PROJECT

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

V3 description of the selected solution meeting the established criteria

KPB1 – Design of the virt. solution

type – documentation

submission month – M7

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

1. Cloud Computing 5
1.1. Basic characteristics of CC 5
1.2. Service delivery model: Software as a Service (SaaS) 5
1.2.1. Who are the consumers? 5
1.2.2. What does the consumer get? 5
1.2.3. How are fees calculated? 5
1.2.4. Relationship to the layered model 6
1.2.5. Examples of suitable use 6
1.2.6. Examples of unsuitable use 6
1.2.7. Advantages 6
1.3. Service delivery model: Platform as a Service (PaaS) 7
1.3.1. Who are the consumers? 7
1.3.2. What does the consumer get? 7
1.3.3. How are fees calculated? 7
1.3.4. Relationship to the layered model 7
1.3.5. Examples of use 8
1.3.6. Advantages 8
1.4. Service delivery model: Infrastructure as a Service (IaaS) 8
1.4.1. Who are the consumers? 8
1.4.2. What does the consumer get? 8
1.4.3. How are fees calculated? 8
1.4.4. Relationship to the layered model 8
2. The OpenStack system 10
2.1. Introduction to OpenStack modules 10
2.1.1. KeyStone 10
2.1.2. Nova 10
2.1.3. Neutron 10
2.1.4. Horizon 11
2.1.5. Skyline 11
2.1.6. Glance 11
2.1.7. Swift 11
2.1.8. Cinder 11
2.1.9. Manila 11
2.2. Networking in OpenStack – the Neutron module 11
2.2.1. Switching 11
2.2.2. Routing 12
2.2.3. Load balancing 12
2.2.4. Firewalling 12
2.3. Network architecture of an OpenStack cloud 12
2.3.1. Network zones 12
2.3.2. Network layer options 13

2.4. Network types....14
2.5. Tenant network....15
2.6. Modular Layer 2 (ML2) plugin....15
2.7. OpenVSwitch (OVS)....15
2.8. OpenStack service nodes....16
2.8.1. Controller node....16
2.8.2. Compute node....16
2.9. High availability of the OpenStack system....16
2.9.1. RabbitMQ....16
2.9.2. MariaDB – Galera cluster....17
2.9.3. Keepalived....17
3. Automated production deployment of the OpenStack platform....18
3.1. Tools for automated deployment of OpenStack....19
3.2. OpenStack Charms....20
3.2.1. Metal-as-a-Service(MAAS)....20
3.3. The Juju component....21
4. Cloud management platform....22
4.1. ManageIQ....22
4.1.1. ManageIQ architecture....23
4.1.2. Platform requirements....24
4.2. ManageIQ centralized authentication....25
4.2.1. Configuration of authentication in the ManageIQ environment....25
4.2.2. Installation and configuration of SSSD....25
4.2.3. Testing the configuration....26
4.2.4. Integration with Apache and the ManageIQ UI....26
5. Distributed storage....26
5.1. Principles and advantages of distributed storage....27
5.2. Use of distributed storage....28
5.3. Reasons for deploying distributed storage....28
6. Selection of distributed storage software for OpenStack....29
6.1. Criteria for software selection....29
6.2. Available options....30
6.3. Software selection....30
7. Ceph And its architecture....32
7.1. Key components of the Ceph software....32
7.1.1. Ceph cluster....32
7.1.2. Ceph Monitor....33
7.1.3. Ceph Manager....34
7.1.4. OSD daemons....34
7.1.5. MDS and Ceph File System....35
7.2. RADOS....36
7.3. The CRUSH algorithm....37
7.4. Pools and PGs....38

7.4.1. Pool 39
7.4.2. PG 39
8. IMPLEMENTATION OF A SHARED FILE SYSTEM IN THE OPENSTACK ENVIRONMENT 41
8.1. Storage in the OpenStack platform environment 41
8.2. Manila as a service for managing shared network file systems in the OpenStack environment 41
8.3. Objectives and advantages of using the Manila service 43
8.4. Key components of the Manila service architecture 43
8.5. Interaction of the Manila service components 44
9. Monitoring in the cloud computing environment 46
9.1. Motivation for monitoring 46
9.1.1. Capacity and resource planning 46
9.1.2. Capacity and resource management 46
9.1.3. Data centre management 47
9.1.4. SLA management 47
9.1.5. Billing 47
9.1.6. Problem detection 47
9.1.7. Performance management 47
9.1.8. Security 48
9.2. Perspectives on monitoring 48
List of references 50

1. CLOUD COMPUTING

Document [26] defines cloud computing (CC) as a system model in which the provider enables the consumer convenient, ubiquitous, on-demand access to shared configurable computing power (networks, servers, storage, applications and services). This computing power can be provisioned or released with minimal intervention by the provider.

1.1. Basic characteristics of CC

Document [26] defines the basic characteristics of CC in more detail:

- **On-demand self-service:** the consumer can automatically be provided with the computing power they need without human interaction with the service provider.
- **High network availability:** resources are available over the network and can be accessed from various standard devices such as PCs, mobile phones and tablets.
- **Resource pooling:** the provider's computing power is shared among multiple consumers. Physical and virtual resources are dynamically assigned and reassigned according to their requirements. Thus the consumer does not know the exact location where the resources assigned to them are situated (at a higher level of abstraction, the location can be narrowed down, e.g. to a country or a data centre). Examples of such resources are storage space, CPU, memory and network bandwidth.
- **Rapid elasticity:** resources are elastically allocated and released, in some cases automatically, in order to adapt quickly to the corresponding demand. To the consumer, the resources thus appear to be unlimited and can be used in any quantity at any time.
- **Measured service:** The cloud system automatically controls and optimizes resources by implementing metering at some level of abstraction appropriate to the given type of monitoring (e.g. storage space or active user accounts). Resource usage can be monitored, controlled and summarized in reports, thereby providing transparency for both the provider and the consumer of the given service.

1.2. Service delivery model: Software as a Service (SaaS)

The service for the consumer consists of applications accessible over the internet, provided by the provider. They are accessible from various consumer devices, e.g. via a web browser or a programming interface.

1.2.1. Who are the consumers?

- Organizations that provide their employees with access to typical office applications (spreadsheet, e-mail, etc.).
- End users who use the provided software directly.
- Application administrators who configure the applications for end customers.

1.2.2. What does the consumer get?

The right to use a specific application on demand and data management, such as data backup and sharing between consumers.

1.2.3. How are fees calculated?

They are usually based on the number of users, the time the application is used, the number of uses, launches or processed records; the bandwidth used and the amount of data stored.

1.2.4. Relationship to the layered model

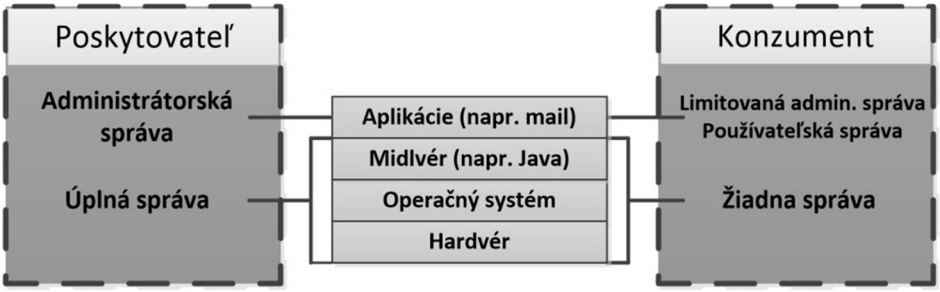


Fig. 7 - SaaS – relationship to the layered model

The consumer pays for access to the application but neither manages nor controls the lower layers of the model (network, servers, operating system, storage space, nor some settings of the application used):

In the SaaS model, the consumer has access to the resources allocated to them by the provider's SaaS application. For example, e-mail, where the consumer can create, send and store mail. If the application allows it, the consumer has the option of limited administrative control (creating accounts for other consumers, an overview of activities, etc.).

The provider is the administrator with all rights to the SaaS application. Its task is the configuration, updating and provision of the required services.

1.2.5. Examples of suitable use

- **business activities –** connecting merchants with suppliers or customers (mail, invoicing, inventory management),
- **collaboration –** applications for team collaboration (screen and document sharing; calendar, online gaming),
- **office productivity –** applications such as text editors, spreadsheets, presentation programs and databases (they extend programs with collaboration capabilities),
- **software tools –** applications for security management, applications for web development, etc.

1.2.6. Examples of unsuitable use

- **real-time software –** applications for aviation, control of industrial robots, etc. (data transmission via the cloud cannot guarantee a sufficiently fast response),
- **transfer of huge data volumes –** applications in medicine or physics where huge amounts of data are generated, which may not be advantageous to transfer via the cloud,
- **critical software –** applications where an error may mean damage to property or loss of life (response problem).

1.2.7. Advantages

- **Simplicity –** the consumer's web browser is efficient enough to display the required data, so there is no need for additional applications. Therefore no software installation is required, and only a minimum of application-related data is thus located on the consumer's PC.
- **Efficient use of licences –** one licence can be used by multiple consumers at the same time. This is made possible by the fact that the software is on the provider's side and the provider includes the consumer's use of licensed products in the price of the

services provided.

- **Centralized management and data –** from the consumer's point of view, all data appear to be centralized, since they are located at the provider (in reality they are decentralized due to redundancy, etc.). Logical centralization guarantees the consumer protection against data loss and security, and makes it possible to perform backups, etc.
- **Responsibility for the platform is assumed by the provider –** the consumer does not participate in the management of the lower layers.
- **Financial savings –** the provider supplies all the hardware, power supply, equipment maintenance, etc., which should ultimately cost the consumer less money (even including the fees) than if they had to provide it themselves.

1.3. Service delivery model: Platform as a Service (PaaS)

The service for the consumer is the possibility of placing their own software in the cloud environment or the possibility of using programming languages, libraries and tools provided by the provider.

1.3.1. Who are the consumers?

- Application developers.
- Application testers who test the application in various environments.
- Programmers who deploy new applications (or their updates) to the cloud environment.
- Administrators who configure, tune and monitor application performance.
- End users who use the application (for them, access to the application is the same as in the SaaS model).

1.3.2. What does the consumer get?

The possibility of using the provider's tools and resources for running, developing, testing, deploying and administering applications.

1.3.3. How are fees calculated?

They are usually based on the number of users, their type (e.g. developer/end user), storage size, computing power used, the bandwidth used by the platform and the length of time it is used.

1.3.4. Relationship to the layered model

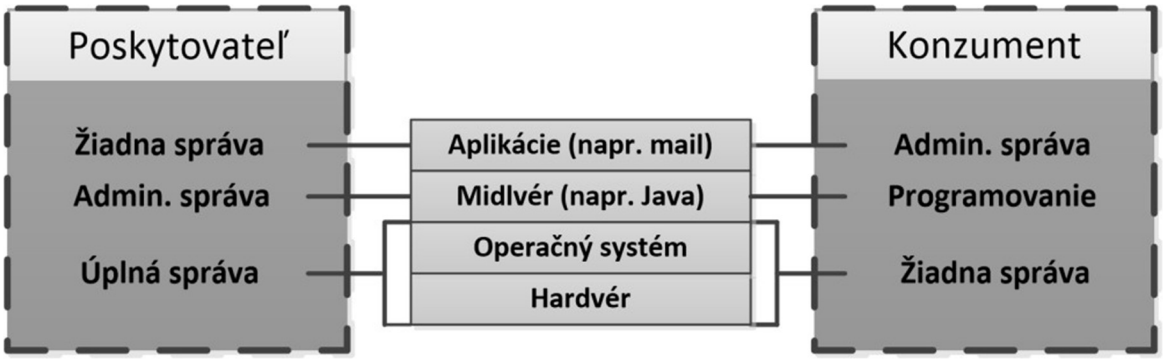


Fig. 8 - PaaS – relationship to the layered model

The consumer neither manages nor controls the lower layers of the model (network, servers, operating system, storage space), but has the ability to control and configure applications.

In the PaaS model, the lowest layers of the model are managed by the provider (operating system, hardware, network in the LAN and between data centres). It provides the consumer with an environment for running and programming their applications, as well as CPU performance, memory, storage space, databases, network connectivity, etc. The provider determines the programming model, i.e. the circumstances under which the consumer will be able to program their own applications (for the purpose of charging). The consumer has full control over their own applications.

The PaaS model provides space for the development, running and administration of software that is intended for a large number of consumers, processes large amounts of data and will potentially be reachable from any location via the internet.

PaaS is very similar to a traditional computer system. Unlike it, applications in the PaaS model can use as much computing power as they need, process large amounts of data, be deployed almost instantly and serve a large number of users.

1.3.5. Examples of use

The PaaS model environment can be used to develop applications providing the services mentioned for the SaaS model.

1.3.6. Advantages

The consumer is freed from the obligation to select, install, maintain or control the platform components. It has many advantages in common with the SaaS model (*simplicity, centralized management and data, responsibility for the platform is assumed by the provider, financial savings*).

1.4. Service delivery model: Infrastructure as a Service (IaaS)

The service for the consumer is the provision of computing power, storage space, network and other fundamental resources so that the consumer can place and run arbitrary software in the cloud system (including the operating system and applications).

1.4.1. Who are the consumers?

System administrators.

1.4.2. What does the consumer get?

Access to a virtual computer, storage space available over the network, network elements such as a firewall, and the ability to configure services.

1.4.3. How are fees calculated?

Usually CPU usage/hour, stored data/hour, consumed bandwidth, use of network infrastructure (e.g. IP addresses)/hour and value-added services (e.g. monitoring, automatic scalability, etc.).

1.4.4. Relationship to the layered model

In the IaaS model, the operating system layer is divided into two parts. A lower (and more privileged) layer is added, where the Virtual Machine Monitor (VMM), called the hypervisor, runs. It is used for the cooperation of multiple Virtual Machines (VMs), where a VM appears to the consumer as a physical computer that can be configured over the network. The operating system that is installed in the VM is called the guest operating system.



Fig. 9 - IaaS – relationship to the layered model

The provider controls the lowest layers of the model (physical hardware and hypervisor). The consumer can submit a request to create a new VM. This request is granted only if it complies with the provider's policy. Through the hypervisor, the provider can provide the consumer with virtual network elements so that the consumer can create and configure their own virtual network within the provider's infrastructure. However, the provider continues to remain responsible for managing, updating and configuring the infrastructure for the purposes of security and reliability.

