UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

Software defined networking and network functions virtualization technologies

Student: FATSEAS IOANNIS
Registration Number: 18387152

Supervisor

CHARALAMPOS Z. PATRIKAKIS
Professor Dept. of Electrical and Electronics Engineering

ATHENS-EGALEO, September 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

Δικτύωση ορισμένη από λογισμικό και τεχνολογίες εικονικοποίησης δικτυακών λειτουργιών

Φοιτητής: ΦΑΤΣΕΑΣ ΙΩΑΝΝΗΣ
ΑΜ: 18387152

Επιβλέπων Καθηγητής

ΧΑΡΑΛΑΜΠΟΣ ΠΑΤΡΙΚΑΚΗΣ
Καθηγητής στο Τμήμα Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, Σεπτέμβριος 2024

Η Διπλωματική Εργασία έγινε αποδεκτή και βαθμολογήθηκε από την εξής τριμελή επιτροπή:

Χαράλαμπος Πατρικάκης, Καθηγητής	Παναγιώτης Καρκαζής, Αναπληρωτής Καθηγητής	Ελένη Λελίγκου, Καθηγήτρια
Digitally signed by CHARALAMPOS PATRIKAKIS Date: 2024.10.08 00:05:33 +03'00' (Υπογραφή)	PANAGIOTIS KARKAZIS Digitally signed by PANAGIOTIS KARKAZIS Date: 2024.10.07 16:40:26 +03'00' (Υπογραφή)	ELENI AIKATERINI LELIGKOU Digitally signed by ELENI AIKATERINI LELIGKOU Date: 2024.10.07 12:52:51 +03'00' (Υπογραφή)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Ιωάννης Φατσέας του Νικολάου, με αριθμό μητρώου 18387152 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής ΜΗΧΑΝΙΚΩΝ του Τμήματος ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.

Ο Δηλών
Ιωάννης Φατσέας

Abstract

The intersection of virtualization, cloud computing, software-defined networking (SDN) and Network Functions Virtualization (NFV) represents a profound shift in how modern networks are architected, managed and optimized. These technologies break free from the strict boundaries of hardware-centric models, turning the digital ecosystem toward agile, scalable and software-driven infrastructures. This thesis explores these technologies, and its main goal is to help readers understand how they work, which are the components they consist of, and which are their advantages in today's demanding IT world. More emphasis will be given to SDN and NFV, which, when harmonized within virtualized cloud infrastructures, unlock new dimensions of network automation, agility and operational efficiency. This study highlights their critical role in shaping the next generation of fully programmable, software-centric networks, leading the way to a more connected and responsive digital future.

Keywords

Cloud Computing, Virtualization, Virtual Machines, Containers, Kubernetes, Software-Defined Networking, Network Function Virtualization

Περίληψη

Η αλληλεπίδραση της εικονικοποίησης, της νεφοϋπολογιστικής, της δικτύωσης που καθορίζεται από λογισμικό (SDN) και της εικονικοποίησης δικτυακών λειτουργιών (NFV) συνιστά μια ουσιαστική αλλαγή στον τρόπο με τον οποίο σχεδιάζονται, διαχειρίζονται και βελτιστοποιούνται τα σύγχρονα δίκτυα. Οι τεχνολογίες αυτές ξεφεύγουν από τα αυστηρά όρια των μοντέλων που επικεντρώνονται στο hardware, στρέφοντας το ψηφιακό τοπίο προς ευέλικτες, επεκτάσιμες και καθοδηγούμενες από λογισμικό υποδομές. Η παρούσα εργασία μελετά αυτές τις τεχνολογίες και ο βασικός της στόχος είναι να βοηθήσει τους αναγνώστες να κατανοήσουν πως λειτουργούν, ποια είναι τα δομικά στοιχεία από τα οποία αποτελούνται και ποια είναι τα πλεονεκτήματα τους στον σημερινό απαιτητικό κόσμο της πληροφορικής. Περισσότερη έμφαση θα δοθεί στο SDN και στο NFV, τα οποία, όταν εναρμονιστούν στο πλαίσιο εικονιζόμενων υποδομών νέφους, δίνουν νέες δυνατότητες στην αυτοματοποίηση, στην ευελιξία και στη λειτουργική αποδοτικότητα ενός δικτύου. Η μελέτη αυτή αναδεικνύει τον κρίσιμο ρόλο τους στη διαμόρφωση της επόμενης γενιάς πλήρως προγραμματιζόμενων δικτύων με επίκεντρο το λογισμικό, οδηγώντας σε ένα πιο συνδεδεμένο και ευέλικτο ψηφιακό μέλλον.

Λέξεις – κλειδιά

Εικονικοποίηση, Νεφοϋπολογιστική, Δικτύωση καθοριζόμενη από λογισμικό (SDN), Εικονικοποίηση δικτυακών λειτουργιών (NFV)

Contents

List of Figures

Alphabetic Index

API: Application Programming Interface

COTS: Commercial Off the Shelf

CPU: Central Processing Unit

EIGRP: Enhanced Interior Gateway Routing Protocol

ETSI: European Telecommunications Standards Institute

GUI: Graphical User Interface

IaaS: Infrastructure as a Service

IBM: International Business Machines Corporation

IOT: Internet of Things

IT: Information Technology

KVM: Kerne-based Virtual Machine

LTE: Long-Term Evolution

MANO: Management and Orchestration

MPLS: Multiprotocol Label Switching

NBI: Northbound Interface

NFV: Network Function Virtualization

NFVI: NFV Infrastructure

NFVO: NFV Orchestrator

OS: Operating System

OSPF: Open Shortest Path Fist

OVF: Open Virtualization Format

PaaS: Platform as a Service

PC: Personal Computer

SaaS: Software as a Service

SBI: Southbound Interface

SDN: Software Defined Networking

VIM: Virtualized Infrastructure Manager

VMM: Virtual Machine Monitor

VMs: Virtual Machines

VNF: Virtualized Network Function

VNFM: Virtualized Network Manager

INTRODUCTION

In an era defined by the dramatic growth of digital devices, the demand for scalable, flexible and cost-effective network infrastructures has never been greater. Traditional network architectures, which focus on dedicated hardware devices to control network traffic and have high operational costs, have struggled to keep up with the agility required by modern applications and services. As a result, the networking field has changed in many ways due to the synergy of several technologies such as virtualization, containerization, cloud computing, SDN and NFV. Virtualization once limited mainly to computing, has evolved into a fundamental part of modern network architectures. Because of the fact that it creates virtual versions of physical resources that are typically attached to physical hardware, virtualization makes resource utilization more efficient, simplifies management and improves scalability. Containerization a more recent advancement extends the benefits of virtualization by offering lightweight, portable units of software, known as containers, that encapsulate applications and their dependencies. Cloud computing builds on these concepts because it offers on-demand access to computing resources over different places. Each of these technologies have transformed the landscape of IT infrastructure, giving businesses the opportunity to deploy and scale services with exceptional speed and efficiency. At the same time the evolution of SDN has redefined how networks are managed and controlled. By separating the control plane from the data plane, SDN gives in network management a more centralized and programmable approach, making networks much more responsive and adaptable. In a similar way, NFV transforms network services by not using dedicated hardware for specific network functions. This transition from hardware-centric to software-driven architectures reduces both capital and operational costs while enhancing hardware flexibility. As these technologies converge, they present a good opportunity to create dynamic, programmable, and automated networks that will be capable of meeting the huge demands of next-generation services like 5G, 6G and IoT.

Diploma Thesis Subject

This thesis is a survey on modern network architectures such as SDN and NFV. Both of these architectures are driving forces behind the transformation of networking because their technological design focuses on having flexible networks with inbuilt support for large multitenant environments. They provide agility, scalability and cost savings by simplifying network management; that’s why most cloud providers and enterprises embrace them.

Purpose and objectives

The main goal of this thesis is to explore and analyze the network technologies that shape modern networking and computing infrastructures, specifically virtualization, cloud computing, Kubernetes, Software Defined Networking (SDN) and Network Functions Virtualization (NFV). The thesis aims to provide a comprehensive understanding of how these technologies operate individually and synergistically to enhance network flexibility, scalability and efficiency. This study is mainly addressed to students, researchers and professionals in the field of computer science, IT and network engineering who are interested in gaining a deeper understanding of modern network technologies and applying advanced networking concepts in real-world scenarios.

Methodology

This thesis adopts a theoretical and descriptive approach to explore network technologies such as SDN and NFV. Given the nature of research, the study does not involve complex practical experiments or simulations but rather focuses on a comprehensive analysis of the existing body of knowledge. The research is based on a literature review to gather information from books, academic journals, white papers and conference proceedings.

Structure

This thesis is organized into three major chapters. The first chapter provides an overview of virtualization, detailing the role of hypervisors, virtual machines and containers. Then it explores cloud computing, covering various service and deployment models and presenting some of the cloud computing benefits. The second chapter focuses on Software Defined Networking (SDN), explaining its architecture and highlighting its contribution in modern networks. The third chapter examines how NFV works. It discusses the components of the NFV framework, such as Virtual Networks Functions (VNFs), the NFV infrastructure (NFVI) and the Management and Orchestration layer (MANO). This chapter also includes a brief description of OpenStack and its basic components and a small implementation of a private cloud infrastructure.

1 CHAPTER 1st : Virtualization Technology and Cloud Computing

This chapter centers around virtualization technology and cloud computing, which have become indispensable components of modern IT infrastructure and the digital economy. The first part of this chapter begins with an introduction to the history of virtualization, how it works and its benefits. This is followed by an examination of hypervisors, virtual machines, docker containers and their role. A The chapter then shifts to cloud computing, covering the various service models such as IaaS, PaaS and SaaS. Different cloud deployment models are also analyzed, including public, private, hybrid and community clouds, with a focus on their characteristics and use cases. A summary of the advantages of cloud computing will be presented at the end. Overall, this chapter will provide a solid foundation for understanding both virtualization and cloud computing, two key components of the advanced networking technologies that will be discussed in later chapters.

1.1 Virtualization Technology

The field of virtualization has undergone significant advancements over the past few decades and has played a very important role in the development of modern computing environments. The concept of virtualization can be tracked all the way back to the 1960s when IBM (International Business Machines Corporation) introduced the idea to optimize the use of expensive mainframe computers. In this period IBM developed the CP-40 and CP-67 systems. These systems allowed multiple users to run different operating systems simultaneously on one machine. This early form of virtualization, known as time-sharing, enabled more efficient use of computing resources and helped for future developments. Over the years virtualization technology continued to make steps forward with the introduction of virtual machines (VMs) and in 1972, IBM's VM/370 was released. VM/370 allowed users to run isolated instances of multiple operating systems on a single mainframe. Although there were many advancements virtualization was barely used outside mainframe environments because it was too expensive and the technology was very complex [1] [2]. The 1990s was a turning point for virtualization with the introduction of x86-based servers which were more affordable. In 1998, VMware was founded whose technology allowed organizations to consolidate workloads onto fewer servers something that reduced hardware costs dramatically. Virtualization saw a huge advancement in the early 2000s because of the growth of cloud computing. For most service providers virtualization was a key factor, which allowed them to manage and allocate resources across a distributed network of servers. Moreover, at this time, hypervisors became more sophisticated, with the development of both Type 1 and Type 2 hypervisors. In the 2010s, virtualization became more and more popular thanks to the rise of containerization. Platforms like Docker provided a fast and more secure way for developers to deploy, create and manage applications. [3] Today, virtualization remains a foundational technology, helping many cloud computing models and services and has an enduring impact on the IT landscape.

1.1.1 What is Virtualization?

Virtualization is a core technology in today's IT world, making better use of physical hardware by creating virtual versions of servers, networks, storage, and operating systems. It works by using software called a hypervisor, which lets multiple operating systems run on a single physical machine at the same time. The hypervisor takes care of managing these operating systems, known as guest OS, and how they use the system's resources like the CPU, memory and storage. Each virtual machine acts like its own separate computer, with its own application software and guest OS, as illustrated in

Figure 1.1. The job of a hypervisor is to ensure that these VMs stay isolated from each other while running on the same host, so whatever happens in one virtual machine doesn't affect the others. Then there's something called a virtual appliance, which is basically a complete software package ready to be installed on one or more VMs. It's usually delivered in OVF files and makes it easy to deploy applications [4].

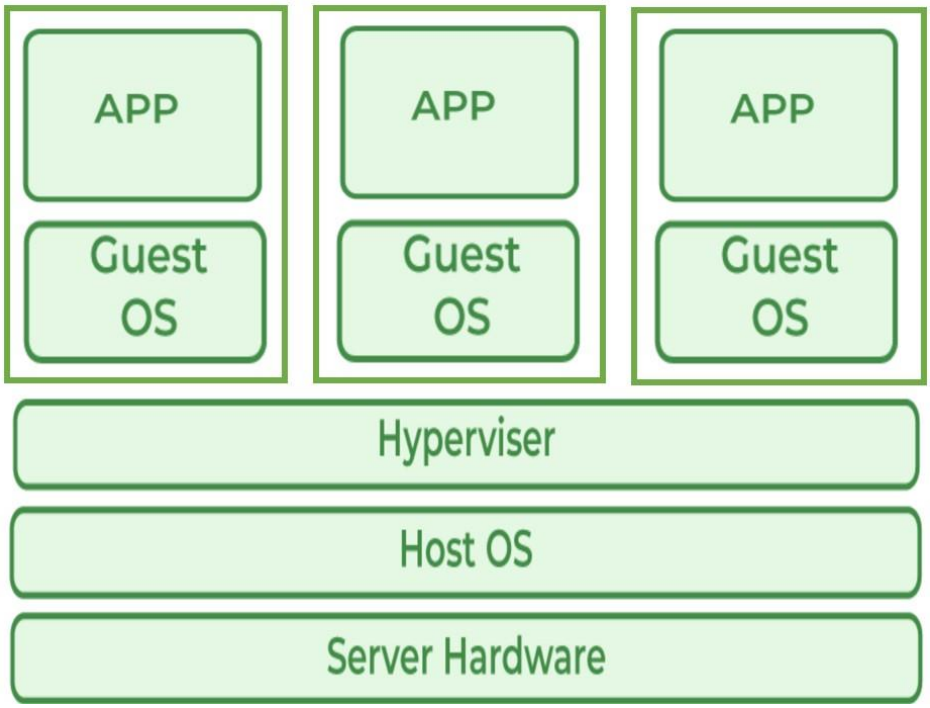

Figure 1.1: Virtualizing Resources

There are several types of virtualization that aim to optimize specific aspects of computing resources. Some of them are:

Network virtualization: According to IBM, "Network virtualization uses software to create a "view" of the network that an administrator can use to manage the network from a single console." This software abstracts the physical network infrastructure, such as routers and switches to create virtual networks that can be managed and controlled by the administrator without having contact with the underlying physical hardware. As a result, the whole network management becomes more simplified [5]. Some examples of network virtualization are software-defined networking and network function virtualization, which will be examined in depth in the following chapters.

