PROJECT

Multi-tenant cybersecurity operations centre designed as an open cloud service with machine learning elements

Linked to work package KPB1 – Design of the virt. solution

Linked to output
V3 – description of the selected solution meeting the defined criteria

CONTENTS

INTRODUCTION **4**
1. WHAT IS DISTRIBUTED STORAGE? **5**
1.1 PRINCIPLES AND ADVANTAGES OF DISTRIBUTED STORAGE 5
1.2 USE OF DISTRIBUTED STORAGE 6
1.3 REASONS FOR DEPLOYING DISTRIBUTED STORAGE 7
2. SELECTION OF DISTRIBUTED STORAGE SOFTWARE FOR OPENSTACK **8**
2.1 CRITERIA FOR SOFTWARE SELECTION 9
2.2 AVAILABLE OPTIONS 9
2.3 SOFTWARE SELECTION 10
3. CEPH AND THE ARCHITECTURE OF CEPH SOFTWARE **12**
3.1 CEPH SOFTWARE 12
3.2 KEY COMPONENTS OF CEPH SOFTWARE 13
3.2.1 CEPH CLUSTER 13
3.2.2 CEPH MONITOR 14
3.2.3 CEPH MANAGER 15
3.2.4 OSD DAEMONS 16
3.2.5 MDS AND CEPH FILE SYSTEM 17
3.3 RADOS 18
3.4 CRUSH ALGORITHM 21
3.5 POOLS AND PGS 22
3.5.1 POOL 23
3.5.2 PG 24
4. DEPLOYMENT OF A CEPH CLUSTER IN THE TEST ENVIRONMENT **26**
4.1 TEST ENVIRONMENT 26
4.2 SERVER PREPARATION 27
4.3 DEPLOYMENT OF THE MONITOR NODE 28
4.4 DEPLOYMENT OF THE MANAGER 31
4.5 DEPLOYMENT OF OSD NODES 34
5. DEPLOYMENT AND INTEGRATION OF THE CEPH CLUSTER USING OPENSTACK CHARMS TOOLS **36**
5.1 DEPLOYMENT CONCEPT 36
5.1.1 TOPOLOGY 37
5.1.2 VIRTUAL SERVERS 38
5.2 PREPARATION OF THE MAAS AND JUJU SERVERS 39
5.3 CEPH COMPONENTS IN THE BUNDLE FILE 40
5.3.1 CEPH MON 41
5.3.1.1 APPLIED CONFIGURATION 41
5.3.1.2 FURTHER OPTIONS [48] 42
5.3.2 CEPH OSD 42
5.3.2.1 APPLIED CONFIGURATION 42
5.3.2.2 FURTHER OPTIONS [48] 43

5.3.3 CEPH FS 43
5.3.3.1 APPLIED CONFIGURATION 44
5.3.3.2 FURTHER OPTIONS [48] 44
5.3.4 CEPH RADOSGW 45
5.3.4.1 APPLIED CONFIGURATION 45
5.3.4.2 FURTHER OPTIONS [48] 45
5.3.5 CEPH DASHBOARD 46
5.3.5.1 APPLIED CONFIGURATION 46
5.3.5.2 FURTHER OPTIONS [48] 46
5.3.6 CINDER CEPH 47
5.3.6.1 APPLIED CONFIGURATION 47
5.3.6.2 FURTHER OPTIONS [48] 47
5.3.7 GLANCE 48
5.3.7.1 APPLIED CONFIGURATION 48
5.3.7.2 FURTHER OPTIONS [48] 48
5.3.8 MANILA GANESHA 49
5.4 MODULE RELATIONS 50
5.5 DEPLOYMENT OF OPENSTACK WITH A CEPH CLUSTER 51
5.5.1 DEPLOYMENT VERIFICATION 52
6. CONFIGURATION OPTIONS OF CEPH 56
6.1 ADDING OSD DISKS 56
6.1.1 ADDING A NEW UNIT 56
6.1.2 ADDING DISKS TO AN EXISTING OSD UNIT 58
6.2 REPLACING AND REMOVING OSD DISKS 60
6.3 CEPH MULTI-BACKEND FOR CINDER 61
6.3.1 CREATING POOLS 62
6.3.2 ASSIGNING POOLS TO THE CINDER MODULE 66
6.3.3 CONFIGURING OPENSTACK TO USE MULTIPLE POOLS 69
6.3.4 FUNCTIONALITY VERIFICATION 71
6.4 MANUAL DOWNLOAD OF AN IMAGE FROM A POOL 73
6.4.1 EXPORTING USING OPENSTACK 73
6.4.2 EXPORTING USING THE CEPH CLIENT 74
7. RECOMMENDATIONS FOR DEPLOYING A CEPH CLUSTER 75
7.1 DEPLOYMENT CONCEPT 75
7.2 DEPLOYMENT ENVIRONMENT 75
7.3 CEPH CONFIGURATION 75
7.3.1 BUNDLE FILE CONFIGURATION 76
7.3.2 CONFIGURATION OF THE DEPLOYED CLUSTER 77
CONCLUSION 79
LIST OF REFERENCES 81

INTRODUCTION

In today's digital environment, efficient data storage and management are an essential prerequisite for technological development and for the operation of modern information systems. Developments in the field of data storage have undergone significant changes – from early low-capacity devices to today's large-scale data infrastructures processing data volumes in the range of petabytes and beyond. Such exponential data growth places increased demands on the capacity of storage systems, their availability and the preservation of data integrity.

Traditional centralized architectures show a limited ability to respond to growing requirements, mainly due to physical and geographical limitations. Shortcomings in scalability and redundancy cause problems in ensuring highly available services in the context of globally distributed cloud systems.

In this context, the implementation of distributed storage, which enables data to be spread across multiple nodes in different locations, appears to be an effective solution. Such an approach provides higher availability, increased resilience to outages and improved performance for parallel data access.

The theoretical part deals with the characteristics of distributed storage systems and their integration with the OpenStack cloud platform. On the basis of an analytical review of the options, suitable software was identified, and its architecture, functional modules and the basic concepts ensuring its correct functionality were examined in detail.

The practical part focuses on testing the selected solution, with an emphasis on deployment in a real environment. The initial steps include manual deployment of the system, followed by the use of the OpenStack Charms orchestration tools, which streamline the deployment process and facilitate integration with the cloud infrastructure itself. The analysis also covers configuration options, system management and the adjustment of parameters for specific operational requirements by means of the Charms tools.

The final part contains a proposed methodology and recommendations for integrating the selected software solution into the existing cloud architecture, including the specification of requirements for ensuring the reliability, scalability and performance of the distributed storage.

1. WHAT IS DISTRIBUTED STORAGE?

In a rapidly changing digital environment, data storage, data access and data management have undergone significant changes. One of the most important changes in recent years has been the shift to distributed storage systems – **distributed storage**. This approach to data storage is characterized by **decentralization**, **scalability** and **reliability**, and offers a comprehensive solution to the challenges posed by the exponential growth of digital information.

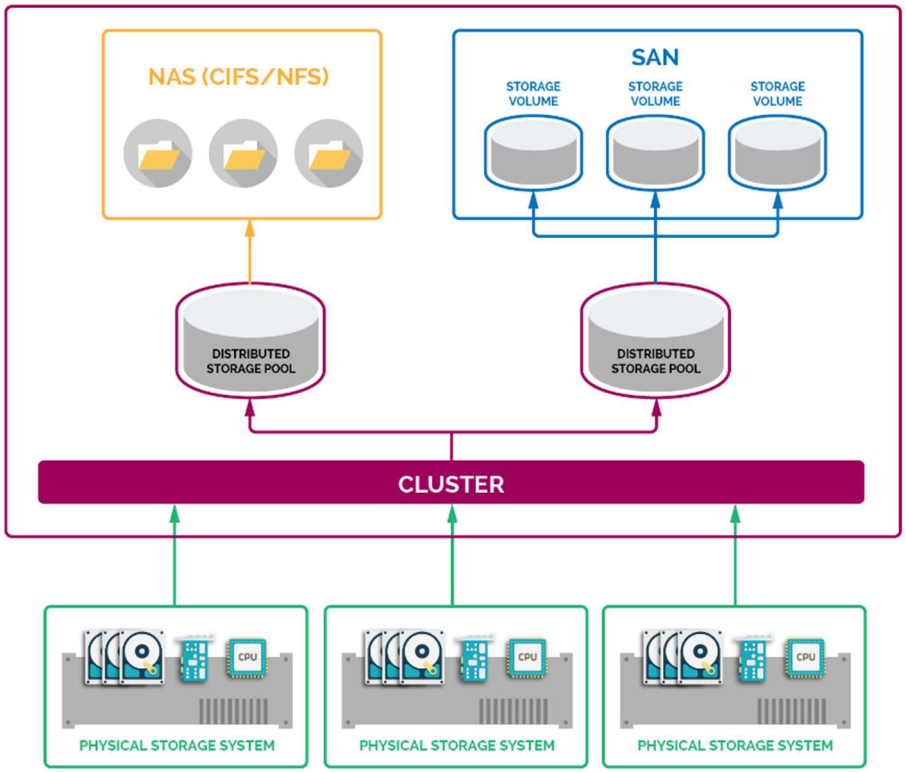


Figure 1 General diagram of a distributed storage system [1]

1.1 Principles and advantages of distributed storage

Distributed storage achieves decentralization by spreading data across multiple physical servers, which are often located in different geographical locations. These servers are interconnected via a network. This method contrasts with traditional storage models, which centralize data in a single location. The distributed model increases data access speed, reliability, scalability and fault tolerance.

Figure 1 above shows the basic diagram of distributed storage, where decentralization is represented in the form of multiple physical servers ("physical storage system") combined into a so-called **cluster** ("cluster"). Access to distributed storage can be provided by connecting to the cluster in the form of file-level network storage, **NAS (Network Attached Storage)**, or block-level network storage, **SAN (Storage Area Network)** [1][2][3].

The foundation of distributed storage lies in its ability to expand capacity seamlessly (scalability) and to partition and replicate data across multiple servers. This configuration enables efficient data management and parallel data access and processing, which significantly reduces bottlenecks. Data partitioning, or "**sharding**", is the division of data into smaller segments that are distributed to different storage servers. In addition, replication strategies ensure that multiple copies of the stored data are maintained on different servers, providing a robust layer of data protection and availability, thereby ensuring reliability. This replication guarantees that the system can access data from other undamaged servers even if one or more servers fail. The replication factor can be configured by administrators to determine the number of data copies that the system maintains. Higher replication factors increase data security but also require more storage space [3][4].

1.2 Use of distributed storage

Distributed storage is widely used in various sectors, including media, healthcare and big data analytics. It enables efficient handling of large multimedia content libraries, secure storage of sensitive patient data and management of large data sets for analytical processing [3][5].

Cloud platforms make extensive use of distributed storage owing to its scalability and fault tolerance. **Amazon S3** is the best example of distributed storage in practice, offering users highly available and secure storage solutions. Distributed storage is particularly useful in addressing large-scale data storage and processing needs and supports a wide range of applications. As data volumes grow, the importance of distributed storage will become even greater, leading to advances and improvements in data management technologies [5][6].

1.3 Reasons for deploying distributed storage

The reasons for deploying distributed storage for the cloud practically mirror all the advantages of distributed storage already mentioned above. The cloud is deployed on top of the OpenStack platform. The main advantage of cloud platforms is their versatility, the possibility of migrating virtual machines and their availability in the event of a device outage in the cluster. However, these properties are considerably limited if the storage for each given device is handled locally. As a result, when multiple clients access it, the demands on the particular server increase, which, in addition to slower request processing, increases the risk of failure of that server over time. In the event of such a failure, owing to local data storage, data loss and the inability to provide the services hosted on that server are very likely.

By deploying distributed storage in the cloud, the risk of data loss or service interruption is thereby significantly reduced. It is precisely the properties of distributed storage that ensure parallel data access and substantial data redundancy. The storage is no longer local, and thanks to data replication across multiple servers, no data is lost if they fail. Simple scalability ensures that any expansion of the department's cluster is considerably easier. Last but not least, the management and monitoring of the storage are also simplified. Distributed storage software provides tools for detailed management and monitoring that are not possible to such an extent in OpenStack.

2. SELECTION OF DISTRIBUTED STORAGE SOFTWARE FOR OPENSTACK

Selecting suitable software for distributed storage in OpenStack is crucial for several reasons. The right storage backend is essential for ensuring the optimal performance of the OpenStack cluster. If the storage cannot provide the input/output operations needed for the required cluster workload, this leads to performance problems that directly affect the functioning of applications. A well-chosen storage solution increases the efficiency of storage operation and thus also of the services provided. It can also simplify maintenance, backup and troubleshooting processes, whereas a poorly chosen solution may increase the workload of the administration team [7].

Just as with providing the necessary computing power, ensuring the reliability of the storage solution is directly related to the reliability of the entire cluster. Data reliability must therefore be the highest priority when designing a private cloud. Choosing the wrong storage solution may compromise this aspect. In addition, another key factor to be taken into account is cost-effectiveness. The storage solution should balance performance and cost attributes. Waste of valuable resources may result from a solution that increases overall costs without adequately meeting the computing performance requirements defined in the relevant SLAs (Service Level Agreements) [7][8].

Another important aspect in selecting software is understanding the nature of the data and how it needs to be accessed or distributed. For example, structured and unstructured data require different storage solutions, which significantly affects data access and storage. In addition, it is essential to consider the scalability and future growth potential of the storage solution. The selected storage should not only meet current needs but should also be scalable to accommodate future growth in capacity and performance requirements [8].

Selecting suitable distributed storage software for OpenStack requires a strategic decision-making process that affects the performance, reliability, and operational and cost efficiency of the entire cloud infrastructure. It is important to consider the specific needs and circumstances of each OpenStack deployment.

2.1 Software selection criteria

Developing a specific set of software selection criteria is an essential step in ensuring that the software we select corresponds to our stated objectives. This will allow us to quantify and compare the features and capabilities of different software.

The selected software must meet the following criteria:

- freely available for use
- support for object, block and file storage types
- scalability
- backup capability, appropriate redundancy of stored data
- compatibility with the OpenStack platform
- compatibility with OpenStack Charms tools

These criteria form the basic framework for selecting the software with regard to the requirements, infrastructure and architecture of a private cloud built on top of OpenStack.

2.2 Available options

In the case of software for OpenStack, there are three significant options in this area: **Ceph**, **GlusterFS** and **MinIO**, each of which has its advantages and disadvantages.

Ceph is a widely adopted choice that stands out for its seamless integration with OpenStack and offers robust block, object and file storage in a unified system, making it highly scalable and resilient. Likewise, the self-healing and self-managing capabilities of the Ceph software significantly reduce the administrative burden [9][10].

GlusterFS is a popular option owing to its simplicity and usability, providing a scalable network file system suitable for data-intensive tasks. Its ability to integrate with OpenStack through the **Cinder** and **Manila** modules for block and file

storage, respectively, makes it a versatile choice for diverse workloads [10][11][12].

MinIO specializes in high-performance object storage compatible with the S3 system. It is a computationally lightweight yet powerful solution, ideal for environments that prioritize simplicity and efficiency in processing large volumes of unstructured data. Its compatibility with the S3 API enables seamless integration into various cloud applications, including the OpenStack platform [13].

Each of the options, Ceph, GlusterFS and MinIO, represents a unique set of features. A summary of the given criteria and how the individual software products fulfil them is shown in Table 1 below.

Criteria	Ceph	GlusterFS	MinIO
Free/OpenSource	Yes/Yes	Yes/Yes	No support/Yes
Supported storage type	Object, Block, File	Object, Block, File	Object
Scalability	Yes	Yes	Yes
API support	RADOS, S3	S3	S3
Snapshot/Cloning	Yes	No	Not applicable
OpenStack Driver	Yes	No	No
Charms support	Full	No	Partial

Table 1 Comparison of criteria for the Ceph, GlusterFS and MinIO software

2.3 Software selection

After a thorough evaluation of the criteria and available options, we decided to choose **Ceph** as the software for distributed storage. One of the main reasons for this choice is that Ceph is an open-source platform, which corresponds to our preference for free, community-supported solutions. In addition, scalability is a significant advantage, as it can efficiently process data in up to the petabyte range, ensuring that the storage infrastructure can grow along with possible future needs without a significant decrease in performance. The integrated redundancy features of the Ceph software were also an important factor in our decision. These features are essential for maintaining data integrity and availability, especially in a distributed environment. The ability to replicate and recover data seamlessly provides the resilience that is necessary for providing cloud ser-

vices. Another reason for choosing Ceph is its native integration with the OpenStack platform, which is a key part of the department's cloud infrastructure. Ceph supports the **OpenStack Charms** tools, which facilitate management and deployment in the OpenStack environment.

As for other potential options, such as GlusterFS, it is currently not supported by the OpenStack platform, since the release of its Ocata version in 2017. The lack of support for GlusterFS may result in integration problems and cause potential incompatibility. Moreover, the OpenStack Charms tools do not support GlusterFS, which is one of the most important criteria for selecting the software. Both GlusterFS and MinIO have their strengths, but they are not the most suitable for use in the department's cloud. Although the design of MinIO focuses primarily on high-performance object storage compatible with the S3 system, it does not inherently include the level of redundancy that is required. This limitation could pose a risk to data resilience and availability. Compatibility with the OpenStack Charms tools is also highly questionable, as the official documentation does not include this variant.

3. CEPH AND THE ARCHITECTURE OF CEPH SOFTWARE

The following chapter deals with the **Ceph** software and its architecture. The chapter presents what Ceph is and the basic architecture of the Ceph software, with an emphasis on its key components, the **RADOS** technology and the **CRUSH** algorithm, which are the basis of its **efficiency** and **scalability** [14].

3.1 Ceph software

Ceph is an open-source distributed storage system managed by the **Ceph Foundation** within the **Linux Foundation**. Contributions from a wide range of organizations, including **Red Hat** and **IBM**, ensure its continuous development and improvement. It provides scalable and unified storage for block, object and file data and is designed to run on commodity hardware, which makes it a cost-effective and fault-tolerant solution for various data storage needs. Since January 2023, IBM has been responsible for the development, maintenance and support of Ceph, which was renamed **IBM Storage Ceph** for business – "**enterprise**" – use. This version includes enterprise features and support services with SLA agreements and is deployed using Linux containers for easier management [15][16].

Every year, with a target date in March, a new stable version is released, named in alphabetical order. Version numbers use the format **x.y.z,** where "x" denotes the release cycle (the position of the letter in the alphabet), "y" denotes the release type (0 for development, 1 for release candidates, 2 for stable/bugfix releases) and "z" denotes bugfix versions. Stable versions are updated every 4 to 6 weeks and are supported for **2 years** [17].

According to **HG Insights**, major companies such as **NVIDIA**, the logistics giant **Maersk Line** and the telecommunications company **Ericsson** also use Ceph as storage [18].

3.2 Key components of Ceph

The Ceph architecture consists of several key components, each of which plays a significant role in the functionality and performance of the system. These components work together to provide a storage solution that can easily be scaled and adapted to user needs, from terabytes to exabytes and beyond, without any disruption. In the following subchapters, we explain how the individual components work, what role they have in the **cluster** and what advantages they provide [14].

3.2.1 Ceph cluster

The basis of the entire Ceph system is the **cluster**. The cluster consists of **nodes**, which are the individual devices – servers – on which the Ceph software is installed. The individual cluster **components** are then deployed on the nodes. Creating a Ceph cluster is a complex process that integrates various components and creates a robust storage solution. These are **Ceph Monitor, Ceph Manager, OSD and MDS**, each of which plays a unique role in the cluster architecture. The orchestration of these elements enables efficient management of block storage, the file system and object storage within a distributed network environment. The cooperation of these components creates a comprehensive and adaptable data storage system capable of meeting complex data storage requirements in today's data environments [14].

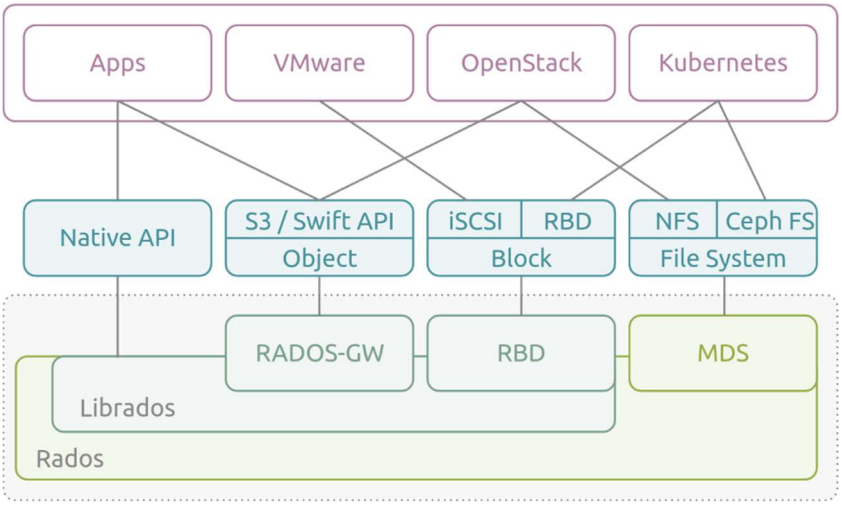


Figure 2 Ceph cluster architecture [19]

Figure 2 shows the architecture of the Ceph cluster, its components and the flexibility of its use. It provides extensive options for communication with clients, whether in the form of **API interfaces,** or by means of **gateways**, through which it provides storage for other platforms such as VMware, Kubernetes or, indeed, OpenStack.

3.2.2 Ceph Monitor

The **Ceph Monitor** (also referred to as "*ceph-mon*") is an important component in maintaining the functional state and stability of a Ceph cluster. Its main function is to provide a reliable and up-to-date view of the cluster state and to maintain the cluster maps. Specifically, these are the **monitor maps**, **administrative maps**, **OSD maps**, **MDS maps** and **CRUSH maps**, which are essential for the coordination and efficient management of the cluster. As a result, the cluster can adapt to changes, for example the addition or removal of nodes, without interrupting service provision. Monitors also play a key role in cluster management, including the authentication and authorization of clients and daemons, thereby increasing its security [14][20][21].

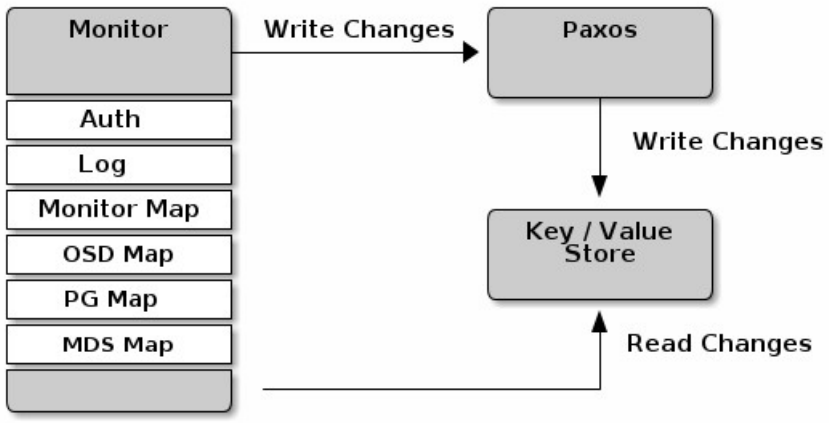


Figure 3 Illustration of the Ceph monitor tasks and the decision-making process [21]

Figure 3 shows the individual tasks of the monitor through which it ensures the smooth operation of the cluster with the help of the **Paxos** consensus algorithm. This algorithm enables the monitors to agree on the cluster state and make decisions jointly, which increases fault tolerance and thus the reliability of the

cluster. It is precisely the consensus nature of the algorithm that guarantees that the cluster can withstand failures and preserves both data integrity and availability, since more than half of the monitors must be in agreement for a decision to be made [14][21].

