Università
Ca'Foscari
Venezia

Master's Degree
in Computer Science

Final Thesis

Virtualization and Containerization

a new concept for data center management
to optimize resources distribution

Supervisor
Dottor Fabrizio Romano

Graduand
Enrico Martin
Matriculation number
815635

Academic Year
2020 / 2021

Abstract

The role of cloud computing has led the way data center services are offered, utilized and handled over the internet. Typically, applications run inside the virtual machines in an isolated environment. Nevertheless, a considerable hardware virtualization overhead seems to be inevitable. Recently, Docker containers have gained noticeable attention because of their substantially lower overhead if compared to virtual machines using operating system virtualization. This prominent technology mainly provides isolation, portability, interoperability, scalability and high availability. Hence that is being widely adopted and everybody is trying to shift their software to Docker containers with the support of tools and frameworks.

Containers are revolutionizing the way software is delivered and deployed. Particularly with the use of cloud orchestration tools like Kubernetes, a new approach to high availability and fault tolerance is emerging.

The thesis illustrates the evolution of virtualization technology and the switch to containerization, to focus on the migration of data center processes in a containerized environment using Kubernetes as an orchestration tool. Two case examples are discussed in order to evaluate the performance of a web server and a file server. In addition to that, a cost/benefits analysis is introduced to estimate the advantages such a strategy could lead to.

Acknowledgements

I would first like to thank my thesis advisor Professor Fabrizio Romano who constantly mentor me trough this research, by providing guidance and support.

Finally, I must express my very profound gratitude to my parents, my brother and to all my friends and work colleagues for always expressing constant support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them all.

Thank you.

Contents

Chapter 1. Virtualization....4
1.1 A brief history of virtualization....5
1.2 Hardware virtualization and the role of the hypervisor....10
1.3 Operating-system-level virtuaqlization....12
1.4 Open source impletations....12
1.5 Commercial applications....15
1.6 Security vulnerabilities in virtualized environments....17
1.7 High availability and fault tolerance....20
1.8 Virtualization benefits and drawbacks....21

Chapter 2. Containerization....24
2.1 Differences between virtualization and containerization....25
2.2 Where’s the hypervisor?....27
2.3 What happends in Windows? The hidden Linux VM....31
2.4 Docker architecture. Containers/Images/Repos/UnionFS....32
2.5 Other container engines....36
2.6 Security in containerized environments....38
2.7 High availability and fault tolerance issues....39

Chapter 3. Orchestration....41
3.1 Brief introduction....42
3.2 Kubernetes architecture....46
3.2.1 etcd....47
3.2.2 kube-scheduler....47
3.2.3 kube-controller-manager....47
3.2.4 kubelet....48
3.2.5 Container runtime....48
3.2.6 API Services....48
3.2.7 PODs....48
3.2.8 Configurations and secrets....50
3.2.9 Resource management....51

Chapter 4. From virtualized hosts to containerized applications....52
4.1 Datacenter context....53
4.2 Tests and bechmarking....53
4.2.1 Apache webserver....53
4.2.2 Fileserver....54
4.2.3 OverLeaf....55
4.2.4 Voting webAPP....59
4.3 A cost/benefits analysis....62
4.4 From on-premise solutions to cloud platforms....65
4.5 Hybrid solutions....67

Chapter 5. Conclusions and future works....69

Chapter 1

Virtualization

In this chapter, a brief history of virtualization will be presented in order to understand the evolution of computer architecture that led to the introduction of virtualization, its benefits and drawbacks

1.1 A brief history of virtualization

Virtualization is a technique which traces its origins back to late 1960s / early 1970s when a single operating system and batch jobs were used to carry out IT tasks using punched cards, processing only one operation at a time, resulting in inefficient use of CPU time.[1]

Things branched out when the Boston Massachusetts Institute of Technology (MIT) announced the MAC Project in 1963 for which a computer hardware capable of more than one simultaneous user was needed. For this purpose the Compatible Time-Sharing System (CTSS)[2] was conceived using Defense Advanced Research Projects Agency (DARPA) funds. CTSS was developed at the MIT Computation Center and it was one of the first time-sharing operating systems.[3] Among others, the MAC Project aimed to develop the successor of CTSS, known as MULTICS, meant to be the first high availability computer, developed as part of a consortium of companies that included General Electric and Bell Laboratories.[4]

Time-sharing refers to the partitioning of computing resources among many users through multiprogramming and multi-tasking at the same time allowing the CPU to run different processes concurrently, giving them a slot of execution time and stopping them when I/O access is required or the time slot ends.

On June 30th, 1970 IBM announced the System/370 as the successor generation of mainframes to the System/360. As with the System/360 series, this new generation of mainframes didn’t support virtual memory. However, in 1972, IBM changed direction announcing that the option would have been made available on all System/370 models.[5] Those were the first steps into computer virtualization and the introduction of a layer to virtualize machines, called hypervisor.

1 *"History of Operating Systems", University of Washington. Retrieved Nov 21st, 2019*

2 *“An Experimental Time Sharing System” , Fernando J. Corbató, Marjorie Merwin Daggett, Robert C. Daley, Computation Center, Massachusetts Institute of Technology, Cambridge, Massachusetts*

3 *CTSS used a modified IBM 7090 mainframe computer that had two 32,768 (32K) 36-bit-word banks of core memory instead of the normal one. One bank was reserved for the time-sharing supervisory program, the other for user programs. CTSS had a protected-mode kernel, the supervisor's functions in the A-core (memory bank A) could be called only by software interrupts, like in the modern operating systems. Causing memory-protection interrupts were used for software interrupts.*

4 *John McCarthy, “Reminiscences on the History of Time Sharing”, Archived 2007-10-20 at the Wayback Machine (Stanford University 1983).*

5 *Pugh, E.W.; L.R. Johnson; John H. Palmer (1991). “IBM's 360 and early 370 systems.”, Cambridge: MIT Press.*

Virtual memory is a memory management technique that maps memory addresses used by a program, called virtual addresses, into physical addresses in computer main memory. Virtual memory makes processes see main memory as a contiguous address space. The mapping between virtual and real memory is managed by the operating system and the Memory Management Unit (MMU), a hardware component usually implemented as part of the central processing unit (CPU) that automatically translates virtual addresses to physical addresses.

Moreover, virtual memory enables the operating system to provide a virtual address space that can also be bigger than real memory so processes can reference more memory than the size that is physically present. This technique is called paging.

Virtual memory first appeared in the 1960s, when main memory was very expensive and very few. Virtual memory enabled software systems with large memory demands to run on computers with less real memory and also eliminated the need for coders to handle memory explicitly. Soon, because of the savings introduced by this technique, many software systems started to turn on virtual memory by default.

The concept of virtual memory, or virtual storage as IBM used to refer to on mainframe operating systems, was first developed by German physicist Fritz-Rudolf Güntsch at the Technische Universität Berlin in 1956.[6] Paging was first implemented at the University of Manchester as a way to extend the Atlas Computer's working memory. The primary benefits of virtual memory included freeing applications from having to manage a shared memory space, increased security due to memory isolation, and increased memory availability for processes by exploiting the paging technique.

