Running a CEPH-Cluster from a containerized infrastructure

Performance test with a mySQL-Database

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

April 2021

Contents

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 administion are scrutinized. Major findings are that ... [insert findings here later] ... . Therefore running a distributed CEPH-storage in a containerization environment is ... [draw conclusion] ... .

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 exist several options to adress 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 and speed also cost effectiveness, vendor lock-in, complexity and granular customizability.

In general 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 *in principle* less performant. However this can be mitigated for the most part by scaling up.

Apart from propretary cloud storage providers i.e. AWS, Azure or Google, the **free market [correct term?]** is heavily dominated by CEPH by more than twice as much as the next competitor:
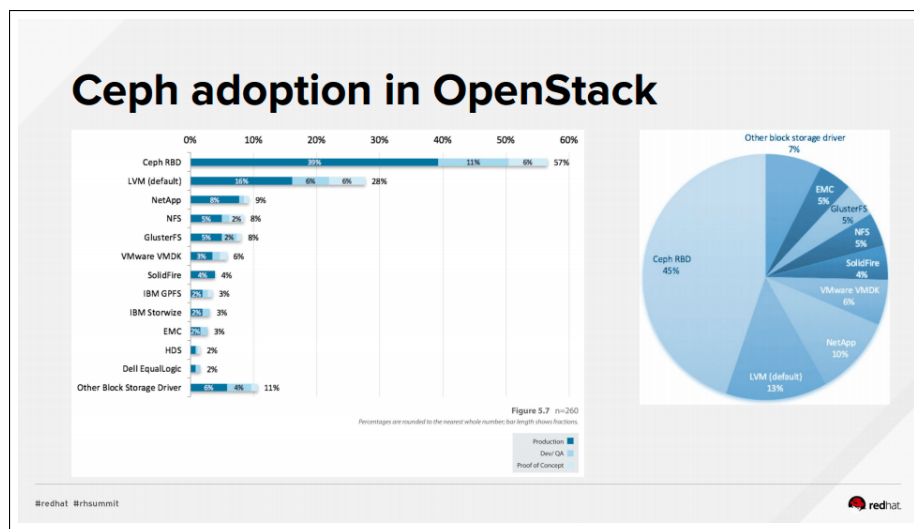

Figure 1: Adoption of CEPH in OpenStack in 2016, [1]

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. **Source?**

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 decreased the complexity, which is inherent to deploying several different applications to one single host machine. **[/Refactor this section /Since when?]**

Lastly, no storage solution exists without its use case. One major application, which requires flexible scaling are databases e.g. relational databases. To guarantee deterministic behaviour, database servers need to have specific properties to be suitable as database providers. The underlying storage solution is one of these.

1.1 Problem domains

Managing a highly available storage cluster is not a trivial task. 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 tweak the configuration of these tools in order to make it work on a complex

---

[1]Input/Output Operations per Second

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 early 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 means of using an *orchestration software*. Therefore the virtualized production environment has to be configured in a way that allows is 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 of chapter **??**.

1.2 Definition of research goal

The goal of this research is to evaluate if setting up a CEPH-storage cluster with containers is possible and if so, if it is a feasable option for a production environment. In order to make a conclusive assessment, three main points have to be examined. In order to draw a meaningful conclusion, these three main topics must be evaluated in contrast to a *non*-containerized cluster.

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. As for reasons which will be discussed in chapter **??**, 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 next to data integrity the second most important concern. Depending on the overall cost structure of the environment, a performance penalty might outweigh the benefits. In this case a containerized cluster would be *technically* possible but not economically feasable. 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 reasearch 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 **??** will try to generate more abstract metrics for an objective evaluation.

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 wheter it is feasable to run a SQL-Database on a CEPH-Cluster [4]. They concluded that indeed running a database on an optimized

---

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

CEPH-cluster exceeds industry standard database solutions. Furthermore scaling horizontally is easier 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 chapter1.1 that containers aren't persistent ways to store data [5]. However this paper does not use a CEPH-cluster to host a database and has therefore only very little relevance to the topic of this paper.

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.