The use of multiple monitors guarantees that the cluster remains functional even if almost half of the total number of monitors fail. Monitors facilitate the scalability of the cluster, which makes it possible to accommodate more data and clients without a significant increase in management costs. The consensus mechanism enables the cluster to maintain a consistent state even in the event of a network partition or hardware failure. They oversee the **cephx** authentication system, which authenticates users and daemons and protects against attacks such as man-in-the-middle [14][20].

3.2.3 Ceph Manager

The **Ceph Manager**, or "*ceph-mgr*", is one of the key components, providing better options for monitoring and managing the cluster. Since version 12.x, it has become a mandatory component for normal cluster operation. The daemon runs alongside the monitor daemons, providing additional monitoring functions together with interfaces for external monitoring and management systems. It requires no further configuration apart from ensuring that it is running. Deployment tools such as "*ceph-ansible*" or "*cephadm*" are usually used to set up ceph-mgr daemons on the monitor nodes, although they do not necessarily have to be located on the same nodes as the monitors [22].

The architecture of the manager allows its functions to be extended through modules such as **dashboard**, **prometheus** and **balancer**. The dashboard module provides a built-in web interface for management, prometheus offers monitoring capabilities and balancer optimizes the distribution of **Placement Groups (PGs)** among OSD daemons. In addition to the modules for prometheus, the manager also provides modules for Zabbix, sending alerts, modules for orchestration and a RESTful API interface [22].

Among the greatest advantages of the manager for a Ceph cluster are precisely its modularity and the simplified interface for cluster management. This

modular approach allows administrators to adapt the management functions to their specific needs, thereby optimizing the performance and reliability of the cluster as a whole [22].

3.2.4 OSD daemons

Ceph **Object Storage Daemons** (**OSDs**) – **OSD daemons** are the most important components of the architecture and operation of a Ceph cluster. They serve as the basis for data storage, replication, recovery and monitoring of the cluster state. OSD daemons, also known as "*ceph-osd*", store data within logical **pools** and ensure data availability and system resilience through their roles in data balancing and health checking. They are essential for achieving high availability and fault tolerance, which facilitates the ability to process huge volumes of data efficiently [9][14].

Each physical disk or logical partition intended for data storage usually has its own OSD daemon. Data storage and replication in the cluster are carried out precisely by means of the OSD daemons, which store data as **objects** within **PGs**. To ensure data resilience and availability, Ceph uses a replication strategy that involves replicating objects across multiple OSD daemons. Their number is specified by the **replication factor**, whose value is given in the configuration. The distribution and management of data across OSD nodes is handled efficiently by the **CRUSH** algorithm. CRUSH enables dynamic placement of objects and can adapt to changes in the cluster topology without requiring manual intervention [14][23].

OSD daemons also include monitoring functions. At regular intervals, they send so-called "***heartbeat***" messages to each other and to the monitor to report their current state. This monitoring mechanism helps the cluster to quickly detect and respond to any OSD daemon failures. In the event of an OSD daemon failure, Ceph automatically starts data recovery processes whose task is to replicate the affected data from other OSD daemons. When new OSD daemons are added to the cluster, Ceph rebalances the data distribution in order to optimize storage utilization and performance [14][23].

The implementation of OSD daemons in the cluster provides numerous advantages in terms of scalability. By means of OSD daemons, the Ceph architecture enables smooth increases in storage capacity and performance. Adding further disks to the cluster increases the overall storage capacity and spreads the workload over a larger pool of resources, thereby ensuring high availability. Moreover, data replication between the individual OSD daemon units ensures that no single point of failure can jeopardize data availability. For this reason, the cluster is able to continue functioning even if some OSD daemons fail. Ceph uses replication or “erasure coding” to increase data resilience and protect against data loss due to hardware failure. They have self-healing capabilities that enable automatic recovery after a failure by redistributing data in order to maintain redundancy and performance without the need for administrator intervention [14][23].

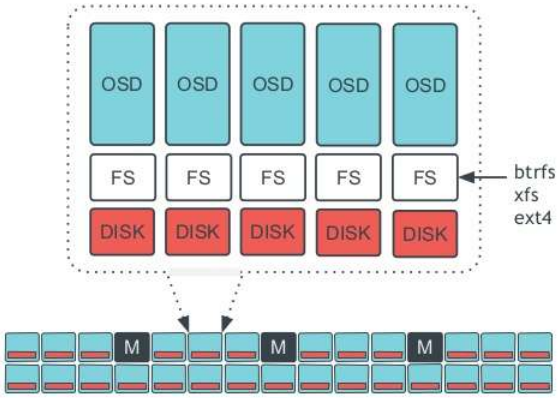


Figure 4 Illustration of OSD daemons on a node server [24]

3.2.5 MDS and Ceph File System

The **Metadata Server (MDS)** is a key component of the **CephFS (Ceph File System) distributed file system**. Its main task is to manage metadata for file operations such as opening, reading, writing and listing files. Unlike traditional data storage systems, which store file metadata on the same nodes as the data, Ceph uses dedicated **MDS** nodes to manage metadata. This choice improves the performance of metadata operations by allowing the storage nodes (OSDs) to concentrate primarily on storing data without the additional overhead of metadata management [25].

It is important to note that the MDS server is relevant only when using a shared file system such as **CephFS** or **NFS (Network File System)**. CephFS is a distributed file system that provides a **file storage** layer on top of the Ceph cluster. MDS is specifically designed to manage metadata for the CephFS system, which makes it a critical element for deployments that require a distributed file system. For other types of Ceph storage, such as block storage (RBD) or object storage (RGW), MDS servers are not needed and are therefore not used without the CephFS system [14][25].

MDS servers manage the CephFS "***namespace*"** and coordinate access to the storage itself provided by the OSD daemons. They process client requests concerning file metadata, thereby enabling efficient file system operations in a distributed environment. Metadata is stored in a dedicated part within the Ceph cluster, which enables fast access and management. This setup allows the Ceph system to provide clients with a file system interface that complies with **POSIX** standards and to support a wide range of file operations in a distributed system [26].

Separating the storage of metadata and data allows a cluster using CephFS to significantly optimize file-based operations. Processing metadata separately reduces the load on OSD disks, which leads to better performance and scalability of both data and metadata operations. The Ceph architecture allows metadata management to be scaled out by adding further MDS nodes, which is essential in large-scale deployments. This capability ensures that the system can efficiently handle a growing number of files and directories. CephFS also supports high availability through the deployment of multiple MDS servers in an **active/standby** configuration, thereby ensuring continuous access to files and minimizing outages in the event of an MDS failure. MDS servers increase the efficiency of metadata operations through techniques such as **caching** and **journaling**. Journaling is extremely important for ensuring consistency and fast recovery, as it records metadata changes before they are applied [14][25][26].

3.3 RADOS

RADOS (Reliable Autonomic Distributed Object Store) is the technology

that forms the basis of the Ceph distributed system and offers scalable, reliable and efficient data storage solutions. It is a distributed object storage system specifically designed to store large amounts of data in a cluster of commodity hardware. RADOS forms the core of the Ceph software and supports block, file and object storage interfaces. The system is designed for high performance, extensibility and reliability. It manages data replication, recovery and distribution without a single point of failure [14][27].

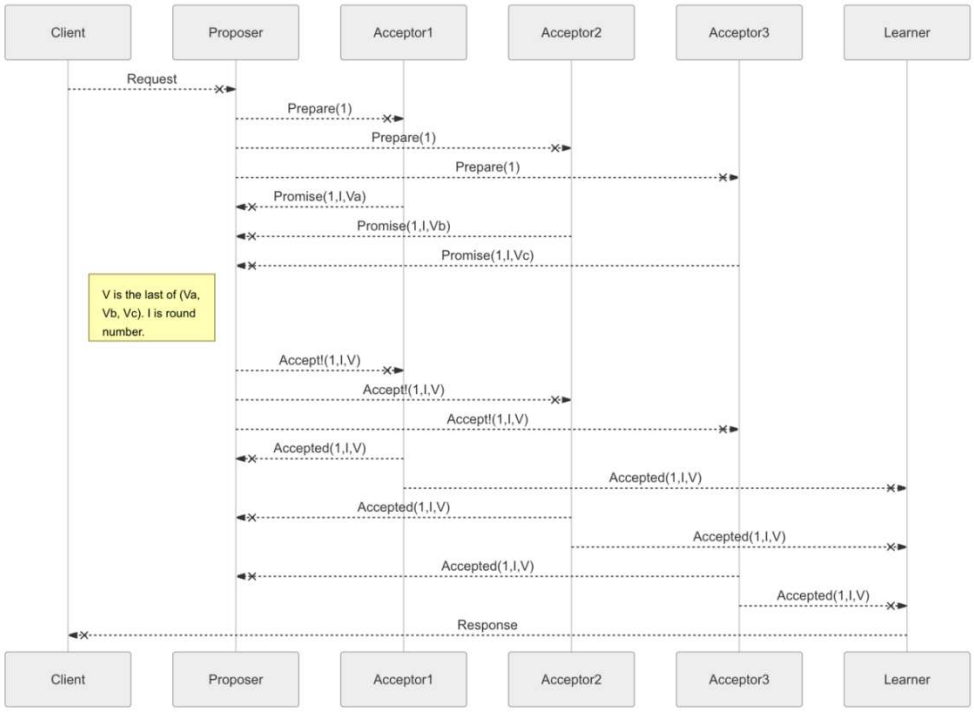


Figure 5 Principle of reaching consensus using the Paxos algorithm [28]

Since the architecture of a Ceph cluster is primarily built on RADOS with additional functions added, RADOS clusters likewise consist of two types of daemons, namely **OSD** and **Monitor**. It is RADOS that is the technology providing almost all of Ceph's well-known advantages for distributed storage. It uses the **CRUSH** and **Paxos** algorithms for the efficient distribution and management of data within the cluster, the principle of which is shown in Figure 5 above. RADOS also ensures data durability and availability through replication and, last but not least, through its self-healing property. In the event of a node failure, RADOS automatically redistributes the data to other nodes, thereby ensuring that the replication factor is maintained. This process is transparent to the user and thus ensures continuous data availability [14][27][29].

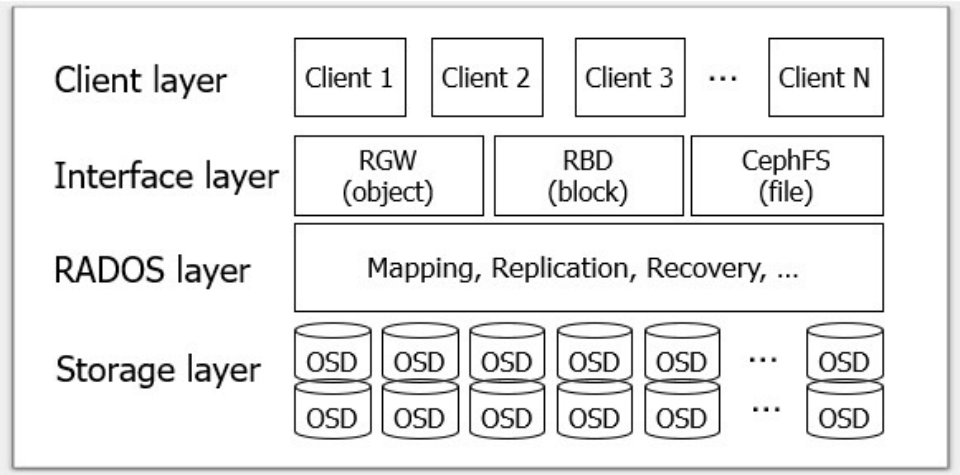


Figure 6 Representation of the individual layers of the Ceph architecture [30]

The logical division of a Ceph cluster into layers is shown in Figure 6 above. The client layer (**Client layer**) contains the applications, devices or platforms that use the distributed storage of the Ceph cluster. Access to the data is provided by the interface layer (**Interface layer**). Ceph offers **Ceph Rados Gateway (RGW)** – an object storage interface that provides applications with a RESTful API, **Ceph Rados Block Device (RBD)** for storing blocks and images, e.g. for virtual machines and virtualization platforms, and **CephFS**, which can be mounted into the operating system as an ordinary disk in the form of shared storage. **RADOS layer** – the RADOS layer is the control layer responsible for all the properties mentioned in the previous paragraph. The last layer is the storage layer (**Storage layer**), which consists of all the OSD daemons in the cluster [14].

In addition, in the interface layer Ceph also offers direct access for a native API by means of the "librados" library, as shown in Figure 2. The **librados** library provides an API that allows direct interaction with the RADOS storage backend. Applications built on it are thus able to perform operations directly on the Ceph cluster, which leads to efficient data storage and retrieval without the need for a client intermediate layer. Once connected, applications can perform various operations, such as reading and writing data, managing extended attributes (XATTR) and listing objects within the storage. Librados supports asynchronous I/O operations and provides libraries for several programming languages, such as C, C++, Python, Java and PHP [14][31].

3.4 CRUSH algorithm

CRUSH (Controlled Replication Under Scalable Hashing) is an advanced data placement algorithm that efficiently determines how data should be stored on the storage nodes of the cluster. As a result, Ceph clusters are highly resilient and scalable, because they distribute data in a way that maximizes availability and minimizes the need to rebalance data when the cluster grows or shrinks. Unlike traditional data allocation methods, which require a central directory to record the location of individual pieces of data, CRUSH allows Ceph clients to calculate the location of data directly using the cluster map. This significantly reduces overheads and the risk of potential bottlenecks, enabling Ceph to achieve reliability and performance when processing huge amounts of data across many storage nodes [14][20].

The CRUSH algorithm uses a hierarchical structure consisting of so-called "*buckets*" and OSDs to map data objects to physical storage locations. Each level of the hierarchy can contain several devices or other buckets. These in turn form a tree structure reflecting the physical layout of the storage environment. This structure is encoded in the so-called **CRUSH map**, which is distributed to all Ceph clients and OSD daemons. An example of a CRUSH map can be seen in Figure 7 [14][32].

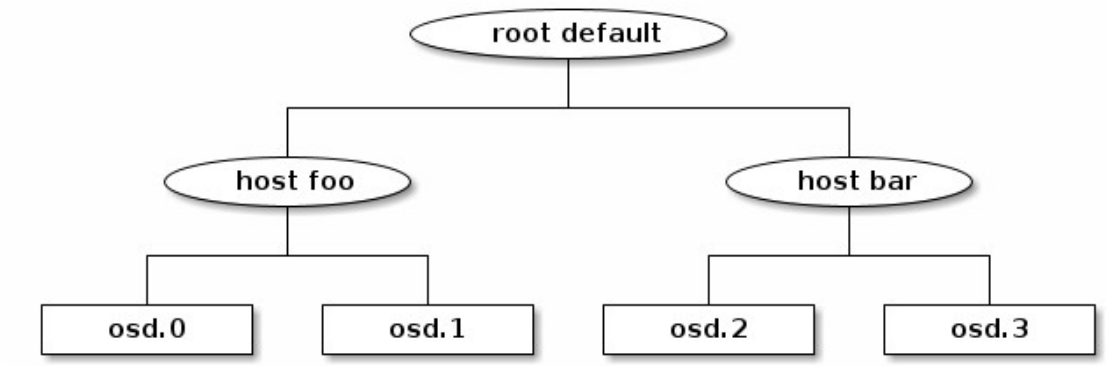


Figure 7 Example of a CRUSH map structure [32]

When a Ceph client needs to write data, it uses the CRUSH algorithm and the latest CRUSH map to calculate which OSD should store a particular data

object. This calculation takes into account the hierarchy, device weights and the replication or erasure coding rules defined for the data, thereby ensuring an even distribution of data within the cluster and compliance with the specified redundancy requirements [14][32].

By distributing data in a predictable yet random manner across multiple OSDs, CRUSH increases data availability and fault tolerance. It allows the cluster to rebalance data dynamically when OSDs are added or removed, which is crucial for large-scale data storage systems that must scale to accommodate growing data volumes without disrupting ongoing operations [14][20][32].

CRUSH allows Ceph clusters to scale out horizontally with minimal impact on performance by distributing the workload evenly across all available resources. It reduces latency and increases throughput by providing direct client communication with the OSDs without a centralized lookup table. Administrators can define their own hierarchical structures that correspond to their actual hardware layout, thereby optimizing performance and data durability [14][20][32].

3.5 Pools and PGs

Pools are the basic units in which objects reside within a Ceph cluster. They play a key role in orchestrating data distribution and resilience. **PGs (Placement Groups)** are a key element of the Ceph architecture that streamlines the organization, allocation and access to data in a distributed environment. This mechanism allows Ceph to scale easily while maintaining availability and a balanced distribution of data within the cluster. Understanding how PGs work is essential for understanding how Ceph achieves redundancy and efficiency in data management [14][33][34].

3.5.1 Pool

In a Ceph cluster, pools are **logical partitions** that contain objects. They serve as an interface for clients to store and retrieve data. Ceph supports two types of pools: "**replicated**" and "**erasure coded**". Replicated pools provide redundancy by storing multiple copies of an object on different OSDs. Erasure coded pools use parity to protect data, thereby providing a redundancy mechanism that is more storage-efficient than replicated pools [14][33].

The functioning of pools is closely linked to the cluster's ability to manage and distribute data. The data are distributed to the OSDs using PGs and the CRUSH algorithm [14].

Ceph pools are configurable, allowing administrators to adjust settings such as the number of object replicas, CRUSH rules and performance-related parameters. They allow data to be grouped logically, which makes the data easier to manage and access. Different pools can be created for different types of data, such as images, videos or backups, each with its own settings [14][33].

However, it is important to understand the following concept. Once a pool has been created, Ceph automatically manages the distribution of data across the available OSDs based on the configuration and the CRUSH algorithm. Pools are therefore configured to govern replication and data placement strategies rather than to encompass specific nodes. By default, all OSDs participate in storing data for all pools. OSD daemons are not specifically assigned to a particular pool. Instead, the CRUSH algorithm determines how the data are distributed across the OSDs based on the pool configuration and the cluster topology [33].

In a Ceph cluster, the decision on where an object will be stored – specifically, in which pool – is made at the time the object is written to the cluster. This decision is based on the application or user that specifies the target pool for the data [14].

3.5.2 PG

Placement Groups (PGs) are logical partitions within Ceph pools. They group objects together, thereby making access to objects more efficient. PGs reduce the overhead associated with tracking the location of objects and metadata, which becomes increasingly demanding as the number of objects grows into the millions. Each PG is then assigned to individual OSD daemons within the pool, with one serving as the primary (to coordinate write or read operations for the client) and the others as secondaries for data redundancy [14][34].

The process begins with the Ceph client identifying the PG to which the object belongs, usually based on its name or another unique identifier. This identification is performed using a hash function. The result of the hash function is a number, and this number is used to select one of the available PGs. This process is designed to replace the conventional way of storing metadata, e.g. in a tree structure. If we consider a situation where we want to find one specific object among a million objects without a previous entry in the cache, determining the exact location of the object is extremely time-consuming, which reduces cluster performance. Using PGs minimizes the time needed to locate an object, since, on the basis of the calculated hash, the CRUSH algorithm can almost instantly determine the OSD and PG in which the object is stored. Based on these data, the CRUSH algorithm directs the client to the specific OSD to which it writes the data or from which it reads the data [35].

Ceph minimizes computational and memory overhead by grouping objects into PGs. These enable efficient distribution and replication of data across a potentially large number of OSD daemons. Data resilience and availability are ensured by replicating data across multiple OSDs within PGs, even in the event of an OSD failure. Ceph has a feature called PG autoscaling, which can adjust the number of PGs in a pool based on storage utilization and defined rules. This helps maintain performance and resource utilization without manual intervention. Autoscaling takes into account the total storage capacity, the distribution of data across different storage devices and the expected data size in order to adjust the number of PGs as needed [14][36].

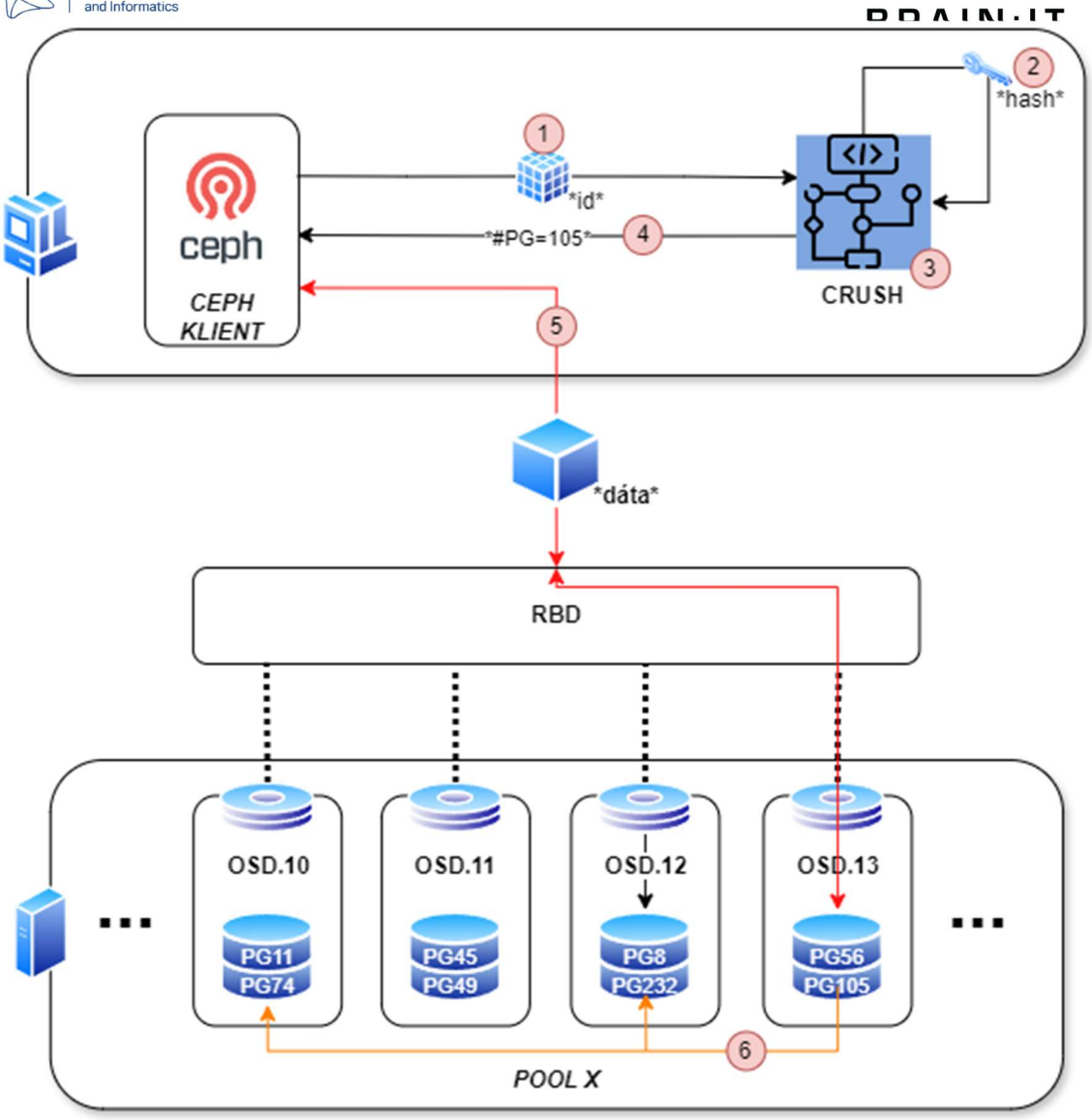


Figure 8 Principle of writing/reading data using the client

To illustrate how the process of writing or reading data works, we use Figure 8. The process consists of six steps. In the first step, the client sends an identifier to the CRUSH algorithm. In the second step, the algorithm calculates a hash from the identifier. In the third step, the PG number is calculated from the computed hash according to a predetermined algorithm. The fourth step is sending the PG number to the client. Since the client has the entire current CRUSH map, in the fifth step, based on the PG number, it establishes a connection directly with the relevant OSD of the selected pool on which the PG is located. Depending on the operation, the data are either written or read. The last, sixth step, in the case of a data write, is replication to the secondary OSDs, or more precisely to their respective PGs.