In the IaaS model, the consumer has full control over the guest operating system in each VM and over all the layers above it.

2. THE OPENSTACK SYSTEM

2.1. Introduction to OpenStack modules

OpenStack is a cloud operating system that works by means of modules. Each module has an assigned function or area that it manages. These modules communicate with each other via APIs. A new version of OpenStack is released approximately every six months. With each new version, a number of these modules are added, discontinued, or have their functionality modified/extended. In the latest version (2025.1), there are around 60 of these modules.

The most important modules, essential for the functioning of OpenStack, include: KeyStone, Nova, Neutron, Horizon (or, in the latest version, Skyline), Glance, Swift, Cinder, Manila.

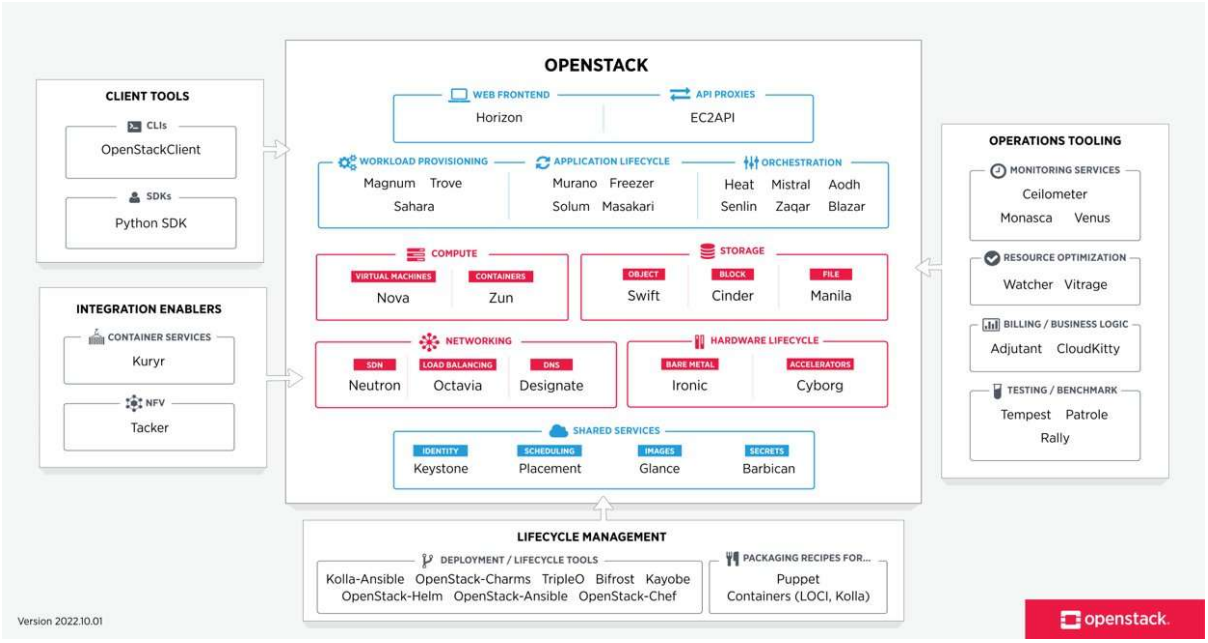


Figure 2.1 Source: (Techtarget, 2025)

2.1.1. KeyStone

Keystone is an OpenStack module that provides API client authentication, service discovery and distributed multi-tenant authorization by implementing the OpenStack Identity API. It supports LDAP, OAuth, OpenID Connect, SAML and SQL.

2.1.2. Nova

The Nova module implements services and associated libraries that provide massively scalable, on-demand, self-service access to compute resources, including bare metal, virtual machines and containers.

2.1.3. Neutron

OpenStack Neutron is an SDN (Software-Defined Networking) networking project focused on delivering networking-as-a-service (NaaS) in virtual compute environments.

2.1.4. Horizon

Horizon provides the dashboard service – the OpenStack web interface, which is extensible and provides a web-based user interface for the individual OpenStack services.

2.1.5. Skyline

Skyline is an optimized OpenStack dashboard. It has a modern and technologically advanced interface. It is easier for developers to maintain and for users to operate. It provides higher concurrency performance.

2.1.6. Glance

Glance is a module that handles the management of virtual machine (VM) images. Within VM image management, images can be discovered and stored in various locations, ranging from simple file systems to object storage systems such as the OpenStack module Swift.

2.1.7. Swift

The OpenStack Object Store project, known as Swift, offers cloud storage software so that large amounts of data can be stored and retrieved with a simple API. It is built and optimized for durability, availability and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without limit.

2.1.8. Cinder

The Cinder module is intended for managing OpenStack block storage. It virtualizes the management of block storage devices and provides end users with a self-service API to request and consume these resources without requiring any knowledge of where their storage is actually deployed or on what type of device. This is done by means of a reference implementation of LVM (Logical Volume Management) or plugin drivers for other storage.

2.1.9. Manila

Manila is a module that provides coordinated access to shared or distributed file systems.

2.2. Networking in OpenStack – the Neutron module

OpenStack networking is a scalable API-based system that makes it possible to manage networks and IP addresses in an OpenStack cloud. Like other key components, it can be a tool for administrators and users to maximize the utilization of the existing physical resources of the cloud infrastructure. Neutron is the component that implements the SDN solution. It gives tenants full control over the creation of virtual network resources. This is often achieved by means of tunnelling protocols that create encapsulated communication paths over the existing network infrastructure in order to segment tenant traffic. This method varies depending on the specific implementation, but some of the most common methods include GRE tunnelling, VXLAN encapsulation and VLAN tagging [4].

Neutron is a standalone service that can be installed independently of the other services. It includes many technologies that could be found in a data centre, including switching, routing, load balancing, firewalling and virtual private networks.

2.2.1. Switching

A virtual switch is defined as a software application that connects virtual machines to virtual networks at the second layer, or data link layer, of the OSI model. Neutron supports multiple switching platforms, including Linux bridges, provided by the kernel bridge module, as well as the well-known software switch OVS (Open vSwitch). OVS is an open-source virtual

switch that supports standard interfaces and protocols such as NetFlow, SPAN, RSPAN, LACP and 802.1q VLAN tagging. In addition to VLAN tagging, users can create overlay networks using L2-in-L3 tunnelling protocols such as GRE or VXLAN. OVS can be used to facilitate communication between instances and devices outside OpenStack, which include hardware switches, firewalls, storage devices and dedicated servers [3].

2.2.2. Routing

Neutron provides routing and NAT capabilities by utilizing IP forwarding, iptables and network namespaces. In a network namespace we can find the sockets, bound ports and interfaces that were created in that namespace. Each namespace has its own routing table, interfaces and iptables, which provide filtering and network address translation. Thanks to the division of networks into different namespaces, there are no concerns about overlapping subnets between different users. The router configuration within Neutron allows instances to communicate between cloud and external networks. Namespaces are also used by advanced network services such as “Firewall as a service” and “Virtual Private Network as a service” (Denton, 2015).

2.2.3. Load balancing

Load balancing provides users with the ability to distribute client requests across multiple instances or servers. Users can create monitors, set connection limits and apply persistence profiles to traffic passing through the virtual load balancer [3].

2.2.4. Firewalling

OpenStack offers two ways of securing instances: security groups and firewalls. When security groups are used, instances are placed into groups that share common functions and rule sets. The rules in iptables are configured on the compute nodes and filter traffic to and from the Linux bridges connected to each instance. When a virtual firewall such as FWaaS is used, traffic is filtered at the network edge on the Neutron router rather than on the compute node [3].

Virtual private networks, or VPNs, extend a private network across a public network such as the Internet. A VPN allows a computer to send and receive data over public networks as if it were directly connected to the private network. Neutron provides a set of APIs that allow users to create IPsec-based VPN tunnels from routers to remote gateways [4].

2.3. Network architecture of the OpenStack cloud

OpenStack provides a rich networking environment. In this section we describe the requirements and options that need to be taken into account when designing a private cloud. These include recommendations on network implementations to be considered, and information on some OpenStack network layouts and on network services that are essential for stable operation. The cloud environment fundamentally changes the ways in which networks are provided and used. When deciding on the architecture, it is essential to understand the following concepts and decisions.

2.3.1. Network zones

Cloud networks are divided into several logical zones. It is recommended to define at least four distinct network zones.

OpenStack Underlay network – The underlay is defined as the physical network switching infrastructure that connects the storage, compute and control platforms. There is a large number of potential underlay network options.

OpenStack Overlay network – The overlay is defined as any L3 connectivity between cloud

components and may take the form of SDN solutions, such as the Neutron overlay solution or third-party SDN solutions.

Edge – The edge zone is where network traffic transitions from the cloud overlay or SDN networks into traditional network environments.

External – The external zone is defined as the configuration and components needed to provide access to cloud resources. External is defined as all components outside the cloud edge gateways [4].

Data flow

Within a cloud infrastructure there are two primary types of traffic flow; the choice of network technologies is influenced by the expected cloud load:

- **East/West** – Internal data flow between virtual devices in the cloud and data flow between compute nodes and storage nodes. This is usually the largest flow and, because of the need to serve storage, a minimum of hops and low latency must be ensured.
- **North/South** – Data flow between virtual devices and all external networks, including clients and remote services. This data flow depends heavily on the workload in the cloud and on the type of network services offered [4]

2.3.2. Network layer options

When deciding whether to use a layer 2 network architecture or a layer 3 network architecture, several factors need to be taken into account.

Benefits of the L2 layer

There are several reasons why a network designed on L2 protocols is preferred over an L3 network.

- Ethernet frames contain everything essential for networking. This includes, among other things, globally unique source addresses, globally unique destination addresses and error checking.
- Ethernet frames can carry any kind of packet. A layer 2 network is independent of the layer 3 protocol.
- You can add supplementary network functions to an Ethernet network, for example class of service (CoS) or multicasting, as easily as to IP networks.
- VLANs are an easy mechanism for isolating networks.
- Speed
- Reduced overhead of the IP hierarchy.
- When systems move, there is no need to track or change the address configuration.

While the simplicity of L2 protocols may work well in a data centre with hundreds of physical machines, cloud data centres carry the additional burden of having to track all the addresses and networks of virtual machines. In these data centres it is not unusual for a single physical node to support 30 to 40 virtual instances [4].

Limitations of the L2 layer

L2 network architectures have certain limitations that become noticeable when they are used outside traditional data centres.

- The number of VLANs is limited to 4096.
- The number of MAC addresses stored in switching tables is limited.

- Troubleshooting a network without IP addresses and ICMP can be difficult.
- ARP configuration and operation can be complicated on large L2 networks.
- Migration of MAC addresses (instance migration) to other physical locations is a potential problem if you do not set the ARP table timeouts correctly.

The L2 layer has limited network management tools. It is difficult to control traffic because it has no mechanisms for network management or traffic shaping. Troubleshooting the network is also unpleasant, partly because the network devices have no IP addresses. There is essentially no reasonable way to check network latency [4].

Benefits of the L3 layer

The advantages of using an L3 network architecture are as follows:

- L3 networks provide the same level of resilience and scalability as the Internet
- Controlling network traffic using routing metrics is simple
- The possibility of configuring a BGP confederation for scalability. As a result, they will have state proportional to the number of racks, not to the number of servers or instances.
- The existence of a whole range of well-tested tools, for example the ICMP protocol, for monitoring and traffic management.
- L3 architectures allow QoS to be used to manage network performance [4].

Limitations of the L3 layer

The main limitation of L3 networks is that there is no built-in isolation mechanism comparable to VLANs at the L2 layer. The hierarchical nature of IP addresses means that an instance is in the same subnet as its physical host, which makes migration difficult. For these reasons, network virtualization must use IP encapsulation and a software solution on the end hosts, which serves to isolate and separate the addressing in the virtual layer from the addressing in the physical layer. Other potential disadvantages of L3 networking include the need to design an IP addressing scheme instead of relying on switches to track MAC addresses automatically, and the configuration of an interior gateway routing protocol in the switches [4].

2.4. Network types

The selected server hardware must have a suitable number and type of network ports to support the proposed architecture. Although each OpenStack installation is specific to its use, the following networks are recommended:

Internal or management network – Used as the internal communication network between compute and controller nodes. It can also be used for iSCSI communication between compute nodes and iSCSI storage nodes. It should be at least a 1 GB NIC and should be a non-routed network. This interface should be redundant for high availability (HA).

Tenant network (tenant) – a private network that enables communication between the instances of each tenant. This network should also be isolated from all other networks for security reasons. For HA, a redundant 1Gb interface should be sufficient. In OpenStack, the term tenant is an older designation for a project in the Keystone service. Since API version 3, project is the preferred term. More on the tenant network and its types in Chapter 6.4.

Storage network – a private network that can be connected to a Ceph front end or other shared storage. For HA, it should be a redundant configuration with the recommended 10Gb

networks. This network isolates instance storage from other networks. Under load, this storage traffic could overload other networks and cause outages in other services.

External or public network – This network is used for external communication from virtual machines to the public network space. These addresses are usually handled by the Neutron agent on the controller nodes, but they can also be handled by another SDN. However, if you use Neutron DVR with OVS, this network must be present on the compute node, because north–south traffic will be handled not by the controller nodes but by the compute node itself [4].

2.5. Tenant network

It is created by users for interconnection within projects. By default, such networks are completely isolated and are not shared with other projects. OpenStack supports several types of these networks.

- Flat – all instances are located in the same network, which can be shared with the host. There is no VLAN tagging or other network segregation.
- VLAN – support for creating multiple networks using VLAN IDs that correspond to VLANs present in the physical network.
- VXLAN – uses network overlays to support private communication between instances. A router is required to enable traffic outside the tenant network and to external networks, including the Internet. The router provides the ability to connect to instances directly from the external network using floating IPs [4].

2.6. Modular Layer 2 (ML2) plugin

The modular layer 2 (ml2) plugin is an OpenStack Neutron plugin that allows the L2 technologies found in complex data centres to be utilized. It currently supports existing L2 agents such as openvswitch and linuxbridge.

2.7. OpenVSwitch (OVS)

Open vSwitch (OVS) is an open-source switch implementation that runs on Linux. It makes it possible to create multiple switches, VLANs, interfaces and virtual machines. One of the most important features of OVS is that it is a distributed switch. This means that we can have several virtual machines on different hypervisors and OVS will allow them to be interconnected.

