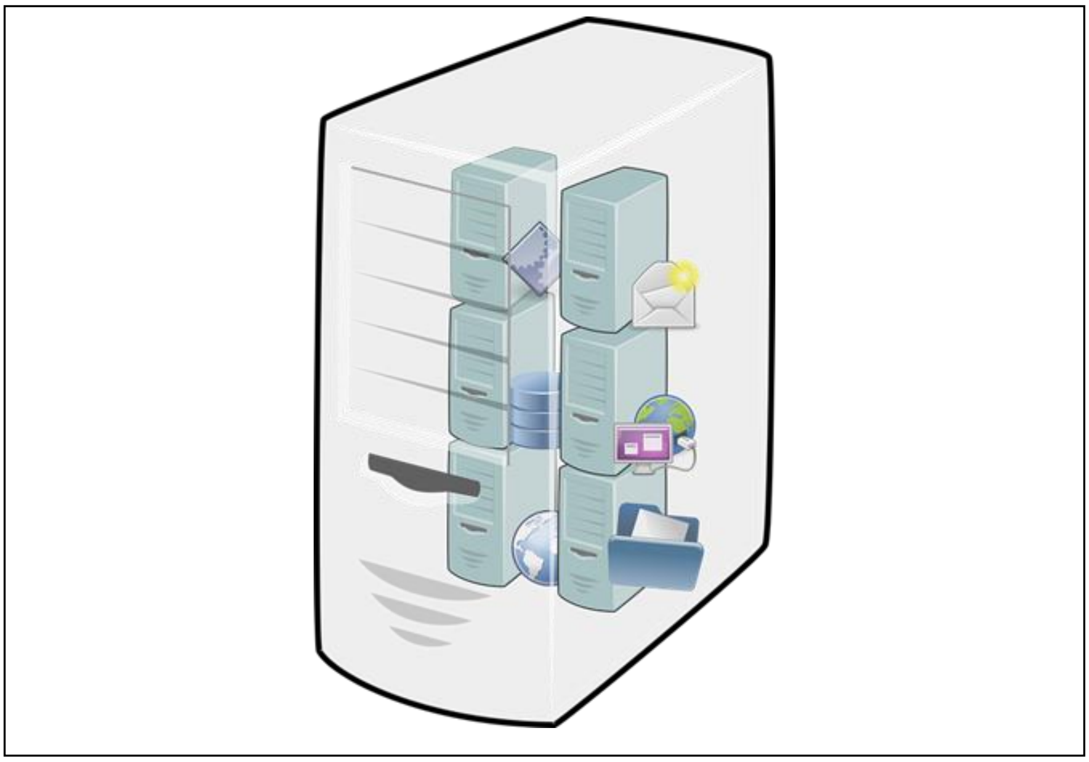
UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

A study of virtualization technologies for the support of virtual labs



Student: ARISTEIDIS BALTZOIS
Registration Number: 04363

Supervisor

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

ATHENS-EGALEO, SEPTEMBER 2022

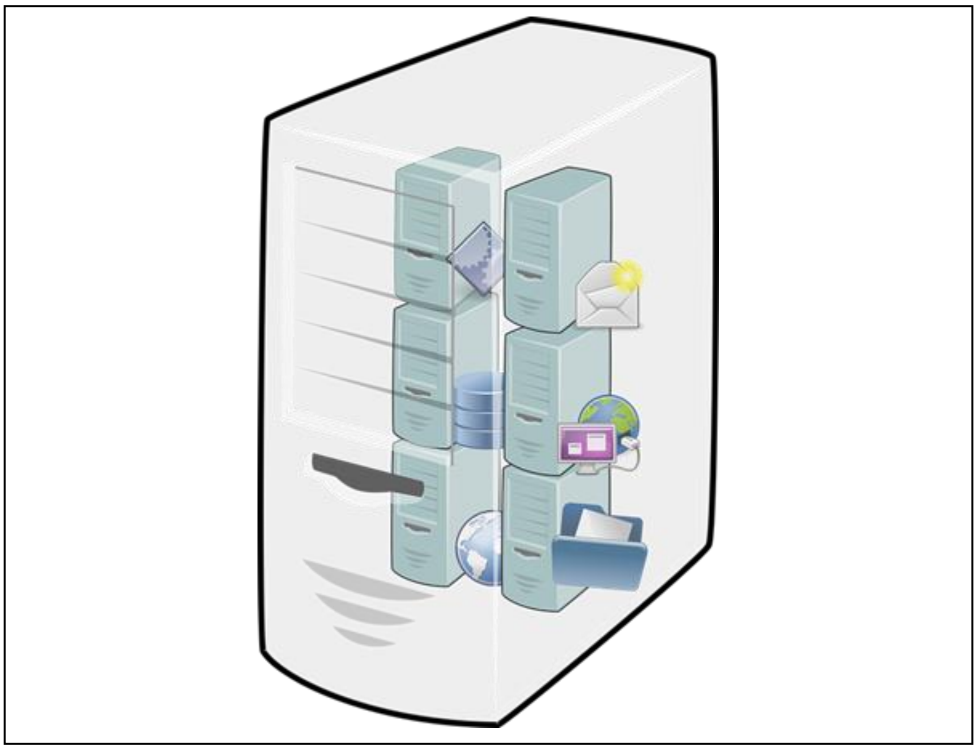
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

Μια μελέτη των τεχνολογιών εικονικοποίησης για την υποστήριξη των εικονικών εργαστηρίων



Φοιτητής: ΑΡΙΣΤΕΙΔΗΣ ΜΠΑΛΤΖΩΗΣ
Αριθμός Μητρώου: 04363

Επιβλέπων Καθηγητής

ΧΑΡΑΛΑΜΠΟΣ ΠΑΤΡΙΚΑΚΗΣ
Καθηγητής στο Τμήμα Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΣΕΠΤΕΜΒΡΙΟΣ 2022

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

(Ονοματεπώνυμο), (βαθμίδα)	(Ονοματεπώνυμο), (βαθμίδα)	(Ονοματεπώνυμο), (βαθμίδα)
[illegible] Digitally signed by Charalampos Patrikakis Date: 2022.10.13 17:48:37 +03'00' (Υπογραφή)	Periklis Papadopoulos Digitally signed by Periklis Papadopoulos Date: 2022.10.13 18:17:36 +03'00' (Υπογραφή)	Michail Feidakis Digitally signed by Michail Feidakis Date: 2022.10.13 16:09:07 +03'00' (Υπογραφή)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Αριστείδης Μπαλτζώης του Δημητρίου, με αριθμό μητρώου 04363 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.

Ημερομηνία 18/9/2022
Ο Δηλών
Αριστείδης Μπαλτζώης

[illegible]

DEDICATION

I dedicate this Diploma Thesis to all those who try so many things and don't give up until they find what gives them real meaning in life. In addition, I dedicate it to my family and to all those who supported me, and those who tried to prevent me from making any choices during these six years of study.

ACKNOWLEDGEMENTS

I would like to thank my professor and supervisor Prof. Patrikakis Charalampos. Thanks to him and his love for the subject he teaches, I became interested in further involvement in the field of networks, at a time when I had lost all interest in continuing my studies. Furthermore, I thank him for his support, his guidelines, his availability and his attention to every detail.

Moreover, I would like to also thank Ph.D. candidate Mr. Michael G. Xevgenis. His role as an Assistant to the Supervisor was also important, as his knowledge in the field of cloud computing and virtualization technologies was crucial in addressing problems that emerged. I thank him for the time he devoted to me and for his prompt response to any issue.

Περίληψη

Η παρούσα διπλωματική παρουσιάζει την πρόοδο και την εξέλιξη των τεχνολογιών της εικονικοποίησης, του Cloud Computing καθώς και του containerization. Η χρήση της εικονικοποίησης οδήγησε στην μέγιστη χρήση και εκμετάλλευση των φυσικών εξυπηρετητών, ενώ η εισαγωγή του Cloud Computing παρέχει την δυνατότητα σε οποιοδήποτε οργανισμό ή χρήστη να έχει πρόσβαση σε πόρους όποια στιγμή επιθυμεί, χωρίς να διαθέτει αναγκαστικά το δικό του κέντρο δεδομένων. Επιπλέον, η τεχνολογία της εικονικοποίησης εισήγαγε τα εικονικά μηχανήματα ενώ το containerization τα containers. Τα τελευταία χρόνια, ο τρόπος διαχείρισης και ενορχήστρωσης των containers αποτελεί αντικείμενο ευρείας μελέτης, ενώ εργαλεία όπως το Kubernetes αποκτούν ολοένα και μεγαλύτερη ζήτηση για την δημιουργία υποδομών που θα υποστηρίξουν εφαρμογές και υπηρεσίες. Τα τελευταία χρόνια η εξέλιξη αυτών των τεχνολογιών χρησιμοποιείται και για καθαρά εκπαιδευτικούς σκοπούς, με την δημιουργία εικονικών εργαστηρίων. Στόχος τους είναι η βελτίωση της ηλεκτρονικής μάθησης.

Το πειραματικό μέρος της παρούσας διπλωματικής, στοχεύει στο σχεδιασμό και τη δημιουργία εικονικών εργαστηρίων χρησιμοποιώντας το λογισμικό Kubernetes και προσαρμοσμένες εικόνες Docker, προσομοιώνοντας την διαδικασία ενός εργαστηρίου με φυσική παρουσία.

Λέξεις – κλειδιά: εικονικοποίηση, εικονικά μηχανήματα, κέντρα δεδομένων, cloud computing, Docker, containers, Kubernetes

Abstract

This thesis presents the progress and evolution of virtualization, Cloud Computing and containerization technologies. The use of virtualization has led to the maximum use and utilization of physical servers, while the introduction of Cloud Computing provides the ability for any organization or user to access resources whenever they wish, without necessarily having their own data center. In addition, virtualization technology introduced virtual machines while containerization introduced containers. In recent years, the way containers are managed and orchestrated has been widely studied, while tools such as Kubernetes are becoming increasingly in demand for creating infrastructure to support applications and services. In recent years, the development of these technologies has also been used for purely educational purposes, with the creation of virtual laboratories. Their aim is to improve e-learning.

The experimental part of this thesis aims to design and create virtual labs using Kubernetes software and custom Docker images, simulating the process of a physical lab.

Keywords: virtualization, virtual machines, data centers, cloud computing, Docker, containers, Kubernetes

Περιεχόμενα

List of Tables

List of Figures

Alphabetical Index

API	Application Programming Interface
AWS	Amazon Web Services
CAPEX	Capital Expenditure
CLI	Command Line Interface
CPU	Central Processing Unit
CRM	Customer Relationship Management
EC2	Amazon Elastic Compute Cloud
ERP	Enterprise resource planning
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IBM	International Business Machines Corporation
IT	Information Technology
MAC	Media Access Control address
OPEX	Operational Expenditure
OS	Operating System
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
VLAN	Virtual Local Area Network
VMM	Virtual Machine Monitor
VMs	Virtual Machines
YAML	Yet Another Markup Language

INTRODUCTION

Object of the Diploma Thesis

This thesis aims to analyze how virtualization and containerization technologies have been adopted into various sectors either in organizations or to each person itself. In particular, the theoretical part will analyze the progress from the traditional architecture of a simple IT infrastructure to the virtualized architecture and lastly into the container architecture.

The experimental part will present a solution that utilizes this progress of two containerized technologies: Docker and Kubernetes, to create virtual labs that can be used from students.

Purpose and objectives

The purpose of this thesis is to create an infrastructure to support virtual labs using the Kubernetes software and Docker images. The goal of the experiment is to provide a virtual environment for students and improve the e-learning process.

Methodology

The composition of the theoretical part of the thesis was based on an extensive research and study in books, web publications, theses, journal articles conference papers, and dissertations.

The experimental part was based a lot on the documentation of the different tools and software used, which can be found on the internet. In addition, to compose the code of the web solution we relied a lot on information retrieved from GitHub.

Innovation

Using the Kubernetes tool to manage and orchestrate Docker images via Pods, the creation of a cluster that will be connected with a web solution are innovative elements as Kubernetes is a relatively new tool and its use for creating infrastructure to support virtual labs has not been presented before.

Structure

The first chapter analyzes the technology of virtualization and the way in which IT technology has shifted from traditional data centers to it. The second chapter analyses the technology of cloud computing. The third chapter analyzes the technology of Containerization, the different metaphors of a virtual machine with containers, Docker and Kubernetes.

Chapter four introduces the virtual labs and analyzes the use case scenario of the practical part of the thesis. The fifth chapter details the steps taken to implement the supporting infrastructure of the virtual labs. The sixth and last chapter contains the results as well as ways in which our experimental part can be developed in the future.

1 Virtualization vs Data Centers

In this chapter, we will provide an extensive analysis of virtualization technology and the closely related technologies of virtual machines and hypervisors. Then we will refer to data centers and how virtualization has penetrated them, creating virtualized data centers. Finally, we will refer to the benefits that data center virtualization provides.

1.1 Defining Virtualization.

Virtualization is a software-based technology that has been around for almost half a century. It is an evolving technology that creates a virtual image or "version" of something such as a server, operating system, storage devices, or network resources so that they can be utilized on multiple machines at the same time. The main aim of this technology is to manage the workload by transforming traditional computing to make it scalable, efficient and economical [1].

The main goal of virtualization is to separate the physical hardware from the activities operating on top of it, introducing a layer of abstraction. The elements of an IT infrastructure, which are network, compute, storage, once separated from the material, create together a pool of resources that can be used automatically and meet any need that arises.

Virtualization was firstly implemented by IBM as the logical division of the computer's mainframe into separated virtual computers called virtual machines. These partitions allowed mainframes to run many different operations and applications simultaneously. To keep the Quality of Service at the right level, virtualization technology is also responsible for the proper distribution of computing resources into these applications. IBM defined it as: "Virtualization is the creation of substitutes for real resources, that is substitutes that have the same functions and external interfaces as their counterparts, but that differ in attributes, such as size, performance, and cost." [2].

Moreover, virtualization has been reintroduced to many IT devices such as servers and desktop computers. For this reason, many IT companies shifted their attention towards this new technology and this evolution has a major and beneficial impact on modern organizations.

The following figure 1.1 is a proper example of virtualization. It depicts a traditional IT infrastructure versus a new IT infrastructure based on virtualization. They both consist of an application, operating system, server, network and storage layer.

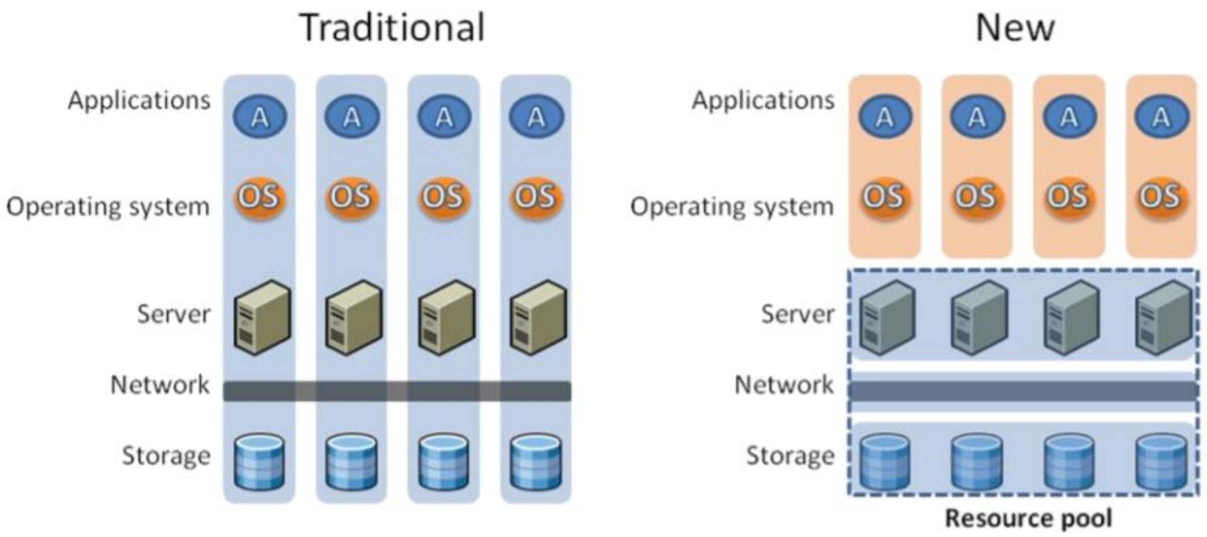


Figure 1.1:Traditional vs New IT infrastructure [3].

In the traditional infrastructure, one out of every layer component is connected. A single database is connected with a single server in which a single operating system is installed. Only the network layer has the ability to connect different storage and server systems together. The vertical blue pillars point out a specific type of configuration of storage, network, server, OS and applications. In this type, when a component in one of the layers has a failure, the whole pillar becomes dysfunctional. For instance, if the server has a hardware failure the operating system and the applications are down until it is repaired.

In the new infrastructure, the horizontal blue pillar that includes storage, network and server layers display a new configuration. These layers cooperate with each other and function as a pool of resources. Moreover, the orange vertical pillars indicate the software, which includes the operating system and the application, that is not tied to a specific server. Compared to the traditional infrastructure above, if a server has become non-operational the applications can continue to function on a different server from the resource pool.

1.2 Historical background.

The following table illustrates some of the most important accomplishments in the evolution of virtualization technology:

Year	Accomplishment
Early 1960's	IBM Virtual Machine – System /360 model67
Mid 1960's	IBM time sharing – IBM 7044(44X)
Mid 1960's	Led to widely used VM/timesharing systems – IBM VM/370
Mid 1960's	Introduce concept of hardware virtualization.
Mid 1970's	Virtualization well accepted by users of various operating systems.
1980& 1990's	Declined when low-cost minicomputers and personal PCs were introduced.
1990's	Explosion in numbers of servers per enterprise.
Late 1990's	Underutilized servers, deployment, update and support challenges. Security and Disaster recover issues.
1997 & 1998	Disco project – Multiple O.S. on single multiprocessor, led to development of VMware
2003	Development Xen (open source VMM)
2007	Microsoft Oracle, Red Hat & Sun introduce new virtualization capabilities.
2007 & beyond	Continuous growth and increased popularity

Table 1.1 : Historical background of virtualization [4].

1.3 Defining virtual machines.

Virtual machine (VM) is a necessary component in virtualization technology. It is a virtual environment that uses computing resources such as memory, CPU and its operating system.VM is an isolated virtual computer that operates on a "physical machine" or a host computer and behaves like a separate computer. A host computer can execute many VMs with different operating systems and applications. However, a VM is not aware of other VM' existence, instead, it has the illusion that it operates alone.

A VM is typically comprised of either a single file or a group of files that can be read and executed by the hypervisor.[5] Managing a virtual machine is similar to that of a standard folder as it can be easily moved, copied and deleted at will. For this reason, VMs are extremely portable.

Each VM utilizes virtual memory, virtual CPU, virtual Network Interface Card (NIC) and other types of hardware. To create this virtual version of physical hardware, a set of drivers is needed. A driver is a software that provides the proper information to the operating system in order to communicate with the computer hardware. When a VM is activated and online, the hypervisor automatically assigns memory, CPU and disk space processor capacity.

Usually, a virtual machine is depicted as a box with the initials VM, mounted on a hypervisor. That box consists of computational resources, a network, an operating system and applications running inside of it, as the following figure:

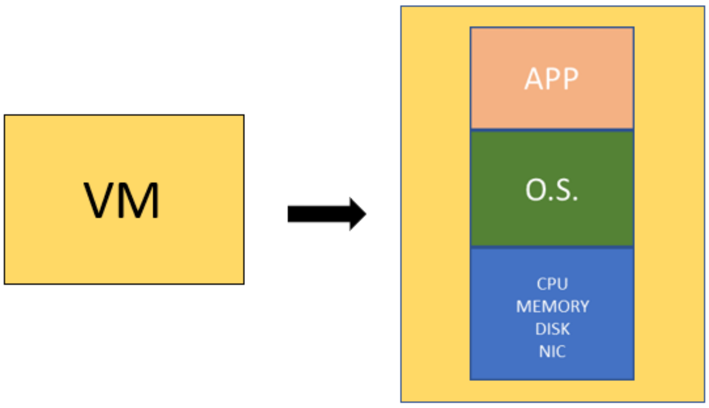


Figure 1.2. : Virtual Machine.

1.4 Hypervisor.

The hypervisor is a key-factor component in virtualization technology because it is software that is responsible for the creation and management of virtual machines. Without this virtualization layer, the implementation of a VM is not feasible. A hypervisor is a layer of software between VMs and the physical hardware. It is able to create and manage multiple VMs that are operating on the same physical server. A hypervisor distributes resources like memory and peripherals to the VMs. It is in charge of providing each VM with the illusion of being run on its hardware, which is done by exposing a set of virtual hardware resources (e.g. CPU, Memory, NIC, Storage) whose tasks are then scheduled on the actual physical hardware [6].In the IT world, the term Virtual Machine Monitor (VMM) is also used to identify a hypervisor. Hypervisors are categorized into two types: Type 1 and Type 2.

- Type 1 hypervisor is also called “native hypervisor” or “bare-metal” because it is installed directly on the hardware. His position is between the virtual machines and the underlying hardware. It is software that distributes system resources to virtual machines. This type of hypervisor is the most preferable from the companies because they are more efficient than the hosted hypervisors. Examples of bare-metal hypervisors are Citrix Xen-Server, Oracle VM, VMware ESX, Microsoft Hyper-V and ESXi.

- Type 2 hypervisor is also called “hosted” hypervisor. Its main difference between bare-metal hypervisors is that it is placed on top of an existing operating system, known as a host operating system. It provides the ability to use a different type of operating system on top of another operating system. For instance, if an employee, who uses a personal computer with Windows operating system wants to run an application in Linux, the hosted hypervisor can create a virtual environment with a Linux operating system on top of the Windows operating system and vice versa. Examples of hosted hypervisors are Oracle VM VirtualBox, Microsoft Virtual PC, VMware Server, VMware Workstation Player.

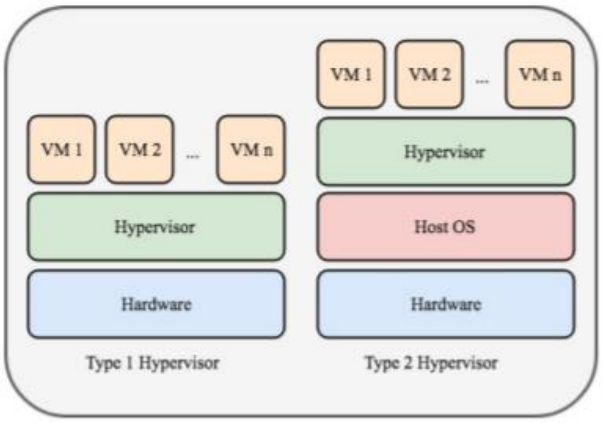


Figure 1.3: Type 1 and type 2 Hypervisors [7].

In contrast to type 2, type 1 hypervisors are more secure due to the separation from the operating system. Moreover, type 2 hypervisors have higher latency. Requests between hardware and hypervisor must traverse an extra layer of the operating system. To choose the right hypervisor that satisfies the needs of infrastructure, a comparison between their performance metrics is necessary. These include support for virtual processors, CPU overhead and amount of maximum host and guest memory. Moreover, a verification that the guest operating system is supportable by the hypervisor is also necessary.

1.5 Advantages and Disadvantages of Virtualization.

Following are some of the most recognized advantages and disadvantages of virtualization, which are explained in detail in the next chapters.

Advantages	**Disadvantages**
Efficient hardware utilization	Limitations
Portability	Staff specialization
High-availability	
Disaster Recovery	
Isolation of services	
Ease of testing in a development environment	

Table 1.2: Advantages and disadvantages of virtualization.