4. DEPLOYMENT OF A CEPH CLUSTER IN A TEST ENVIRONMENT

There are several recommended methods for installing the Ceph software. **Cephadm** and **Rook** are tools designed primarily for container-based deployment. Cephadm works directly with containers, and Rook is designed for deployment on Kubernetes clusters. Other methods include **ceph-ansible** and **ceph-salt**. For the OpenStack platform, installation is possible using the OpenStack Charms tool – **Juju** – or as a **Puppet** module. The specific method depends on the environment and preferences. Each method is designed with an emphasis on ease of use, thereby ensuring compatibility with various infrastructures [37].

Manual installation is also an option. It provides the highest level of control and configuration options during installation. For this reason, we decided to use manual installation for the deployment in the test environment, which is described in this chapter. By using manual installation, we can better understand how the Ceph software works and how it is configured [37].

4.1 Test environment

To deploy the Ceph software, we use a virtual test environment with virtual servers running the **Ubuntu 22.04** operating system. Based on the hardware recommendations [38], each server has a sufficient amount of computing resources.

The topology of the virtual test environment can be seen in Figure 9 below.

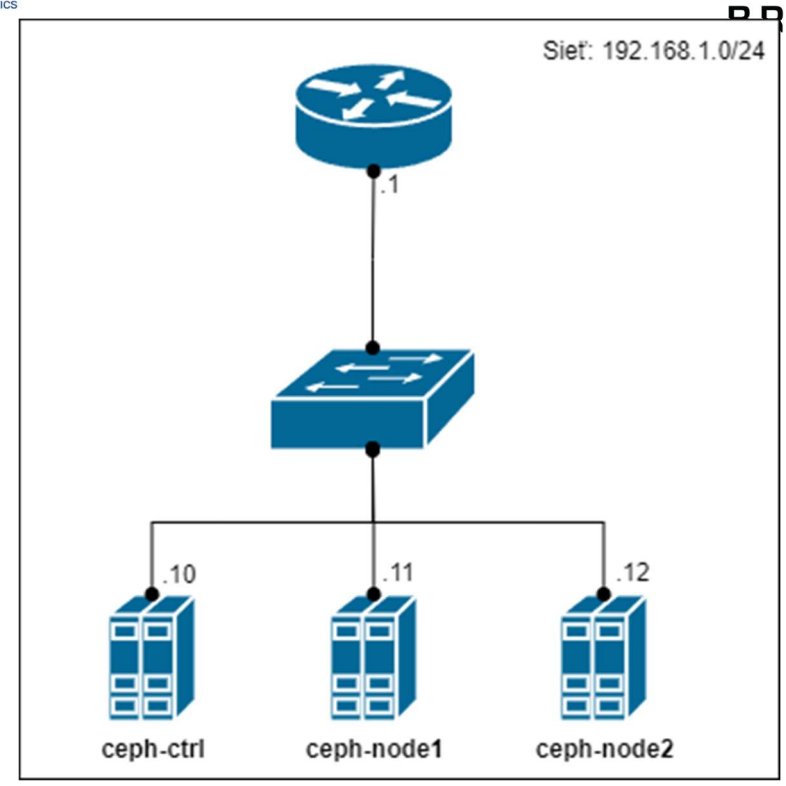


Figure 9 Topology of the virtual test environment

The environment consists of three servers. One server – "ceph-ctrl" – will serve as the monitor and management node. The other two servers are designated as OSD nodes, each of which additionally has one 32 GB disk.

4.2 Preparation of the servers

Before the deployment itself begins, the Ceph software needs to be installed on each server that is part of the cluster. The installation can be divided into two steps:

1. Adding keys and repositories

```
wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-key add –
apt-add-repository 'deb https://download.ceph.com/{release}/ {codename} main'
```

The variables `release` and `codename` must be replaced with specific values. The `release` variable denotes the version of the Ceph software being installed, and the `codename` variable denotes the code name of the distribution on which we are adding the repository. In this case, the `release` variable is replaced with the version "*reef*" and `codename` is replaced with the name "*jammy*" [39].

2. Installing the required packages

```
apt-get update && apt-get install ceph ceph-mds
```

We then proceed to update the repositories and install the ceph and ceph-mds packages [40].

4.3 Deployment of the monitor node

The deployment of the Ceph software itself starts with the deployment of the monitor node. At least one monitor must be deployed, and a sufficient number of disks must be provided for the OSD daemons to store copies of objects. The initial set-up of the monitor is important because it establishes the initial key parameters, such as the replication factor, the number of PGs per OSD, intervals and authentication requirements [41].

To deploy the monitor, a unique cluster ID – the so-called "FSID" – must be configured on the specific server. We create a configuration file in which we set the FSID, the monitor name, network and other settings. The name of the configuration file determines the name of the cluster. This is followed by creating a monitor key for secured communication and an administrative key for console access (CLI). The next step is generating the monitor map, creating the required directories and files, and setting their permissions. The penultimate step is initializing the monitor daemon with the given configuration and the keys created. Finally, we start the monitor services [41].

The process of deploying the monitor with the specific commands is described in the following points:

- Verifying that the /etc/ceph directory exists [41].

```
ls /etc/ceph
```

- Creating the FSID [41].

```
uuidgen
```

We copy the generated UUID and then use it as the FSID in the configuration file.

- Creating the configuration file and setting the required variables [41].

```
sudo nvim /etc/ceph/ceph.conf
```

Inside the configuration file, we set the FSID in the `[global]` context, as well as the values of other variables.

```
[global]
fsid = {uuid}
mon_initial_members = cephmon2
mon_host = 192.168.1.10
public_network = 192.168.1.0/24
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_min_size = 2
osd_pool_default_pg_num = 333
osd_crush_chooseleaf_type = 1
```

The `mon_inintial_members` variable specifies the domain names of the monitor nodes. In this case we are deploying only one monitor node, but if there are several, the domain names must be separated by commas. The same principle applies to the `mon_host` variable. The `public_network` variable specifies the network over which the individual cluster servers will communicate with clients and, unless a dedicated separate network is specified, also the network for communication among the cluster servers themselves, for both data and administrative traffic. Redundancy, i.e. the number of copies of objects in the cluster, is determined by the `osd_pool_default_size` variable. The `osd_pool_default_min_size` variable sets the minimum number of replicas that must be available for the cluster to allow an object read or write operation. The `osd_crush_chooseleaf_type` variable sets the type of replica distribution using the CRUSH algorithm. In this case we set it to "1", which corresponds to redistributing replicas to specific physical servers.

- Creating a private key for the cluster and an administrative key [41].

```
ceph-authtool --create-keyring /tmp/ceph.mon.keyring --gen-key -n mon. --cap mon 'allow *'
ceph-authtool --create-keyring /etc/ceph/ceph.client.admin.keyring --gen-key -n client.admin --cap mon 'allow *' --cap osd 'allow *' --cap mds 'allow *' --cap mgr 'allow *'
```

- Setting permissions and ownership for the administrative key.

```
chmod 644 /etc/ceph/ceph.client.admin.keyring
chown ceph:ceph /etc/ceph/ceph.client.admin.keyring
```

This step is not part of the official documentation, but without it neither the daemon nor the monitor service can be started.

- Generating the OSD key, creating the **client.bootstrap-osd** user and assigning the key to the user [41].

```
ceph-authtool --create-keyring /var/lib/ceph/bootstrap-osd/ceph.keyring --gen-key -n client.boot-strap-osd --cap mon 'profile bootstrap-osd' --cap mgr 'allow r'
```

- Adding the administrative and OSD keys to the cluster keys [41].

```
ceph-authtool /tmp/ceph.mon.keyring --import-keyring /etc/ceph/ceph.client.admin.keyring
ceph-authtool /tmp/ceph.mon.keyring --import-keyring /var/lib/ceph/bootstrap-osd/ceph.keyring
```

- Setting permissions and ownership for the cluster key [41].

```
chown ceph:ceph /tmp/ceph.mon.keyring
```

- Creating the monitor map [41].

```
monmaptool --create --add {hostnames} {addresses} --fsid {fsid} /tmp/monmap
```

In this command, the same rules apply to the **hostnames** and **addresses** variables as in the case of the configuration file. In this deployment we use only one monitor server, so we use only one domain name and one IP address. In the case of several monitor nodes, the domain names and IP addresses would be separated by commas.

- Creating the default data directory [41].

```
sudo -u ceph mkdir /var/lib/ceph/mon/{monitor-hostname}
```

This command must be applied on each monitor node separately. The **monitor-hostname** variable must be replaced with the corresponding domain name of the given server. In our specific case we are creating only one monitor node, so we apply it on the same machine as before and replace the **monitor-hostname** variable with the domain name of the server – "*cephmon2*". In this case it is important to use the **sudo** command, since the directory must be created as the "**ceph**" user.

- Adding the monitor map and keys to the daemon [41].

```
sudo -u ceph ceph-mon [--cluster {cluster-name}] --mkfs -i {hostname} --monmap /tmp/monmap --keyring /tmp/ceph.mon.keyring
```

As in the previous cases, the **hostname** variable is replaced with the domain name of the server(s), and the command must be executed as the "*ceph*" user. The **--cluster** option needs to be used only if the cluster name is not "*ceph*", which is the default value. In our case it is, so this option was not used.

- Starting the monitor service [41].

```
systemctl start ceph-mon@{hostname}
```

As in the case of creating the default directories, the command to start the service must also be run on all servers.

If a firewall is running on the servers, it is necessary to make sure that the settings allow communication of the monitor(s). This process creates the foundation of the Ceph cluster, which allows further extension with OSDs and other components. Subsequently, using the `ceph -s` command, we can verify that the monitor node has been correctly started and deployed [41].

```
kysel@cephmon2:~$ sudo ceph -s
  cluster:
    id:     1f624cfb-1bb8-4b53-a9e9-4c18ed4b3938
    health: HEALTH_WARN
            mon is allowing insecure global_id reclaim
            1 monitors have not enabled msgr2

  services:
    mon: 1 daemons, quorum cephmon (age 17h)
    mgr: no daemons active
    osd: 0 osds: 0 up, 0 in

  data:
    pools:   0 pools, 0 pgs
    objects: 0 objects, 0 B
    usage:   0 B used, 0 B / 0 B avail
    pgs:

kysel@cephmon2:~$_
```

Figure 10 Output of the ceph -s command after deployment of the cluster

4.4 Deployment of the manager

When deploying and managing a Ceph cluster, the **ceph-mgr** management module needs to be configured. This is achieved by configuring the **Ceph Dashboard** module, which is a built-in web dashboard for monitoring and management [42].

Although it is not mandatory for the dashboard to be running in a Ceph cluster, it is recommended to have at least one ceph-mgr instance with the dashboard module enabled for effective management and monitoring of the cluster. The dashboard offers an easy-to-use graphical interface for monitoring the status and performance of the cluster, as well as for managing the functions of the Ceph software. However, it is important to note that a Ceph cluster can also operate

without the dashboard and instead rely on CLI tools for management and monitoring [42].

When deploying the ceph-mgr daemon manually, it is necessary to create an authentication key, place it in the designated directory and then start the ceph-mgr daemon. The management daemon is deployed on the same server as the previous deployment of the monitor node [42].

To create the authentication key, we use the command

```
ceph auth get-or-create mgr.$name mon 'allow profile mgr' osd 'allow *' mds \ 'allow *'
```

we create a file named "*keyring*", insert into it the key generated by the previous command, move the "*keyring*" file to the corresponding directory using the command `mv keyring /var/lib/ceph/mgr/ceph-{hostname}/keyring` and start the ceph-mgr service using the command `ceph-mgr -i {hostname}` , where in both cases we replace `hostname` with the domain name of the server. The command `ceph mgr module enable dashboard` starts the dashboard module [42].

This is the recommended process for deploying the manager according to the official documentation at the time of writing this thesis. However, after executing these commands, the manager does not start and the dashboard does not work.

The dashboard module works only over the HTTPS protocol and therefore requires two additional things – a certificate for the HTTPS protocol and login credentials for the specific module. Therefore, in the deployment process, the following commands must be executed before starting the daemon.

```
ceph dashboard create-self-signed-cert
```

This command generates a self-signed SSL certificate for the dashboard, thereby ensuring encrypted communication between the dashboard and its users. It is also possible to use a custom certificate [43].

```
ceph mgr module enable dashboard
```

The dashboard module is not enabled by default. This command activates the module within the manager, thereby making the dashboard accessible. In the deployment process according to the official documentation, this was the last step. In this case, the module must be enabled before the login credentials are set [43].

```
ceph dashboard set-login-credentials <username> <password>
```

This command sets the username and password for access to the dashboard. We recommend writing the password to a file and substituting the path to this file for the `password` variable. In some cases, the password is set incorrectly because of the formatting of the command [43].

After entering these commands, the ceph-mgr daemon can be started. To verify that it works, we use the `ceph status` or `ceph -s` command. These commands are equivalent. To obtain the dashboard address, we use the `ceph mgr services` command, where we find the IP address and port for the "dashboard" module. Unless specified otherwise, the default port is 8443 [42][43].

```
kysel@cephmon2:~$ ceph status
  cluster:
    id:     0822a806-92a8-4ee0-a028-c802ebcc3d37
    health: HELATH_WARN
            mon is allowing insecure global_id reclaim
            1 monitors have not enabled msgr2
            1 daemons have recently crashed
            OSD count 0 < osd_pool_default_size 3

  services:
    mon: 1 daemons, quorum cephmon2 (age 19m)
    mgr: cephmon2(active, since 7m)
    osd: 0 osds: 0 up, 0 in

  data:
    pools:   0 pools, 0 pgs
    objects: 0 objects, 0 B
    usage:   0 B used, 0 B / 0 B avail
    pgs:

kysel@cephmon2:~$
```

Figure 11 Output of the ceph status command after deployment of the monitor daemon

As in the previous Figure 10, in Figure 11 we can see that the cluster contains one monitor. In addition, we can see that one manager node, or service, has indeed been started and added.

```
kysel@cephmon2:~$ ceph mgr services
{
    "dashboard": "https://192.168.1.13:8443/"
}
kysel@cephmon2:~$
```

Figure 12 Output of the ceph mgr services command

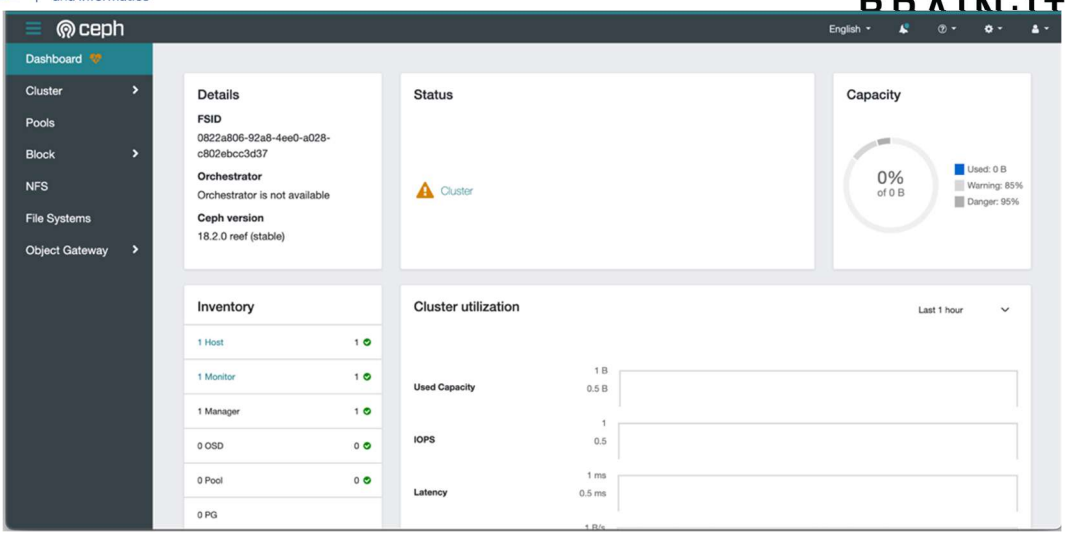


Figure 13 Ceph dashboard after login

4.5 Deployment of OSD nodes

The process of adding OSDs involves copying the `/var/lib/ceph/bootstrap-osd/ceph.keyring` file from the monitor node to the OSD node server. We then use the **ceph-volume** tool, which can prepare logical or physical partitions for storage. It is important to note that a file system must already have been created on the respective disks. On both nodes, we therefore divide the 32GB disk into four 8GB partitions in order to simulate multiple disks, and create an ext4 file system on them [41].

Once the file system has been created, OSDs can be created in two ways, with or without preparation. Since these procedures are equivalent, we use the simpler procedure without preparation. The deployment is carried out using a single command:

```
ceph-volume lvm create --data /dev/{#disk}
```

The `#disk` variable is replaced with the disk designation and partition number of the disks/partitions we want to add. The command must be entered for each disk/partition separately and on each OSD node. In our case, on both servers these were the disks designated "sdb" with partition numbers 1 to 4 (sdb1, sdb2,..) [41].

Verification can be performed using the `ceph status` or `ceph -s` command, or in the dashboard in the **Cluster -> OSDs** menu.

```
kysel@cephmon2:~$ ceph status
  cluster:
    id:     0822a806-92a8-4ee0-a028-c802ebcc3d37
    health: HEALTH_ERR
            mon is allowing insecure global_id reclaim
            Module 'devicehealth' has failed: table Device already exists

  services:
    mon: 1 daemons, quorum cephmon2 (age 12d)
    mgr: cephmon2(active, since 11d)
    osd: 8 osds: 8 up (since 12d), 8 in (since 12d); 1 remapped pgs

  data:
    pools:   2 pools, 33 pgs
    objects: 1 objects, 320 KiB
    usage:   2.7 GiB used, 61 GiB / 64 GiB avail
    pgs:     1/3 objects misplaced (33.333%)
             32 active+clean
             1  active+clean+remapped

kysel@cephmon2:~$
```

Figure 14 Output of ceph status after adding the OSD node servers

In the "**services:**" section, in the "**osd:**" part of Figure 14, we can see that there are currently eight OSDs in the cluster, and in the "**data:**" section, in the "**usage:**" and "**pgs:**" parts, that the cluster is functional, since 6GB of the available 64GB of space is used and all PGs are in the *"active+clean"* or "*active+clean+remapped"* state.

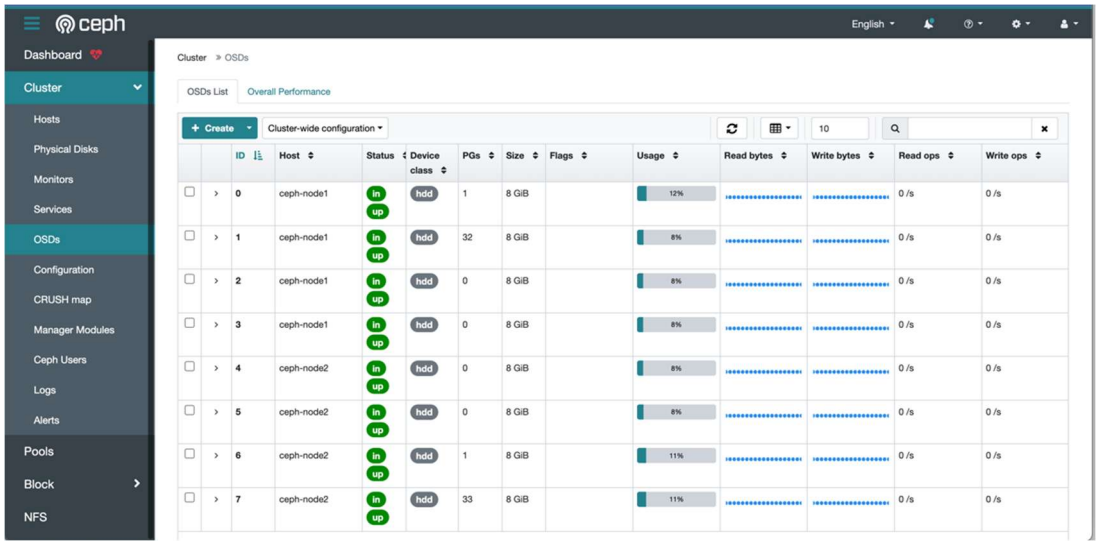


Figure 15 The Cluster -> OSDs menu in the dashboard

Figure 15 provides a more detailed view of the OSDs in the cluster. The "Host" column provides information on which machine the individual OSDs are located. In addition, the dashboard also displays the capacity, utilization, current status and number of PGs.

5. DEPLOYMENT AND INTEGRATION OF A CEPH CLUSTER USING OPENSTACK CHARMS TOOLS

OpenStack Charms are specialized so-called "charmed" operators (generally referred to as **charms**), designed for the efficient deployment and management of OpenStack within the Juju ecosystem. **Juju** is an open-source tool that simplifies the deployment, configuration and management of cloud services through an intuitive interface. Individual charms are encapsulated scripts used to automate the deployment and management of a specific OpenStack service/module, together with the underlying services that are not part of OpenStack (RabbitMQ, MySQL, etc.) [44][45].

The integration of **MAAS** (Metal as a Service) with OpenStack charms makes it possible to provision and manage the physical servers used in the cloud infrastructure. This seamless interaction between the physical provisioning layer provided by MAAS and the application management capabilities of Juju offers a comprehensive solution for deploying flexible cloud solutions on top of the OpenStack platform. This approach reduces deployment time and complexity and makes it possible to create a scalable cloud environment that can be adapted to various needs without increasing operational overhead [44][45].

5.1 Deployment concept

Unlike the previous chapter, where Ceph was installed as a standalone cluster, in this case it is part of the OpenStack cloud platform. Since the current cloud is deployed using OpenStack Charms tools, the deployment and integration of Ceph will likewise be handled using the MAAS and Juju tools mentioned above. To deploy OpenStack and Ceph, we use a so-called "**bundle**", which is a **YAML** configuration file that consolidates the individual OpenStack deployment steps into one compact file in which all the necessary parameters and settings are defined.