In addition to that, another memory management technique was introduced to handle the retrieving of data from secondary storage to be used in main memory: paging; which is a memory management scheme by which an operating system stores and retrieves data from secondary storage (e.g., hard disks) for use in main

6 *Bhattacharjee, Abhishek; Lustig, Daniel (2017). Architectural and Operating System Support for Virtual Memory. Morgan & Claypool Publishers.*

memory. The operating system retrieves data from secondary storage in same-size blocks called pages, hence the name paging.[7]

Those two techniques; time-sharing and virtual memory, made possible the future implementation of a software layer called hypervisor. In their 1974 article, "Formal Requirements for Virtualizable Third Generation Architectures" Gerald J. Popek and Robert P. Goldberg classified two types of hypervisors[8]: Type-1; native or bare-metal hypervisors and Type-2; or hosted hypervisors. The two types helped distinguish between hypervisors that run on a physical machine and those that run on top of an existing operating system. Type-1 hypervisors provide increased security because of their location in the physical hardware, which eliminates the attack surface often offered by an operating system. Type-2 hypervisors, on the other hand, are closely related to the beginnings of x86 virtualization and are generally used for client or end-user systems. This is because Type-2 hypervisors run on top of the host operating system, which can lead to latency issues and security risks.

Virtualisation was dropped during the late 2000 when client-server software and cheaper x86 servers and desktops open towards distributed computing. The success of Windows and the appearance of Linux as server operating systems settle a new de facto industry standard. Opposing to mainframes, x86 computers weren't intended to achieve full virtualisation.

In 1999 Vmware Virtual Platform for Intel architecture was launched, the first x86 hardware based on research at Stanford University[9]. Hardware-assisted virtualization is a way that makes possible effective usage of the underlining hardware functionalities. It emulates entirely the hardware components for the virtual machine, where a guest operating system, using the same instruction set architecture as the host machine, executes the instructions in complete isolation.[10]

A series of new techniques for hardware support were adopted during the definition of x86 standard which led to protected mode where the operating system is

7 *Deitel, Harvey M. (1983). "An Introduction to Operating Systems.", Addison-Wesley. pp. 181, 187. ISBN.*
8 *"Formal Requirements for Virtualizable Third Generation Architectures," Gerald J. Popek and Robert P. Goldberg, 1974*
9 *See https://en.wikipedia.org/wiki/Hardware-assisted_virtualization*
10 *See https://en.wikipedia.org/wiki/X86_virtualization*

executed in a privileged mode while the rest of operations are executed without those privileges.[11]

Those techniques had the effect to worsen the performance of the host operating systems so a first attempt used in x86 software-based virtualization to overcome this limitation was *ring deprivileging*.[12]

Three techniques were adopted:

- binary translation[13] operates when an unprivileged instruction is asked to be executed by the host operating system without the necessary privileges, so that a rewritten operation is passed to kernel thus avoiding the trap-and-emulate phase[14].

- a shadowing process must be apply to certain key data structures because operating systems use paged virtual memory, so giving the guest operating system direct access to memory will result in a uncontrolled access to the main memory, causing possible conflict in the addressing mechanism by the the hypervisor. A technique was introduced to avoid it, called *shadow page tables* for which a local version of the page table is presented to the guest operating system shares the actual page table and descriptor are managed by the host operating system by trapping the access requests[15].

- I/O emulation for hardware which is not directly supported by the host operating system.[16] [17]

A distinct path was chosen by other systems called paravirtualization demanding a software porting of the entire operating systems to be executed on the virtual ma-

11 *“A Comparison of Software and Hardware Techniques for x86 Virtualization”, Keith Adams and Ole Agesen, VMware, ASPLOS’ 06 October 21–25, 2006, San Jose, California, USA*
12 *"Intel Virtualization Technology Processor Virtualization Extensions and Intel Trusted execution Technology", Intel 2007*
13 See https://en.wikipedia.org/wiki/X86_virtualization
14 *Trap-and-emulate is a technique used by the virtual machine to emulate privileged instructions and registers and pretend to the operating system that it is still in kernel mode.*
15 *In memory addressing for Intel x86 computer architectures, segment descriptors are a part of the segmentation unit, used for translating a logical address to a linear address. Segment descriptors describe the memory segment referred to in the logical address.*
16 *"VMware and Hardware Assist Technology", Archived from the original on 2011-07-17. Retrieved 2010-09-08*
17 *“A comparison of software and hardware techniques for x86 virtualization”, Keith Adams and Ole Agesen, Vmware - ACM SIGOPS Operating Systems Review, Volume 40, Issue 5 December 2006*

chine so that doesn’t implement the x86 ISA resulting in a significant I/O performance gaining.

Those limitations opened the way to full virtualization and paravirtualization.[18] Both mimic the physical underlining hardware to accomplish the operating system abstraction from the physical hardware, displaying some loss in performance and increased complexity.

1. **Full virtualization** also known as Native virtualization was introduced in first generation x86 to run unmodified guest operating systems to take advantage of direct accessible instruction set. In this scenario, a guest operating system uses the same instruction set, interrupts and memory access as the host one, as result most of the instructions can be directly executed on the bare physical system without performance loss. VirtualBox, VMware Workstation, and Microsoft Virtual PC, are well-known commercial implementations of full virtualization.

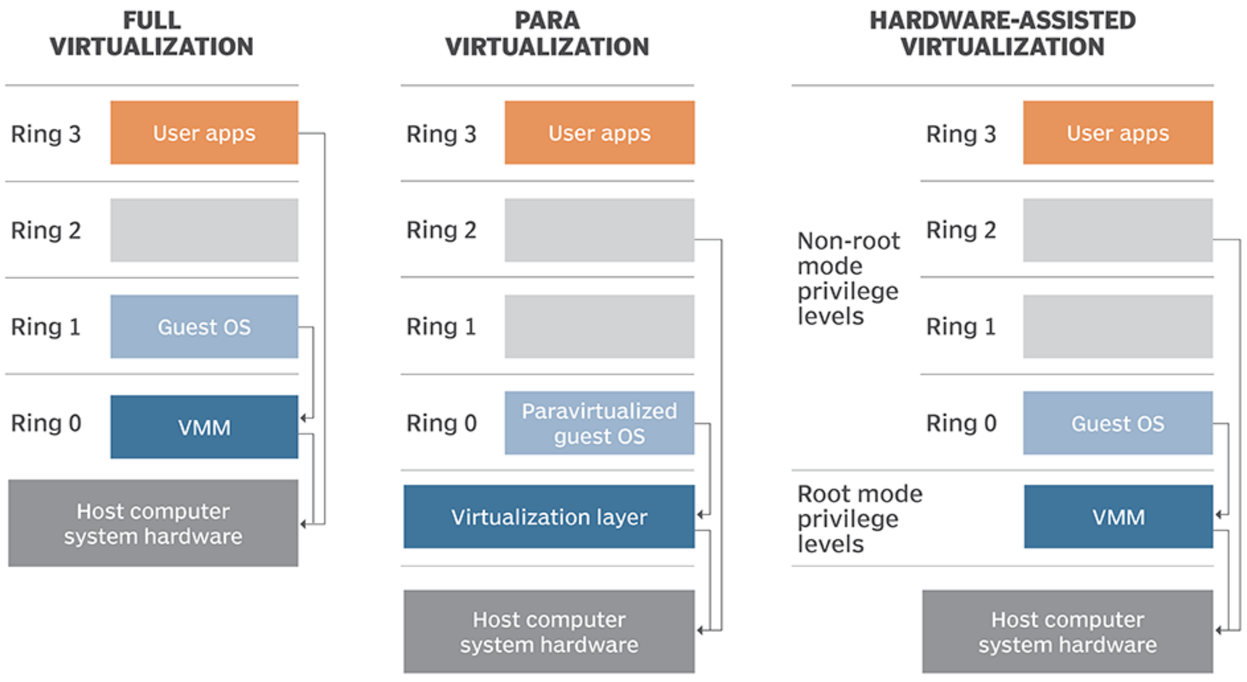
2. **Paravirtualization[19]** is a different approach to virtualization that opt for a partial modification of the kernel operating system to boost the performance of some specific operations. The way such performance are achieved is using hypervisor’s APIs, thus the need to modify the kernel, giving the guest operating system direct access to the hypervisor so that the execution id done without the intervention of the guest virtual machine, avoiding calling the virtualization layer, where performance are much worse if compared to a non virtualized environment.