Server virtualization: This type of virtualization divides the central physical server into multiple virtual servers. Each one of these servers can operate its own operating system and applications. Server virtualization is widely used in the IT infrastructure and the logic behind it, is the minimization of costs by increasing the utilization of existing resources.

Desktop virtualization: It allows users to access their desktop virtually, by a different device from anywhere in the world. It is very useful for organizations that need to manage a big number of desktops or maybe want to support remote work.

Storage virtualization: Storage virtualization makes all the storage devices on the network accessible and manageable as if they were a single storage device. In particular, storage virtualization places all storage devices into a virtual shared pool, from which they can then be allocated to any virtual machine on the network as required [5].

Application virtualization: This is a kind of virtualization in which applications run in isolated virtual environments separate from the operating system. This isolation allows applications to be deployed on any compatible system without affecting other native applications.

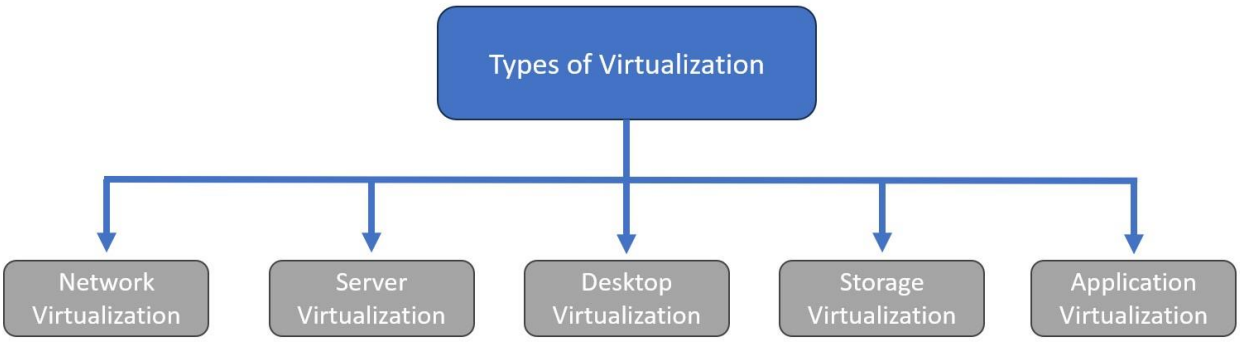

Figure 1.2: Types of Virtualization

1.1.2 Hypervisors

One of the most important components of virtualization is the hypervisor. A hypervisor also known as virtual machine monitor (VMM), is a software layer that is responsible for the creation, execution and management of virtual machines on a host machine. More specifically it allocates hardware resources such as CPU, memory and storage to the VMs in such a way as to ensure that each VM operates efficiently and independently. A hypervisor is located between the hardware and the virtual machines as shown in Figure 1.3 [6].

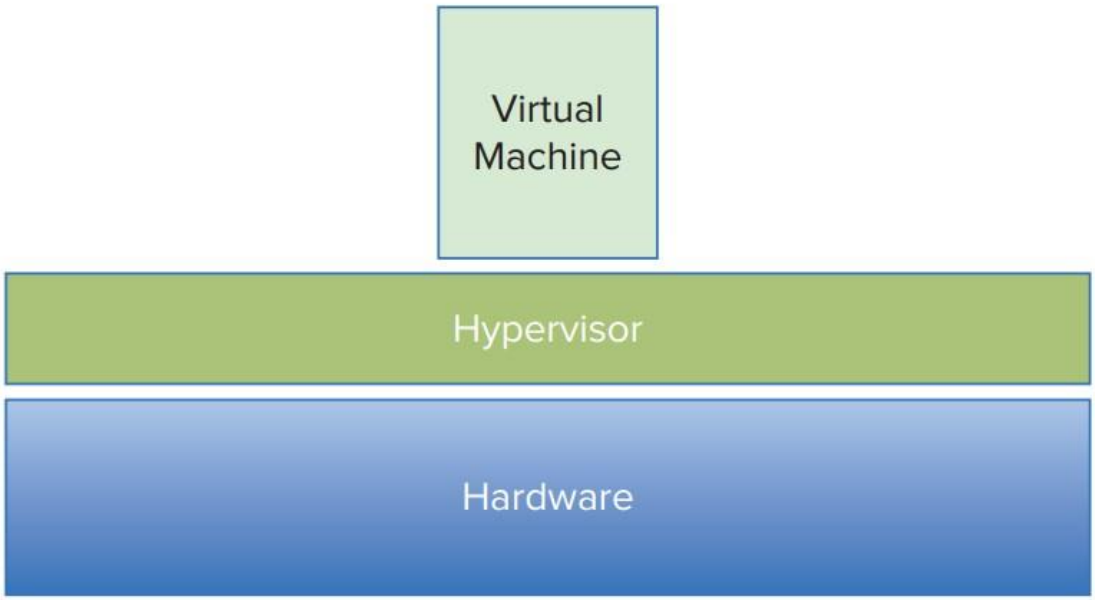

Figure 1.3: Where the hypervisor is located [6]

Two types of hypervisors exist:

Type 1 hypervisors (often referred as bare metal) are installed directly on the physical hardware of a host machine which means that there isn't intermediate layer between the physical hardware and the

hypervisor. As a result, type 1 are more efficient than the type 2 hypervisors, providing much better performance. This also makes them very secure as there is no intermediary between them and the CPU that an attacker could possibly compromise. Thus, they are more useful in enterprises, data centers and cloud computing platforms where high performance, resource efficiency and scalability are crucial. Some well-known examples of the type 1 hypervisors are VMware vSphere/ESXi, KVM, Microsoft Hyper-V, Citrix XenServer and Red Hat Enterprise Virtualization [6].

Type 2 hypervisors (frequently referred as hosted hypervisors) run on top of a conventional operating system. Unlike type 1 hypervisors, which have a direct interface with the underlying hardware, type 2 hypervisors are installed as applications within a host OS, and they rely on the host operating system to manage hardware resources. Their installation process is very easy and quick (through a GUI), and once the hypervisor is installed, it runs as a regular application in the system [6]. A few paradigms of the type 2 hypervisors are Oracle's VirtualBox, VMware Workstation and Microsoft virtual PC.

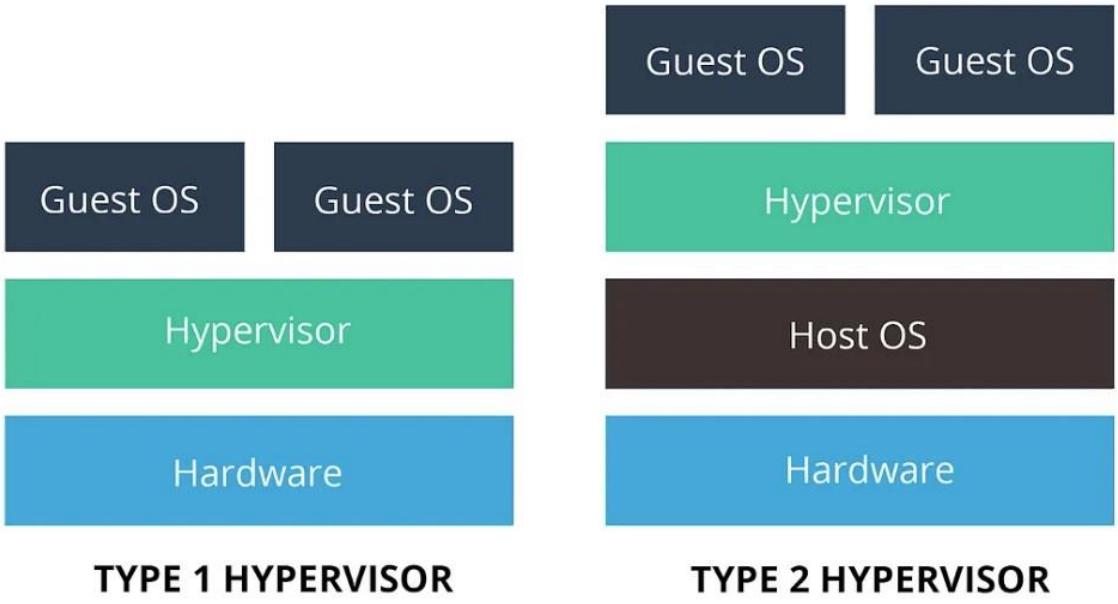

Figure 1.4: Two types of hypervisors [7]

1.1.3 Virtual Machines

A virtual machine (VM) could be seen as a physical server because it can run an operating system and has resources for applications to use. But, in contrast to a physical server, which runs only one operating system at a time, a big number of VMs can operate on a single physical server, each one of them with a different OS and applications. If someone would like to understand VMs better, he could say that VMs are just a collection of files that define its virtual hardware and disk space. The configuration file specifies the virtual resources like processor, memory and storage. A VM could also be compared to a blank server, ready to be set up with whatever software and virtual hardware configuration is needed [6].

VMs exist both as files that define their configuration and as active instances in memory when they're running. Using a VM could possibly feel like working with a physical server, making everyday tasks

like managing applications and adjusting settings, but the real advantage of VMs is their file-based nature, which makes them easy to manage, move and replicate, just like any other data file on a computer. For example, even an amateur PC user can easily transfer a VM to another location, duplicate it or create backups in the same way he would do with a simple document file. As can be seen, this flexibility and simplicity make VMs extremely useful [6]. However, there are some cons to using VMs, like the fact that VMs need time to build and regenerate. Also, VMs take up a lot of storage space. These problems can be overcome by other technologies, such as containers, which will be examined in a later section.

1.1.3.1 Virtual Machine Cloning

Before server virtualization, provisioning a physical server required a lot of time, money and a lot of effort. Ordering a server could take weeks, and once it arrived, administrators had to install the operating system, apply updates, configure storage, install tools and connect it to the network, a process that could take several days or even longer. One of the benefits of virtual machines is that this process is much faster. An existing server can be cloned by just copying its files. While cloning might take minutes or a few hours depending on various factors, it's still significantly quicker and cheaper than setting up a physical server and of course it requires less time and manpower [6].

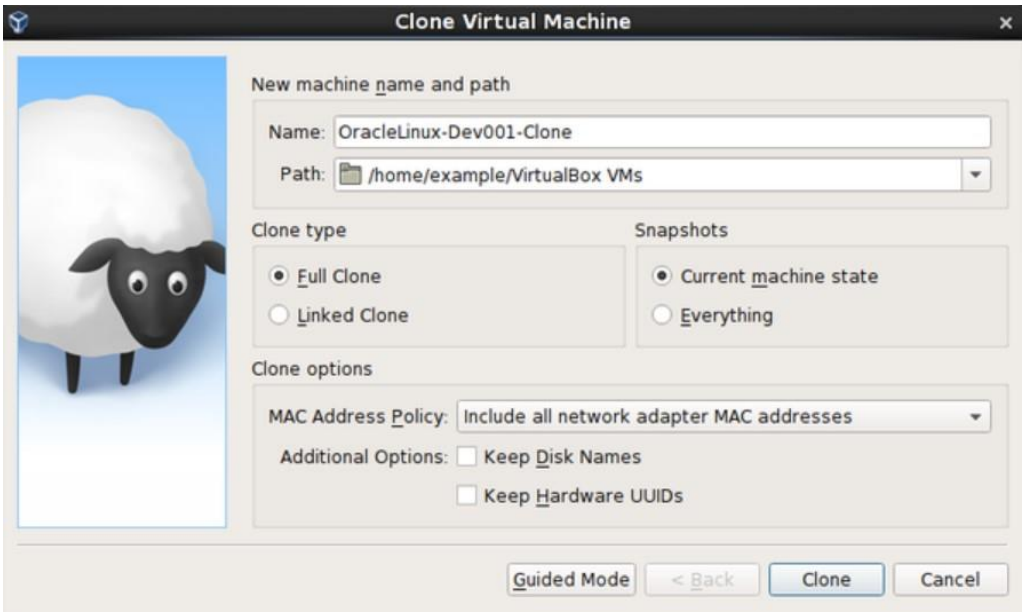

Figure 1.5: Cloning a Virtual Machine [8]

1.1.3.2 Virtual Machine Templates

VM templates are preconfigured VMs used to quicky create fully set up servers. In addition to clones, templates are not running and must be converted back into a VM for updates or changes. When the updates are applied, the VM is converted again back into a template. In the same way that clones do, each VM created from a template requires a unique identity. Creating a VM from a template is much faster than setting up a physical server. Templates can deliver not only operating systems but also preinstalled applications, allowing users to quickly deploy ready-to-use virtual machines. That's the reason why some vendors even provide applications as downloadable VM templates for easy deployment [6].

1.1.4 Docker Containers

UNIX-style operating systems originally used the concept of a "jail" to restrict a program's access to certain protected resources. Over time, this evolved into the idea of containers, whose purpose was to isolate processes from other resources. While containers have been valuable for security in most cases, manually creating them is a complex and prone to errors procedure. So, a solution to this was Docker which simplifies this process by automatically building containers following the best practices, enhancing security and reducing costs. By using Docker, users can stay up to date with container technology, saving a lot of their time, without the need to have deep technical knowledge [9]. To the question, is the container actually virtualization? The answer in no! If an organization is not using Docker, it relies on VMs for isolation which involves running a complete operating system and requires significant time and resources. In case of the Docker, containers run directly on the host's Linux kernel, avoiding the need for virtual hardware and extra operating systems layers. As a result, they don't consume resources by running redundant systems.