1.5.1 Advantages.

- Efficient hardware utilization: With virtualization, the need for a physical hardware system is reduced. Since a single physical server is able to host numerous applications, physical space, cooling, cabling and maintenance costs are reduced. With the use of a powerful machine we can create many virtual machines, thus saving physical space in a data center and energy to run the corresponding physical machines. Consequently, CAPEX and OPEX are reduced.
- Portability: One of the major advantages of virtualization is portability. Virtual machines can easily be copied and relocated from one VM to another.
- High-Availability: The Virtual machines' properties provide the ability to create services with high-availability specifications. Services in which the user does not perceive an interruption even in case of partial disasters.
- Disaster Recovery: Virtual machines also provide the ability to take snapshots of each machine. Snapshots are copies that reflect the state of a virtual machine at a particular point in time, maintaining and preserving the data and memory of the machine. By regularly taking snapshots and backups, services can be restored to an operational state within a short amount of time. In case of a disaster, we can use the backups to deploy new VMs and make our applications available again.
- Isolation of services: Server virtualization provides the ability to create separated virtual environments which run different applications. In the case where a single physical server is used, different applications increase the possibility of the services crashing with each other. Moreover, security vulnerabilities or possible failures in an application cannot affect services located on different virtual machines.
- Ease of testing in the development environment: Through virtual machines, we can copy an application, which is already running on another virtual machine, deploy it on a new virtual machine and either make changes or perform penetration tests. Additionally, we have the ability to create new applications or extensions of existing ones without creating a problem in the smooth operation of our infrastructure.

1.5.2 Disadvantages.

- Limitations: Virtualization is not compatible with every server and application. Therefore, the solutions provided by this technology cannot be introduced in all IT infrastructures.
- Staff specialization: Although virtualization services are widespread now, the creation and maintenance of such a complex infrastructure requires IT, staff, with expertise in virtualization technologies

1.6 Data centers.

A data center is a specially designed structure that is deployed to house an organization's IT operations equipment. A data centers mainly consist of computing, network and storage resources such as servers, firewalls, routers, switches and systems able to assist one or more organizations in smooth operation. Some of these systems are cooling systems, fire and smoke systems, power backup structures, ventilation and physical security systems. All these components interact with each other and they are inextricably linked to achieving the main objective of data centers, which is to process information. A data center stores manage and disseminate critical information, data and applications for one or more businesses. For this reason, the security of this infrastructure must be ensured and its management must be carried out by qualified and specialized personnel. The following figure depicts a typical data center room:

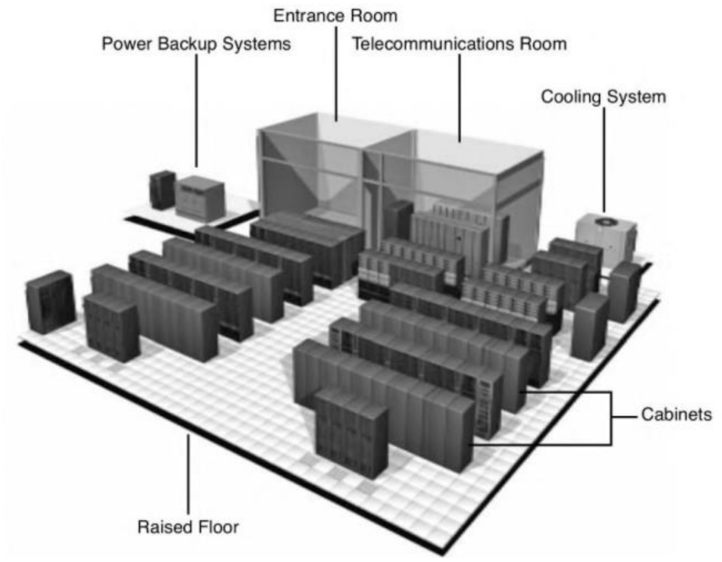


Figure 1.4 : A typical data center room [8].

However, modern data centers are made up of multiple of these data centers rooms. Due to the rapid growth of the Internet economy and the explosion of information technology data centers are key factors in this evolution and they have become the pillar in the IT world. Businesses have turned their attention on how to utilize data centers to increase their profits. It is not compulsory for a data center to be used only by a particular organization, nor is it compulsory for a business to use only one data center. The services provided by a data center that is useful for the operation and development of an enterprise can be summarised as follows [9]:

- Email and file sharing
- Customer relationship management (CRM)
- Databases and e-commerce
- Enterprise resource planning (ERP)
- Big data, machine learning, artificial intelligence

For this reason, the size of a data center and the facilities that provide have changed over the years. A typical categorization by size is shown in the following table:

Data Centers		
Size	Racks Crowd	Space(m^2)
Mini	1-10	1-25
Small	11-200	26-500
Medium	201-800	501-2000
Large	800-3000	2001-7500
Massive	3001-9000	7501-22500
Mega	>= 9001	>= 22501

Table 1.3: Data centers categorization by size [10].

A different way of categorizing data centers is the ANSI/TIA-942 standard. Through this standard, data centers are divided into 4 tiers. Each of them has its specifics and characteristics as is shown in the following table:

	TIER 1	TIER 2	TIER 3	TIER 4
Customer	startups	Small business	Large business	organizations
Availability	99.671%	99.749%	99.982%	99.995%
Downtime (hrs)	28.8	22	1.6	0.04
Component level redundancy	N	N+1	N+1	2N+1
Months to deploy	3	3-7	14-20	15-20
1st year of use	1965	1970	1985	1995

Table 1.4: Data centers categorization in tiers [11].

This tiering standard provides a better method to assess the quality and reliability of a data center's server hosting capability. In addition, businesses have a comprehensive view on what type of data center they need to choose, in order to be able to implement or transfer to a larger data center their applications and services, in case they tend to scale up. A data center classified as a tier 4, is a data center with the highest availability and the least downtime per year.

1.7 Data center virtualization.

As mentioned above, a data center is a physical infrastructure that contains mainly servers and, understandably, can become saturated. Due to the increasing amount of data used daily and the use of the internet by more and more businesses, the need for larger data centers with faster speeds is increasing. However, it is not feasible for a data center to expand indefinitely and the solution to this problem is brought by virtualization technology. Through server virtualization, a machine is able to operate as if it were 2 or more individual machines. In this way, the number of machines required to implement different processes is reduced. In this way, the existing hardware can accommodate more applications and be used to the fullest extent. Using high-speed data transfer technology, it is possible to retrieve information stored in remote locations as quickly as if it were stored on a local server. In this way, the information is extracted from the location, whereas in traditional data centers, in order for information to be accessible by staff, it was stored on-site. Network virtualization provides the ability to create multiple logical networks, as the downstream network is treated as a pool of carrying capacity that can be continuously reused on demand. Storage virtualization provides the ability to create pools of storage resources, separating the storage management software from the hardware, in order to achieve scalability and flexibility.

A physical data center has the ability to operate in parallel with a virtual data center. This interaction of physical and virtualized resources has led to the creation of Hyperconverged infrastructure (HCI). Hyperconverged infrastructure (HCI) is the integration of different technologies such as compute, storage and storage networking, virtualization, data networking and automation, all under the umbrella of software-defined storage (SDS) and software-defined networking (SDN)[12]. For the creation of legacy data centers, personnel with different specializations are used, for example, system and storage administrators, network engineers, software virtualization engineers. As a result, the infrastructure itself is divided into separate silo groups, which aim to solve their problems and optimize their own processes, without worrying about what is happening in the rest of the infrastructure. Through HCI, all silo groups and different technologies are integrated, enabling centralized management, easier scaling, and flexibility.

The widespread use of virtualization in data centers has paved the way for the implementation of modern cloud IT infrastructures. In cloud computing infrastructures, providers make use of virtualization technologies to offer flexible, on-demand provisioning of resources to customers. Usage of these resources is charged on a pay-for-use model[13]. Data centers that are implemented, managed and used by the company itself are called on-premises data centers. In contrast, cloud data centers are managed by the cloud company and enable their customers to operate their services and manage their data within a virtual infrastructure. Therefore, companies that do not want to invest in creating an on-prem data center, have the option of using infrastructure as a service from a cloud provider. Cloud-based services create an innovative growth model from which all businesses of any size can benefit. Moreover, the types of cloud provided by cloud providers are public, private, hybrid and community cloud.

1.8 Benefits of virtual data centers.

The introduction of virtualization technology in data centers can bring several benefits, both for the infrastructure itself and for the businesses that use them.

- Reducing the costs: With virtualization, fewer servers can support several applications and services, while with the same hardware more data can be stored. Therefore, there is no need to buy more hardware and therefore the costs of purchasing and maintaining it are reduced. With virtualization, less equipment is required and therefore less space is needed. Within a data center, due to the wide range of equipment that is being used, the heat in data centers rooms is increasing rapidly. As the equipment grows, so does the number of cooling systems in order to maintain the temperature at an ideal level. In addition to saving space, virtualization also reduces the number of cooling systems and thus the amount of electricity required. In general, the carbon footprint of data centers is also decreasing.

- Disaster Recovery: Through the virtual data center, a disaster recovery plan becomes an easier process. The virtual machines can take snapshots of their operations. Therefore, in the event of any malfunction, the virtual machines can easily retrieve their last snapshot and continue to operate normally. In addition, moving the virtual machines to a different location can be a simple task.
- Better scalability: Virtual data centers provide the ability to companies to become more flexible as their demands and needs grow. Companies do not have to expand their physical infrastructure by buying more equipment as both new applications and new optimizations can either be tested at the primary level or be implemented and integrated into the infrastructure through virtualization and the efficient management of VMs.
- Customer growth: A virtualized data center is able to support more than one business and be able to meet the needs and requirements agreed through SLAs, in order to deliver the best possible performance. Service Level Agreement (SLA) is a fundamental contract between service consumer and service provider that defined the qualities of the agreed service[14].

2 Cloud Computing.

In this chapter, an extensive analysis will be presented on the definition of the Cloud Computing technology, and its basic characteristics. In addition, a historical background and the needs for which this technology is used by organizations will also be presented. Both Cloud service and Cloud deployment models will be analyzed and finally, the advantages and disadvantages of Cloud Computing will be highlighted.

2.1 Defining Cloud computing.

As with virtualization, the definition of cloud computing is not specific. According to the National Institute of Standards and Technology (NIST): Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [15]. This definition has dominated the world literature, yet NIST has been forced to change it several times. Similar to nature, the cloud has no specific location, its shape is constantly changing and its function is several, so in the field of IT, the cloud is a construct that actually harbors many services and is not located anywhere in particular. The cloud metaphor was used to represent the internet in architecture books and their blueprints. With the rise of the cloud computing model and its separation as a separate part of IT, the cloud refers to a discrete IT environment designed to provide scalable and measurable IT resources remotely [16].

Figure 2.1: Cloud [17].

Through this technology, the user does not have to provide his/her own computing power and resources but only a broadband connection in order to connect to the provider and connect to the cloud services offered. Moreover, the user doesn’t need to have specialized knowledge as the management of the environment and the system he/she uses are also managed by the provider. In addition, the NIST definition states that: This cloud model is composed of five essential characteristics, three service models, and four deployment models[15], which are depicted in the following table.

CLOUD COMPUTING		
CHARACTERISTICS	DEPLOYMENT MODELS	SERVICE MODELS
Broad network access	Private Clouds	Infrastructure as a Service (IaaS)
Measured service	Hybrid Clouds	Software as a Service (SaaS)
Resource pooling	Community Clouds	Platform as a Service (PaaS)
On-demand self-service	Public Clouds	_
Rapid elasticity	_	_

Table 2.1: Cloud Computing [15].

Professor Ramnath Chellappa gave the first reference to the term cloud computing as:" a computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone". IBM has defined this technology like Cloud computing as on-demand access, via the internet, to computing resources—applications, servers (physical servers and virtual servers), data storage, development tools, networking capabilities, and more—hosted at a remote data center managed by a cloud services provider (or CSP)[18]. Through cloud computing, a user, once connected to the internet, has the ability through a simple browser to access data, files, programs and third-party services. The term cloud computing is inextricably linked to the hardware and software of data centers that host applications delivered over the internet as services. The volume and type of services and data to which each user has access are determined by the service level agreement (SLA) between him/her and the provider. Generally, in cloud computing, the pay-for-use policy prevails.

2.2 Historical background.

Cloud computing as technology was not created recently. Its start dates back to the creation of the first internet, but the first reference to it was only in 1997. Since then, companies such as Google and Amazon have introduced this model into the global economy, to the point that the market associated with cloud computing is estimated at $370 billion. In the following table, highlights of cloud computing’s evolution over time are depicted and these are retrieved from [19][20].

CLOUD COMPUTING HISTORY	
YEAR	ACCOMPLISHMENT
1960s	John McCarthy introduces mainframe timesharing and public utility computing.
1969	ARPANET is developed by J.C.R. Licklider.
1970	Launch of Virtualization Software.
1991	Launch of WWW (World Wide Web).
1997	First definition of Cloud Computing by Prof. Ramnath Chellapa.
1999	SalesForce offers business apps over the Internet (arrival of SaaS).

2006	Amazon creates AWS and introduces Elastic Compute Cloud (EC2).
2008	Google releases GCP and Google App Engine.
2014	$170 billion estimated global cloud spending.
2014	Docker Container Services by Windows server and EC2.
2015	Google introduced VMs with half the cost, since the year of the first deployment.
2016	Google Cloud provides computing, data, storage, data analytics and machine learning
2017	GCP offers to pay pre second billing for VM.
2017	AWS offers to pay pre-second billing for Linux VMs.
2019	CDN market is estimated to be more than $12billion.
2020	Growth Global Market for Cloud Computing is estimated to exceed over $370 billion.

Table 2.2: Historical background of Cloud Computing.

2.3 Needs for Cloud computing in businesses.

Before we start to analyze the characteristics, service and deployment models of cloud computing, it is essential to mention some of the main reasons[20] why the largest companies have turned their interest in cloud technology. The overriding objective of these companies is to integrate this technology in order to meet the requirements created for the automation of their businesses. In addition, other companies were motivated to delve further into this technology. As a result, they have become cloud infrastructure providers with the main objective of implementing products that meet the needs of consumers in the market. In this chapter, three main needs for cloud computing in businesses will be analyzed, which are: capacity programming, reduction in costs and business agility.

- Capacity programming: The main objective of an organization for its proper and smooth operation is to respond to the demand and needs of its customers throughout their cooperation. The main objective of an organization for its proper and smooth operation is to meet the demand and needs of its customers throughout their cooperation. To achieve this, every available resource of the organization should offer its maximum capacity to the infrastructure. By capacity we mean the percentage of the maximum work of an IT resource that it can deliver at a given time. The balance between demand and capacity must be constantly maintained as any mismatch between them leads to problems and malfunctions. For any organization, it is imperative to plan its capacity to cope with future needs for resources, products and services. This is called capacity programming. It is a difficult task for any organization as it has to predict and estimate the variation of the utilization load. Apart from that, it is also necessary as it can save a company, for example, from naive investment in equipment or force it to invest in order to balance future demand and avoid losing customers.

- Business agility: An organization is often faced with challenges and changes that come either from external or internal factors. It is an imperative task for every organization to be able to assess these changes and be able to adapt and evolve to cope with them successfully. Business agility is the measure of an organism's receptiveness to changes. This section is largely linked to the available IT resources of an infrastructure. It is well known that these resources will eventually become saturated and every business has to take appropriate measures. However, many times, a business does not have the required budget for scaling its hardware, leading to unresponsive utilization fluctuations.

- Reduction in costs: Principal consultant of every company is to increase profit in combination with cost reduction. In order to achieve this, each undertaking shall turn to reduce its CAPEX and OPEX. In companies that deal with IT infrastructure, the expansion of this infrastructure is a scenario that they try to avoid and find other solutions. This is because this expansion burdens the company both with the costs of acquisition and ownership of the new infrastructure, as well as with additional operating costs. Indicatively, some of these operating costs have to do with technical staff, electricity and refrigeration costs, upgrades and security.

2.4 Cloud Computing Characteristics.

According to NIST, Cloud Computing’s characteristics are listed and described below:

- Broad network access: A cloud service must be accessible from the whole network. Depending on the different needs of consumers, the architecture of the cloud service should be adapted to them. Furthermore, to achieve broad network access, different communication, security, and interface protocols should be supported.

- Rapid elasticity: The ability of a cloud to be able to automatically scale IT resources and respond to emerging needs promptly is called elasticity. It is obvious for a provider that the more IT resources it has, the more elasticity it can offer to its consumers. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time [15].

- On-demand self-service: Every consumer has the freedom to be able to select and access cloud-based IT resources on their own. In this way, there is no need for human intervention with each service provider as the IT resources provided are assembled and can be used automatically.

- Multitenancy and Resource pooling: The ability of a software program to allow a snapshot to simultaneously serve different and isolated consumers is called multiple leasing. Thus, providers using a multiple leasing model create pools of computing resources and can dynamically meet the needs of multiple consumers. The consumer does not know the exact location from which the requested resources are provided. Multi-tenancy models are based on virtualization technology.

- Measured service: With this feature, a cloud platform has the ability to control and monitor the use of its resources mainly by consumers. Through this measurement, it can charge the consumer accordingly. In addition to monitoring with a view to billing, general monitoring of the resources used by both consumers and providers is carried out in order to draw statistical conclusions.

2.5 Cloud service models.

The combination of IT resources offered by a cloud provider constitutes a cloud service model. Cloud service models are divided into three well-known categories which can all be accessed via the internet. The pricing of cloud services depends on the requirements agreed between customers and providers. These are: Software-as-a-Service (SaaS), Infrastructure-as-a-Service (IaaS),Platform-as-a-Service (PaaS).

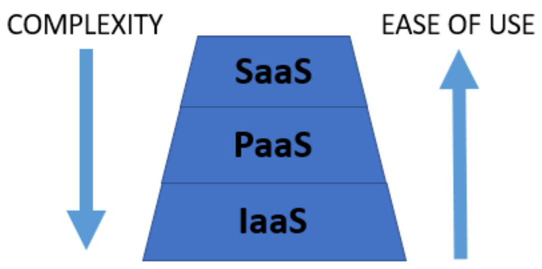


Figure 2.2: Cloud service models pyramid.

In Figure 2.7, we observe that the complexity of those services increases as we move towards to the bottom of the pyramid, while the ease of use of the services increases as we move towards to the top of the pyramid. SaaS runs on top of PaaS which in turn runs on top of IaaS[21]. However, with the evolution and the increasing use and adoption of cloud computing by organizations, some variations of the three basic services have been created. Indicatively, some of them are:

- Storage-as-a-Service.
- Data-center-as-a-Service.
- Security-as-a-Service.
- Function-as-a-Service.

2.5.1 Infrastructure-as-a-Service (IaaS).

In type of service model, the provider offers its cloud consumers virtual computing resources by creating an autonomous and non-default environment ready for use. Using these provided resources, cloud consumers can implement and develop their applications, without being obliged to buy servers or create data centers in order to set up their infrastructure. Instead, they pay the cloud provider for as long as they use the existing infrastructure offered to them as a service, reducing the capital expenses. The resources offered through this service are not structured and organized in advance, as they deliberately allow customers to have a high degree of responsibility for their management.

2.5.2 Platform-as-a-Service (PaaS).

PaaS benefits from the virtualized resources of IaaS. PaaS providers offer programming language, software tools, access to APIs and development middleware. The cloud consumer of a PaaS, does not deal with the management of the resources and the configuration of the development environment. Instead, he/she utilizes this software environment to develop and implement cloud applications and custom solutions. The cloud provider of PaaS is responsible for the maintenance of the operating system and intermediate services such as servers and databases.

2.5.3 Software-as-a-Service (SaaS).

A cloud provider offers an application software that is accessible either via a dedicated desktop client or an API or a simple web browser. The cloud consumer has minimal management control over the resources and infrastructure in general. The environment offered to them is completely predefined and cannot be configurated from the cloud consumer. Only the cloud provider is entirely responsible for the proper functioning of the infrastructure.

2.5.4 Comparison of cloud services.

In the following table, we observe for each service which segments of the infrastructure can be managed by the provider or by the consumer of the cloud service. The boxes in the table with the blue color refer to the consumers while the orange color refers to the provider.

ON-PREMISES	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Operation System	Operation System	Operation System	Operation System
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Table 2.3: Cloud services comparison [22].

The less administration required by the user, the more ease of use of the service. Another comparison concerns the users of each service. Specifically, an IaaS is used by IT administrators, a PaaS by software developers and an IaaS can be used by any end-user.

2.5.5 Cloud service providers.

In the table below, some of the most popular cloud service providers are listed, according to the service model they offer.

Cloud Service Provider (CSP)		
IaaS	**PaaS**	**SaaS**
Amazon AWS	Amazon AWS IoT Core	SalesForce
Microsoft Azure	Microsoft Azure IoT	Oracle
GoGrid	Google Cloud IoT	NetSuite
Vmware	Google App Engine	IBM
Rackspace	GAE	Google App

Table 2.4: Cloud service providers [23].

2.6 Cloud deployment models.

Choosing the type of cloud that can be used to deploy a cloud computing solution is an important process. Cloud deployment models are divided into 4 types. These models are public, private, community and hybrid. Cloud infrastructure can host and operate any of these 4 models.

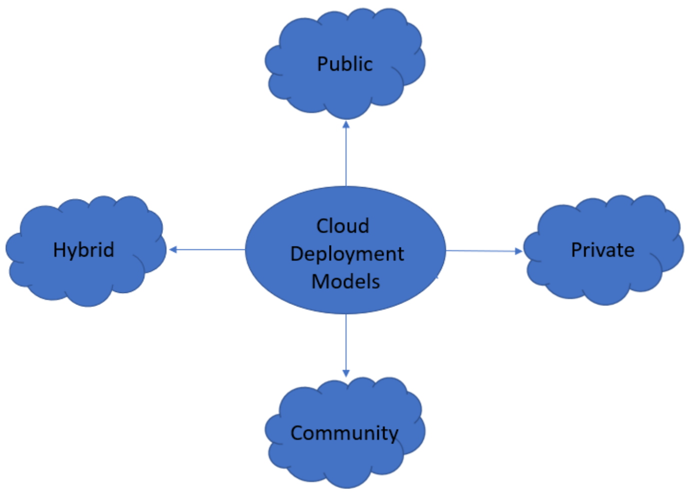


Figure 2.3: Cloud deployment models.

2.6.1 Private cloud.

This cloud is created and used exclusively for a single cloud consumer or a single organization. Apart from the cloud owner, third-party organizations can also have access to it solely to maintain its proper operation and manage it on behalf of the owner. Its infrastructure is used and managed for only one organization, which makes it feasible to maintain a sufficient level of control, privacy, and security, as only the members of the organization that uses it can access it.

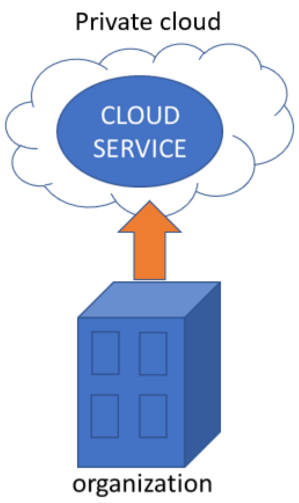


Figure 2.4: Organization using a cloud service from its own private cloud.

2.6.2 Public cloud.

It is a model that consists of cloud services managed and hosted by cloud service providers and produced by third-party organizations. Providers are responsible for implementing, installing, maintaining and ensuring the smooth operation of the model. Users of a public cloud, unlike a private cloud, can be more than one. The choice of services, resources and the time for which they are used, determines the final charge. Compared to a private cloud, the public cloud has less security and reliability while the applications it offers are more vulnerable to malicious attacks. Moreover, a public cloud is more scalable and location independent while the private cloud is location restricted.

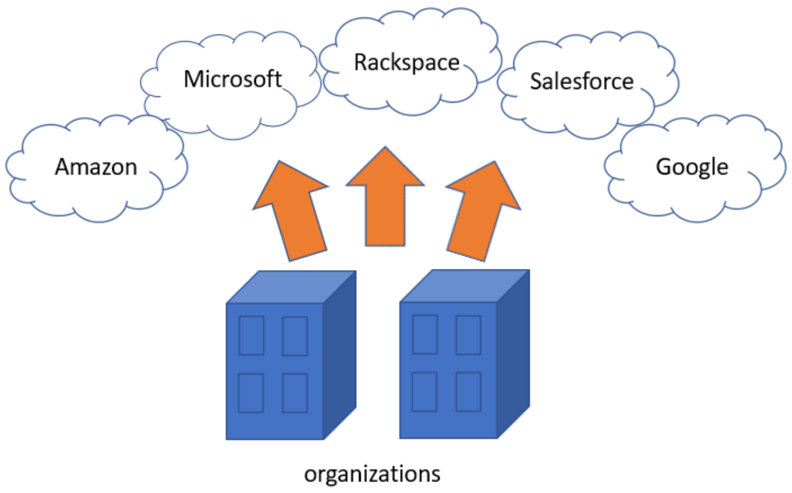


Figure 2.5: Organizations using cloud services from different providers.