A naif hardware-assisted virtualization concept is possible at a cost of using virtual machine APIs or triggers which reduce CPU performance, thus reducing efficiency and scalability. Even though no modifications have to be implemented on the ker-

18 Chris Barclay, *New approach to virtualizing x86s*, Network World, 20 October 2006
19 *"VMware Introduces Support for Cross-Platform Paravirtualization – VMware". VMware. 16 May 2008. Archived from the original on 13 April 2011.*

nel side, the performance lacks could be soothed adopting a so-called hybrid-virtualization.[20] [21]

Nowadays those techniques paved the way to containerization, which will be thoroughly discussed in the next chapter.



1.2 Hardware virtualization and the role of the hypervisor

Virtualization technology involves the abstraction of computer hardware like CPU, HHD, network interfaces, RAM, software stack, and so on. Virtualization mimic the hardware components behavior running a virtual version of them.

20 *"Hybrid Virtualization: The Next Generation of XenLinux", Archived March 20, 2009, at the Wayback Machine*
21 *Jun Nakajima and Asit K. Mallick, "Hybrid-Virtualization—Enhanced Virtualization for Linux", Archived 2009-01-07 at the Wayback Machine, in Proceedings of the Linux Symposium, Ottawa, June 2007.*

Hardware Level Virtualization has extensively been adopted over the last twenty years using an hypervisor to abstract the interface layer between software and the underlying hardware. Hypervisor is a piece of software that makes the illusion of a virtual component directly running on the hardware, isolated from the underlying host operating system. Two types of hypervisors exist:

- **Type-1 - Native or Bare-metal Hypervisor** is executed without external intervention to the host hardware, without any operating system in between as shown in figure 2.1, handling the hardware requests from the guest operating system so that playing the role of the intermediary;

- **Type-2 - Hosted Hypervisor** is part of the underlining operating system providing an access to the hardware as shown in figure 2.2, so that it assures full isolation that guarantees abstraction from the host computer hardware. That kind of virtualization is as well acknowledged as full-virtualization.

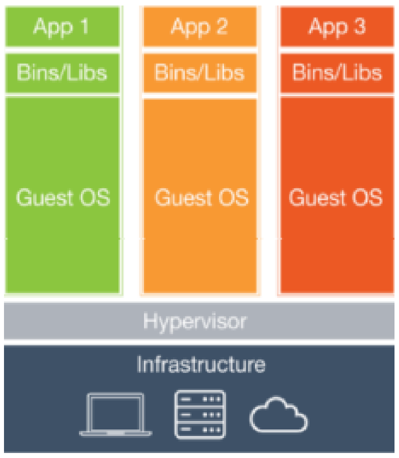


Figure 2.1: Type 1: Native or Bare-metal Hypervisor

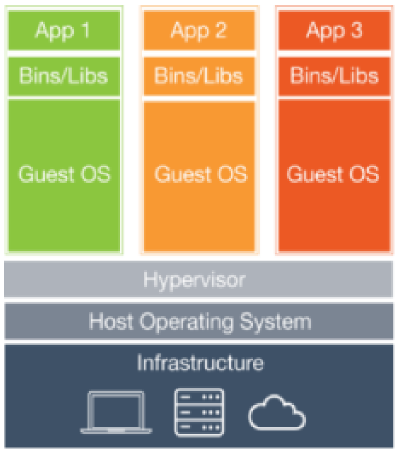


Figure 2.2: Type 2: Hosted Hypervisor

The importance of having an isolated environment where numerous virtual machines are able to run independently at the same time is obvious if one consider the case a virtual machine try and trigger a privileged command to shut down itself. If there isn’t an isolated environment there will be a chance the instruction will turn the entire system down and not only that very virtual machine.[22]

1.3 Operating-system-level virtualization

The wide adoption of virtualized computers has welcomed a variety of beneficial outcomes on computing systems and cloud computing as well. The major and outstanding improvement of the adoption of virtualized environments is that the investments in infrastructure have been lowered.

The necessity of CPU virtualization is caused by the requirement of a dedicated processor for each of the virtual machines running and, as physical hosts typically have one or two central processing units, the abstraction of the CPU is mandatory.

Operating system level virtualization, otherwise called lightweight virtualization, is a technique where a kernel of an operating system grants the copresence of more than one isolated userspaces rather than just one. Those isolated userspaces are also referred to as containers or jail or chroot jails in BSD environments. That kind of resource segmentation often comes free of costs of with irrelevant overhead as containers benefit from the underlying operating system calls so avoiding any form of emulation.[23]

1.4 Open source implementations

- **Xen Project**

  Is an open-source virtualization project focused on pushing virtualization in both commercial and open source applications, not to mention server virtualization with Linux or Windows operating system. Its free and open source hypervisor is considered to be one of the best for paravirtualization and it is of-

22 *Smith, Jim; Nair, Ravi (2005). Virtual Machines. Morgan Kaufmann. Performance Enhancement of System Virtual Machines.*

23 *Gerardus Blokdyk, Operating System Level Virtualization – A complete guide, 2020*

ten judged the default standard in Linux. Xen offers a reliable, flexible and secure setting with a particular attention to security issues, both on-premise and in cloud giving the flexibility to run different operating systems.[24]

- **VirtualBox[25]**

  VirtualBox from Oracle is a performing x86/AMD64 open source software running on Linux, Windows and many other operating systems, providing full virtualization for a variety of host operating systems. It takes advantage of hardware virtualization (Intel VT-x and AMD-V) to execute code natively thus abstracting from the underneath OS.

  It furthermore can run without hardware virtualization, thus effectively running on any system without Intel VT-x or AMD-V technology and is possible to demote among VMs or cloud transparently.

- **KVM**

  Kernel-based Virtual Machine, also known as KVM is a solution running both on Intel64 and AMD64 architecture. The project was first released in 2006 as part of the Linux ecosystem. KVM components incorporate among others, automatic NUMA (Non-Uniform Memory Access) balancing, Virtual CPU, and capping disk I/O bandwidth between host and guest machines. KVM keeps a high degree of integration with the operating system, and as is part of Linux OS it offers great performance ensuring near-bare-metal performance. It is scalable and secure, due to its coupling with the operating system and benefits from a worldwide support open source community.