5.1.1 Topology

The deployment itself will be carried out in a virtual environment. A virtual environment provides flexibility and makes it easy to implement changes in the topology without the need for physical access to the devices. Figure 16 below shows the topology we will subsequently work with.

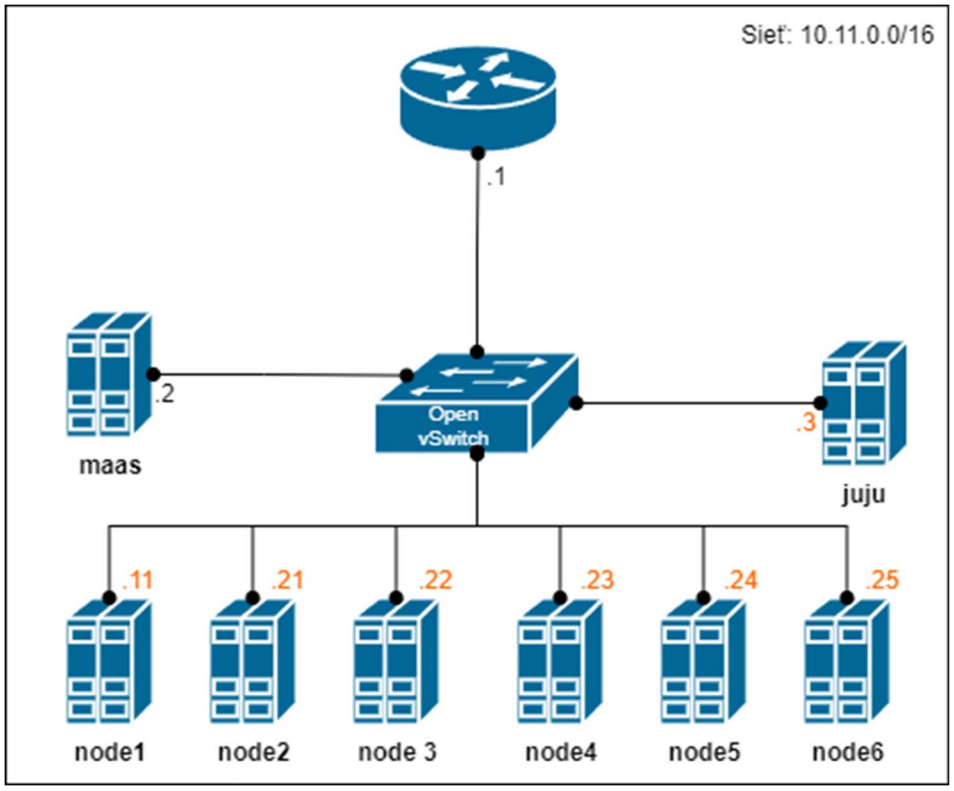


Figure 16 Topology for deploying OpenStack and Ceph using charms

In the topology shown, we can see that a total of 8 machines will be used for the deployment, representing the planned extension of the current cloud. All machines are located in the same subnet 10.11.0.0, and the virtual servers have static IPv4 addresses from this subnet. The last octets shown in orange indicate that, even though these IP addresses are static, MAAS is responsible for assigning these IPv4 addresses. Since VLANs are used for the OpenStack deployment, so-called "bridge" interfaces of the Open vSwitch (OVS) type must be configured on the MAAS servers. Details are described later in subsection 5.2 .

5.1.2 Virtual servers

All machines used for the deployment are virtual servers. Logically, the servers can be divided into management and compute servers. Management servers require fewer computing resources than compute servers. In our case, the management servers are "*maas"* and "*juju"*. The compute servers are the machines "*node1*" to "*node6*".

Machine	CPU	RAM	Disks
maas	4	4 GB	60 GB
juju	4	4 GB	60 GB
node1	6	16 GB	2x 100 GB
node2	6	16 GB	60 GB, 100 GB
node3	6	16 GB	60 GB, 100 GB
node4	6	16 GB	60 GB, 100 GB
node5	6	16 GB	60 GB, 100 GB
node6	6	16 GB	60 GB, 100 GB

Table 2 Hardware resources allocated to the virtual servers

A more detailed allocation of resources is shown in Table 2 above. When allocating resources to the individual servers, it was necessary to take into account the minimum requirements stated for Juju and the compute nodes in the "Getting started" section of the charm guide website [44], and for MAAS on its official website [46]. For the MAAS server, given the size of the managed infrastructure, 4 GB were considered sufficient despite the recommended minimum of 4.5 GB of RAM. In the case of the compute nodes, in addition to the 60GB disk, which will be used for deploying the operating system and the individual OpenStack components, the servers additionally have one disk with a size of 100 GB. These disks will be used as disks for the Ceph OSD daemons. The exception is "node1", where the system disk has a size of 100GB, since a larger number of charms will be run on it.

5.2 Preparation of the MAAS and Juju servers

The deployment process begins with the preparation of the MAAS and Juju servers. The first step is installing the Ubuntu 22.04 operating system on the "maas" server. For the deployment of OpenStack, including the preparation of the MAAS and Juju servers, we follow the guide in the OpenStack Charms documentation exactly, with minor modifications. As the bundle file, we use a modified bundle according to which the current production cloud is deployed.

The first change is the allocation of external networks. In the original preparation, the "EXT1" and "EXT2" subnets are assigned a range from the public range. In this case, these networks have no use and are not needed. However, to keep the bundle simple and consistent, these networks must be retained, and therefore they were changed from a public range to a private one (Table 3). We omit the steps for creating the OpenStack networks for communication with external networks.

Subnet	Original address range	Assigned address range
EXT1	10.255.153.0/24	10.193.153.0/24
EXT2	10.255.154.0/24	10.193.154.0/24

Table 3 Change of the address range for the EXT1 and EXT2 subnets

Another modification of the procedure relates to the server interfaces. Originally, the process involves creating so-called "bond" interfaces on the compute nodes. However, since all the virtual servers used have only one network interface, this step is unnecessary. Therefore, we merely replace the original interface name with the name "*bond*".

To distinguish between individual Juju servers, we also modify the name of the Juju controller. In our case, the controller is named "kysel-dp-juju-controller".

The last change is omitting the installation of the Juju dashboard. This component is not needed for the purposes of this thesis, and this step was therefore omitted.

The versions of the installed MAAS and Juju software also differ. In our case, the installed version of MAAS is 3.4.1 and Juju is version 3.4.2 .

5.3 Ceph components in the bundle file

The deployment of the Ceph cluster is part of the OpenStack installation, the aim of which is to deploy a basic Ceph cluster integrated into the OpenStack platform, in the form of providing "backend" functions for the individual storage modules. In this case, Ceph will provide all types of storage – object, block and file – for the **Glance**, **Cinder** and **Manila** modules.

Juju offers a total of 10 charms for Ceph within the Openstack Charms project. We use 7 of them to deploy the cluster. The unused charms are **Ceph Proxy**, **Ceph Rbd Mirror** and **Ceph Iscsi**. Ceph Proxy is used to connect an external Ceph cluster to OpenStack. Since our goal is a cluster integrated directly on the compute nodes, this charm is not used. Likewise unused are Ceph Rbd Mirror, because there is no other Ceph cluster to which data would be mirrored, and Ceph Iscsi – a charm that allows RADOS block device (RBD) images to be exported as iSCSI disks over the network, and which, moreover, is no longer being developed [48][49].

In the following subsections, we focus on the parameters that must or can be defined within the bundle file for deploying a Ceph cluster with the required functions (the "options:" and "bindings:" sections). The charms themselves are defined in the "applications:" section. Mandatory parameters such as "charm:" and "channel:" will not be described, since these parameters are mandatory for every application and can be found on the charmhub.io website [48]. In our case, the channel will be "reef/stable" for all Ceph applications. For the "source:" parameter, we use "cloud:focal-yoga". To make configuration easier, we use the variables feature that the bundle offers [47].

```
variables:
  openstack-origin: &openstack-origin cloud:focal-yoga
  data-port: &data-port brEX:bond.153 brEX2:bond.154 brVXLAN:bond.500
  osd-devices: &osd-devices /dev/sdb
  expected-osd-count: &expected-osd-count 5
  expected-mon-count: &expected-mon-count 3
```

We set the `expected-osd-count` and `expected-mon-count` variables to 5 and 3, which means that we plan to deploy at least 5 OSD daemons and 3 monitors.

In the listed options in the form <option,..>, the option marked "*" denotes the default value.

5.3.1 Ceph Mon

As in the case of the manual installation of the Ceph cluster, it is necessary to start with the monitor. Unlike the manual installation, however, there will no longer be just one monitor but three, so that in the event of a failure of one monitor, access to the data will not be lost. In this case, we will not set any parameters other than the expected number of OSD daemons and the number of monitors.

However, the Ceph Mon charm does not deploy only the monitor itself. When each monitor instance is deployed, one ceph-mgr instance is also created inside the instance. The number of deployed managers is therefore affected by the number of monitors in a cluster deployed in this way.

5.3.1.1 Applied configuration

```
ceph-mon:
  bindings:
    "": openstack-mgmt
    public: openstack-mgmt
    cluster: openstack-mgmt
  charm: ch:ceph-mon
  channel: reef/stable
  num_units: 3
  options:
    expected-osd-count: *expected-osd-count
    monitor-count: *expected-mon-count
    source: *openstack-origin
  to:
  - lxd:21
  - lxd:22
  - lxd:23
```

In the "bindings:" section, we define the so-called "space" in which the application will communicate. In our case, all communication takes place in the "openstack-mgmt" space. We have also set the same space for the "public:" and "cluster:" parameters. The " **"":** " parameter represents the default network. These parameters work in the same way as in the case of the manual installation. This means that they allow us to set the network over which the cluster will communicate with clients and, where applicable, a separate network intended for administrative communication. For the parameters in the "options:" section, we set the corresponding variables.

5.3.1.2 Other options [48]

In addition to the options used in the bundle, it is also possible to set other parameters that we did not use but which may be useful in a real deployment.

- **monitor-data-available-critical:** (int) – the monitor storage availability value in % that signals the HEALTH_ERR cluster status <*5>
- **monitor-data-available-warning:** (int) – the monitor storage availability value in % that signals the HEALTH_WARN cluster status <*30>
- **pg-autotune:** (string) – automatic tuning of PGs <*auto, true, false>
- **pgs-per-osd:** (int) – manual setting of the number of PGs according to PGcalc [50]
- **prefer-ipv6:** (bool) – enabling IPv6 support
- **use-syslog:** (bool) – logging to the system syslog

All available options can be found on the charmhub.io website in the Ceph Mon section [48].

5.3.2 Ceph Osd

Deploying the OSD daemons is the next step, which is essential for deploying a Ceph cluster. As already mentioned above, we currently use 5 servers for the deployment, specifically the 100GB disks **/dev/sdb,** which we defined in the variables section as the **osd-devices** variable. It is important to note that we do not deploy OSD daemons on the disk of the controller node (node1), in order to demonstrate later how OSDs are added to an already created cluster using charms.

5.3.2.1 Applied configuration

```
ceph-osd:
  charm: ch:ceph-osd
  channel: reef/stable
  num_units: 5
  options:
    osd-devices: *osd-devices
    source: *openstack-origin
  to:
  - '21'
  - '22'
  - '23'
```

```
- '24'
- '25'
```

We use the basic configuration. We specify the names of the disks to be used for deploying the OSD daemons (the **osd-devices** variable) and deploy them on the servers labelled 21 – 25.

5.3.2.2 Other options [48]

In the case of the charm for the Ceph OSD daemons, there are also a few other options that may be suitable to set.

- **bdev-enable-discard:** (string) – enabling the "TRIM" function for SSD disks <*auto, enable, disable>
- **ignore-device-errors:** (bool) – if set to "true" and the charm is unable to initialize a disk, it reports only a warning instead of an error
- **max-sectors-kb:** (int) – makes it possible to use larger I/O sizes (e.g. when using RAID cards) <*1048576>
- **osd-encrypt:** (bool) – setting encryption of the OSD disks
- **osd-encrypt-keymanager:** (string) – setting the manager responsible for storing the encryption keys <vault, *ceph>
- **osd-journal:** (string) – setting the device to be used as a shared journaling disk for all OSDs in the node <*(each OSD disk has its own)>
- **prefer-ipv6:** (bool) - enabling IPv6 support
- **use-direct-io:** (bool) – setting the use of direct I/O access for the journaling OSD disks
- **use-syslog:** (bool) - logging to the system syslog

All configuration options are available on the charmhub.io website in the Ceph Osd section [48].

5.3.3 Ceph Fs

When deployed, the Ceph Fs charm not only creates a pool for the CephFS file system but also deploys MDS servers with each deployed instance. These then provide an interface for connecting to the pool as shared storage.

5.3.3.1 Applied configuration

```
ceph-fs:
   charm: ch:ceph-fs
   channel: reef/stable
   num_units: 5
   options:
     source: *openstack-origin
   bindings:
     "": openstack-mgmt
   to:
   - lxd:21
   - lxd:22
   - lxd:23
   - lxd:24
   - lxd:25
```

In this case, we again use only the basic configuration of the charm. As in the case of the OSD daemons, we deploy the MDS servers on machines 21 – 25. We do not deploy any MDS server on the "node1" server, for reasons of load distribution, since "node1" is responsible for more OpenStack control instances.

5.3.3.2 Other options [48]

Ceph Fs provides only a few settings useful for our purposes.

- **ceph-osd-replication-count:** (int) – setting the number of replicas of stored objects <*3>
- **mds-cache-memory-limit:** (string) - maximum size of the MDS server cache in bytes <*4Gi>
- **metadata-pool:** (string) - name of the metadata pool to be created/used
- **pool-type:** (string) - setting the pool type <*replicated,erasure-coded>
- **prefer-ipv6:** (bool) - enabling IPv6 support
- **rbd-pool-name:** (string) – name of the data pool to be created/used
- **use-syslog:** (bool) - logging to the system syslog

If we were to set the **pool type:** parameter to "erasure-coded", there are further configuration options specific to this pool type, which can be found in the Ceph Fs charm documentation on the charmhub.io website [48].

5.3.4 Ceph Radosgw

The Ceph Radosgw charm deploys and manages the RADOS Gateway (RGW) in a Ceph cluster. In the test environment, we did not deploy this component in order to simplify the deployment, but it is essential for integration with OpenStack.

5.3.4.1 Applied configuration

```
ceph-radosgw:
   bindings:
      "": openstack-mgmt
      public: openstack-mgmt
      internal: openstack-mgmt
      admin: openstack-mgmt
   charm: ch:ceph-radosgw
   channel: reef/stable
   num_units: 1
   options:
     source: *openstack-origin
   to:
   - lxd:11
```

The basic configuration is currently extended with the **admin:** network, which is used for managing the RGW.

5.3.4.2 Other options [48]

The charm offers several relevant configuration options directly in the bundle file.

- **cache-size:** (int) – number of keystone module keys in the local cache
- **ceph-osd-replication-count:** (int) - setting of the number of replicas of stored objects <*3>
- **pool-type:** (string) - setting the pool type <*replicated,erasure-coded>
- **port:** (int) - setting of the port on which RGW listens <*80, 443>
- **prefer-ipv6:** (bool) - enabling IPv6 support
- **ssl_ca:** (string) – setting of a custom certification authority (CA) when the HTTPS protocol is used for the API.
- **ssl_cert:** (string) – setting of the SSL certificate for the HTTPS API
- **ssl_key:** (string) – setting of the SSL certificate key for the HTTPS API

- **use-syslog:** (bool) - logging to the system syslog

Further configuration options can be found on the charmhub.io website [48].

5.3.5 Ceph Dashboard

As in the case of the manual installation, Ceph Dashbord does not require any specific configuration in order to function. However, it provides a few configuration options that could be used for deployment on the department cloud.

5.3.5.1 Applied configuration

```
ceph-dashboard:
  charm: ch:ceph-dashboard
  channel: reef/stable
```

The applied configuration of the dashboard charm is simple. It requires no configuration apart from the essential parameters – the name of the charm and of the channel. Since the Ceph dashboard is initialized on the server where ceph-mgr is running, it is not necessary to specify on which server the dashboard should be deployed, as the charm deploys it automatically – on the monitor instance.

5.3.5.2 Other options [48]

The options for setting up the ceph-dashboard module in the bundle file are not extensive. All of them concern either password security, or rather the requirements for creating a password for logging in to the dashboard, or the setting of private SSL (Secure Socket Layer) certificates for the API interfaces. For use in the cloud, the most useful are the policies for setting the login password.

- **password-policy-min-length:** (int) – setting of the minimum password length
- **password-policy-check-username:** (boolean) – does not allow the creation of a password that contains the user name

- **password-policy-check-length:** (bool) – rejects the creation of a password if its length is shorter than the set minimum

As in the previous cases, all options with details are listed on the charm-hub.io website for the Ceph Dashboard charm [48].

5.3.6 Cinder Ceph

We now move on to the charms that are not directly part of Ceph but are essential for integration with OpenStack. The Cinder Ceph charm integrates Ceph with the **Cinder** module, thereby enabling Cinder to use Ceph block devices (RBD).

5.3.6.1 Applied configuration

```
cinder-ceph:
  charm: ch:cinder-ceph
  channel: yoga/stable
  num_units: 0
```

For the deployment of this charm, we likewise use the basic configuration for deployment with default values. The reason is the subsequent addition of various pools as Cinder “backends” for OpenStack.

5.3.6.2 Other options [48]

When describing the OpenStack modules that provide options for integration with Ceph, we will focus only on the options that are relevant for setting the behaviour of the pools or of Ceph itself.

- **ceph-osd-replication-count:** (int) - setting of the number of replicas of stored objects <*3>
- **pool-type:** (string) - setting the pool type <*replicated,erasure-coded>
- **use-syslog:** (bool) - logging to the system syslog
- **volume-backend-name:** (string) – setting of the name for the backend pool

The Cinder Ceph charm offers more settings, mainly for the configuration of the BlueStore disk format, which OSD disks use for direct communication with physical disks, or settings of erasure-coded pools. These settings are not relevant for use in the cloud, as optimization at such a level is unnecessary. All detailed settings are again available on the charmhub.io website [48].

5.3.7 Glance

In order to use the Ceph cluster for storing virtual machine images in the OpenStack environment, it is necessary to use the charm of the **Glance** module. This module does not have a special charm that would integrate Glance and Ceph, despite the fact that Ceph is the recommended storage for the Glance module [48].

5.3.7.1 Applied configuration

```
glance:
  bindings:
    "": openstack-mgmt
    public: openstack-mgmt
    internal: openstack-mgmt
    admin: openstack-mgmt
    shared-db: openstack-mgmt
  charm: ch:glance
  channel: yoga/stable
  num_units: 1
  options:
    openstack-origin: *openstack-origin
  to:
  - lxd:11
```

The configuration used again contains only the basic parameters required for the given charm. The reason is also the fact that the most important aspect of integrating Ceph with the Glance module is the relation between these two components, which will be described in more detail in chapter 5.4.

5.3.7.2 Other options [48]

Glance naturally contains several configuration options for the operation of the module itself. However, the following options directly affect Ceph and its components.

- **ceph-osd-replication-count:** (int) - setting of the number of replicas of stored objects <*3>
- **pool-type:** (string) - setting the pool type <*replicated,erasure-coded>
- **prefer-ipv6:** (bool) - enabling IPv6 support
- **use-syslog:** (bool) - logging to the system syslog

As in the previous cases, several options are repeated, since a separate pool with its own settings is created for each module. Detailed configuration options of the Glance module are described on the charmhub.io website [48].

5.3.8 Manila Ganesha

The last charm for Ceph integration is Manila Ganesha. Similarly to the Cinder Ceph charm, this charm integrates CephFS with the Manila module by deploying NFS-Ganesha. NFS-Ganesha provides an interface for the NFS (Network File System) protocol, which is able to serve file systems from various storage systems. It acts as a bridge that provides shared storage to OpenStack instances via the NFS system. These instances use the CephFS system as their backend storage and thus integrate it seamlessly into the OpenStack cloud infrastructure [48][51].

This charm does not provide any options in the bundle file that would affect the Ceph cluster. For this reason, neither the applied configuration nor further options will be included.

5.4 Module relations

In Juju, relations are a mechanism that enables services to be interconnected and to share information about configuration, services and management. They can be defined in the bundle file and describe how the various applications should be interconnected. Relations are crucial because they determine how the various parts of the cloud environment communicate, share data and pass control instructions to one another.

All relations are defined in the bundle file in the “relations:” section. As in the previous cases when describing the configuration options for the individual charms in the bundle file, in the case of relations we will also deal only with the relations of those components that directly affect the Ceph cluster.

We begin with the relations related to authentication. For authentication, OpenStack uses the Keystone module as well as the Vault software.

```
relations:
- - ceph-radosgw:identity-service
  - keystone:identity-service
- - vault:certificates
  - ceph-radosgw:certificates
- - ceph-dashboard:certificates
  - vault:certificates
```

The relation between `ceph-radosgw:identity-service` and `keystone:identity-service` connects Ceph RGW with the Keystone module for authentication services. The relations between `vault:certificates` and `ceph-radosgw:certificates`, as well as `ceph-dashboard:certificates`, manage secure communication and the sharing of certificates between the Vault and Ceph components.

The following relations describe how ceph-mon cooperates with the other services and components in the cluster in order to ensure efficient operation and connectivity.

```
- - ceph-mon:client
  - nova-compute:ceph
- - ceph-mon:client
  - cinder-ceph:ceph
- - ceph-mon:client
  - glance:ceph
- - ceph-mon:client
  - manila-ganesha:ceph
- - ceph-osd:mon
  - ceph-mon:osd
- - ceph-radosgw:mon
  - ceph-mon:radosgw
- - ceph-mon:mds
  - ceph-fs:ceph-mds
- - ceph-dashboard:dashboard
```

```
- ceph-mon:dashboard
```

The relations involving `ceph-mon:client` enable the Nova, Cinder, Glance and Manila modules to communicate with the Ceph cluster. Communication between the OSD daemons and the monitors is set up by means of `ceph-osd:mon` and `ceph-mon:osd`. Ceph RGW is connected to the cluster monitors via `cephradosgw:mon` and `ceph-mon:radosgw`. The monitors are connected to the MDS and CephFS instances via the relation between `ceph-mon:mds` and `ceph-fs:ceph-mds`. Ceph Dashboard is connected to Ceph Mon by defining the relation `ceph-dashboard:dashboard` and `ceph-mon:dashboard`.

What remains last is to define the relations that ensure access and how storage is provided to the connected services that require it (virtual machines or containers).

```
- - cinder-ceph:storage-backend
  - cinder:storage-backend
- - nova-compute:ceph-access
  - cinder-ceph:ceph-access
```

Connecting `cinder-ceph:storage-backend` with `cinder:storage-backend` configures Ceph as the backend for Cinder. Although the Nova module was not mentioned in the configuration, it is important that the relation between `nova-compute:ceph-access` and `cinder-ceph:ceph-access` is defined. This is because it ensures that the Nova module has the necessary access to Ceph storage.