2.6.3 Community cloud.

This type of cloud is often referred to as a semi-private cloud as it is created and used by several but predefined organizations with similar requirements and policies. This creates a private cloud where its infrastructure can be hosted or managed by either member of the community using it or from a third-party provider.

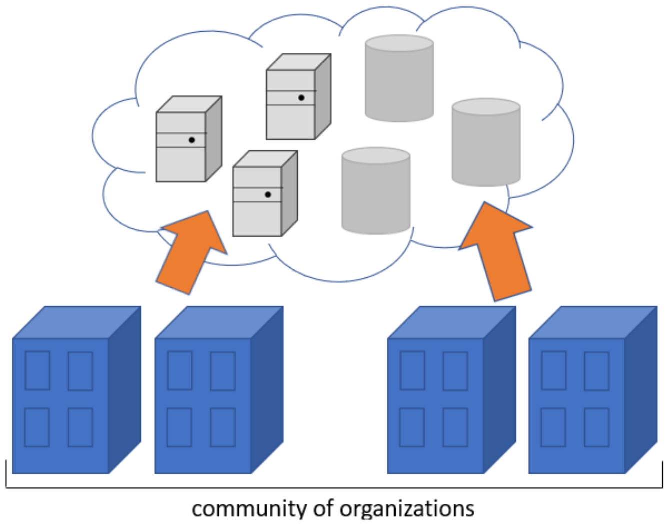


Figure 2.6: A community of organizations which are using resources from a community cloud.

2.6.4 Hybrid cloud.

This type of cloud is the most complex as it consists of a combination of two or more development models (public, private, community). Each member remains a unique entity but is bound to others through standardized or proprietary technology that enables application and data portability among them[]. The choice of models to be combined depends on the needs and requirements of the organization. Each hybrid cloud consists of at least one private and one public cloud. The private hosts core activities while the public for less important services.

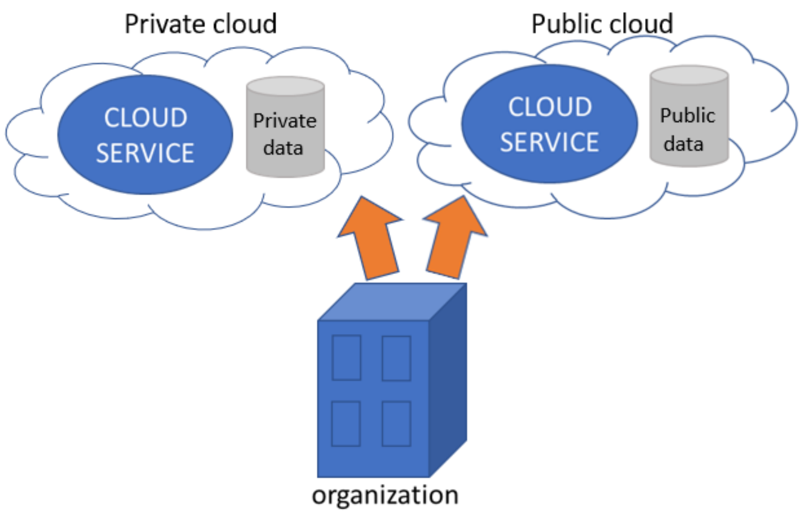


Figure 2.7: An organization that uses a hybrid cloud consisting of a private and a public cloud.

2.6.5 Comparison of cloud deployment models.

Each of the cloud deployment models has its characteristics and some advantages that distinguish between them. For any organization, choosing the right cloud model to deploy a cloud computing solution is critical as it needs to serve the best of its ability. In the first of the next two tables, some of the main advantages for each cloud are illustrated, while in the second table comparison is made between them, to understand where each model can be used better.

Public	**Private**	**Community**	**Hybrid**
Available for all	Only for private use	Ideal for collaboration	Highly-flexible
Scalable	Scalable	Cost saving	Scalable
Reliable	Reliable	Cross-organization data security and information privacy	Requires strong cloud expertise
Cost-effective	Customizable		

Table 2.5: Major advantages of cloud deployment models [24].

	Public	**Private**	**Community**	**Hybrid**
Set up and use	Handle in-house	Requires IT professionals	Requires IT professionals	Requires IT professionals
Data control	Low	High	Medium	Medium
Security and privacy	Low	High	Medium	Varies from low to high
Overall reliability	Medium	High	Medium	Medium to high
Cost	Low	High	Variable	Variable
Flexibility and Scalability	High	Medium	Medium	Very High
Hardware	Third-party	Variable	Variable	Variable

Table 2.6: Comparison of cloud deployment models [24].

From these tables, we conclude that as the deployment model becomes private, security, customization, data control and reliability and the cost of using them increase, as shown in the next figure.

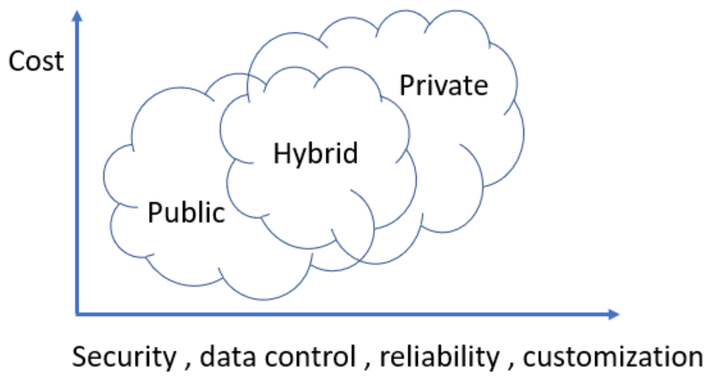


Figure 2.8: Public vs Hybrid vs Private

2.6.6 Relationships between cloud deployments and cloud service models.

Each of the four major deployment models cannot be implemented by any of the three service models. In particular, an IaaS can support either a private, community or public cloud. A PaaS can support either a public, community or hybrid cloud. A PaaS can support either a public, community or hybrid cloud. A SaaS can support either a private, community or hybrid cloud. For each service model, there is a limitation, however, the options for deployment model are still many. As mentioned in chapter 2.1 the NIST definition contains five characteristics, three service models and four deployment models. All of these components that describe cloud computing according to NIST, do not operate independently but are necessarily interrelated and connected to each other. The figure below illustrates all these components and how they are connected to each other.

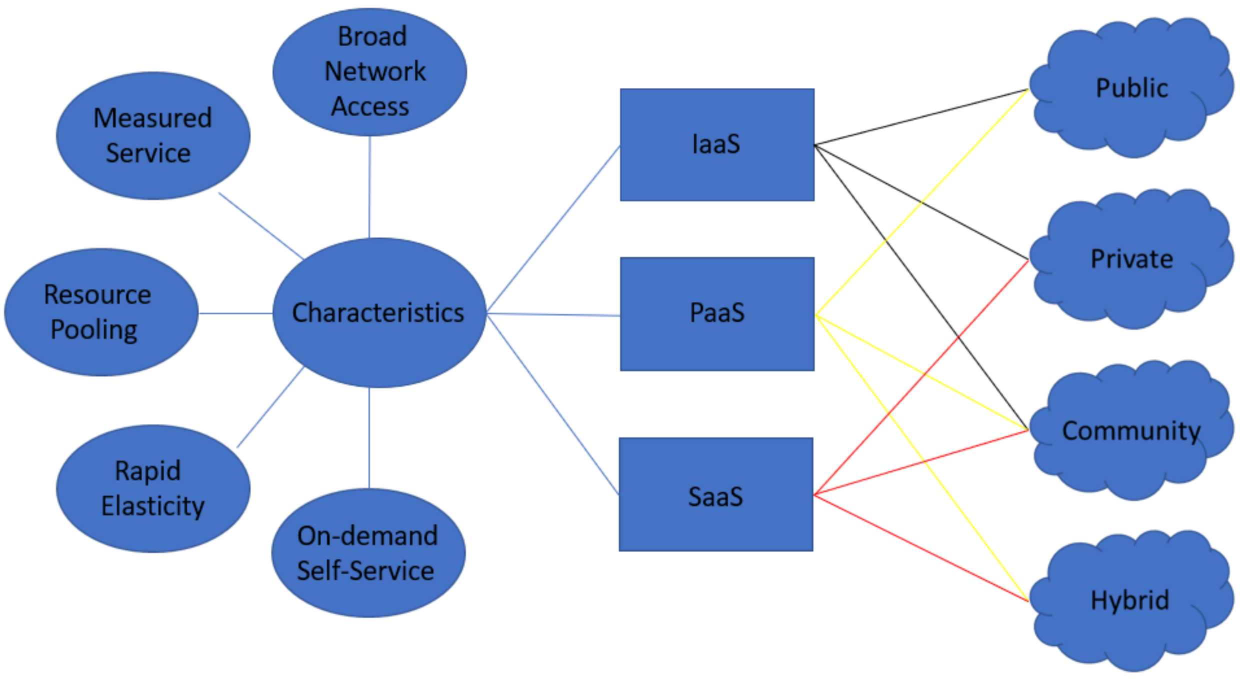


Figure 2.9: Relationships of cloud computing components according to NIST.

From this figure, we can easily understand that a cloud-based strategy can have different configurations in order to meet the needs of an organization. It is not unlikely that an organization will use as a pilot a combination of service and deployment models, e.g. SaaS and public cloud, and if that pilot proves to be satisfying, can either scale it up or use another pilot. Furthermore, there is the ability to use multiple deployment models to support one or more service models.

2.7 Advantages and Disadvantages of cloud computing.

Since the arrival of this technology, several studies have followed that list its advantages and disadvantages. The advantages of cloud computing are the right motivation for the migration of a company or an organization to the cloud while highlighting the disadvantages enables them to devise policies and solutions to prevent any malfunctions and forfeitures. The following table contains some of the main advantages and disadvantages which will be further analyzed in the following subsections.

Advantages	**Disadvantages**
Cost Reduction	Legal Issues
Increased Collaboration	Security
Time and Location Independent	Data Transfer
Disaster Recovery	

Table 2.7: Advantages and Disadvantages of cloud computing.

2.7.1 Advantages of cloud computing.

- Cost Reduction: One of the main incentives for moving to the cloud is to limit and reduce the costs of an IT infrastructure. This is because there is no compelling need to purchase hardware equipment and the necessary software to operate, while at the same time the daily costs for maintenance and electricity procedures are considerably reduced.

- Increased Collaboration: As mentioned in chapter 2.8, there are two development models which allow to be used and be accessible by more than one organization. In this way, those organizations that decide to adopt a cloud-based strategy and migrate to the cloud, gain the tool of collaboration with other organizations. This is because they can easily and quickly share and use data and services of different and multiple organizations.

- Time and Location Independent: With cloud computing, all data, services and necessary information are available all the time and can be utilized from anywhere as long as there is an internet connection. In this way, employees are enabled to work from anywhere, which leads to increased productivity.

- Disaster Recovery: Cloud computing can provide mechanisms for automated scheduled network-wide backup systems in order to store the data in off-site data centers [25].

2.7.2 Disadvantages of cloud computing.

- Legal Issues: Storing data and information in virtual data centers located in different locations around the world is the underlying idea of cloud computing. However, there are different approaches to access control and the fact that no similar and global set of regulations have yet been agreed upon is a key factor why many organizations are not migrating to the cloud.

- Security: Another main discouraging factor of migrating to the cloud is security concerns. The interception of sensitive data stored on a virtual server is one of the biggest problems.

- Data Transfer: In cloud computing, a remote server is used to store all the data. Therefore, the file transfer time for large chunks of data can become inconvenience for users.

3 Containerized technologies: Kubernetes and Docker.

In this chapter, an extensive analysis will be presented on the definition of the containerization technology. In addition, the differences between virtual labs and containers will be presented and the Docker and Kubernetes platforms will be extensively reviewed.

3.1 Defining Containerization and Containers.

Containerization is a type of operating system-level virtualization [26] and its fundamental concept is to make virtual instances, share a single host OS and relevant libraries, drivers or binaries[27]. The name containerization comes from the way in which container shipping is standardized, where containers are packages that can contain different contents but can be transported by different means. Thus, containers are light weighted packages containing software code along with all its dependencies so that it can operate in any computing environment. The following picture illustrates the difference between an application running with and without a container.

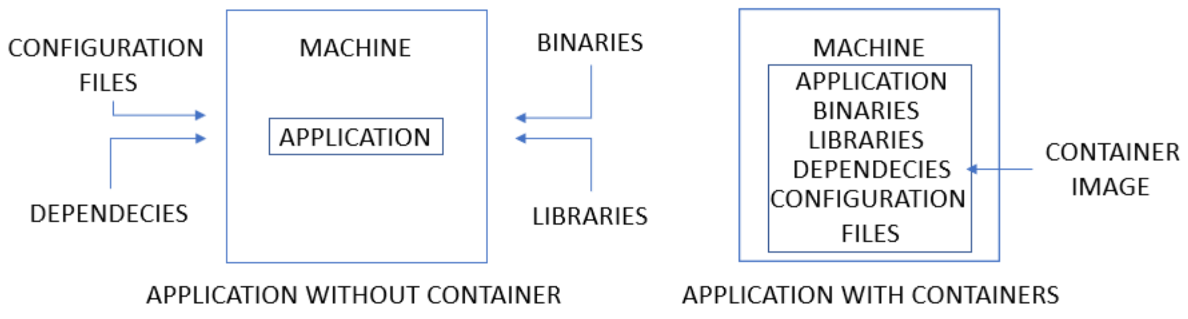


Figure 3.1: Application with or without containers.

In order to install and run an application, it is necessary that the machine on which it will run, has the required libraries, dependencies, configuration files and binaries. The same application, in order to run on a different machine must also have all the libraries, dependencies, configuration files and binaries available on that machine or the application will not run. This problem is solved through containerization, by creating a container image that contains everything needed to run the application to any machine. For example, in order to run Jenkins, it is required java to be running on the machine, whereas, with the use of containers, it is not compulsory to install java on the machine, as the container image will contain everything needed to execute Jenkins. The following picture is an overview on the architecture using containers for application development:

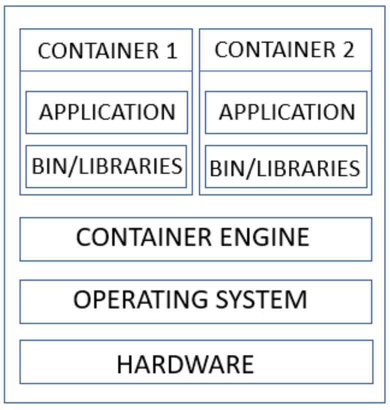


Figure 3.2: Container architecture.

Each container, containing its application and its dependencies, operates on top of a container engine, which is responsible for managing them. Both containers and container engine operate on top of a single operating system kernel which runs on top of the hardware. Two features that allow a physical machine to host different containers are namespaces and cgroups:

- Namespaces: Since containers share the same kernel, namespaces are responsible for separating resources in a way that each container appears to have its own separated and isolated resources. These could be process IDs, host names or network interfaces. It is quite common for different containers to run the same process. Through namespaces, while the process IDs are similar, the containers do not collide.
- Cgroups: It is a Linux Kernel mechanism responsible for dynamically allocating resources to the up and running containers. Containers cannot exceed the resource limits that cgroups place, only system administrators have the ability to configurate these limits on network or CPU memory. In contrast to namespaces, that deal with a single process, cgroups allocate resources for multiple processes.

3.2 Virtual Machines vs Containers.

Both virtual machines and containers are technologies able to deploy applications but they appear to have several differences. The following picture is an overview on both virtual machines and containers architecture.

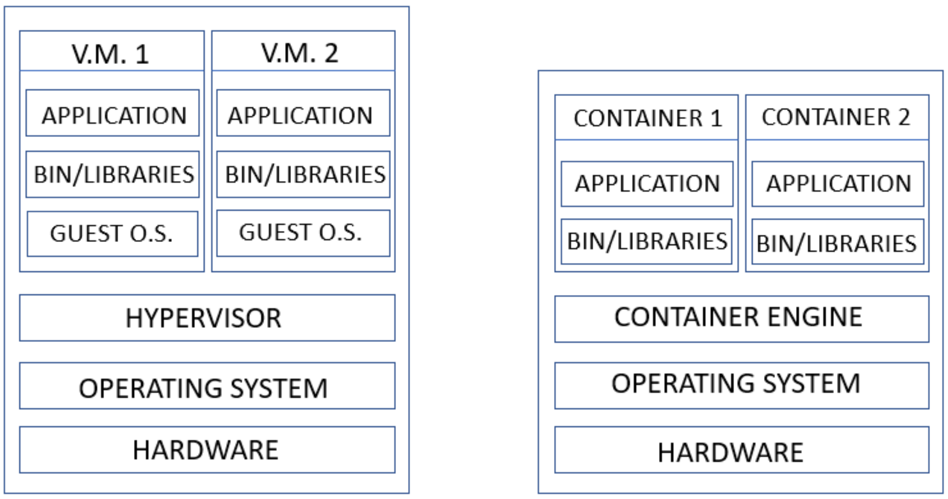


Figure 3.3: Virtual machines vs Container architecture.

From this figure we can retrieve three major differences. Firstly, a V.M. contains a guest O.S. which gives it the ability to run a different O.S. from the host machine but increases the size in space. In contrast, each container does not have a guest O.S. but shares a common O.S. so it is more lightweight. Secondly, containers do not require hypervisor. For a V.M. environment, the translation of a V.M. instruction to a host excecutable instruction is made through hypervisor. Containers do not need any priviledged instruction and communicate with the O.S. through system calls. As a result, containers architecture is reduced by one layer.Thirdly, each V.M. has a separate image file, which is isolated from the image file of another V.M. Container images are not only not isolated but also shared with each other.For example, one container image can be the basis for the creation of another as they are created in a layered manner. Taking these main differences as a starting point, we arrive at the following table which shows more differences between them:

Virtual Machines	**Containers**
Hardware level process isolation	O.S. level process isolation
Complete isolation from the host	Containers share resources with the host O.S.
Each V.M. has a separated O.S.	Each container share O.S. resources
Boot up in minutes	Boot up in milliseconds
More resource usage	Less resource usage
Pre-configured V.M.s are hard to find	Pre-built containers already available
Customizing pre-configured V.M.s require work	Building a custom setup is easy
Bigger size	Smaller size
They can easily be moved to a new host	Containers can be destroyed and recreated
V.M. creation requires time	Container creation requires minutes
Virtualized apps are harder to find and take more time to install and run them	Containerized apps can be found and installed within minutes

Table 3.1: Virtual machines vs Containers.

Compared to virtual machines some key advantages that containers provide are:

- Portability: Containers can be executed virtually over a wide range of computing and operating systems through virtual or physical machines, either by cloud computing technologies, or locally on each engineer's computer.
- Separation of responsibilities: Containers provides a clear segmentation of layers of an application's processes. Developers are concentrated on the process of software development and its dependencies while systems management engineers on the process of building and deploying the application.
- Application isolation: Containers provide operating system isolation for each application in each container, since there is virtual memory space for the memory,computing power, network and storage space that the application at the operating system level.

3.3 Docker

One of the technologies used to develop, run and deploy services and applications in containers is Docker. Docker is an open-source platform through which infrastructure and application management are managed in a similar way. Moreover, docker allows the separation of infrastructure with applications which leads to faster software delivery. In addition to the Docker platform, docker also provides a wide range of tools used to manage the lifecycle of Docker containers. The programming language in which docker is written in Go.

Figure 3.4: Docker's official logo [28].

3.3.1 Reasons to use docker.

1) Consistent and quick delivery of your application: Docker allows developers to utilize local containers and work with them in standardized environments. Developers can write code and collaborate with other developers by sharing their work and progress through docker containers. In addition, developers can push an application to a test environment and run either manual or automated tests. They can fix bugs in the development environment and when the result is acceptable, they can deliver the application to the client as easily as a simple push of an updated image to the production environment.

2) Dynamic scaling and deployment: Docker containers can run on cloud providers, on a virtual or physical machine, on a developer's local laptop, or even on a mix of environments. The docker platform in combination with the highly portable and lightweight docker containers have the ability to scale up or down applications or services in real time, depending on the demands. As a result, the workload can be dynamically managed via Docker.

3) The same hardware supports multiple workloads: Compared to hypervisor-based virtual machines, Docker uses less compute capacity, so it can support more containers. It is a more efficient and sustainable solution for less resource-intensive infrastructure and high-density environments as it is fast and lightweight.

3.3.2 Docker architecture.

The architecture that docker uses is client-server. The following figure displays the Docker architecture, the main components that make up the Docker architecture and how they communicate with each other.

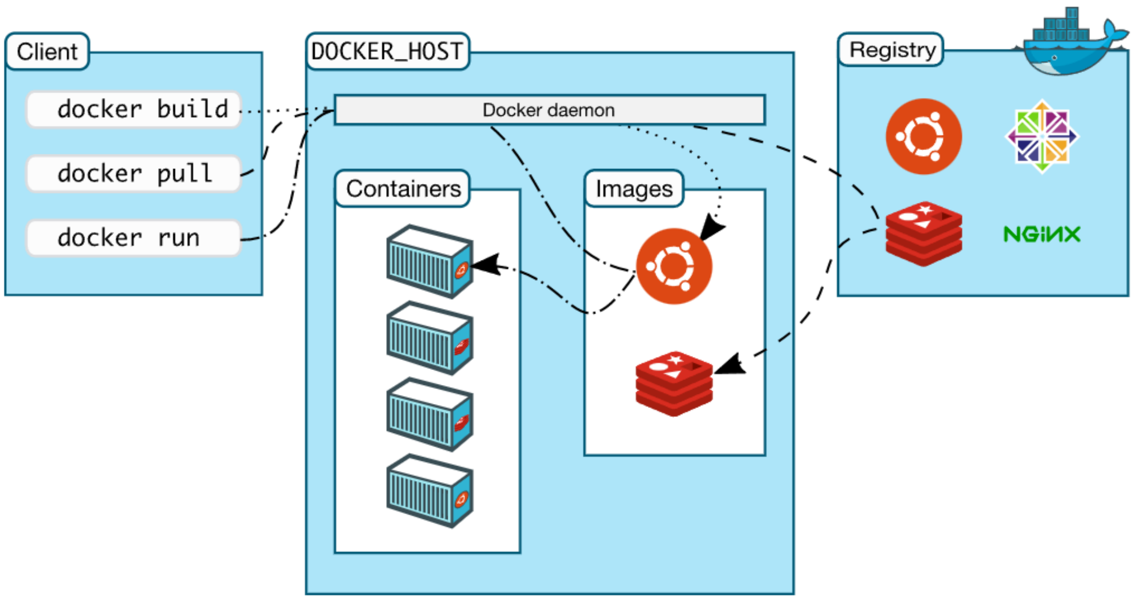


Figure 3.5: The Docker architecture [28].

The three main components are docker client, docker host and docker registry. Docker client contains a set of docker commands and communicates with docker host via docker daemon. Docker daemon is responsible for deploying, running and distributing Docker containers. In addition, another important work is to publish docker images to the registries. Docker daemon and Docker client can operate on the same system or a Docker client can be connected to a remote Docker daemon. The communication between these two is achieved via REST API, over a network interface or UNIX sockets.

- Docker daemon: Also known as dockerd, Docker daemon is responsible for managing Docker objects and responding to Docker API or CLI requests. These objects could be containers, networks, volumes and images. A Docker daemon can interact and communicate with other daemons in order to manage Docker services. Once the Docker client is prompted to create or pull an image from a registry, the Docker daemon creates a working model for the container using the built file. Also, the built file contains instructions to preload files before the container is executed and instructions to send to the local command line after the container is built.

- Docker Host: Docker Host includes Docker daemon, images, network, storage, containers and is the environment that takes on the responsibility to execute and run the applications.

- Docker client: It is the primary way that users interact with Docker. A Docker client can communicate with multiple daemons. When a user uses a command, for example, docker built, the docker client sends that command to the daemon and executes it.

- Docker registry: A Docker registry contains repositories of Docker images. Docker's public registries, which are accessible by any user, are Docker Hub and Docker Cloud. A Docker user can also use his/her own private registry. The commands used to download an image from a configured registry are docker run or docker pull. Docker push command is used to upload and store an image to the registry. By default, Docker Hub is the registry where Docker searches for images.