- **oVirt**

  A community project initially founded by RHAT, oVirt is a free and open-source virtualization management platform alternative that implements a centralized integration between host machine, storage infrastructure and

24 See https://xenproject.org/
25 See https://www.virtualbox.org/

virtual/physical network. Based upon KVM and the library libvirt, oVirt includes support for virtualized networks and storage and Red Hat, Canonical, Cisco, IBM, Intel, NetApp and SUSE are involved in this project, giving to it a wide open source and commercial support.

Like it commercial concurrent ESXi, oVirt also has paravirtualized drivers for VMs called VirtIOs. These drivers are able to improve the performance of guest operating systems, making their performance comparable to a bare-metal system. VirtIO drivers are available for virtualizing network cards, hard drives, video cards, and optimizing memory usage. When these drivers are used, the guest OS then becomes aware that it is virtualized on a hypervisor and uses VirtIO as a front-end for the virtualized peripherals using QEMU. Where these drivers are not available, emulated peripherals are used, with obviously lower performance.[26]

- **ProxMox**

ProxMox is a virtualization environment open source, reliable, flexible and easy to use by implementing most of the features of an Enterprise solution. Is a bare metal hypervisor based on a GNU Linux distribution and melting together KVM and LXC technologies. Its capability to handle both Kernel-based Virtual Machine (KVM) and LXC for containers using an integrated interface is ideal to operate an entire stack of a virtual data center.

By natively integrating support for high availability (HA) and thanks to the multi-master design, an additional management server is not required, saving resources and allowing high availability without a single point of failures (SPOF). Using the integrated live/online migration feature, it is possible to move running virtual machines from one Proxmox Virtual Environment cluster node to another without downtime or noticeable effects on the end user side.[27]

26 *See https://www.ovirt.org/*
27 *See https://www.proxmox.com*

1.5 Commercial applications

- **Red Hat Enterprise Virtualization (RHEV)**

  Red Hat Enterprise Virtualization (RHEV) is a commercial implementation of the KVM Type-1 hypervisor. Red Hat Enterprise Virtualization uses SPICE[28] protocol and VDSM (Virtual Desktop Server Manager) with a RHEL-based centralized management server. RHEV offers support the following advanced features:

  Network bonding, VLAN, and 10GB;
  Live migration, policy-based workload balancing, high availability, power saving, cluster maintenance, image management, templating, thin-provisioning, and event monitoring;
  Hosts support up to 160 cores and 2 TB of RAM. Guests support up to 64 vCPUs and 512 GB of RAM;
  Reporting and monitoring, detailed historical reporting capabilities, monitor historical usage, trending, quality of service.

- **Microsoft Windows Server 2008 Hyper-V**

  Afterwords implementing in Windows Server 2012 and now available on Windows Server 2022, along with XenServer and vSphere, Hyper-V is one of the top 3 Tier-1 hypervisors. First released with Windows Server 2008, Hyper-V has now been greatly enhanced with Windows Server 2022 Hyper-V which offers:

  Live migration;
  Storage migration;
  VM Replication;
  Dynamic memory;
  Extensible virtual switch;
  High availability;

28 *SPICE (the Simple Protocol for Independent Computing Environments) is a remote-display system built for virtual environments which allows users to view a computing "desktop" environment – not only on its computer-server machine, but also from anywhere on the Internet – using a wide variety of machine architectures.*

Scale up to 320 logical processors, 4TB of memory, 2,048 virtual CPUs per host, 64 vCPUs per VM, 1TB of memory per VM, and 64 nodes / 8000 VMs per cluster.

- **VMware vSphere / ESXi**

The leader in the Tier-1 hypervisors is VMware with their vSphere/ESXi product. VMware led the market in developing innovative features such as memory overcommitment[29], vMotion, Storage vMotion, Fault Tolerance, and more. Previously, VMware called their free hypervisor “Free ESXi” as ESXi Server is what is loaded directly on the physical server. However, VMware calls the “suite” of features “vSphere”, available in various editions. Today, even the free hypervisor is called “The VMware vSphere Hypervisor“. While the free vSphere hypervisor does have a graphical interface (the vSphere Client) and memory over-commitment, it doesn’t offer features like vMotion, storage vMotion, high availability, or centralized management. The free version also has the limitation of supporting up to 32GB of RAM per physical server. The commercial versions of vSphere include features like:

Memory over commitment;
High availability (called vSphere HA);
vMotion;
Storage vMotion (svMotion);
vSphere Data Protection (for backup and recovery);
vSphere Replication;
vShield Endpoint protection (the option to use agentless anti-virus solutions);
Hot add of memory and hot plug for CPU;
Fault tolerance (FT) for availability;
Distributed resource scheduler (DRS) for VM “load balancing”;
Distributed virtual switch (dvSwitch);

29 ***Memory overcommitment*** *is a concept in computing that covers the assignment of more memory to virtual computing devices (or processes) than the physical machine they are hosted, or running on, actually has. This is possible because virtual machines (or processes) do not necessarily use as much memory at any one point as they are assigned, creating a buffer. If four virtual machines each have 1 GB of memory on a physical machine with 4 GB of memory, but those virtual machines are only using 500 MB, it is possible to create additional virtual machines that take advantage of the 500 MB each existing machine is leaving free. Memory swapping is then used to handle spikes in memory usage. The disadvantage of this approach is that memory swap files are slower to read from than 'actual' memory, which can lead to performance drops.*

Storage I/O control (SIOC) and network I/O control (NIOC);
Host profiles;
Autodeploy;
Storage distributed resource scheduler (SDRS);
Single root I/O virtualization (SR-IOV);
vCenter Single Sign On (SSO);
Scale up to 512 VMs per host, up to 2048 vCPUs per host, up to 64 vCPUS and 1TB of vRAM per VM;

1.6 Security vulnerabilities in virtualized environments

The majority of safety flaw that can be found in a virtual machine are basically the same as the security defects related with any actual framework or personal computer. Truth be told, assuming a specific working framework or application configuration, if it is proven to be insecure when run on an actual equipment, it will undoubtedly will be weak while running in a virtualized climate. The utilization of virtualized frameworks adds some security concerns, including:

- **Guest operating system isolation**: Isolation is one of the essential advantages that virtualization brings along. Guest operating system isolation guarantees that software executing within a virtual machine may just access and use the resources allocated to it, and not secretly interact with programs or data either in other virtual machines or in the hypervisor. If not properly configured and maintained, isolation can also represent a risk for the entire virtual environment. The isolation level should be sufficiently strong to stop potential break-ins into a compromised VM and to prevent it from gaining access either to the other virtual machines in the same environment or to the underlying host machine. For instance, while shared memory areas is a useful feature that allows data to be transferred between VMs and the host, it can also be exploited as a gateway for transferring data between cooperating mischievous program in VMs.[30]

Besides, there are virtualization technologies that do not achieve isolation between the host and the VMs deliberately. Those are intended to support applications specifically designed for one operating system to be executed into another operating system. The absence of isolation can be very unsafe to the host operating system since it potentially grant limitless access to the host's assets, just to mention, the filesystem and networking interfaces.

