Running a CEPH-Cluster from a containerized infrastructure

Performance test with a mySQL-Database

Julius Neudecker
Bachelor of Science
julius.neudecker@haw-hamburg.de

May 2021

Contents

1 Introduction **4**
1.1 Problem domains . . . 4
1.2 Definition of research goal . . . 5
1.3 Related Work . . . 5

2 Setting up CEPH on Docker **6**
2.0.1 Containerization . . . 6
2.1 CEPH Architecture . . . 7
2.1.1 Cluster Access . . . 7
2.1.2 Object Storage Devices - OSD . . . 8
2.1.3 Monitor Nodes - MON . . . 8
2.1.4 Metadata Server - MDS . . . 9
2.1.5 Manager - MGR . . . 9
2.2 System Architecture . . . 9
2.2.1 Hardware Diagram . . . 9
2.2.2 Docker Config for CEPH Image . . . 10
2.2.3 CRUSH Fail mode . . . 12
2.2.4 Creating the cluster . . . 13

3 System Analysis **14**
3.1 SCBENCH . . . 14
3.2 Mysqlslap . . . 18
3.3 Disk Failure . . . 21

4 Conclusion **22**
4.1 Advantages . . . 22
4.2 Disadvantages . . . 22
4.3 Performance . . . 23
4.4 In Summary . . . 23

Setting up and operating a storage cluster with high availability is a complex task. By using containerization, it is possible to abstract away and encapsulate some repetitive tasks. Therefore this study aims to analyse the possibility, implications and findings of a multi host CEPH-Cluster, where the individual daemons are entirely running within a containerized environment. To put the findings into a frame of reference, this study utilizes a mySQL-database which has special requirements on data storage. The key points in terms of advantages and disadvantages, data integrity, performance and administration are scrutinized. Major findings are that there is a performace penalty which can be alleviated by optimizing the cluster configuration. However, these findings in the context of this paper are sometimes theoretical in nature since some constraints of the experimental environment had a significant impact on the results of the study.

1 Introduction

In times where information is a valuable asset, it is of paramount importance to have a scalable and reliable way of storing data and information. Considerations about data throughput and IOPS[1] are also a major design parameter on modern storage solutions. These different storage solutions provide different approaches on these considerations. For any given use case, there exists several options to address these. Depending on the architecture and scope of the problem some are better suited than others. A few major considerations are apart from scalability, reliability, portability and speed also cost effectiveness, vendor lock-in, complexity and granular customizability.

Generally hardware based solutions have advantages in terms of raw performance but they often have significant disadvantages when it comes to vendor lock-in, easy scalability or restoration of corrupted disks. Software and network based solutions are usually less performant due to the network overhead. However this can be mitigated for the most part by scaling horizontally.

Apart from proprietary cloud storage providers like AWS, Azure or Google, the open source community is heavily dominated by CEPH by more than twice as much as the next competitor:

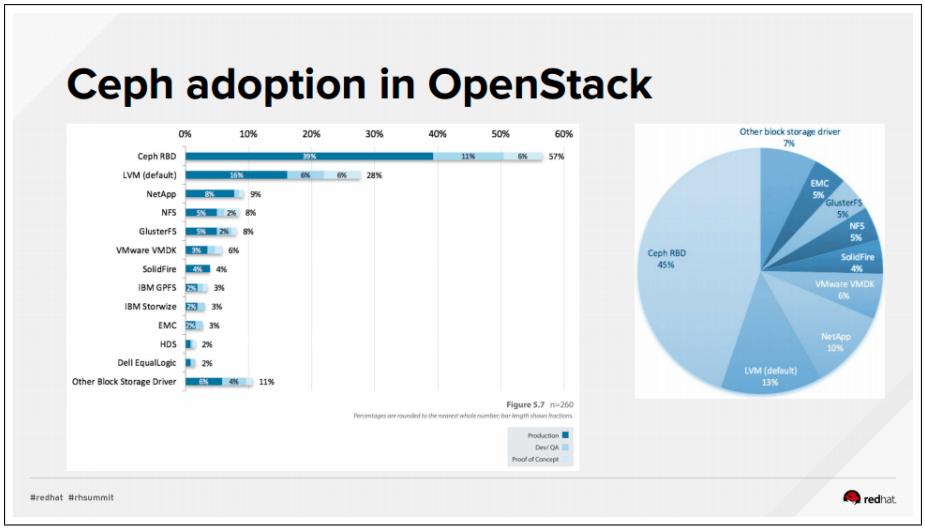

Figure 1: Adoption of CEPH in OpenStack in 2016, [1]

Being conceived by Sage Weil for his doctoral thesis [2], CEPH became part of the Linux Kernel in 2010 and was acquired by RedHat in 2014, CEPH is gaining popularity steadily since its introduction as figure 1 aims to show.

Another modern important concept which is increasingly shaping modern technology stacks is *OS level virtualization* or *containerization* as its colloquially called. This way of deploying applications and services decreased the complexity of deploying several services and application on any given host dramatically by providing each service with an isolated runtime environment with the necessary dependencies

Lastly, no storage solution exists without its use case. One major application, which requires flexible scaling are databases. Scaling up databases is not an easy task as the tech stack is fairly monolithical in nature. A transparent, performant and scaleable storage solution would alleviate many problems.

1.1 Problem domains

Managing a highly available storage cluster is not a trivial task because of creating configuration files and making all applications and services work in coexistence is sometimes tedious work. Unforeseen edge cases and undiscovered bugs are creating an additional layer of complexity. Often securitry policies and concerns also come into play. Apart from provisioning and monitoring the hardware, setting up multiple systems concurrently is a daunting task. Nowadays with infrastructure automation tools like Salt, Chef or Puppet, this is easier than ever. However, it can still be a tedious task to

[1] Input/Output Operations per Second

tweak the configuration of these tools in order to make it work on a complex or diverse infrastructure. Especially in times when upates and EOL[2] events create incompatibilites between working application stacks on any given host machine.

As for CEPH, this is especially true, since there are three major versions in production [3], as of Q1 2021. Deprecated features and bugfixes create functional inconsistencies between major versions, hence it is a necessity to keep a cluster in production up to date. This necessitates constant modification and testing of the previously mentioned automation tools.

One way to isolate these problems is *OS Level Virtualization*. By doing so, every application or application stack exists within a so called *Container* and is therefore isolated from the host to a certain degree. One inherent issue in this context is that they are stateless and ephemeral. This means that they can *by principle* be created and deleted according to momentary requirements. This process can be fully automated by *orchestration software*. Therefore the virtualized production environment has to be configured in a way that allows it to provide the stability which is required for a storage engine.

Trying to overcome the difficulties in setting up and manage a CEPH cluster with containerization might seem contradicting at first sight. The following sections focus on the major problem and adresses these. There are many related topics such as further optimizations for one or another particular use case. To adress these would go beyond the scope of this paper. Nevertheless a brief outlook on further considerations will be provided at the end in chapter 4.3.

1.2 Definition of research goal

The goal of this study is to evaluate if setting up a CEPH-storage cluster with containers is possible and if so, if it is a iable option for a production environment. In order to make a conclusive assessment, three main points have to be examined:

Data Integrity This means running a service on the cluster, which is very sensitive to data inconsistencies. In this particular example a mySQL database is chosen. Databases have some unique properties, which makes them more sensitive to issues with data replication and keeping clustered storages in sync. Therefore this use case is chosen as a suitable real world application.

Performance Since performance is a main consideration in production environments, this is the second most important key point. Depending on the overall cost structure of the environment, a performance penalty might outweigh the benefits. In that case a containerized cluster would be *technically* possible but not economically feasible. Depending on the use case the important metric also differs. Fileserving services like storage clouds or streaming services rely more on raw throughput. The data traffic with databases is generally rather small, therefore IOPS[3] are more important.

Administration One important reason to do research in this topic is to evaluate if the time and effort to set up a cluster brings benefits in terms of administrative expenditure. At first sight a viable metric could be the spent time from starting to have a cluster up and running. However, depending on the production environment the results may vary to a wide degree. Therefore chapter 4 will provide insight in the process of setting up this cluster and its problems and pitfalls.

1.3 Related Work

Since containerization and CEPH are already well established technologies, this section gives a brief overview of other research which was already conducted in this area. RedHat in cooperation with Percona and Supermicro already conducted extensive research whether it is feasible to run a SQL-Database on a CEPH-Cluster [4]. They concluded that indeed running a database on an optimized CEPH-cluster exceeds industry standard database solutions. Furthermore scaling horizontally is easier

[2]End Of Life
[3]Input/Output Operations per Seconds

with a CEPH Cluster. Nevertheless it should be mentioned, that in this study the Percona SQL distribution was utilized, which provides a native interface for kernel based RBD[4].

Hong et.al. used CEPH to create a framework which aims to mitigate handling issues with database containers. This includes the issues mentioned in chapter 1.1 that containers aren't persistent ways to store data [5]. However this paper does not host the whole database container on a CEPH-cluster but only the volume to store the database tables and has therefore only little relevance to this topic.