3.3.3 Docker image and Docker container.

A Docker image is a read-only template that contains all the proper instructions in order to launch a Docker container. Also, an image contains all the metadata related to the needs and capabilities of the container. Each image uses another existing image as a base image and adds some extra customization on top of it. An image consists of several layers. Docker utilizes Union File System (UFS) in order to combine all these layers into a one unique image. A user can either download the images from a Docker registry and use them or create their own. Creating a docker image is carried out through a set of steps called instructions. These instructions are stored in a specific Docker folder known as Dockerfile. Each of these instructions creates a layer in the image. When a user wants to build an image, Docker reads its Dockerfile and returns the final image.

A Docker container is an encapsulated environment that contains everything that is required for an application to run. Docker container is an executable instance of a Docker image. A Docker client can start, delete, stop or remove them. If a container is stopped without static storage space being created for it, then all information about it will be deleted. When Docker runs a container from an image, it adds a read-write layer on top of the image using UFS which the application can then run[29]. Each container can be connected to more than one network.

3.3.4 Docker Networking.

Docker offers to Docker containers the ability to either communicate with each other or communicate with the external world. Therefore, depending on the usage of each container, Docker supports different types of networks. For instance, a container running a single application will have a different network configuration compared to an application that is connected to load balancers and applications that also interact with other containers and databases. Docker has as default five network drivers: none, host, bridge, macvlan and overlay [30].

1. None: Used for containers that want to create their own custom network or for those that don't want a network. None drives do not configure the container's network namespace but simply put the container inside it. A container with none driver is not able to communicate with other containers.

2. Host: With the host driver, a Docker container loses its network isolation. The Host shares its network namespace and this leads to the exposure of its public network if it is not firewalled. A container with host driver has multiple server ports delivered to itself which is why port allocation is necessary. It is a configuration based on communication through sockets.

3. Bridge: Containers with bridge drivers are connected through a virtual Ethernet bridge. This virtual bridge is created by the Docker daemon and is called docker0. Through docker0, all networks attached to it, automatically share packets between them. Each container has a

valid IP address and a private subnet. Port mapping must be configured for those containers that are attached to the bridge network. In this way, they can communicate with each other or be accessible from the external world.

4. Macvlan: The Macvlan driver removes the bridge between host and container when the overlay and bridge networks are used. In this way, container resources are exposed to external networks without the corresponding port forwarding. To achieve this, MAC addresses are used instead of IP addresses.

5. Overlay: As the name implies, an overlay network overlays an existing network and creates custom virtual connections between nodes. Containers store the mapping between their host IP and their private IP addresses in a key-value store. All hosts can access this storage. An additional layer of overlap is added to the host's network stack. When a container sents a packet the overlay layer looks up the destination host IP address in the KV store using the private IP address of the destination container in the original packet. It then creates a new packet with the destination host IP address and uses the original packet sent by the container as the new packet's payload [31].

Docker containers can by default reach the outside world, but the outside world cannot talk to the container [32]. To accomplish this, each container must expose its endpoints. Through Docker, each container can expose some or all of its ports.

3.3.5 Docker Storage.

By default, a writable container layer stores all files created inside a container. Thus, data cannot easily be removed because both the host machine and the writable layer are tightly coupled. Moreover, if the container is deleted or stopped, its data do not persist. In order to keep the files even after deleting the container and storing them on the host machine, docker provides two solutions/options: the volumes and the bind mounts. Depending on the operating system, docker supports containers that store files in memory on the host machine. If docker is operating on windows, then named pipes are used and if it is operating on Linux, tmpfs mounts are used.

- Volume: Volumes are the most preferred mechanism to store and share data among containers. They are stored in the host filesystem and only Docker processes can modify them. It is Docker Host's responsibility to manage the Docker volumes and not the containers that use them. Furthermore, a container's use of a volume does not increase its size, since Docker volumes do not exist within the container's lifecycle.

- Bind mounts: In contrast to volumes, they can be modified by both Docker and non-Docker processes. They can be either directories or filesystems and they can be stored anywhere inside the host.

- Tmpfs mounts: They cannot be written to host filesystem but they can only be stored in the host system's memory.

- Named pipes: They can be utilized for communication between a container and the Docker Host. Through named pipes a third-party tool can be installed and run inside a container and be connected to the Docker Engine API.

If a user wants to store ephemeral data in the writeable layer of a container, then it is necessary to use storage drivers. These drivers provide space efficiency and they use a pluggable architecture. Their main responsibility is to monitor the way in which containers and images are managed and stored on the Docker Host. On Linux, the storage drivers which are provided by the Docker Engine are: overlay, overlay2, vfs, fuse-overlays, aufs, btrfs and zfs, and devicemapper. Overlay, overlay2 and aufs are file-level storage drivers, while btrfs,zfs and devicemapper are block-level storage driver.

3.3.6 Container orchestration and Docker Swarm

Containers guarantee that the applications they encapsulate can be executed in any environment. This makes it possible to move and scale these applications easily and quickly. Therefore, tools are needed that are capable of automatically replacing failed containers, automatically maintaining applications, and automatically performing reconfigurations and updates to containers during their lifecycle. In addition, they are responsible for load balancing, reallocation of resources, provisioning, availability and redundancy of the containers. These tools are called orchestrators. The most common and diffused in the market orchestrators are Docker Swarm and Kubernetes. Kubernetes will be furtherly analyzed in the following subchapter.

Container orchestration allows to define automated provisioning and change management workflows to operate so as to always grant agreed policies and service levels [33]. The following figure illustrates the Container Orchestration Layers.

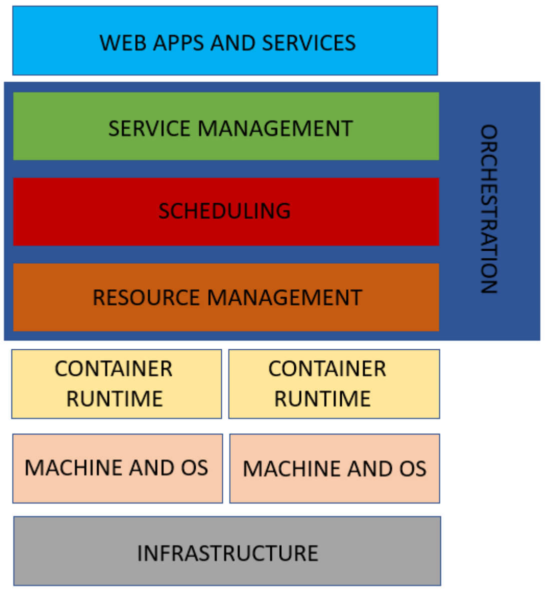


Figure 3.6: Layers of Container Orchestration [33].

The orchestration engine layer consists of three layers: service management, scheduling and resource management [33]. It is placed below the Web applications and services and on the top of each container runtime, machine and O.S. substrate.

- Resource management layer: Its main responsibility is to manage low level resources such as: Disk space, persistent volumes, memory, CPU/GPU, containers' IPs and ports inside a virtual network. Its main goals are to minimize the interference between containers competing for resources and to maximize utilization.

- Service management layer: Its main responsibility is to manage high-level aspects in to deploy complex applications. These aspects are: load balance to separate the incoming load, namespaces and groups in order to isolate containers, dependencies between microservices, labels in order to attach metadata to containers objects and readiness checking in order to allow an application to be accessible to the internet if it is capable and ready to accept the incoming traffic.

- Scheduling layer: Its main goal is to efficiently utilize cluster resources. Depending on the indications it receives from the users, it determines how to place the containers. Its main responsibilities include: placement to monitor the scheduling decisions, resurrection for the processes that need to always be up and running, co-location to assert deployment constraints, rescheduling in order to restart and reschedule failed containers and rolling deployment to automatically up/down-grade an application's version.

Docker Swarm is a cluster management and orchestration tool that connects and controls several Docker nodes to form a single virtual system [34]. Docker Swarm is deployed inside the Docker Engine. It utilizes two kinds of nodes: Manager and Worker node. The manager node is responsible for cluster management tasks and instructs the worker. The worker node is receiving instructions from the manager and executes them. The communication between them is achieved through API over HTTP. Each worker node has an agent who is responsible to report the worker's node status to the manager. In this way, the manager is always aware of the worker node status in any cluster. According to [35], some of the most fundamental features of the docker cluster are: scaling, load balancing, multi-host networking, declarative service model, rolling updates, service discovery, decentralized design, secure by default.

3.4 Kubernetes

Kubernetes is an open-source system for automating deployment, scaling and management of containerized applications [36]. The name is based on the Greek word *κυβερνήτης*, which is a high lord with the capacity to rule. Instead of the full word, the abbreviation K8s is often used, where the number 8 represents the 8 letters between the K and the s. Google's Borg system was the springboard for the creation of Kubernetes, which was donated by Google in July 2015 to the Cloud Native Computing Foundation (CNCF). e. As of early 2018 Kubernetes is considered the de facto industry standard for container orchestration, akin to the Linux kernel for the case of a single machine[37]. It is an orchestration tool whose versions are updated every four months. The current version is 1.23 and it was released in April 2022. Kubernetes is written in Go programming language.

Figure 3.7: Kubernetes official logo [36].

3.4.1 Reasons to use Kubernetes.

1) Portability: It is an orchestration tool that can be deployed in multiple environments such as bare metal, cloud setups, and local or remote virtual machines.

2) Extensibility: Multiple third-party open-source tools can support or be supported by Kubernetes. This is because of the pluggable and modular architecture of Kubernetes. This results in an increase in the capabilities of K8s.

3) Automated updates and self-healing: K8s continuously monitor the health of an application and automatically makes decisions about upgrades or configuration changes to avoid downtime. In case there is a failed container inside a node, it automatically overwrites and reconfigures them. In addition, it manages traffic and avoids routing it to irrelevant containers.

4) Storage orchestration: Kubernetes utilizes software-defined storage solutions and automatically place them to containers from external cloud providers, network storage systems, local storage or distributed storage.

5) Load balancing and Service discovery: Kubernetes assigns each container an IP, and in order to load-balance the container requests, it creates sets of containers and assigns them a Domain Name System (DNS).

3.4.2 Kubernetes architecture

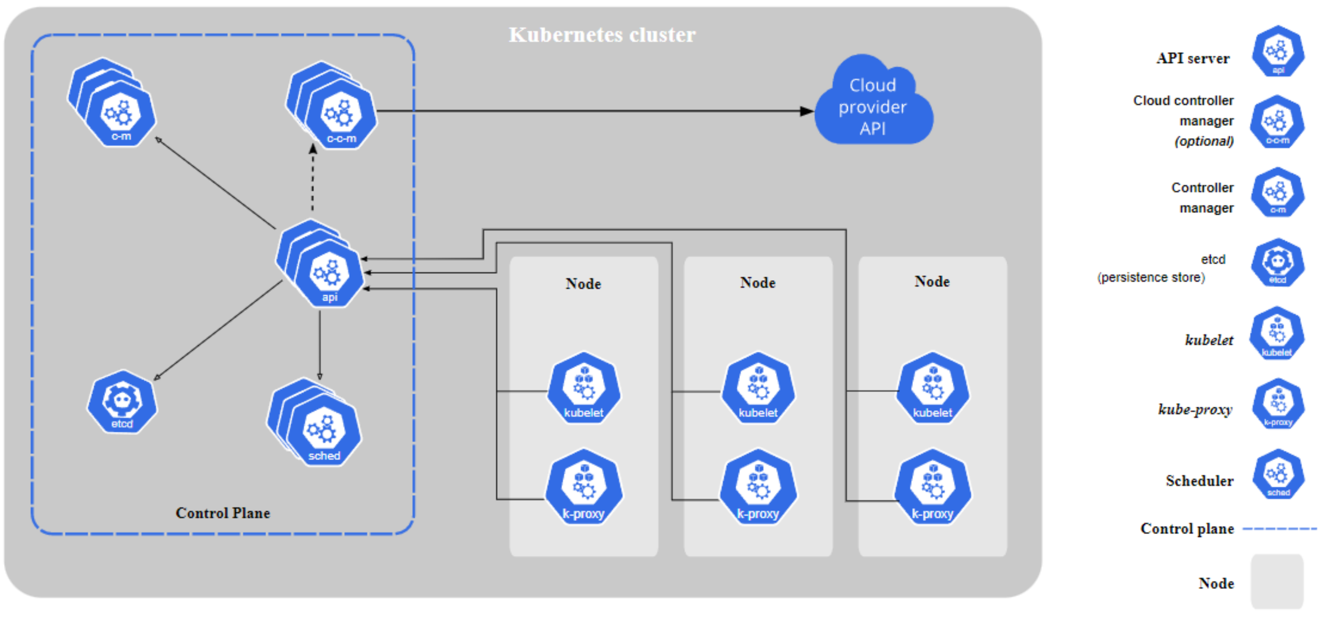


Figure 3.8: Kubernetes architecture [38].

Based on the previous Figure, we observe that the Kubernetes architecture is based on two distinct roles, the control plane or master and the worker node. The number of master and worker nodes that can be used depend on the final architecture that each user wishes to create. The components that compose the master node are: API-server, Controller Manager, Scheduler, key-value data store (etcd), while the components that compose the worker node are: Kubelet, Kube-proxy and Container Runtime. All these components coexist together and create the Kubernetes cluster.

3.4.3 Master Node overview and its components.

This node is responsible for all operations inside the cluster and it is managing the Kubernetes cluster's state. It is crucial to always keep the master node up and running. Master's failure implies downtime, which in turn creates service disruptions with the clients. The best way to prevent this scenario is to add master node replicas, which are configured in High-Availability (HA) mode. These additional master nodes stay in sync and only one of all the master nodes manages the cluster. Via this configuration, the cluster's control plane acquires resiliency. Furthermore, through an Application Programming Interface (API) or a Web-User Interface (Web UI) Dashboard or a Command Line Interface (CLI) tool, user can send requests to the control plane. Through these options the communication between the users and the K8s cluster is achieved. The most common CLI tool used, is kubectl. Moreover, all the cluster configurated data, which are important to persist the Kubernetes cluster's state, are saved in a key-value store. This store does not hold clients' data and it can be either configured on its dedicated host or on the master node.

As mentioned above, each master node contains four components which are explained separately below:

1) API-server: API-server coordinates all the administrative tasks. It receives requests from users, it evaluates the cluster's state and then process them. After a successful execution of a user's RESTful call the resulting cluster's state is stored to the key-value store. Only the API-server is allowed to communicate with it, either to read or store the cluster's state data. API-server can be characterized as the front end of a Kubernetes master node. It can provide horizontal scale and it is also highly customizable and configurable.

2) Scheduler: The way in which new workload is assigned to the worker nodes is defined by the scheduler. These assignments are heavily depended on the cluster state. For this reason, scheduler must always communicate with the API-server, in order to receive the appropriate information about worker's resource usage data. API-server sends to the scheduler the new workload requirements, then the scheduler activates its filters to choose the work node which fulfills these requirements. After the decision is made, scheduler informs the API-server. Similar to API-server, scheduler is also highly customizable and configurable.

3) Controller Managers: Their main responsibilities are to evaluate and adjust the cluster's state. They compare the current state of the cluster, using data via the API-server, with the desired state. If a mismatch occurs, operations are activated to reset the cluster's state to the desired level. They are always active and are classified into 2 types: Kube-controller-manager and cloud- controller-manager.

4) Etcd: Etcd is a distributed key-value data store. It is an essential component that stores the cluster's state data. It is an open-source project written in Go programming language and in Kubernetes also is used to store Config Maps, subnets, etc. Etcd does not allow to replace data, but it periodically compacts obsolete data to increase data store's size. Etcd has two different topologies: stacked and external. In external topology, data store is separated from the control plane nodes, while in stacked topology data store operated alongside them.

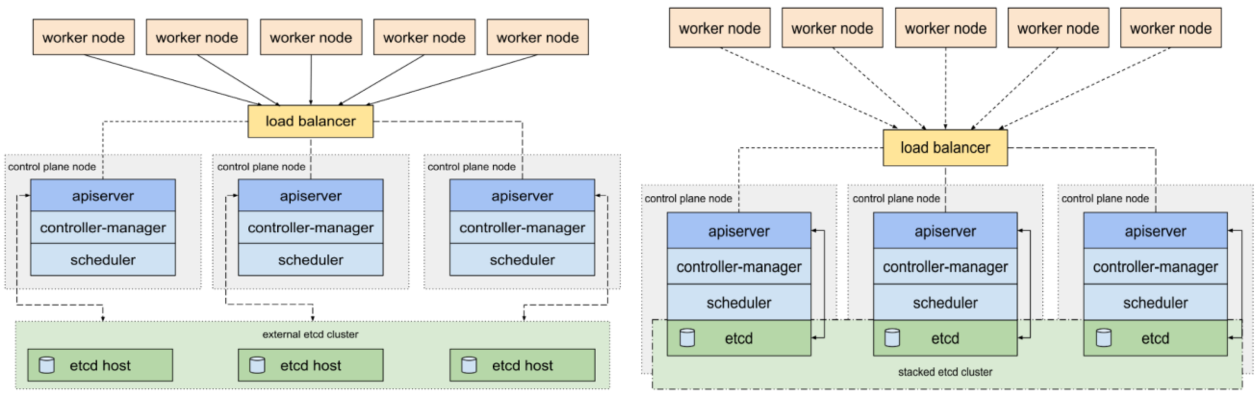


Figure 3.9: External vs Stacked etcd topology [39].

3.4.4 Worker node overview and its components.

Worker node is the node where client applications are operating. One of the primary components in this node is called Pod. Pod encapsulates client applications and it utilizes worker's memory, storage and compute resources to operate and it also utilizes network to communicate with other pods or with the external world. In Kubernetes, it is the smallest scheduling work unit. Pods can be replicated, and it is mainly for redundancy and scalability [40]. Via Pods, the logical connection between one or more containers is achievable. All the containers inside a pod share the same IP address. Moreover, it is the worker's node responsibility to manage the network traffic between applications and client users. In contrast to master processes, worker processes require more resources.

As mentioned above, each worker node contains three components which are explained separately below:

1) Container Runtime: The management and execution of containers using Kubernetes are accomplished through software called Container Runtime. Each Pod ready to run its container also needs a Container Runtime. The Container Runtimes which are supported by Kubernetes are: containerd, CRI-O, Mirantis Container Runtime and Docker.

2) Kubelet: Kubelet is the component which receives Pod definitions and communicates with the Container Runtime. This component operates on every node. It also monitors the resources and the health of the Pod, where it is placed. The connection between Kubelet and Container Runtime is achieved through the Container Runtime Interface (CRI). CRI is a plugin-based interface that executes two types of services. The first one is called RuntimeService and it concerns container-related operations. The second one is called RuntimeService and it concerns image-related operations. Containers not created by Kubernetes are not managed by Kubelet.

3) Kube-proxy: Similar to Kubelet, this component operates on every node. Its main responsibility is to manage network communication between nodes. All of the networking rules are dynamically maintained and updated by Kube-proxy. It forwards the connection requests to the Pods' containers and it abstracts the networking details of the Pods.

3.4.5 Kubernetes Networking

Networking is a complex and challenging part of Kubernetes. Kubernetes offers its users the ability to share machines between applications. In order to achieve this, it is essential for two applications not to utilize the same ports. Since Kubernetes is used for scaling and many developers or administrators take part, coordinating ports has become a difficult task. According to [41], before implementing a K8s cluster, the administrator needs to address the following four network challenges:

1) Container-to-Container communication in Pods: When a container starts, an isolated network space is created by the container runtime. Each container acquires a network space which in Linux is called a network namespace. For each Pod, which encapsulates a group of containers, the Container Runtime creates a special container, which is called the Paused container. Its goal is to create a network namespace for the Pod. That namespace will be shared by the containers of the Pod. In this way, containers can communicate and interact through localhost.

2) Pod-to-Pod communication either across cluster nodes or on the same node: In Kubernetes, the pods that compose a cluster communicate without the implementation of Network Address Translation. In order to ensure the pod-to-pod communication, the IP-per-pod model is utilized. Through this model, each pod obtains a unique IP address. The containers inside the pods use the Container Network Interface (CNI) as well as the CNI plugins. Container Network Interface (CNI) is a set of libraries and specifications that allow plugins to configure the containers' network.

3) Pod-to-Service communication: The communication between pods and the three types of services offered by Kubernetes will be further discussed in the next subchapter.

4) External-to-Service communication: In order for applications to become accessible to clients from the outside world, Kubernetes utilizes Services. Services contain network routing rules. They are created by Kube-proxy agents and stored in iptables. By exposing the services, these applications become accessible either through a dedicated port or through a virtual IP address.

3.4.6 Replica Sets and Deployments.

Replica set is an object responsible for maintaining the required number of active pods within a cluster at any time. It allows us to have multiple instances of the same Pod, so if a Pod fails, the application which is encapsulated within, can be accessible from another Pod. Replica set can also be utilized for a single Pod. In this case, if that single existing Pod fails, the Replica Set is responsible to automatically bring up a new Pod. In addition to creating or restoring incorrect Pods, Replica Sets can also delete healthy existing Pods if their number exceeds. The pods to be acquired by each replica set are defined by the selector field inside its YAML file.

Deployment is an object which is mainly used to declarative update pods and replica sets. In contrast to Replica sets, rolling updates can be accomplished to without any service downtime, because a Deployment can handle the traffic even if the update is not yet finished. In conclusion, multiple Pods can be deployed and managed by Replica sets and these Replica Sets are created and monitored by Deployments. For this reason, Deployment comes higher in the hierarchy of Kubernetes objects.

3.4.7 Kubernetes services.

In Kubernetes, each Pod has a single IP address. Since Pods are ephemeral, static IP addresses cannot be utilized. If a Pod is terminated, a new Pod with a new IP address will be created. In order for Pods or users who connect directly to them via their IP addresses, to know and be constantly informed about which IP addresses to use, Kubernetes provides Services. In Kubernetes, a Service is an abstraction that defines a logical set of Pods and a policy by which to access them[42]. For runtime service discovery, Kubernetes utilizes two methods: DNS and environmental variables, while Kubernetes provides four types of services depending on the way we want the service to be accessible. These are:

- ClusterIP: It is the default Service Type. Cluster IP is a virtual IP address received by a Service and it is reachable only from the inside of a cluster.
- NodePort: Via this type the service is reachable from the outside world, exposing a static port to each Node's IP. The port range is 30000-32767.
- LoadBalancer: Via this type the service is reachable from the outside world, using a cloud provider’s load balancer, while the ClusterIp and the NodePort will be automatically created by Kubernetes.
- ExternalName: Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up[42].

In order for a service to be exposed, Kubernetes also supports Ingress. This acts as an entry point to the cluster and can expose several services under one IP. However, Ingress is not a type of service.

3.4.8 Kubernetes storage

The data and information stored inside a container are deleted if the container is deleted or crashes. In Kubernetes, the kubelet restarts the container but with a clean state[43]. To avoid this problem, and to allow containers inside a Pod to share files with each other, Kubernetes uses storage abstractions called Volumes. These are intertwined with the lifecycle of the pod they belong to. By extension, a volume is deleted if its pod is deleted, not if the containers that compose it expire. Kubernetes provides several volume types. These are: hostPath, awsElasticBlockStore, emptyDir, gcePersistentDisk, azureFile, azureDisk, secret, nfs, configMap, iscsi, cephfs and persistentVolumeClaim.

In contrast to typical IT environment whereas storage administrators are responsible for storage management, Kubernetes provides a subsystem called PersistentVolume. This subsystem allows either users or administrators to consume or manage persistent storage through two APIs, which are listed below:

- PersistentVolume(PV): It is a storage abstraction inside a cluster that can be either dynamically provisioned via storage classes or statically by an administrator. The lifecycle of a PV does not depend on the lifecycle of the pods using it.

- PersistentVolumeClaim(PVC): A PersistentVolumeClaim is a request for storage by a user [43] and is divided into three types: ReadWriteMany(read-write by many nodes), ReadOnlyMany(read-only by many nodes), ReadWriteOnce(read-write by a single node).

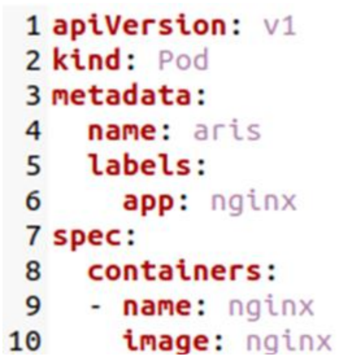
3.4.9 YAML configuration files in Kubernetes.