- **VM Escape**: The execution of a hypervisor might weaken the entire security level of a virtualized environment if a vulnerability already exists, which an offender can exploit. Those kind of vulnerabilities might give a software running in a guest operating system an entry point to the host machine or, worse, to the hypervisor itself, allowing the execution of malicious code, and even other guest operating system resources running on the very same physical machine. This is known as VM escape attack and is maybe one of the most dangerous risk to VM security. In addition, considering that the host machine is running with root privileges, the exploit which get access to the host machine also gets the root privileges. This outcome in a total break down for what security is concerned and a compromised system.

- **Mobility:** Virtual machines are intrinsically not physical, and that implies their theft can occur without actual robbery of the host machine. What's inside of the virtual disks of every virtual machine is saved as a file by the majority of hypervisors, which permits virtual machines to be replicated and executed in different physical computers. Whilst that is a advantageous factor, on the other hand it can lead to a security risk.[31] Attackers can steal the virtual machine trough the network or copying to a portable storage media and then revealed the data inside of it employing their own infrastructure without physically stealing anything. Once hackers have fully control of the stolen data, they have limitless chances to exploit all security restraints, for example password crackers, offline intrusion, reverse software engineering,

30 *William Stallings and Lawrie Brown. "Computer Security: Principles and Practice." Pearson, 3rd edition, 2014.*
31 T. Garfinkel, M. Rosenblum, "When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments," USENIX Association, 2005

and considering the attacker has a compromised copy of the virtual machine and not the original one, no records of intrusion will be logged.

- **Virtual machine monitoring**: The physical host machine is single point of control for the entire virtualized environment, and there are implementations that give access the host in order to monitor the execution, to supervise the guest virtual machines and to communicate with the software running inside the VMs. When designing the virtual machine architecture, extreme care should be taken so that the isolation level is sufficiently robust to avoid the host from being an entry point for any intrusion in the virtual machines.

- **Guest-to-Guest attack**:[32] In the event the hypervisor or the host machine are attacked by an external intruder who gains root privileges, all other virtual machines could be compromised. This kind of attack, also known as “hyperjacking” is extremely sneaky as traditional security measures aren’t effective as the guest operating system cannot be aware that the computer has been compromised. In this case the hypervisor represents a single point of failure when it comes to the security and protection of sensitive information.[33]

- **VM monitoring from another VM**: It is considered as a danger when one virtual machine with no difficulty might be permitted to monitor resources of another virtual machine. At the point that comes to the network traffic, complete isolation completely relies upon the network security of the virtualized environment. Assuming the host machine is linked to the guest machine using exclusive network connections, then it is not likely that packets could be sniffed by the guest machine to the host and viceversa. Anyway is to consider that the vast majority of virtualized environments are connected to the host machines and by means of a virtual hub or by a virtual switch. In that case, a guest machines might be capable of sniffing packets over the network or even worse redirect the packets going to and coming from another guest as in a man-in-the-middle like attack.

32 *"HYPERJACKING". Telelink. Archived from the original on 27 February 2015. Retrieved 27 February 2015.*

33 *Ryan, Sherstobitoff. "Virtualization Security". Virtualization Journal.*

- **Denial-of-Service:** Since guest machines and the underlying host share the physical resources such as CPU, memory disk, and network resource, it is at a guest's hands to unleash a Denial-of-Service (DoS) attack to other guests settled in the same environment. DoS attacks consist of an threatening device that makes another machine or group of machine to run out all the available resources, denying accessing the services to other users or processes.

- **Compromised VM snapshots**: Virtualization software frequently offers help for suspending an executing guest operating system in a snapshot, saving that image, and then restarting execution at a later time, possibly even on another system. While this is helpful in distributed contexts, it can also be extremely dangerous. If an attacker succeeds in inspecting or modifying the snapshot image, it can also compromise the security of the guest operating system, along with the data and softwares running on top of it.

1.7 High availability and fault tolerance

Fault tolerance is the property that allows a system to continue to function accurately in case of an event of failure of one or more of its parts. If performance drop off, the degradation is proportional to the gravity of the failure, in comparison to a bad planned architecture, in which also the lowest failure could end in a whole breakdown. Fault tolerant systems are requested while essential in high-available environment or life-critical systems. The capacity of a system of keeping up its functionalities despite portions are affected or non functional is called as graceful degradation.[34]

A fault-tolerant paradigm make possible for a system to remain stable and to go on with its expected operations, perhaps at a decreased degree, instead of failing entirely, when some part of the system fails. The term is most commonly used to describe computer systems designed to continue more or less fully operational with a

34 *Daniel P. Siewiorek; C. Gordon Bell; Allen Newell (1982). Computer Structures: Principles and Examples. McGraw-Hill.*

reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software. Fault tolerance can be accomplished by anticipating extraordinary events and engineer the system to adapt to them, and, generally speaking, adapting for self-stabilization[35] so that the system restores towards an error-free state. Anyhow, if the outcomes of a system failure are irreversible, or the cost of making it sufficiently reliable is very high, a better solution could be to duplicate the resources or perform a roll-over task to fix up to a safe mode. This is similar to an operating system roll-back recovery but it can be automatically implemented or manually done, if a human intervention is required. [36]

1.8 Virtualization benefits and drawbacks

Virtualization happens when a virtual version of a piece of hardware of software is made-up in place of an actual one. With state-of-the-art virtualization, that can comprehend storage devices, networks, operating systems, or even servers. Since 50 years ago, virtualization has extended into pretty much every type of digital settings. Including virtual machines that behave like a real computer, to console emulation, many IT tech, enterprises and organizations take advantage of what virtualization can bring.

Like most technologies, there are benefits and drawback that must be taken into consideration before implementing a virtualized framework.

- **Cheapness:** since virtualization doesn't need real equipment components to be utilized or installed, IT organizations consider it a less expensive system to be implemented. Dedicated large areas of space and huge investments to create an on-premise datacenter are no more needed. It is sufficient to buy the license or the access from a third-party supplier and start working, exactly as if the hardware was locally installed.

35 *Dijkstra, Edsger W. (1974), "Self-stabilizing systems in spite of distributed control", Communications of the ACM, 17 (11): 643–644, doi:10.1145/361179.361202.*
36 *C. Bressoud and F. B. Schneider, "Hypervisor-based Fault Tolerance," in 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, USA, Dec. 1995.*

- **Costs uniformity:** because third-party providers typically equip with virtualization options their offer, individuals and corporations can have predictable costs for their information technology infrastructures. Those options typically come in form of monthly, annual or pay as you go subscriptions.
- **Workload reduction:** most virtualization providers automatically modernize their hardware and software that will be leased. Instead of employing technicians to do these updates locally, they are installed by the third-party companies. This allows local IT professionals to focus on other tasks and save money for different tasks.
- **Improved uptime:** on accounts of virtualization technologies, uptime has improved significantly. A few suppliers offer an uptime that is nearly 100%. Also low-budget providers offer uptime at 99.99% nowadays, and that’s hardly comparable with the uptime a single company can attempt.
- **Faster resources deployment:** even the planning of physical machines setup is no longer needed because provisioning is fast, simple and reliable when virtualization is used. Also establishing local networks or installing other technology components isn’t necessary anymore as long as there is at least one entry point to the local virtual environment and that can be connected to the rest of the organization.
- **Digital businesship:** before virtualization took hold on a large scale, digital entrepreneurship was essentially beyond the realm of possibilities for average people. Thanks to the numerous platforms, servers, and storage devices that are accessible today, nearly everybody can start their own business or internet activity. This technology shift made possible new business otherwise impossible.
- **Energy savings:** for most actors in the market, virtualization represents an energy-efficient system, because there aren’t actual hardware being running, energy consumption rates can be lowered or set to zero. Instead of paying for the air conditioning costs of a data center and the operational costs of hardware keeping, money can be used for other operational financing over time, improving the business and at the end the return on investments.