Although there are numerous other papers which discuss the process of setting up CEPH itself or in the context of an OpenStack environment, none of them discussed a similar scope as this.

2 Setting up CEPH on Docker

This section is about setting up CEPH with containerization. Firstly, an extensive insight into CEPH and its underlying structure is provided. Secondly The system architecture for the experiments in the scope of this study is explained. This second section will make a small detour into containerization such as what it means, what the benefits and what the drawbacks are.

2.0.1 Containerization

Although in the previous text the term containerization was mentioned a few times already, there was no definitive explanation as of what a container is or what containerization means. Firstly it is important to establish that containers are not *something like* a virtual machine. In fact virtual machines are entire systems which are executed on the hosts hypervisor[5] and entirely self contained. This approach is really useful to deploy and maintain infrastructure since a machine with a defined behaviour can be launched in little time with no effort. However in the domain of application development or microservices especially with CI/CD[6] pipelines in mind, this is an u necessary overhead. Containers on the other hand are small entities, which contain only the necessary components of an application or service to run and still have some degree of encapsulation to the host system.

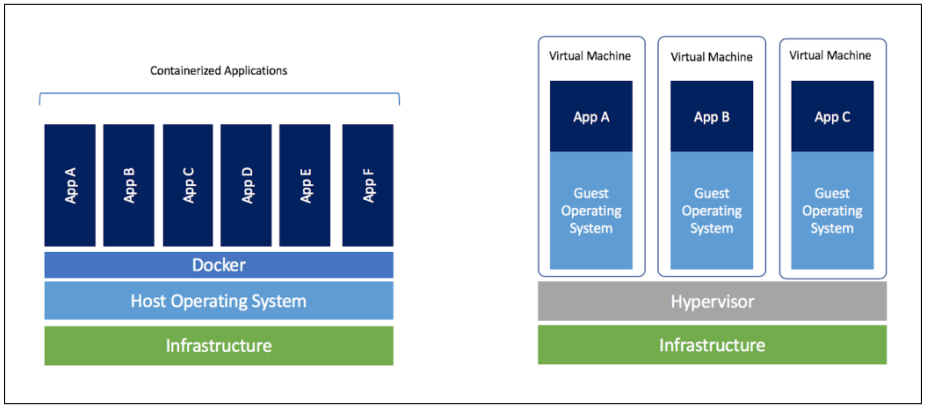

Figure 2: Containerization compared to Virtualization, [6]

By deploying applications or services this way, conficiting software versions, applications and configurations can be encapsulated and thus solved. This makes scaling also easier, since scaling doesn't have to happen on a virtual machine level, which might cause additional costs for licensing, services can be scaled on a more granular basis with containers and orchestration tools like *Docker Swarm* or *Kubernetes*, which will be explained in detail in section 2.0.1. As of Q1 2021, there exists a vast variety of different containerization solutions such as Rkt, LXD and Docker [7]. However Docker is unarguably the biggest solution in terms of adoption, available images[7] and community.

[4]Replicated Block Devices
[5]An abstraction layer for hardware and operating systems
[6]Continuous Integration/Continuous Deployment
[7]Docker images are the blueprints to start containers from

2.1 CEPH Architecture

In short CEPH is a distributed solution for storage clusters. On one side it is specifically tailored to provide maximum reliability by distributing data over different disks, machines or even datacenters and therefore also improving overall scalability and performance. On the other side it is also a cost effective solution because it is open source [8] and runs on commodity hardware.

The general structure includes several services which manage different functions within the cluster. In the context of CEPH these are called daemons, which are discussed in detail in the following sections.

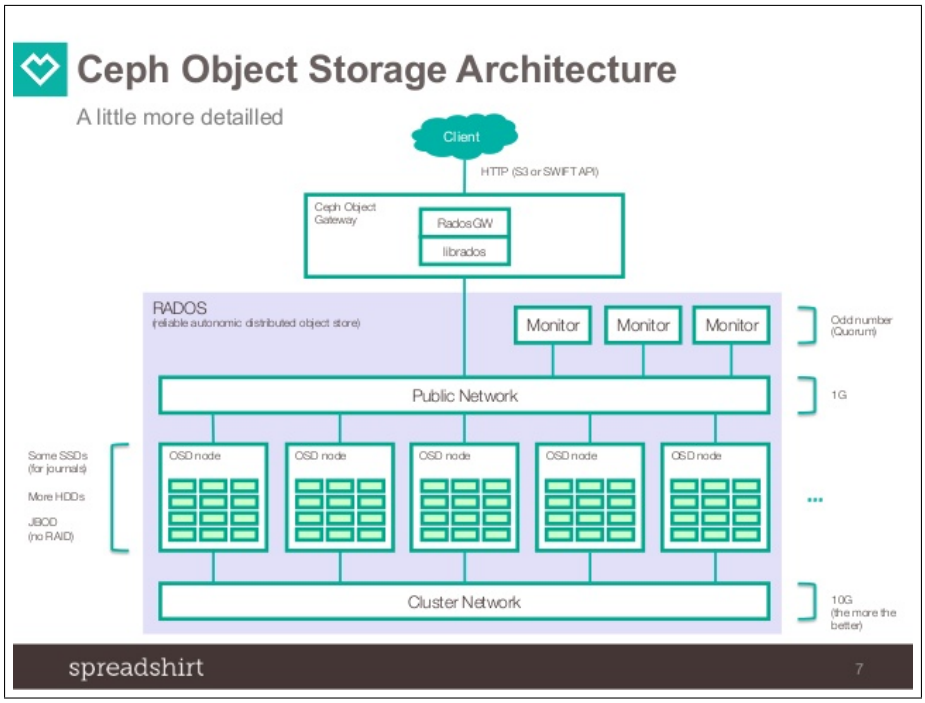

Figure 3: CEPH Architecture, [9]

Because it is a distributed system and may span over several physical machines, racks or even datacenters, inter-daemon communication plays a key role. For the purpose of autonomous data replication the cluster relies on a dedicated network. This network is separated from the public network, which is for client data access. Figure 3 provides a general overview.

How the data is distributed internally is determined by two main factors: *Placement Groups* and the *CRUSH* algorithm. Placement Groups (abbreviated *PG*) is CEPHs way to keep track of the physical location of objects. Files are split up into different objects, where each object is stored in a different PG, which is determined by the CRUSH[8] algorithm [2]. Figure 4 shows a schema of how this works. The reasoning behind PGs is that keeping track of millions of objects via metadata is computationally expensive [10]. Therefore objects are hashed and assigned to PGs. A so called CRUSH-Map keeps track of the structure of the whole cluster. This will become important in section 2.2.3.

2.1.1 Cluster Access

To access the cluster CEPH provides different interfaces depending on the use-case:

- *CephFS* - A POSIX[9] conform filesystem
- *LIBRBD/KRBD* - A block device
- *RADOSGW* - An REST gateway for storage buckets

[8] Controlled, Scalable, Decentralized Placement of Replicated Data
[9] Portable Operating System Interface - Part of the Unix specification

- *LIBRADOS* - The API to access the cluster directly from applications

All these interfaces are based on LIBRADOS, provides the interface to access RADOS. This is the underlying object store responsible for distributing the data over several phsyical disks. This task is done by the *OSD*-Daemon.

CephFS is the filesystem, which allows for CEPH to behave to the user or the operating system like any other filesystem would. However is has to be distinguished from from a block device. The key is that it is not a filesystem like EXT4 or NTFS. It is more a translation layer to the RADOS[10] interface. A required metadata service called *MDS* provides journaling functionality and handles client access. Because the throughput of the whole cluster scales linearly with the size of the cluster, several different MDSs are required on big clusters to handle the client load and distribute the file metadata. Thus also mitigating single points of failure. It is best practice to store and serve the MDS from fast solid state drives to reduce hardware bottlenecks.

LIBRBD/KRBD is a virtual block device. The difference to CephFS is that is behaves like a physical disk. Therefore it can be formatted with any filesystem like EXT4 or NTFS. The distinction between *LIB*RBD and *K*RBD means userspace for LIBRBD and kernelspace for KRBD. To explain these differences between these two would go beyond the scope of this paper and requires knowledge about kernels, operating systems and memory handling.

RADOSGW provides as REST gateway access to S3[11] buckets. Unlike filesystems or block devices, these S3 buckets don't have a hierarchical structure with directories. All objects are stored on the same level, therefore the analogy to a bucket. This bucket is addressed by means of an API which provides also a metadata filter to select specific files directly. A widely adopted use case is serving static files on the internet for media content, javascript or css files by content delivery networks. They can also be used to store virtual disk images to be used in conjunction with RBDs to name a few use cases.

LIBRADOS provides a native API to access RADOS directly without any file system or block device translation layer in between. The higher development costs for a more complex implementation might be worth the gains in throughput and IOPS for critical applications. This is implemented in the Percona mySQL Server [4].