5.5 Deployment of OpenStack with a Ceph cluster

The advantage of deploying OpenStack by means of a bundle file is simplicity. We begin by creating a model with the `juju add-model` command. Using the `juju switch` and `juju spaces` commands, it is necessary to verify that we are controlling the correct module and that the individual spaces have the correct networks assigned. With the `juju deploy` command, we start the deployment of the cloud [47].

```
juju add-model <model name>
juju deploy <path to bundle file>
```

Subsequently, it is necessary to carry out the steps for starting the Vault charm, which are described in the OpenStack documentation. It should be noted, however, that the Juju commands have changed significantly, and therefore before each deployment it is necessary to read the documentation for the current version of Juju, including the commands in this thesis.

5.5.1 Deployment verification

After executing the commands for deploying the cloud and the Vault charm, it is necessary to wait until all charms and their respective containers and services have started. We will be interested specifically in the components of the Ceph cluster and the modules required for integration with OpenStack. Information on whether the services are active and running is obtained by entering the **juju status** command.

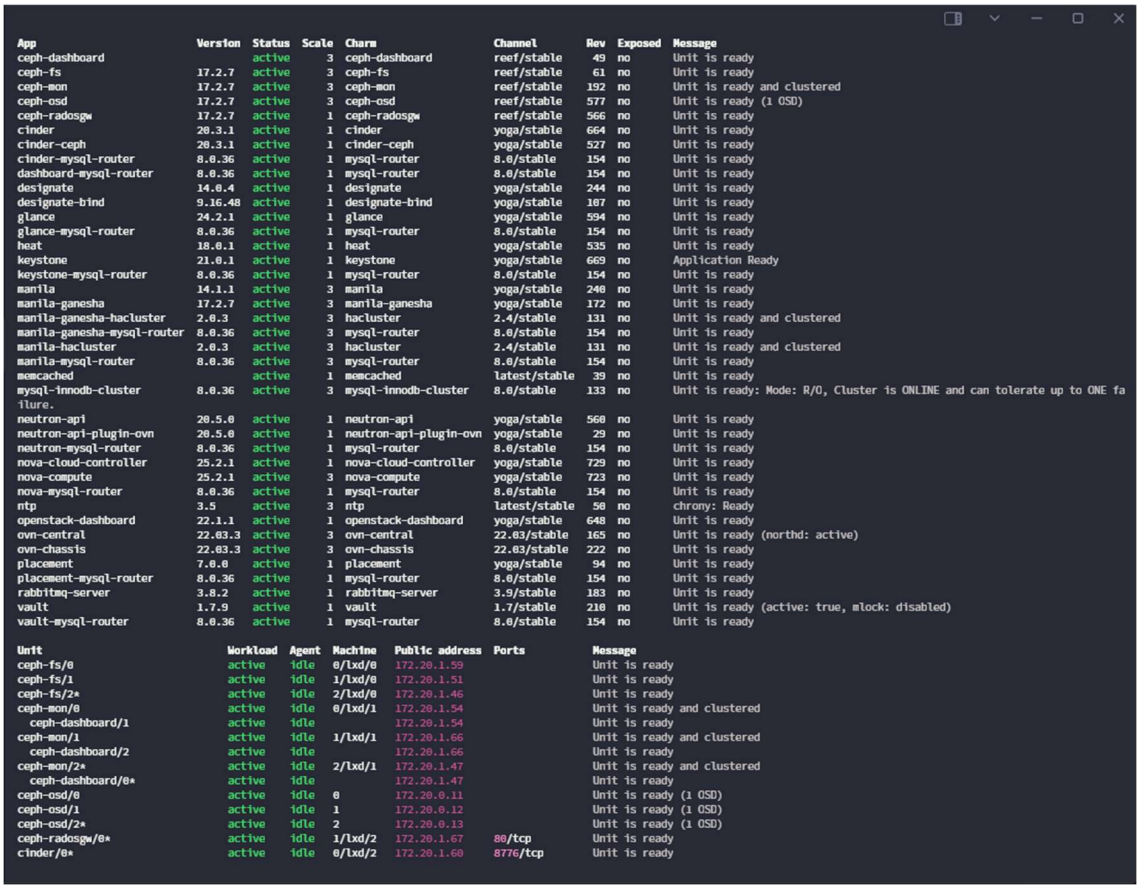
While working on the thesis, problems occurred with the virtual environment. During repeated deployment, modules became non-functional. In each new deployment, different modules failed for different reasons. Over time, some of the problems were resolved, but the deployment could not be used for testing if the “ovn-chassis” module was not functional.



Figure 17 Output of the juju status command - problems in the virtual environment

For this reason, we agreed with the thesis supervisor that another environment would also be used, this time with real devices and a modified topology and configuration. Should the problems not be resolved, testing would continue in the physical environment.

However, the problems were partially resolved, in particular the one with the "ovn-chassis" module. Figures 18 and 19 below show the outputs of the `juju status` command from both environments.



```
App                       Version  Status  Scale  Charm                     Channel        Rev  Exposed  Message
ceph-dashboard                     active      3  ceph-dashboard            reef/stable     49  no       Unit is ready
ceph-fs                   17.2.7   active      3  ceph-fs                   reef/stable     61  no       Unit is ready
ceph-mon                  17.2.7   active      3  ceph-mon                  reef/stable    192  no       Unit is ready and clustered
ceph-osd                  17.2.7   active      3  ceph-osd                  reef/stable    577  no       Unit is ready (1 OSD)
ceph-radosgw              17.2.7   active      1  ceph-radosgw              reef/stable    566  no       Unit is ready
cinder                    20.3.1   active      1  cinder                    yoga/stable    664  no       Unit is ready
cinder-ceph               20.3.1   active      1  cinder-ceph               yoga/stable    527  no       Unit is ready
cinder-mysql-router       8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
dashboard-mysql-router    8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
designate                 14.0.4   active      1  designate                 yoga/stable    244  no       Unit is ready
designate-bind            9.16.48  active      1  designate-bind            yoga/stable    187  no       Unit is ready
glance                    24.2.1   active      1  glance                    yoga/stable    594  no       Unit is ready
glance-mysql-router       8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
heat                      18.0.1   active      1  heat                      yoga/stable    535  no       Unit is ready
keystone                  21.0.1   active      1  keystone                  yoga/stable    669  no       Application Ready
keystone-mysql-router     8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
manila                    14.1.1   active      3  manila                    yoga/stable    240  no       Unit is ready
manila-ganesha            17.2.7   active      3  manila-ganesha            yoga/stable    172  no       Unit is ready
manila-ganesha-hacluster  2.0.3    active      3  hacluster                 2.4/stable     131  no       Unit is ready and clustered
manila-ganesha-mysql-router 8.0.36 active      3  mysql-router              8.0/stable     154  no       Unit is ready
manila-hacluster          2.0.3    active      3  hacluster                 2.4/stable     131  no       Unit is ready and clustered
manila-mysql-router       8.0.36   active      3  mysql-router              8.0/stable     154  no       Unit is ready
memcached                          active      1  memcached                 latest/stable   39  no       Unit is ready
mysql-innodb-cluster      8.0.36   active      3  mysql-innodb-cluster      8.0/stable     133  no       Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE fa
ilure.
neutron-api               20.5.0   active      1  neutron-api               yoga/stable    560  no       Unit is ready
neutron-api-plugin-ovn    20.5.0   active      1  neutron-api-plugin-ovn    yoga/stable     29  no       Unit is ready
neutron-mysql-router      8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
nova-cloud-controller     25.2.1   active      1  nova-cloud-controller     yoga/stable    729  no       Unit is ready
nova-compute              25.2.1   active      3  nova-compute              yoga/stable    723  no       Unit is ready
nova-mysql-router         8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
ntp                       3.5      active      3  ntp                       latest/stable   50  no       chrony: Ready
openstack-dashboard       22.1.1   active      1  openstack-dashboard       yoga/stable    648  no       Unit is ready
ovn-central               22.03.3  active      3  ovn-central               22.03/stable   165  no       Unit is ready (northd: active)
ovn-chassis               22.03.3  active      3  ovn-chassis               22.03/stable   222  no       Unit is ready
placement                 7.0.0    active      1  placement                 yoga/stable     94  no       Unit is ready
placement-mysql-router    8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready
rabbitmq-server           3.8.2    active      1  rabbitmq-server           3.9/stable     183  no       Unit is ready
vault                     1.7.9    active      1  vault                     1.7/stable     210  no       Unit is ready (active: true, mlock: disabled)
vault-mysql-router        8.0.36   active      1  mysql-router              8.0/stable     154  no       Unit is ready

Unit                      Workload  Agent  Machine  Public address  Ports      Message
ceph-fs/0                 active    idle   0/lxd/0  172.20.1.59                Unit is ready
ceph-fs/1                 active    idle   1/lxd/0  172.20.1.51                Unit is ready
ceph-fs/2*                active    idle   2/lxd/0  172.20.1.46                Unit is ready
ceph-mon/0                active    idle   0/lxd/1  172.20.1.54                Unit is ready and clustered
  ceph-dashboard/1        active    idle            172.20.1.54                Unit is ready
ceph-mon/1                active    idle   1/lxd/1  172.20.1.66                Unit is ready and clustered
  ceph-dashboard/2        active    idle            172.20.1.66                Unit is ready
ceph-mon/2*               active    idle   2/lxd/1  172.20.1.47                Unit is ready and clustered
  ceph-dashboard/0*       active    idle            172.20.1.47                Unit is ready
ceph-osd/0                active    idle   0        172.20.0.11                Unit is ready (1 OSD)
ceph-osd/1                active    idle   1        172.20.0.12                Unit is ready (1 OSD)
ceph-osd/2*               active    idle   2        172.20.0.13                Unit is ready (1 OSD)
ceph-radosgw/0*           active    idle   1/lxd/2  172.20.1.67  80/tcp        Unit is ready
cinder/0*                 active    idle   0/lxd/2  172.20.1.60  8776/tcp      Unit is ready
```

Figure 18 Output of the juju status command after deployment on physical devices

Figure 18 above shows all the charms deployed in the real environment. From the output we can see that all of them are in the "*active*" state. In the "*Channel*" column, we can see the versions of the deployed charms. Figure 19 shows the "Unit" section of the `juju status` output. It displays the individual containers on which the specific services of the given charms also run, together with the IP addresses and the identifiers of the machines on which the containers are located (the first number is the machine number – "*Machine*"). In the output, all components of the Ceph cluster are in the "active" state.

Unit	Workload	Agent	Machine	Public address	Ports	Message
ceph-fs/0	active	idle	1/lxd/0	10.11.1.141		Unit is ready
ceph-fs/1	active	idle	2/lxd/0	10.11.1.149		Unit is ready
ceph-fs/2*	active	idle	3/lxd/0	10.11.1.135		Unit is ready
ceph-fs/3	active	idle	4/lxd/0	10.11.1.145		Unit is ready
ceph-fs/4	active	idle	5/lxd/0	10.11.1.136		Unit is ready
ceph-mon/0	active	idle	1/lxd/1	10.11.1.142		Unit is ready and clustered
ceph-dashboard/1	active	idle		10.11.1.142		Unit is ready
ceph-mon/1	active	idle	2/lxd/1	10.11.1.150		Unit is ready and clustered
ceph-dashboard/2	active	idle		10.11.1.150		Unit is ready
ceph-mon/2*	active	idle	3/lxd/1	10.11.1.134		Unit is ready and clustered
ceph-dashboard/0*	active	idle		10.11.1.134		Unit is ready
ceph-osd/0*	active	idle	1	10.11.0.21		Unit is ready (1 OSD)
ceph-osd/1	active	idle	2	10.11.0.22		Unit is ready (1 OSD)
ceph-osd/2	active	idle	3	10.11.0.23		Unit is ready (1 OSD)
ceph-osd/3	active	idle	4	10.11.0.24		Unit is ready (1 OSD)
ceph-osd/4	active	idle	5	10.11.0.25		Unit is ready (1 OSD)
ceph-radosgw/0*	active	idle	0/lxd/0	10.11.1.114	80/tcp	Unit is ready
cinder/0*	active	idle	0/lxd/1	10.11.1.117	8776/tcp	Unit is ready
cinder-ceph/0*	active	idle		10.11.1.117		Unit is ready
cinder-mysql-router/0*	active	idle		10.11.1.117		Unit is ready
designate-bind/0*	active	idle	0/lxd/3	10.11.1.131		Unit is ready
designate/0*	error	idle	0/lxd/2	10.11.1.118	9001/tcp	hook failed: "identity-service-relation-changed"
glance/0*	active	idle	0/lxd/4	10.11.1.119	9292/tcp	Unit is ready
glance-mysql-router/0*	active	idle		10.11.1.119		Unit is ready
heat/0*	active	idle	0/lxd/5	10.11.1.122	8000,8004/tcp	Unit is ready
keystone/0*	active	idle	0/lxd/6	10.11.1.128	5000/tcp	Unit is ready
keystone-mysql-router/0*	active	idle		10.11.1.128		Unit is ready
manila-ganesha/0	active	idle	1/lxd/3	10.11.1.139		Unit is ready
manila-ganesha-hacluster/2	active	idle		10.11.1.139		Unit is ready and clustered
manila-ganesha-mysql-router/2	active	idle		10.11.1.139		Unit is ready
manila-ganesha/1	active	idle	2/lxd/3	10.11.1.147		Unit is ready
manila-ganesha-hacluster/4	active	idle		10.11.1.147		Unit is ready and clustered
manila-ganesha-mysql-router/4	active	idle		10.11.1.147		Unit is ready
manila-ganesha/2*	active	idle	3/lxd/3	10.11.1.132		Unit is ready
manila-ganesha-hacluster/0*	active	idle		10.11.1.132		Unit is ready and clustered
manila-ganesha-mysql-router/0*	active	idle		10.11.1.132		Unit is ready
manila-ganesha/3	active	idle	4/lxd/2	10.11.1.146		Unit is ready
manila-ganesha-hacluster/3	active	idle		10.11.1.146		Unit is ready and clustered
manila-ganesha-mysql-router/3	active	idle		10.11.1.146		Unit is ready
manila-ganesha/4	active	idle	5/lxd/2	10.11.1.137		Unit is ready
manila-ganesha-hacluster/1	active	idle		10.11.1.137		Unit is ready and clustered
manila-ganesha-mysql-router/1	active	idle		10.11.1.137		Unit is ready
manila/0	active	idle	1/lxd/2	10.11.1.140	8786/tcp	Unit is ready
manila-hacluster/2	active	idle		10.11.1.140		Unit is ready and clustered
manila-mysql-router/2	active	idle		10.11.1.140		Unit is ready
manila/1	active	idle	2/lxd/2	10.11.1.151	8786/tcp	Unit is ready
manila-hacluster/4	active	idle		10.11.1.151		Unit is ready and clustered
manila-mysql-router/4	active	idle		10.11.1.151		Unit is ready
manila/2*	active	idle	3/lxd/2	10.11.1.133	8786/tcp	Unit is ready
manila-hacluster/0*	active	idle		10.11.1.133		Unit is ready and clustered
manila-mysql-router/0*	active	idle		10.11.1.133		Unit is ready
manila/3	active	idle	4/lxd/1	10.11.1.143	8786/tcp	Unit is ready
manila-hacluster/3	active	idle		10.11.1.143		Unit is ready and clustered
manila-mysql-router/3	active	idle		10.11.1.143		Unit is ready
manila/4	active	idle	5/lxd/1	10.11.1.138	8786/tcp	Unit is ready
manila-hacluster/1	active	idle		10.11.1.138		Unit is ready and clustered
manila-mysql-router/1	active	idle		10.11.1.138		Unit is ready
memcached/0*	active	idle	0/lxd/7	10.11.1.127	11211/tcp	Unit is ready
mysql-innodb-cluster/0	active	idle	0/lxd/8	10.11.1.115		Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/1	active	idle	2/lxd/4	10.11.1.148		Unit is ready: Mode: R/O, Cluster is ONLINE and can tolerate up to ONE failure.
mysql-innodb-cluster/2*	active	idle	4/lxd/3	10.11.1.144		Unit is ready: Mode: R/W, Cluster is ONLINE and can tolerate up to ONE failure.
neutron-api/0*	active	idle	0/lxd/9	10.11.1.123	9696/tcp	Unit is ready
neutron-api-plugin-ovn/0*	active	idle		10.11.1.123		Unit is ready
neutron-mysql-router/0*	active	idle		10.11.1.123		Unit is ready
nova-cloud-controller/0*	active	idle	0/lxd/10	10.11.1.116	8774-8775/tcp	Unit is ready
nova-mysql-router/0*	active	idle		10.11.1.116		Unit is ready
nova-compute/0*	active	idle	1	10.11.0.21		Unit is ready
ntp/1	active	idle		10.11.0.21	123/udp	chrony: Ready
ovn-chassis/1	active	idle		10.11.0.21		Unit is ready

Figure 19 Output of the juju status command after deployment in the virtual environment

We will use the output to create login credentials for the dashboard. From it, we find out the IP address of the container on which the dashboard service is running and enter the command for creating a user.

```
juju run ceph-dashboard/0 add-user username=admin role=administrator
```

The command prints a password that needs to be saved. We can add more users and also assign them different privileges. After obtaining the credentials, as in the case of the manual installation, we connect to the address https://<IP_address>:8443 . We applied the same procedure in the deployment on physical devices as well.

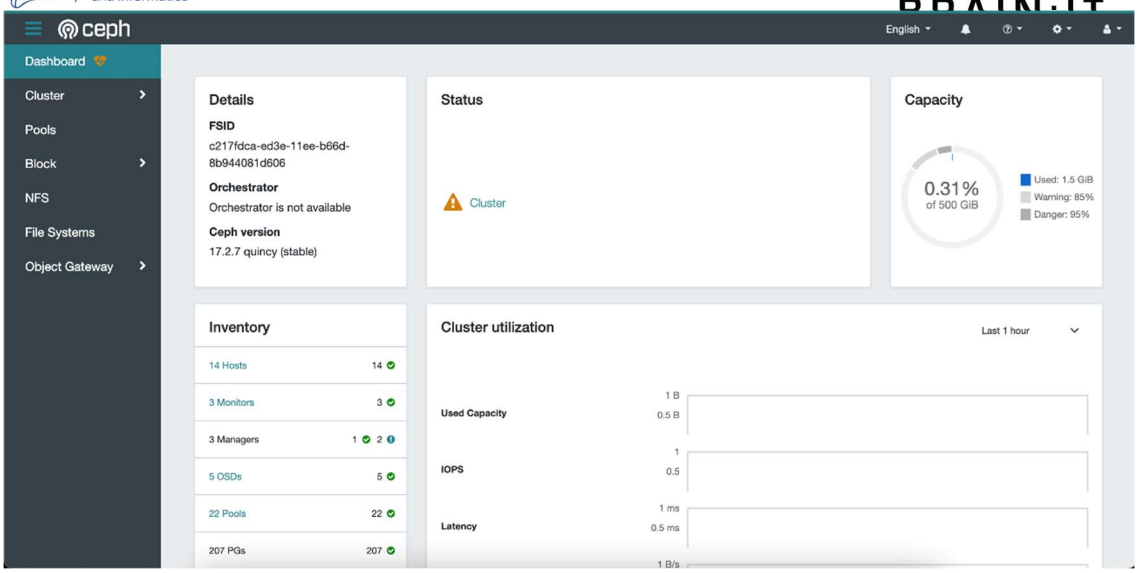


Figure 20 Ceph dashboard after deployment in the virtual environment

Figure 20 shows the state of the Ceph cluster after deployment. 5 OSD daemons are deployed, the total capacity of the cluster is 500GB and the cluster is in the "*warning*" state. When the cursor is moved over "*Cluster*", we will see the details of the warning. In this case, it is a bug whereby the manager restarts from time to time at random intervals, which causes a change in the "health" of the cluster. More detailed outputs of the cluster state can be obtained using the following commands for connecting to the Ceph monitor container:

```
juju ssh ceph-mon/leader
sudo -u ceph ceph status
```

Another bug that currently exists is that despite specifying the Ceph version for deployment as "*reef*", the charm deploys the "*quincy*" version. These bugs are present in the deployments in both environments.

After consideration, we finally decided to use both environments for the subsequent testing of Ceph configuration options with OpenStack, which we will use for different testing processes and their verification.

6. CEPH CONFIGURATION OPTIONS

This chapter describes the options that can be carried out after the cluster has been deployed. Ceph provides robust mechanisms for optimizing data management. We will focus on the basic processes of configuring and managing the cluster as a whole. Although the processes are described on the basis of integration with OpenStack, all the processes described below can also be applied in the case of a standalone cluster.

6.1 Adding OSD disks

Increasing the capacity of a Ceph cluster requires adding new disks to existing nodes, or adding new nodes. This process is crucial for expanding the storage capabilities of the cluster and thereby also for a possible increase in data redundancy. In addition to increasing capacity and data redundancy, the workload is distributed more evenly within the cluster. The adding process involves identifying a suitable node for the disk, preparing the disk and integrating it into the cluster using the Ceph OSD tools.

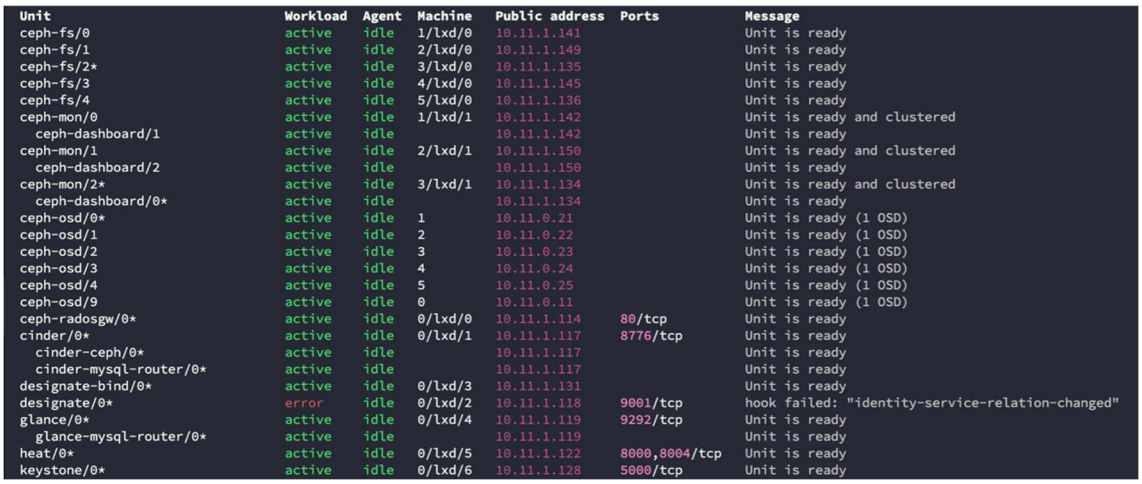
Adding disks to a cluster that was deployed using Juju is specific. In a standalone cluster, identifying the server to which we have added a disk is straightforward, and the entire configuration is carried out on the given node. In the case of Juju, it is more complicated. In the bundle file, we defined so-called applications. Applications are common to the entire model in Juju. What is important for us is the so-called “unit”. In the case of Ceph, a unit can be compared to one specific real node. Although Juju would allow another unit to be created on the same server (a so-called “machine”), this procedure is not recommended.

6.1.1 Adding a new unit

Adding a new unit works on the assumption that we already have the “machine” prepared. To add a unit, we use the **juju add-unit** command with the appropriate flags.

```
juju add-unit ceph-osd --m <model_name> --n <count> --to <id_machine>
```

The `--m` flag assigns the model to which we want to add the unit, `--n` determines the number of units and `--to` determines the machine on which the unit will be deployed [52].