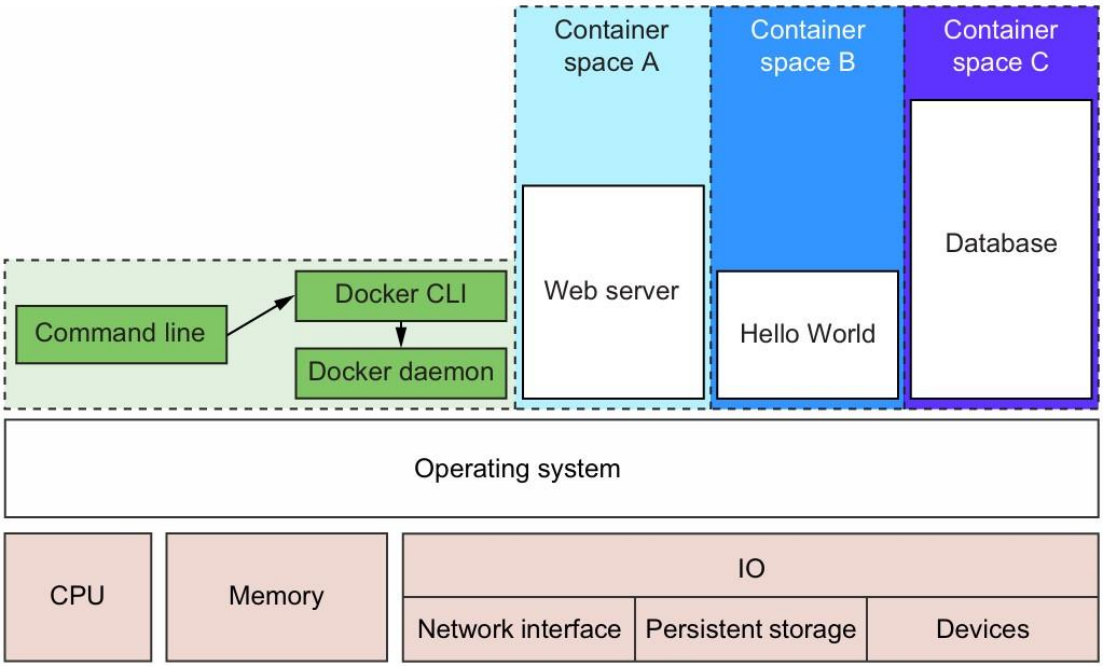

Figure 1.6: Docker running three containers on a basic Linux computer system [9].

When using Docker, two main programs run on a machine. The Docker daemon, which is always running, and the Docker CLI, which users use to give the necessary commands. As we can see in Figure 1.6, Docker containers are child processes of the Docker daemon, each running in its isolated memory [9].

If we wanted to use an analogy to define Docker container, we could say that Docker container is like a shipping container for applications, where inside it there are all the things we will need to run an application. Docker's job is to run, copy and distribute these containers in the same way shipping containers are handled in transport. The "box" that stores an application in Docker is called an image, which is a snapshot of all the files required for a container. We can create multiple containers from the same image, but each container will be isolated and will not share filesystem changes with others. Docker images are the core units for software distribution in the Docker ecosystem and Docker uses registries and indexes to manage and distribute these images [9].

So, what are the difficulties that someone might face and Docker solves them? First of all, the process of installing and managing software can be a complex one due to the large variety of installation methods and the potential conflicts between applications. It sometimes requires very careful planning and regular updates to ensure that applications are compatible and safe. All of these difficulties become even more when additional software is added and the risk for errors and security problems increases. Someone could say that these challenges are common and normal, but wouldn't it be better if there was a way to avoid them and save valuable time? Docker does exactly that because it simplifies these processes by making installation and management more straightforward [9].

1.1.5 Benefits of Virtualization

- Reduced Hardware Costs: Virtualization allows organizations to reduce hardware purchases, power consumption and maintenance costs by combining multiple virtual machines onto one physical server.
- Flexibility: Virtual machines can be created and modified very easily. As a result, they allow dynamic allocation of resources based on the needs of each costumer.
- Increased Utilization: Virtual machines share the available computing resources of a physical server in an efficient way.
- Isolation and Security: Virtual machines operate independently and are isolated from each other, so in case of a threat or failure there is no impact on other VMs or the overall system.
- Simplified Management and Deployment: Virtual environments can be managed through centralized management tools that provide features such as live migration and cloning. Also new VMs can be deployed very fast by cloning existing VMs or using pre-configured templates.

1.2 Kubernetes

As Google's infrastructure kept growing, managing thousands of deployable components became increasingly difficult. In an attempt to handle the complexity of running hundreds of thousands of servers, Google developed Borg and after a few years, Omega, which streamlined software deployment and maximized infrastructure efficiency. At that time, even small gains in resource utilization meant huge cost savings for a company of that scale. After years of using these systems internally, Google released Kubernetes in 2014, an open-source platform built on their experience with Borg and Omega to help other companies manage large-scale deployments [10]. Kubernetes is a software platform that simplifies the process of deploying and managing containerized applications across many servers, without requiring from users to know the internal details of each app. In this way, it ensures that applications run independently of each other, which is extremely important, especially for the cloud providers, which want to maximize hardware utilization and at the same time want to maintain isolation between apps [10].

Kubernetes is made of a master node and multiple worker nodes, in which developers submit their apps to the master and Kubernetes delivers them to the cluster. Kubernetes handles infrastructure tasks like service discovery, scaling and load balancing, allowing developers to emphasize on

building new features for their projects. Furthermore, Kubernetes makes sure that apps can communicate with each other no matter where they're deployed and at the same time optimizes resource usage by dynamically relocating apps as needed.

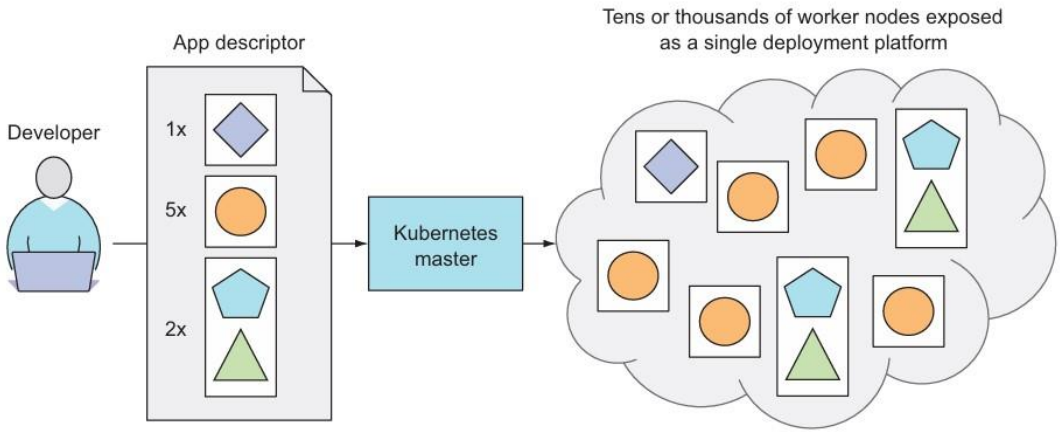

Figure 1.7: How a Kubernetes system looks like [10]

Two different nodes exist that a Kubernetes cluster can split on. The first is the master node that hosts the Control Plane, whose basic job is to control the cluster's operations. The most important components of the Control plane are the Kubernetes API server for communication purposes, the Scheduler for assigning tasks to worker nodes, the Controller Manager for cluster-level tasks like replication and failure handling, and etcd, which stores the cluster's configuration. The second type is the worker nodes, whose job is to execute the containerized programs. They rely on components such as Docker, the Kubelet, which oversees containers and kube-proxy, which is responsible for load-balancing network traffic between app parts [10].

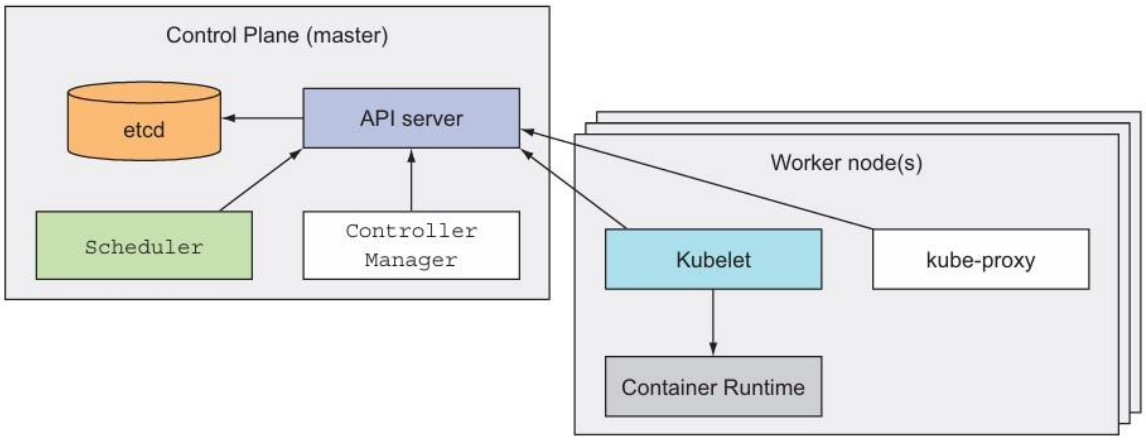

Figure 1.8: Architecture of Kubernetes cluster [10]

If a user wants to run an application on Kubernetes, he starts by packaging it into container images. After that he uploads them to an image registry and then he provides a description to the Kubernetes API. The description that he provided has details about what the application's components are and what the connection is between them. It also contains information about where they should run and how many replicas to create. Kubernetes then uses the Scheduler to assign containers to worker nodes and the Kubelet pulls (on each node) the necessary images and runs them as illustrated in Figure 1.9. When the application is eventually deployed, Kubernetes makes sure that it matches user's

description. If something breaks (for example a container crashes or a node goes down), Kubernetes automatically restarts the container or moves it to another node. Another feature of Kubernetes is that the user can easily adapt the app by just changing the number of replicas and if he wants, he also has the option to let Kubernetes do it based on real-time statistics such as network traffic or CPU load. Finally, to keep things simple for users, Kubernetes assigns services with a static IP, so even if containers are moved or replicated, users can always connect to them [10].

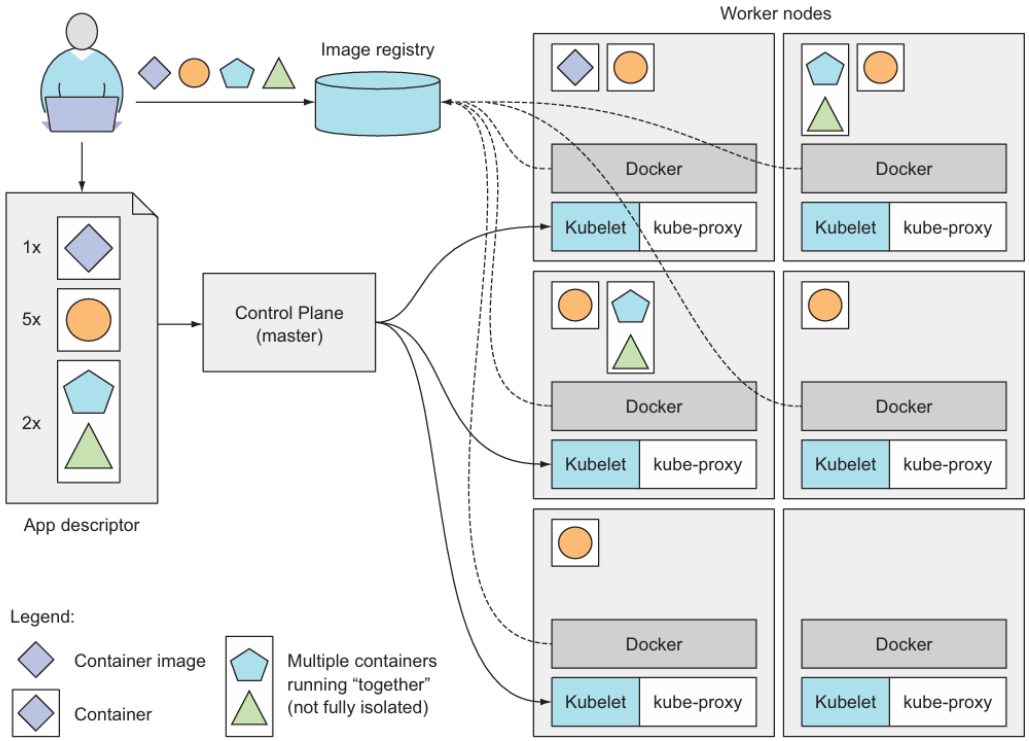

Figure 1.9: The architecture of Kubernetes [10]

1.3 Cloud Computing

Cloud computing could be defined as the delivery of computing services such as physical or virtual servers, data storage, networking, software and more over the Web without the need to manage or own the underlying infrastructure [11]. Cloud computing follows a pay-as-you-go model. Once someone is connected to the cloud, he can access as many computing resources as he needs and be billed for what he actually uses. This condition lets the cloud providers to share resources among several customers, making it more efficient and often cheaper than everyone having their own separate systems. Similar to how most people buy electricity from a power company instead of generating it on their own, using the cloud is usually an easier and cheaper solution for most of them than maintaining private servers at their workplaces or houses. The whole idea is just to outsource these services to a provider who can deliver them more efficiently [4].

1.3.1 Cloud Computing service models

- **Infrastructure as a Service (IaaS):**
  It is a cloud computing service model that provides on-demand access to customers on cloud-hosted computing hardware, such as servers, storage and network resources. Customers can manage the infrastructure from their own PC (via the internet) by paying a subscription

without the need for physical hardware investment and maintenance. In this way, the need for upfront capital expenditures on hardware is reduced significantly, allowing customers to save money. Google Cloud, Microsoft Azure and Amazon Web Services are some of the biggest cloud services providers in the world [12].

- **Platform as a service (PaaS):**
  It is a type of cloud computing services that provides a cloud-based platform allowing customers to develop, manage and run applications. Customers do not handle or control the underlying cloud infrastructure, something for which the cloud provider is responsible. As a result, users don't have to install physical hardware and software to develop or run a new application. In most cases, users just access the PaaS through a GUI, where they can build, test and deploy their applications very quickly and at a low cost. The Microsoft Azure App Services, Amazon Web Services and Google App Engine are some of the leading PaaS providers [12].

- **Software as a service (SaaS):**
  It is a form of cloud computing where the provider delivers software applications over the internet to clients and manages all the hardware and software resources used by the application. All software upgrades and updates, as well as security and performance, are the responsibility of the provider. Clients simply access SaaS applications through a web browser from any device that is connected to the internet. In our days, most people use some form of SaaS in their lives. Some of the most basic everyday SaaS applications are social media like Instagram, email services and streaming services such as YouTube and Netflix [12].