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 paper is explained. This second section will make a small detour into containerization, what it means, what the benefits and 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 definition of what a container or more general containerization. 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 terms of application development or microservices especially in the realm of CI/CD[6] this is an uneccessary overhead. Containers on the other hand are in comparison small entities, which contain only the neccessary components of an application or service to run.
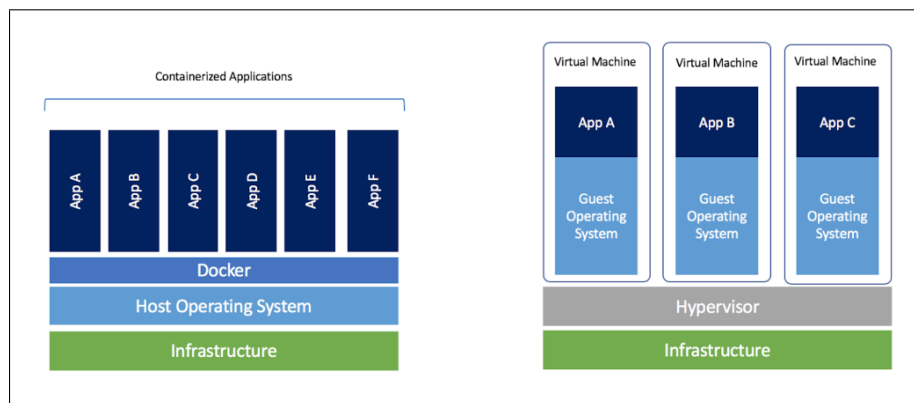

Figure 2: Containerization compared to Virtualization, [6]

By deploying applications or services this way, problems such as conflicting software versions or applications can be encapsulated, hence 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 early 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 because it is open source[**Github Link**] 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 deamons, which are discussed in detail in the following sections.
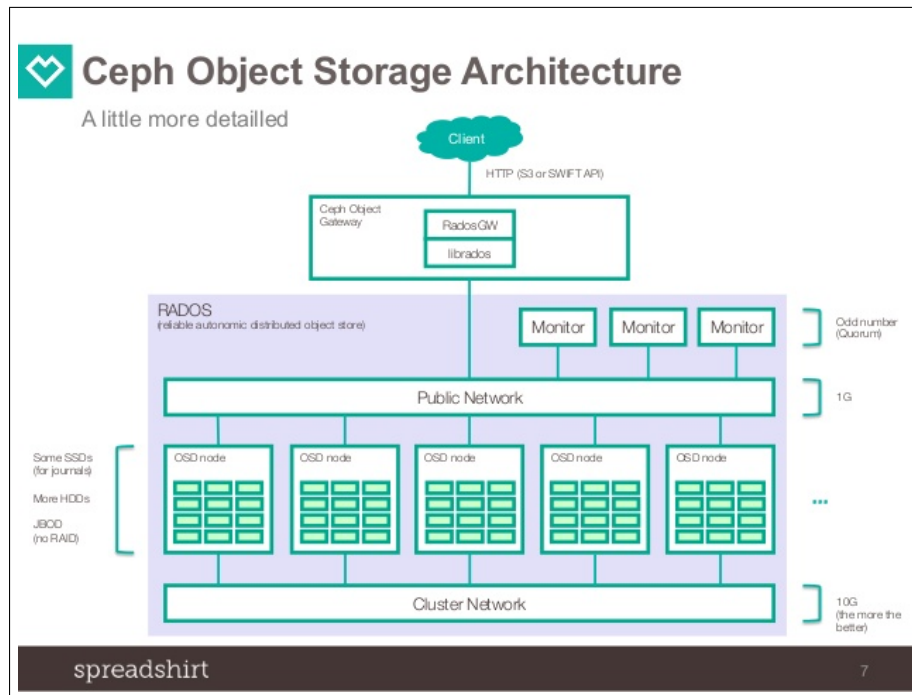

Figure 3: CEPH Architecture, [8]

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 *PGs*) are a 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 one 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 [9]. 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 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 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, and provides features like snapshots. A required metadata service called *MDS* provides journaling functionality and handle multi user data access. Because the throughput of the whole cluster scales linearly with the size of the cluster, often several different MDSs are required to handle the load and distribute the file metadata mitigate single points of failure. Is is best practice store and server the MDS from fast solid state drives to migigate 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 storage buckets. Unlike filesystems or block devices, these storage 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. This way of storing data is also compatible to Amazons S3[11] product. A widely adopted usecase 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.