To create objects such as services, deployments, pods, etc, Kubernetes utilizes YAML files. Each YAML file is required to have four fields. These, in order, are API version, kind, metadata and spec. Depending on the type of object, the correct API version is required. The following table depicts the API version of the most known and used objects in Kubernetes:

Object	API version
Pod	v1
Service	v1
Replica set	Apps/v1
Deployment	Apps/v1

Table 3.2: Api versions.

The kind of object, that want to create, is inserted in the kind field. Metadata consists of data about the object. In this field, we add information about the object, in order to be uniquely identified. The last field is called spec and contains data about the state of the object. To better understand the structure and hierarchy of a YAML file, the following figure depicts a simple YAML file for creating a pod.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: aris
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

Figure 3.10: Simple YAML file of a Pod.

Api version and kind are being specified with a string value. However, metadata is in a form of a dictionary. Everything under metadata is moved to the right. In this case, name and labels are children of metadata. The space before name and labels should be the same, as they are siblings, and bigger than the space before metadata, since it is their parent. Name is a string value but labels are also a dictionary, inside the metadata dictionary. Labels are used to identify objects and can have any key value pair. In this case, the key is app and the value is nginx.

So far, we defined the type and the name of object. To define which container will be encapsulated by the pod, we add a property called containers under the spec field. Spec is also a dictionary. However, containers are a list inside a YAML file, since pods are able to contain multiple containers. The dash in the property name under the container indicates that it is the first item of the list. In this case, the name of the container is nginx and the value of the image is also nginx. The value of the image key inside a container list indicates the name of the Docker image inside the Docker repository.

4 Virtual lab use case.

This chapter includes a brief mention of virtual labs and the presentation of the architecture we followed, which includes the development of a Kubernetes infrastructure, the development and connection of a web solution to control this infrastructure and the creation of API calls.

4.1 Virtual labs

Virtual labs use the power of computerized models and simulations and a variety of other instructional technologies to replace face-to-face lab activities[44]. Virtual labs are a cloud solution based on the three main features of cloud computing to meet the needs required to implement a lab course. These are multitenancy, shared resource pooling and self-organizing capabilities.[45] The combination of these features makes it possible to overcome a large percentage of the problems faced by a lab. These problems are mainly related to hardware equipment. Educational institutions that are either unable to meet their computing resource needs or cannot upgrade their existing equipment with new software tools are unable to deliver a comprehensive educational experience to their students. For this reason, Cloud Computing technology has also penetrated the educational sector with the implementation of educational software tools or infrastructures.

4.2 Virtual lab solution.

The aim of the experimental part is to implement an infrastructure that can support two different virtual labs. The first will be for the computer networking lab course and the second for the dev-ops lab course. This infrastructure will be based on the docker orchestration tool, Kubernetes. Kubernetes supports a variety of tools that are able to set up and run a Kubernetes cluster. In this thesis, Minikube will be installed to run Kubernetes in our local machine, because it requires few computing resources, its installation is simple and it is ideal for learning purposes. Through Minikube, a cluster with one master node and two workers nodes will be created (one each for the two labs). The pods of each worker will encapsulate a different docker image depending on the lab, according to the following table.

Lab-name	Image
Computer Network	nginx
Dev-ops	pbitty/hello-from:latest

Table 4.1: Docker images of the virtual labs.

In this thesis, we are not concerned with what the pods will contain. The aims is to implement the infrastructure for the virtual labs and therefore the choice of docker images is random.

The number of pods managed by each worker will be finite and equal to both labs. In addition, each pod will be assigned to each user. Kubernetes does not allow containers with the same image to be encapsulated inside a single pod. In this case, the pod will fail. However, pods are able to encapsulate multiple containers with different images and wrap all these as a single unit. For instance, a single pod can contain a container with a nginx image, a container serving data storage etc. In our thesis, the "one-container-per-pod" model will be utilized, since each of the pod will have a single container.

For each virtual lab, a set of two YAML files will be created. The deployment's YAML file will create the pods and the service's YAML file will expose these pods, in order to be accessible to the participants of the virtual lab.

Following the logic of a semester's curriculum, the laboratory courses will be activated and terminated at a specific time and the duration will be 25 minutes. The following table depicts the curriculum of the virtual labs.

Lab-name	Computer Network	Dev-ops
Day	Thursday	Thursday
Time	12:05-12:30	12:35-13:00
Number of participants	5	5

Table 4.2: Curriculum of the virtual labs.

After the creation of the Kubernetes cluster, the second step is to create a web solution. Its goal is to automatically activate and terminate the cluster and the virtual labs at specific time intervals every Thursday, according to the schedule below.

Time	Event
12:00	Launch of the cluster.
12:05	Launch of the computer networking lab.
12:30	Termination of the computing network lab.
12:35	Launch of the devops lab.
13:00	Termination of the devops lab.
13:05	Termination of the cluster.

Table 4.3: Timeline of events.

The implementation of this web solution will be accomplished through NodeJs and all these automatic actions will be executed through cronjobs. In addition, the final result of the experimental part should enable the infrastructure's administrator to monitor the status of the up and running pods at any time. Moreover, if the number of participants needs to be increased for a particular course, the administrator will instruct the up and running worker node to increase the number of available pods by a certain number. For this reason, the third step involves the creation of REST Application Programming Interface calls in order to achieve the communication between the client(administrator) and the web-server that will be deployed with NodeJs. In this client-server architecture, client sends a request to the server and the server sends data back to the client as response, typically in the form of JSON data, over the HTTP. The data sent by the server in response is received by the operating system via commands. In this thesis, three actions must be accomplished. To retrieve the state of the up and running pods of the cluster, the HTTP request method GET will be executed. To increase the pods of a worker node, the HTTP request method POST will be executed. GET request method is a request for information and it only retrieves data. This type of request is not able to modify the underlying data that interacts. This ability is possessed by the POST request method. Via POST we can update or create a new resource.

For these request methods, we implemented three endpoints. The following table depicts these endpoints for each method.

Method	Purpose	URL
GET	State of the cluster pods.	http://localhost:8000/count
POST	Add pods to the computer network lab.	http://localhost:8000/add-network
POST	Add pods to the dev-ops lab.	http://localhost:8000/add-devops

Table 4.4: URL for each request method.

After the end of the lab courses and the termination of the Pods, the progress of each user will not be saved. Therefore, each user will have to take care of how to save their actions and results as he/she will not be able to return to where he/she left off the previous time. In addition, this infrastructure. once implemented, it will not be able to dynamically change the activation or termination time of the virtual labs. To accomplish this, the application code will have to be modified. Therefore, after the implementation, the schedule of Table 4.2 should be strictly followed. The following Figure depicts the overall architecture of our infrastructure as described above:

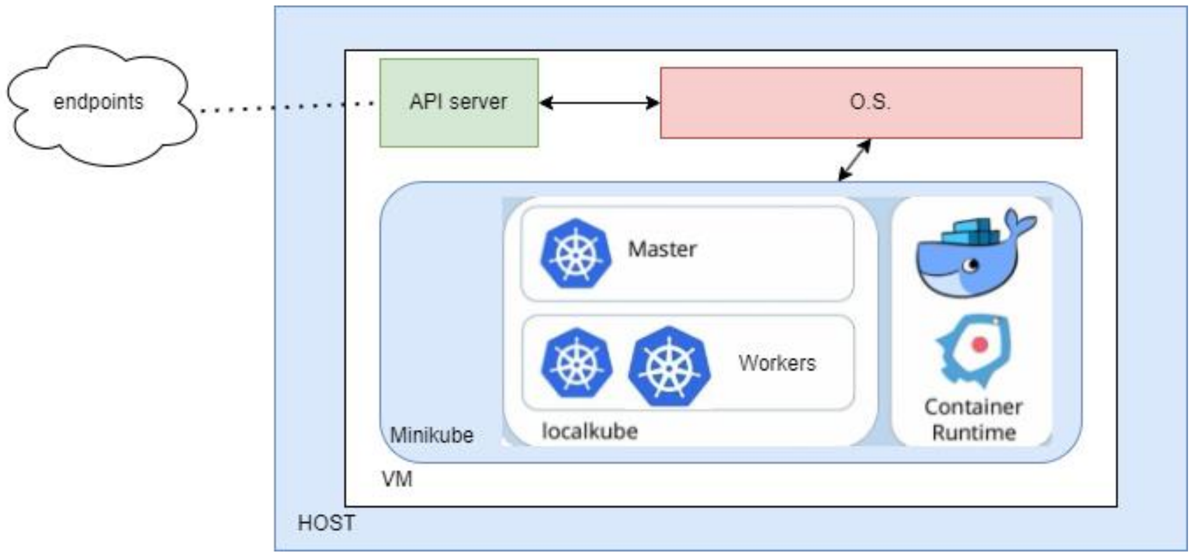


Figure 4.1: Architecture of our thesis.

5 Implementation of virtual labs.

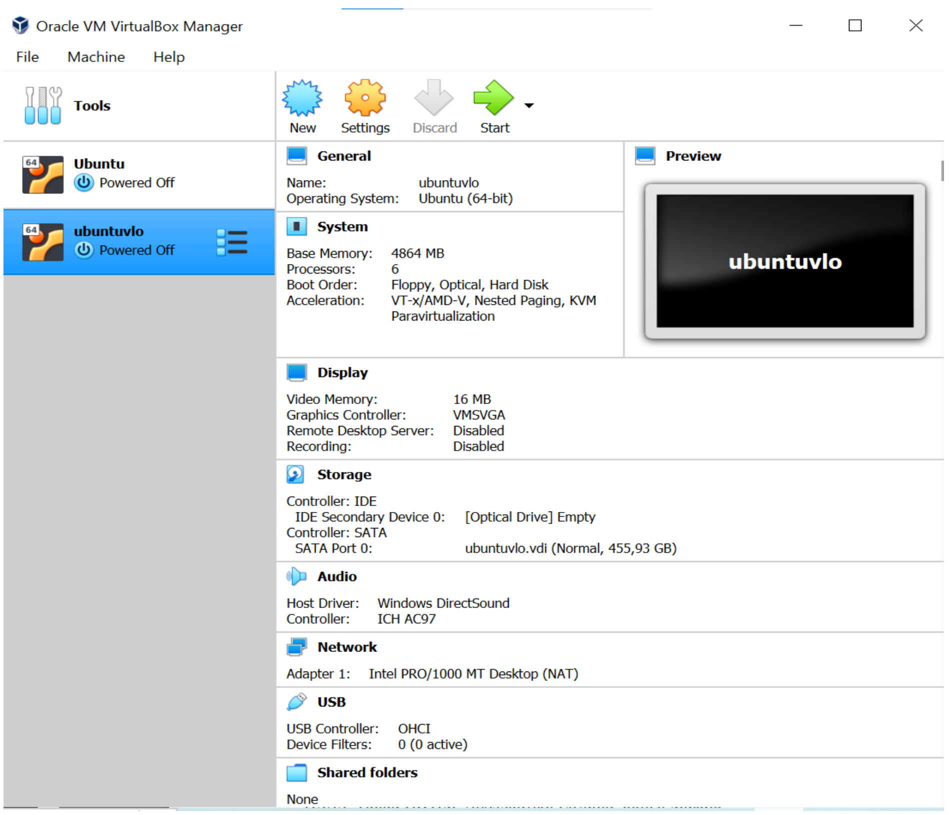
In this chapter we will review in more detail the steps we followed to create the infrastructure to support the virtual labs. The YAML files and code will be presented whereas the tools and platforms to be used are the following:

- Windows O.S.
- Linux O.S.
- Oracle Virtual Box virtualization software.
- Minikube.
- NodeJs.

5.1 Host Machine and Oracle VM VirtualBox Manager.

The host machine is a Windows 10 x64, 8 GB RAM with i5-5500U CPU. In order to host a Linux environment in a host machine that runs Windows, we have to install a virtualization software. In our case, that software is the Oracle VM VirtualBox Manager and it will be used to create a VM with Linux O.S. The version which is installed, is the 6.1.26. The following step is to download an Ubuntu image in our local machine. The version which is downloaded is the 22.04.1.

We open the VirtualBox Manager, we create a new VM with the name ubuntuvlo. Then, we configure it to run the Ubuntu image. The following picture depicts the initial menu of our VirtualBox, which has two VMs. Our VM is selected and ready to start.



General
Name: ubuntuvlo
Operating System: Ubuntu (64-bit)

System
Base Memory: 4864 MB
Processors: 6
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: ubuntuvlo.vdi (Normal, 455,93 GB)

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

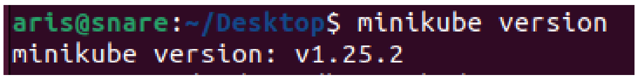
USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Figure 5.1: The Oracle VM Virtual Box Manager.

5.2 Minikube installation and cluster configuration.

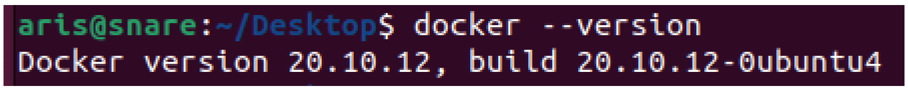
After installing and configuring the VM in the virtual box, the next step is to install Minikube. The instructions we followed for the correct installation are in the Minikube documentation at the following link: https://minikube.sigs.k8s.io/docs/start/. The following figure depicts the version which is installed.



```
aris@snare:~/Desktop$ minikube version
minikube version: v1.25.2
```

Figure 5.2: Minikube version.

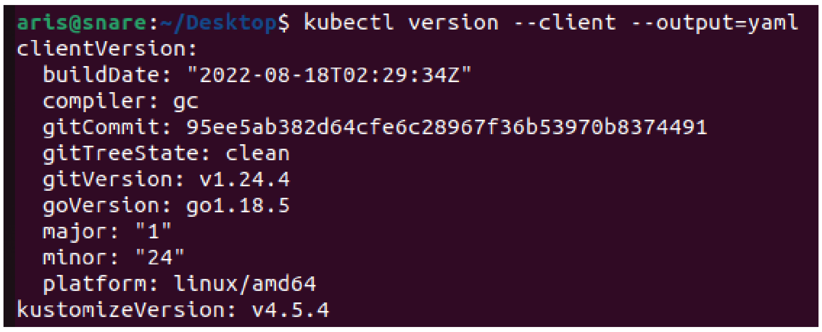
Before starting our cluster, we have to set a driver for Minikube. In our case, we chose the Docker driver. To install docker on our virtual machine and set docker as the default driver, we followed the instructions from the following link: https://minikube.sigs.k8s.io/docs/drivers/docker/. The following figure depicts the Docker version which is installed.



```
aris@snare:~/Desktop$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu4
```

Figure 5.3: Docker version.

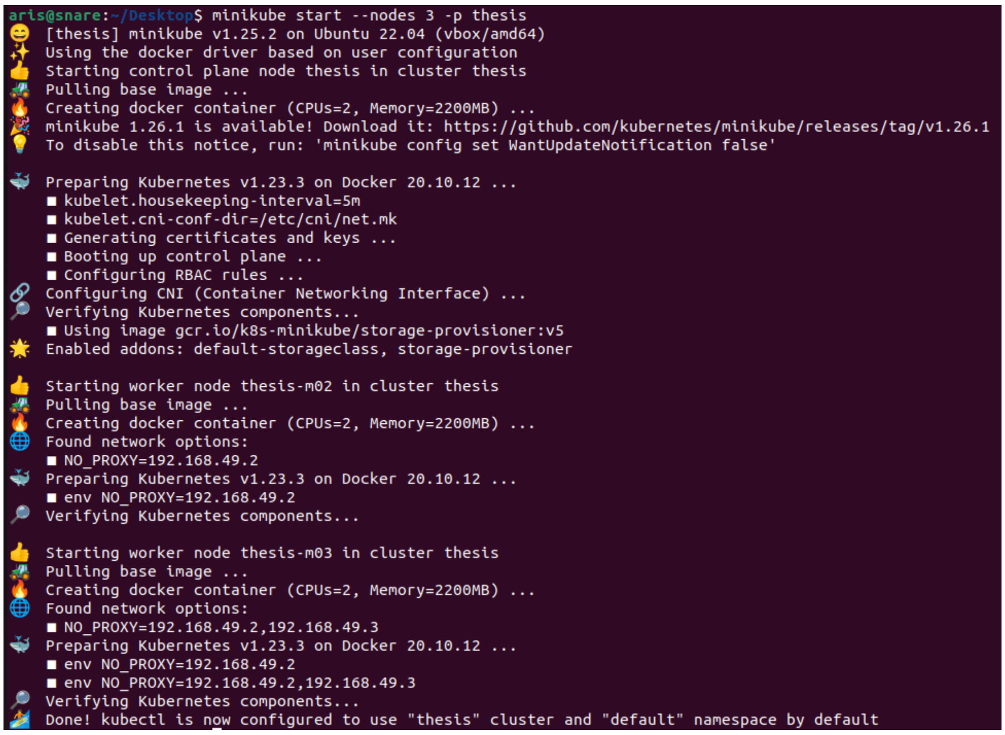
In order to interact with our cluster, we will use the CLI tool called kubectl. To download kubectl, we followed the instructions provided by the following link of Kubernetes documentation: https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/. The following figure depicts the version of kubectl which is installed.



```
aris@snare:~/Desktop$ kubectl version --client --output=yaml
clientVersion:
  buildDate: "2022-08-18T02:29:34Z"
  compiler: gc
  gitCommit: 95ee5ab382d64cfe6c28967f36b53970b8374491
  gitTreeState: clean
  gitVersion: v1.24.4
  goVersion: go1.18.5
  major: "1"
  minor: "24"
  platform: linux/amd64
kustomizeVersion: v4.5.4
```

Figure 5.4: Version of kubectl.

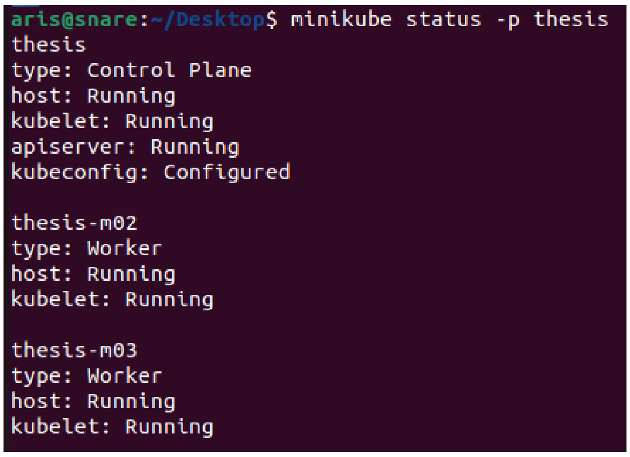
Now, we are ready to set up our cluster. Our cluster will be named thesis and will contain 3 nodes. To accomplish that, the command is: minikube start --nodes 3 -p thesis. In a few second the output is:



```
aris@snare:~/Desktop$ minikube start --nodes 3 -p thesis
😄  [thesis] minikube v1.25.2 on Ubuntu 22.04 (vbox/amd64)
✨  Using the docker driver based on user configuration
👍  Starting control plane node thesis in cluster thesis
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🎉  minikube 1.26.1 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.26.1
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

🐳  Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
    ▪ kubelet.housekeeping-interval=5m
    ▪ kubelet.cni-conf-dir=/etc/cni/net.mk
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: default-storageclass, storage-provisioner

👍  Starting worker node thesis-m02 in cluster thesis
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.49.2
🐳  Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
    ▪ env NO_PROXY=192.168.49.2
🔎  Verifying Kubernetes components...

👍  Starting worker node thesis-m03 in cluster thesis
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.49.2,192.168.49.3
🐳  Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
    ▪ env NO_PROXY=192.168.49.2
    ▪ env NO_PROXY=192.168.49.2,192.168.49.3
🔎  Verifying Kubernetes components...
🏄  Done! kubectl is now configured to use "thesis" cluster and "default" namespace by default
```

Figure 5.5: Implementation of the cluster.

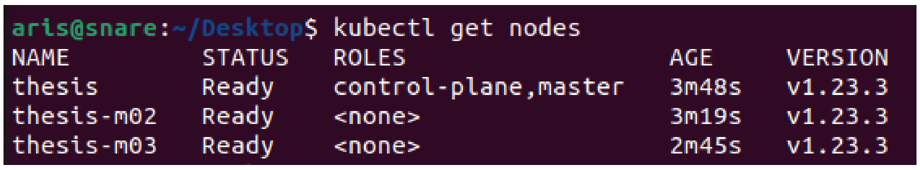
Our cluster is now started. The command to check the status of the created nodes is minikube status -p thesis. In our case, the output is depicted in the following figure.



```
aris@snare:~/Desktop$ minikube status -p thesis
thesis
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

thesis-m02
type: Worker
host: Running
kubelet: Running

thesis-m03
type: Worker
host: Running
kubelet: Running
```

Figure 5.6: Status of the nodes.

Alternatively, we can also use the kubectl get nodes command to see the name, status, type, age and version of each node that has been deployed. The following figure depicts the output of this command.

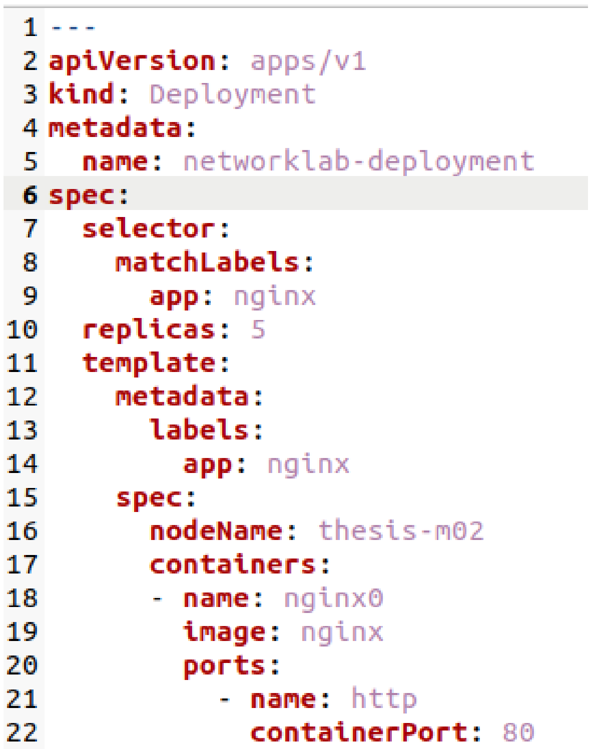


```
aris@snare:~/Desktop$ kubectl get nodes
NAME         STATUS   ROLES                  AGE     VERSION
thesis       Ready    control-plane,master   3m48s   v1.23.3
thesis-m02   Ready    <none>                 3m19s   v1.23.3
thesis-m03   Ready    <none>                 2m45s   v1.23.3
```

Figure 5.7: List of the nodes.

5.3 Deployments.

For each virtual lab we will need one deployment which will create a Replica set to bring up 5 Pods with the same encapsulated image. The following figure depicts the content of the YAML file responsible for creating the deployment for the computer networking lab.



```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: networklab-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeName: thesis-m02
      containers:
      - name: nginx0
        image: nginx
        ports:
          - name: http
            containerPort: 80
```

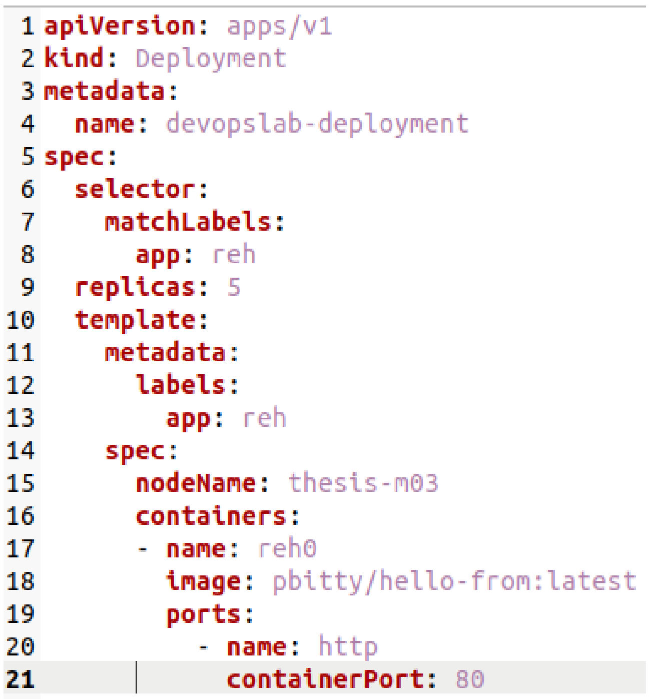
Figure 5.8: Deployment for the computer network lab.