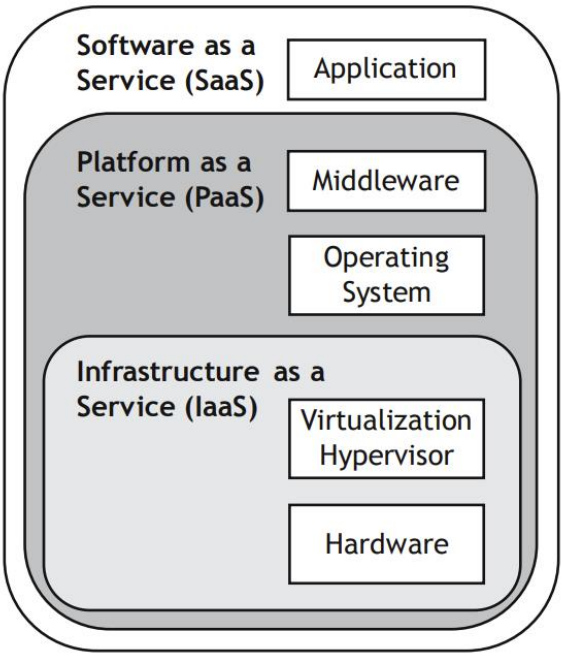

Figure 1.10: Cloud computing service models [4]

1.3.2 Cloud Deployments Models

Cloud deployment models define how cloud services become available to users, who controls the servers, and the location where these servers are hosted. Also, it specifies what changes could be made and what the cloud infrastructure is going to look like. Companies have to make the right choice

of a cloud model if they want to have a successful cloud implementation, so a careful study and an accurate selection of a model are necessary in order to avoid a serious risk of failure in the implementation. According to previous researches on cloud computing, the cloud deployment models have been categorized as the following [13].

- **Public cloud:**
  The public cloud is a cloud deployment model in which the infrastructure services are open to the general public. These services are provided over the internet and are owned and hosted by third-party cloud service providers such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP). It is a pay-as-you-go service, which means that customers expenses are based on how much of the product they have actually used. So, it is appropriate for enterprises that need quick access to specific amounts of resources. The zero-setup cost and the absence of infrastructure management are some additional advantages of the public cloud. However, since there are a lot of users who use the same resources, it raises big security and privacy concerns [14].

- **Private cloud:**
  In contrast with the public cloud, the private cloud offers exclusive access to a single user or organization and it's not accessible by the public. Its services are maintained on a private network and it's not mandatory for customers to share their hardware with someone else. It can be managed and hosted either by the user or by a third-party provider. One of the main advantages of private cloud is the enhanced security measures. For many companies, the security of their data is of major importance, so that's why they prefer a private cloud model that is protected by powerful firewalls, plenty of security tools and in most cases by a whole IT department [14].

- **Hybrid Cloud:**
  The hybrid cloud is a cloud deployment model that combines two or more cloud architectures. It provides great flexibility because businesses can design their own custom solutions based on their specific needs. But considering the complexity of setting up a hybrid cloud, this model is more useful for businesses that need to separate their private and sensitive data. For instance, hospitals can store their sensitive data on a private cloud and at the same time store other less important data on a public cloud [15].

- **Community Cloud:**
  The community cloud operates in such a way that services are shared among a group of organizations. These organizations have common goals and shared concerns, such as policies and security requirements. It is managed and operated either by one or more organizations in the community or a third-party provider and it offers many benefits, such as better security, low running costs and collaboration between multiple businesses. But, if an organization wants to make any changes, it is not easy to implement them as all data and resources are shared between them and thus there will be consequences for the other organizations [14].

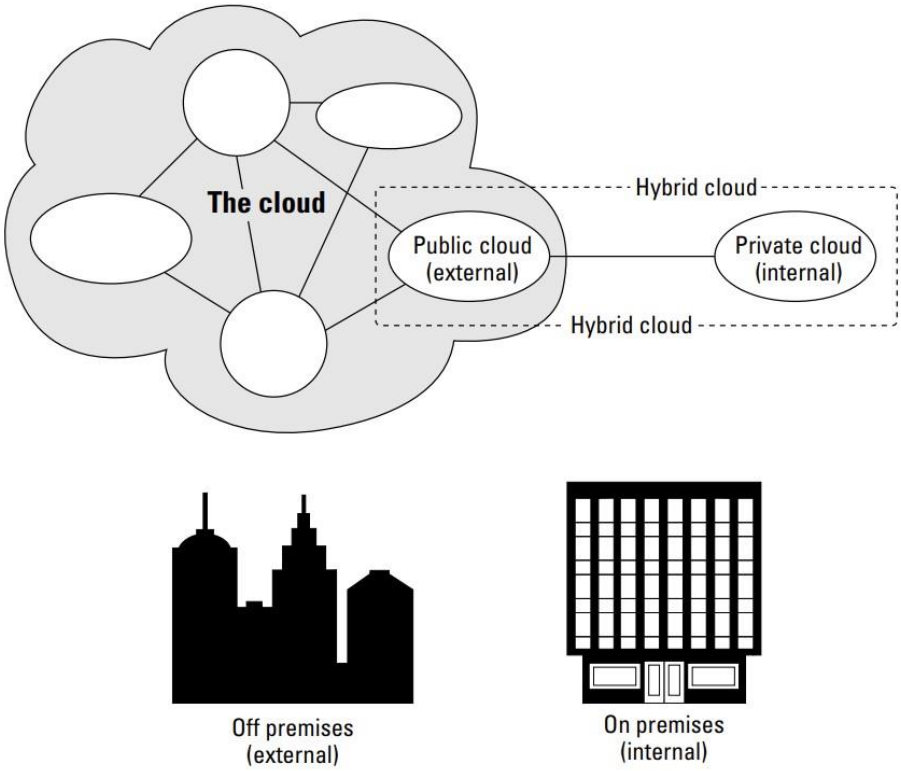

Figure 1.11: Deployment locations for different cloud types [16]

1.3.3 Advantages of Cloud Computing

- Reduced costs: Cloud computing is a pay-as-you-go activity which means that users have to pay only for the resources they use. As a result, the total capital expenditures and the costs for maintaining physical hardware are reduced [4].

- Data loss prevention: Cloud service providers offer disaster recovery and backup options. So, in case of an emergency or hardware malfunction, the data that are stored in the cloud and not locally are safe [17].

- Rapid elasticity: Cloud services offer on-demand access to resources and for that reason they have the ability to scale up or down according to their clients' needs [4].

- Increased Effectiveness: Cloud computing allows cloud consumers to prioritize their limited resources on developing solutions to enterprise issues rather than investing in the maintenance and deployment of computing infrastructure, thereby enhancing the effectiveness of the organization [4].

- Energy Efficiency: Cloud service providers have the ability to share their storage, networking and data center resources effectively across multiple cloud customers which means that the total amount of power consumption will be reduced [4].

2 CHAPTER 2nd : Software Defined Networking

This chapter provides a deep dive into software-defined networking (SDN), a revolutionary technological approach on how networks are managed that connects applications, network devices and services, enabling centralized and programmable network control. SDN represents a change from traditional networking, making it feasible to program network devices and do them scalable and flexible by providing dynamic management through software applications. By centralizing the network's intelligence in a software controller, the SDN controller, which is reside between network devices and applications, SDN simplifies network operations and facilitates automated network management, making it a critical technology for data centers and cloud environments [18]. This chapter focuses on SDN architecture, OpenFlow, SDN controllers and SDN benefits. It also presents information about the SDN in 5G and 6G networks.

2.1 How SDN Works?

As we said, network control needs to be centralized, moving away from the traditional model where each device operates as an isolated unit. This shift to centralized control makes simpler the processes of network discovery, management and connectivity, all of which are very complicated in large conventional networks. So, in case of a change or whenever a new application is added, by having centralized control, the entire network becomes programmable and there is no need to manually configure every device [19].

Furthermore, a distinct separation between the network OS and the applications running on it is very essential and this is accomplished through an application programming interface (API). This separation allows third parties to develop applications easily and rapidly, leading to a significant pace of innovation. This marks a significant shift in networking because applications now need to interact with network control systems far more frequently than they did a couple of years ago. All of these concepts define SDN as we know it today [19].

SDN could be seen as a flexible system that centralizes network control. With the help of an API, this system allows different applications to manage the forwarding plane of network devices. Essentially, SDN removes the decision-making intelligence from the network devices to a central controller. But to understand this better, in Figure 2.1, a traditional network is presented.

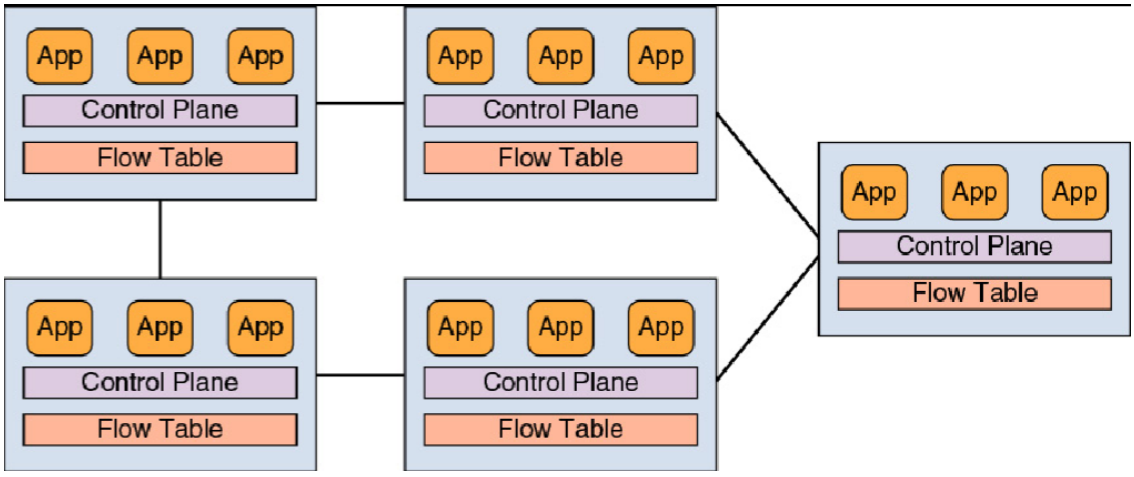

Figure 2.1: A traditional network [19]

In a typical network setup, every switch and router operates independently and runs its own applications, such as VoIP, monitoring and load balancing. Each of these devices needs to be configured separately, and as data travels through the network, each device makes routing decisions based on its local settings. This means that if there are any changes to the flows or applications, every device must be updated individually, which can be a complicated and a time-consuming process [19].

Now, let's take a look at the SDN approach (Figure 2.2). In this model, the intelligence and applications that traditionally reside within the switches and routers are moved to a centralized controller. This controller serves as the command center for the network, which makes it programmable and much easier to manage. Applications communicate with this central controller, which then directs their functionalities across the entire network. The controller also oversees the traffic flows, using flow tables that it updates and distributes to each network device. These flow tables collect detailed data and send it back to the central controller, making the resolution of possible issues a lot easier [19].

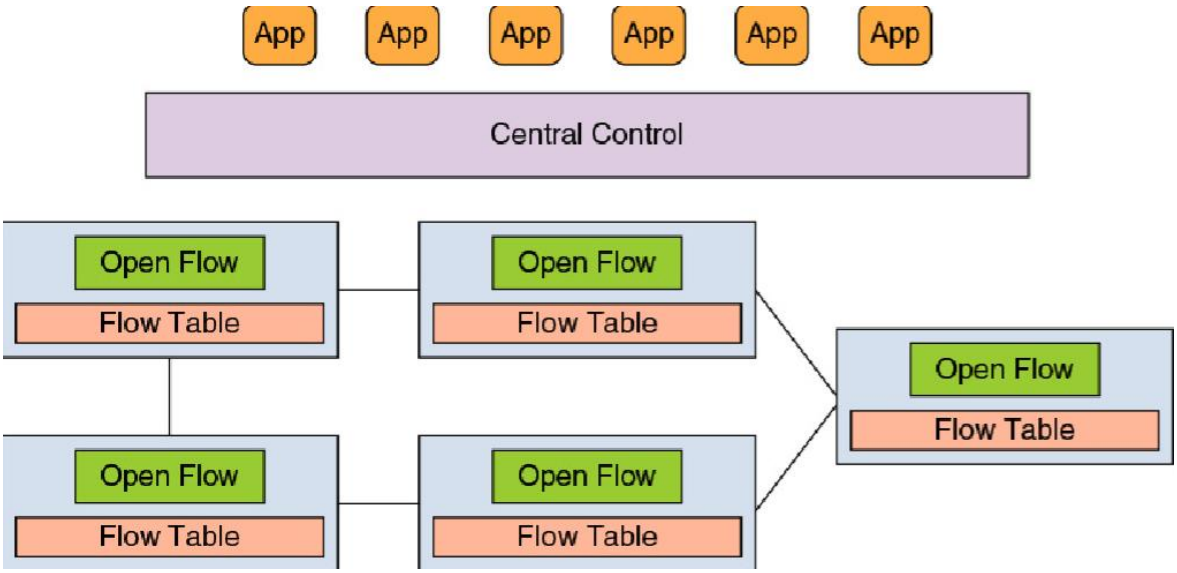

Figure 2.2: Software-defined Network with a centralized controller [19]

2.2 Basic Architecture of SDN

In order to overcome the drawbacks of classical networking architectures, software-defined networking architecture has arisen as a solution. The architecture of SDN has many differences from the traditional networking architectures because its purpose is to offer a more dynamic, programmable and flexible approach to managing network resources. SDN achieve to do this by dividing the control plane from the data plane, which means that it lets administrators utilize network resources, makes resource provisioning simpler and enables programmability through software [20].

The SDN architecture can be broken down into three primary layers as shown in Figure 2.3, the Control Layer, the Data Layer and the Application Layer.