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.

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 Placement Groups. 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.

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 this map might not reflect the state of the cluster accurately. Therefore an odd number of MON nodes have to agree 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 [10]. Although a cluster could technically

---

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

work with just one MON node, it is highly recommended to have at least three MON nodes, preventing a single point of failure.
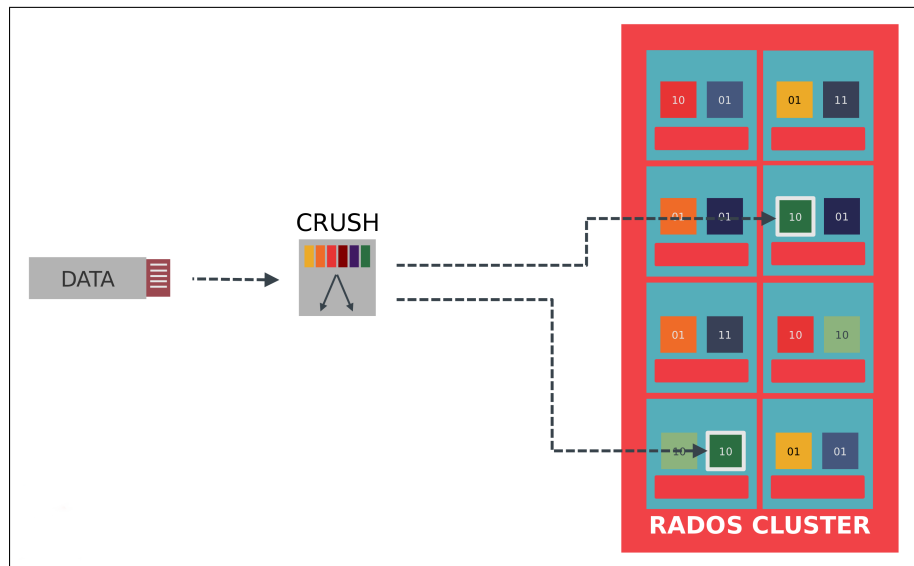

Figure 4: Placement Groups and CRUSH Algorithm, [11]

2.1.4 Metadata Server - MDS

The metadata server is not neccessary 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 [12]. 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 [12].

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 ressources in order to do my experiments. Since these are somewhat limited I configured a few things different than it would be done in reality otherwise.

I assume that the reader has a basic understanding of Docker and how commands are executed or how containers work in detail. Details which are immaterial to this topic will not be explained in great depth in the scope of this work.

2.2.1 Hardware Diagram

The cluster is distributed across three different machines for storage. Since CEPH isn't available as Docker Image for 32bit and 64bit ARM 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 bottlenecks weren't available CPU performance and RAM the performance impact of running a few more containers is negligible.

Figure 5: Hardware Diagram

Furthermore is the Docker host a server in production, therefore a certain amount of around 10% 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 seperate networks: one for replicating data between the OSDs and one to access the cluster for data 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 in the stock Docker Image. The official image **INSERT LINK HERE** 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 **insert link to docher hub here**. 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 seperate 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 **INSERT CONFIG AND REPO HERE**.

---

[12]As of 04.2021 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 **refer to section**. This is not neccessary for the other nodes since the configuration will be distributed after the monitor nodes are online.

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

Manager Nodes   This containers runs in parallel to a monitor node and doesn't need any special configuration.