Unit	Workload	Agent	Machine	Public address	Ports	Message
ceph-fs/0	active	idle	1/lxd/0	10.11.1.141		Unit is ready
ceph-fs/1	active	idle	2/lxd/0	10.11.1.149		Unit is ready
ceph-fs/2*	active	idle	3/lxd/0	10.11.1.135		Unit is ready
ceph-fs/3	active	idle	4/lxd/0	10.11.1.145		Unit is ready
ceph-fs/4	active	idle	5/lxd/0	10.11.1.136		Unit is ready
ceph-mon/0	active	idle	1/lxd/1	10.11.1.142		Unit is ready and clustered
ceph-dashboard/1	active	idle		10.11.1.142		Unit is ready
ceph-mon/1	active	idle	2/lxd/1	10.11.1.150		Unit is ready and clustered
ceph-dashboard/2	active	idle		10.11.1.150		Unit is ready
ceph-mon/2*	active	idle	3/lxd/1	10.11.1.134		Unit is ready and clustered
ceph-dashboard/0*	active	idle		10.11.1.134		Unit is ready
ceph-osd/0*	active	idle	1	10.11.0.21		Unit is ready (1 OSD)
ceph-osd/1	active	idle	2	10.11.0.22		Unit is ready (1 OSD)
ceph-osd/2	active	idle	3	10.11.0.23		Unit is ready (1 OSD)
ceph-osd/3	active	idle	4	10.11.0.24		Unit is ready (1 OSD)
ceph-osd/4	active	idle	5	10.11.0.25		Unit is ready (1 OSD)
ceph-osd/9	active	idle	0	10.11.0.11		Unit is ready (1 OSD)
ceph-radosgw/0*	active	idle	0/lxd/0	10.11.1.114	80/tcp	Unit is ready
cinder/0*	active	idle	0/lxd/1	10.11.1.117	8776/tcp	Unit is ready
cinder-ceph/0*	active	idle		10.11.1.117		Unit is ready
cinder-mysql-router/0*	active	idle		10.11.1.117		Unit is ready
designate-bind/0*	active	idle	0/lxd/3	10.11.1.131		Unit is ready
designate/0*	error	idle	0/lxd/2	10.11.1.118	9001/tcp	hook failed: "identity-service-relation-changed"
glance/0*	active	idle	0/lxd/4	10.11.1.119	9292/tcp	Unit is ready
glance-mysql-router/0*	active	idle		10.11.1.119		Unit is ready
heat/0*	active	idle	0/lxd/5	10.11.1.122	8000,8004/tcp	Unit is ready
keystone/0*	active	idle	0/lxd/6	10.11.1.128	5000/tcp	Unit is ready

Figure 21 Verification of successful unit deployment

After the unit has been deployed, the current configuration for the Ceph-Osd charm is applied, which we can display using the `juju config ceph-osd` command. It contains all the disks that we already defined in the bundle file in the "osd-devices:" section. Since we specified the device /dev/sdb and "node1" contains this disk, the unit automatically deploys an OSD daemon for the given disk. As shown in Figure 21, the deployment went well and we can see the added unit "ceph-osd/9". Verification directly in the Ceph cluster can be performed using the CLI command `sudo -u ceph ceph osd tree` after logging in to the monitor unit, or in the dashboard – Figure 22.

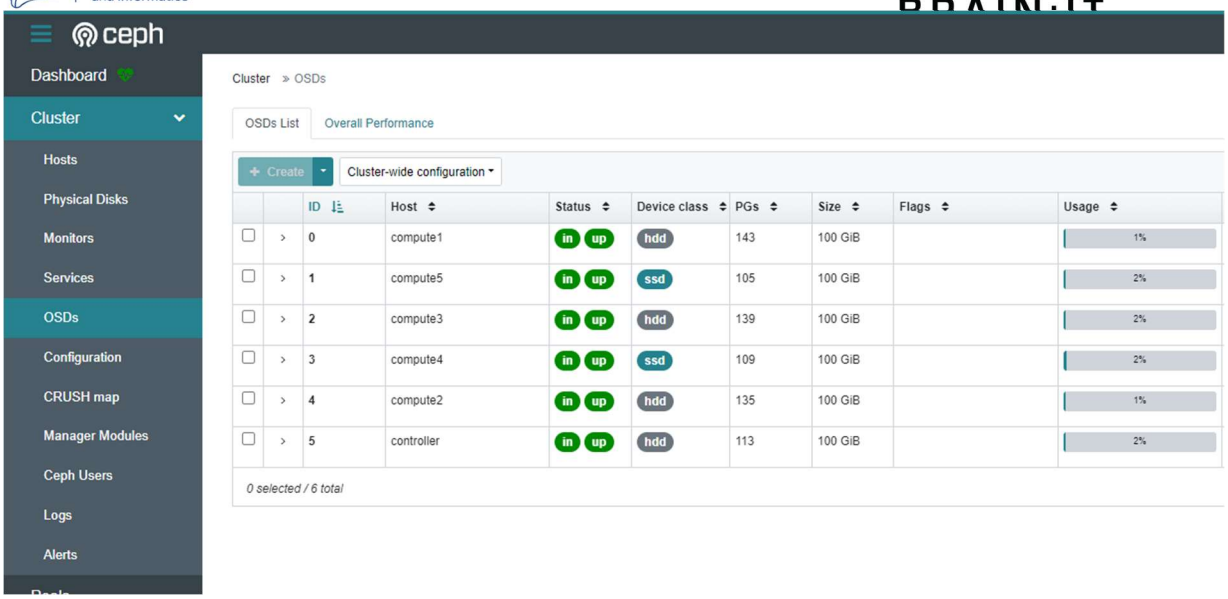


Figure 22 Verification of adding a node and disk to the cluster in the Ceph dashboard

To add disks not defined in the charm configuration file, we proceed as in the case of adding disks to an already existing unit.

6.1.2 Adding disks to an existing OSD unit

When adding disks to an already existing unit, we use so-called “actions”, which can be found on the charmhub.io website [48]. We use the “add-disk” command and define the disks by means of the **osd-devices** parameter.

```
juju run ceph-osd/<#osd> add-disk osd-devices=/dev/<disk/disks_designation>
```

After entering the **juju status** command, the “*maintenance*” state and a message about the initialization of the specified disk should be displayed for the respective unit (Figure 23). If we have specified more disks, the disks are initialized one after another and the message displays the disk currently being initialized [52].

Unit	Workload	Agent	Machine	Public address	Ports	Message
ceph-fs/0	active	idle	0/lxd/0	172.20.1.59		Unit is ready
ceph-fs/1	active	idle	1/lxd/0	172.20.1.51		Unit is ready
ceph-fs/2*	active	idle	2/lxd/0	172.20.1.46		Unit is ready
ceph-mon/0	active	idle	0/lxd/1	172.20.1.54		Unit is ready and clustered
ceph-dashboard/1	active	idle		172.20.1.54		Unit is ready
ceph-mon/1	active	idle	1/lxd/1	172.20.1.66		Unit is ready and clustered
ceph-dashboard/2	active	idle		172.20.1.66		Unit is ready
ceph-mon/2*	active	executing	2/lxd/1	172.20.1.47		Unit is ready and clustered
ceph-dashboard/0*	active	idle		172.20.1.47		Unit is ready
ceph-osd/0	maintenance	idle	0	172.20.0.11		Initializing device /dev/sdc
ceph-osd/1	active	idle	1	172.20.0.12		Unit is ready (1 OSD)
ceph-osd/2*	active	idle	2	172.20.0.13		Unit is ready (1 OSD)
ceph-radosgw/0*	active	idle	1/lxd/2	172.20.1.67	80/tcp	Unit is ready
cinder/0*	active	idle	0/lxd/2	172.20.1.60	8776/tcp	Unit is ready
cinder-ceph/0*	active	idle		172.20.1.60		Unit is ready
cinder-mysql-router/0*	active	idle		172.20.1.60		Unit is ready
designate-bind/0*	active	idle	0/lxd/4	172.20.1.44		Unit is ready
designate/0*	active	idle	0/lxd/3	172.20.1.42	9001/tcp	Unit is ready
glance/0*	active	idle	2/lxd/2	172.20.1.45	9292/tcp	Unit is ready
glance-mysql-router/0*	active	idle		172.20.1.45		Unit is ready
heat/0*	active	idle	1/lxd/3	172.20.1.72	8000,8004/tcp	Unit is ready

Figure 23 Disk initialization after adding it to the unit

After the initialization has finished successfully, the state returns to “active”. When comparing Figures 24 and 18, in the “Message” column in Figure 18 the unit “ceph-osd/0” reports 1 OSD. In Figure 24, the same unit, as expected, reports 2 OSDs, which means that the disk was added successfully.

Unit	Workload	Agent	Machine	Public address	Ports	Message
ceph-fs/0	active	idle	0/lxd/0	172.20.1.59		Unit is ready
ceph-fs/1	active	idle	1/lxd/0	172.20.1.51		Unit is ready
ceph-fs/2*	active	idle	2/lxd/0	172.20.1.46		Unit is ready
ceph-mon/0	active	idle	0/lxd/1	172.20.1.54		Unit is ready and clustered
ceph-dashboard/1	active	idle		172.20.1.54		Unit is ready
ceph-mon/1	active	idle	1/lxd/1	172.20.1.66		Unit is ready and clustered
ceph-dashboard/2	active	idle		172.20.1.66		Unit is ready
ceph-mon/2*	active	idle	2/lxd/1	172.20.1.47		Unit is ready and clustered
ceph-dashboard/0*	active	idle		172.20.1.47		Unit is ready
ceph-osd/0	active	idle	0	172.20.0.11		Unit is ready (2 OSD)
ceph-osd/1	active	idle	1	172.20.0.12		Unit is ready (1 OSD)
ceph-osd/2*	active	idle	2	172.20.0.13		Unit is ready (1 OSD)
ceph-radosgw/0*	active	idle	1/lxd/2	172.20.1.67	80/tcp	Unit is ready
cinder/0*	active	idle	0/lxd/2	172.20.1.60	8776/tcp	Unit is ready
cinder-ceph/0*	active	idle		172.20.1.60		Unit is ready
cinder-mysql-router/0*	active	idle		172.20.1.60		Unit is ready

Figure 24 Output of the juju status command after adding a disk to an existing unit

As when adding an entire unit to the cluster, verification can also be performed directly in Ceph using CLI commands, or in the dashboard. The difference is that no new node is added to the OSD table, but one node (cctest-comp1) will contain 2 OSD daemons, as shown in Figure 25.

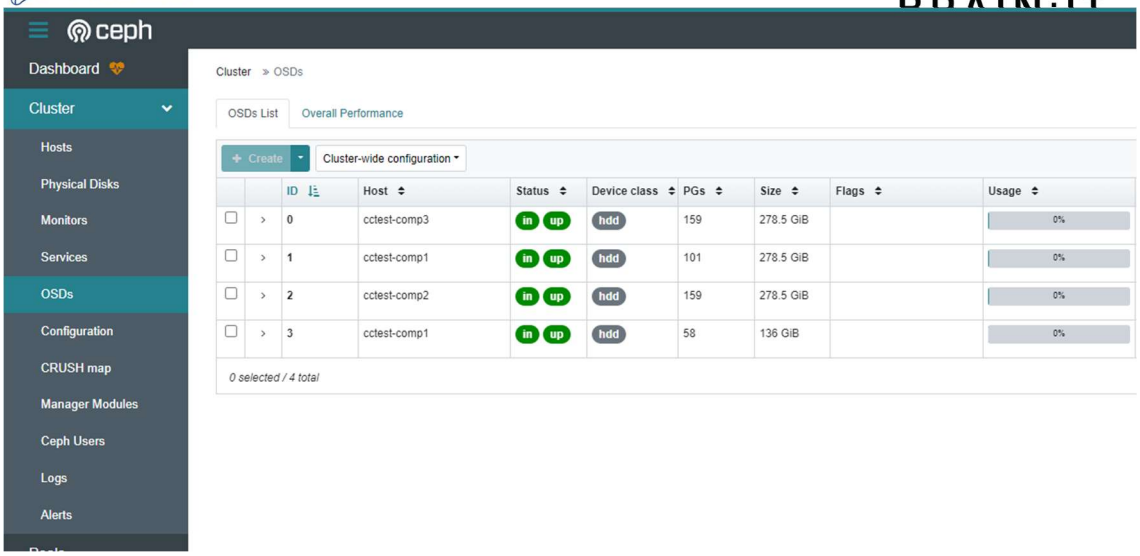


Figure 25 Verification of adding a disk to the cluster in the Ceph dashboard

If we add disks to the nodes in such a way that the same number of disks is added to each node, it is not necessary to perform the addition in this way for each disk. In such a case, we can leave the adding to Juju itself by changing the global configuration. However, the disk names must be the same on all nodes.

```
juju config ceph-osd osd-devices="$disky"
```

We replace the variable `disky` with the names of the disks, including the already existing ones, separated by a space [52].

6.2 Replacement and removal of OSD disks

Replacing and removing disks from a Ceph cluster using Juju is almost identical. Replacing a disk differs in that, in such a case, it is assumed that the physical disk has failed and we plan to replace it. First, we identify the disk that has failed. To find it, we can use the `sudo -u ceph ceph osd tree` command after logging in to the monitor, or the dashboard, where the failed disk will be in the "down" state (Figure 26). We mark the given disk in the cluster as "out" and then remove the disk using the `remove-disk` command.

```
juju run ceph-osd/<#osd> osd-out osds=<$ids>
juju run ceph-osd/<#osd> remove-disk osd-ids=<$ids>
```

If several OSDs have failed, it is possible to mark and remove them at once. In that case, the `$ids` variable needs to be replaced with the OSD IDs of the disks separated by commas [52].

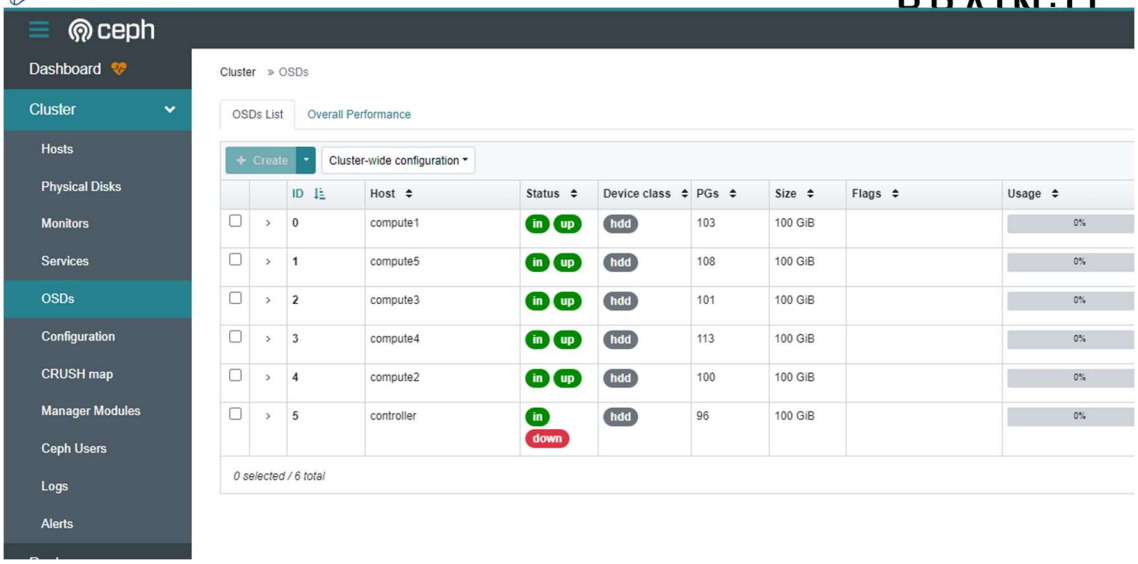


Figure 26 Display of an OSD disk failure in the dashboard

If we only want to remove the disk, it is necessary to also set the purge parameter to "*true*". Sometimes a timeout may occur and the safe removal may fail. In that case, it is possible to add the force parameter, which we likewise set to "*true*" [52].

```
juju run ceph-osd/<#osd> remove-disk osd-ids=<$ids> purge=true /force=true/
```

In a situation where we want to replace the disk, we do not specify these parameters. After the specified OSDs have been removed, we proceed further as in the case of the standard addition of a disk to an existing unit.

6.3 Ceph multi-backend for Cinder

Cinder is the module providing block storage for OpenStack. It manages and secures storage images, so-called "volumes", which are used as virtual disks for OpenStack instances. The implementation of multiple backends in Cinder is important because it enables different technologies to be used within a single cloud environment. This configuration enables users to meet different performance requirements while ensuring that the storage is able to support a wide range of applications. In addition, the use of multiple backends can improve the resilience of the system by dividing storage resources, thereby reducing the impact of any single point of failure.

Multiple pools make it possible to use the available infrastructure more efficiently. When a Ceph cluster is created using Juju charms, a number of preset

pools are created, each with its own role. It can be said that each service we configured for integration with OpenStack (Manila, Glance, Cinder,..) has its own assigned pool. These pools share all OSD disks jointly. In a simple infrastructure, such as the test environment, or one that uses the same type of physical disks, no problem arises. However, in the case of more complex requirements, e.g. the deployment of OpenStack or the use of different types of disks (HDD, SSD), this concept no longer provides sufficient efficiency and resilience. In the case of different types of disks, the SSD disks (or rather the PGs on these disks) will always be limited by the speeds of the PGs on the HDD disks [27]. Sharing OSDs among all pools means that, in the event of an outage of one or more OSDs, more than one service is affected.

6.3.1 Creating pools

As a demonstration, we will use the most basic type of storage division. We will create one pool that will be provided purely by SSD disks, and a second pool by HDD disks. In this way, it is possible to allocate storage more efficiently on the basis of the requirements of the given instance. Juju actions can be used to create the pools, but the actions no longer provide any options for the subsequent configuration. For this reason, we will apply the entire configuration, including the creation of the pool, on the unit of the main monitor directly in Ceph.

The first step is the classification of the disks into classes. Ceph is able to distinguish between HDD and SSD disks during installation. However, the classification can be done in any way. Using the `sudo -u ceph ceph osd tree` command, we find out the OSD IDs and the current class of the specific disks (Figure 27).

```
juju ssh ceph-mon/leader
sudo -u ceph ceph osd tree
```

```
ubuntu@juju-befae6-3-lxd-1:~$ sudo -u ceph ceph osd tree
ID   CLASS  WEIGHT   TYPE NAME           STATUS  REWEIGHT  PRI-AFF
 -1         0.58612  root default
 -3         0.09769      host compute1
  0    hdd  0.09769          osd.0           up   1.00000  1.00000
-11         0.09769      host compute2
  4    hdd  0.09769          osd.4           up   1.00000  1.00000
 -7         0.09769      host compute3
  2    hdd  0.09769          osd.2           up   1.00000  1.00000
 -9         0.09769      host compute4
  3    ssd  0.09769          osd.3           up   1.00000  1.00000
 -5         0.09769      host compute5
  1    ssd  0.09769          osd.1           up   1.00000  1.00000
```

Figure 27 Output of the ceph osd tree command

Since in our case the disks are virtual, all disks were labelled with the “hdd” class. For testing, we will set aside 2 disks, specifically from nodes 4 and 5, which we will classify as SSD disks. These OSDs had IDs 1 and 3. To change the class of a disk, we must first remove the original class from the OSD and then assign a new one. As already mentioned, the class name is arbitrary. For our purpose, we choose the name “ssd”.

```
sudo -u ceph ceph osd crush rm-device-class osd.<#osd>
sudo -u ceph ceph osd crush set-device-class <class_name> osd.<#osd>
```

The classification of the disks can be verified either directly in the CLI or in the dashboard.

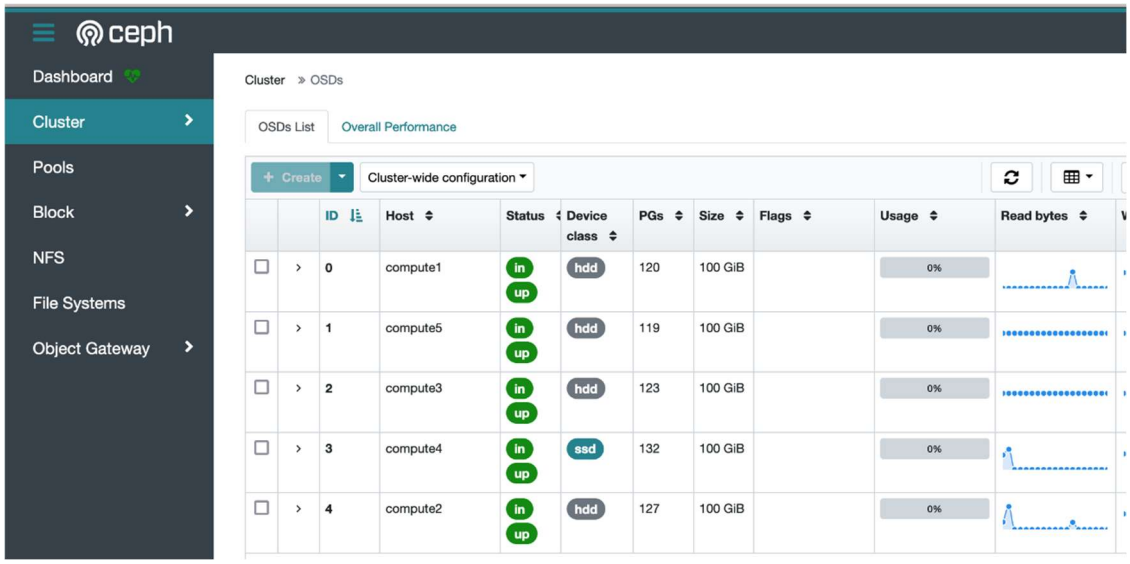


Figure 28 Verification of the OSD class change

Reserving OSDs for the individual pools is carried out by means of CRUSH algorithm rules. We will create the rule “*SSD_allocate*”, which we will set up so that objects intended to be written to the pool on which this rule is applied are

written precisely to the OSD disks classified into the specified class (ssd). A pool can have only one rule applied at any given time.

```
sudo -u ceph ceph osd crush rule create-replicated <rule_name> default \ host ssd
```

The command creates replicated storage. The **default** parameter specifies that the rule applies globally and is not intended only for a specific region, rack, etc. The **host** parameter specifies the priority of where the algorithm tries to create replicas, and the **ssd** parameter specifies the name of the class (bucket).

An alternative to this procedure is the creation of a bucket. In our case, however, we made use of the fact that Ceph creates the default buckets "*default_hdd*" and "*default_ssd*" in advance. If we set the class to "*ssd*", the disk is automatically moved to the default bucket. However, the administrator has the option of defining custom buckets and then applying them to individual CRUSH rules. All buckets can be exported in JSON format with the **sudo -u ceph ceph osd crush dump** command.

The next step is to create the pool and configure the basic settings.

```
sudo -u ceph ceph osd pool create <pool_name> replicated <CRUSH_rule>
sudo -u ceph ceph osd pool set <pool_name> size 2
sudo -u ceph ceph osd pool set <pool_name> pg_autoscale_mode on
sudo -u ceph ceph osd pool application enable <pool_name> <$aplikacia>
```

The **pool create** command creates the pool itself, and the **replicated** parameter determines the redundancy type of the pool. The **pool set** commands are used to configure the individual settings within the pool. The **size** parameter determines the replication factor. The **pg_autoscale_mode** parameter sets automatic scaling of the number of PGs in the pool. The **pool application enable** command sets the type of storage for which the pool is intended, in the form of interfaces (rbd - block, rgw - object, cephfs – file), specified by the **$aplikacia** variable. This setting is mandatory and without it the pool will not be functional.