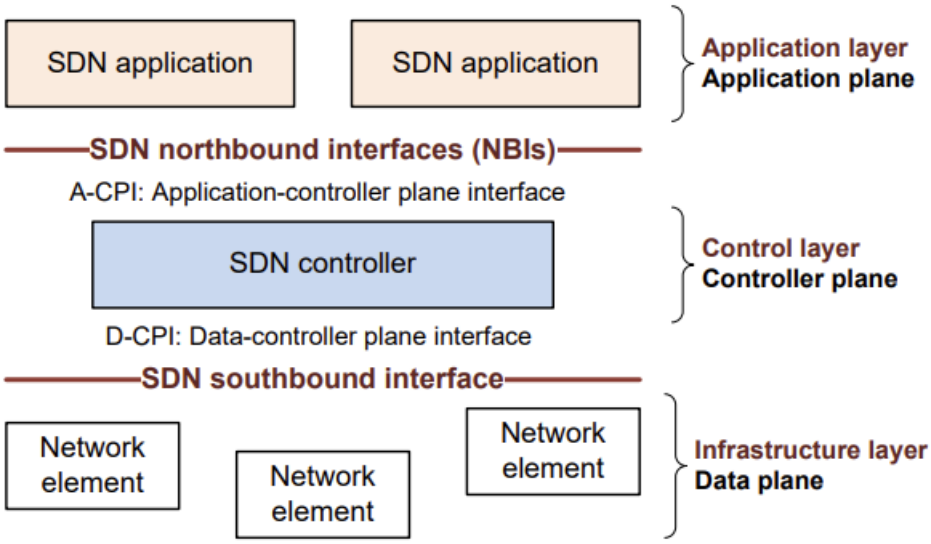

Figure 2.3: SDN architecture [21]

Application Layer: As the name indicates, this layer contains a lot of network applications (such as VoIP prioritization, firewalls, QoS management tools, etc.) and scripts that interact with the control layer through its northbound interface (NBI), requesting application resources according to needs [19].

Control Layer: The control layer is the central layer of the SDN architecture and it includes the SDN controller, which is making decisions about how packets should be routed and handled. Although this is a separate concept from the control plane, this layer also contains the centralized control plane of the network. Moreover, this layer connects the application and the infrastructure layer.

Infrastructure Layer: Finally, there is the infrastructure layer, which includes the actual network devices (switches and routers) that are responsible for forwarding messages across the network. These devices route network traffic to where it needs to go according to the controller's rules and policies.

These three layers communicate with each other via APIs. APIs are software interfaces that allow two applications to communicate with each other and they are necessary not only for network automation but for all kinds of applications. In SDN architecture, REST APIs are usually used to communicate between applications and the SDN controller via the Northbound Interface (NBI). NETCONF and RESTCONF are two of the APIs that are used for the communication between the SDN controller and the network devices via the Southbound Interface (SBI).

2.3 Openflow

Network engineers face difficulties with virtualizing routers and switches because of the fact that these devices are closed proprietary systems. This means that each device’s internal functions, such as traffic management and data movement, are not transparent, forcing engineers to configure each one individually. This process can be quite difficult when it comes to large networks that are constantly changing, where updates are slow and the network's response to changes is delayed. The solution to improving network management is to split the control and data planes, enabling centralized control through an API. This approach was successfully used as a solution to the problem of enterprise WiFi, where a centralized controller allowed administrators to manage many access points without manually updating each one. As a result, the access points were made simpler, with their basic role

being packet forwarding, while all the complex functions were moved to the controller. Now, networks that are moving towards SDN are adopting this concept with OpenFlow becoming the primary protocol for centralized management.

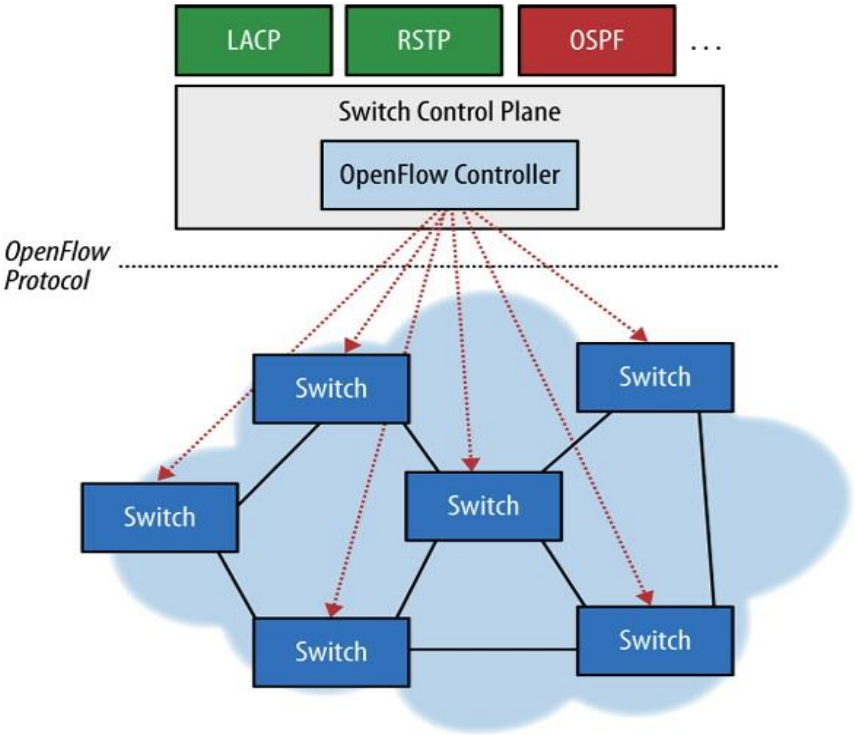

Figure 2.4: OpenFlow architecture [22]

Figure 2.4 demonstrates the OpenFlow architecture and shows how it works.

In OpenFlow, a flow table is structured around three fields: a packet header that specifies the characteristics of the flow, the action that need to be taken for every one of the flows, and statistics that monitor the flow's packets and bytes [19]. OpenFlow switches must support four basic tasks:

- Forward specified packets to a designated port which ensures that the packets are directed along a defined path through the network.
- Encapsulate and send specified packets to the controller which is used for the first packet of a new flow. This permits the controller to decide whether to add the flow to the table or not.
- Drop specified packet flow which is crucial for security purposes.
- Process packets.

There are some valid concerns about the reliability, scalability and performance of controllers that dynamically manage flows. Despite these concerns, testing has shown that even a basic setup can handle thousands of new flows per second, which is sufficient enough for large networks like those in colleges. That's why many switch vendors adopted OpenFlow into their devices because it provides a high level of control to users over their networks. OpenFlow also allows users to create customized flows and optimize traffic paths based on real-time conditions, in contrast to traditional routing protocols, which often ignore bandwidth and congestion. This last feature is also found in Multiprotocol Label Switching (MPLS) but because MPLS is a Layer 3 protocol, it is limited by the vendor's capabilities, making OpenFlow (which operates a Layer 2) particularly useful in data centers where MPLS isn't applicable [19].

2.4 SDN Controllers

The SDN controller could be characterized as the central brain of a software-defined network. It manages the communication between the application layer and the physical network devices and has a complete overview of all network devices, their connections and the optimal paths between them. This centralized view permits the controller to quickly and intelligently manage traffic flows and respond to failures more efficiently than a traditional network. An idealized controller is shown in Figure 2.5. Unlike conventional routing protocols such as EIGRP and OSPF, which need to detect a failure, announce it, run algorithms, and update routing tables (which takes time), the SDN controller has already a broad overview of all possible routes. As a result, it can switch to an alternative path in a very short period of time without the need to recalculate, which is something very important as the network becomes faster and more seamless [19]. However, centralized controllers may have to deal with performance issues and sometimes become bottlenecks. One way to avoid this is to deploy many controllers, which will serve as peers and backups. It is very important for the controllers to keep a consistent global view of the network otherwise possible false data may lead to poor network decisions. A common solution involves using publish systems like HyperFlow, which allows controllers to publish updates when changes occur, keeping all controllers synchronized. Another approach, "SDNi", facilitates communication between controllers across SDN domains, giving them the opportunity to share network status and coordinate decision-making procedures [23].

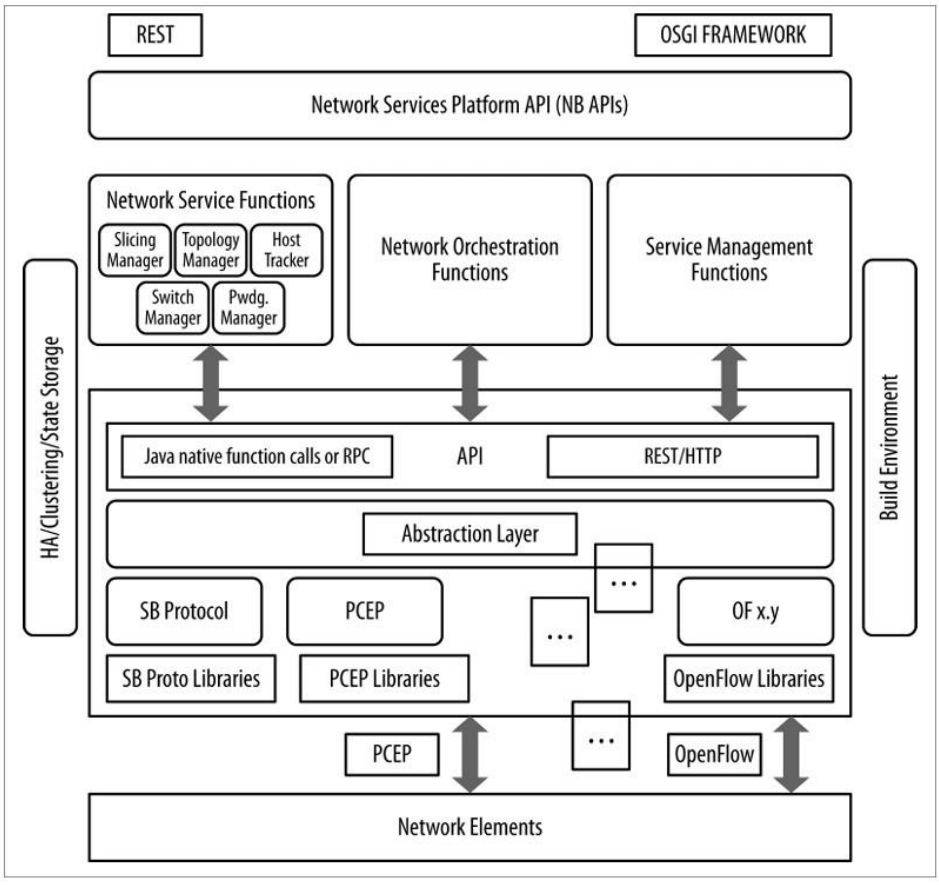

Figure 2.5: Idealized SDN controller [22]

There are plenty of SDN controllers on the market, some of which are developed and maintained by big companies, while others are open-source. Here are some worth mentioning examples:

OpenDaylight: It is among the most widely used open-source SDN controllers. It is written in Java and it supports many southbound APIs, including OpenFlow, NETCONF and BGP-LS.

ONOS: It is an open-source SDN controller that was created by the Open Networking Foundation. It is programmed in Java and it supports protocols such as OpenFlow and NETCONF.

Ryu: It is a component-based controller framework written in Python that supports multiple protocols, including OpenFlow, NETCONF and OF-config.

Floodlight: It is an open-source SDN controller written in Java that supports OpenFlow protocols 1.0 through 1.5.

Cisco Application Policy Infrastructure Controller (APIC): It is Cisco's ACI main centralized controller and it is designed to provide automation and management especially for data centers.

VMware NSX: It is a distributed control system that can manage virtual networks and overlay transport tunnels over an existing infrastructure [19].

2.5 SDN in 5G and 6G

5G and 6G technologies are designed to solve key issues in both business and network performance, improving things like user experience, data handling and connectivity. These technologies are designed to handle high traffic in busy areas, provide fast and reliable service and support applications like smart cities and industrial control systems. For instance, they will enable large-scale data collection in smart cities while they use very little power and will offer ultra-reliable, low-latency connections for things like self-driving cars and manufacturing. To make all these possible, the network architecture needs to be flexible and adaptable and to be able to adjust according to the specific needs of different scenarios. Moreover, it needs to support seamless coordination between different network technologies (like RAT) and to offer broader coverage and greater network capacity while remaining backward compatible with older systems [24].

If we want to deal with the significant challenges of 5G and 6G, some of which are broad coverage, low latency and low power connections, new technical solutions are needed. For example, for wide area coverage, we need to use lower frequency bands and improve how efficiently the spectrum is used. Furthermore, large-scale antenna arrays, combined with better multi-user access methods, can boost network performance while in high-traffic areas, techniques such as ultra-dense networking and full-spectrum access help increase the speed of data and the overall capacity. When it comes to lower power, large-scale connections such as those needed for smart cities and IoT devices, the main focus is on improving device connectivity and cutting power consumption. Finally, for low-latency, high-reliability scenarios, the minimization of transmission delays and the optimization of network signaling is the key [24].

In earlier wireless network designs, managing functions like control and data processing was very complex and it required base stations, service networks and gateways to work together. This made optimizing network very difficult, especially if we think that there wasn't a central controller and that manufacturers had their own unique systems. So, operators had left with limited flexibility and

minimum innovation; that's why the concept of software-defined networking was introduced. In 5G/6G networks, the architecture consists of three main layers: control accessing and forwarding. The control layer centralizes network management, while in the meantime the access and forwarding layer handle user data across many wireless technologies. Virtualization allows for dynamic allocation of resources, making the network more efficient and responsive, with high reliability and low latency for users. The control layer plays a crucial role in managing the entire network, with key modules handling tasks such as radio resource management, mobility (trucking users and managing handovers), policy control (setting network rules and managing QoS) and path management (choosing the best data routes on user and network information). Of course, all these modules make sure that the network performance will be smooth. Moreover, the system is quite flexible because it uses APIs, whose job is to manage the infrastructure and help to improve user's experience [24].

The control cloud is a step forward from traditional LTE networks because it centralizes and redefines network control by turning functions into software and virtualizing network elements. It also gathers information from both the access cloud and the forwarding cloud and for that reason it enables centralized control [24]. The access cloud or intelligent RAN, is designed for flexible coverage based on specific requirements and it virtualizes wireless resources, making it easy to adapt to different user demands. In addition, with a user-centric virtual cell, it reduces unnecessary switching and creates a better overall experience by allowing users to connect according to their location. Lastly, the forwarding cloud handles high-speed data traffic and it is controlled by the control cloud, which optimizes the data flow, aiming to reduce latency [24]. In Figure 2.6, a 5G/6G network system architecture is presented.