Via this YAML file:

- A deployment is created and its name is networklab-deployment. This is indicated by the .metadata.name field.
- The .spec.replicas field indicates that 5 Pods are created.
- How the deployment will find which Pods to monitor is specified by the .spec.selector field. Here we choose to have a type label (app: nginx) that is defined by the .metadata.labels field.

- The .template.spec.nodeName field defines the specific node in which the Pods will be deployed. In this case, the node named thesis-m02 has been selected.
- Each pod contains a container named nginx0. Each container contains a nginx image and runs on port 80 named Http. These are defined by the .spec.template.spec.containers field.

For the virtual lab related to the dev-ops lab, an additional deployment will be created. The following figure depicts the content of the additional deployment YAML file.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devopslab-deployment
spec:
  selector:
    matchLabels:
      app: reh
  replicas: 5
  template:
    metadata:
      labels:
        app: reh
    spec:
      nodeName: thesis-m03
      containers:
      - name: reh0
        image: pbitty/hello-from:latest
        ports:
          - name: http
            containerPort: 80
```

Figure 5.9: Deployment for the devops lab.

The structure of this YAML file is exactly the same as the one we used in the case of the first virtual lab. The differences are mainly in the string values involved:

- The name of the deployment, which is devopslab-deployment.
- The node on which it will be implemented, which is the thesis-m03 node
- The name of the container, which is reh0.
- The type of image, which is pbitty/hello-from and its version which is the latest.
- The value of the app label, which is reh.

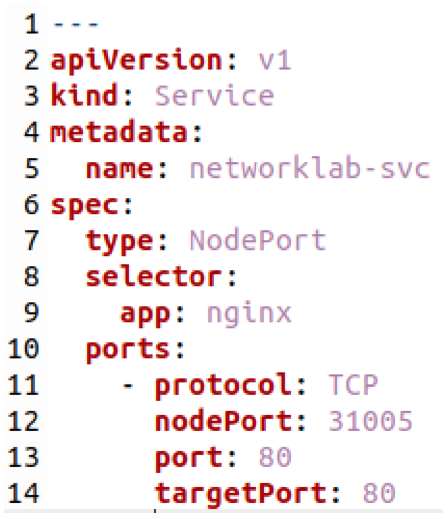
After completing and storing the YAML files in the Linux environment, the commands to be used for the deployments in this particular thesis are listed below:

- To create a deployment, the command is: kubectl apply -f «the name of the YAML file».
- To check which deployments run in our cluster, the command is: kubectl get deployments.

- To check the Replica Sets in our cluster, the command is: kubectl get replicaset.
- To check the Pods in our cluster the command is: kubectl get pods.
- To delete a deployment, the command is: kubectl delete deployment «deployment's name».
- To scale a deployment, the command is: kubectl scale --current-replicas= «number of the current replicas» --replicas= «new number of the replicas» deployment/ «deployment's name».

5.4 Services.

In order for users, in this case, students, to have access to the Pods, a Service must be created for each Deployment. Similar to the deployments, two YAML files must be created. The following figure depicts the content of the YAML file responsible for creating the service of the computer networking lab.



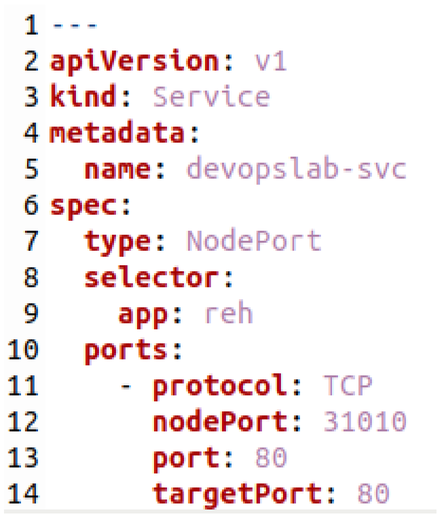
```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: networklab-svc
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      nodePort: 31005
      port: 80
      targetPort: 80
```

Figure 5.10: Service for the computer network lab.

Via this YAML file:

- A service is created and its name is networklab-svc. This is indicated by the .metadata.name field.
- The type of the service is NodePort. This is indicated by the .spec.type field.
- The .spec.ports field contains information about the ports that will be utilized. Ports is an array and since protocol is the first element in the array, it has a dash. The protocol that will be used is the TCP protocol. The node's port is set to 31005. The targetPort:80 is the pod's port and service's port is the port:80.
- The .spec.selector field is used to link the pods of the networklab deployment with this specific service. These pods have the label app:nginx, which also must be inserted in the networklab YAML file, in the .spec.selector field.

For the virtual lab related to the dev-ops lab, an additional service will be created. The following figure depicts the content of the additional service YAML file:



```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: devopslab-svc
spec:
  type: NodePort
  selector:
    app: reh
  ports:
    - protocol: TCP
      nodePort: 31010
      port: 80
      targetPort: 80
```

Figure 5.11: Service for the devops lab.

The structure of this YAML file is exactly the same as the one we used in the case of the first virtual lab. The differences are mainly in the string values involved:

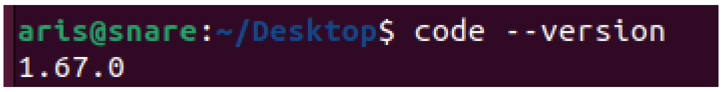
- The name of the deployment, which is devopslab-svc.
- The string value of the label, which is reh. This links the pods of the devopslab deployment with this service.
- The nodePort, which is 31010.

After completing and storing the YAML FILES in the Linux environment, the commands to be used for the services in this particular thesis are listed below:

- To create a service, the command is: kubectl apply -f «the name of the YAML file».
- To check which services run in our cluster, the command is: kubectl get services.
- To delete a service, the command is: kubectl delete service «service's name».
- To list the URL of each service, the command is: minikube service list -p «cluster's name».

5.5 Nodejs.

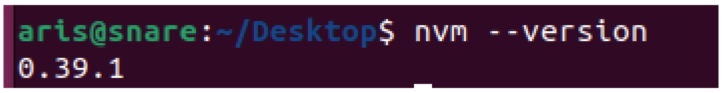
In order to deploy our web solution with NodeJs, the first step is to install an editor. In this thesis we have chosen the Visual Studio Code, which is a source-code editor. For this installation, we run the command sudo apt install code. The following figure depicts the version of VS code that is installed.



```
aris@snare:~/Desktop$ code --version
1.67.0
```

Figure 5.12: Visual Studio Code version.

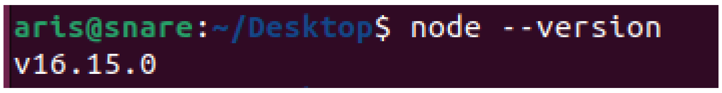
The second step, before installing the NodeJs, is to install the Node Version Manager. It is a source version manager that enables a developer to use different versions of the NodeJs without having to install each one of them. For this installation, we run the command sudo apt install nvm. The following figure depicts the version of the NVM that is installed.



```
aris@snare:~/Desktop$ nvm --version
0.39.1
```

Figure 5.13: Node Version Manager version.

To install the latest version of the NodeJs, we execute the command nvm install --lts. The following figure depicts the version that is installed.



```
aris@snare:~/Desktop$ node --version
v16.15.0
```

Figure 5.14: NodeJs version.

The fourth step is to create a directory, that will contain all the files for our web solution. The name of this directory is api. To create it, we simply run the command mkdir api and in order to change the directory to api, we run the command cd api. Now, since we are in our project’s directory, we initialize the Node Packet Manager, via the command npm init. Since it is initialized, we install the packages that will be needed to accomplish our web solution via the command npm install express node-cron.

- Express – API framework.
- Node-cron – Cronjob’s driver for Node.js.

After installing and initializing all these packages and tools, we open the VS code, we browse into our api directory and we create a file named index.js. When the program is completed, we run the node index.js command to execute it.

5.6 Cronjobs.

Cronjobs are used to automatically execute tasks or commands. Cronjobs format contains five asterisks and the command which are going to execute. Each asterisk has its own purpose in relation to the day of the week, month, day of the month, hour and minute as the following figure depicts.

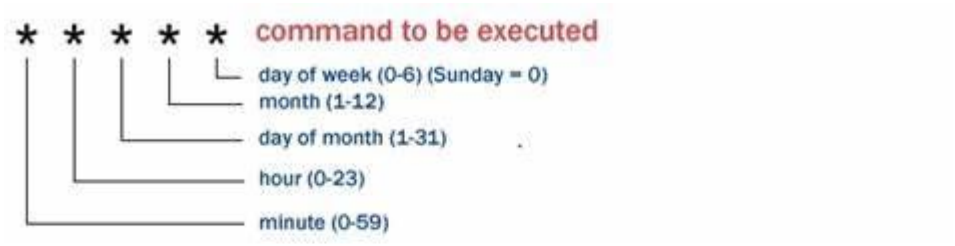

* * * * * command to be executed
day of week (0-6) (Sunday = 0)
month (1-12)
day of month (1-31)
hour (0-23)
minute (0-59)

Figure 5.15: Cronjob format [46].

Table 4.3 depicts the timeline of the events that we want to be executed automatically. The following table is a new version of the table 4.3 which also contains the cronjob formats depending on the time we want them to be executed.

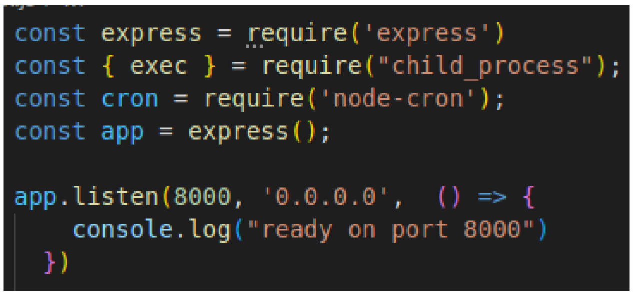
Time	Event	Cronjob format
12:00	Launch of the cluster.	00 12 * * 4
12:05	Launch of the computer networking lab.	05 12 * * 4
12:30	Termination of the computing network lab.	30 12 * * 4
12:35	Launch of the devops lab.	35 12 * * 4
13:00	Termination of the devops lab.	00 13 * * 4
13:05	Termination of the cluster.	05 13 * * 4

Table 5.1: Cronjob formats of the thesis.

5.7 Code presentation in sections.

The overall code is illustrated in Appendix A. In this subsection, we will present the code of the program in sections depending on what is being accomplished.

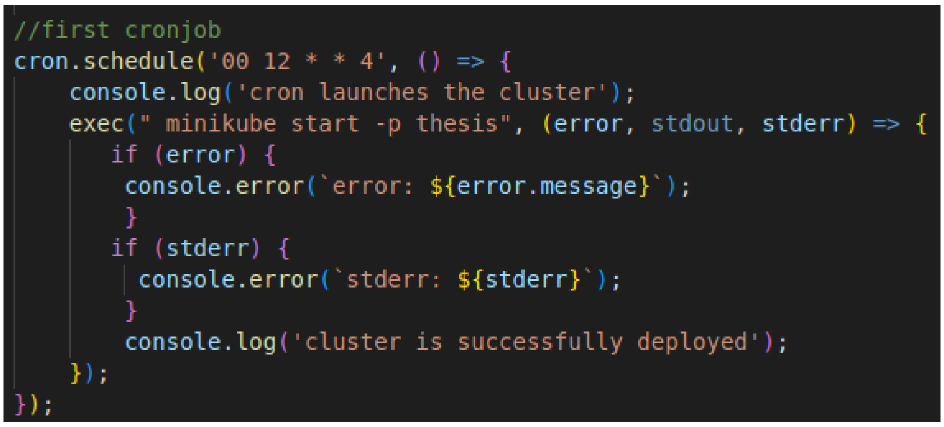
➢ First section:



```
const express = require('express')
const { exec } = require("child_process");
const cron = require('node-cron');
const app = express();

app.listen(8000, '0.0.0.0',  () => {
    console.log("ready on port 8000")
  })
```

Figure 5.16: Boilerplate code for express server to start.

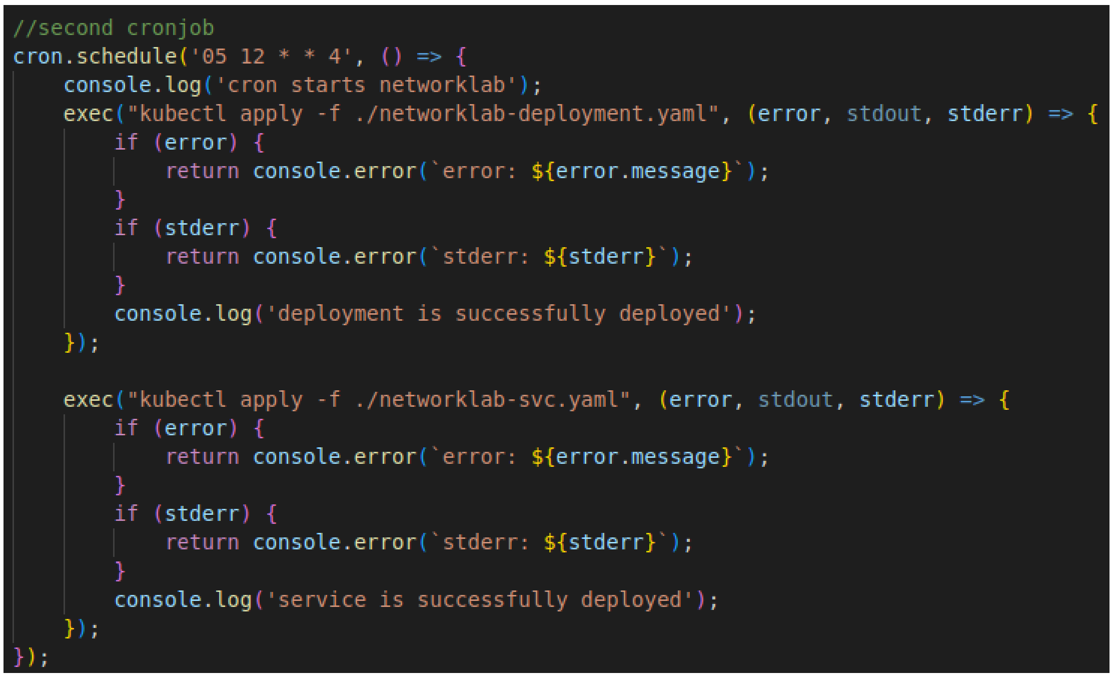
➢ Second section:



```
//first cronjob
cron.schedule('00 12 * * 4', () => {
    console.log('cron launches the cluster');
    exec(" minikube start -p thesis", (error, stdout, stderr) => {
        if (error) {
          console.error(`error: ${error.message}`);
          }
        if (stderr) {
          console.error(`stderr: ${stderr}`);
          }
          console.log('cluster is successfully deployed');
    });
});
```

Figure 5.17: Cronjob that launches the cluster.

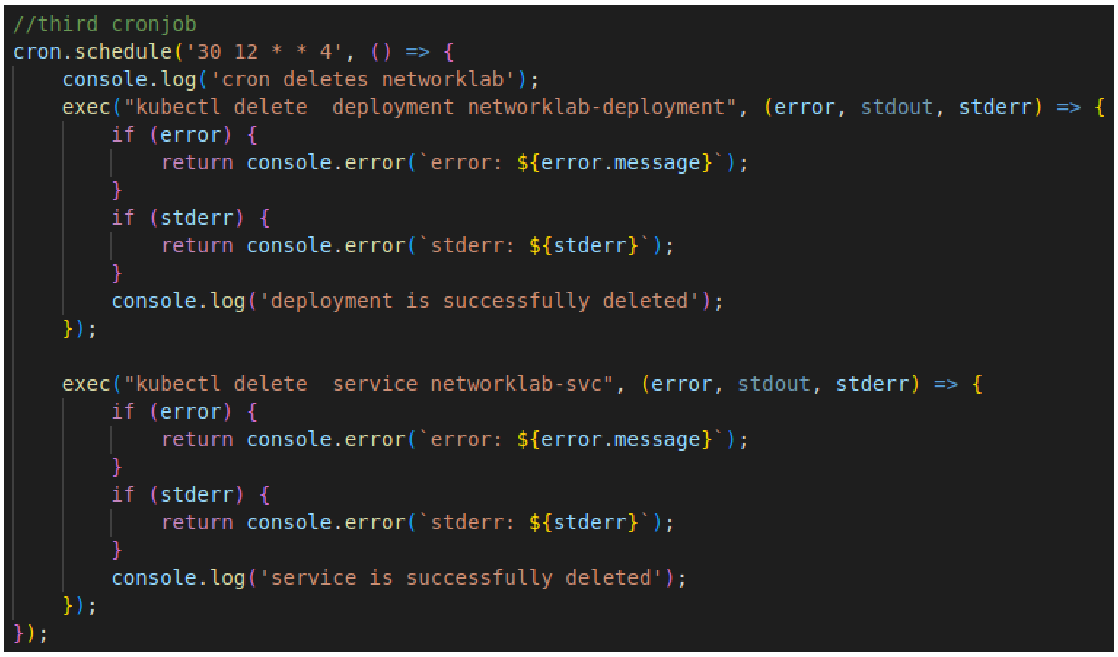
➢ Third section:



```
//second cronjob
cron.schedule('05 12 * * 4', () => {
    console.log('cron starts networklab');
    exec("kubectl apply -f ./networklab-deployment.yaml", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('deployment is successfully deployed');
    });

    exec("kubectl apply -f ./networklab-svc.yaml", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('service is successfully deployed');
    });
});
```

Figure 5.18: Cronjob that launches the computer network lab.

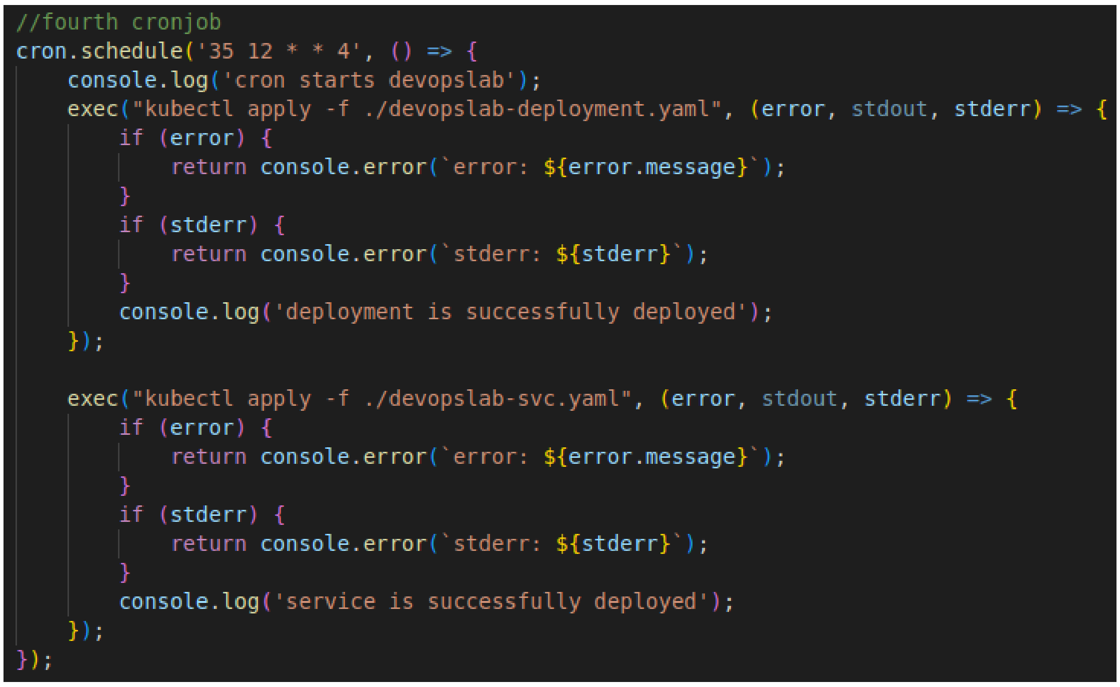
➢ Fourth section:



```
//third cronjob
cron.schedule('30 12 * * 4', () => {
    console.log('cron deletes networklab');
    exec("kubectl delete  deployment networklab-deployment", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('deployment is successfully deleted');
    });

    exec("kubectl delete  service networklab-svc", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('service is successfully deleted');
    });
});
```

Figure 5.19: Cronjob that terminates the computer network lab.

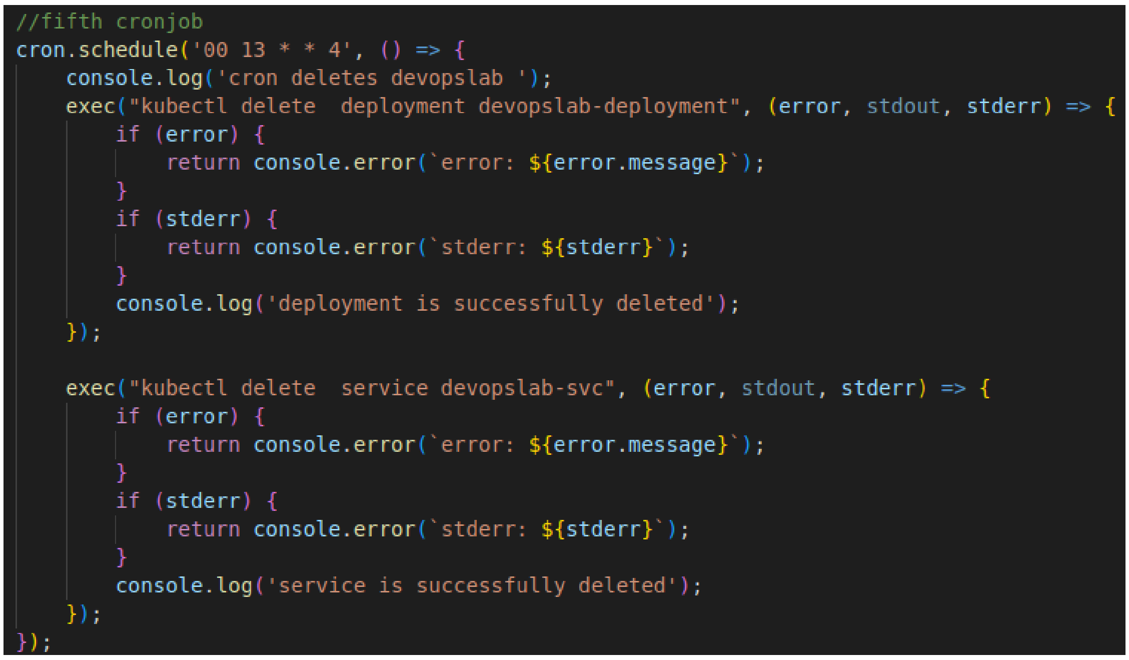
➢ Fifth section:



```
//fourth cronjob
cron.schedule('35 12 * * 4', () => {
    console.log('cron starts devopslab');
    exec("kubectl apply -f ./devopslab-deployment.yaml", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('deployment is successfully deployed');
    });

    exec("kubectl apply -f ./devopslab-svc.yaml", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('service is successfully deployed');
    });
});
```

Figure 5.20: Cronjob that launches the devops lab.

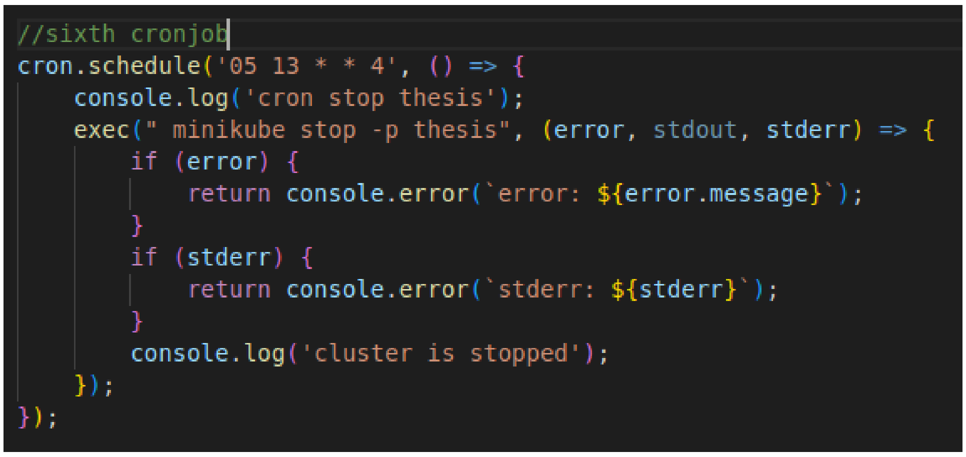
➢ Sixth section:



```
//fifth cronjob
cron.schedule('00 13 * * 4', () => {
    console.log('cron deletes devopslab ');
    exec("kubectl delete  deployment devopslab-deployment", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('deployment is successfully deleted');
    });

    exec("kubectl delete  service devopslab-svc", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('service is successfully deleted');
    });
});
```

Figure 5.21: Cronjob that deletes the devops lab.

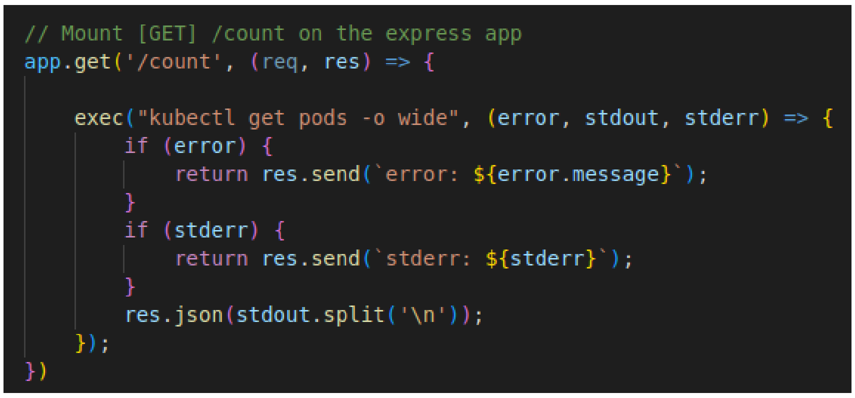
➢ Seventh section:



```
//sixth cronjob
cron.schedule('05 13 * * 4', () => {
    console.log('cron stop thesis');
    exec(" minikube stop -p thesis", (error, stdout, stderr) => {
        if (error) {
            return console.error(`error: ${error.message}`);
        }
        if (stderr) {
            return console.error(`stderr: ${stderr}`);
        }
        console.log('cluster is stopped');
    });
});
```

Figure 5.22: Cronjob that stops the cluster.

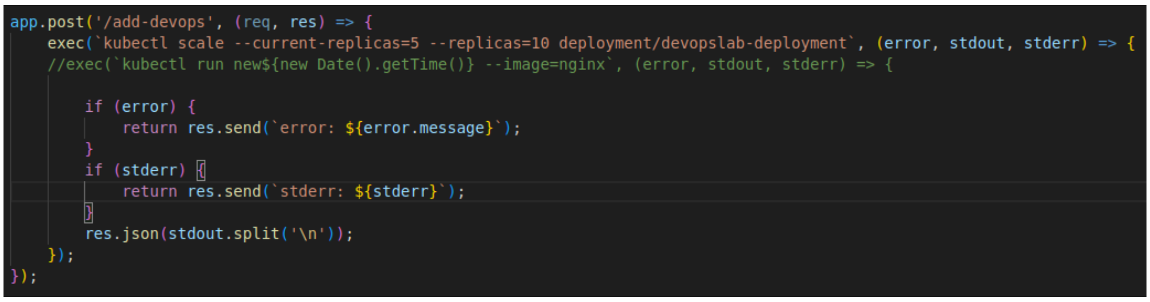
- Eighth section:



```javascript
// Mount [GET] /count on the express app
app.get('/count', (req, res) => {

    exec("kubectl get pods -o wide", (error, stdout, stderr) => {
        if (error) {
            return res.send(`error: ${error.message}`);
        }
        if (stderr) {
            return res.send(`stderr: ${stderr}`);
        }
        res.json(stdout.split('\n'));
    });
})
```

Figure 5.23: Implementation of the GET method.

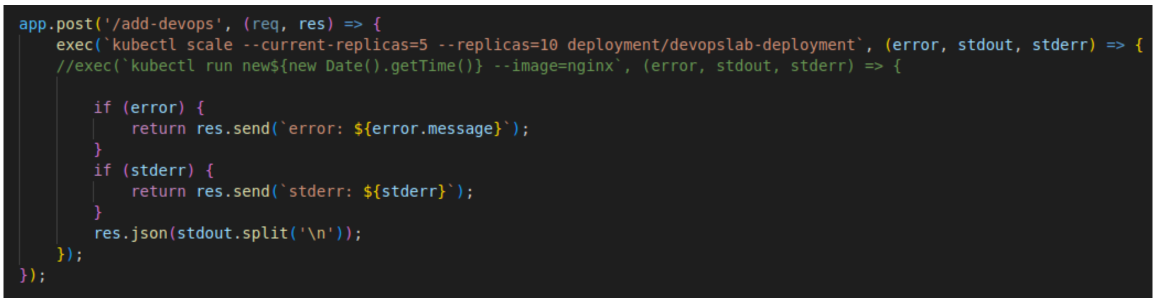
- Ninth section:



```javascript
app.post('/add-devops', (req, res) => {
    exec(`kubectl scale --current-replicas=5 --replicas=10 deployment/devopslab-deployment`, (error, stdout, stderr) => {
    //exec(`kubectl run new${new Date().getTime()} --image=nginx`, (error, stdout, stderr) => {

        if (error) {
            return res.send(`error: ${error.message}`);
        }
        if (stderr) {
            return res.send(`stderr: ${stderr}`);
        }
        res.json(stdout.split('\n'));
    });
});
```

Figure 5.24: POST method implementation for the computer network lab.

- Tenth section:

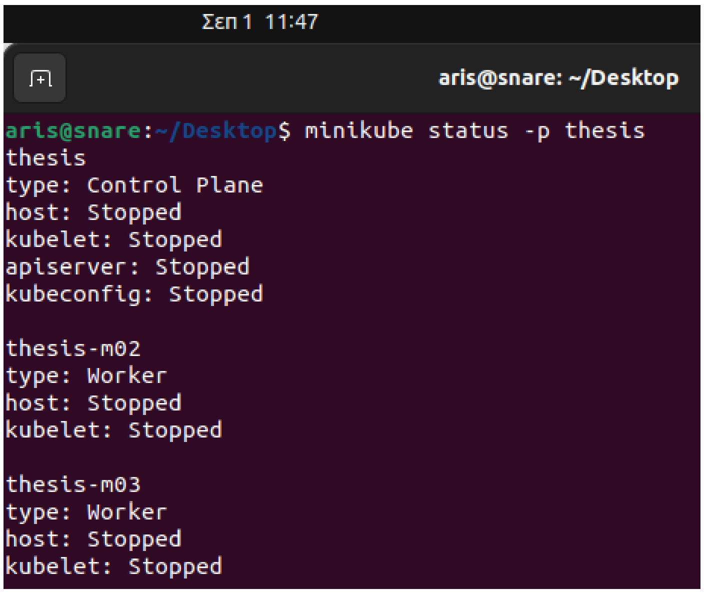


```javascript
app.post('/add-devops', (req, res) => {
    exec(`kubectl scale --current-replicas=5 --replicas=10 deployment/devopslab-deployment`, (error, stdout, stderr) => {
    //exec(`kubectl run new${new Date().getTime()} --image=nginx`, (error, stdout, stderr) => {

        if (error) {
            return res.send(`error: ${error.message}`);
        }
        if (stderr) {
            return res.send(`stderr: ${stderr}`);
        }
        res.json(stdout.split('\n'));
    });
});
```

Figure 5.25: POST method implementation for the devops lab.

5.8 Deploying our infrastructure.

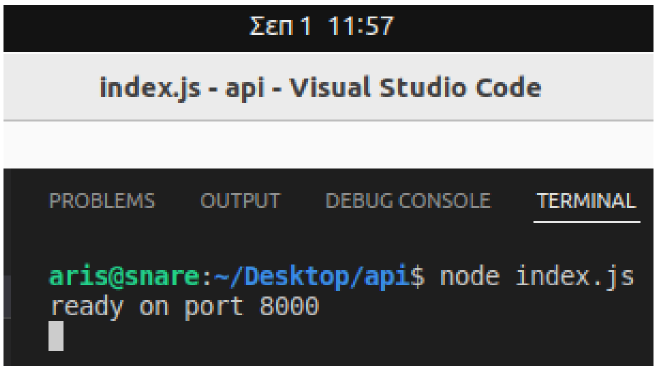
First of all, we have to ensure that our cluster is configured and stopped. From the command line, we run the command minikube status -p thesis. The output is depicted in the following figure:



```
aris@snare:~/Desktop$ minikube status -p thesis
thesis
type: Control Plane
host: Stopped
kubelet: Stopped
apiserver: Stopped
kubeconfig: Stopped

thesis-m02
type: Worker
host: Stopped
kubelet: Stopped

thesis-m03
type: Worker
host: Stopped
kubelet: Stopped
```

Figure 5.26: Status of the cluster before starting the infrastructure.

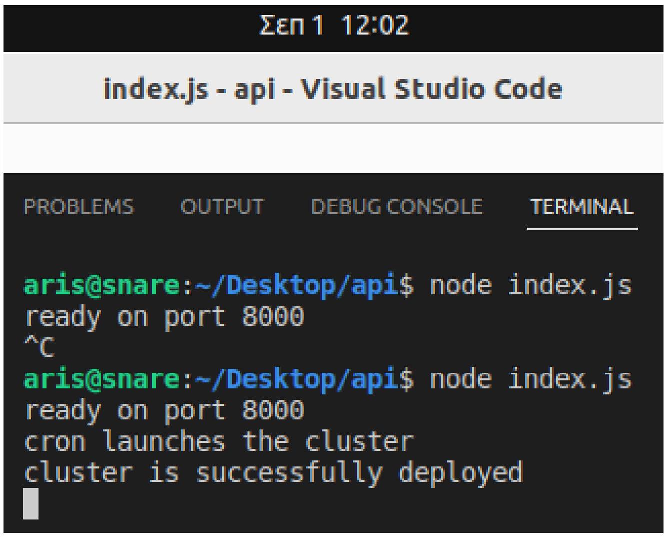
A few minutes before 12:00 p.m., we run the command node index.js to execute our program. The output is depicted in the following figure:



```
aris@snare:~/Desktop/api$ node index.js
ready on port 8000
```

Figure 5.27: First output after the execution of our program.

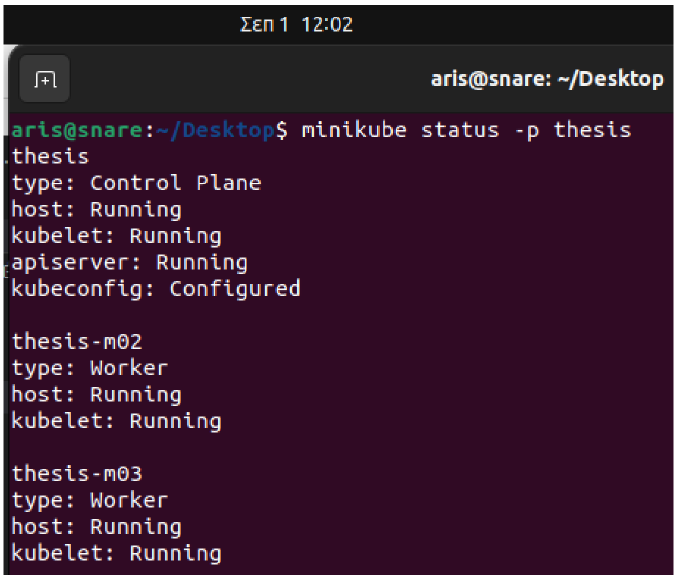
When the time is 12:00 o'clock, the log that signifies the start of the first cronjob, appears in the terminal. A few seconds later, a second log that signifies the successful launch of our cluster, also appears.



```
aris@snare:~/Desktop/api$ node index.js
ready on port 8000
^C
aris@snare:~/Desktop/api$ node index.js
ready on port 8000
cron launches the cluster
cluster is successfully deployed
```

Figure 5.28: Logs from the first cronjob.

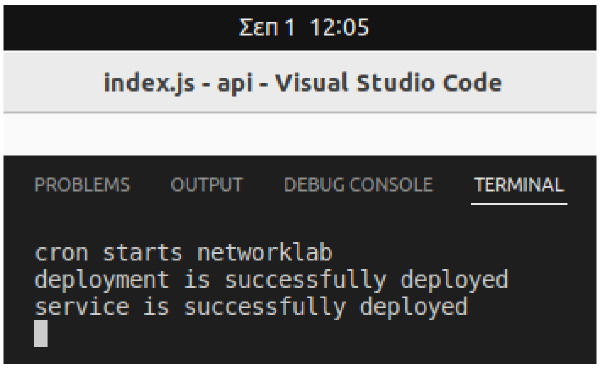
If we repeat the command minikube status -p thesis, that we used to check the status of cluster, the output will be different because now the master and the two worker nodes are running.



```
aris@snare:~/Desktop$ minikube status -p thesis
thesis
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

thesis-m02
type: Worker
host: Running
kubelet: Running

thesis-m03
type: Worker
host: Running
kubelet: Running
```

Figure 5.29: Status of the cluster.

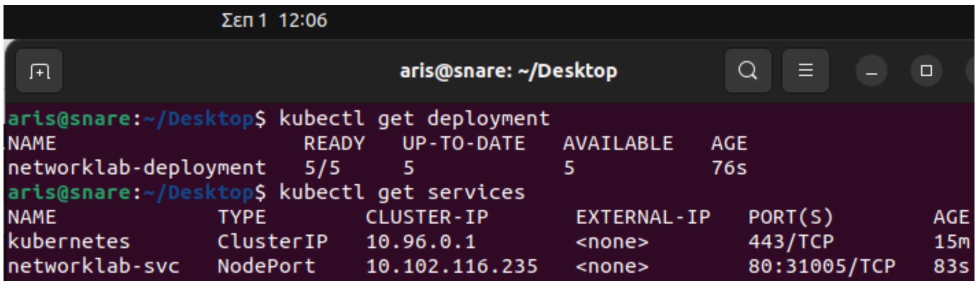
When the time is 12:05 p.m., three more logs will appear to the terminal. The first signifies the start of the second cronjob which activates the computer network virtual lab. The following two logs signify the successful activation of the deployment and the service of the computer network lab.



```
cron starts networklab
deployment is successfully deployed
service is successfully deployed
```

Figure 5.30: Logs from the second cronjob.

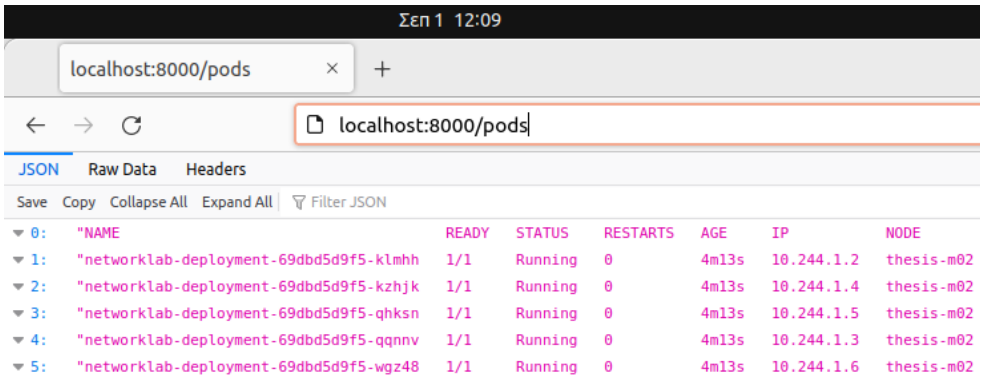
To check if the deployment and the service are activated, we run the kubectl get deployments and kubectl get services commands from our command line. The outputs are depicted in the following two figures.



```
aris@snare:~/Desktop$ kubectl get deployment
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
networklab-deployment   5/5     5            5           76s
aris@snare:~/Desktop$ kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        15m
networklab-svc  NodePort    10.102.116.235   <none>        80:31005/TCP   83s
```

Figure 5.31: Outputs.

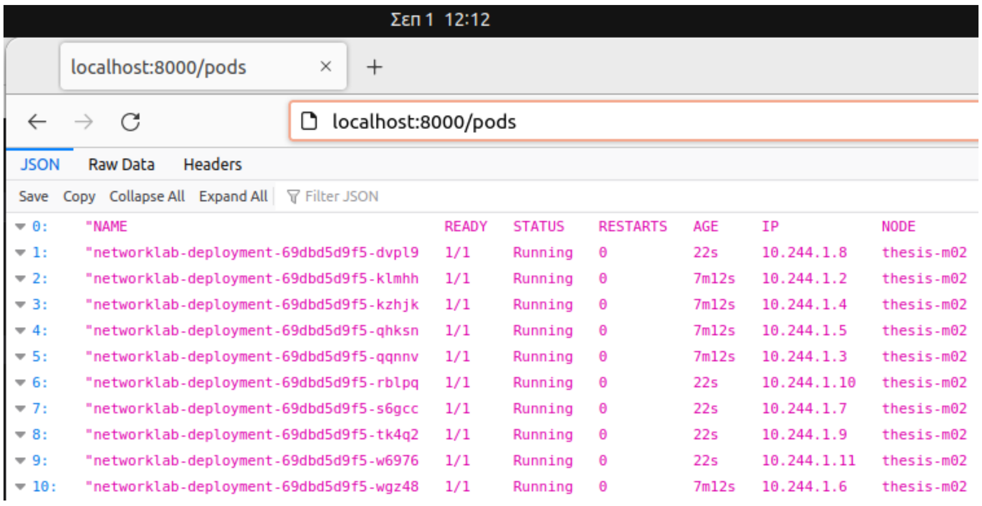
Now, if we want to check the status of the pods that are up and running, we open a browser and hit the http://localohost/pods URL. The GET method is activated and the following figure depicts the response that we receive.



```
▼ 0:  "NAME                                    READY  STATUS   RESTARTS  AGE    IP          NODE
▼ 1:  "networklab-deployment-69dbd5d9f5-klmhh  1/1    Running  0         4m13s  10.244.1.2  thesis-m02
▼ 2:  "networklab-deployment-69dbd5d9f5-kzhjk  1/1    Running  0         4m13s  10.244.1.4  thesis-m02
▼ 3:  "networklab-deployment-69dbd5d9f5-qhksn  1/1    Running  0         4m13s  10.244.1.5  thesis-m02
▼ 4:  "networklab-deployment-69dbd5d9f5-qqnnv  1/1    Running  0         4m13s  10.244.1.3  thesis-m02
▼ 5:  "networklab-deployment-69dbd5d9f5-wgz48  1/1    Running  0         4m13s  10.244.1.6  thesis-m02
```

Figure 5.32: GET response.

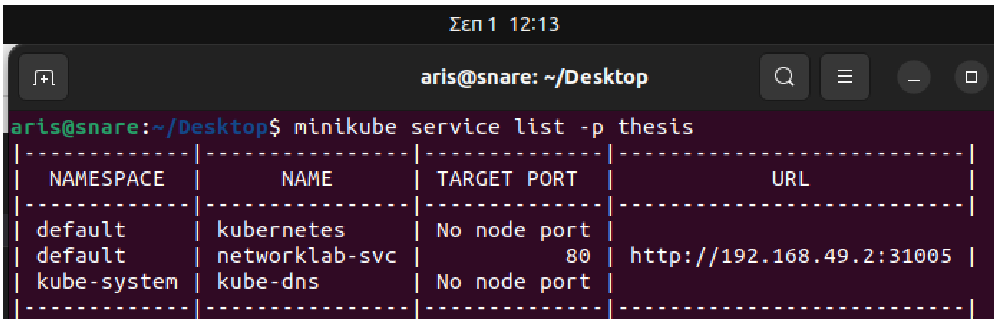
If we want to scale the number of pods, we run the command curl -X POST http://localhost:8000/add-network and if we hit again the http://localohost/pods URL, the output illustrates that five more pods are added, as it is depicted in the following figure.



```
▾ 0:   "NAME                                  READY   STATUS    RESTARTS   AGE     IP            NODE
▾ 1:   "networklab-deployment-69dbd5d9f5-dvpl9   1/1     Running   0          22s     10.244.1.8    thesis-m02
▾ 2:   "networklab-deployment-69dbd5d9f5-klmhh   1/1     Running   0          7m12s   10.244.1.2    thesis-m02
▾ 3:   "networklab-deployment-69dbd5d9f5-kzhjk   1/1     Running   0          7m12s   10.244.1.4    thesis-m02
▾ 4:   "networklab-deployment-69dbd5d9f5-qhksn   1/1     Running   0          7m12s   10.244.1.5    thesis-m02
▾ 5:   "networklab-deployment-69dbd5d9f5-qqnnv   1/1     Running   0          7m12s   10.244.1.3    thesis-m02
▾ 6:   "networklab-deployment-69dbd5d9f5-rblpq   1/1     Running   0          22s     10.244.1.10   thesis-m02
▾ 7:   "networklab-deployment-69dbd5d9f5-s6gcc   1/1     Running   0          22s     10.244.1.7    thesis-m02
▾ 8:   "networklab-deployment-69dbd5d9f5-tk4q2   1/1     Running   0          22s     10.244.1.9    thesis-m02
▾ 9:   "networklab-deployment-69dbd5d9f5-w6976   1/1     Running   0          22s     10.244.1.11   thesis-m02
▾ 10:  "networklab-deployment-69dbd5d9f5-wgz48   1/1     Running   0          7m12s   10.244.1.6    thesis-m02
```

Figure 5.33: GET response after POST request.

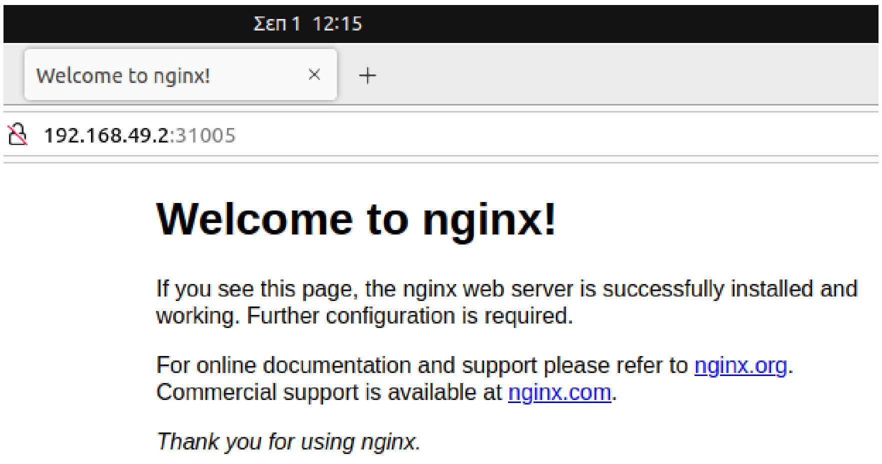
In order to have access to the image that these pods are containing, we run the command minikube service list -p thesis from our terminal and the output is depicted in the following figure.