We use the same procedure for the HDD pool as well. However, the configuration of the HDD pool is simplified by omitting the change of OSD class and the creation of the pool, since the class is already "hdd" and the pool was created by the Cinder-Ceph charm. The creation of the pools can be verified using the **sudo -u ceph ceph osd pool ls** command, or in the dashboard in the "Pools" tab.

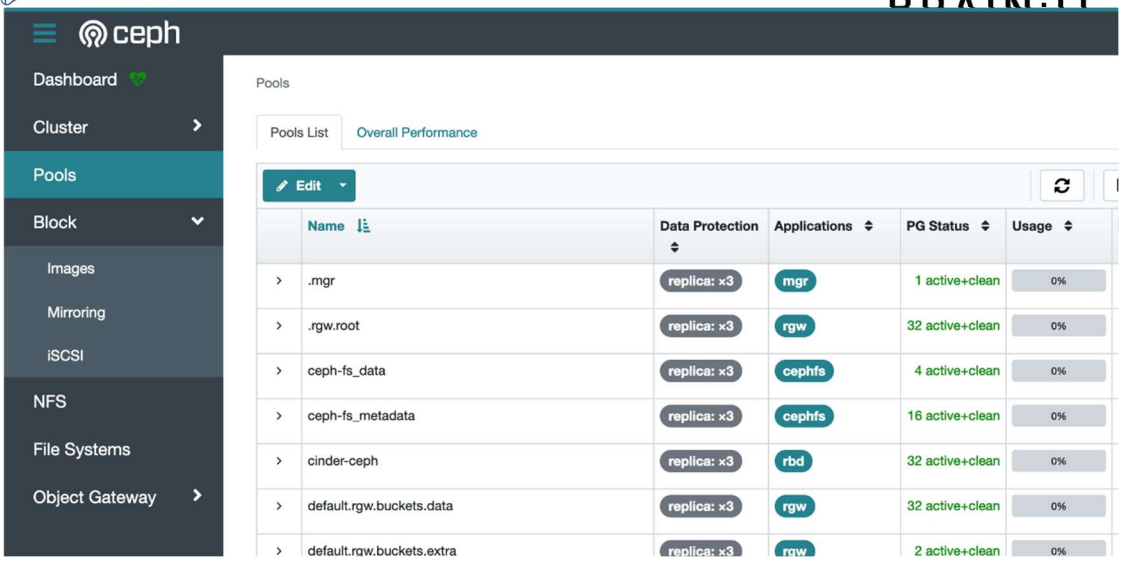


Figure 29 Pools in the Ceph cluster before adding a pool

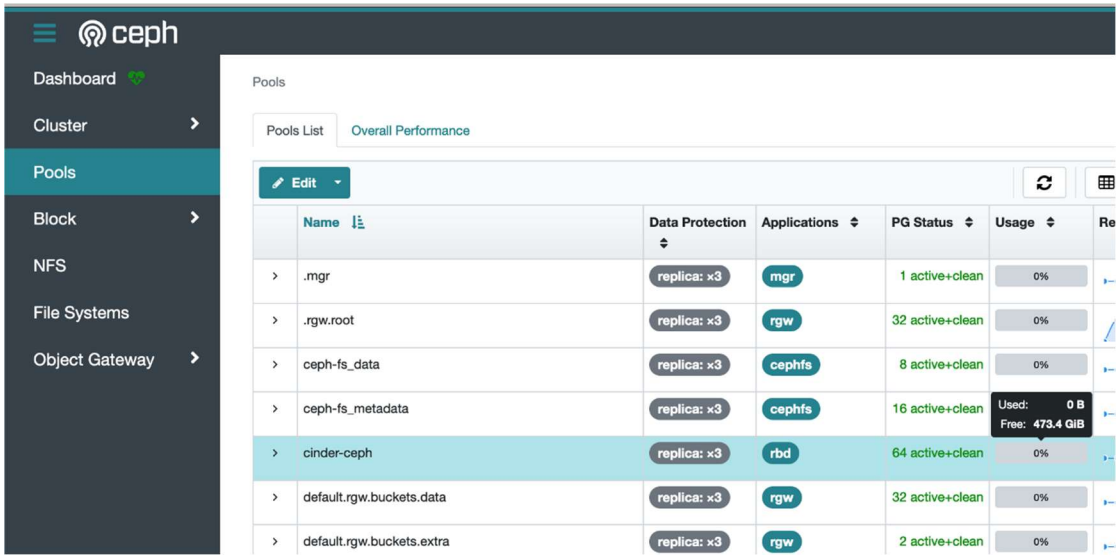


Figure 30 Capacity of the cinder-ceph pool before using the CRUSH rule



Figure 31 Capacity of the pools after applying the rules

Figure 31 serves as verification of the creation of the pool "cinder-ceph-ssd" and the assignment of the application to it, and of the change in capacity after applying the CRUSH rules to the individual pools.

However, it is necessary to point out the fact that the pool "cinder-ceph-ssd" created in this way has only one PG available. It is therefore necessary to set an appropriate number of PGs for the given pool. The PG values should be powers of 2, but this is not mandatory.

```
sudo -u ceph ceph osd pool set cinder-ceph-ssd pg_num <number_of_PGs>
```

If these were real SSD disks, it would be necessary to enable the TRIM function with the following command:

```
sudo -u ceph ceph config set osd bluestore_volume_selection_policy \ spdk_dmcrypt_discard
```

Name	Data Protection	Applications	PG Status
.mgr	replica: ×3	mgr	1 active+clean
.rgw.root	replica: ×3	rgw	32 active+clean
ceph-fs_data	replica: ×3	cephfs	8 active+clean
ceph-fs_metadata	replica: ×3	cephfs	16 active+clean
cinder-ceph	replica: ×3	rbd	64 active+clean
cinder-ceph-ssd	replica: ×2	rbd	32 active+clean
default.rgw.buckets.data	replica: ×3	rgw	32 active+clean

Figure 32 Verification of the change in the number of PGs

6.3.2 Assigning pools to the Cinder module

We will divide the multi-backend setup for Cinder into two steps. The first step is to modify the configuration file **/etc/cinder/cinder.conf** for the Cinder module. It can be found in the unit "*cinder/0*". Juju has already preconfigured this file for OpenStack and, thanks to the Cinder-Ceph charm, has set Ceph as the backend with all parameters. We will make use of this fact to set up a further pool/pools. All pools that are created and enabled in a so-called context in the Cinder configuration file are automatically available to Cinder. A context is the name or keyword in square brackets inside the configuration file. We copy the previously created context "*cinder-ceph*", which Juju created during deployment

with the Cinder-Ceph charm, and gradually modify its individual values.

```
#original configuration of the file /etc/cinder/cinder.conf
#...

enabled_backends = cinder-ceph

[backend_defaults]

[cinder-ceph]
volume_backend_name = cinder-ceph
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = cinder-ceph
rbd_user = cinder-ceph
rbd_secret_uuid = 21696ef1-d740-4f59-9eb7-70799c8d9ded
rbd_ceph_conf = /var/lib/charm/cinder-ceph/ceph.conf
report_discard_supported = True
rbd_exclusive_cinder_pool = True
rbd_flatten_volume_from_snapshot = False

#...
```

In our case, we copy this configuration, paste it and modify it slightly so that it contains another context.

```
#original configuration of the file /etc/cinder/cinder.conf
#...

enabled_backends = cinder-ceph,<name_of_next_pool>

[backend_defaults]

[cinder-ceph]
volume_backend_name = cinder-ceph
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = cinder-ceph
rbd_user = cinder-ceph
rbd_secret_uuid = 21696ef1-d740-4f59-9eb7-70799c8d9ded
rbd_ceph_conf = /var/lib/charm/cinder-ceph/ceph.conf
report_discard_supported = True
rbd_exclusive_cinder_pool = True
rbd_flatten_volume_from_snapshot = False

[<name_of_new_context>]
volume_backend_name = <pool_name_for_openstack>
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = <name_of_pool_created_in_ceph>
rbd_user = cinder-ceph
rbd_secret_uuid = 21696ef1-d740-4f59-9eb7-70799c8d9ded
rbd_ceph_conf = /var/lib/charm/cinder-ceph/ceph.conf
report_discard_supported = True
rbd_exclusive_cinder_pool = True
rbd_flatten_volume_from_snapshot = False


#...
```

Although the names of the context and of the `volume_backend_name` and `rbd_pool` parameters do not have to be the same, it is best practice. Only the names of the pools in Ceph and in the `rbd_pool` parameter must be the same. In our case, we therefore set all three names to "*ceph-cinder-ssd*". It is also necessary to modify the `enabled_backends` parameter, to which we add the name of the context. The

individual contexts are separated by a comma without a space. In this way, we can add any number of pools as backends for Cinder. After these modifications, it is necessary to restart all services of the Cinder module.

```
sudo systemctl restart cinder-volume.service
sudo systemctl restart cinder-scheduler.service
```

If, after restarting these services, both services start and are active, the configuration is correct. If there were an error in the configuration, or if we assigned incorrect authentication data, the cinder services would return an error and fail. Entering a non-existent pool name in the `rbd_pool` parameter does not return an error, but any attempt to read or write data to the given pool will fail. These failures are recorded in the logs of the "*ceph-volume*" service. Verification can be performed using CLI commands or in the OpenStack dashboard [53].

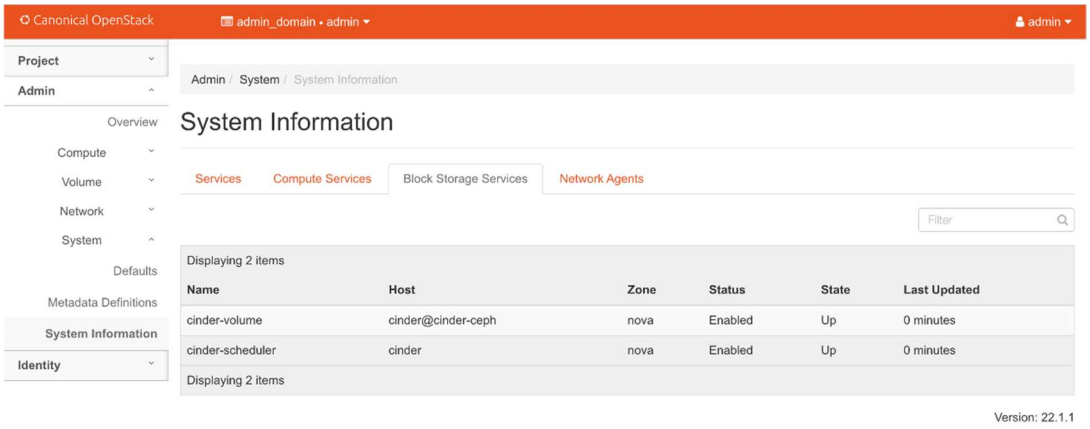


Figure 33 State before adding the backend in the dashboard

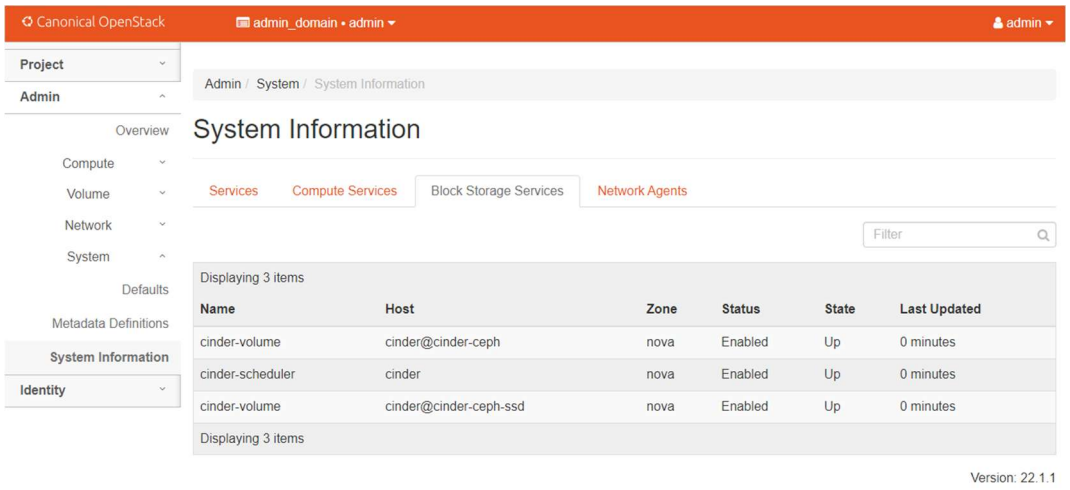


Figure 34 Verification of adding the backend in the dashboard

```
kysel@kysel-dp-maas:~$ cinder get-pools
+----------+--------------------------------+
| Property | Value                          |
+----------+--------------------------------+
| name     | cinder@cinder-ceph#cinder-ceph |
+----------+--------------------------------+
+----------+----------------------------------------+
| Property | Value                                  |
+----------+----------------------------------------+
| name     | cinder@cinder-ceph-ssd#cinder-ceph-ssd |
+----------+----------------------------------------+
kysel@kysel-dp-maas:~$
```

Figure 35 Verification of adding the backend in the CLI

6.3.3 Configuring OpenStack to use multiple pools

The second step in setting up the multi-backend for Cinder is configuring OpenStack itself so that it is possible to choose which pool we want to use for a given volume or instance. To achieve this, we need to set up so-called "*volume types*". Volume types can be created using CLI commands or in the dashboard. Configuration using the dashboard is slightly clearer, so we choose this method.

After logging in to the OpenStack dashboard, we go to the **Admin -> Volume -> Volume Types** tab. We click the **Create Volume Type** button. We enter a name and check that the "Public" option is ticked. We click the **Create Volume Type** button and wait until it is created.

After creation, it is still necessary to specify the backend for the given volume type. On the right, we click the downward-pointing arrow next to the **Create Encryption** button and select the **View Extra Specs** option from the menu. We click the **Create** button. In the "*Key*" field, we enter the parameter "*volume_backend_name*", and in the "*Value*" field we enter the name of the backend that we assigned in the same parameter in the Cinder configuration file. It is important that these names are the same. We then click the **Create** button. In this way, we create one volume type for each backend.

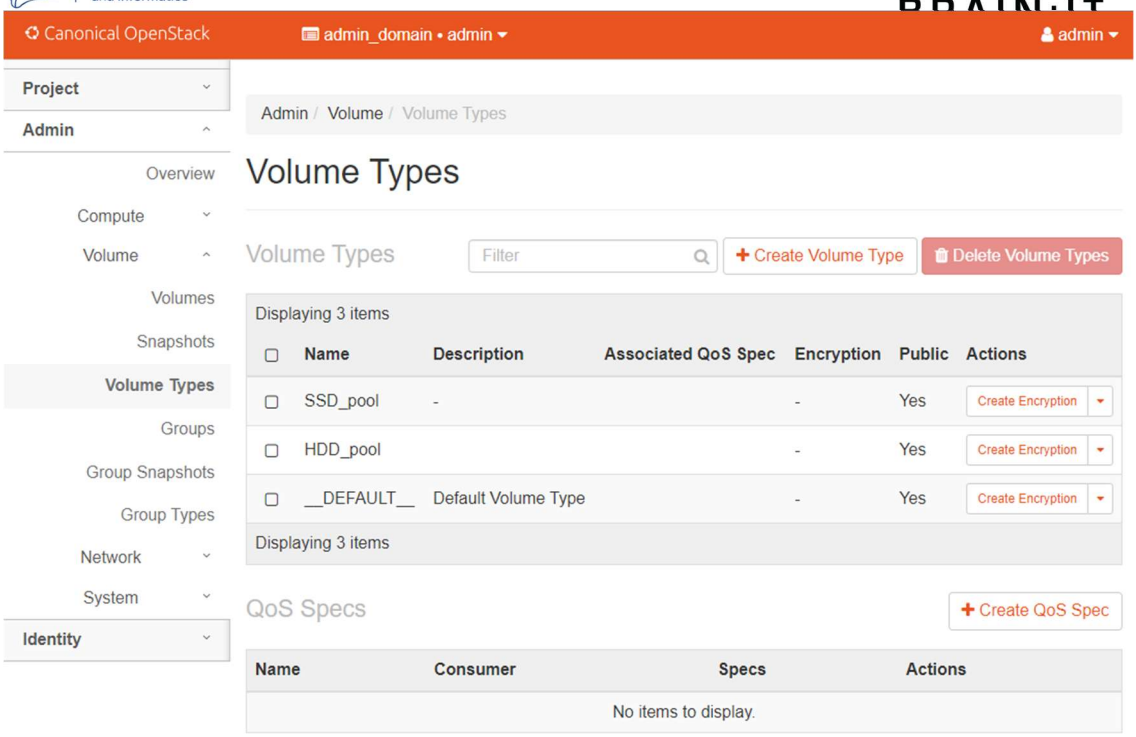


Figure 36 Verification of the Volume Types settings in the dashboard

The terminal can likewise be used for verification. The **openstack volume type list** command displays all volume types in OpenStack. To display more detailed information about a given volume type, the **openstack volume type show <volume_type_name>** command is available.

```
kysel@kysel-dp-maas:~$ openstack volume type list
+--------------------------------------+-------------+-----------+
| ID                                   | Name        | Is Public |
+--------------------------------------+-------------+-----------+
| 0c29c9e1-59c8-4a4c-affb-92d8a2314a79 | SSD_pool    | True      |
| 44f44076-fbe5-4d8d-a5c7-c8625bf0209b | HDD_pool    | True      |
| f0343753-da12-4a8c-b2ff-0db5d1aee19a | __DEFAULT__ | True      |
+--------------------------------------+-------------+-----------+
kysel@kysel-dp-maas:~$ openstack volume type show SSD_pool
+--------------------+-----------------------------------------+
| Field              | Value                                   |
+--------------------+-----------------------------------------+
| access_project_ids | None                                    |
| description        | None                                    |
| id                 | 0c29c9e1-59c8-4a4c-affb-92d8a2314a79    |
| is_public          | True                                    |
| name               | SSD_pool                                |
| properties         | volume_backend_name='cinder-ceph-ssd'   |
| qos_specs_id       | None                                    |
+--------------------+-----------------------------------------+
kysel@kysel-dp-maas:~$
```

Figure 37 Verification of volume types using the CLI

6.3.4 Functionality verification

We will use a simple procedure for verification. We upload an arbitrary cloud image of a Linux distribution to OpenStack, thereby verifying the operation of the Glance module. We then create a volume in each pool by copying this image onto the given volume.

In our case, we chose a cloud image of the Ubuntu operating system in QCOW2 format. We used the OpenStack dashboard to upload the image. During this process, we logged in to the Ceph dashboard and focused on the "*glance*" pool.

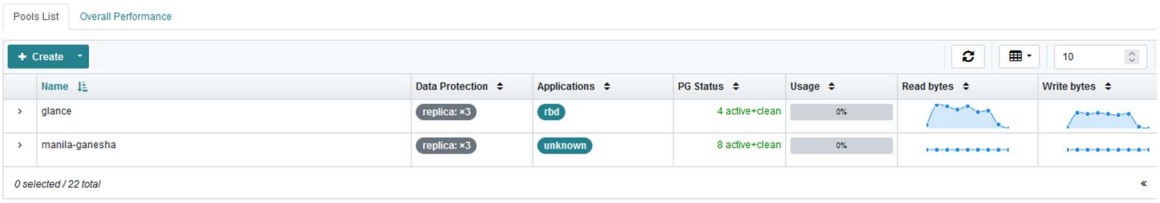


Figure 38 Disk activity during image upload

In Figure 38, we can see that there was activity on the disk during the image upload process. In Figure 39, we can see the utilization of the "glance" pool after the image was uploaded. However, the pool utilization shows the total space used on the disks, including the replicas created. If we use a replication factor of 3, the pool capacity must be divided by three. For a replication factor of 5, we divide it by five, etc.

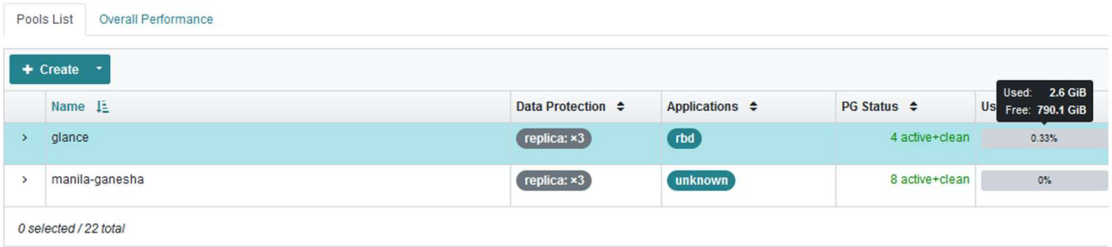


Figure 39 Utilization of the glance pool

To verify the backend, we created a volume for each volume type using the OpenStack dashboard. During creation, we used the "Image" option for "Volume Service" and selected the uploaded Ubuntu cloud image as the image. We

set the required “Type” to the SSD_pool or HDD_pool volume type. As in the previous case, we monitored this process in the Ceph dashboard.

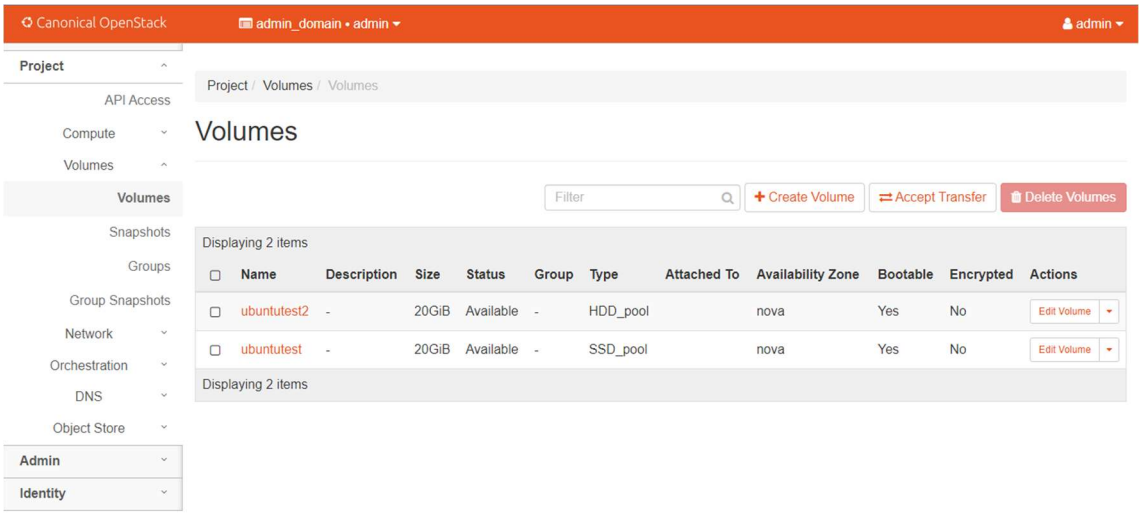


Figure 40 Created volumes in the OpenStack dashboard

As in the case of the “glance” pool, during the testing of these two pools we also observed activity on the disks during the creation of the individual volumes (Figure 41).