To connect virtual machines to the physical network, OVS uses three bridges. It introduces additional virtual interfaces inside the host, identified as bridges and classified as follows:

- **br-int**: Represents the integration bridge. All virtual instances that use the network service connect to this bridge. These include virtual instances, routers and the DHCP server.
- **br-tun**: In the case of tunnelled networks, br-ethX is replaced by br-tun, which handles the encapsulation and decapsulation of packets.
- **br-ex**: This bridge is connected to the physical interface that provides connectivity to the outside world.

Virtual machines connect to the integration bridge using VLAN segmentation. The VLAN IDs used to connect a VM to the integration bridge are local to the compute node. If the destination VM is on the same compute node, the integration bridge switches the packet locally

to the port of the destination VM. If the destination port is on another node, the packet is forwarded to the br-tun bridge via a patch port. Br-tun encapsulates the L2 packet in VXLAN and replaces the local VLAN with the VXLAN tunnel ID, which is allocated by the ML2 VXLAN driver for the virtual network. On reaching the destination node, br-tun decapsulates the header, replaces the VXLAN ID with the local VLAN ID and passes it to br-int, which delivers the packet to the destination port [4].

2.8. OpenStack service nodes

The overall concept of the OpenStack service consists of a controller node (called controller), compute nodes (called compute) and storage nodes. Although OpenStack is designed to be deployed on multiple physical machines, this is not always necessary. The OpenStack system can be deployed on just one server, which will serve as a common physical server for all nodes of the OpenStack service.

2.8.1. Controller node

This node hosts the communication services that support the operation of the entire cloud, including the databases, the Horizon Dashboard and possibly the monitoring system. This node can optionally host the Nova scheduler service and server load. To ensure redundancy, there must be at least two controller nodes in the cluster [63]

2.8.2. Compute node

This node hosts the hypervisor and the virtual instances and provides them with compute resources. A compute node can also serve as a network controller for the instances it hosts if a multi-host network scheme is used. It can also host lightweight internal OpenStack services [63].

2.9. High availability of the OpenStack system

One way of achieving a higher level of high availability is to duplicate the controller node. This step eliminates the SPOF, as the controller node services will remain available even if one of the controller nodes stops working. These two servers will be operated as a single common cluster, which will run, where possible, in an active-active configuration, and the server load will be shared evenly between both nodes.

When creating a controller cluster for the OpenStack service, in addition to the basic programs for running the OpenStack service, it is necessary to install further programs that are used for the clustering and redundancy of the entire system. The complete installation and configuration procedure is attached as an annex to this work. The annex describes in detail the procedure for deploying a complete OpenStack controller-compute topology, together with the installation and configuration of the supporting programs for clustering the controller node.

The subsections below describe the programs and technologies needed to run basic OpenStack, as well as the programs that can be used to cluster the controller nodes.

2.9.1. RabbitMQ

RabbitMQ is open-source message broker software (sometimes also called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support the Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT) and other protocols.

RabbitMQ is written in the Erlang programming language and is built on the Open Telecom

Platform for clustering and failover. Client libraries are available for all major programming languages.

2.9.2. MariaDB – Galera cluster

MariaDB Galera Cluster is a virtually synchronous multi-master cluster for MariaDB. It is available only on Linux and supports only the InnoDB storage engine. OpenStack uses the MariaDB database service in essentially all its components, and each module installed on the controller node will have its own database. MariaDB is installed on all controller nodes. Features:

- Synchronous replication
- Active-active multi-master topology
- Read from and write to any cluster node
- Automatic membership control, failed nodes dropped from the cluster
- Automatic node joining
- Parallel replication at row level
- Direct client connections, native MariaDB look and feel

2.9.3. Keepalived

A high-availability cluster usually consists of two servers: an active primary server and a standby secondary server. These two servers share the same VIP (virtual IP address), which is valid only for the primary server. When the primary server fails, the secondary server takes over the VIP so that it can continue providing services. This mode is widely used in MySQL source/replica switching and in Ngnix web access.

Keepalived is high-availability software based on VRRP (Virtual Router Redundancy Protocol) that can be used to build a high-availability cluster. In traditional physical networks, the primary/secondary state can be negotiated with the VRRP protocol of Keepalived. The primary device periodically sends ARP messages to flush the MAC table or the terminal ARP table of the uplink switch in order to trigger the migration of the VIP to the primary device [64].

3. AUTOMATED PRODUCTION DEPLOYMENT OF THE OPENSTACK PLATFORM

An environment becomes a production environment on the day it is deployed into live operation, in most cases on the day of transition from the test environment. In case the above term “live operation” is unclear, it refers to the state in which an environment developed by us or by another company is deployed after testing and either is assigned some function that it performs, or is provided directly to customers or employees for use, or is assigned another purpose, often a commercial one. In our case, such an environment is precisely the cloud environment, more specifically OpenStack. The cloud environment in the OpenStack distribution, as already mentioned, offers 2 approaches, or purposes, for its creation. In the cloud world these are known as: Private cloud and Public cloud.

The production environment in the first case, namely when a private cloud is created, is mostly used for private purposes, which also follows from its name private (private, non-public). Private clouds have diverse uses and in most cases we do not pay for their provision, nor do we require any financial contributions from clients. Companies and enterprises use such environments for internal systems, infrastructure, teaching and the like.

On the other hand, the public cloud serves, as its name suggests, the public. In most cases, such an environment is created for a commercial purpose, to provide companies and enterprises with an entire environment or only its resources. A very good example of such an environment, as already mentioned, is Amazon Web Services. AWS provides its environment to the public for a certain fee, from which the customer benefits because it does not have to take care of the availability of all the resources. For example, company xy wants to create or migrate its entire network infrastructure to the cloud. If it had to create a private cloud and then operate or migrate the network infrastructure on or into it, this would be very time-consuming and costly, because they would have to build 2 environments from scratch. And this is exactly the moment where the use of a public cloud excels. The public cloud provider provides the company with an environment (virtual servers, etc.) for a certain fee, in most cases an hourly one, and the company can start the migration without worries. However, if we as a company are the creator and provider of a public cloud, then this is also one of our tasks: to provide resources, a cloud environment, to the public.

OpenStack in production deployment is becoming more popular thanks to its strong support and maintenance and the fact that it is a free open-source project. One of its very first production environments was created at an organization that was also directly involved in its development, namely NASA [18]. Its effort to standardize its websites inspired a breakthrough in cloud computing technology. Eventually, owing to the complexity of their task, they arrived at the topic of cloud computing and the Nebula project was created. It is precisely the members of the Nebula project who are behind the creation of the cloud computing environment called OpenStack.

Nowadays a considerable part of NASA's resources already runs in the AWS cloud environment, but OpenStack is still used at the NASA Center for Climate Simulations (NCCS). They use Ansible for its deployment and management. This environment consists of 9000 cores and 322 teraflops of computing power [19].

It may also come as a surprise that, apart from NASA, other government organizations such as the United States civil service or the French Ministry of the Interior also use production deployments of OpenStack.

The well-known companies that already operate or use an OpenStack cloud today are countless. Such well-known foreign companies include: Cert, Wallmart, NASA, Yahoo, Target, T-mobile, Nike and many others [19].

Czech and Slovak companies that operate an OpenStack cloud include seznam.cz [20]. Their OpenStack environment runs in three data centres consisting of 12 thousand physical servers.

3.1. Tools for automated deployment of OpenStack

Installing OpenStack differs from installing a program on a device; it is a much more complex matter, and therefore the installation makes use of tools and so-called frameworks that help maintain the life cycle of the deployed OpenStack. A framework is a structure that serves as a foundation for building software, and its advantage is that one does not start completely from zero (from scratch). Several frameworks currently exist [21]:

TripleO

TripleO is the name for “OpenStack on OpenStack”. It is an official OpenStack project aimed at enabling the deployment and management of a production cloud on physical hardware using existing OpenStack components. With TripleO, one starts by creating an “undercloud”, which will contain the OpenStack components needed to deploy and manage the “overcloud”, i.e. the working cloud. The overcloud is the deployed solution and can represent a cloud for any purpose. TripleO uses several existing core OpenStack components such as Nova, Ironic, Neutron, Heat, Glance and Ceilometer.

OpenStack-Helm

OpenStack-Helm is a solution for deploying OpenStack with Helm using Kubernetes containers. Helm is a tool used to define, install and upgrade applications running on Kubernetes. In its most basic form, Helm is a templating tool. Kubernetes is used to automate the operational tasks of container management and contains built-in commands for deploying applications, rolling out changes to applications and scaling applications, either up or down. To deploy a Kubernetes cluster for a production environment, it is recommended to use a highly resilient distribution such as Airship or KubeADM.

Kolla-Anisble

Kolla-Ansible deploys a containerized OpenStack control plane using Kolla containers, orchestrated via Ansible. The project focuses on simplicity and reliability while providing a flexible and intuitive configuration model. Kolla provides containers and tools for operating OpenStack clouds that are scalable, fast, reliable and upgradable using community best practices. Ansible is a software tool that provides simple but powerful automation for cross-platform computer support.

Bifrost

Bifrost is a set of Ansible playbooks that uses the Ironic component to automate the task of deploying a base image onto hardware. It provides a modular tool for one-off operating system deployment with as few operational requirements as possible.

Kayobe

Kayobe deploys a containerized OpenStack control plane onto physical devices. Bifrost is used to discover and provision cloud servers. Kolla is used to build container images for OpenStack services. Kolla Ansible is used to deploy the containerized OpenStack control plane.

Openstack-Ansible

OpenStack-Ansible provides Ansible playbooks and roles for deploying and configuring an OpenStack environment. Thus, in contrast to a Kolla-Ansible deployment, we will not use containers, but only the Ansible tool and OpenStack components.

OpenStack-Charms

Openstack-Charms is a collection of Charmed Operators, also simply called “charms”, which are used to deploy and manage OpenStack clouds using MAAS and Juju. Each OpenStack charm is responsible for the deployment and life-cycle management of a single cloud service (for example, the nova-compute charm for the Nova Compute service). The project also includes charms for selected software not supported by OpenStack, such as Ceph, MySQL and RabbitMQ.

OpenStack-Chef

Chef is an open-source platform for systems management and cloud infrastructure automation. A deployment of OpenStack using Chef should contain “cookbooks”.

3.2. OpenStack Charms

For the installation of OpenStack itself, OpenStack-Charms was chosen as the most suitable method; Charmed Operators, also called “Charms”, are used to deploy and manage OpenStack clouds using Metal-as-a-Service (MAAS) and Juju. OpenStack Charms was created by Canonical, the company behind the development of the Ubuntu Linux operating system, which is also directly involved in the development of the OpenStack project itself, so this deployment solution is supported and recommended by OpenStack itself as well. Even on the official OpenStack website, most guides are built and explained directly for this type of deployment, which makes it relatively easy to find solutions to problems that arise. This type of deployment is also recommended by the community, and in case of questions everyone is willing to help, whether on the forums and pages of the OpenStack project itself, or we can find help, guides and documentation for the individual parts (MAAS [21],[24] and Juju [25]) on the Canonical Ubuntu forums and pages. OpenStack Charms names the individual modules Charms, or in translation “spells”. These spells are stored in CharmHub, which we can imagine as an online shop, with the only difference being that everything there is free. On CharmHub [22] we can find all OpenStack modules as well as a number of other applications that are not implemented directly in the OpenStack project, such as Nginx, Grafana, Prometheus, Dokuwiki and many others. Thanks to the simple approach that OpenStack Charms offers, we are able to deploy these additional applications and services into the running environment either with a single comprehensive command or via the Juju web interface.

OpenStack Charms, with everything it brings, simplifies the whole process of deploying an entire cloud environment in the OpenStack distribution, but it likewise cannot do without knowledge and an analysis of the options for how it works

3.2.1. Metal-as-a-Service(MAAS)

Metal-as-a-Service (MAAS) provides complete automation of physical servers for efficient server/data centre operation. The software is open-source and is backed by Canonical, the company behind the Ubuntu Linux distribution. MAAS treats physical servers like virtual machines (instances) in the cloud. This solution makes it possible to manage several servers at once and from a single place, so that they do not have to be managed individually and in a complicated way. It is thus possible to deploy standard or customized operating systems onto physical or virtual machines remotely. MAAS comprehensively meets the need for rapid deployment, destruction and reconfiguration of the operating system.

It is currently deployed in banking, telecommunications and industrial environments, as well as for specialized purposes such as front-end supercomputer validation, music streaming services, disaster recovery and computer security risk analysis. The main key function of the MAAS server for our purposes will be automation, whereby MAAS uses the Intelligent Platform Management Interface (IPMI) to power on/restart the given machine and uses the Preboot eXecution Environment (PXE), over either an IPv4 or IPv6 network, to install the required or default operating system for this task. Other useful functions include the configuration of storage, network interfaces and IP address management using the Dynamic Host Configuration Protocol (DHCP) [24].

3.3. The Juju component

Juju is an open-source framework that uses Charmed Operators, or “Charms”, to deploy cloud infrastructure and applications and to manage their operations from start-up. With Juju, it is possible to install, maintain, upgrade and integrate applications across clusters, Kubernetes containers, and virtual and physical machines on public or private clouds. The vendors describe Juju as software that manages your software. It helps you take control of all your applications, infrastructure and environments. Using this software should ensure:

- saving the time spent on managing scripts
- minimizing costs
- ensuring redundancy and resilience
- monitoring all activities across the infrastructure
- maximizing the hybrid cloud architecture

Charms are small applications that contain common sets of function instructions in order to ensure repeatable and reliable code. This makes it possible to manage applications and scenarios instead of configurations, although, if necessary, there is still the option of diving into the YAML (YAML Ain’t Markup Language) configuration files. Charmed Operators are used not only to deploy and manage individual applications, but also to connect them with one another in models. The application model defines which applications provide a service and how they relate to each other. Juju can be controlled via the command line (CLI) or via a graphical web interface (GUI) such as the Juju GUI [25].

Further information on the automated deployment of the OpenStack platform using Charms is available in the supporting document entitled “Automated deployment of OpenStack”.