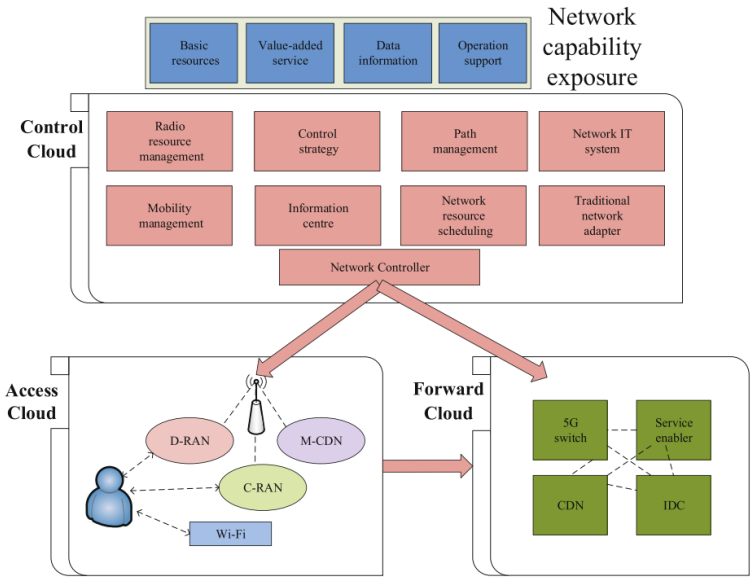

Figure 2.6: 5G/6G network system architecture [24]

2.6 Benefits of SDN

- Improves network agility and makes the deployment of applications and services faster.
- Creates virtual Ethernet networks without the complexity and limitations of the VLANs.
- Simplifies configuration and link setup, which makes networks easier to manage.

- Lowers capital expenditure (CapEx) by using switches that run on off-the-shelf chips.
- Enables precise traffic management based on individual traffic flows.
- Supports dynamic movement, replication and allocation of virtual resources.
- Facilitates centralized orchestration for efficient application delivery and resource provisioning.
- Simplifies the implementation of quality of service (QoS).

3 CHAPTER 3rd : Network Functions Virtualization (NFV)

Network functions virtualization (NFV) is rapidly reshaping the networking world by moving away from customized hardware to more flexible and software-based solutions. These changes support the growing demands for scalable, elastic and agile networks that are essential for cloud-based services [25]. This chapter first explores the transition from traditional networks to NFV, its basic concepts, ETSI's NFV framework and the benefits of NFV. Furthermore, it includes a review of OpenStack and presents a simple OpenStack implementation.

3.1 The Evolution of Network Architecture

If someone want to understand why the networking industry is quickly adopting NFV, it's important to consider the evolution of networks and the challenges they face today. Despite the fact that in today's world, the networks have improved in speed and capacity, they still face problems with the demands of cloud services, huge data centers and IoT. Traditional networks cannot handle the new reality effectively. For example, traditional data transport networks, like early phone and telegram systems, were designed with a focus on low latency, high availability and minimal data loss. All of these networks were relied on specialized hardware that was built for specific functions and had a specific software inside them. With the passage of time, however, the demand for bandwidth became gigantic, making these devices a bottleneck. So, service providers had to find new technologies such as NFV to overcome these constrains [25].

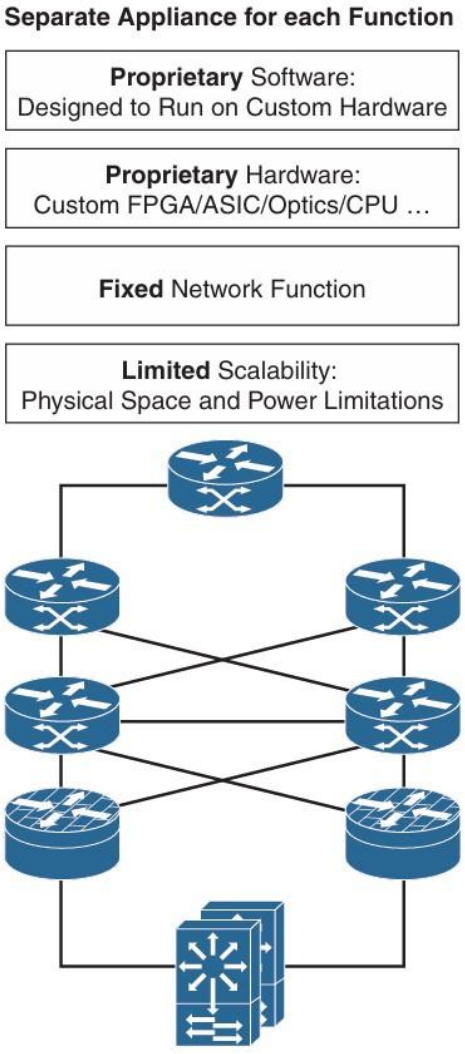

Figure 3.1: Traditional Network Devices [25]

Some of the limitations of the traditional network devices were:

1. Flexibility Limitations
2. Manageably Issues
3. Scalability Constrains
4. Time-To-Market Challenges
5. Capacity Over-Provisioning
6. High Operational Costs
7. Migration Considerations

Network Function Virtualization takes the central idea of virtualization, which is already common in data centers with servers, and applies it to network devices. Instead of using dedicated hardware for specific network functions, NFV allows these functions to be run as software on shared hardware. This means that the software is separated from the hardware, which also means that network functions are no longer depend on customized equipment. With the help of NFV, off the shelf hardware (COTS) can run virtual versions of firewalls, routers and more, making the networks more adaptable and cheaper to operate. This leads to changes about how networks are built and managed and enables new designs and innovations that weren't possible before [25].

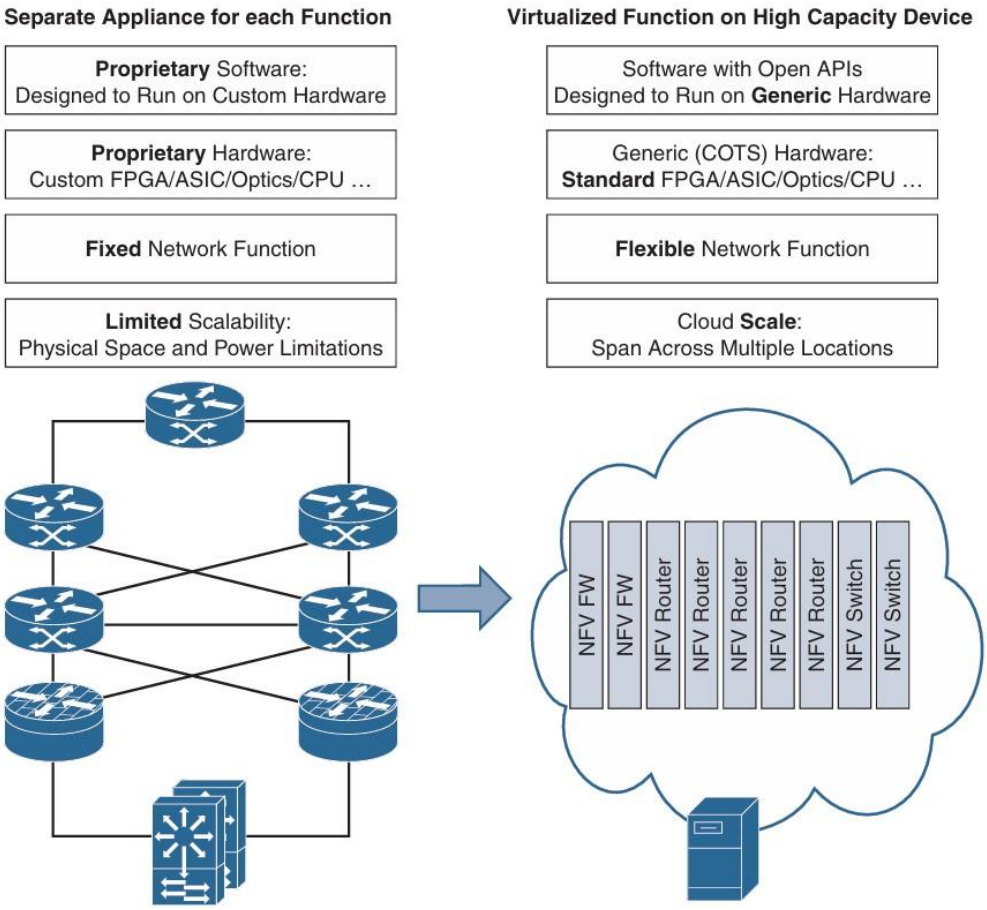

Figure 3.2: Transition to NFV [25]

3.2 ETSI NFV Framework

NFV was introduced in 2012 by a group of service providers to address the challenges of relying on new hardware for network services. Among these challenges were the deployment cost, design changes, the complexity of the hardware and the rapid obsolescence of equipment. In order to solve these issues, NFV was proposed as a solution. In 2013, some leading telecom operators formed a group known as ETSI with the purpose of creating standards for NFV. The three main criteria they focused on were the full split of software and hardware (decoupling), the precise control and monitoring of the network and a way to make the deployment of the network functions scalable and automated [25]. As a result, an architectural framework specifying separated areas of focus, as shown in Figure 3.3, was created from these requirements.

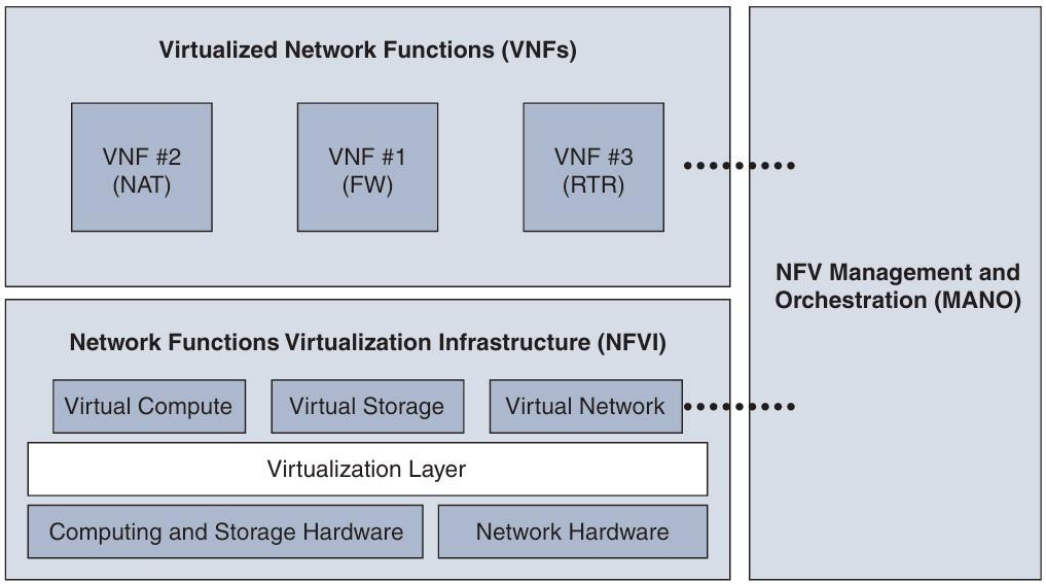

Figure 3.3: High-Level ETSI NFV framework [25]

This framework covers the management of VNFs, their interactions, data flow between VNFs and resource allocation. So, all these tasks were divided into three components:

- *Network Functions Virtualization Infrastructure block (**NFVI**):* This component is the base of the entire architecture. It includes the hardware (servers, ram, disk storage, NAS, switches, firewalls) needed to run virtual machines, the software that enables virtualization and the virtualized resources (such as virtual network, virtual storage and virtual compute) [25] [26].

- *Virtualized Network Function block (**VNF**):* This component utilizes the virtual machines provided by NFVI and enhances them by incorporating software that executes the virtualized network functions [25].

- *Management and Orchestration block (**MANO**):* MANO is a different part of the architecture that interact with both the NFVI and VNF blocks. It is responsible for managing all the resources from the infrastructure layer and creating, deleting and managing resource allocation for the VNFs [25].

For someone to understand this framework, he must first start with understanding the concept of VNFs. VNFs are software implementations of network functions like firewalls, NAT devices or routers, which operate separately from the physical hardware. These VNFs can run on general-purpose hardware, known as COTS, rather than on specialized devices. Virtualization technologies, such as hypervisors or containers, enable multiple VNFs to share hardware resources efficiently. Moreover, VNFs need the necessary infrastructure to operate, so the NFVI helps them by using COTS hardware as a pool of resources that can be dynamically allocated as required by the VNFs. This setup allows flexibility, but it also means that VNFs cannot control the hardware resources they use and therefore they rely on the virtualization layer to manage resource allocation without knowing about other VNFs sharing the same hardware. To effectively manage this virtualized environment, the ETSI framework includes the MANO block. MANO has the responsibility to oversee the deployment, operation and interconnection of VNFs on the NFVI and ensure that all resources are allocated right. It could be characterized as the central management system, providing detailed visibility into the operational status and the usage of the resources. Because of this, MANO is the best interface for collecting utilization statistics from the operational and billing systems [25].

3.3 MANO Framework

The previous section gave an overview of the ETSI NFV architecture and its core components. The ETSI framework goes even further by breaking these components into specific functional blocks, each with its own role and responsibilities. In this section, a short reference will be given to the most important elements that compose the management and orchestration block (MANO).

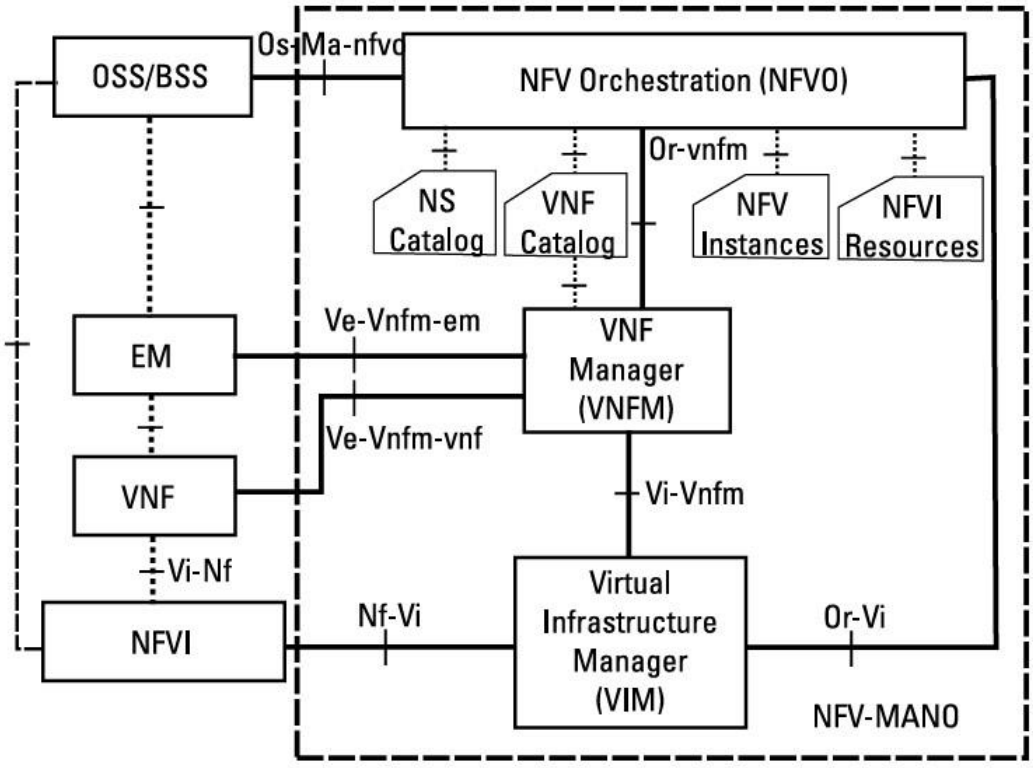

Figure 3.4: NFV MANO architectural framework

The MANO block is composed of the Virtualized Infrastructure Manager (**VIM**), Virtualized Network Manager (**VNFM**) and NFV Orchestrator (**NFVO**).