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

OSD Nodes   The configuration of the OSD nodes is compared to the other nodes more complicated. 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 pre-treated. A prerequisite is that any disk has to be completely wiped of any partition table. This is done by executing *sgdisk* on every single in question.

```
docker run --rm \textbackslash
--privileged=true \textbackslash
-v /docker/ceph/data/lib:/var/lib/ceph/ \textbackslash
-v /var/log/ceph:/var/log/ceph/ \textbackslash
-v /docker/ceph/data/etc:/etc/ceph \textbackslash
-v /dev/:/dev/ \textbackslash
-v /run/lock/lvm:/run/lock/lvm:z \textbackslash
-v /run/lvm/:/run/lvm/ \textbackslash
-v /var/run/udev/:/var/run/udev/:z \textbackslash
--ip=192.168.178.230 \textbackslash
--network=macvlan0 \textbackslash
--entrypoint ceph-volume ceph/daemon lvm prepare \textbackslash
--bluestore \textbackslash
--data /dev/sdX \textbackslash
--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 \textbackslash
--privileged=true \textbackslash
--pid=host \textbackslash
-v /docker/ceph/data/etc:/etc/ceph \textbackslash
-v /docker/ceph/data/lib:/var/lib/ceph/ \textbackslash
-v /dev/:/dev/ \textbackslash
```

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

There are some important things to consider here: firstly the *–pid=host* configuration is neccessary to prevent that the host system runs out of open file handles. This is a bug, which has to be fixed**INSERT LINK TO GITHUB ISSUE HERE**. The *–privileged=true* setting is neccessary 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 phyisical 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 going to be created. In my case I had to create the filesystem manually from within the monitor container by means of a *docker exec* command. Therefore this flag has to be set to "0" in this case.

```
docker run -d \textbackslash
-v /docker/ceph/data/lib:/var/lib/ceph/ \textbackslash
-v /docker/ceph/data/etc:/etc/ceph \textbackslash
-e CEPHFS_CREATE=0 \textbackslash
-e MDS_NAME=mds-$(hostname)-a \textbackslash
--ip=192.168.178.245 \textbackslash
--network=macvlan0 \textbackslash
--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 [13]. 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 sophisticated rack mounted servers, where a multitude of servers comprise 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 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 feasable. In the context of the system architecture in the scope of this paper (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. [**Revisit this section after finished setup**]

2.2.4 Creating the cluster

Although there are different solutions to automate the deployment of a cluster such as *CEPH-Ansible***ADD LINK** or *CEPH for Rook***ADD REPO** my cluster is rolled out and configured with a simple shellscript - *startup.sh* - 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

```
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 following lines are only the additional monitor and mds nodes being created. These are not strictly neccessary 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 for such a cluster. As for the raw read/write performance test I can only evaluate the performance of the cluster at hand for reasons mentioned in 2.2. However since my server has different storage technologies, I can evaluate the database performance with the cluster, an HDD and SSD.

3.1 SCBENCH

This tool measures the raw performance of a CEPH cluster by performing a write and subsequent read test sequential and random**REF TO REDHAT**. 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

My goal is to load the cluster in a way that eventually all caches and short term performance enhancements become saturated to gauge the stable long term performance. What exactly scbench does under the hood, is described in great detail under **RESEARCH LINK**.

This test yields three different performance parametres for read and write actions:

- Bandwidth

- IOPS

- Latency

For reading the test was executed for sequential and random access of the data. However the tests for sequential read access weren't sufficient enough to evaluate, 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 [ADD REPOLINK]

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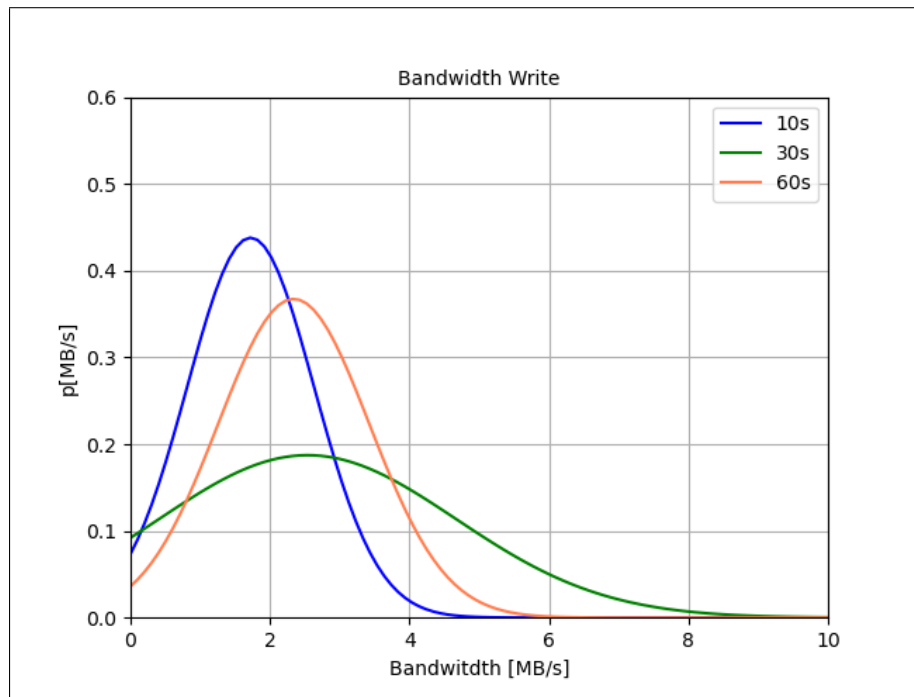