2.1.2 Object Storage Devices - OSD

Within the CEPH domain, these are the disks, where the actual data objects are stored to. One or more OSDs can be handled from one CEPH OSD Daemon, which provides the connecting layer to the whole cluster. Also one OSD can contain several PGs. This depends on the configuration of the cluster. On which disk and in which PG and object is stored is determined by the CRUSH algorithm as shown in figure 4. Since data is replicated within the cluster and constantly updated, one OSD could fail and the data and PGs of this failed OSD will be written to another OSD. This constant data shuffling takes place transparent to the user and is the reason why CEPH needs a dedicated network as mentioned in section 2.1. How the data is physically stored on the disk differs between the two different backends used in older or newer releases: Bluestore and Filestore. Where Bluestore is the newer and more performant standard.

2.1.3 Monitor Nodes - MON

Monitor Nodes (abbreviated as *MON*) have two main purposes: They maintain a copy of the cluster map to hand this out to a client, which connects to the cluster. Because of a fault in a MON this map might not reflect the state of the cluster accurately. Therefore an odd number of MON nodes

[10] Reliable Autonomous Distributed Object Storage
[11] Simple Storage Service

have to reach a quorum on the current state of the cluster and distribute the correct cluster map. According to the CEPH documentation this is negotiated via the *PAXOS* algorithm [11]. Although a cluster could technically work with just one MON node, it is highly recommended to have at least three MON nodes, removing a single point of failure.

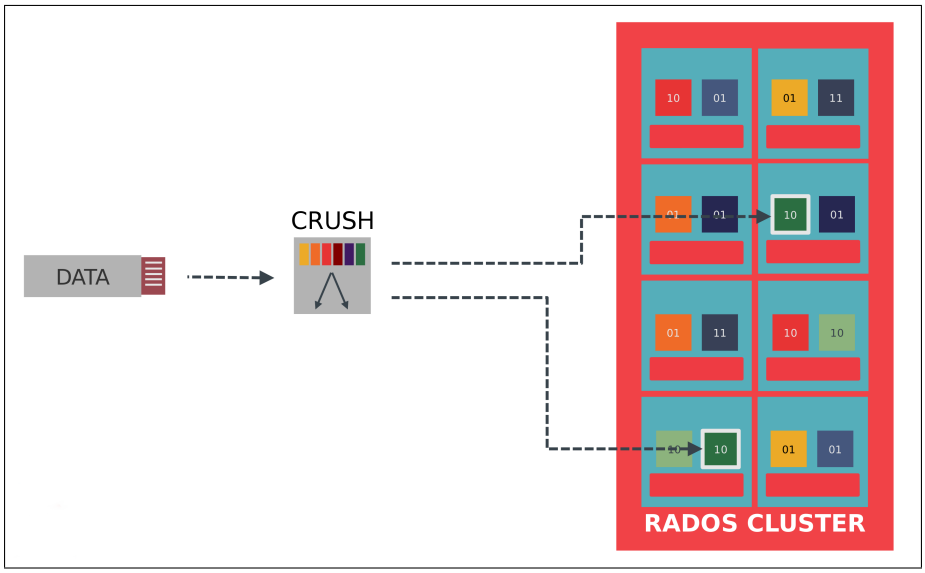

Figure 4: Placement Groups and CRUSH Algorithm, [12]

2.1.4 Metadata Server - MDS

The metadata server is not necessary for the cluster to operate as RBD, storage bucket or via Librados API. However to provide a journaled filesystem via the CephFS, this service is needed to provide the filesystem metadata such as permissions and timestamps. As stated from RedHat, it is best practice to deploy a MDS with big and fast nVME drives [4]. These can be also used in conjunction with memcached, where the recent metadata are stored into the RAM of the MDS. Overprovisioning, automatic failover and scalability are the main considerations for MDS to have high availability and high performance.

2.1.5 Manager - MGR

Since the 12.x release, CEPH needs a manager node to operate [13]. This manager does not provide any functionality to the cluster itself but acts as an interface to external monitoring tools and systems. Without it the state of the cluster won't appear to be healthy nor be externally visibly updated [13].

2.2 System Architecture

In this section I'll go over the system architecture in my experiments since its understanding is vital to the understanding of the configuration of the containers later on.

Firstly, because of the Covid-19 pandemic I couldn't use the lab in the university so I had to rely on my personal resources in order to do my experiments. Since these are somewhat limited I configured a few things different than it would be usually done.

I assume that the reader has a basic understanding of Docker and how commands are executed or how containers work in detail. Therefore topics which are immaterial to this topic won't be explained in great detail.

2.2.1 Hardware Diagram

The cluster is distributed across three different machines for storage. Since the latest CEPH version isn't available as Docker Image for ARMv7 and ARMv8 architectures, the only available x86_64 host acts as Docker host for all containers including the OSDs. However to create constraints such as

network latency and bandwith, the physical volumes were attached to the ARM-Machines and mounted as iSCSI devices on the Docker host. Because in this small setup the available CPU performance and RAM is not a concern, running a few more containers isn't a big concern.

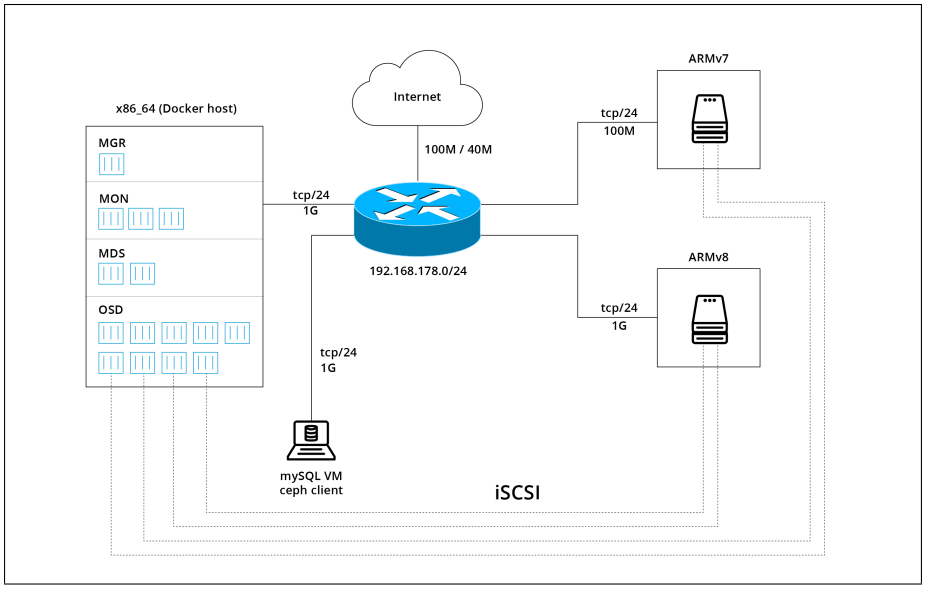

Figure 5: Hardware Diagram

Furthermore the Docker host is a server in production for serving websites and services. Therefore a certain amount of system resources were already allocated to other running processes. Since my tests were exclusively conducted in the local network, the bandwith of the internet connection wasn't a concern either. Usually a CEPH cluster uses two separate networks: one for replicating data between the OSDs and one to client access to the cluster or configuration. In this case only one physical network was available, which has a bandwith of 1GBit/s

2.2.2 Docker Config for CEPH Image

By starting CEPH as a container, a lot of configuration work is abstracted away already in the Docker image. The official image [14] provides two options to distribute the configuration between the different nodes:

- On disk
- KV-Store[12]

The same Image provides all the different daemons, which are needed in a CEPH cluster and thus every container is configured different. The configuration is stored *on Disk* because all containers are executed on the same physical host, hence a remote KV-Store isn't required. This is reflected by the two directories */docker/ceph/data/etc/* and */docker/ceph/data/lib*, which all containers have in common.

Another issue is that the MON and OSD nodes run into timeout issues, when deployed on the same host and configured to use Dockers *–net=host* option, as suggested by the official manual. This would effectively bind the ports of any container, which is configured in that way, directly to the host interface. I chose another approach: providing each container with a dedicated IP-Adress within the network. This is possible by using the *MACVLAN* network driver which is provided by the Docker engine. This allows every container which has this driver enabled to appear as a separate host apart from the docker host. The IP is configured with the *–ip=192.167.178.2XX* option.

The complete configuration is made available under a repository [15].