The Virtualized Infrastructure Manager controls and manages the interaction between NFV and NFVI resources. It also owns useful deployment and monitoring tools that keep a detailed inventory of hardware resources.

The VNF Manager is responsible for lifecycle management of VNF instances. It is also responsible for initialize, update, scale, query and terminate VNF instances.

The NFV Orchestrator is the pivotal component that allocates resources for a VNF through the VIM and hands it over to the VNF manager for lifecycle management. Moreover, it is responsible for instantiation, policy management, KPI monitoring and performance measurement.

The NFVO is further supported by four repositories. The Network Service Catalog (NS Catalog) which contains usable network services, VNF catalog which contains all the available VNF descriptors, NFVI resources repository which tracks all the resources that were used and NFV instances which keeps a record of all relationships between VNFs and NS.

3.4 Benefits of NFV

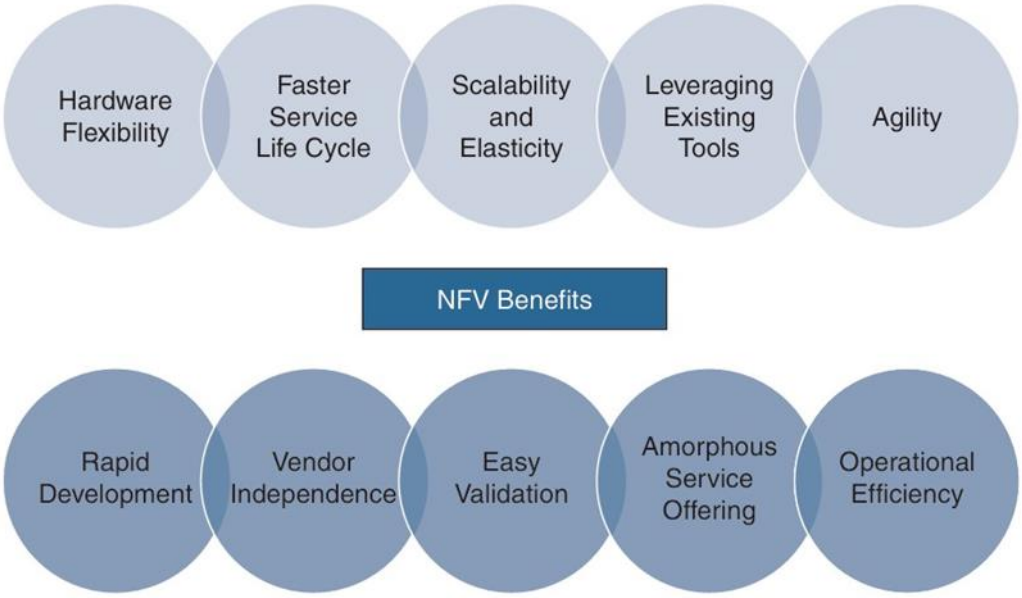

Figure 3.5: Benefits of NFV [25]

- Hardware Flexibility: Network operators can use flexible, standard hardware and easily adjust their resources compared to traditional equipment that requires costly upgrades [25].
- Faster Service Life Cycle: NFV enables rapid, on-demand deployment and removal of network functions, something which will significantly reduce setup times and costs compared to traditional hardware installations [25].
- Agility: It is the capability to swiftly deploy, modify or relocate VNFs across the network as needed [25].

- Scalability: It allows quick resource adjustments and workload distribution for virtual network functions [25].
- Leveraging Existing Tools: NFV can leverage and reuse the deployment and management tools, which will lead to faster deployments without the need for new tools and additional costs [25].
- Rapid Development and Vendor Independence: Networks operators can quickly adopt different vendors' solutions without high costs and they can rapidly develop new features with open-source support [25].
- Validation of New Solutions: NFV makes testing solutions more cost effective by allowing test setups without having to replicate the full production environment [25].
- Amorphous Service Offering: NFV helps network providers to add or scale down services depending on demand and shift resources to different locations according to needs [25].

3.5 OpenStack

Traditional applications that run on VMs are most of the time managed by coordination tools like VMware and rely on high availability through traditional infrastructure like SAN storage. But cloud-based applications, such as Hadoop and MySQL, are built completely differently because they are designed to scale horizontally across multiple servers and to handle failures independently of the infrastructure. Unlike common applications that depend on high availability, cloud applications expect failures and manage their own resiliency, making monolithic architectures unsuitable. As a result, Cloud platforms take a different approach from virtualization. They don't rely on shared infrastructure for availability but instead they prefer to use commodity hardware for horizontal scaling and they move application resiliency up the software stack. This philosophy enables cloud-based applications to operate in an efficient way without the need for expensive and redundant infrastructure [19].

OpenStack is a cloud management platform designed to handle this new architecture. It's an open-source project supported by major tech companies that acts as an operating system for building public and private clouds. Among many other things, OpenStack is a VIM solution, which means that it offers control over NFV infrastructure, handling storage networking and compute across both virtual machines and physical hardware. The platform also allows users to easily deploy, manage and remove VMs within the cloud environment [27]. OpenStack consists of multiple components. The most important ones are:

Nova: This is the primary controller in any IaaS system and its basic job is to control and automate computer resources.

Horizon: This is a GUI that users enter to manage and monitor the whole network.

Neutron: This is a system that its purpose is to manage networking and IP address assignment.

Glance: This is an image service that manages disk and server images.

Keystone: This is a system that provides authentication and identity services.

Cinder: This is a block storage service that it is designed to give users storage resources and it virtualizes block storage device management.

OpenStack is a critical component in NFV architectures, especially in telecommunication environments, because of its flexible and multi-tenant infrastructure. Its standardized interfaces allow seamless orchestration of NFV elements such as virtual machines, containers and storage, simplifying the management of tasks through automation. In fact, its architecture is well-proven for creating scalable clouds by using existing tools such as APIs and other services. OpenStack integrates without difficulty with Kubernetes especially in telecom sectors preparing for 5G, where containers usually outperform virtual machines. As a result, OpenStack, NFV and containerization form an ideal combination.

3.6 OpenStack implementation

In this simple OpenStack implementation, a basic private cloud environment will be set up. The easiest and most straightforward way to deploy OpenStack is a single-node installation, where all the OpenStack services run on a single machine. In this case, this will be done using DevStack, a set of scripts which will automate the installation and configuration of OpenStack components. The base operating system will be an Ubuntu Desktop which will run inside Oracle's VirtualBox.

The first step is to download and install DevStack from the official repository with the following command:

```
$ git clone https://opendev.org/openstack/devstack
```

Next, a configuration file named local.conf in the DevStack directory must be created. This file will contain the basic configuration for a minimal installation of DevStack. This can be done using:

```
$ cd devstack
$ cat <<EOL > local.conf
  [[local|localrc]]
  ADMIN_PASSWORD=password
  DATABASE_PASSWORD=password
  RABBIT_PASSWORD=password
  SERVICE_PASSWORD=password
  HOST_IP=your_server_ip
  EOL
```

After the configuration, the DevStack's installation script is:

```
$ ./stack.sh
```

Once the installation is complete, OpenStack will be accessible through the Horizon dashboard by opening a web browser and going to the following address: http://user's_server_ip/dashboard. Users can login by using the credentials they set in the local.conf file.

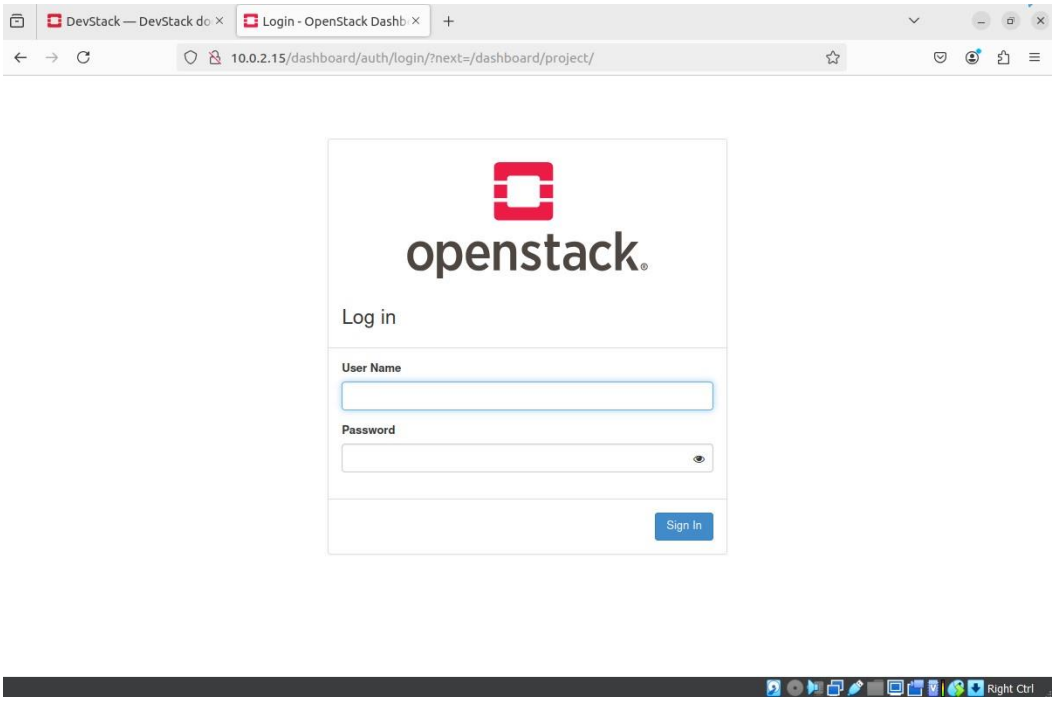

Figure 3.6: OpenStack dashboard (Horizon)

After the installation the core services can be checked that they run properly from the OpenStack CLI:

```
$ openstack service list
```

```
stack@giannisfs-VirtualBox:~/devstack$ openstack service list
+----------------------------------+-------------+----------------+
| ID                               | Name        | Type           |
+----------------------------------+-------------+----------------+
| 01c2cb1104fe4271abfa0260c6510d40 | cinder      | block-storage  |
| 04ce495ce9604cb5b7ba583d941dcce0 | placement   | placement      |
| 0679259055174b2b9fd65af4e3a1880a | cinderv3    | volumev3       |
| 0a67042e3eb242eebd2dfe442255de77 | neutron     | network        |
| 263cfb485e3249689af3aaecbb103f94 | nova        | compute        |
| 86ab7ff0972b4bea95241a706c8591e8 | nova_legacy | compute_legacy |
| c2c23694292642d79cc61ede2e2a8d2d | keystone    | identity       |
| dbea491e378b441cae30112479d9ef5a | glance      | image          |
+----------------------------------+-------------+----------------+
```

Figure 3.7: List of running services

We can set up a new project and a new user to start testing OpenStack, using the next commands:

```
$ openstack project create --domain default --description "Test Project" name_of_project
$ openstack user create --domain default --password-prompt name_of_user
```

Also, we can ensure that the required components are available such images, flavors, networks and keypairs with the following commands:

```
$ openstack image list
$ openstack flavor list
$ openstack network list
$ openstack keypair list
```

```
stack@giannisfs-VirtualBox:~/devstack$ openstack image list
+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| 2e45ff89-19ed-4224-a62d-db7a94aca2a3 | cirros-0.6.2-x86_64-disk | active |
+--------------------------------------+--------------------------+--------+
stack@giannisfs-VirtualBox:~/devstack$ ^C
stack@giannisfs-VirtualBox:~/devstack$ openstack flavor list
+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 1  | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3  | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4  | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 42 | m1.nano   |   192 |    1 |         0 |     1 | True      |
| 5  | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
| 84 | m1.micro  |   256 |    1 |         0 |     1 | True      |
| c1 | cirros256 |   256 |    1 |         0 |     1 | True      |
| d1 | ds512M    |   512 |    5 |         0 |     1 | True      |
| d2 | ds1G      |  1024 |   10 |         0 |     1 | True      |
| d3 | ds2G      |  2048 |   10 |         0 |     2 | True      |
| d4 | ds4G      |  4096 |   20 |         0 |     4 | True      |
+----+-----------+-------+------+-----------+-------+-----------+
stack@giannisfs-VirtualBox:~/devstack$
```

Figure 3.8: Lists of available images and flavors

If one of these components is missing, we have to create it. For example, if we want to create an SSH keypair the command is:

```
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
```

If we want to create a new network and a subnet, the commands are:

```
$ openstack network create mynetwork
$ openstack subnet create --network mynetwork --subnet-range 10.0.0.0/24 mysubnet
```