4. CLOUD MANAGEMENT PLATFORM

Managing clouds can be demanding. Cloud management platform software is designed to reduce this complexity. It usually provides the following functions:

- Viewing and managing multiple clouds in the same environment.
- Creating and managing virtual machines in an integrated environment. It is able to provide templates for the rapid creation of virtual machines.
- Offers services across multiple clouds
- Has the ability to utilize existing IT infrastructure.
- Provides unified reports and analyses of cloud usage.
- Provides application deployment across multiple environments

The choice of platform depends on the functions provided by the software and on user experience. Some are intended for a specific hypervisor, while others may be usable with various hypervisors. Some of them are easy to use, while others may require more hands-on experience. The choice of platform depends on the types of tasks that need to be performed, on budget constraints, on software vendor support and on compatibility with the cloud technologies used in the hybrid cloud [2].

4.1. ManageIQ

ManageIQ is an open-source platform for managing hybrid cloud infrastructure. It can be deployed to manage small as well as large infrastructures, with support for technologies such as virtual machines, private/public clouds and containers. ManageIQ can be deployed in several ways, namely as a Vagrant box, a Docker container, a Kubernetes cluster or in a public cloud. Installation by any of these methods is relatively simple; for the POC solution we chose the Docker container [66][65]. Our motive is not to try out different deployment methods, but simply to understand the architecture and functionality that is useful for our hybrid cloud.

ManageIQ and other similar orchestration platforms communicate with clouds using API keys. These take different forms depending on the cloud provider. For example, with AWS it is a combination of an access key and a secret key. For GCP it is a special JSON file. OpenStack uses the user name and password defined in the Keystone service. Whatever it is, it is simply a way of authenticating the API calls made by the ManageIQ platform.

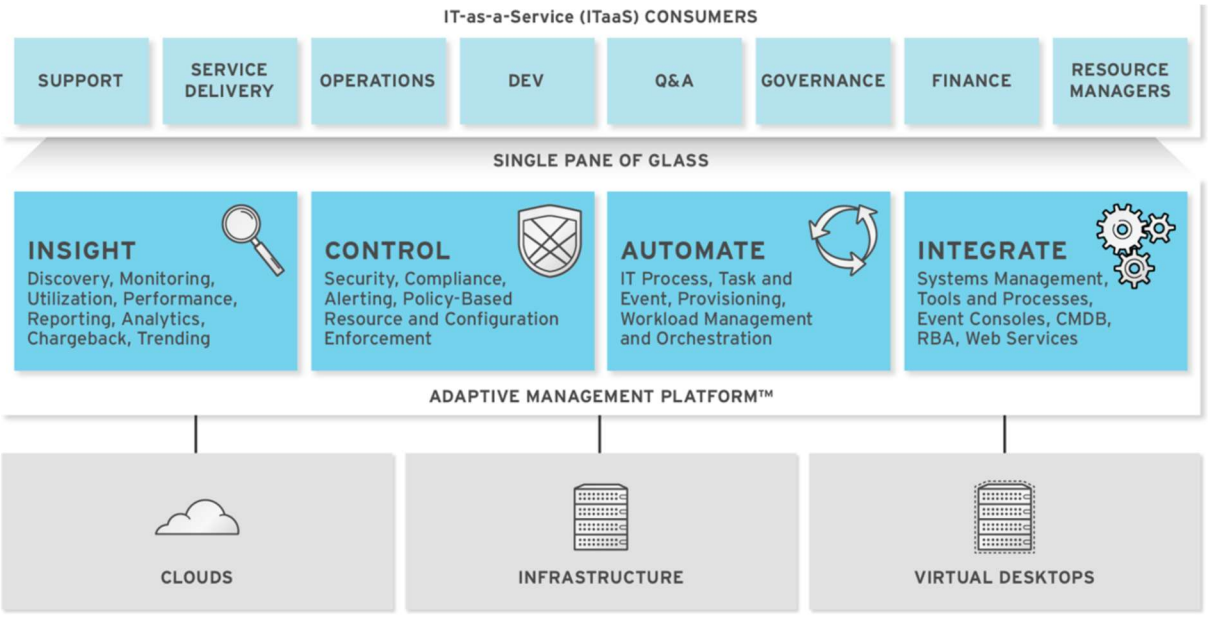


Figure 4.1 Overview of ManageIQ features and functions [5]

ManageIQ provides integration with the following cloud providers: Amazon Web Services, Microsoft Azure, Google Cloud Platform, RedHat Openstack Platform, IBM PowerVC, IBM Power Systems Virtual Servers, IBM Cloud VPC, Oracle Cloud.

It also offers integration with containerization providers: OpenShift Container Platform, Kubernetes and Amazon Elastic Kubernetes Service.

Last but not least, it can also integrate with infrastructure virtualization technologies: VMware v Sphere, Red Hat Virtualization, Microsoft System Center Virtual Machine Manager, OpenStack Platform Director.

4.1.1. ManageIQ architecture

The platform architecture consists of several components:

- The ManageIQ appliance, which is deployed in the form of a secure, high-performance, preconfigured virtual machine with native support for HTTPS communication.
- The ManageIQ Server, which is part of the appliance. It is a software layer that provides communication between the SmartProxy and the Virtual Management Database (VMDB). It also supports HTTPS communication.
- The Virtual Management Database, which is either directly part of the appliance or located on another server accessible to the appliance. The data in the database represent information about the overall virtual infrastructure integrated with ManageIQ. It also maintains information about the current state of the tasks running in the appliance.
- The ManageIQ Console is a web interface used to view and manage the ManageIQ Server and the appliance.
- The SmartProxy can be deployed directly on the appliance or on an ESX server. If it is not part of the ManageIQ server, it can be deployed directly from the appliance. The SmartProxy Agent must be configured at each storage location and must be visible to the appliance. This component communicates with the appliance via HTTPS.

4.1.2. Platform requirements

To use the ManageIQ platform, several specific hardware, database and browser requirements must be met.

Minimum hardware requirements:

- 4 vCPUs
- 12 GB RAM
- 44 GB HDD + optional size

Database requirements

Red Hat recommends allocating the disk for the virtual machine persistently at the time of its creation. The size of the database is influenced mainly by three factors:

- Number of virtual machines – the most important factor affecting the size of the database (VMDB).
- Number of host servers (Host count).
- Number of individual storage servers (this is not the total number of virtual disks belonging to the virtual machines).

The following table can be used to estimate the size required for the database [5]:

Table 4.1 Table for estimating the required disk size for the ManageIQ database

Number of VMs	Number of servers	Number of storage servers	After 1 year	After 2 years
100	5	50	3.5 GB	5 GB
500	10	100	17 GB	25 GB
5000	50	500	173 GB	251 GB

Browser requirements

To access its graphical web interface, ManageIQ supports common web browsers such as Google Chrome, Mozilla Firefox, Safari and Internet Explorer 10+.

Deployment options

In production environments, the platform is mostly deployed by downloading a pre-prepared virtual machine and then deploying it into a virtualization platform such as OpenStack or VMware. This is followed by connecting it to the virtualization platforms so that they can be managed using ManageIQ.

In addition to the traditional form of virtual machines, ManageIQ provides the option of deployment in the form of containers running in either Kubernetes or OpenShift. A list of platforms compatible with ManageIQ is given below.

Table 4.2 List of platforms supported by ManageIQ

Platform	Virtual machine / Pod
Amazon Web Services (AWS)	VM
Microsoft Azure	VM
Google Cloud Platform (GCP)	VM
Microsoft System Center Virtual Machine Manager (SCVMM)	VM
Kubernetes	Pod
Openshift Container Platform (OCP)	Pod
Red Hat OpenStack Platform (OSP)	VM
Red Hat Virtualization (RHV)	VM
VMware vSphere	VM

Another deployment method is to use Docker containers with any deployment. Once the application has been deployed, it is possible to log in to it and manage it. With a simple installation, the application offers a graphical web interface that runs on the IP address, or the domain name, of the virtual machine, on port number 8443.

4.2. Centralized authentication in ManageIQ

Centralized authentication is a way of controlling user access to information systems through a single source of identity verification. In this model, the user authenticates only once against a central authentication server, which then grants access to multiple services. The key element of this system is **LDAP (Lightweight Directory Access Protocol)**, which is used to access directory services containing data about users, their roles, groups and permissions. LDAP enables efficient searching and verification of credentials in a hierarchical structure of entries, thereby ensuring unified identity management and significantly simplifying administration in large networks. This protocol is widely supported and often used in combination with other security mechanisms, such as SSL/TLS encryption, to increase the protection of authentication data.

4.2.1. Configuration of authentication in the ManageIQ environment

The test instance used ManageIQ version *Petrosian-1*, running on the CentOS Stream 8 operating system on the VMware virtualization platform. The instance is configured with two virtual disks – one for the system and one for user data. LDAP/AD authentication is implemented through an external AD server. Owing to the absence of a trusted certification authority, the setting ldap_tls_reqcert = never was used in the SSSD service configuration file, which disabled TLS certificate verification. To ensure functionality, ports 389 (LDAP) and 636 (LDAPS) were opened at the network layer.

Summary of components and parameters:

- ManageIQ version: *Petrosian-1*
- OS: CentOS Stream 8
- Virtualization: VMware
- LDAP ports: 389 (LDAP), 636 (LDAPS)
- TLS certificate: verification disabled (ldap_tls_reqcert = never)

4.2.2. Installation and configuration of SSSD

The SSSD package with supplementary tools (realmd, oddjob, krb5-workstation, etc.) was used to

configure authentication. The configuration was based on the official ManageIQ documentation and the manual pages of the sssd-ldap.conf, ldap.conf and authconfig tools.

After installing the packages, the availability of the LDAP server was verified using the ldapsearch tool. The communication test was carried out over the LDAPS protocol with certificate verification disabled.

The sssd.conf configuration file was created on the basis of a template provided by ManageIQ. The configuration parameters defined:

- the LDAP URI,
- the method of authentication and identity mapping,
- binding to a specific domain and the attributes of user and group objects,
- management of the Kerberos realm.

The domain parameters specified the use of the Active Directory schema and attributes such as sAMAccountName, displayName, mail and msSFU30UidNumber.

After the changes, the system was restarted (systemctl restart sssd) to load the new settings.

4.2.3. Testing the configuration

Tools such as dbus-send, id, getent passwd and sss_cache were used to verify that the configuration works correctly. The commands made it possible to verify access to user and group attributes, as well as the functionality of lookups via NSS (Name Service Switch). Temporarily disabling SELinux enforcement (setenforce 0) made it possible to perform D-Bus queries in test mode.

The key log files for error analysis were:

- /var/log/sssd/sssd.log
- /var/log/sssd/sssd_*.log
- /var/log/sssd/sssd_ifp.log

If necessary, the debugging level could be increased via the debug_level parameter.

4.2.4. Integration with Apache and the ManageIQ UI

The authentication integration included deploying configuration files for the Apache server and adjusting SELinux rules. In the ManageIQ environment, group and access rights settings were made in the *Access Control* interface, where synchronization of LDAP groups and ManageIQ roles was ensured. The authentication mode was changed to *External (httpd)* with retrieval of groups from the server enabled.

After successful configuration and testing, the environment was ready for secure user access through central LDAP/AD authentication. This implementation represents an efficient and scalable way of controlling access within a manageable cloud environment.

5. 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.

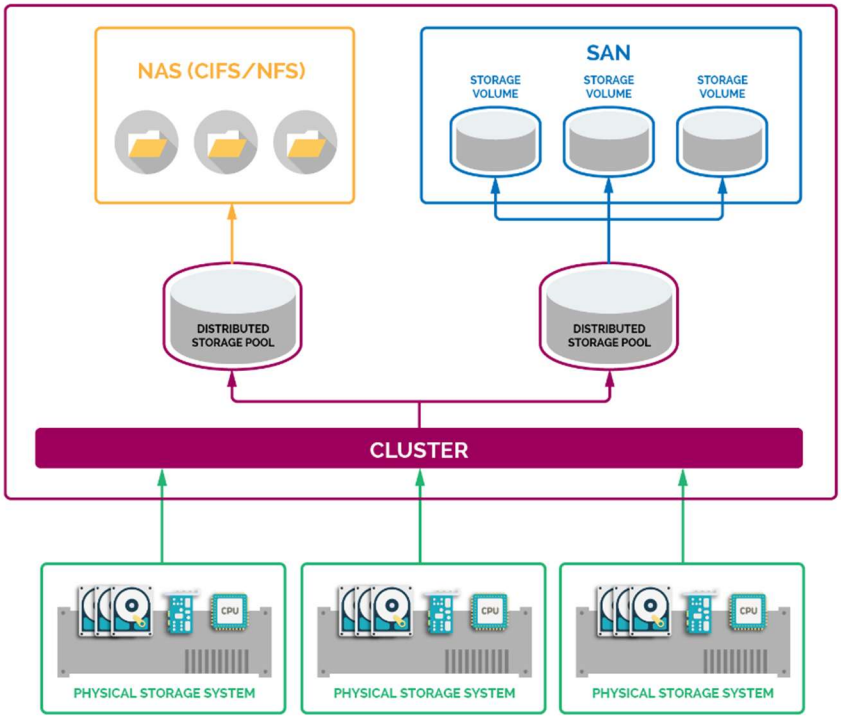


Figure 5.1 General schematic of a distributed storage system

5.1. Principles and advantages of distributed storage

Distributed storage ensures 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 Figure 5.*1* above shows the basic scheme of distributed storage, where decentralization is represented in the form of multiple physical servers ("physical storage system") unified into a so-called cluster ("cluster"). Access to the distributed storage can be provided by connecting to the cluster in the form of file-based network storage, NAS (Network Attached Storage), or block-based network storage, SAN (Storage Area Network) [27][28][29].

The foundation of distributed storage lies in its capabilities for seamless capacity expansion (scalability) and for partitioning and replicating data across multiple servers. This configuration enables efficient data management and parallel access to and processing of data, which significantly reduces bottlenecks. Data partitioning, or "sharding", is the division of data into smaller segments that are distributed across different storage servers. In addition, replication strategies ensure that multiple copies of the stored data are maintained on different servers, which provides 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 the system maintains. Higher replication factors increase data security but also require more storage space [29][30].

5.2. Use of distributed storage

Distributed storage is widely applied in various sectors, including media, healthcare and big data analytics. It enables efficient handling of extensive libraries of multimedia content, secure storage of sensitive patient data and management of large datasets for analytical processing [29][31].

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 significance of distributed storage will become even more important, leading to advances and improvements in data management technologies [31][32].