[12] As of Q1 2021 only etcd is supported by the official image

Monitor Nodes The network configuration of the monitor nodes is of special interest here: The public network has to be configured in case there are two distinct networks as explained in 2.1. This is not necessary for the other nodes since the configuration will be distributed after the monitor nodes are online.

```
docker run -d \
-v /docker/ceph/data/etc:/etc/ceph \
-v /docker/ceph/data/lib:/var/lib/ceph/ \
-e MON_IP=192.168.178.220 \
-e CEPH_PUBLIC_NETWORK=192.168.178.0/24 \
--ip=192.168.178.220 \
--network=macvlan0 \
--name ceph_mon1 ceph/daemon mon
```

Manager Nodes This container runs in parallel to a monitor node and doesn't need any special configuration. For a healthy cluster only one node instance is required.

```
docker run -d \
-v /docker/ceph/data/etc:/etc/ceph \
-v /docker/ceph/data/lib:/var/lib/ceph/ \
--ip=192.168.178.223 \
--network=macvlan0 \
--name ceph_mgr ceph/daemon mgr
```

OSD Nodes The configuration of the OSD nodes is more complicated compared to the other nodes. I chose to provide a physical disk to the container. Therefore it had to be initialized as a logical volume before the OSD node is able to utilize it. The official CEPH image provides a configuration option for this purpose. The following command creates a temporary container, which is configured by the *–rm* command. The *–data /dev/sdX* command specifies the physical disk, which will be pretreated. A prerequisite is that any disk has to be completely wiped of any partition table. This is done by executing *sgdisk -xz* prior to launching the preparation container on every drive to wipe out any existing partition table.

```
docker run --rm \
--privileged=true \
-v /docker/ceph/data/lib:/var/lib/ceph/ \
-v /var/log/ceph:/var/log/ceph/ \
-v /docker/ceph/data/etc:/etc/ceph \
-v /dev/:/dev/ \
-v /run/lock/lvm:/run/lock/lvm:z \
-v /run/lvm/:/run/lvm/ \
-v /var/run/udev/:/var/run/udev/:z \
--ip=192.168.178.230 \
--network=macvlan0 \
--entrypoint ceph-volume ceph/daemon lvm prepare \
--bluestore \
--data /dev/sdX \
--no-systemd
```

When the disk is prepared, it appears as *lvm-partition* with a ceph-prefix. The OSD node can then be started by executing the following command:

```
docker run -d \
--privileged=true \
--pid=host \
-v /docker/ceph/data/etc:/etc/ceph \
-v /docker/ceph/data/lib:/var/lib/ceph/ \
```

```
-v /dev/:/dev/ \
-v /run/udev/:/run/udev/ \
-e OSD_DEVICE=/dev/sde \
-e OSD_TYPE=disk \
-e OSD_BLUESTORE=1 \
-e CEPH_DAEMON=OSD_CEPH_VOLUME_ACTIVATE \
-e OSD_ID=0 \
--ip=192.168.178.230 \
--network=macvlan0 \
--name ceph_osd_sde ceph/daemon osd
```

There are some important things to consider here: firstly the *–pid=host* configuration is necessary to prevent that the host system runs out of open file handles. This is a known bug, which is explained here on the Github page of the Docker Image The *–privileged=true* setting is necessary to give the required access to disks from within the container. However from a security standpoint this is not good practice and should never be used in that way in a production environment! *OSD_BLUESTORE=1* configures the way CEPH stores the data in the background and *OSD_ID=X* defines the ID under which the OSD will be reffered to by the cluster.

Metadata Nodes Since one physical host can host many different metadata nodes to provide peak performance, these MDS nodes have to be named uniquely. This is done by using the *$hostname-x* variable. Also the official image is recommending to set the *CEPHFS_CREATE=1* flag. On my setup however this resulted in an error and the filesystem was not being created. In my case I had to create the filesystem manually from within the monitor container by using the *docker exec* command to open a terminal within the container. Therefore this flag has to be set to "0" in this case.

```
docker run -d \
-v /docker/ceph/data/lib:/var/lib/ceph/ \
-v /docker/ceph/data/etc:/etc/ceph \
-e CEPHFS_CREATE=0 \
-e MDS_NAME=mds-$(hostname)-a \
--ip=192.168.178.245 \
--network=macvlan0 \
--name ceph_mds_a ceph/daemon mds
```

2.2.3 CRUSH Fail mode

In order to define the behaviour of the cluster in case of failures, CEPH provides a setting which is called *CRUSH Fail Mode*. This setting defines the biggest possible failure domains, which should not affect data integrity [16]. In consequence, data is replicated in a way which guarantees this. To pick a few examples:

OSD (or device) means that a single drive could fail and no data will be lost. This would be the preferred setting when the cluster runs on a single machine. However, if the cluster consists of more than one machine and the whole machine fails because of a PSU[13] failure for example, data integrity cannot be guaranteed anymore.

Chassis is the failure mode for rack mounted servers, where a multitude of servers make up the whole cluster. This would cover the previously mentioned case, which might be as well interesting for maintenance, when individual machines have to be maintained or being decommissioned.

Rack provides data integrity across multiple chassis or hosts on a datacenter level. If for example a critical component of the network infrastructure fails, this could provide some security. Although

[13]Power Supplying Unit

CEPH clusters are in principle set up in a way which prevents SPOF[14], this must not be true for the peripheral infrastructure.

Datacenter usually means that a huge number of several racks, consisting of multiple chassis are grouped together as a common failure domain - in case of a power outage or natural disaster. If the company which owns the datacenter, operates on a global scale this failure mode is also a consideration for performance. Because of physical and network topology, the RTT[15] to a certain datacenter might be longer than desired. If all data is replicated on datacenter granularity, a global scale data integrity is guaranteed as well as the possibly shortest response time.

In general the failure mode should be as granular as possible but also as fine as financially feasible. In the context of the system architecture in the scope of this study (see section 2.2), the fail mode will be on an OSD level. Because my experimental setup consists of multiple disks and flash storages with different ages and levels of degradation. However, the appropriate setting would be *host*-level data replication.

2.2.4 Creating the cluster

Although there are different solutions to automate the deployment of a cluster such as *CEPH-Ansible* [17] or *CEPH for Rook* [18] my cluster is rolled out and configured with a simple shellscript - *startup.sh*[16] - which creates a fully functional cluster:

```
#!/bin/sh

## start MON and MGR
docker run -d -v /docker/ceph/data/etc:/etc/ceph -v
    /docker/ceph/data/lib:/var/lib/ceph/ -e MON_IP=192.168.178.220 -e
    CEPH_PUBLIC_NETWORK=192.168.178.0/24 --ip=192.168.178.220
    --network=macvlan0 --name ceph_mon1 ceph/daemon mon
docker run -d -v /docker/ceph/data/etc:/etc/ceph -v
    /docker/ceph/data/lib:/var/lib/ceph/ --ip=192.168.178.223
    --network=macvlan0 --name ceph_mgr ceph/daemon mgr

docker exec -it ceph_mon1 ceph auth get client.bootstrap-osd -o
    /var/lib/ceph/bootstrap-osd/ceph.keyring

## prepare and create OSDs
./prepare.sh
./create.sh

## create fs
docker exec -it ceph_mon1 ceph osd pool create cephfs_data
docker exec -it ceph_mon1 ceph osd pool create cephfs_metadata
docker exec -it ceph_mon1 ceph fs new cephfs cephfs_metadata cephfs_data

##create the additional MON and MDS after the cluster has been created
docker run -d -v /docker/ceph/data/etc:/etc/ceph -v
    /docker/ceph/data/lib:/var/lib/ceph/ -e MON_IP=192.168.178.221 -e
    CEPH_PUBLIC_NETWORK=192.168.178.0/24 --ip=192.168.178.221
    --network=macvlan0 --name ceph_mon2 ceph/daemon mon
docker run -d -v /docker/ceph/data/lib:/var/lib/ceph/ -v
    /docker/ceph/data/etc:/etc/ceph -e CEPHFS_CREATE=0 -e
    MDS_NAME=mds-$(hostname)-a --ip=192.168.178.245 --network=macvlan0
    --name ceph_mds_a ceph/daemon mds
```

[14] Single Point Of Failure
[15] Round Trip Time
[16] see Github repo for the whole document

```
docker run -d -v /docker/ceph/data/etc:/etc/ceph -v
    /docker/ceph/data/lib:/var/lib/ceph/ -e MON_IP=192.168.178.222 -e
    CEPH_PUBLIC_NETWORK=192.168.178.0/24 --ip=192.168.178.222
    --network=macvlan0 --name ceph_mon3 ceph/daemon mon
docker run -d -v /docker/ceph/data/lib:/var/lib/ceph/ -v
    /docker/ceph/data/etc:/etc/ceph -e CEPHFS_CREATE=0 -e
    MDS_NAME=mds-$(hostname)-b --ip=192.168.178.246 --network=macvlan0
    --name ceph_mds_b ceph/daemon mds

exit 0
```

The docker run commands are the same as above but executed in a specific order. One monitor node has to be launched first to create an initial cluster configuration. Shortly thereafter follows a manager node. The next line deploys the keyring-file to the other bootstrap directories. Since all containers are deployed on a single machine, every node can access this keyring in its respective bootstrap directory. The script *prepare.sh* and *create.sh* executes the commands mentioned under the OSD part of section 2.2.2 for each disk. The next step is the creation of the filesystem. According to the docker documentation the FS will be created upon launch of the MDS node with the *CEPHFS_CREATE=1* parameter defined. However this didn't work properly in my case. The lines with *docker exec ...* execute a command within the denominated container (in this case *ceph_mon1*) from the host shell. These three commands create the data pools and the filesystem, which requires as metadata pool and a data pool [19]. The following lines are only the additional MON and MDS nodes being created. These are not strictly necessary for the cluster to function but provide redundancy to secure the *no single point of failure* design of CEPH.