Now that we have the necessary components, we can launch VM instances with the following command:

```
$ openstack --os-project-name <project_name> server create --image <image_name>  --flavor <flavor_name>
--network <network_name>   --key-name <keypair_name> <instance_name>
```

```
stack@giannisfs-VirtualBox:~/devstack$ openstack role add --project test_project --user admin member
stack@giannisfs-VirtualBox:~/devstack$ openstack --os-project-name test_project server create --image cirros-0.6.2-x86_64-disk -
-flavor m1.nano --network shared --key-name mykey giannisf
+-------------------------------------+-----------------------------------------------------------------+
| Field                               | Value                                                           |
+-------------------------------------+-----------------------------------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                                          |
| OS-EXT-AZ:availability_zone         |                                                                 |
| OS-EXT-STS:power_state              | NOSTATE                                                         |
| OS-EXT-STS:task_state               | scheduling                                                      |
| OS-EXT-STS:vm_state                 | building                                                        |
| OS-SRV-USG:launched_at              | None                                                            |
| OS-SRV-USG:terminated_at            | None                                                            |
| accessIPv4                          |                                                                 |
| accessIPv6                          |                                                                 |
| addresses                           |                                                                 |
| adminPass                           | CKQ89QenZR5s                                                    |
| config_drive                        |                                                                 |
| created                             | 2024-09-09T04:39:18Z                                            |
| flavor                              | m1.nano (42)                                                    |
| hostId                              |                                                                 |
| id                                  | 81cb40ab-8ec9-423e-9678-5a05f3a51f0e                            |
| image                               | cirros-0.6.2-x86_64-disk (2e45ff89-19ed-4224-a62d-db7a94aca2a3) |
| key_name                            | mykey                                                           |
| name                                | giannisf                                                        |
| os-extended-volumes:volumes_attached | []                                                              |
| progress                            | 0                                                               |
| project_id                          | 0c35211ba9ed4e8daaa232d0e00916b1                                |
| properties                          |                                                                 |
| security_groups                     | name='default'                                                  |
| status                              | BUILD                                                           |
| updated                             | 2024-09-09T04:39:17Z                                            |
| user_id                             | 715cd6334ea24955bc74f5ba8c551c6e                                |
+-------------------------------------+-----------------------------------------------------------------+
```

Figure 3.9: The first instance

```
stack@giannisfs-VirtualBox:~/devstack$ openstack --os-project-name test_project server create --image cirros-0.6.2-x86_64-disk -
-flavor m1.micro --network shared --key-name mykey giannisf2
+-------------------------------------+-----------------------------------------------------------------+
| Field                               | Value                                                           |
+-------------------------------------+-----------------------------------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                                          |
| OS-EXT-AZ:availability_zone         |                                                                 |
| OS-EXT-STS:power_state              | NOSTATE                                                         |
| OS-EXT-STS:task_state               | scheduling                                                      |
| OS-EXT-STS:vm_state                 | building                                                        |
| OS-SRV-USG:launched_at              | None                                                            |
| OS-SRV-USG:terminated_at            | None                                                            |
| accessIPv4                          |                                                                 |
| accessIPv6                          |                                                                 |
| addresses                           |                                                                 |
| adminPass                           | tETfbUbATJH9                                                    |
| config_drive                        |                                                                 |
| created                             | 2024-09-09T04:43:04Z                                            |
| flavor                              | m1.micro (84)                                                   |
| hostId                              |                                                                 |
| id                                  | ceb224a6-4a08-4d4a-9335-eb9ee6478ad3                            |
| image                               | cirros-0.6.2-x86_64-disk (2e45ff89-19ed-4224-a62d-db7a94aca2a3) |
| key_name                            | mykey                                                           |
| name                                | giannisf2                                                       |
| os-extended-volumes:volumes_attached | []                                                              |
| progress                            | 0                                                               |
| project_id                          | 0c35211ba9ed4e8daaa232d0e00916b1                                |
| properties                          |                                                                 |
| security_groups                     | name='default'                                                  |
| status                              | BUILD                                                           |
| updated                             | 2024-09-09T04:43:04Z                                            |
| user_id                             | 715cd6334ea24955bc74f5ba8c551c6e                                |
+-------------------------------------+-----------------------------------------------------------------+
stack@giannisfs-VirtualBox:~/devstack$
```

Figure 3.10: The second instance

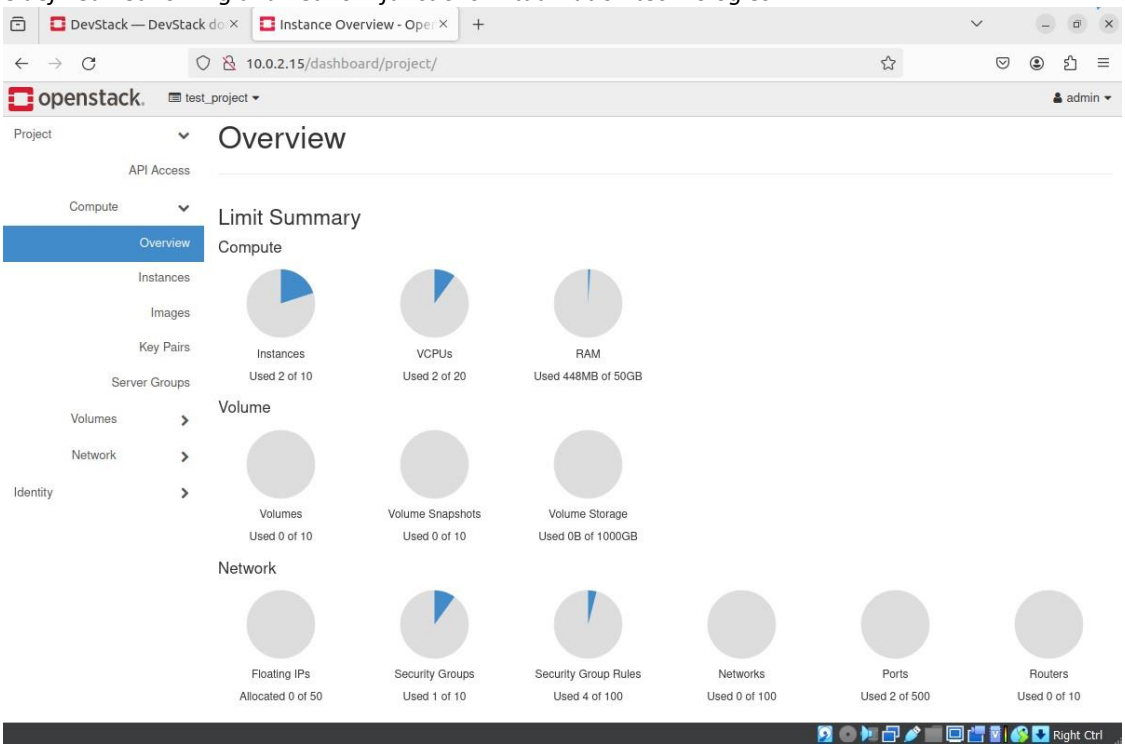

Figure 3.11: OpenStack's dashboard overview

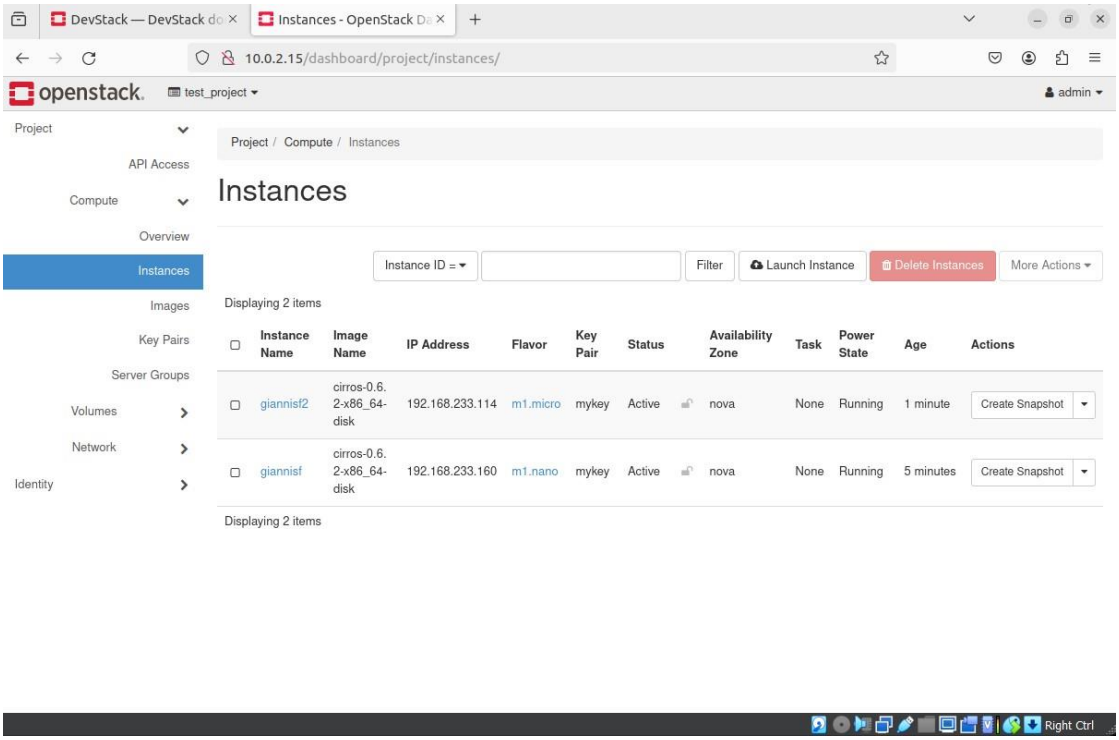

Figure 3.12: The two created instances in the dashboard

That was a simple way to create an instance in OpenStack using the OpenStack CLI. We can further manage and customize instances by resizing or attaching volumes through the OpenStack CLI or Horizon dashboard.

4 Conclusions

The convergence of virtualization, containerization, cloud computing, SDN and NFV is changing significantly the modern network architectures. This thesis has explored how these technologies work in synergy to satisfy the ever-increasing demands for flexible, scalable and cost-effective network solutions. Because they abstract physical resources and decouple network functions from hardware, these innovations have made a big step toward more agile and software-driven infrastructures that can quickly adapt to dynamic and complex network environments. Virtualization has proven to play an important role in networking by utilizing resources efficiently and simplifying management. Meanwhile containerization is a technology that offers lightweight, portable solutions that make the deployment of applications very fast, especially inside cloud-native environments. Together these technologies have changed to a great extent the way that resources are provisioned and managed, making cloud computing the best option for scalable and on-demand services. SDN has completely changed network control because it has left from the traditional model in which each device works as an isolated unit and has moved to centralized management through a controller. Finally, NFV, by virtualizing traditional network functions, has advanced even further in the direction of software-defined infrastructures, which means lower capital and operational expenses plus rapid deployment of network services.

Through the analysis presented in this thesis, it is clear that the integration of SDN, NFV and containerized environments within cloud infrastructures has huge potential for future network evolution. These technologies not only optimize the performance of the networks but also introduce new levels of automation, flexibility and customization. Their big impact is especially obvious in emerging fields such as 5G and 6G networks, where the ability to deploy and manage services dynamically, is paramount.

In conclusion, this thesis tries to help readers understand how SDN and NFV are essential components in the development of next-generation network infrastructures. These technologies enable a shift away from hardware-centric architectures toward fully software-based, programmable and automated networks. As our area is characterized by 5G, IoT and edge computing, the continued evolution of these technologies will be definitive if we want to build flexible, scalable and intelligent networks.

5 Bibliography – References – Online Sources

[1] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," in *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483-490, Sep. 1981, doi: 10.1147/rd.255.0483.

[2] R. R. P. Goldberg, "Survey of virtual machine research," in *Computer*, vol. 7, no. 6, pp. 34-45, June 1974, doi: 10.1109/MC.1974.6323581.

[3] "What is containerization? | IBM." https://www.ibm.com/topics/containerization [Accessed June 2024]

[4] E. Bauer and R. Adams, *Reliability and availability of cloud computing*. Wiley-IEEE Press, 2012.

[5] "What is virtualization? | IBM." https://www.ibm.com/topics/virtualization [Accessed June 2024]

[6] M. Portnoy, *Virtualization essentials*. Indianapolis, In Wiley Pub., Inc, 2012.

[7] A. K. Alnaim, A. M. Alwakeel and E. B. Fernandez, "A Pattern for an NFV Virtual Machine Environment," *2019 IEEE International Systems Conference (SysCon)*, Orlando, FL, USA, 2019, pp. 1-6, doi: 10.1109/SYSCON.2019.8836847.

[8] "Oracle VM VirtualBox User Manual," Feb. 04, 2020. https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/clone.html [Accessed June 2024]

[9] J. Nickoloff, *Docker in Action*. Manning Publications, 2016.

[10] M. Luksa, *Kubernetes in action*. Manning Publications, 2018.

[11] "What is cloud computing? | IBM." https://www.ibm.com/topics/cloud-computing [Accessed May 2024]

[12] "What are IAAS, PAAS and SAAS? | IBM." https://www.ibm.com/topics/iaas-paas-saas [Accessed May 2024]

[13] T. Diaby and B. B. Rad, "Cloud Computing: A review of the Concepts and Deployment Models," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 9, no. 6, pp. 50-58, 2017.

[14] GeeksforGeeks, "Cloud deployment models," *GeeksforGeeks*, May 03, 2023. https://www.geeksforgeeks.org/cloud-deployment-models/ [Accessed May 2024]

[15] "Cloud Deployment Model - javatpoint," *www.javatpoint.com*. https://www.javatpoint.com/cloud-deployment-model [Accessed June 2024]

[16] B. Sosinsky, *Cloud Computing Bible*. John Wiley & Sons, 2011.

[17] "Advantages of Cloud Computing | Google Cloud," *Google Cloud*. https://cloud.google.com/learn/advantages-of-cloud-computing [Accessed June 2024]

[18] Z. Μακαλού, "A survey on next generation networks and cloud computing," *Uniwa.gr*, 2021, doi: https://polynoe.lib.uniwa.gr/xmlui/handle/11400/577.

[19] J. Doherty, *SDN and NFV Simplified: A Visual Guide to Understanding Software Defined Networks and Network Function Virtualization*, Addison-Wesley Professional, 2016.

[20] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, Jan. 2017, doi: https://doi.org/10.1016/j.comnet.2016.11.017.

[21] Open Networking Foundation, "SDN architecture," no. 1, 2014.

[22] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks*. O'Reilly Media, 2013.

[23] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015, doi: https://doi.org/10.1109/comst.2014.2330903.

[24] Q. Long, Y. Chen, H. Zhang, and X. Lei, "Software Defined 5G and 6G Networks: a Survey," *Mobile Networks and Applications*, vol. 27, no. 5, pp. 1792–1812, Nov. 2019, doi: 10.1007/s11036-019-01397-2.

[25] R. Chayapathi, S. F. Hassan, and P. Shah, *Network Functions Virtualization (NFV) with a Touch of SDN*. Addison-Wesley Professional, 2016.

[26] M. S. Bonfim, K. L. Dias, and S. F. L. Fernandes, "Integrated NFV/SDN Architectures," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–39, Feb. 2019, doi: 10.1145/3172866.

[27] K. Gray and T. D. Nadeau, *Network Function Virtualization*. Morgan Kaufmann, 2016.