- **Implementation costs:** the average cost for business when virtualization is considered to be implemented on premises is quite considerable. For providers of virtualization environment, however, the implementation costs, even though extremely high at the beginning, are cut out among many clients.
- **Limitations:** few out of all software or server will work inside a virtualized environment. That implies an individual or organization might require a hybrid system to work appropriately. This actually saves time and cash in the long term, however since few vendors support virtualization and some of them might quit supporting it after a starting period of time, a grade of uncertainty is always in the way when it comes to fully carry out this type of architecture.
- **Security threats:** information is considered to be the most valuable currency nowadays. Who is able to control information and extract knowledge from the data can make money out of it. Because business consider information vital, it is often targeted. The average cost of a data security breach in 2021, according to IBM Security Group, was $ 4.24 million. A great effort has been spent to secure IT infrastructures and much more will be spent in the future, because chances of experiencing a data breach in virtualized environments is not negligible.[37]
- **Availability concerns:** one of the first consideration that might come to mind when outsourcing to a third-party virtualized environment is the information accessibility. On the off chance that an organization cannot get trough an internet connection to their virtual machines, so their assets happens to be unavailable, resulting in a loss of money as the connection is not on client’s hands.

The benefits and drawbacks of choosing to go virtualized we have seen so far demonstrate that, when used properly, are worth it.

37 IBM Security Group, “Cost of a data breach report 2021”, July 2021

Chapter 2

Containerization

In this chapter we will focus on containerization, starting from the differences with virtualization to then analyzing the techniques adopted to reach the goal of isolation the environment where containers run. We will look into security, high availability and fault tolerance in containerized contexts.

2.1 Differences between virtualization and containerization

Containers are a lightweight, more rapid way of approaching virtualization as they do not depend on a hypervisor, quicker resource provisioning and speedier accessibility of new software can be achieved.

In place of turning on a whole virtual machine, containerization bundles together everything expected to run a single application or a microservice (along with runtime libraries they need to run). The container incorporates all the code, its dependencies and even the operating system itself. That makes the applications able to run roughly anywhere a desktop PC, a traditional IT infrastructure or the cloud.

Containers utilize a type of operating system virtualization. In simple words, they benefit of features of the host operating system to isolate processes and control the way CPUs, memory and disk space are accessed.

It has been roughly 20 years now that containers have been used, but it is commonly recognized that the modern container era began in 2013 with the introduction of Docker, an open source ecosystem for building, deploying and managing containerized applications.[38]

In conventional virtualization, a hypervisor virtualizes actual equipment components, resulting in virtual machines each containing a guest operating system, a virtual copy of the hardware that the guest operating system requires to run, the software and its related libraries and dependencies. Virtual machines with different operating systems can be executed on the very same physical server. For example, a VMware VM can run next to a Linux VM, which runs next to a Microsoft VM, and so forth.

Rather than virtualizing the underlying hardware, a container virtualizes the operating system (normally Linux or Windows) so each single container only hold the software and its libraries and dependencies. Containers are small, fast, and portable because, not alike a virtual machine, containers don’t have to incorporate a guest operating system in every instance and can, alternatively, just pull the components and features the host operating system offer.

38 See https://www.docker.com/

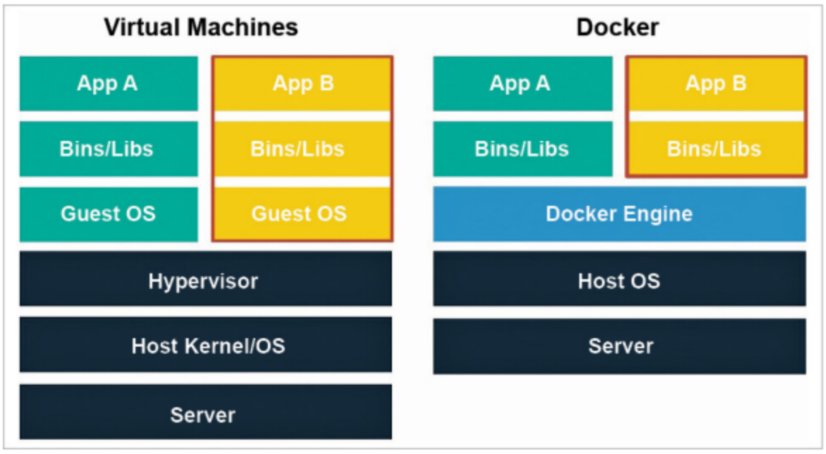
Like virtual machines, containers give developers the ability to boost CPU and RAM utilization of physical computers. Thanks to containers, though, microservice architecture can be pushed even further because they also enable such paradigm where software components can be put in production and dimentioned in a more precise size, making the process more appealing to coders and IT staff, instead of having to rebuild a whole monolithic software just because of a tiny section of the code has changed.

Providing an isolated environment, inside the hosting operating system, is commonly known as operating-system level virtualization and such an isolated environment can be named as container: "A container is a self contained execution environment that shares the kernel of the host system and which is, optionally, isolated from other containers in the same system"[39]. Virtual machines create a new instance of an operating system for every virtual machine running and that gives several benefits such as the ability to run entirely different guest operating systems. Compared to host it also comes with many drawbacks. The first aspect that comes to mind is the virtual machine execution overhead, caused either by the virtual machine monitor's instruction patching and translation, the paravirtualization drivers needed to virtualize the actual hardware or, in case of hardware assisted-virtualization, the source of the overhead is the CPU context switching. In addition to that, the virtual machines take much more disc space and are more demanding to maintain.
If compared to virtual machines, the additional overhead introduced by the hypervisor and the operating system that containers take away. Containers only require the application and its dependencies, while the kernel is shared among them. Considering that the operating system is already running on the host machine, starting a container tends to be much quicker than spinning up a virtual machine. Shared kernel may not always be a benefit though, as regarding as running Windows applications in containers on Linux is not possible.