3 System Analysis

To measure the performance of the cluster, I chose to use two tools: *scbench* and *mysqlslap*. Both tools give detailed insight into the critical performances figures of such a cluster. Although there are several factors which are affecting overall performance, this test only provides a rather coarse overview of the system under test in general. For a more precise evaluation of the performance difference between docker and native ceph, parametres such as network latency, disk I/O and other system constraints have to be evaluated beforehand.

3.1 SCBENCH

This tool measures the raw performance of a CEPH cluster by performing a write and subsequent read test for sequential and random disk access [20]. I performed three different tests with different loads on the cluster:

- 10 seconds with 16 threads
- 30 seconds with 32 threads
- 60 seconds with 64 threads

One thread can be considered as one client reading and writing data to the cluster. My goal is to load test the cluster in a way that eventually all caches and short term performance enhancements become saturated to see the stable long term performance. This test yields three different performance figures for read and write actions:

- Bandwidth
- IOPS
- Latency

The reading test was executed for sequential and random access of the data. However the tests for sequential read access didn't provide sufficient enough data to evaluate because this test only yielded one whole dataset [21]. Therefore I will only show the results for random read, which is more important in the context of database operations anyway. All graphs were created from the raw data, which scbench provides. The code to reproduce these results is available under [15].

Write performance Before executing the read performance test, *scbench* has to execute the write test in order to have readable data available.

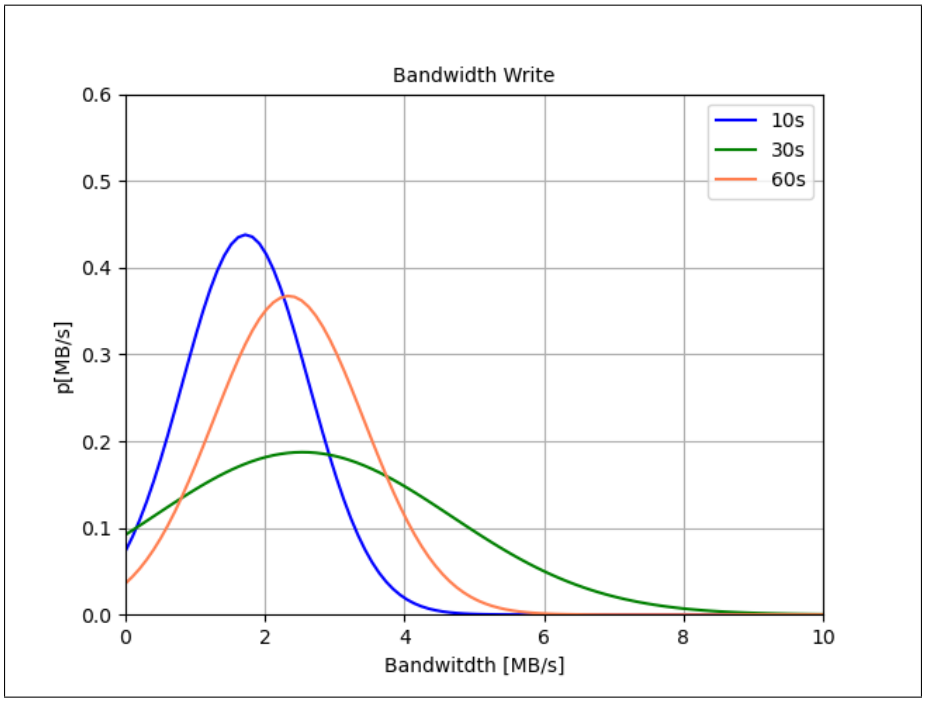

Figure 6: Normal distribution of the writespeed for three different tests

Figure 6 shows the saturation of the write speed with increasing load. On write operations with a short duration, the cost of the whole procedure of contacting the cluster to receive the correct endpoints to write to is relatively high, therefore the initial period where no data is written has an overall bigger impact on the measurement. As the client is writing more data to the OSDs, the writespeed increases in average. However after the different caching entities within the cluster become saturated, the writespeed eventually stabilizes at around 2.5MB/s.

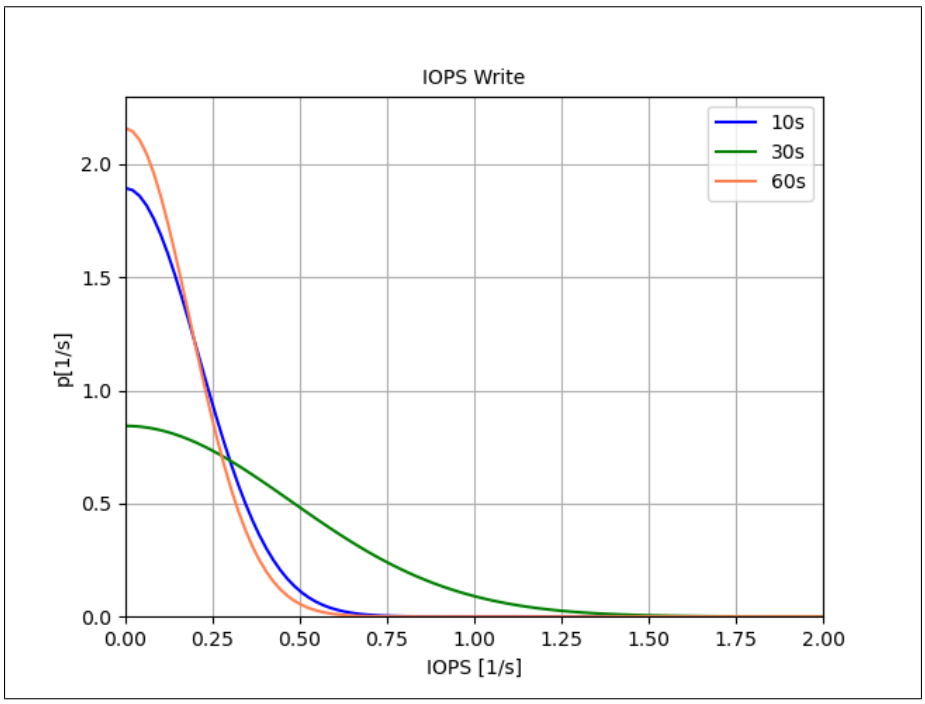

Figure 7: Normal distribution of the write IOPS for three different tests

The write IOPS in figure 7 exhibits the same result: Initial IOPS start slow due to the cost of initializing, subsequently increasing, which is depicted by the standard deviation becoming bigger. So there are more samples with higher IOPS than before. With saturation, the overall IOPS goes back to an even worse value compared to the start of the test. The reason being likely that at this stage the reorganizing of PGs takes away some performance.

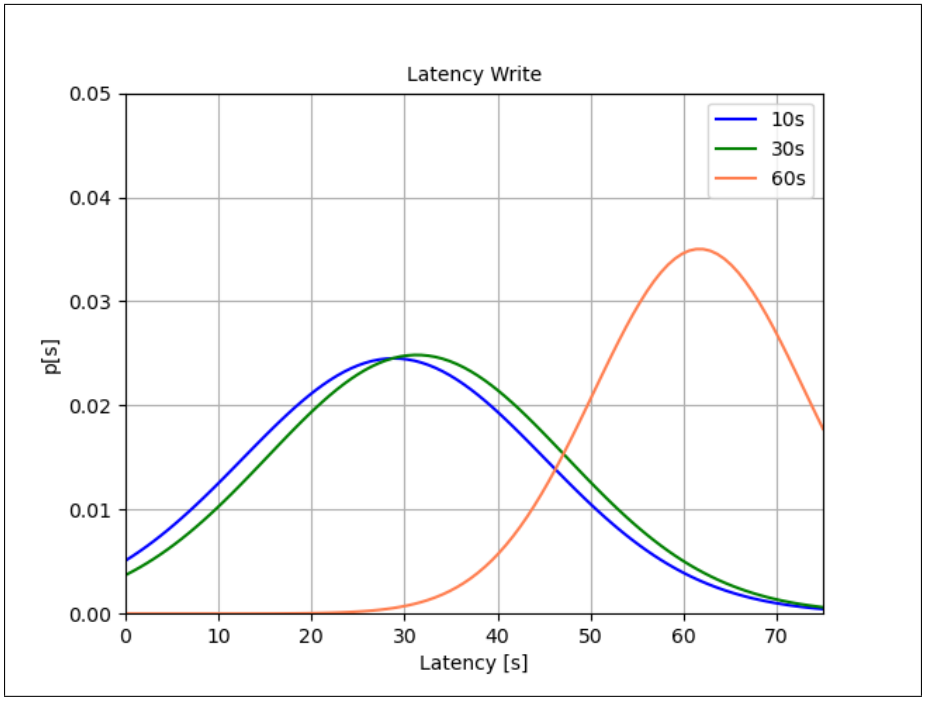

Figure 8: Normal distribution of the write latency for three different tests

The write latency for the 10s and 30s mark shown in figure 8 is about the same. The saturated case after 60s however is an interesting anomaly. The best explanation is that the process of reorganizing data and placement groups takes away performance and therefore increasing latency. This is especially plausible since all data transactions are processed over the same physical network interface since this

cluster does not have a dedicated network for this task.