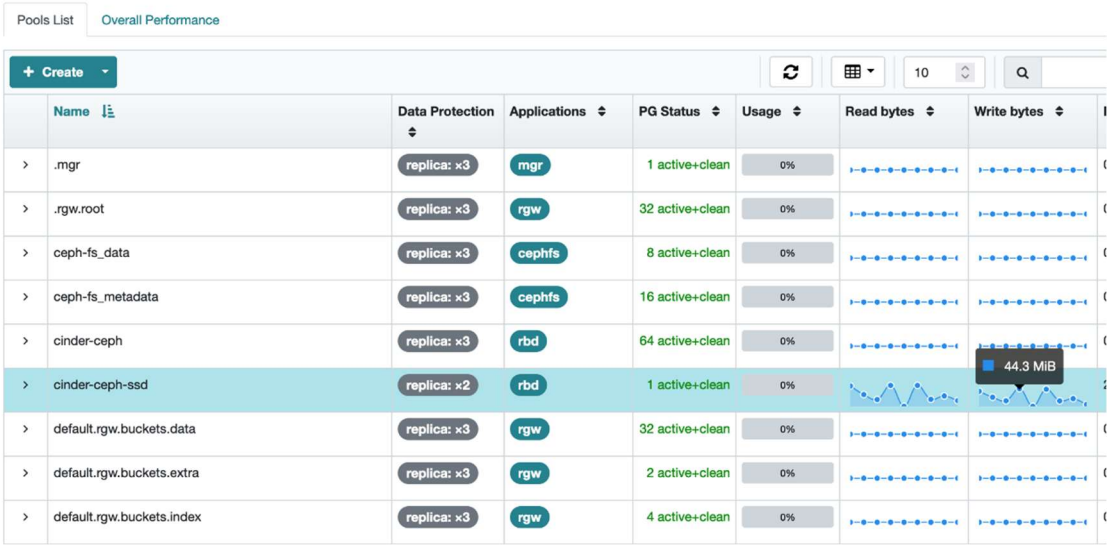


Figure 41 Activity on the disks during volume creation

After the creation of the volumes was completed, we can see in Figure 42 that disk utilization increased in both cases. This means that the multi-backend is set up correctly and the individual volume types are also set up correctly. Pools configured in this way can be fully used in OpenStack as storage for instances.

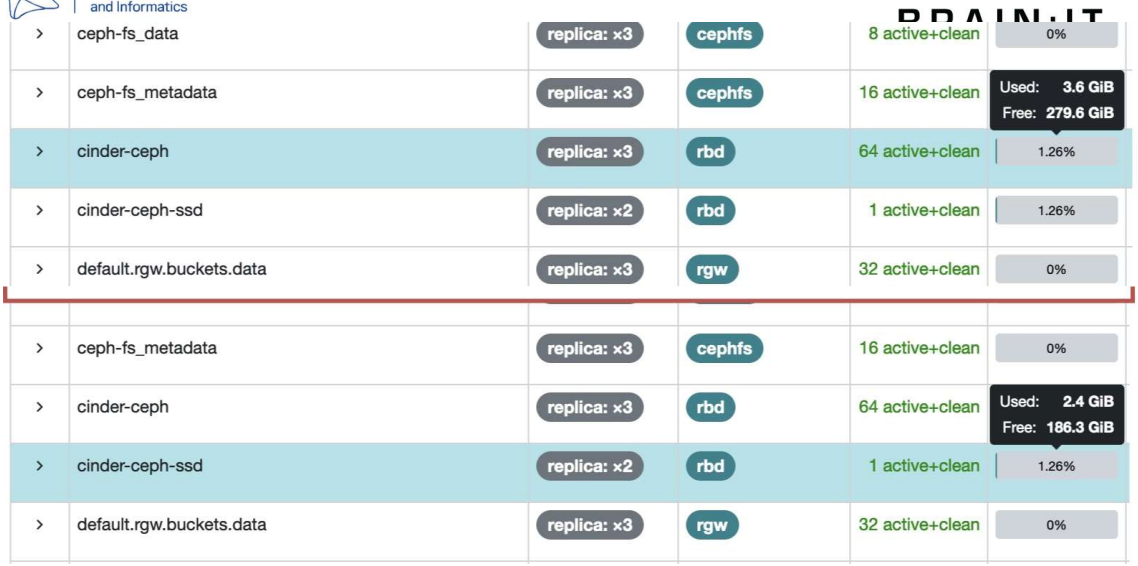


Figure 42 Verification of writing the volumes to the respective pools

6.4 Manual download of an image from a pool

Managing OpenStack virtualized environments with Ceph storage as the backend may require exporting a volume for various reasons, e.g. for backup, migration or archiving. Currently, no tool is available that would enable automated export. This subchapter describes the exporting process via the OpenStack CLI or directly from the Ceph cluster via the Ceph client. These processes can then be used in custom-made scripts, which can subsequently make up for the missing automation.

6.4.1 Exporting using OpenStack

We begin the process using OpenStack by finding out the ID of the volume we want to export. We then create a snapshot, obtain the snapshot ID and create a new volume from the given snapshot. We obtain the ID of the new volume and create an image from it. We then save this image as a file that we can download.

```
openstack volume list
openstack volume snapshot create --volume <volume_id> <snapshot_name>
openstack volume snapshot list
openstack volume create --snapshot <snapshot_id> --size <size> \ <new_volume_name>
openstack volume list
openstack image create --volume <new_volume_name> <image_name>
openstack image save --file <file_name> <image_name>
```

The created file can be handled like an ordinary file, so we can transfer it, e.g. using the scp tool [54].

6.4.2 Exporting using the Ceph client

A volume can also be exported directly from the Ceph cluster. In this case, however, it is important that we have administrative access to the cluster, since to use this method we need to configure a Ceph client.

First, we install the "*ceph-common*" package. Then we transfer the client keys and the configuration file from the monitor container to the client device. We verify the functionality of the client and display all volumes in the pool. We then export this volume to the device on which the Ceph client is installed.

```
sudo apt update
sudo apt install ceph-common
```

After installing the package, we connect to the monitor unit and copy the client key and the Ceph configuration file to the client.

```
sudo scp /etc/ceph/ceph.conf <client_username>@<client_IP>:/etc/ceph/
sudo scp /etc/ceph/ceph.client.admin.keyring \ <client_username>@<client_IP>:/etc/ceph/
```

On the client device, we enter the following commands:

```
ceph -s #verifying access to the cluster
rbd ls <pool_name>
rbd export <pool_name>/volume-<volume_id> <file_name_on_client>
```

A major difference between exporting via OpenStack and exporting directly from the cluster is that, when exporting directly from Ceph, the resulting file we obtain is a so-called "raw" block image. When exporting from OpenStack, we obtain the image as a file in a certain format, e.g. QCOW2. Another difference is that the exporting process via OpenStack is more "laborious", but it is also safer, thanks to the creation of a snapshot and the subsequent work with it. When exporting directly from the cluster, a situation may arise in which an already transferred sector inside the block image changes and the newly transferred sectors take this change into account, whereas the image contains outdated data. Although this situation is not frequent, in isolated cases the volume may become corrupted.

7. RECOMMENDATIONS FOR THE DEPLOYMENT OF A CEPH CLUSTER

In this chapter, we will deal with the creation of recommendations for the deployment of a Ceph cluster integrated into the OpenStack platform. The recommendations are based on the official recommendations from the official Ceph documentation and on the knowledge gained during testing and while working on this thesis.

7.1 Deployment concept

The recommendations will focus on the configuration of Ceph, or rather the management of the cluster as a whole, so that they fulfil three important conditions. The first condition is efficient use of the available cluster resources. The second condition is ensuring sufficient data redundancy in the event of a node or disk outage. The last condition is to design the recommendations so that they encompass the expansion of the cloud, and thereby also the expansion of the Ceph cluster, whether in the form of adding nodes or adding disks to already existing nodes.

7.2 Deployment environment

The environment in which the cloud will be deployed will consist of the currently planned 6 nodes. In terms of the number and capacity of disks, these nodes are identical. Each node will have two SSD disks with a size of 1.92 TB and five HDD disks. Of these five disks, one is designated as the system disk with a size of 300 GB and the other four HDD disks as data disks for Ceph, each with a size of 1.2 TB.

7.3 Ceph configuration

We will divide the configuration of Ceph during integration with OpenStack into two parts. In the first part, we deal with the configuration within the bundle file. The second part is devoted to the configuration of the cluster after deployment, including the logical division of the cluster.

7.3.1 Bundle file configuration

For the configuration in the bundle file, we recommend making only slight changes. Based on the experience gained during testing, I recommend deploying Ceph-Osd on all nodes, but initializing only the HDD disks that are available. The reason is clarity within the OSD disks. Since OSD disks are assigned IDs starting from zero in the order in which they were initialized, we can thereby make the subsequent configuration easier.

We recommend the use of "*replicated*" pools. The reason is that "*erasure-coded"* pools do not provide a sufficient level of redundancy. The advantages of these pools, namely the use of almost the full capacity of the cluster and the speed of service, do not outweigh, in a cloud environment, the lack of redundancy, which is one of the main conditions.

Although several configuration parameters have suitable preset values, as best practice we recommend specifying the following parameters manually. With version changes, the preset values may change, which may subsequently cause unpredictable behaviour. These parameters are listed in Table 4 below.

In the Ceph-Osd application, we recommend setting one important parameter. The **ignore-device-errors:** parameter prevents the unit from becoming non-functional in the event that the initialization of one disk fails. Since the unit will contain more than one OSD, based on the experience from testing, we recommend enabling this option.

For each application, we recommend setting the **use-syslog** parameter to "*true*". Although this may significantly increase the number of logs in the system, the logs do not have to be searched for in various files, but are in one place – in the syslog. With suitable log aggregation, this leads to easier management and detection of faults or errors in the cluster.

Application	Parameter	Value
Ceph-Mon	expected-osd-count:	11
	monitor-count:	5
	pg-autotune:	true

Ceph-Osd	ignore-device-errors:	true
Ceph Fs	ceph-osd-replication-count: pool-type:	3 replicated
Ceph Radosgw	ceph-osd-replication-count: pool-type: port:	3 replicated 80
Cinder Ceph	ceph-osd-replication-count: pool-type: volume-backend-name:	3 Replicated cinder-ceph-hdd
Glance	ceph-osd-replication-count: pool-type:	3 replicated

Table 4 Recommended parameters and their values for the bundle file

7.3.2 Configuration of the deployed cluster

After deploying the cluster with the bundle file, we recommend logically separating the individual OSDs into three to five buckets, since by default all pools use all OSD daemons. One bucket will consist only of SSD disks, which will be intended for the SSD pool for Cinder. The second bucket will use HDD disks for the HDD pool as the second back-end for Cinder. At our discretion, we can then proceed in two ways:

- Create a third bucket that will contain the separated OSD disks, and apply this bucket to the other pools created by Juju, including the pools for the Glance and Manila modules.
- Reserve disks and create additional buckets for the Glance and Manila modules and, as in the first case, create a general bucket for the other pools.

Both approaches have their advantages and disadvantages. With the creation of only three buckets, a failure of an OSD disk in the “general” bucket affects several services at once. In the case of five buckets, this risk is significantly reduced, which is the advantage of this approach. The disadvantage, however, is the increased demand for disks, and thereby on the capacity of the individual pools, which is not a problem in the first approach.

Having considered several factors, for deployment on 6 nodes we recommend the first approach, reserving one OSD disk on each node for the common bucket. In the future, when expanding with further compute nodes or when the number of disks is sufficiently increased, we recommend proceeding by creating five buckets.

CONCLUSION

The aim of this thesis was to analyse the available options and to select suitable software for deploying distributed storage for a cloud built on the OpenStack platform. At the beginning, free software solutions providing distributed storage deployment services were examined. On the basis of predefined criteria, the Ceph software was selected.

To gain a better understanding of how it works, the architecture of the Ceph system and its main components were analysed. Basic concepts such as pools, PGs and the CRUSH and Paxos algorithms, an understanding of which is essential for the correct deployment and configuration of a Ceph cluster, were also explained.

In the practical part, a manual deployment of the cluster was carried out, which made it possible to understand in detail the operation of the individual components during a real deployment and their configuration. Subsequently, the Ceph cluster was installed and integrated with the OpenStack platform using the OpenStack Charms tools, which simplify the deployment process into a cloud environment. During this procedure, the configuration options when using a so-called bundle file were examined and documented in detail.

After the functionality of the cluster had been verified, attention focused on the configuration of the basic components. Owing to technical problems with both the virtual and the physical environment, it was not possible to implement data caching. However, the annexes include documentation of the caching configuration, which has not yet been tested and is therefore not part of the main procedure.

Finally, recommendations were drawn up for the deployment of the Ceph software in the department's cloud environment, based on the knowledge gained during the testing of the manual deployment, the use of charms and the subsequent configuration. The aim was to create reliable distributed storage for the OpenStack cloud platform that meets all the specified conditions.

Monitoring of OpenStack and the Ceph cluster using the Prometheus software was not carried out, as it exceeded the scope of the project. However, the

results of the work can serve as a basis for further optimization and the introduction of monitoring using caching and the Prometheus software.

LIST OF REFERENCES

1. BMC SOFTWARE, INC. Distributed Storage Model - Concept and Principles - Documentation for BMC Discovery content reference - BMC Documentation [online]. [Date 29/3/2024]. Available at: https://docs.bmc.com/docs/discovery/contentref/distributed-storage-model-concept-and-principles-997880678.html
2. NUTANIX. What is Distributed Storage? Explore Distributed Cloud | Nutanix [online]. [Date 29/3/2024]. Available at: https://www.nutanix.com/info/distributed-storage
3. TELNYX. What is distributed storage, and why does it matter? [online]. [Date 29/3/2024]. Available at: https://telnyx.com/resources/what-is-distributed-storage
4. WEKA.IO. Distributed File Systems Explained | Features and Advantages [online]. [Date 29/3/2024]. Available at: https://www.weka.io/learn/distributed-file-systems/distributed-file-system/
5. HOSTKEY. What is Distributed Storage? Types and Examples [online]. [Date 21/3/2024]. Available at: https://hostkey.com/blog/53-what-is-distributed-storage/
6. CLOUDIAN, INC. Distributed Storage: What's Inside Amazon S3? [online]. [Date 21/3/2024]. Available at: https://cloudian.com/guides/data-backup/distributed-storage/
7. JULIO'S BLOG. Selecting the right storage backend for your OpenStack cloud [online]. [Date 9/10/2023]. Available at: https://www.juliosblog.com/selecting-the-right-storage-backend-for-your-openstack-cloud/
8. STORAGE SWISS. 2015. Selecting the right OpenStack storage module [online]. [Date 9/10/2023]. Available at: https://storageswiss.com/2015/10/14/selecting-the-right-openstack-storage-module/
9. CEPH DOCUMENTATION. Reef [online]. [Date 9/10/2023]. Available at: https://docs.ceph.com/en/reef/
10. OPENSTACK. 2023. OpenStack Administration Guide [online]. [Date 9/10/2023]. Available at: https://docs.openstack.org/2023.2/admin/
11. GLUSTER DOCS. Latest documentation [online]. [Date 19/10/2023]. Available at: https://docs.gluster.org/en/latest/
12. RED HAT. 3.3. Red Hat Gluster Storage 3.3 Administration Guide [online]. [Date 19/10/2023]. Available at: https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.3/html-single/administration_guide/index#sect-Managing_Block_Storage
13. MINIO. MinIO [online]. [Date 19/10/2023]. Available at: https://min.io/
14. CEPH. Architecture — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/architecture/
15. CEPH FOUNDATION. Ceph.io — About the foundation [online]. [Date 21/3/2024]. Available at: https://ceph.io/en/foundation/about/
16. CEPH. Red Hat's Ceph team is moving to IBM [online]. [Date 21/3/2024]. Available at: https://ceph.io/en/news/blog/2022/red-hats-ceph-team-is-moving-to-ibm/
17. CEPH AUTHORS AND CONTRIBUTORS. Ceph Releases (general) — Ceph Documentation [online]. [Date 21/3/2024]. Available at: https://docs.ceph.com/en/latest/releases/general/
18. HG INSIGHTS. Companies Using Ceph, Market Share, Customers and Competitors [online]. [Date 11/3/2024]. Available at: https://discovery.hgdata.com/product/ceph
19. Canonical Ltd. What is Ceph? | Ubuntu [online]. [Date 18/2/2024]. Available at: https://ubuntu.com/ceph/what-is-ceph
20. CEPH AUTHORS AND CONTRIBUTORS. Intro to Ceph — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/latest/start/intro/

21. CEPH. Monitor Config Reference — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/rados/configuration/mon-config-ref/
22. CEPH. Ceph Manager Daemon — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/mgr/
23. CEPH. OSD Service — Ceph Documentation [online]. [Date 21/2/2024]. Available at: https://docs.ceph.com/en/reef/cephadm/services/osd/
24. CICIMOV, Igor. Ceph cluster on Ubuntu-14.04 - DevOps [online]. [Date 21/1/2024]. Available at: https://icicimov.github.io/blog/storage/Ceph-cluster-on-Ubuntu-14.04/
25. CEPH. ceph-mds -- ceph metadata server daemon — Ceph Documentation [online]. [Date 16/2/2024]. Available at: https://docs.ceph.com/en/reef/man/8/ceph-mds/
26. CEPH. MDS Journaling — Ceph Documentation [online]. [Date 21/2/2024]. Available at: https://docs.ceph.com/en/reef/cephfs/mds-journaling/
27. WEIL, Sage, LEUNG, Andrew, BRANDT, Scott, MALTZAHN, Carlos, 2007. RADOS: A scalable, reliable storage service for petabyte-scale storage clusters. In: Proceedings of the 2nd International Workshop on Petascale Data Storage. pp. 35-44. [Date 22/2/2024]. DOI: 10.1145/1374596.1374606.
28. FLYDEAN. 04 Multi Paxos [online]. [Date 22/2/2024]. Available at: https://docs.flydean.com/www.flydean.com/distribution/04-multi-paxos
29. CEPH AUTHORS AND CONTRIBUTORS. Ceph Storage Cluster — Ceph Documentation [online]. [Date 22/2/2024]. Available at: https://docs.ceph.com/en/reef/rados/
30. CHUM, Sopanhapich, PARK, Heekwon and CHOI, Jongmoo, 2021. Supporting SLA via Adaptive Mapping and Heterogeneous Storage Devices in Ceph. Electronics. Vol. 10, p. 847. [Date 22/2/2024]. DOI: 10.3390/electronics10070847.
31. CEPH AUTHORS AND CONTRIBUTORS. Introduction to librados — Ceph Documentation [online]. [Date 23/2/2024]. Available at: https://docs.ceph.com/en/latest/rados/api/librados-intro/
32. CEPH AUTHORS AND CONTRIBUTORS. CRUSH Maps — Ceph Documentation [online]. [Date 19/2/2024]. Available at: https://docs.ceph.com/en/latest/rados/operations/crush-map/
33. CEPH AUTHORS AND CONTRIBUTORS. Pools — Ceph Documentation [online]. [Date 13/3/2024]. Available at: https://docs.ceph.com/en/latest/rados/operations/pools/
34. CEPH AUTHORS AND CONTRIBUTORS. Placement Groups — Ceph Documentation [online]. [Date 14/3/2024]. Available at: https://docs.ceph.com/en/reef/rados/operations/placement-groups/
35. D'ATRI, Anthony, Vaibhav BHAMBRE, and Karan SINGH. Learning Ceph - Second Edition. Birmingham: Packt Publishing, 2017. ISBN 978-1787127913.
36. CEPH AUTHORS AND CONTRIBUTORS. PG (Placement Group) notes — Ceph Documentation [online]. [Date 14/3/2024]. Available at: https://docs.ceph.com/en/latest/dev/placement-group/
37. CEPH DOCUMENTATION. Install [online]. [Date 14/11/2023]. Available at: https://docs.ceph.com/en/reef/install/
38. CEPH DOCUMENTATION. Hardware recommendations [online]. [Date 14/11/2023]. Available at: https://docs.ceph.com/en/reef/start/hardware-recommendations/
39. CEPH DOCUMENTATION. Get packages [online]. [Date 20/11/2023]. Available at: https://docs.ceph.com/en/reef/install/get-packages/
40. CEPH DOCUMENTATION. Install storage cluster [online]. [Date 14/11/2023]. Available at: https://docs.ceph.com/en/reef/install/install-storage-cluster/
41. CEPH DOCUMENTATION. Manual deployment [online]. [Date 26/11/2023]. Available at: https://docs.ceph.com/en/reef/install/manual-deployment/
42. CEPH DOCUMENTATION. Mgr administrator guide [online]. [Date 8/12/2023]. Available at: https://docs.ceph.com/en/reef/mgr/administrator/#mgr-administrator-guide

43. GITHUB. 2018. Issue comment on ceph-container [online]. [Date 10/1/2024]. Available at: https://github.com/ceph/ceph-container/issues/1158#issuecomment-412949409
44. OPENSTACK AUTHORS. OpenStack Charm Guide — charm-guide 0.0.1.dev804 documentation [online]. [Date 31/2/2024]. Available at: https://docs.openstack.org/charm-guide/latest/
45. OPENSTACK AUTHORS. OpenStack Charms Deployment Guide — charm-deployment-guide 0.0.1.dev519 documentation [online]. [Date 31/2/2024]. Available at: https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/
46. Canonical MAAS. MAAS installation technical reference (snap/2.8/CLI) - Docs - Canonical MAAS | Discourse [online]. [Date 3/3/2024]. Available at: https://discourse.maas.io/t/maas-installation-technical-reference-snap-2-8-cli/4523
47. REŠKO, Dominik, 2023. Návrh katalógu cloudových služieb pre akademické prostredie. Žilinská univerzita v Žiline.
48. CHARMHUB. Charmhub | The Open Operator Collection [online]. [Date 21/03/2024]. Available at: https://charmhub.io/
49. CEPH AUTHORS AND CONTRIBUTORS. Ceph iSCSI Gateway — Ceph Documentation [online]. [Date 21/3/2024]. Available at: https://docs.ceph.com/en/latest/rbd/iscsi-overview/
50. CEPH AUTHORS AND CONTRIBUTORS. PG Calc — Ceph Documentation [online]. [Date 25/3/2024]. Available at: https://docs.ceph.com/en/latest/rados/operations/pgcalc/
51. NFS-GANESHA PROJECT. nfs-ganesha.github.io [online]. [Date 26/3/2024]. Available at: https://nfs-ganesha.github.io/
52. Canonical Ltd. How-to guides | Ubuntu [online]. [Date 26/3/2024]. Available at: https://ubuntu.com/ceph/docs/howtos
53. OPENSTACK AUTHORS. Configure multiple-storage back ends — cinder 24.1.0.dev10 documentation [online]. [Date 27/3/2024]. Available at: https://docs.openstack.org/cinder/latest/admin/multi-backend.html
54. VINCHIN. How to Export and Import Image or Instance in OpenStack? | Vinchin Backup [online]. [Date 11/4/2024]. Available at: https://www.vinchin.com/vm-backup/openstack-export-import-image-instance-vm-snapshot-volume.html