39 *"OS-level virtualization with Linux containers: process isolation mechanisms and performance analysis of last generation container runtimes" , G. Minì and A. Giorgio, 2020*

2.2 Where’s the hypervisor?

The concept of containerization was originally developed to isolate namespaces in a Linux operating system for security purposes. LXC (Linux Containers) was the first, most complete implementation of Linux container manager. LXC provides operating system-level virtualization through a virtual environment that has its own process and network space, instead of creating a fully fledged virtual machine. LXC containers faced some security threats. At the platform service level, packaging and application management is an additional requirement. Containers can match these requirements, but a more in-depth elicitation of specific concerns is needed.[40]



Container virtualization is done at the operating system level, rather than the hardware level. Each container (as a guest operating system) shares the same kernel of the base system.[41] As each container is sitting on top of the same kernel, and sharing most of the base operating system, containers are much smaller and lightweight compared to a virtualized guest operating system. As they are lightweight, an operating system can have many containers running ahead of it, compared to the limited number of guest operating systems that could be run. Although the hy-

40 *D. Bernstein. “Containers and Cloud: From LXC to Docker, to Kubernetes”, IEEE Cloud Computing, volume 1, no. 3, pp. 81-84, 2014.*

41 *Carlos Arango, Remy Dernat, John Sanabria. “Performance Evaluation of Container-based Virtualization for High Performance Computing Environments”, arXiv:1709.10140v1 [cs.os], 28 September 2017.*

pervisor-based approach to virtualization does provide a complete isolation for the applications, it has a huge overhead: overhead of resources allocation, the overhead of managing the size of a virtual machine, just to mention a few. On the other hand, sharing between guest operating systems in a virtualized environment, which is very similar to sharing between independent systems, because virtualized hosts are not aware of each other, and the only method of sharing is via shared file system. The basic principle in container based virtualization is that, without virtual hardware emulation, containers can provide a separated environment, similar to virtualization, where every container can run their own operating system by sharing the same kernel. Each container has its own network stack, file system, etc.[42]

So the shift to containerization has been made conceivable employing a technology first launched around 2000 in Linux; cgroups and namespaces. Control groups (abbreviated *cgroups*) is a Linux kernel innovation that restraints, accounts for, and isolates the resource utilization (CPU, memory, disk I/O, network, etc.) of a group of processes.
Engineers at Google, above all Paul Menage and Rohit Seth, initiated the study on this feature in 2006 under the name "process containers". It was only in 2007, the naming convention turned into "control groups" to avoid confusion caused by multiple meanings of the term "container" in the Linux kernel context, and the control groups functionality was merged into the Linux kernel mainline in kernel version 2.6.24, which was released in January 2008[43]. Since then, developers have added many new features and controllers, such as support for kernfs in 2014[44], firewalling, and unified hierarchy. cgroup version2 was merged in Linux kernel 4.5 with significant changes to the interface and internal functionality.[45]

Cgroups was originally written by Paul Menage and Rohit Seth, and mainlined into the Linux kernel in 2007. Afterwards this is called cgroups version 1. Development and maintenance of cgroups was then taken over by Tejun Heo who rewrote cgroups. This rewrite is now called version 2 and first appeared in Linux kernel 4.5 released on 14 March 2016.

42 *Pete Brey. “Containers vs. Virtual Machines (VM’s): What’s the Difference?”, NetApp Blog https://blog.NETapp.com/blogs/containers-vs-VM’s/*

43 *For a complete review of the linux kernel version history, see https://en.wikipedia.org/wiki/Linux_kernel_version_history*

44 *"cgroup: convert to kernfs". Linux kernel mailing list. 28 January 2014.*

45 *https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html*

Unlike version1, cgroup version2 has only a single process hierarchy and discriminates between processes, not threads. One of the design goals of cgroups is to provide a unified interface to many different use cases, from controlling single processes (by using nice, for example) to full operating system-level virtualization (as provided by OpenVZ, Linux-VServer or LXC, for example) providing:

- **Resource limitation:** it can be established that groups cannot top a predetermined memory threshold, just to mention a few the CPU % usage, the filesystem cache size, the I/O access to and from block devices such as physical drives (hard disks, SSDs, USBs, and so on), the precedence of network traffic for each network interface;

- **Priority**: a differentiation on CPU share/utilization or external memory I/O throughput can be set considering different groups;

- **Auditing**: it takes into account a group's resource utilization in order to keep track of the resource amount used by a certain group. That information can be used, for example, for payment schemas;

- **Checkpoint**: stopping the execution of particular groups of processes in case of failure to be put off.[46]

A control group is a batch of processes grouped together by the same criteria and restraint under certain limitations or policy. Those limitations are, just to mention a couple of them, related to CPU % usage or RAM availability and are under the direct control of the kernel that can group them together in a hierarchical way.[47]

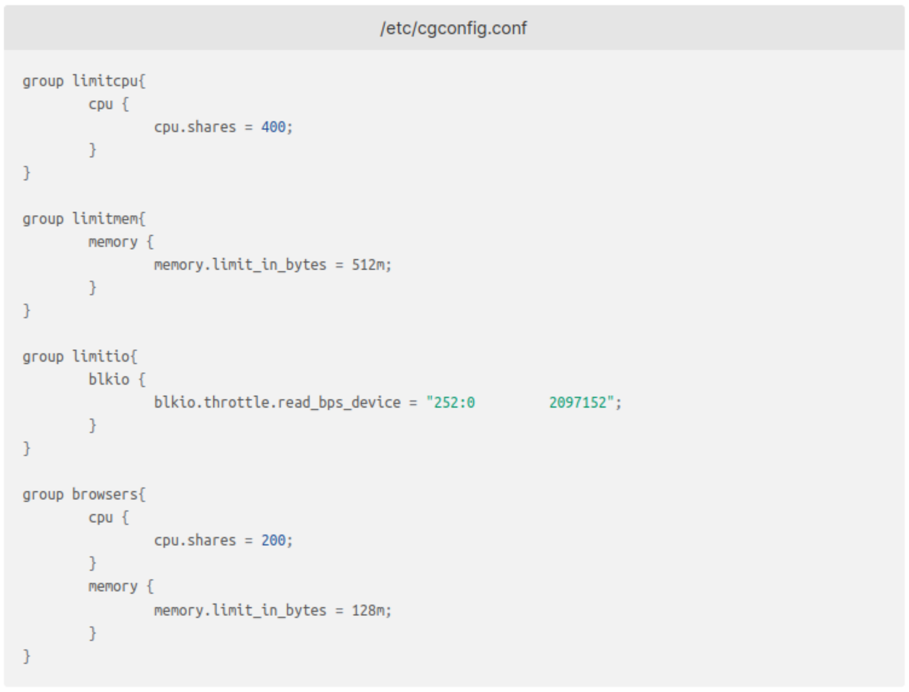
Control groups can be:

- accessed using the cgroup virtual file system manually;
- modified using on the fly tools like cgcreate, cgexec, and cgclassify (from libcgroup);

46 Bouteiller, B., Lemarinier, P., Krawezik, K., & Capello, F. (2003, December). Coordinated checkpoint versus message log for fault tolerant MPI. In Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on (pp. 242-250). IEEE.

47 See https://en.wikipedia.org/wiki/Cgroups

- configured via the "rules engine daemon" that can automatically move processes of certain users, groups, or commands to specific cgroups;
- managed by other software such as Docker, LXC, Fairjail, libvirt, systemd.



/etc/cgconfig.conf

```
group limitcpu{
        cpu {
                cpu.shares = 400;
        }
}

group limitmem{
        memory {
                memory.limit_in_bytes = 512m;
        }
}

group limitio{
        blkio {
                blkio.throttle.read_bps_device = "252:0         2097152";
        }
}

group browsers{
        cpu {
                cpu.shares = 200;
        }
        memory {
                memory.limit_in_bytes = 128m;
        }
}
```

In this example of the cgroup definition, contained in /etc/cgconfig.conf, the limitcpu limits the CPU share available to processes in this cgroup to 400. cpu.shares specifies the cputime available to the tasks in this group.

limitmem cgroup set a threshold of 512 MB of available memory.