Figure 6: Normal distribution of the writespeed for three different tests

Figure6 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. However after the different caching entities within the cluster become saturated, the writespeed eventually stabilizes at around 2.5MB/s.
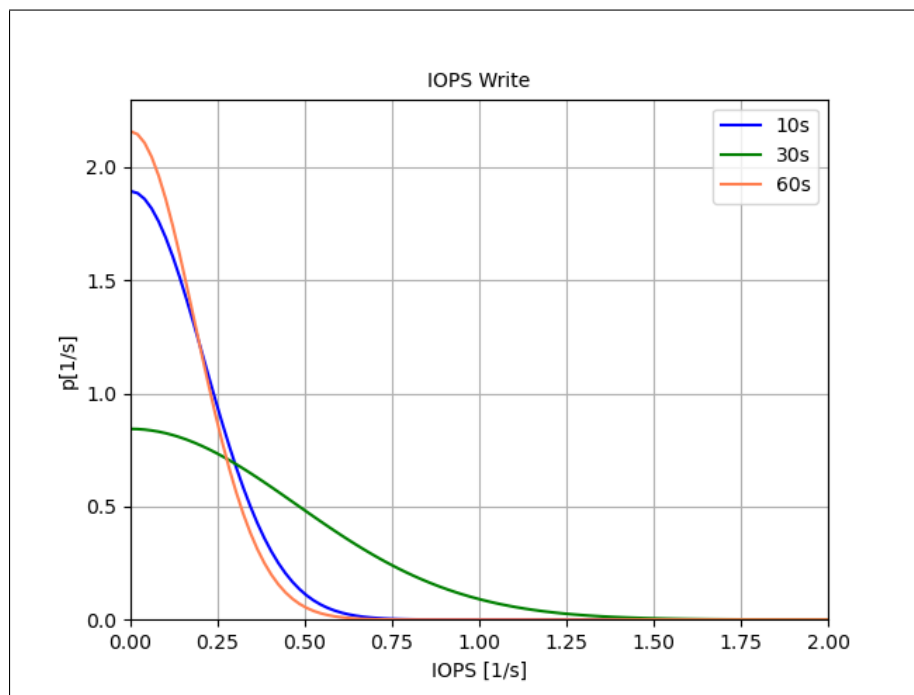