5.3. Reasons for deploying distributed storage

The reasons for deploying distributed storage for the cloud at KIS practically replicate all of the advantages of distributed storage mentioned above. The cloud is deployed on 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 machine is handled locally. As a result, when multiple clients access it, the demands on the specific server increase, which, in addition to slower request processing, over time increases the risk of that server failing. In the event of such a failure, owing to local data storage, data loss and the inability to provide the services located on that server are very likely.

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

6. 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 optimal performance of the OpenStack cluster. If the storage cannot provide the necessary input/output operations for the required cluster workload, this leads to performance problems that directly affect the operation 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 an incorrectly chosen solution can increase the workload of the administration team [33].

Just like providing the necessary computing power, ensuring the reliability of the storage solution is directly related to the reliability of the entire cluster. Therefore, data reliability must be the highest priority when designing a private cloud. Choosing the wrong storage solution can jeopardize 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 can result from a solution that increases overall costs without adequately meeting the computing performance requirements specified in the relevant SLAs (Service Level Agreements) [33][34].

Another important aspect in selecting software is understanding the nature of the data and the way in which 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 [34].

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

6.1. 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 software selection with regard to the requirements, infrastructure and architecture of the private cloud provided at KIS.

6.2. 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, which makes it highly scalable and resilient. Likewise, the self-healing and self-managing capabilities of Ceph significantly reduce the administrative burden [35][36].

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 [36][37][38].

MinIO specializes in high-performance S3-compatible object storage. It is a computationally lightweight yet powerful solution that is 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 [39].

Each of the options, Ceph, GlusterFS and MinIO, represents a unique set of features. A summary of the given criteria and how they are met by the individual software products is shown in the table below.

Table 6.1 Comparison of the criteria for Ceph, GlusterFS and MinIO

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

6.3. 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 and community-supported solutions. In addition, scalability is a significant advantage, as it can efficiently process data at up to the petabyte scale, thus ensuring that the storage infrastructure can grow along with possible future needs without a significant decrease in performance. Ceph's integrated redundancy features 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 seamlessly replicate and recover data provides the resilience that is necessary for the provision of cloud services. 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 OpenStack Charms tools, which facilitate management and deployment in the OpenStack environment.

As for other potential options, e.g. 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, OpenStack Charms tools do not support GlusterFS, which is one of the most important software selection criteria. 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 S3-compatible object storage, 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 OpenStack Charms tools is also highly questionable, as the official documentation does not include this variant.

7. CEPH AND ITS ARCHITECTURE

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 has been renamed IBM Storage Ceph for business ("enterprise") use. This version includes enterprise features and support services with SLAs and is deployed using Linux containers for easier management [41][42].

Every year, with a target date in March, a new stable version is released, named in alphabetical order. Version numbers are used in 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 [43].

According to HG Insights, Ceph is also used as storage by major companies such as NVIDIA, the logistics giant Maersk Line, and the telecommunications company Ericsson [44].

7.1. 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 be easily scaled and adapted to user needs, from terabytes to exabytes and beyond, without any disruption. In the following subsections, we will explain how the individual components work, what role they have in the **cluster** and what advantages they provide [40].

7.1.1. Ceph cluster

The foundation of the entire Ceph system is the cluster. The cluster consists of nodes, which are formed by 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 the complex data storage requirements of current data environments [40].

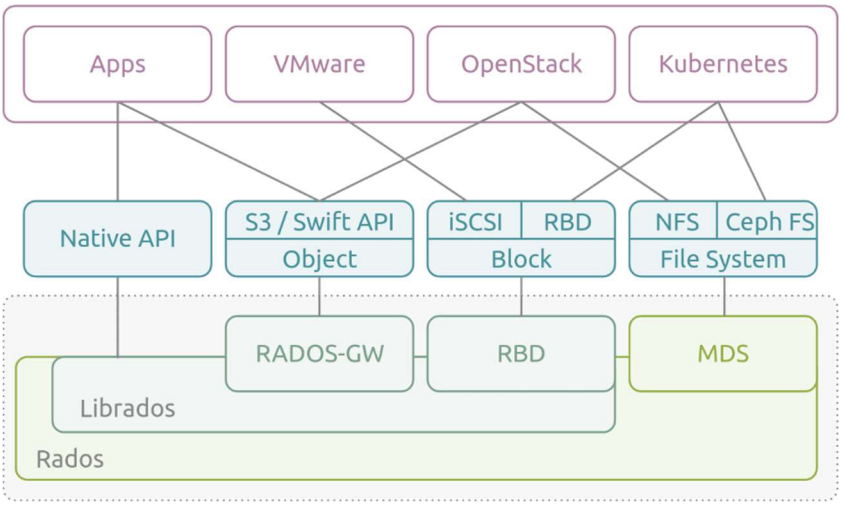


Figure 7.1 Ceph cluster architecture [40]

Figure Figure 7.*1* 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.

7.1.2. Ceph Monitor

Ceph Monitor (also referred to as "*ceph-mon*") is an important component in maintaining the functional state and stability of the 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 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 [40][46][47].

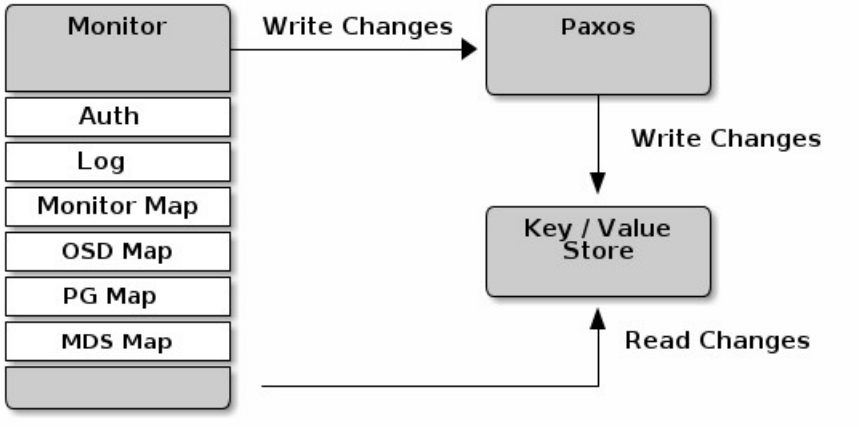


Figure 7.2 Illustration of the Ceph monitor's tasks and the decision-making process [21]

Figure 7.*2* shows the individual tasks of the monitor, through which it ensures the smooth running of the cluster with the help of the Paxos consensus algorithm. This algorithm allows the monitors to agree on the cluster state and make decisions jointly, thereby increasing

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 the integrity and availability of data, since making a decision requires more than half of the monitors to be in agreement [40][47].

Using 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 allows 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 [40][46].

7.1.3. Ceph Manager

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 other than ensuring its operation. Deployment tools such as "*ceph-ansible*" or "*cephadm*" are usually used to set up ceph-mgr daemons on the monitor nodes, although they need not necessarily be located on the same nodes as the monitors [48].

The manager's architecture 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 prometheus modules, the manager also provides modules for Zabbix, for sending alerts, modules for orchestration and a RESTful API interface [48].

Among the greatest advantages of the manager for the Ceph cluster are precisely its modularity and simplified interface for cluster management. This modular approach allows administrators to adapt management functions to their specific needs, thereby optimizing the performance and reliability of the cluster as a whole [48].

7.1.4. OSD daemons

Ceph Object Storage Daemons (OSDs) – OSD daemons are the most important components of the architecture and operation of the Ceph cluster. They serve as the basis for data storage, replication, recovery and cluster health monitoring. OSD daemons, also known as "*ceph-osd*", store data within logical pools, ensuring 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 efficiently process huge volumes of data [35][40].

Each physical disk or logical partition designated for data storage usually has its own OSD daemon. Data storage and replication in the cluster are carried out precisely by means of 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, the value of which is given in the configuration. The distribution and management of data on 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 [40][49].

OSD daemons also include monitoring functions. At regular intervals, they send so-called "*heartbeat*" messages to one another 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 [40][49].

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 more disks to the cluster increases the total storage capacity and distributes the workload over a larger set of resources, thereby ensuring high availability. Moreover, data replication between individual OSD daemon units ensures that no single point of failure can jeopardize data availability. For this reason, the cluster is able to continue operating 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 [40][49].

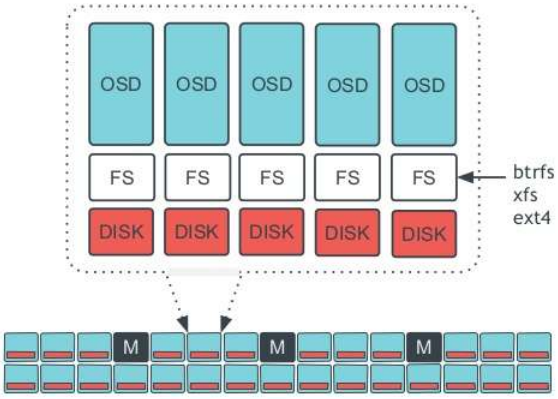


Figure 7.3 Illustration of OSD daemons on a node server [24]

7.1.5. MDS and Ceph File System

The Metadata Server (MDS) is a key component of the distributed file system CephFS (Ceph 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 [51].

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 therefore they are not used without the CephFS system [40][51].

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 allows fast access and management. This setup

enables 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 [52].

Separating metadata storage from data storage allows a cluster using CephFS to significantly optimize file-based operations. Handling metadata separately reduces the load on the OSD disks, which leads to better performance and scalability of both data and metadata operations. Ceph's architecture allows metadata management to be scaled by adding more 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 [40][51][52].

7.2. 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 on a cluster of standard hardware. RADOS forms the core of Ceph and supports block, file and object storage interfaces. The system is designed for high performance, extensibility and reliability. It manages the replication, recovery and distribution of data without a single point of failure [40][52].

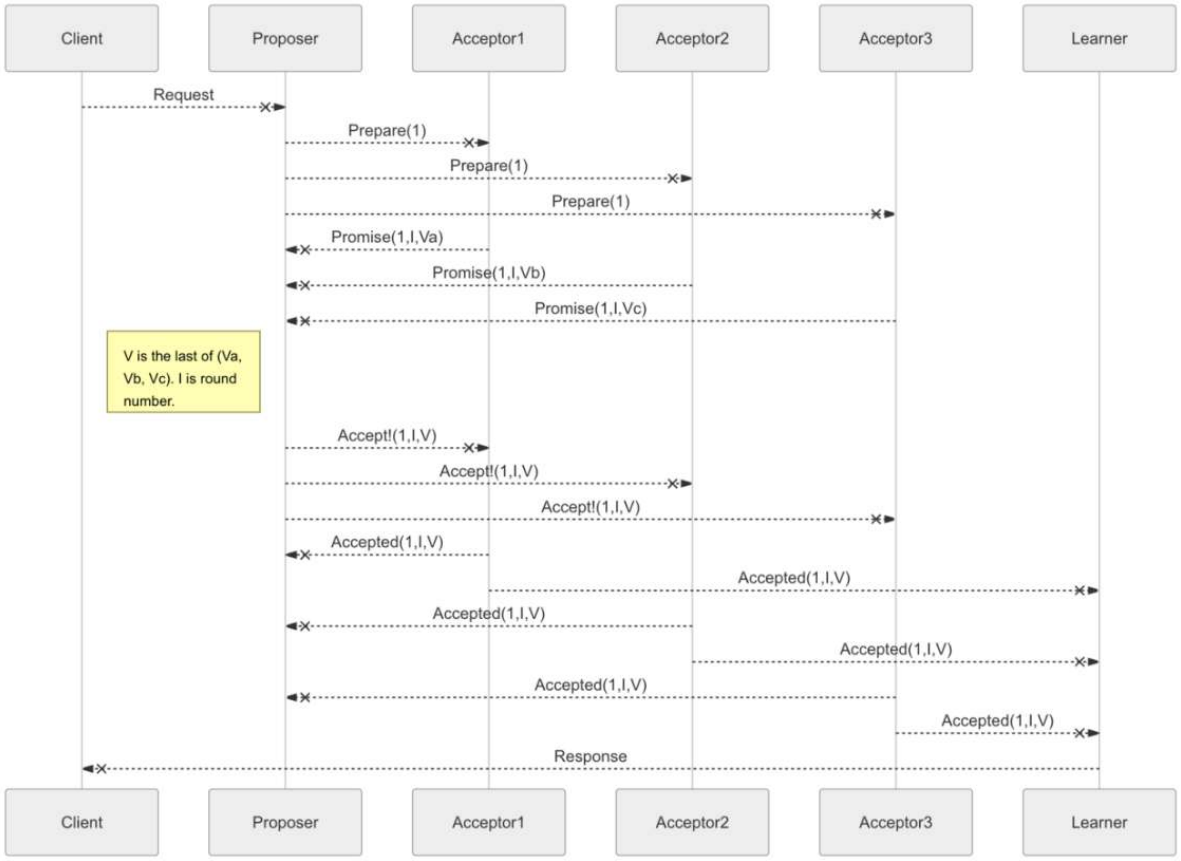


Figure 7.4 Principle of reaching consensus using the Paxos algorithm [28]

Since the architecture of the Ceph cluster is primarily built on RADOS with additional functions, RADOS clusters likewise consist of two types of daemons, namely OSD and Monitor. It is precisely RADOS that is the technology providing almost all of the well-known advantages of Ceph for distributed storage. It uses the CRUSH and Paxos algorithms for efficient distribution and management of data within the cluster, the principle of which is shown in Figure Figure 7.*4* above. RADOS also ensures data durability and availability through replication and, last but not least, the 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 [40][53].

