 180
```

Management Node file location.

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/beegfs-mgmt.conf
sysTargetOfflineTimeoutSecs = 180
```

MetadataNode file location.

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/inst1.d/beegfs-meta.conf
sysTargetOfflineTimeoutSecs = 60
```

Client Node file location.

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/beegfs-client.conf
sysTargetOfflineTimeoutSecs = 900
```

9. BeeGFS Metadata Mirroring

To active metadata mirroring, use the beegfs-ctl tool.

```
$ beegfs-ctl –mirrormd
```

Running this command will enable metadata mirroring for the root directory of the BeeGFS, as well as all the *files* contained in it. Note that existing directories except for the root directory will **not** be mirrored automatically.

Please see the help of beegfs-ctl for more information on available parameters.

```
$ beegfs-ctl --mirrormd --help
```

Notes: When applying the beegfs-ctl --mirrormd command, **no clients may be mounted**. This can be achieved by stopping the beegfs-client service beforehand, and starting it again after beegfs-ctl --mirrormd was successfully performed. In addition, please **restart the** beegfs-meta **service** on all nodes afterwards.

List metadata mirroring groups:

```
$ beegfs-ctl --listmirrorgroups --nodetype=meta
```

9.1 Metadata Sync

If a secondary target or server is considered to be out-of-sync, you can initiate resync using the below commands.

```
$ beegfs-ctl --startresync --help
$ beegfs-ctl --resyncstats --help
```

The following command could be used to stop the automatic resynchronization and start a full resynchronization instead.

```
$ beegfs-ctl --startresync --nodetype=storage --targetid=X --timestamp=0 --restart
```

10. Performance Test with IOR

IOR is a benchmark tool to measure performance of a single or multiple clients with one or more processes per client. IOR is based on MPI for distributed execution. It can be used to measure streaming throughput or small random IO performance (IOPS). Refer to @IOR for more details.

Follow the instruction below to set up IOR.

10.1 Setting up IOR

1. If configure is missing from the top-level directory, you probably retrieved this code directly from the repository. Run ./bootstrap to generate the configure script.
2. Run ./configure. For a full list of configuration options, use ./configure --help.
3. Run "make", then run "make install." The installation prefix can be changed via ./configure --prefix=....
4. Export the below paths.

```
$ export PATH=$PATH:/usr/mpi/gcc/openmpi-4.0.3rc4/bin/
$ export OMPI_ALLOW_RUN_AS_ROOT=1
$ export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
```

10.2 Testing

IOPS Benchmark Example

```
$ mpirun -hostfile /tmp/nodefile --map-by node -np ${NUM_PROCS} /usr/bin/IOR -w -i5
-t4k -b ${BLOCK_SIZE} -F -z -g -o /mnt/beegfs/test.ior
```

mpirun Parameters

-hostfile $PATH (file with the hostnames of the clients/servers to benchmark)

-np $N (number of processes)

IOR Parameters

-w (write benchmark)

-r (read benchmark)

-i $N (repetitions)

-t $N (transfer size, for dd it is the block size)

-b $N (block size, amount of data for a process)

-g (use barriers between open, write/read, and close)

-e (perform fsync upon POSIX write close, make sure reads are only started are all writes are done.)

-o $PATH (path to file for the test)

-F (one file per process)

-z (random access to the file)

10.3 IOR Performance Results

The maximum performance achieved with IOR without buddy mirroring.

IOR Test Results	GBps
Read	40 GBps
Write	18 GBps

The maximum performance achieved with IOR with buddy mirroring.