Read performance This test reads the previously created data during the write test randomly.

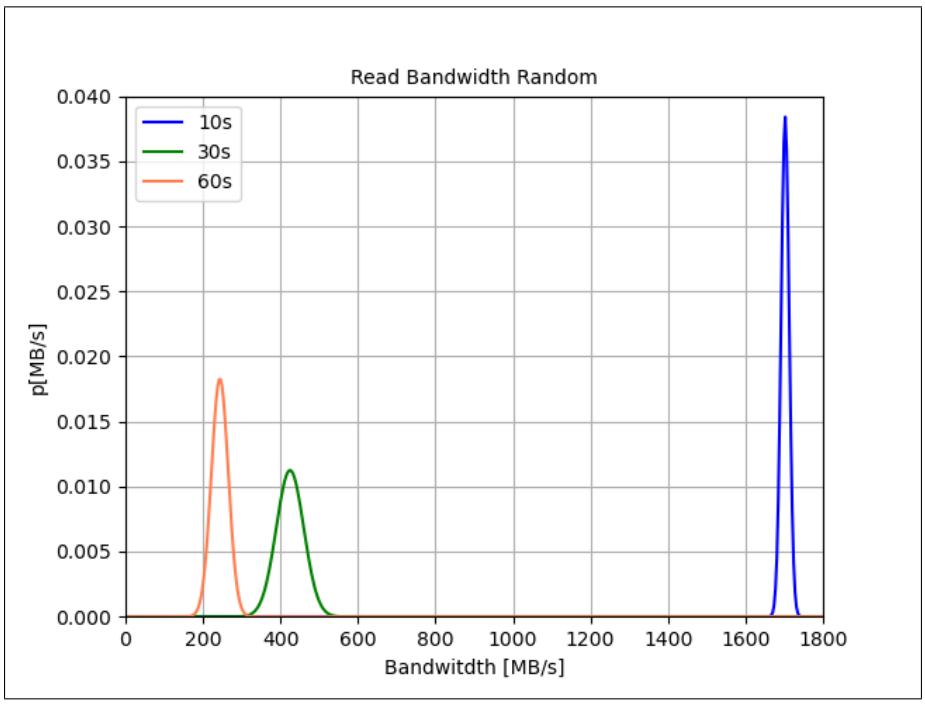

Figure 9: Normal distribution of the readspeed for three different tests

The read speed starts at around 1700MB/s, which is multiple times more than the other more saturated tests were able to achieve. Considered the hardware architecture of the cluster, this value is only really achievable with caching data in the RAM. In fact the hardware requirements for a CEPH-Cluster demands 2GB of RAM minimum per OSD for that reason [22].

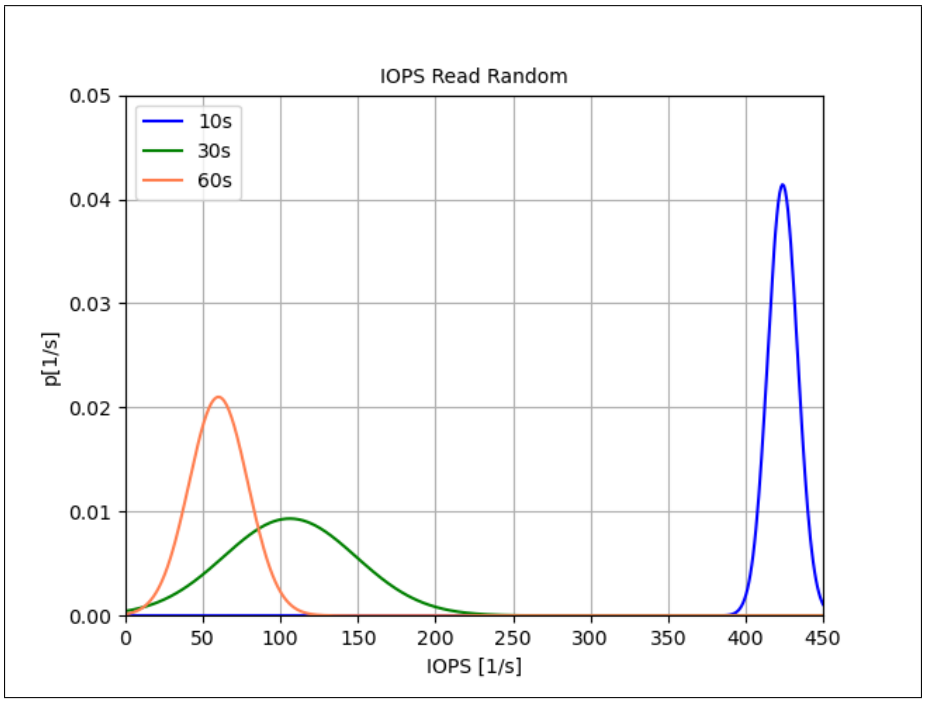

Figure 10: Normal distribution of the read IOPS for three different tests

The IOPS test paints the same picture: very high cluster speed on the first few seconds with a

more stabilized value after saturation of the caching chain. An interesting point is the high standard deviation before everything becomes saturated. I assume that this is the timespan, where some caches are already saturated and other caches are still filling up. Also the CPU of the host machine has only four logical cores and therefore quickly becomes task-saturated. The other cores in the cluster aren't very fast either. This results in the value stabilizing after around a minute.

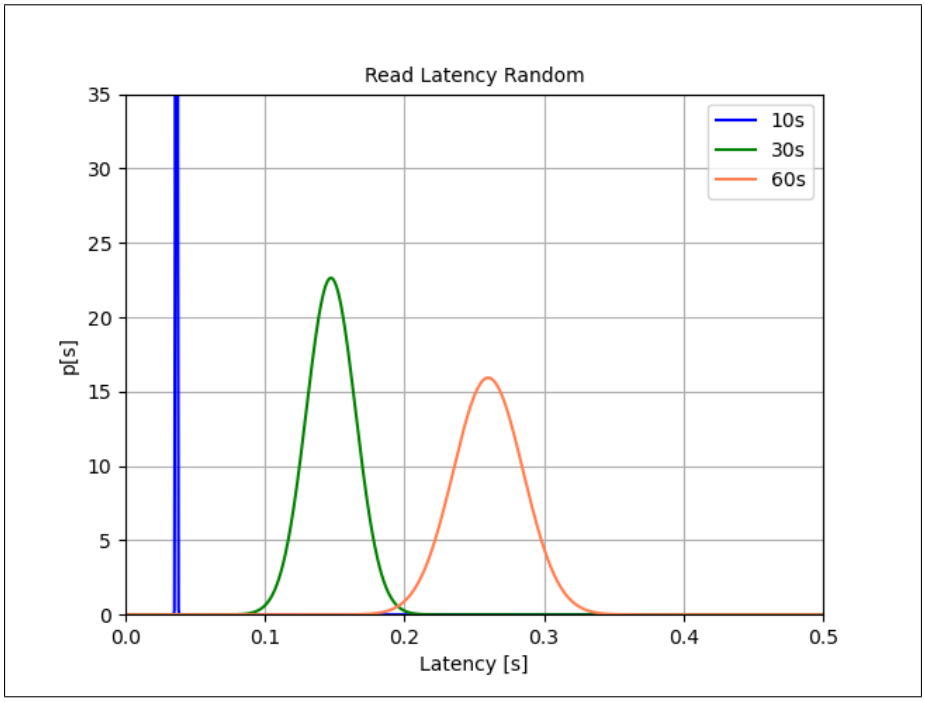

Figure 11: Normal distribution of the read latency for three different tests

The very fast response in the first ten seconds is the best argument that the data actually is read from the RAM instead of the disks. The very small standard deviation can't really be achieved with different kinds of disks on iSCSI hosts. Referring to figure 8, it is likely the same mechanic of reorganizing placement groups and data objects within the cluster which leads to increased latency. Also in conjunction with task saturation of the host machine.

3.2 Mysqlslap

The second test was a dedicatated load test specifically for SQL-Database[17] workloads. The *mysqlslap* application allows for some configuration on how the test is carried out:

- Concurrent users
- Iterations
- Size and structure of automatically generated database table
- Total number of queries

I load tested the database on three different storage technologies to point out potential advantages and disadvantages of ceph in contrast to a purely HDD and SSD based storage solution - both were discrete disks in this example. The mySQL-Server was running on a VM[18] with the CentOS8 distribution. The VM ran on a physically separate machine, which had the storage mounted by either NFS[19] or the ceph kernel-module. The HDD and SSD based load test was carried out using the official

[17] Although I used mySQL in this example, this applies also to other SQL-Based Database distributions.
[18] Virtual Machine
[19] Network File Storage

mySQL-Docker Image [23], because the configuration of different volumes is fairly unproblematic. This didn't work with the ceph kernel module for reasons which I was unable to resolve entirely. Therefore the load test on ceph as the underlying storage was carried out using the native mySQL distribution for CentOS. Apart from giving the tests on the discrete disks a slight disadvantage, it doesn't really make a difference.