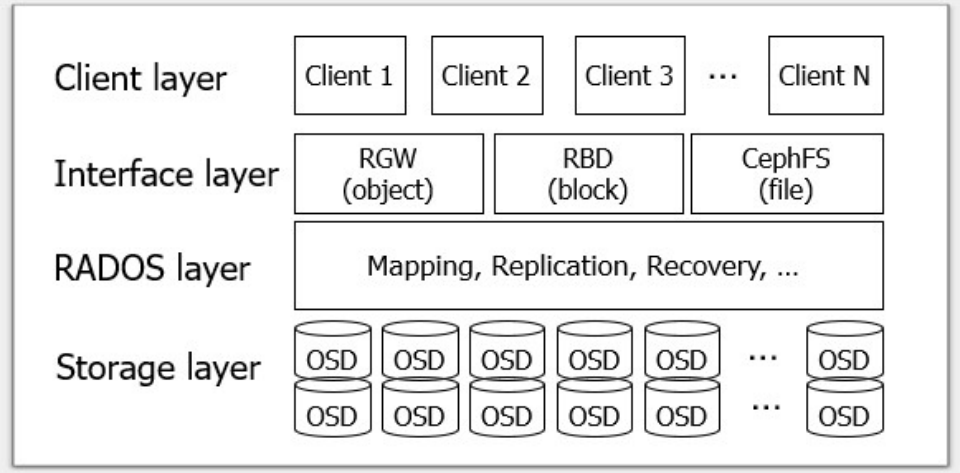


Figure 7.5 Illustration of the individual layers of the Ceph architecture [30]

The logical division of the Ceph cluster into layers is shown in Figure Figure 7.*5* above. The 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. Ceph offers Ceph Rados Gateway (RGW) – an object storage interface that provides applications with a RESTful API interface, Ceph Rados Block Device (RBD) for storing blocks and images, e.g. for virtual machines and virtualization platforms, and CephFS, which can be mounted like an ordinary disk in the operating system in the form of shared storage. RADOS layer – the RADOS layer is the control layer responsible for all the properties mentioned above in the previous paragraph. The last layer is the storage layer, which consists of all the OSD daemons in the cluster [40].

In addition, in the interface layer Ceph also offers direct access for the native API using the "librados" library, as shown in Figure Figure 7.*1*. The librados library provides an API interface that enables 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 an intermediate client 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 offers support for asynchronous I/O operations and provides libraries for multiple programming languages, such as C, C++, Python, Java and PHP [40][57].

7.3. 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 manner that maximizes availability and minimizes the need for data rebalancing when the cluster grows or shrinks. Unlike traditional data allocation methods,

which require a central directory to record the position of individual data, CRUSH allows Ceph clients to calculate data placement directly using the cluster map. This significantly reduces overhead costs and the risk of potential bottlenecks, enabling the Ceph system to achieve reliability and performance when processing huge amounts of data across many storage nodes [40][46].

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 multiple devices or other buckets. These then form a tree structure reflecting the physical layout of the storage environment. This structure is encoded in the so-called CRUSH map, which is supplied to all Ceph clients and OSD daemons. An example of a CRUSH map can be seen in Figure Figure 7.6 [40][58].

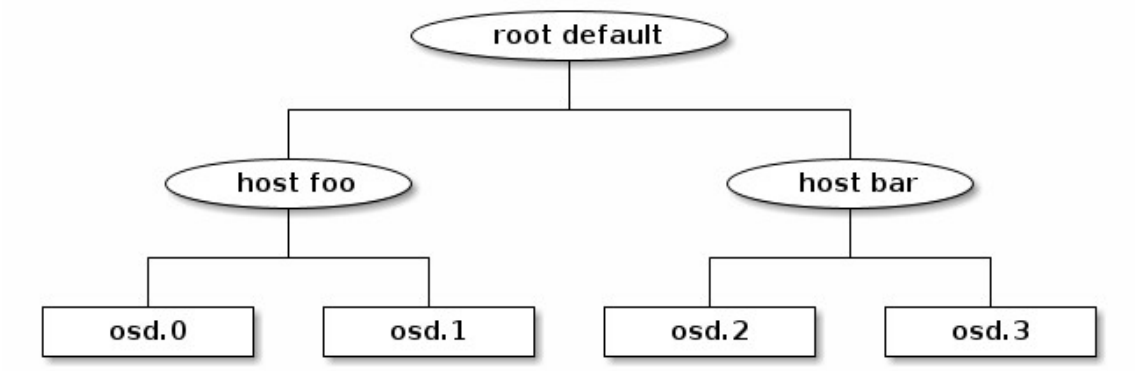


Figure 7.6 Example of a CRUSH map structure [58]

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 rules 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 [40][58].

By distributing data in a predictable yet random manner across multiple OSDs, CRUSH increases data availability and fault tolerance. It allows the cluster to dynamically rebalance data 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 [40][46][58].

CRUSH allows Ceph clusters to scale out horizontally with minimal impact on performance by evenly distributing the workload across all available resources. It reduces latency and increases throughput by providing direct client communication with OSDs without a centralized lookup table. Administrators have the option of defining their own hierarchical structures that correspond to their actual hardware layout, thereby optimizing performance and data durability [40][46][58].

7.4. Pools and PGs

Pools are the basic units in which objects are located 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 [40][59][60].

7.4.1. Pool

In a Ceph cluster, pools are logical partitions that contain objects. They act 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 [40][59].

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

Ceph pools are configurable, which allows administrators to adjust settings such as the number of object replicas, CRUSH rules and performance-related parameters. They enable logical grouping of data, which facilitates its management and access to it. Different pools can be created for different types of data, such as images, videos or backups, each with its own settings [40][59].

However, it is important to understand the following concept. Once a pool has been created, Ceph automatically manages the distribution of data among the available OSDs based on the configuration and the CRUSH algorithm. Pools are therefore configured to govern data replication and placement strategies, not 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 data is distributed among OSDs based on the pool configuration and the cluster topology [59].

In a Ceph cluster, the decision as to 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 [40].

7.4.2. 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 object locations 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 [40][60].

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 hashing function. The result of the hashing 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 in which we want to find one specific object among a million objects without a previous cache entry, determining the exact location of the object is extremely time-consuming, which reduces the performance of the cluster. Using PGs minimizes the time needed to locate an object, since, based on the calculated hash, the CRUSH algorithm can almost instantly determine the OSD and PG in which the object is stored. Based on this information, the CRUSH algorithm directs the client to the specific OSD to which it will write the data or from which it will read the data [61].

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 on 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 to maintain performance and resource utilization without manual intervention. Autoscaling takes into account the total storage capacity, the distribution of data among different storage devices and the expected data size in order to adjust the number of PGs as needed [40][62].

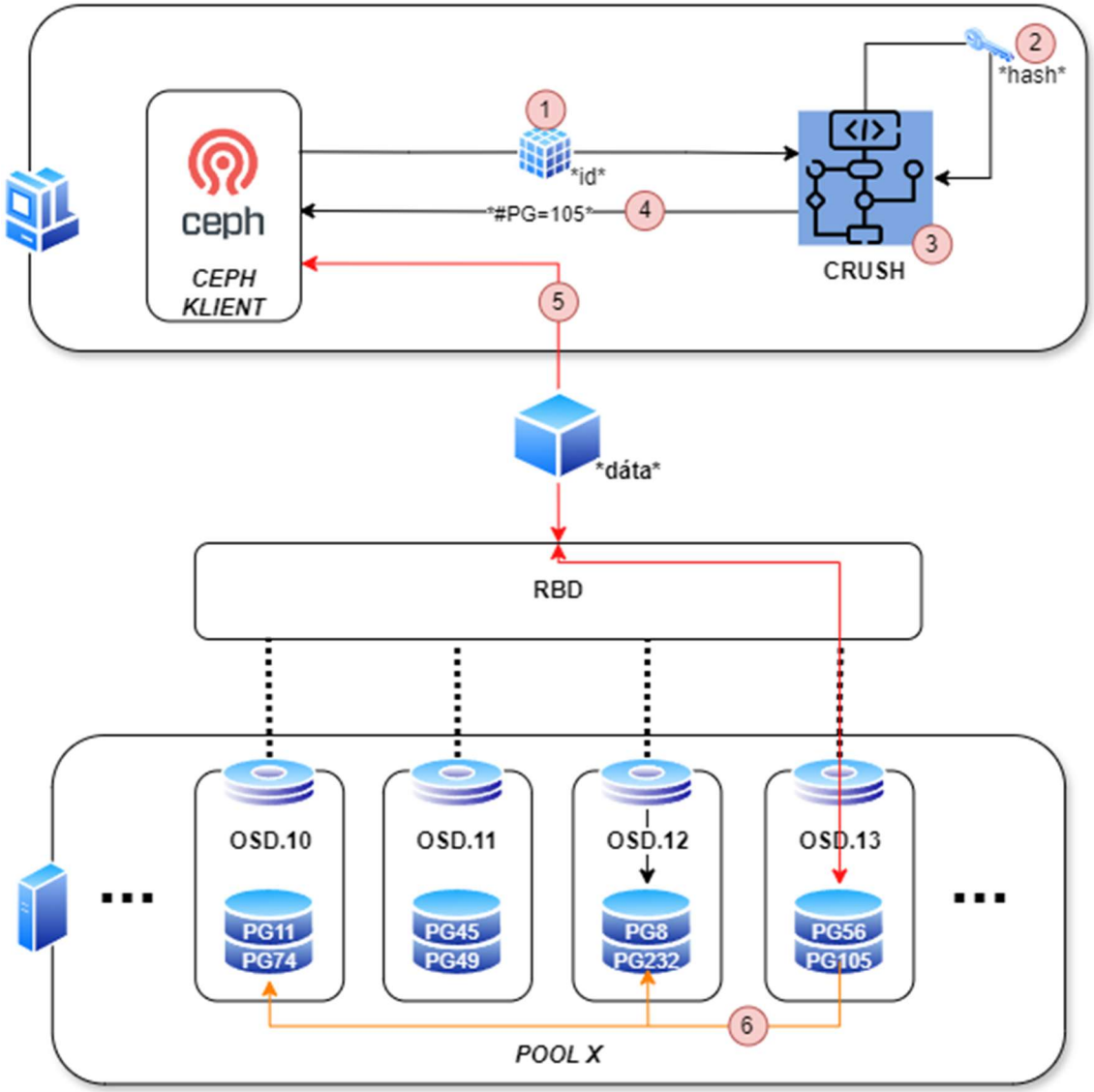


Figure 7.7 Principle of writing/reading data using a client

To illustrate how the process of writing or reading data works, we will use Figure 7.7. The process consists of six steps. In the first step, the client sends the 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 holds the entire current CRUSH map, in the fifth step, based on the PG number, it establishes a connection directly with the relevant OSD from the selected pool on which the PG is located. Depending on the operation, the data is either written or read. The last, sixth step, in the case of writing data, is replication to the secondary OSDs, or rather to their respective PGs.

Further information on the deployment and testing of the Ceph solution for Openstack is available in the supporting document entitled "Distributed storage".

8. IMPLEMENTATION OF A SHARED FILE SYSTEM IN THE OPENSTACK ENVIRONMENT

In this section, we will focus on how a shared network file system can be implemented in a cloud environment on the OpenStack platform using the Manila module.

8.1. Storage in the OpenStack platform environment

Having selected CephFS as the most suitable solution for the OpenStack infrastructure at KIS, as mentioned in the previous chapters, it is essential to ensure that the solution deployed in this way can be managed efficiently and easily. Within the OpenStack platform, two main storage projects are used, namely Cinder and Swift.

Cinder provides an API for managing block storage, which provides a high-performance connection for virtual machines and other data; however, a volume (storage disk) can be accessible to only one such virtual machine at a given time. This limitation arises because it does not include any locking or access synchronization mechanisms [10].

The Swift module, in turn, provides object storage, which is particularly suitable for storing large binary objects such as images, videos or various backups. The disadvantage, however, is that objects stored in it in this way are immutable, i.e. once an object has been stored, it can no longer be edited or modified in parts. An object can be changed in this way only by overwriting it completely, and it is therefore not suitable for managing data that changes frequently or is subject to transactional operations, specifically databases. Moreover, it is not recommended for storing small objects either, because it includes implemented methods for their protection, which, if they were handled frequently, would be reflected in the speed of operations [10].

For these reasons, the Manila module was created; it is derived from the Cinder module and provides a very similar management interface for shared file systems [10].

8.2. Manila as a service for managing shared network file systems in the OpenStack environment

Manila is a service specifically designed for managing shared network file systems in the OpenStack environment. Its main function is to provide the Nova module (responsible for creating and managing virtual instances) with access to shared file storage. It thereby enables virtual instances to read and write data on shared storage, which brings the benefits of centralized file storage [10][11].

The Manila service provides the orchestration components responsible for the process of creating and assigning shared file systems. A major advantage of this module is that it provides only a management layer, whose task is administration, while the actual interaction between the storage and the instances is managed at the backend level. This functionality is implemented as a set of APIs, a command line interface (CLI), and also integration into Horizon (OpenStack dashboard – graphical user interface) [10][12].

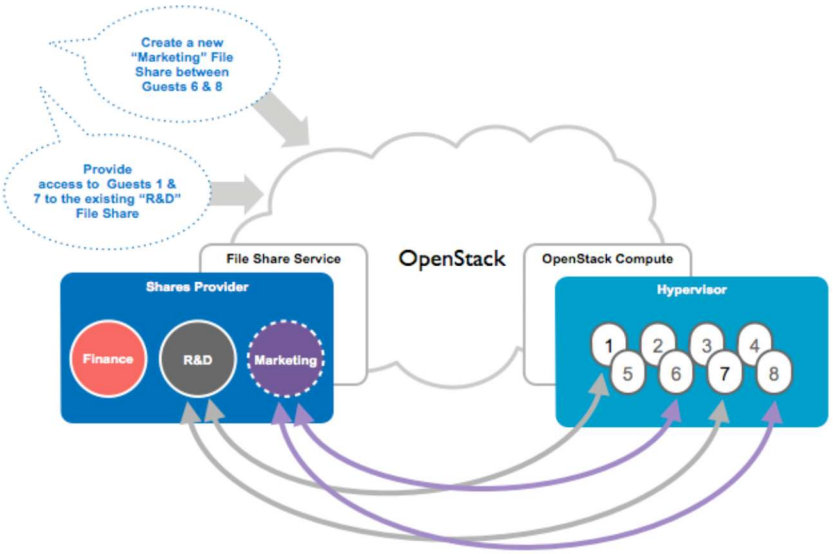


Figure 8.1 Communication flow between the Manila module and Nova instances [12].

Various concepts and terms are used within this service, the most important of which are:

- **Share** – This is a specific instance of a file system – a share, which is assigned to a virtual machine or server and can contain data accessible to multiple instances simultaneously, with the assurance that access by individual instances will be controlled and all instances access the same files [10][12][13].
- **Share Access Rule** – These represent security rules/access rules that define who can access a given share and in what way. They work similarly to ACLs [10][13].
- **Security Service** – It is used to verify the identity of users by means of external services such as "LDAP" (Lightweight Directory Access Protocol) or "Micrsosoft Active Directory". In this way, administrators will be able to easily define groups of users who will have access to a given share [10][12].
- **Share Network** – It represents the network configuration assigned to a given share. In addition to specifying the network or subnet itself, it is also used to implement "multi-tenancy" (a way of partitioning the same infrastructure for multiple users so that they have an isolated and secure environment without affecting one another) by means of VLAN and VXLAN, thanks to which the data is truly separated and inaccessible to users from another network [10][12].