Figure 7: Normal distribution of the write IOPS for three different tests

The write IOPS in figure 7 exhibits the same result: Initial IOPS start slow due to the cost of initializing, subsequently increasing as the initialization has a smaller impact on the results but with saturation decreasing again.
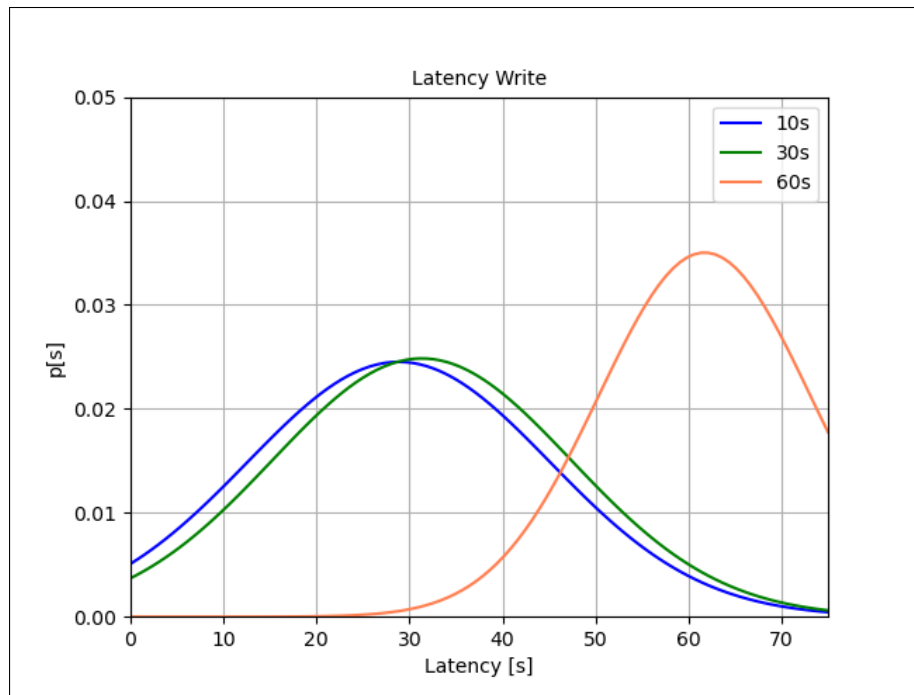

Figure 8: Normal distribution of the write latency for three different tests

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[**INSERT NOTE TO SETUP HERE**].

Read performance   This test reads the previously created data during the write test randomly.
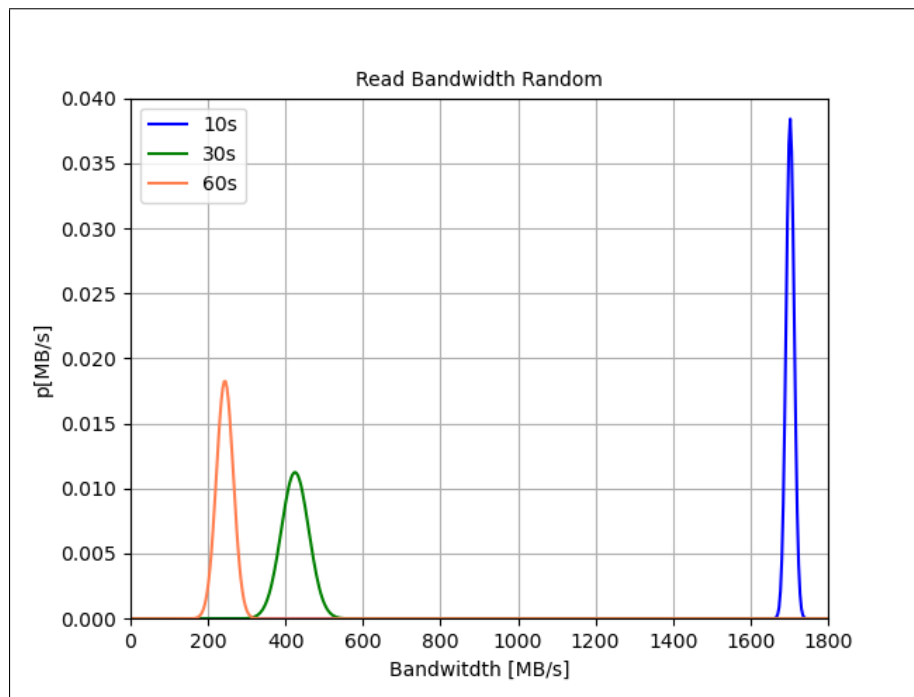
Figure 9: Normal distribution of the readspeed for three different tests