IOR Test Results	GBps
Read	40 GBps
Write	7.6 GBps

Read Test Results

```
root@ client1 ior-3.3.0]# mpirun -hostfile /tmp/nodefile --map-by node -np 48 /usr/local/bin/
ior -k -i3 -t512K -b 1G -F -g -e -o /mnt/beegfs/bdm3/file14.ior
IOR-3.3.0: MPI Coordinated Test of Parallel I/O
Began                : Fri Jan 22 13:55:11 2021
Command line         : /usr/local/bin/ior -k -i3 -t512K -b 1G -F -g -e -o /mnt/beegfs/
                       bdm3/file14.ior
Machine              : client1
TestID               : 0
StartTime            : Fri Jan 22 13:55:11 2021
Path                 : /mnt/beegfs/bdm3
FS                   : 145.5 TiB   Used FS: 11.7%   Inodes: 0.0 Mi   Used Inodes: -nan%

Options:
api                  : POSIX
apiVersion           :
test filename        : /mnt/beegfs/bdm3/file14.ior
access               : file-per-process
type                 : independent
segments             : 1
ordering in a file   : sequential
ordering inter file  : no tasks offsets

nodes                : 4
tasks                : 48
clients per node     : 12
repetitions          : 3
xfersize             : 524288 bytes
blocksize            : 1 GiB
aggregate filesize   : 48 GiB
```

```
Results:

access    bw(MiB/s)  IOPS       Latency(s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)   total(s)   iter
--------  ---------  ---------- ----------  ---------- ---------  ---------- ---------- ---------- ---------- ----
write     1950.46    3907       0.003932    1048576    512.00     0.003327   25.16      0.034753   25.20      0
read      39321      79619      0.000583    1048576    512.00     0.000863   1.23       0.014459   1.25       0
write     2771.65    5545       0.001214    1048576    512.00     0.003059   17.73      0.002343   17.73      1
read      39814      79852      0.000521    1048576    512.00     0.001097   1.23       0.002352   1.23       1
write     2119.78    4240       0.003950    1048576    512.00     0.002774   23.18      0.002074   23.19      2
read      36259      72701      0.000659    1048576    512.00     0.001121   1.35       0.002289   1.36       2
Max Write: 2771.65 MiB/sec (2906.29 MB/sec)
Max Read:  39814.26 MiB/sec (41748.28 MB/sec)

Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev   Max(OPs)   Min(OPs)  Mean(OPs)
StdDev    Mean(s) Stonewall(s) Stonewall(MiB) Test# #Tasks tPN reps fPP reord reordoff
reordrand seed segcnt   blksiz    xsize aggs(MiB)   API RefNum
write        2771.65    1950.46    2280.63     354.02    5543.30    3900.92    4561.26
708.04   22.04045          NA            NA      0     48  12    3   1     0         1
0    0      1 1073741824   524288   49152.0 POSIX      0
read         39814.26   36258.65   38464.79    1572.90   79628.52   72517.30   76929.58
3145.79    1.28004          NA            NA      0     48  12    3   1     0         1
0    0      1 1073741824   524288   49152.0 POSIX      0
Finished               : Fri Jan 22 13:56:21 2021
```

Write Test Results

```
[root@ client1 ior-3.3.0]#  mpirun -hostfile /tmp/nodefile --map-by node -np 120 /usr/local/
bin/ior -w -i3 -t1M -b 40G -F -g -e -o /mnt/beegfs/test1/t16.ior

IOR-3.3.0: MPI Coordinated Test of Parallel I/O
Began                  : Wed Jan 27 15:04:40 2021
Command line           : /usr/local/bin/ior -w -i3 -t1M -b 40G -F -g -e -o /mnt/beegfs/
                         test1/t16.ior
Machine                : client1
TestID                 : 0
StartTime              : Wed Jan 27 15:04:40 2021
Path                   : /mnt/beegfs/test1
FS                     : 145.5 TiB   Used FS: 24.4%   Inodes: 0.0 Mi   Used Inodes: -nan%

Options:
api                    : POSIX
apiVersion             :
test filename          : /mnt/beegfs/test1/t16.ior
access                 : file-per-process
type                   : independent
segments               : 1
ordering in a file     : sequential
ordering inter file    : no tasks offsets
nodes                  : 5
tasks                  : 120
```

```
clients per node       : 24
repetitions            : 3
xfersize               : 1 MiB
blocksize              : 40 GiB
aggregate filesize     : 4.69 TiB

Results:

access   bw(MiB/s) IOPS    Latency(s) block(KiB)  xfer(KiB) open(s)     wr/rd(s)  close(s)   total(s)   iter
-------- ------   ------  ------     ------      ------    ------     ------    ------     ------     -----
write    15804    15819   0.005541   41943040    1024.00   0.007000   310.72    0.279219   311.01       0
remove   -        -       -          -           -         -          -         -          2.06         0
write    16812    16818   0.004707   41943040    1024.00   0.005519   292.26    0.089571   292.36       1
remove   -        -       -          -           -         -          -         -          2.22         1
write    16376    16381   0.006253   41943040    1024.00   0.005603   300.06    0.070431   300.14       2
remove   -        -        -         -           -         -          -         -          1.98         2
Max Write: 16812.35 MiB/sec (17629.03 MB/sec)

Summary of all tests:

Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev   Max(OPs)   Min(OPs)  Mean(OPs)
StdDev    Mean(s) Stonewall(s) Stonewall(MiB) Test# #Tasks tPN reps fPP reord reordoff
reordrand seed segcnt   blksiz    xsize aggs(MiB)   API RefNum

write       16812.35   15804.14   16330.95     412.85   16812.35   15804.14   16330.95
412.85  301.16788          NA             NA      0    120  24    3   1     0         1
0    0      1 42949672960  1048576 4915200.0 POSIX      0

Finished               : Wed Jan 27 15:19:50 2021
```