Other relevant terms in the implementation of Manila include:

- **Snapshot** – Represents a static copy of an existing share at a given point in time. It serves primarily for backup and possible data recovery purposes [12][13].
- **Storage Pools** – These are logical groups of storage resources that effectively enable Manila to allocate and manage storage while giving its users flexibility in choosing where to store their data [12].
- **Share Type** – Allows administrators to define criteria separately for different service levels. For example, creating two classes of shares, where one represents a basic simple share for routine client service, while the other is defined as a higher-performance share for special purposes [12].

- **Share Server** – Forms a logical entity that hosts the shares assigned to a particular share network. This server may be either a configuration within the storage system or other logical resources provided by the OpenStack platform. The share server is also responsible for determining the correct IP addresses for exporting shares within the relevant network [12].

The service itself can run in various configurations, either on one or on several nodes in the cluster, and is capable of supporting several different backends at the same time. Within this diploma thesis, however, we decided on only one backend, namely CephFS, which will serve as the shared file storage [12].

8.3. Objectives and advantages of using the Manila service

Manila, an open-source service for shared network file systems in the OpenStack environment, sets itself several basic objectives, which are:

- **Modular architecture for easy extension** – The service uses a component-oriented architecture, which essentially allows new functions and further extensions to be added quickly. As a result, the entire service is adaptable to the required criteria of the given deployment [11]
- **Scalability for large workloads** – Manila is designed to be able to handle even demanding tasks that place a high load on the system. It supports both horizontal and vertical scalability [11].
- **Resilience to outages** – Within the service, the individual processes are separated from one another, which prevents failures from spreading between different parts of the system, thereby preventing so-called cascading outages that could render the entire service non-functional [11].
- **Secure file sharing with access control** – Since Manila offers as many as two options for controlling user access, namely by means of Share Access Rule and Security Service, one of its greatest advantages is precisely the functionality of controlling user access to shared files. All files are thus secure and accessible only to those servers and users that have previously been permitted by the administrators [11].

8.4. Key components of the Manila service architecture

The service itself consists of 4 main components, namely Manila-api, Manila-data, Manila-scheduler and Manila-share. The installation of the first three is identical for different deployment methods; the last one, however, depends on the selected type of storage backend, which in our case is CephFS [12].

Manila-api

The Manila-Api component represents a kind of entry point to the shared network file system service in OpenStack. It provides a RESTful API through which created shared file system instances can be created, modified and deleted. The task of this component is therefore to receive API requests from clients/users or other services for share management and to send them for processing to the backend, such as CephFS, which we have selected. In addition, this component also handles the authentication and authorization of requests by means of the Keystone module [14].

Manila-data

The main task of this component is to process data operations, which include copying, migrating shared files and backup. The advantage of separating these demanding operations and moving them into a separate service results in better scalability of the individual components of the Manila module [14][15][16].

Manila-scheduler

The next component is responsible for selecting a suitable backend for a new share. When a user or service sends a request to create a new share via manila-api, the latter passes it to manila-scheduler. The scheduler then analyses all configured backends and, when subsequently deciding which one to choose, uses various configurable filters and weights that take into account specific attributes of the offered backends. These weights include, for example, available capacity, availability zone and others. After a suitable backend has been selected, the manila-scheduler component sends a request to manila-share, which then continues to manage the creation of the share [14][15].

Manila-share

The manila-share component is responsible for the actual creation, management and export of shares on the relevant backend. A share represents a persistent readable and writable file system that can be accessed by a varying number of clients [15][16].

In this implementation, it has a defined protocol, size and list of access rights, which determines which clients have access to it. The manila-share service communicates with the storage for the purposes of securing and managing a specific share and manipulating it [16].

8.5. Interaction of the Manila service components

Communication between the individual components usually takes place by means of a message queue, where a "message broker" (a program used to share information between applications) such as RabbitMQ is mostly used. In addition, the service works with a database in which metadata about shared file systems, access rules and the status of individual operations are stored [17].

From an external perspective, the client communicates with the service via python-manilaclient, either through the command line or as part of scripts and programs using the Python library. The library then sends the received request via the REST API to the API server, which redirects it to the Auth manager, which is usually the Keystone module providing authentication. After the user's identity has been verified, the API component processes the request, stores the necessary information in the database and then sends it via the appropriate queue using the AMQP "Advanced Message Queuing Protocol" to the next component, in this case manila-scheduler [17].

Since the task of manila-scheduler is to decide which backend should be used as the storage for the relevant share, it accesses the database, obtains from it current information about the available backends, such as their capacity or status, and on that basis selects the most suitable one. It then places this selection into a message and sends it to the manila-share component for processing [17].

The manila-share component receives this message and performs the relevant task, which, as already mentioned, may be the creation, deletion or update of the relevant shares. In doing so, manila-share communicates directly with the selected backend. Depending on the type of backend selected, the appropriate Manila drivers are used, which mediate communication and operation on the storage side [17].

In the final step, the newly created shares are exported and made available to end users, who access them using the protocols defined by the backend [17].

The entire course of the communication described in this chapter can be seen in the figure below:

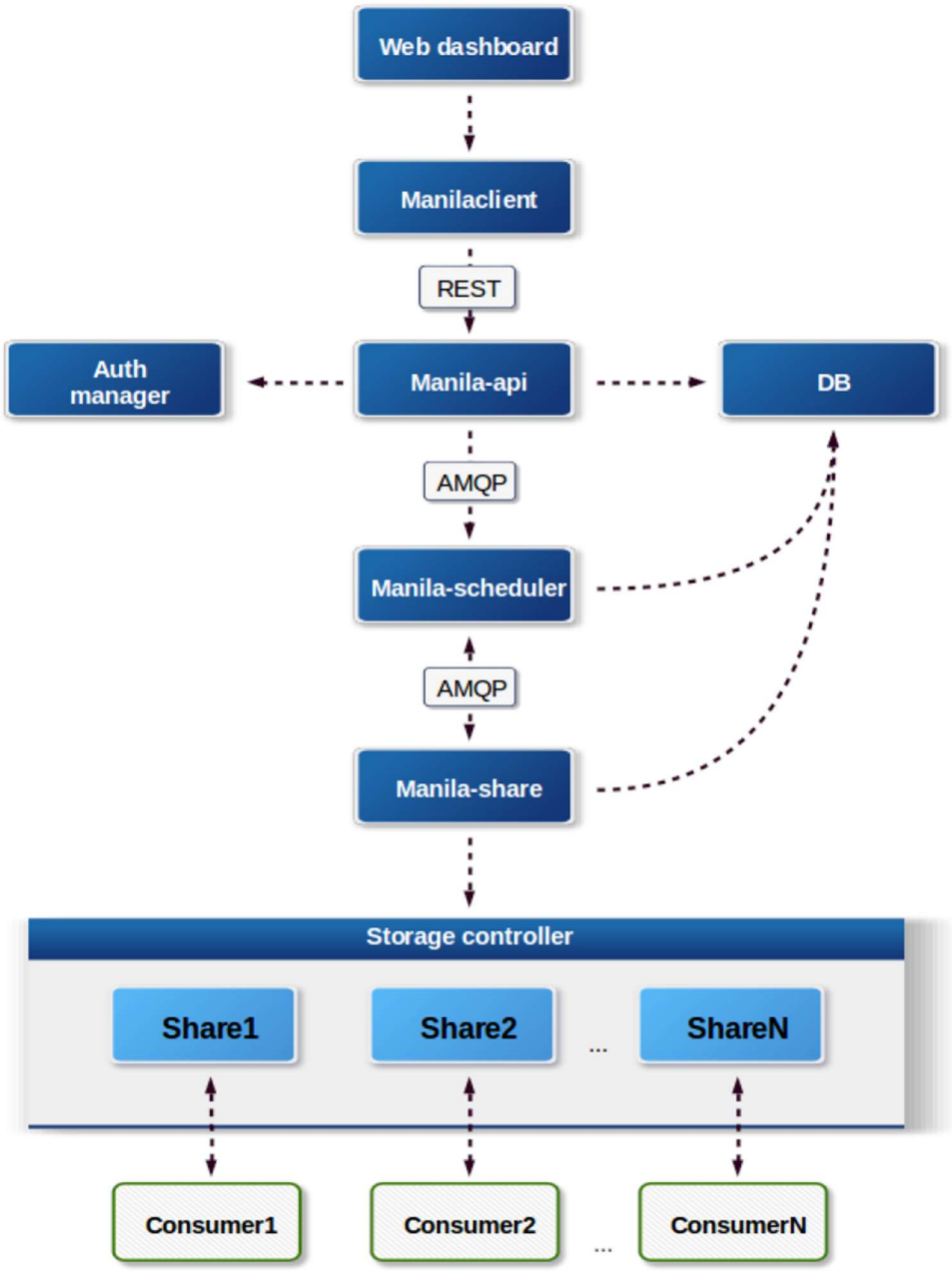


Figure 8.2 Course of communication of the Manila service together with other applications [17].

Within the OpenStack environment, from the user's point of view, this is ordinary network storage that the user can mount (attach) to their operating system, for example within a virtual instance, just like any other network drive, while such created shared systems can be configured with different levels of access rights (read-only, read-write) [17].

More precise and technical information on the deployment and the scripts that deploy the Manila component into the OpenStack system using Juju Charms can be found in the supporting document "podklad pre V3 v rámci KPB1_Nasadenie_Manila_bundle.docx".

9. MONITORING IN THE CLOUD COMPUTING ENVIRONMENT

To begin with, it is appropriate to define what the term *Cloud computing* actually represents. In this chapter, we will attempt to identify the motivation behind monitoring such a system and we will distinguish between different perspectives.

9.1. Motivation for monitoring

Both the provider and the user of such a platform have an interest in monitoring the CC environment. It is evident that the service operator needs to detect failures in both the physical and the virtual infrastructure. Its interest should also be to optimize the operation of the infrastructure [14].

Since the relationship between the provider and the customer should be formally defined, in this case in a service level agreement (SLA), the monitoring system should be able to alert the system owner to possible problems that may affect the performance of the service and thereby cause financial losses.

The customer has an interest in monitoring the resources provided to them, mainly in order to have an overview of whether they need the allocated amount, since a larger amount of allocated resources means higher costs. On the other hand, the customer must know whether the resources allocated to them still meet the requirements of their project and whether they need to be increased [14].

There is a large number of processes that are critical for the operation of a CC platform, and for each of them monitoring is a key task. These processes, as well as the role of their monitoring, are presented below [15].

9.1.1. Capacity and resource planning

Before the mass adoption of CC, one of the biggest challenges for operators and application developers was planning and managing the physical resources on which a given application ran. In order to guarantee a certain quality of service, they needed to quantify how many resources they required for the smooth running of processes. In general, such an estimate can be made on the basis of testing, monitoring and statistical analysis, but the real values may be unpredictable and highly variable. In this respect, CC is a great advantage for application service providers, since this entire process of monitoring and managing the infrastructure passes to the operator of the CC system. Here we come to the fact that we also mention above in chapter 9.1. Monitoring of the underlying infrastructure is critical from the CC service provider's point of view if it wants to track and predict the development of the parameters involved in the process of maintaining quality of service (QoS). On the basis of the outputs of such a monitoring system, the provider should be able to correctly plan the development of the physical infrastructure and the provided resources so that the SLAs concluded with all customers are respected [15].

9.1.2. Capacity and resource management

The basic prerequisite for a functional monitoring system is the ability to capture the current state of the CC platform. The main domain in CC is virtualization, which hides the heterogeneity of the underlying physical infrastructure and thereby brings a higher level of complexity for the provider, who consequently needs to manage both the physical and the virtual infrastructure. Virtual resources can migrate between physical devices, which may cause variability in the resources available on individual physical devices. A properly deployed monitoring system can alert the administrator if there is a shortage of computing resources on any of the nodes. In addition, the aspect of service availability is important for many customers.

Monitoring can reveal anomalies in the operation of the system, and we therefore consider it necessary in the context of guaranteeing the highest possible availability [15].

9.1.3. Data centre management

Cloud services are usually provided on the basis of large data centres, the management of which is an important task for the smooth operation of a complex system. Data centre management activities can be divided into two basic tasks:

- **Monitoring** – Tracking the required hardware and software metrics.
- **Data analysis** – Processing the collected metrics in order to evaluate the state of the system.

It is important that such a monitoring system allows a large number of operations in real time, since a large number of infrastructure elements is expected to be present in a data centre. In this context, the main motivation for monitoring is energy saving, i.e. optimizing the operation of physical devices [15].

9.1.4. SLA management

Monitoring is a mandatory and irreplaceable part of the process of verifying SLA compliance, for example when audit activities are carried out to determine compliance with regulations (e.g. where the service is offered to state authorities). Data from the monitoring system can help the provider formulate SLAs more realistically and design a sustainable billing model [15].

9.1.5. Billing

As already mentioned above, one of the basic characteristics of CC is the provision of services on a "pay for what you use" basis. This allows the customer to pay in proportion to their use of information resources. The basic prerequisite for billing in this way is the existence of a process that tracks the amount of resources used by customers. This process may be either standalone or combined into one with the monitoring solution [16].

9.1.6. Problem detection

CC infrastructure is often complex and presents a challenge when it comes to precisely determining the origin of a problem that has arisen. When a problem occurs, it is necessary to analyse several possible components that may have failed. Another fact that adds to the complexity is that the components themselves may consist of several layers (e.g. the physical or virtual layer, the operating system (OS) of the host or virtual machine (VM), etc.). A comprehensible and detailed monitoring system can help the provider diagnose the problem that has arisen more quickly and more accurately. In addition, the customer should be able to verify whether a possible fault occurs on their side or on the operator's side [15], [16].

9.1.7. Performance management

In practice, some compute nodes in CC provide greater performance than others. For an ordinary customer this has no significant impact, since the load tends to be distributed evenly and the drop in performance is usually not significant. However, there may be customers who provide critical services using CC. For them, every change in performance is significant, and the availability of computing resources is critical. For such services, the customer tends to use several different providers, choosing the current one (on which the service will run at that moment) on the basis of the provided, perceived performance. In this case, the customer has an interest in monitoring their perceived service performance, while the provider wants to provide them with the most reliable resources possible. For both parties, monitoring is a necessity in this case [15].