To carry out the load test, I wrote the following script:

```
#!/bin/bash

## db connection
dbuser=<mysql-user>
dbhost=127.0.0.1
dbpass=<mysql-password>

## db test config
intcols=5
charcols=20

## iterations over concurrent users
user_min=4
user_max=20
iterations=3
samples=5

## queries per perfomance batch
total_queries=1024

for ((s=0; s<$samples; s++)) do
    echo "Starting Sample number: $s"

    for (( i=$user_min; i<=user_max; i++)) do

        queries_corrected=$((total_queries + (total_queries % $i)))
        mysqlslap --user=$dbuser --host=$dbhost --password=$dbpass
            --concurrency=$i --iterations=$iterations
            --number-int-cols=$intcols --number-char-cols=$charcols
            --number-of-queries=$queries_corrected --auto-generate-sql -v

    done
done

exit 0
```

The core of the script is the *mysqlslap* command being executed with different parametres. For the correct interpretation of the result I want to emphasize out a few key points:

Single Threaded The operations of each user-session is single threaded. The VM had four virtual cores, so a test with less than four concurrent user sessions wouldn't yield any better or worse results.

queries_corrected is a variable, which uses the modulo operation to correct the total number of executed queries slightly. A few more operations won't have any great impact on the test campaign. This creates an evenly distribution of all tests across all concurrent user sessions.

Samples and Iterations is how I configured the script to execute multiple executions of the same load test in order to average out possible outliers. Each iteration is the same number depicted by *total_queries* n-times within the same *mysqlslap* command. Each sample is the repetition of the 4-20 users cycle with three iterations each. This means, the same load test is carried out 15 times for each concurrent user setting.

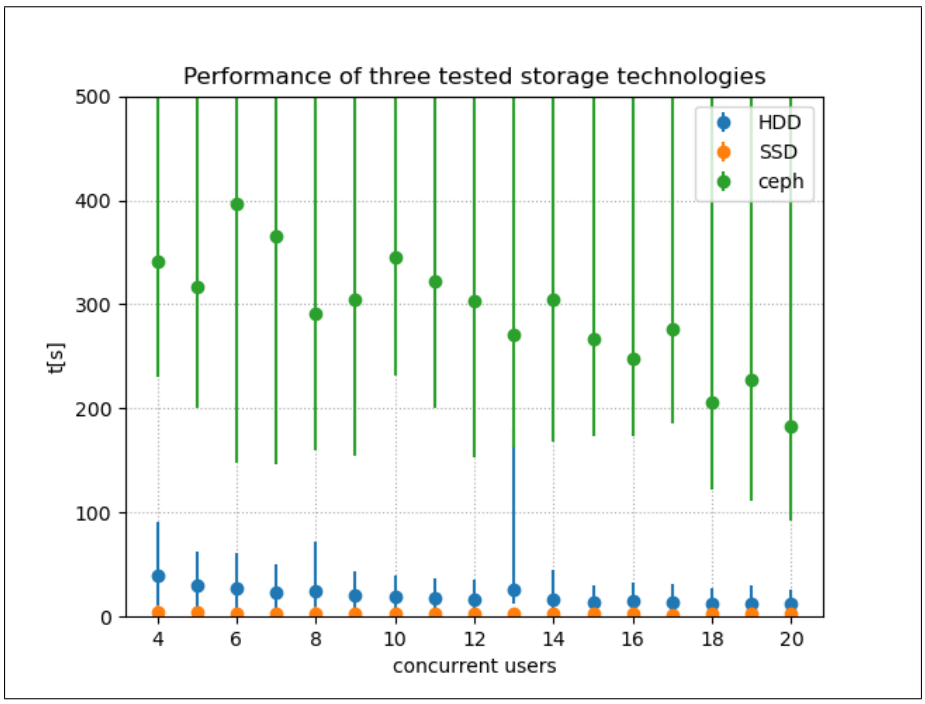

Figure 12: mySQL load test comparison between ceph/HDD/SSD

The horizontal axis is concurrent users and the vertical scale the seconds until all 2048 queries were completed in average with the minimum and maximum time. The exact results of this test are available in the Github repository [15]. By far the best performance was yielded by the SSD-based storage. There is a slight increase in perfomance with more concurrent users, since with a low session count neither the network interface nor the SSD and client machine has reached a bottleneck yet. At around 9-10 concurrent user sessions, the optimal performace is reached and declines slightly with more sessions. However due to the almost instantaneous nature of data access on SSDs, the major part of that time will be network roundtrip times, which add up over many queries.

The same behaviour but much more distinct is shown by the HDD based storage. Since these drives have a longer access time, this adds up to a more distinct increase of performance with increasing concurrency. This will likely be caused by optimizations in the access patterns of the physical disk. I do not have an explanation for the anomaly at the 13 user mark. Since even the minimum time is such an outlier compared to the 12 and 14 user mark, this can't be explained by some random occuring system-behaviour especially due to the fact that the test was carried out 15 times and yielded a similar result every time.

The test of the ceph storage didn't turn out as expected. In fact the time to finish all queries was longer by one magnitudes in comparison to the HDD and two magnitude in case of the SSD. I have a few possible explanations for this behaviour and I think every one of these factors contributed at least partially:

No NAT Reflection The Docker *macvlan* network creates a virtual mac-adress for each container. However, it does not support NAT-Reflection, which results in every packet being routed through the physical network interface to the main switch and back, which adds a lot of round-trip-time to the whole process.

IOWAIT bottleneck During the load test I watched the behaviour of the kernel of the docker-host and it showed that a significant portion - about 1/2 of the total CPU-time was allocated to IOWAIT processes. This indicates that communications with devices and interfaces are very slow. As to where this bottleneck is I could not find a definitive answer. My guess is that the sheer amount of virtual disks in conjunction with a high number of OSDs and the fact, that a few of them were iSCSI devices, caused significant communication overhead, which adds up quickly in this situation.

Missing OSD Network As I explained in section 2.2, my setup doesn't have a dedicated network for the OSD traffic. It is possible that by using the same interface for server-client and OSD-OSD communication resulted in saturating the network interface, which isn't particularly high performance either. This is also a a good explanation for the IOWAIT kernel phenomenon.

Slow disks I used USB pen drives on the ARMv7 host, which were exported as iSCSI devices. It is possible that these drives degraded due to the heavy read and write load under the constant load and therefore decreased drastically in performace.

One thing which is quite interesting though is that with ever increasing concurrent user load, the average time tends to decrease. This is a sign of an optimized access pattern for multiple data objects in the cluster.

3.3 Disk Failure

One of my research goals defined in section 1.2 was to examine the data integrity for database applications. During a dedicated loadtest I created a simulated drive failure by marking one or more OSDs as *out*. This triggers a redistribution and replication of affected data objects as well as a change of the PG-Mapping. Ceph handled this process transparent for the client without any data loss.

The MGR-Node creates an output of the cluster state, where the state of the placement groups is logged:

```
1 undersized+peered, 3 active+undersized, 42 active+clean, 7
    active+clean+laggy, 7 stale+active+clean, 20 peering, 5
    undersized+degraded+peered, 12 active+undersized+degraded; 1.1 GiB
    data, 7.0 GiB used, 27 GiB / 34 GiB avail; 9.6 MiB/s wr, 2 op/s;
    105/1119 objects degraded (9.383%)
```

This is an example log output showing the status of a degraded and recovering cluster. Each unit depicts a placement group. **Active+Clean** is the desired state, whereas **stale** depicts an unresponsive pg. This could be a network issues or a failed host, like in this case. If the log shows **peering**, it is undergoing the process of recovering the **degraded** data objects. During this process some pg's might have the temporary state of **active+undersized**, when the data shuffling is not yet complete and imbalances between the disks are existing. This is shown in the ceph OSD overview:

```
ceph osd df tree
```

Which brings up a general overview of the OSD status, which can be viewed here [24]. Interesting here is the last line:

```
MIN/MAX VAR: 0.66/1.36 STDDEV: 6.22
```

This shows the imbalance of the pg placement on the different disks. The cluster could be configured to prefer certain disks for certain pgs, as with the metadata pool. This would favor faster disks or hosts with a better connection speed. These are possbile considerations for cost vs. performance optimizations.

Conclusion Ceph handles failure of disks gracefully and transparent as long as the failure can be contained and not too many disks fail. With a failed disk, the cluster will eventually reach a stable state after some time, when the data objects are rearranged on the disks. The resulting state will have less storage than the previous one due to storage being traded for stability. Therefore it is best practice to not fill the clusters up completely. For databases this will result in temporarily reduced I/O performance but protects from data getting corrupted.