The read spead 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 within the RAM. In fact the hardware requirements for a CEPH-Cluster demand around 2GB of RAM per OSD**REFERENCE HERE**.
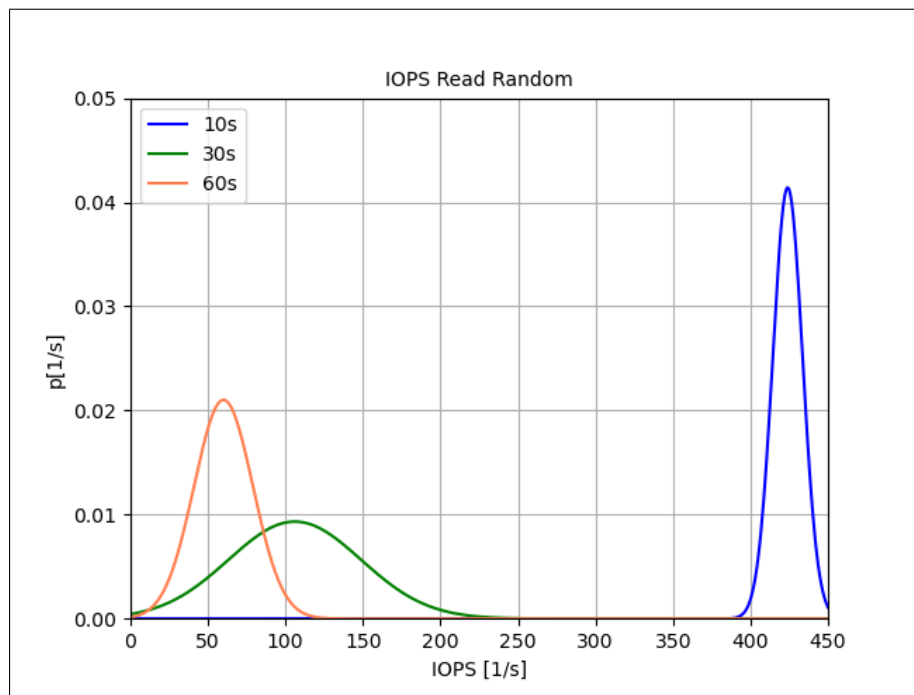

Figure 10: Normal distribution of the read IOPS for three different tests

The IOPS paint 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 presume that it is the timespan, where some disks cache is 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

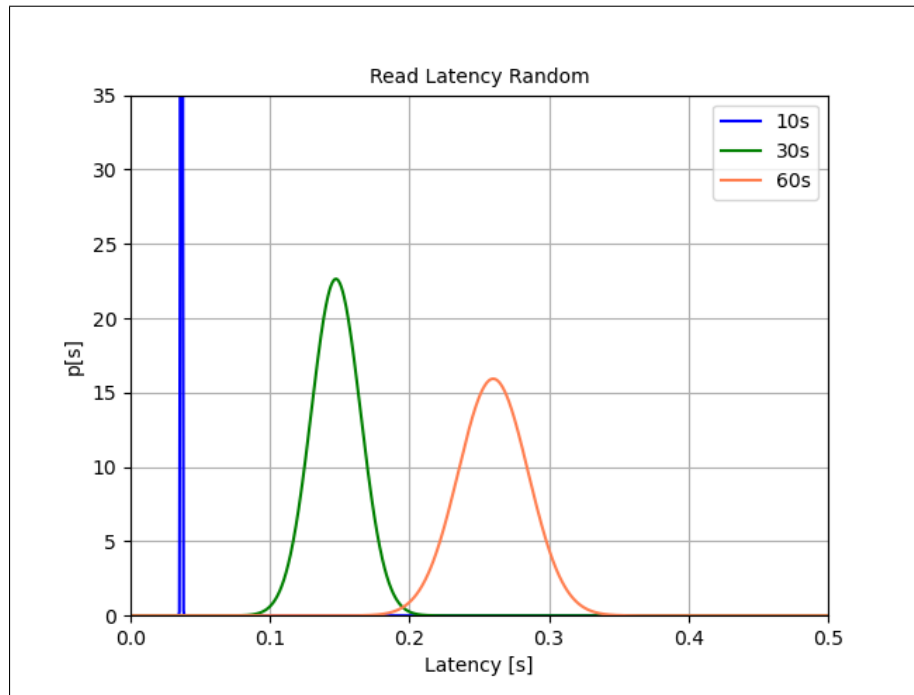aren't that fast either. This results in the value stabilizing after around a minute.

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.

4 Mysqlslap

some lorem ipsum going on here...

5 Conclusion

5.1 Advantages

5.2 Disadvantages

5.3 Performance

5.4 In Summary

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] 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

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

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

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

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

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