9.1.8. Security

Security in CC is important for several reasons. First of all, it is one of the main factors that deters a customer from migrating their infrastructure to such an environment. A proper monitoring system is necessary for managing security in CC infrastructure or applications. When providing services to certain state organizations, CC must meet strict criteria concerning data management regulations. This can be achieved by means of a monitoring system that can credibly prove that the user's data have not left the borders of the country [16].

9.2. Perspectives on monitoring

For the practical deployment of the solution, we need to determine the information that the system will collect. We primarily look at the issue from two perspectives. We describe them below.

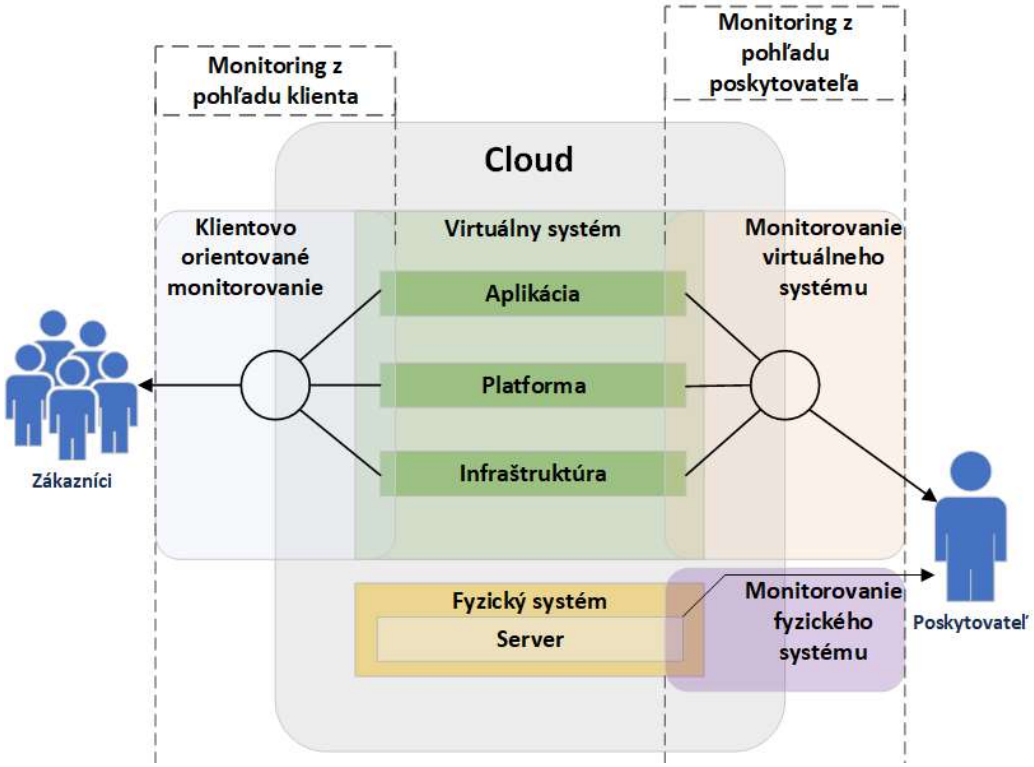


Figure 9.1 Perspectives on monitoring [17]

Figure 9.1 graphically shows what should, in general, be monitored and to whom the data from the individual parts should be presented. From the point of view of this work, neither the platform part nor the application part is of interest. We can see that the cloud service provider (CSP) should have information about the state of the physical system and the virtual infrastructure, which in the case of an IaaS platform means mainly VMs. The customer has no motivation to track the physical system and is therefore interested only in the state of their virtual resources. The provider may make such information available to them in some way.

Since the CC model is always closely linked to the guarantee of quality of service, useful monitoring data should be linked to the conditions defined in the SLA (see chapter **Chyba! Nenašiel sa žiaden zdroj odkazov.**). Defining metrics for both forms of monitoring,

from the point of view of both the client and the provider, is a critical aspect of a monitoring system. However, there are several problems that need to be taken into account in this context [17].

In the case of client-oriented monitoring, the metrics must provide information using the same parameters as defined in the SLA. As an example, we give a scenario in which the CSP guarantees each customer a maximum storage capacity for their VM in GigaBytes (GB). In such a case, the CSP must monitor the storage of every VM of all customers and then appropriately interpret these data in the same units.

The same principle applies in the case of monitoring the virtual system. What should be monitored is also closely linked to the SLA. However, for the CSP, these data should provide additional information [17]. For better understanding, we will build on the example in the previous paragraph. In addition to monitoring the storage on each VM, the CSP should also collect data on the total virtual storage in the CC platform. It should know the total storage utilization by all clients. This information should serve only the CSP.

The case of monitoring the underlying physical system differs from the previous two, since it concerns the lower level of the infrastructure on which the entire CC platform rests. This means that the required metrics are the same as for ordinary distributed systems. On the hardware side, these metrics include, for example:

- CPU utilization,
- RAM utilization,
- physical storage utilization and
- network traffic [17].

On the OS metrics side, we can include here, for example:

- system load and
- virtual memory utilization, since its management is the responsibility of the OS [17].

The information collected in this way serves primarily for the management of the entire system. In this case, the link to the SLA is rooted in the relationship between the low-level physical system and the high-level virtual system that rests on the given infrastructure [17].

More detailed information on monitoring the OpenStack cloud is available in the supporting document entitled "OpenStack monitoring design".

LIST OF SOURCES

[1] Techtarget (2025). OpenStack. Available online: https://www.techtarget.com/searchcloudcomputing/definition/OpenStack (accessed 17.2.2025).

[2] L. Chao, Cloud Computing Networking: Theory, Practice, and Development, Boca Raton: CRC Press, 2016.

[3] Denton, J.(2015) Learning OpenStack Networking (Neutron) Second Edition, Birmingham: Packt Publishing Ltd., 2015.

[4] OpenStack (2021), „Openstack Network architecture," 2021. [Online]. Available online: https://docs.openstack.org/arch-design/design-networking.html.

[5] ManageIQ (2024). [Online]. Available online: https://www.manageiq.org/docs/reference/latest/deployment_planning_guide/index.html (accessed 24.3.2025).

[6] COMPUTERWEEKLY. OpenStack Manila: File access storage for the open source cloud Online. 2015 Available at: https://www.computerweekly.com/feature/OpenStack-Manila-File-access-storage-for-the-open-source-cloud. [cited 2025-04-25].

[7] NASER, Mohammed. An Overview of Cloud File Sharing with OpenStack Manila Online. 2021 Available at: https://vexxhost.com/fr/blog/cloud-file-sharing-with-openstack-manila/. [cited 2025-04-25].

[8] OPENSTACK WIKI. Manila Online. Available at: https://wiki.openstack.org/wiki/Manila. [cited 2025-04-25].

[9] OPENSTACK WIKI. Manila/Concepts Online. Available at: https://wiki.openstack.org/wiki/Manila/Concepts. [cited 2025-04-25].

[10] OPENSTACK WIKI. Glossary Online. 2017 Available at: https://docs.openstack.org/manila/latest/reference/glossary.html#term-manila-api. [cited 2025-04-25].

[11] RED HAT DOCUMENTATION. Chapter 12. Hardening the Shared File System (Manila) Online. Available at: https://docs.redhat.com/en/documentation/red_hat_openstack_platform/16.1/html/security_and_hardening_guide/hardening_the_shared_file_system_manila. [cited 2025-04-25].

[12] GITHUB. Charm-manila Online. Available at: https://github.com/openstack/charm-manila/blob/master/TODO.md. [cited 2025-04-25].

[13] OPENSTACK WIKI. Introduction Online. Available at: https://docs.openstack.org/security-guide/shared-file-systems/intro.html. [cited 2025-04-25].

[14] C. B. Hauser and S. Wesner, "Reviewing Cloud Monitoring: Towards Cloud Resource Profiling", *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2018-July, pp. 678–685, 2018, doi: 10.1109/CLOUD.2018.00093.

[15] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey", *Comput. Networks*, vol. 57, no. 9, pp. 2093–2115, 2013, doi: 10.1016/j.comnet.2013.04.001.

[16] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A novel approach to QoS monitoring in the cloud", *Proc. - 1st Int. Conf. Data Compression, Commun. Process. CCP 2011*, pp. 45–51, 2011,

doi: 10.1109/CCP.2011.49.

[17] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, “GMonE: A complete approach to cloud monitoring”, *Futur. Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2026–2040, 2013, doi: 10.1016/j.future.2013.02.011.

[18] Web Solutions Inspire Cloud Computing Software | NASA Spinoff. Accessed: Apr. 15, 2024. [Online]. Available: https://spinoff.nasa.gov/Spinoff2012/it_2.html

[19] Who Is Using OpenStack? Accessed: Apr. 15, 2024. [Online]. Available: https://openmetal.io/resources/blog/who-is-using-openstack/

[20] CSNOG 2023 (16-17 May 2023): Distribuovaná NVME storage v hyperkonvergované DC infrastruktuře bez dopadu na bežný síťový provoz. · Indico’. Accessed: Apr. 15, 2024. [Online]. Available: https://indico.csnog.eu/event/13/contributions/117/

[21] openstack.org. 2023. OpenStack Deployment Tools. [online]. 2023. Available at: https://www.openstack.org/software/project/navigator/deployment-tools

[22] OpenStack Charmers. 2022. Vault. [online]. 2022. Available at: https://charmhub.io/vault

[23] openstack.org. 2022. OpenStack Charms Deployment Guide. [online]. 2022. Available at: https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/index.html

[24] maas.io. 2022. Canonical MAAS. [online]. 2022. Available at: https://maas.io/

[25] juju.is. 2022. Take control of [online]. 2022. Available at: https://juju.is/

[26] T. G. Peter Mell, „The NIST Definition of Cloud,“ The National Institute of Standards and Technology, September 2011. [Online]. Available: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf. [Cited 13 01 2020].

[27] 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

[28] NUTANIX. What is Distributed Storage? Explore Distributed Cloud | Nutanix [online]. [Date 29/3/2024]. Available at: https://www.nutanix.com/info/distributed-storage

[29] 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

[30] 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/

[31] HOSTKEY. What is Distributed Storage? Types and Examples [online]. [Date 21/3/2024]. Available at: https://hostkey.com/blog/53-what-is-distributed-storage/

[32] CLOUDIAN, INC. Distributed Storage: What’s Inside Amazon S3? [online]. [Date 21/3/2024]. Available at: https://cloudian.com/guides/data-backup/distributed-storage/

[33] 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/

[34] 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/

[35] CEPH DOCUMENTATION. Reef [online]. [Date 9/10/2023]. Available at: https://docs.ceph.com/en/reef/

[36] OPENSTACK. 2023. OpenStack Administration Guide [online]. [Date 9/10/2023]. Available at: https://docs.openstack.org/2023.2/admin/

[37] GLUSTER DOCS. Latest documentation [online]. [Date 19/10/2023]. Available at: https://docs.gluster.org/en/latest/

[38] 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

[39] MINIO. MinIO [online]. [Date 19/10/2023]. Available at: https://min.io/

[40] CEPH. Architecture — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/architecture/

[41] CEPH FOUNDATION. Ceph.io — About the foundation [online]. [Date 21/3/2024]. Available at: https://ceph.io/en/foundation/about/

[42] 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/

[43] CEPH AUTHORS AND CONTRIBUTORS. Ceph Releases (general) — Ceph Documentation [online]. [Date 21/3/2024]. Available at: https://docs.ceph.com/en/latest/releases/general/

[44] HG INSIGHTS. Companies Using Ceph, Market Share, Customers and Competitors [online]. [Date 11/3/2024]. Available at: https://discovery.hgdata.com/product/ceph

[45] Canonical Ltd. What is Ceph? | Ubuntu [online]. [Date 18/2/2024]. Available at: https://ubuntu.com/ceph/what-is-ceph

[46] CEPH AUTHORS AND CONTRIBUTORS. Intro to Ceph — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/latest/start/intro/

[47] CEPH. Monitor Config Reference — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/rados/configuration/mon-config-ref/

[48] CEPH. Ceph Manager Daemon — Ceph Documentation [online]. [Date 15/2/2024]. Available at: https://docs.ceph.com/en/reef/mgr/

[49] CEPH. OSD Service — Ceph Documentation [online]. [Date 21/2/2024]. Available at: https://docs.ceph.com/en/reef/cephadm/services/osd/

[50] 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/

[51] 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/

[52] CEPH. MDS Journaling — Ceph Documentation [online]. [Date 21/2/2024]. Available at: https://docs.ceph.com/en/reef/cephfs/mds-journaling/

[53] WEIL, Sage, LENG, 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.

[54] FLYDEAN. 04 Multi Paxos [online]. [Date 22/2/2024]. Available at: https://docs.flydean.com/www.flydean.com/distribution/04-multi-paxos

[55] CEPH AUTHORS AND CONTRIBUTORS. Ceph Storage Cluster — Ceph Documentation [online]. [Date 22/2/2024]. Available at: https://docs.ceph.com/en/reef/rados/

[56] 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.

[57] 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/

[58] 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/

[59] CEPH AUTHORS AND CONTRIBUTORS. Pools — Ceph Documentation [online]. [Date 13/3/2024]. Available at: https://docs.ceph.com/en/latest/rados/operations/pools/

[60] 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/

[61] D'ATRI, Anthony, Vaibhav BHAMBRE, and Karan SINGH. Learning Ceph - Second Edition. Birmingham: Packt Publishing, 2017. ISBN 978-1787127913.

[62] 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/

[63] „Newton install everview“ [Online] [cited March 2021]. Avaible:https://docs.openstack.org/newton/install-guide-rdo/overview.html, 2019.

[64] Anthony Critelli 2020. „Using Keepalived for managing simple failover in clusters“ [Online] [cited April 2021]. Avaible: https://www.redhat.com/sysadmin/ha-cluster-linux

[65] “33+ Cloud Computing Statistics, Facts, and Trends [2020 Updated].” https://techjury.net/blog/cloud-computing-statistics/#gref (accessed Mar. 16, 2021).

[66] “Rackspace Technology | Multicloud Solutions Provider.” https://www.rackspace.com/ (accessed Apr. 13, 2021).