```
aris@snare:~/Desktop$ minikube service list -p thesis
|-------------|-----------------|-------------|----------------------------|
|  NAMESPACE  |      NAME       | TARGET PORT |            URL             |
|-------------|-----------------|-------------|----------------------------|
| default     | kubernetes      | No node port |
| default     | networklab-svc  |          80 | http://192.168.49.2:31005 |
| kube-system | kube-dns        | No node port |
|-------------|-----------------|-------------|----------------------------|
```

Figure 5.34: List of services.

If the hit the link the output will be:



Welcome to nginx!

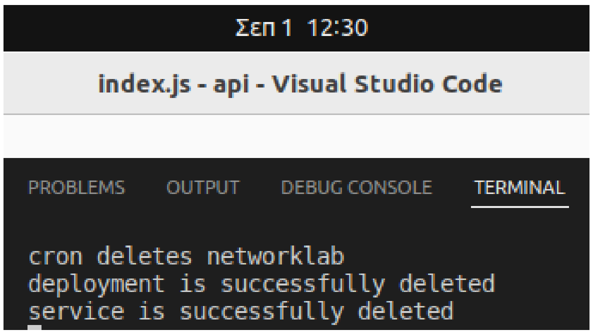
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 5.35: Nginx.

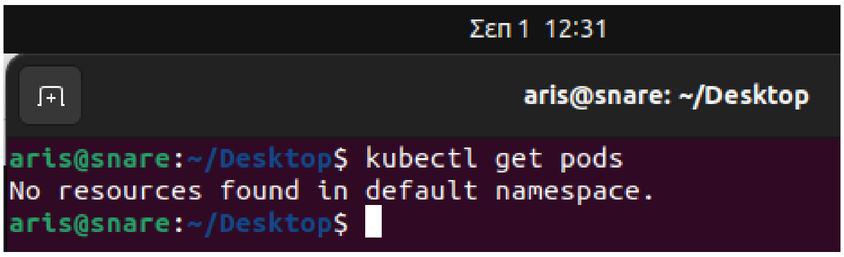
When the time is 12:30 p.m., three more logs will appear to the terminal. The first signifies the start of the third cronjob which deletes the computer network virtual lab. The following two logs signify the successful delete of the deployment and the service of the computer network lab.



```
cron deletes networklab
deployment is successfully deleted
service is successfully deleted
```

Figure 5.36: Logs from the third cronjob.

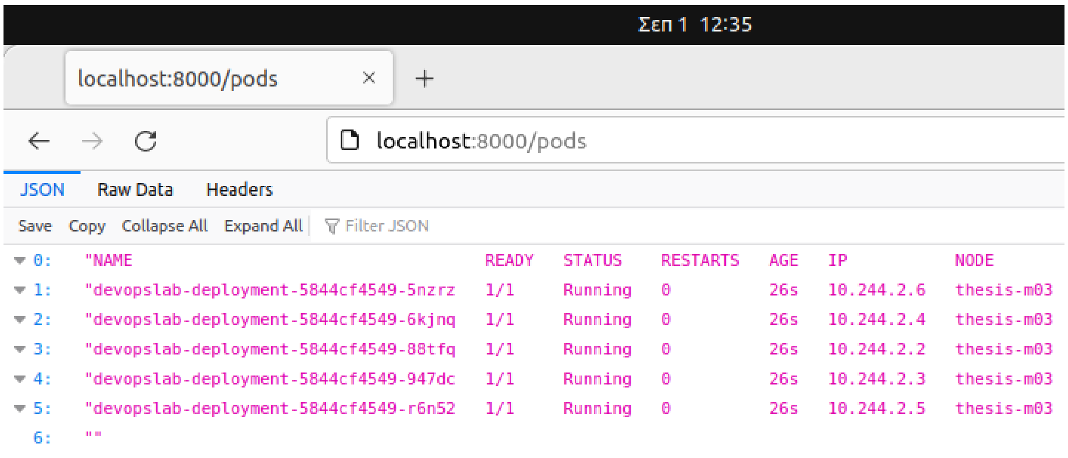
If we run the command kubectl get pods to monitor which pods are running, the output will be that there are no resources found.



```
aris@snare:~/Desktop$ kubectl get pods
No resources found in default namespace.
aris@snare:~/Desktop$
```

Figure 5.37: Output of the kubectl get pods command.

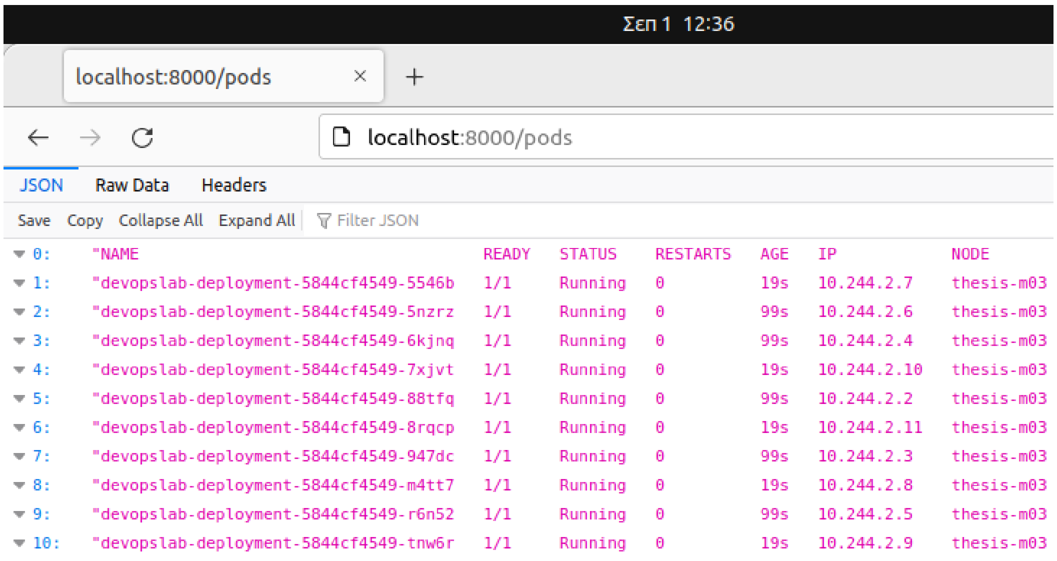
The same procedure will be followed to activate the devops virtual lab. When time is 12.35 p.m., the fourth cronjob will start and three more logs will appear to the terminal. A deployment and a service will be implemented in order to deploy five new pods for the devops virtual lab. Again, if we hit the http://localohost/pods URL the output will be:



```
▼ 0:  "NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE
▼ 1:  "devopslab-deployment-5844cf4549-5nzrz   1/1     Running   0          26s   10.244.2.6    thesis-m03
▼ 2:  "devopslab-deployment-5844cf4549-6kjnq   1/1     Running   0          26s   10.244.2.4    thesis-m03
▼ 3:  "devopslab-deployment-5844cf4549-88tfq   1/1     Running   0          26s   10.244.2.2    thesis-m03
▼ 4:  "devopslab-deployment-5844cf4549-947dc   1/1     Running   0          26s   10.244.2.3    thesis-m03
▼ 5:  "devopslab-deployment-5844cf4549-r6n52   1/1     Running   0          26s   10.244.2.5    thesis-m03
  6:  ""
```

Figure 5.38: GET response of the devops lab.

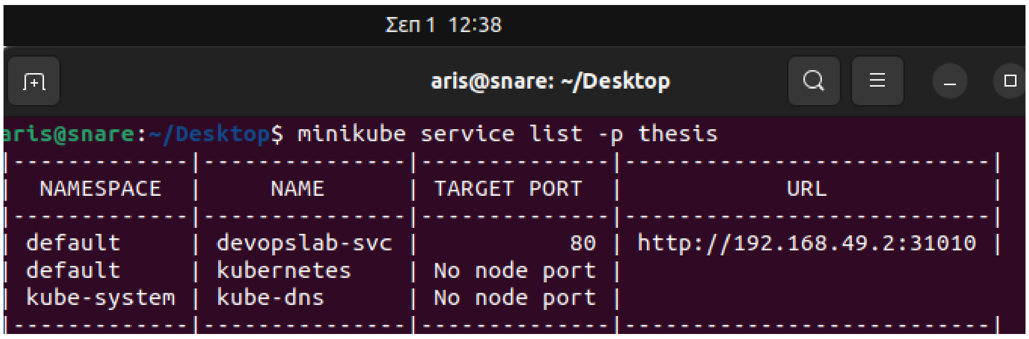
If we want to scale the number of pods, we run the command curl -X POST http://localhost:8000/add-network and if we hit again the http://localhost/pods URL the output illustrates that five more pods are added, as it is depicted in the following figure:



```
0:   "NAME                                     READY  STATUS   RESTARTS  AGE  IP           NODE
1:   "devopslab-deployment-5844cf4549-5546b   1/1    Running  0         19s  10.244.2.7   thesis-m03
2:   "devopslab-deployment-5844cf4549-5nzrz   1/1    Running  0         99s  10.244.2.6   thesis-m03
3:   "devopslab-deployment-5844cf4549-6kjnq   1/1    Running  0         99s  10.244.2.4   thesis-m03
4:   "devopslab-deployment-5844cf4549-7xjvt   1/1    Running  0         19s  10.244.2.10  thesis-m03
5:   "devopslab-deployment-5844cf4549-88tfq   1/1    Running  0         99s  10.244.2.2   thesis-m03
6:   "devopslab-deployment-5844cf4549-8rqcp   1/1    Running  0         19s  10.244.2.11  thesis-m03
7:   "devopslab-deployment-5844cf4549-947dc   1/1    Running  0         99s  10.244.2.3   thesis-m03
8:   "devopslab-deployment-5844cf4549-m4tt7   1/1    Running  0         19s  10.244.2.8   thesis-m03
9:   "devopslab-deployment-5844cf4549-r6n52   1/1    Running  0         99s  10.244.2.5   thesis-m03
10:  "devopslab-deployment-5844cf4549-tnw6r   1/1    Running  0         19s  10.244.2.9   thesis-m03
```

Figure 5.39: GET response after the POST request for the devops lab.

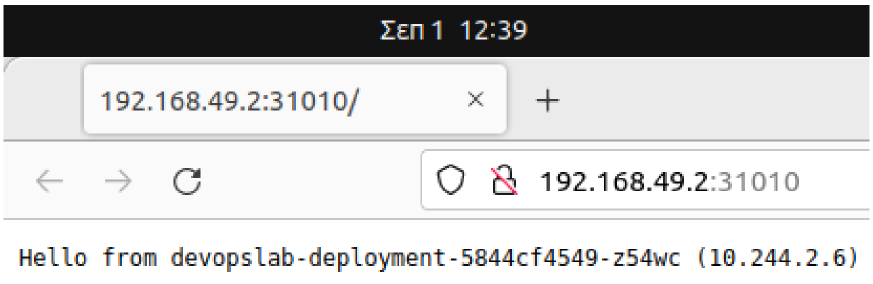
In order to have access to the image that these pods are containing, we run the command minikube service list -p thesis from our terminal and the output is depicted in the following figure.



```
aris@snare:~/Desktop$ minikube service list -p thesis
|-------------|--------------|-------------|---------------------------|
|  NAMESPACE  |     NAME     | TARGET PORT |            URL            |
|-------------|--------------|-------------|---------------------------|
| default     | devopslab-svc |          80 | http://192.168.49.2:31010 |
| default     | kubernetes   | No node port |
| kube-system | kube-dns     | No node port |
|-------------|--------------|-------------|---------------------------|
```

Figure 5.40: List of services.

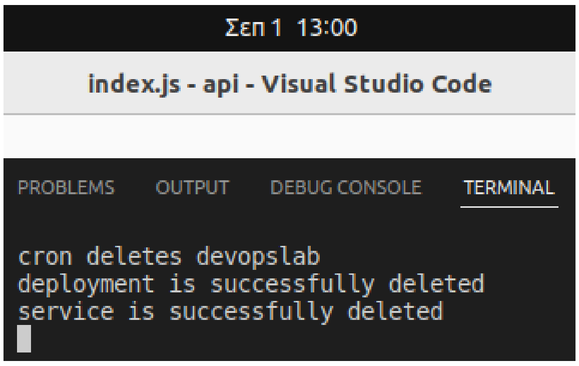
If the hit the link the output will be:



```
Hello from devopslab-deployment-5844cf4549-z54wc (10.244.2.6)
```

Figure 5.41:Figure 5.41: Hello from.

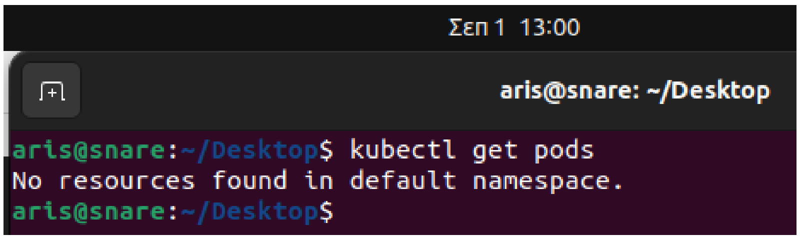
When the time is 13:00 o'clock, three more logs will appear to the terminal. The first signifies the start of the fifth cronjob which deletes the devops virtual lab. The following two logs signify the successful delete of the deployment and the service of the devops lab.



```
cron deletes devopslab
deployment is successfully deleted
service is successfully deleted
```

Figure 5.42: Logs from the fifth cronjob.

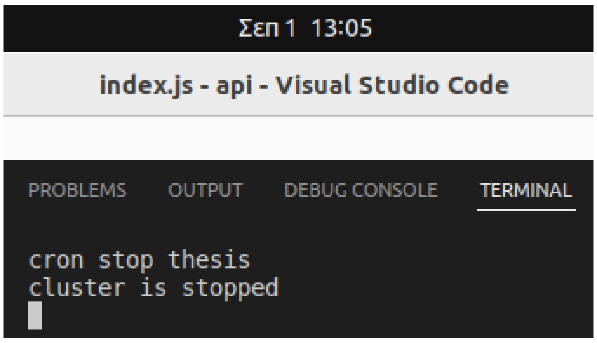
If we run the command kubectl get pods to monitor which pods are running, the output will be that there are no resources found.



```
aris@snare:~/Desktop$ kubectl get pods
No resources found in default namespace.
aris@snare:~/Desktop$
```

Figure 5.43: Output of the kubectl get pods command.

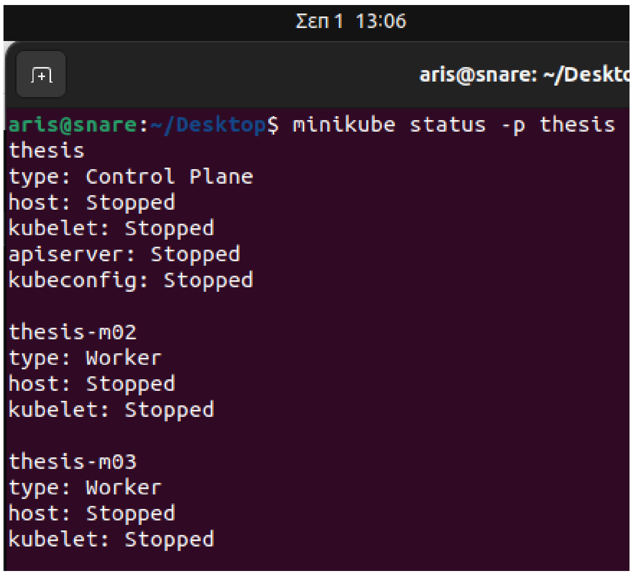
When the time is 13:05 p.m., the log that signifies the start of the sixth and last cronjob, appears in the terminal. A few seconds later, a second log that signifies the successful termination of our cluster, also appears.



```
cron stop thesis
cluster is stopped
```

Figure 5.44: Logs from the sixth cronjob.

If we run the command minikube status -p thesis, the output is depicted in the following figure:



```
aris@snare:~/Desktop$ minikube status -p thesis
thesis
type: Control Plane
host: Stopped
kubelet: Stopped
apiserver: Stopped
kubeconfig: Stopped

thesis-m02
type: Worker
host: Stopped
kubelet: Stopped

thesis-m03
type: Worker
host: Stopped
kubelet: Stopped
```

Figure 5.45: Status of the cluster after the termination of our infrastructure.

We observe that out thesis cluster in now stopped, but it is not deleted and it can be furtherly used in the following week in order to deploy these two virtual labs.

6 Conclusions

The evolution of Information Technology is also heavily based on the evolution of containerized technologies. Virtualization technology combined with cloud computing technology have paved the way for these containerized technologies to make even greater use of physical resources. Its main objective is to create infrastructures and applications with the minimum amount of physical equipment and personnel needed for their control, monitoring, and upgrading. In this thesis, we have demonstrated how to implement and manage an environment using the Kubernetes tool. This tool, while relatively new, will be in the IT spotlight for quite some time, since containers are the basis for deploying infrastructures.

Conclusions using Kubernetes in our thesis:

- Kubernetes ensures that if a container fails, it will automatically kill and will replace it with another one. Furthermore, it does not make it accessible to users until it is healthy and ready to serve. This whole process is done automatically, without any involvement of the administrator. In this way, self-healing is achieved.
- In addition, we saw how easily with a single command we can increase or decrease the number of associated Pods within the cluster. Thus, if the demand for a particular application increase, Kubernetes ensures that it is met quickly and easily.
- In addition, Kubernetes ensures that no matter what happens on our infrastructure, the cluster will contain as many Pods as the administrator defines. By creating a deployment and setting the number of replicas for a particular Pod, we ensure that this number will always exist in the cluster. Even if the administrator himself/herself, deletes one of the Pods created by the deployment, Kubernetes will automatically replace it.
- In addition, we have seen how easily we can assign to each pod either which service it belongs to or which worker node it will activate. That was accomplished using labels. In a large cluster, with a large number of pods, the administrator can easily group cluster elements according to the outcome he/she wants to achieve.

In the future, the practical part of this thesis may be differentiated as follows:

- Instead of using Minikube to implement our cluster, we can use Kubeadm, which according to the Kubernetes documentation, is the appropriate tool for a production environment.
- Instead of using cronjobs to implement different events depending on the day and time, we can implement and connect a database to our infrastructure.
- By creating a Pod, Kubernetes automatically allocates memory and CPU for each container. In order to take full advantage of our resources, we can instruct Kubernetes how much memory or CPU each container needs.

Bibliography – References – Internet Resources

[1] Semnanian, A. A., Pham, J., Englert, B., & Wu, X. (2011, April). Virtualization technology and its impact on computer hardware architecture. In *2011 Eighth International Conference on Information Technology: New Generations* (pp. 719-724). IEEE.

[2] Ameen, R. Y., & Hamo, A. Y. (2013). Survey of server virtualization. *arXiv preprint arXiv:1304.3557*..

[3] Kampert, P. E. (2010). A taxonomy of virtualization technologies.

[4] Virtualization evolution and history. Available at: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=9236&context=theses retrieved [Jan.20,2022]

[5] Kamla, R. Z., Yahiya, T., & Mustafa, N. B. (2018). An Implementation of Software Routing for Building a Private Cloud. *International Journal of Computer Network & Information Security*, *10*(3).

[6] Perez-Botero, D., Szefer, J., & Lee, R. B. (2013, May). Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing* (pp. 3-10).

[7] Alnaim, A. K., Alwakeel, A. M., & Fernandez, E. B. (2019, April). A pattern for an NFV Virtual Machine Environment. In *2019 IEEE International Systems Conference (SysCon)* (pp. 1-6). IEEE.

[8] Santana, G. A. (2013). Data center virtualization fundamentals: understanding techniques and designs for highly efficient data centers with Cisco Nexus, UCS, MDS, and beyond. Cisco Press..

[9] Business applications and activities provided by data centers. Available at: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html retreived [Feb.2,2022]

[10] Eriksson, Martin.(2018) .*Monitoring, modeling and identification of data center servers*.

[11] Data Center Tier Classification. Available at: http://ipwithease.com retreived [Feb.7,2022]

[12] Halabi, S. (2019). *Hyperconverged Infrastructure Data Centers: Demystifying HCI*. Cisco Press..

[13] Riteau, P. (2011, May). Building dynamic computing infrastructures over distributed clouds. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum* (pp. 2097-2100). IEEE.

[14] Alhamad, M., Dillon, T., & Chang, E. (2010, April). Conceptual SLA framework for cloud computing. In *4th IEEE international conference on digital ecosystems and technologies* (pp. 606-610). IEEE.

[15] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

[16] Erl, T., Puttini, R., & Mahmood, Z. (2013). *Cloud computing: concepts, technology, & architecture*. Pearson Education.

[17] Cloud Figure. Available at: https://clipground.com/cloud-computing-clipart.html retreived [Feb 12,2022]

[18] IBM cloud computing definition. Available at: https://www.ibm.com/cloud/learn/cloud-computing retreived [Feb 12,2022]

[19] Velte, A. T., Velte, T. J., Elsenpeter, R. C., & Elsenpeter, R. C. (2010). Cloud computing: a practical approach.

[20] Thomas, E., Zaigham, M., & Ricardo, P. (2013). Cloud Computing Concepts, Technology & Architecture.

[21] Tsai, W., Bai, X., & Huang, Y. (2014). Software-as-a-service (SaaS): perspectives and challenges. *Science China Information Sciences*, *57*(5), 1-15.

[22] Goran, V., Monika, S., Sasko, R., & Marjan, G. (2014). BUSINESS CASE: FROM IAAS TO SAAS. SMEs DEVELOPMENT AND INNOVATION: BUILDING COMPETITIVE FUTURE OF SOUTH-EASTERN EUROPE, 801.

[23] Rimal, B. P., Choi, E., & Lumb, I. (2009, August). A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC* (pp. 44-51). IEEE.

[24] Advantages and comparison of Cloud Computing models. Available at: https://sam-solutions.us/advantages-and-disadvantages-of-cloud-deployment-models/ retreived [Mar ,2022]

[25] Mousavi Shoshtari, S. F. (2013). Cloud Computing Adoption in Iran as a Developing Country: A Tentative Framework Based on Experiences from Iran.

[26] Aspernäs, A., & Nensén, M. (2016). Container Hosts as Virtual Machines: A performance study.

[27] Barik, R. K., Lenka, R. K., Rao, K. R., & Ghose, D. (2016, April). Performance analysis of virtual machines and containers in cloud computing. In *2016 international conference on computing, communication and automation (iccca)* (pp. 1204-1210). IEEE.

[28] Docker Docs, Docker official logo. Available at: https://www.docker.com/company/newsroom/media-resources/ retreived [Mar.14,2022]

[29] Jaramillo, D., Nguyen, D. V., & Smart, R. (2016, March). Leveraging microservices architecture by using Docker technology. In *SoutheastCon 2016* (pp. 1-5). IEEE.

[30] Docker Docs, Docker Network Drives. Available at: https://docs.docker.com/network/ retreived [Mar.17,2022]

[31] Suo, K., Zhao, Y., Chen, W., & Rao, J. (2018, April). An analysis and empirical study of container networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (pp. 189-197). IEEE.

[32] Marmol, V., Jnagal, R., & Hockin, T. (2015). Networking in containers and container clusters. *Proceedings of netdev 0.1*, 14-17.

[33] Al Jawarneh, I. M., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., & Palopoli, A. (2019, May). Container orchestration engines: A thorough functional and performance comparison. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE.

[34] Naik, N. (2021, April). Performance evaluation of distributed systems in multiple clouds using docker swarm. In *2021 IEEE International Systems Conference (SysCon)* (pp. 1-6). IEEE.

[35] Docker Docs, Swarm mode overview. Available at: https://docs.docker.com/engine/swarm/ retreived [Mar.28,2022]

[36] Kubernetes Docs, Kubernetes official logo. Available at: https://kubernetes.io/ retrieved [April.4,2022]

[37] Hausenblas, M. (2018). *Container Networking*. O'Reilly Media, Incorporated.

[38] Kubernetes Docs. Kubernetes architecture. Available at: https://kubernetes.io/docs/concepts/overview/components/ retrieved [April.20,2022]

[39] https://www.techtarget.com/searchitoperations/tip/Ensure-Kubernetes-high-availability-with-master-node-planning

[40] Uphill, T., Arundel, J., Khare, N., Saito, H., Lee, H. C. C., & Hsu, K. J. C. (2017). *DevOps: Puppet, Docker, and Kubernetes*. Packt Publishing Ltd.

[41] Kubernetes Docs. Kubernetes Networking. Available at: https://kubernetes.io/docs/concepts/cluster-administration/networking/ retrieved [April.24,2022]

[42] Kubernetes Docs. Kubernetes services. Available at: https://kubernetes.io/docs/concepts/services-networking/service/ retrieved [April.24,2022]

[43] Kubernetes Docs. Kubernetes volumes. Available at: https://kubernetes.io/docs/concepts/storage/volumes/ retrieved [April.25,2022]

[44] Scheckler, R. K. (2003). Virtual labs: a substitute for traditional labs?. *International journal of developmental biology*, *47*(2-3), 231-236.

[45] Xevgenis, M. G., Toumanidis, L., Kogias, D. G., & Patrikakis, C. Z. (2016, December). The Virtual Lab (VLAB) Cloud Solution. In *2016 IEEE Globecom Workshops (GC Wkshps)* (pp. 1-5). IEEE..

[46] Format of cronjobs. Available at: https://www.looklinux.com/top-20-crontab-examples-to-schedule-tasks/ retrieved [Sept.9,2022]