Read & Write Result

```
[root@ client1 ~]# mpirun -hostfile /tmp/nodefile --map-by node -np 120 /usr/local/bin/ior -i3
-t1M -b 30G -F -g -e -o /mnt/beegfs/new6/m6.ior

IOR-3.3.0: MPI Coordinated Test of Parallel I/O

Began                  : Mon Feb  1 17:22:41 2021
Command line           : /usr/local/bin/ior -i3 -t1M -b 30G -F -g -e -o /mnt/beegfs/new6/
                         m6.ior
Machine                : client1
TestID                 : 0
StartTime              : Mon Feb  1 17:22:41 2021
Path                   : /mnt/beegfs/new6
FS                     : 145.5 TiB   Used FS: 36.5%   Inodes: 0.0 Mi   Used Inodes: -nan%

Options:
api                    : POSIX
apiVersion             :
test filename          : /mnt/beegfs/new6/m6.ior
access                 : file-per-process
type                   : independent
segments               : 1
ordering in a file     : sequential
ordering inter file    : no tasks offsets
nodes                  : 5
```

```
tasks                  : 120
clients per node       : 24
repetitions            : 3
xfersize               : 1 MiB
blocksize              : 30 GiB
aggregate filesize     : 3.52 TiB

Results:

access   bw(MiB/s) IOPS    Latency(s) block(KiB)  xfer(KiB) open(s)     wr/rd(s)  close(s)    total(s)  iter
-------- ------   ------  ------     ------      ------    ------     ------    ------      ------    -----
write    18092    18106   0.004758   31457280    1024.00   0.007581   203.60    0.149147    203.76      0
read     38187    38195   0.002721   31457280    1024.00   0.002878   96.51     0.018613    96.54       0
remove   -        -       -          -           -         -          -         -           0.009631    0
write    18097    18098   0.005534   31457280    1024.00   0.007739   203.69    0.006350    203.70      1
read     37358    37367   0.002884   31457280    1024.00   0.002939   98.65     0.020917    98.68       1
remove   -        -       -          -           -         -          -         -           0.009013    1
write    18345    18346   0.005075   31457280    1024.00   0.007148   200.94    0.008147    200.95      2
read     37787    37795   0.002479   31457280    1024.00   0.003236   97.54     0.015355    97.56       2
remove   -        -       -          -           -         -          -         -           0.008710    2
Max Write: 18344.63 MiB/sec (19235.74 MB/sec)
Max Read:  38186.83 MiB/sec (40041.79 MB/sec)
```

This document provides instructions on how to set up a BeeGFS configuration quickly and easily with OpenFlex as back-end storage. The measured performance numbers are specific for the current configuration. This reference architecture provides high throughput levels with the IOR tests using a set of files that begin to approach the limits of the storage controllers and drives. The tests give insight into performance with different file counts and sizes. The test results are intended to show the upper range of performance expectations.

11. Conclusion

Customers and partners face a variety of challenges managing on-premises environments including space, cost, complexity, and security concerns related to data. For some organizations, an on-premises environment will make the most sense. For others, running some applications partially or entirely in the cloud can be an attractive option because it allows them to outsource non-core requirements, shift to a variable cost model, and free up capital to where it can be employed more productively. By offering a mixed/hybrid approach to HPC, Western Digital addresses both scenarios. The shift to the cloud represents an opportunity for both service providers, and IT organizations to re-think how they deploy HPC infrastructure and evolve from fixed-cost models to usage-based models. Cloud providers can provide a spectrum of services from IaaS to PaaS to full-service application offerings (SaaS) delivered by professionals with application domain expertise. By leveraging Western Digital's customizable framework, enterprises and service providers can:

- Reduce risk by leveraging proven infrastructure components
- Reduce development costs
- Accelerate time to market for new application services

Using this solution based on the latest NVMe-oF and CDI technologies, organizations can quickly deploy a proven, self-service, composable infrastructure solution, helping customers move to a more flexible, variable cost model.

12. References

https://www.westerndigital.com/products/data-center-platforms/openflex-composable-infrastructure

https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/platforms/openflex/user-guide-openflexf3100-western-digital.pdf

https://doc.beegfs.io/latest/index.html

https://www.beegfs.io/wiki/UserGuide

https://www.beegfs.io/wiki/DownloadInstallationPackages

https://community.mellanox.com/s/article/howto-configure-and-test-beegfs-with-rdma

https://www.beegfs.io/wiki/StorageServerTuning#softwareraid

https://www.beegfs.io/wiki/AboutMirroring#restore_from_buddy

https://www.beegfs.io/wiki/BuddyGroups

https://www.beegfs.io/wiki/MDMirror

https://linbit.com/drbd-user-guide/drbd-guide-9_0-en/#p-intro

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/high_availability_add-on_administration/ch-startup-haaa

Western Digital.

5601 Great Oaks Parkway
San Jose, CA 95119, USA
www.westerndigital.com