4 Conclusion

In this section I want to summarize the advantages and disadvantages of running a ceph-cluster by means of a containerized daemon structure.

4.1 Advantages

Ease of configuration Once I figured out how to set up the OSDs and other daemons, it is fairly straighforward to set up a working cluster. I managed to compress this process into a single script [25]. The vanilla docker image is already configured in a way, which doesn't require extensive configuration beyond specifying the container parametres and preparing the disks.

Orchestration by using containers, the cluster could potentially be controlled on a higher level by means of orchestration tools such as *kubernetes* or *docker-swarm*. There also exists several solutions to abstract the cluster management: *ceph-ansible* [17] and *rook* [18]. While reading through several issues on Github, it became clear, that these are very well used in production environments, which is a sign that these solutions are matured enough to be considered stable and performant.

Updates Since every container is stateless and ephemeral, the update process is in theory easier than an application running on bare metal. To update a container, it is sufficient to change the container-definition. However, how an upgrade of a whole cluster is carried out in general is beyond the scope of this paper.

Shared KV-Backend The docker image provides a way to use *etcd* as a KV-storage[20] to share configurations and keyrings. This is handy to setup a cluster on multiple machines and distribute the cluster configuration automatically. This wasn't in the scope of my experiments, since all containers in my case were running on the same host and had access to the same harddrive.

Multiple Daemons on the same host In general it is not a problem to launch multiple MON, MGR, OSD and MDS containers on the same host. In fact, if orchestrated with kubernetes for example, it may happen - if not explicitly specified otherwise - that one worker-node[21], hosts two or more monitor containers. However, as my test in section 3.2 indicates, it is not recommended to run multiple OSD containers on the same host for reasons stated, if the host might suffer certain performance constraints.

4.2 Disadvantages

Documentation Although the Docker-Image has a fairly comprehensive documentation, I found that this approach is not very well documented in contrast to the usual *bare-metal* usage of ceph. All official documentation is written on the assumption that the cluster is running directly on the host machine. Github Issues and Stackoverflow provide some degree of help but it is not as comparable to the official documentation by ceph and RedHat. An excpetion of this are the ready-to-use distributions with *ceph-ansible* and *rook*, which proofed to be helpflul in some cases and have a relatively extensive documentation.

Performance Although it should be obvious, it is clear that running such an application which relies on being "as-close" to the hardware as possible, comes with a perfomance penalty regardless of the degree of optimizations. If this really makes a huge difference on beefy machines is beyond the scope of this paper but should be taken into consideration nevertheless.

Persistence As already explained in section 2.0.1, containers are by principle ephemeral and stateless. Although persistent volumes and KV-Backends alleviate these properties for the most part, it creates some additional overhead, which wouldn't exist otherwise.

[20] Key-Value
[21] A host which runs kubernetes services

4.3 Performance

Although I was not able to compare the containerized solution to a cluster, which runs on *bare-metal*, the nature of containers allows for the educated guess that this solution at least will never be as performant. The extent of the difference however will depend on the machine, the cluster is running on.

There are several factors, which might have a bigger overall impact on the performance of the whole cluster, as I learned over the course of my reasearch. Optimization on these points might even be more effective in increasing performance such that the small performance penalty of running a containerized environment is worth the extra benefits. Possible optimizations could be:

Running the MDS-Damon on fast SSD-based storage Since the MDS only serves small amounts of data but a large quantity of files, it is totally feasible to use smaller but faster SSDs. Best performance would be reached by using modern nVME drives. Within the cluster configuration the placement groups for the metadata can be configured to be stored on these special drives. It is best practice to have a limited amount of users per any given MDS, because having a fast responding MDS is crucial for good cluster performance.

Use optimized Applications Ceph does not only offer a filesystem, wich allows for the cluster to be used as virtual disk or block device but also libraries and API's to be directly implemented in clientside applications. The RedHat study [4] used an optimized mySQL distribution, which directly used the ceph kernel module to circumvent the filesystem bottleneck and yielded excellent results.

Tune the cluster configuration This might include tweaking the configuration for pool sizes, placement groups, stripe configs, object sizes and safety configuration such as crush map to the physical structure of the cluster in order to achieve best performance. The official ceph homepage provides extensive insight on how to achieve optimal performance: [26].

4.4 In Summary

To conclude my research goals pointed out in section 1.2, I found that like the study carried out by RedHat [4] a ceph cluster is well suited to be used as storage for a database in terms of data integrity. My test of removing OSDs during the mysql load test, showed that ceph handles failing disks in critical applications very well. However the fact that the OSDs are hosted within a containerized environment doesn't make a difference here.

However I was not able to show a performance gain for reasons stated in section 3.2. On the administration part I found that using containers drastically descreased the work necessary to set up a cluster. However it cost me more work upfront to figure out the correct configuration. Therefore I'd conclude it does make sense for a huge production environment, where administration is a bigger concern than clusters which are set up once and subsequently aren't touched again as it is the case in smaller or mid-sized deployments.

References

[1] [Online]. Available: https://i2.wp.com/svbtleusercontent.com/mqwrzstn0uje0q.png?ssl=1

[2] S. A. Weil, "Ceph: reliable, scalable, and high-performance distributed storage," Ph.D. dissertation, University of California, Santa Cruz, 2007.

[3] "Ceph releases (index)." [Online]. Available: https://docs.ceph.com/en/latest/releases/

[4] "Deploying mysql databases on red hat ceph storage." [Online]. Available: https://www.redhat.com/de/resources/mysql-databases-ceph-storage-reference-architecture

[5] S. Hong, D. Li, and X. Huang, "Database docker persistence framework based on swarm and ceph," in *Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference*, 2019, pp. 249–253.

[6] S. L. J. . 2021, J.-L. de Morlhon December 16 2020, and B. D. S. P.-G. N. . 2020, "Are containers replacing virtual machines?" Aug 2018. [Online]. Available: https://www.docker.com/blog/containers-replacing-virtual-machines/

[7] [Online]. Available: https://landscape.cncf.io/

[8] Github - ceph/ceph: Ceph is a distributed object, block, and file storage platform. [Online]. Available: https://github.com/ceph/ceph

[9] J. Hadlich, "Ceph object storage at spreadshirt (july 2015, ceph berlin meetup)," Jul 2015. [Online]. Available: https://www.slideshare.net/jenshadlich/ceph-object-storage-at-spreadshirt-july-2015-ceph-berlin-meetup

[10] "Placement groups¶." [Online]. Available: https://docs.ceph.com/en/latest/rados/operations/placement-groups/#:~:text=Aplacementgroup(PG)aggregates,onaper-objectbasis.

[11] "Architecture¶." [Online]. Available: https://docs.ceph.com/en/latest/architecture/#high-availability-monitors

[12] "Ceph." [Online]. Available: https://www.thomas-krenn.com/de/wiki/Ceph

[13] "Ceph manager daemon¶." [Online]. Available: https://docs.ceph.com/en/latest/mgr/

[14] Docker hub. [Online]. Available: https://hub.docker.com/r/ceph/daemon

[15] Github - codebaard/dockerceph: Everything necessary to setup and run a docker based ceph cluster. [Online]. Available: https://github.com/codebaard/DockerCEPH

[16] "Crush maps¶." [Online]. Available: https://docs.ceph.com/en/latest/rados/operations/crush-map/

[17] ceph-ansible — ceph-ansible documentation. [Online]. Available: https://docs.ceph.com/projects/ceph-ansible/en/latest/index.html

[18] Rook docs. [Online]. Available: https://www.rook.io/docs/rook/v1.6/ceph-storage.html

[19] Create a ceph filesystem — ceph documentation. [Online]. Available: https://docs.ceph.com/en/nautilus/cephfs/createfs/

[20] Chapter 7. ceph performance benchmark red hat ceph storage 4 — red hat customer portal. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/4/html/administration_guide/ceph-performance-benchmarking

[21] Dockerceph/resultscbench.txt at main · codebaard/dockerceph · github. [Online]. Available: https://github.com/codebaard/DockerCEPH/blob/main/result_scbench.txt

[22] Hardware recommendations — ceph documentation. [Online]. Available: https://docs.ceph.com/en/latest/start/hardware-recommendations/

[23] Docker hub. [Online]. Available: https://hub.docker.com/_/mysql/

[24] Dockerceph/osdoverview.txt at main · codebaard/dockerceph · github. [Online]. Available: https://github.com/codebaard/DockerCEPH/blob/main/osd_overview.txt

[25] Dockerceph/startup.sh at main · codebaard/dockerceph · github. [Online]. Available: https://github.com/codebaard/DockerCEPH/blob/main/scripts/ceph-provisioning/startup.sh

[26] Ceph making ceph faster: Lessons from performance testing - ceph. [Online]. Available: https://ceph.com/planet/making-ceph-faster-lessons-from-performance-testing/