In the browsers cgroup CPU shares is limited to 200 and available memory to 128 MB.

Namespaces are a keen innovation introduced on the Linux kernel that split resources among processes and that's a major component for containerization.[48]

The Linux Namespaces began in 2002 in the 2.4.19 kernel with work on the mount namespace kind. Additional namespaces were added beginning in 2006 and con-

48 See https://en.wikipedia.org/wiki/Linux_namespaces

tinuing into the future.[49] Adequate containers support functionality was finished in kernel version 3.8 with the introduction of User namespaces. [50]

2.3 What happens in Windows? The hidden Linux VM

Since a decade, Docker and containerization have been one of the most popular subject among programmers and companies and Windows Server 2016 announcement ignite even more the debate by revealing that containers will be added in Windows.[51] [52]

A couple of architectural concerns remains, one above all is the way system processes will be handled, as in Linux, containers often just execute a unique process that shares the resources with the operating system host and other containers. On the other hand in Windows part of the code was relocated externally of the kernel and executed in user mode and this poses a problem because of the need to update every system call to instruct them how to behave when more than one container is in execution. This result in a higher startup time when a new container instance is launched.

Even though VirtualBox, which was executed in DockerToolBox, has been abandoned, Windows is nowadays running a hidden Linux virtual machine to launch containers using Hyper-V to isolate the execution instead. That indicates that today there is a minor usage of resources to run a container in Windows but remains the fact that a virtual machine thus far needed to containerized applications.[53] [54]

49 *"Linux kernel source tree". kernel.org.*

50 *Heo, Tejun (2016-03-18). "[GIT PULL] cgroup namespace support for v4.6-rc1". lkml (Mailing list).*

51 *https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/april/containers-bringing-docker-to-windows-developers-with-windows-server-containers*

52 *“Rethinking the Library OS from the Top Down” , Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, Galen C. Hunt, 2016*

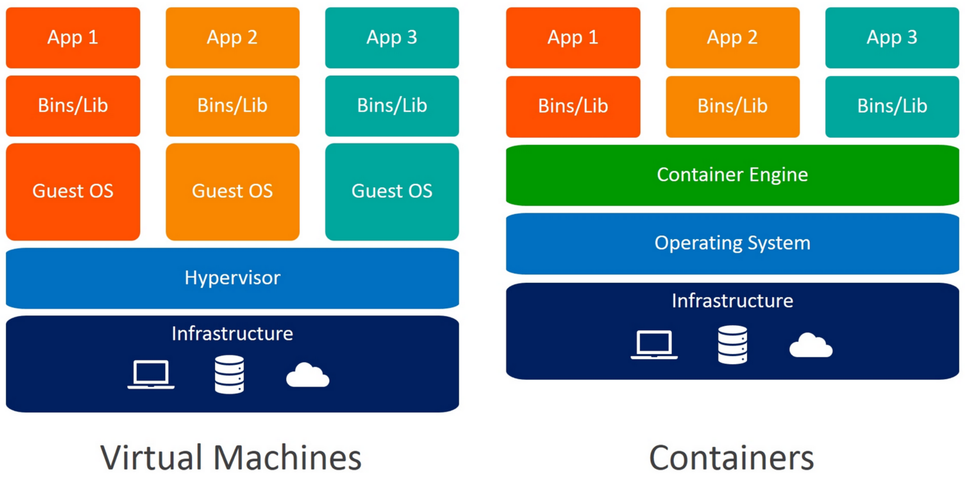
53 *Rzecki, Krzysztof & Niedźwiecki, Michał & Sośnicki, Tomasz & Andrzej, Martyna. (2014). Experimental Verification Of Hyper-V Performance Isolation Level. Computer Science.*

54 *https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012*

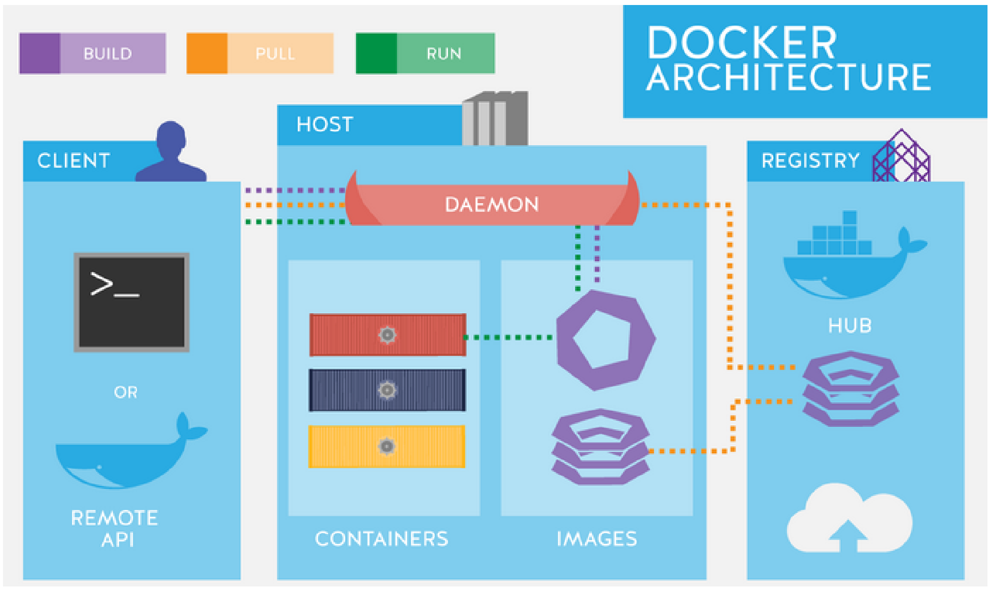
2.4 Docker architecture

Containers / Images / Repos / UnionFS

Docker is a PC software that enables operating system-level virtualization otherwise called containerization. It is maintained by Docker, Inc. and is principally developed for Linux, where it takes advantage of the resource isolation mechanisms offered by the Linux kernel for example cgroups and kernel namespaces, and others to permit self-sufficient ”containers” to be executed within a single Linux instance, so that completely bypassing the overhead due to start phase and the cost of running an entire virtual machine.
Docker makes it easy to deploy and port self-sufficient containers that can be executed both on the cloud or on-premise. A schematization of virtual machines and containers can be seen in the figure below, where is highlighted the absence of the hypervisor layer in containerized environments, thus the execution result faster.



The usage of namespaces and cgroups to completely isolate an application from the underlying operating environment, including process trees, user IDs, and file systems is the key feature of containerization. Docker containers are layered using basic images and non-writable updates. A Docker image can include just the operating system functions and dependencies or it can store and entire prebuilt software stack ready for the execution. During the building phase of an image each command execution, such as apt-get install <package>, is added as a new layer on top of the previous ones. As shown below, the Docker architecture combine a variety of components.[55]



The Docker architecture makes it possible to achieve important performance gain over virtual machines.[56] and the automation obtainable in production environment could be schematize as follow:[57]

55 *Charles Anderson. Docker. THE IEEE COMPUTER SOCIETY, 2015 IEEE , 0740-7459/15/*

56 *Claus Pahl , Brian Lee. Containers and Clusters for Edge Cloud Architectures a Technology Review. 2014*

57 *D. Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, 2014.*

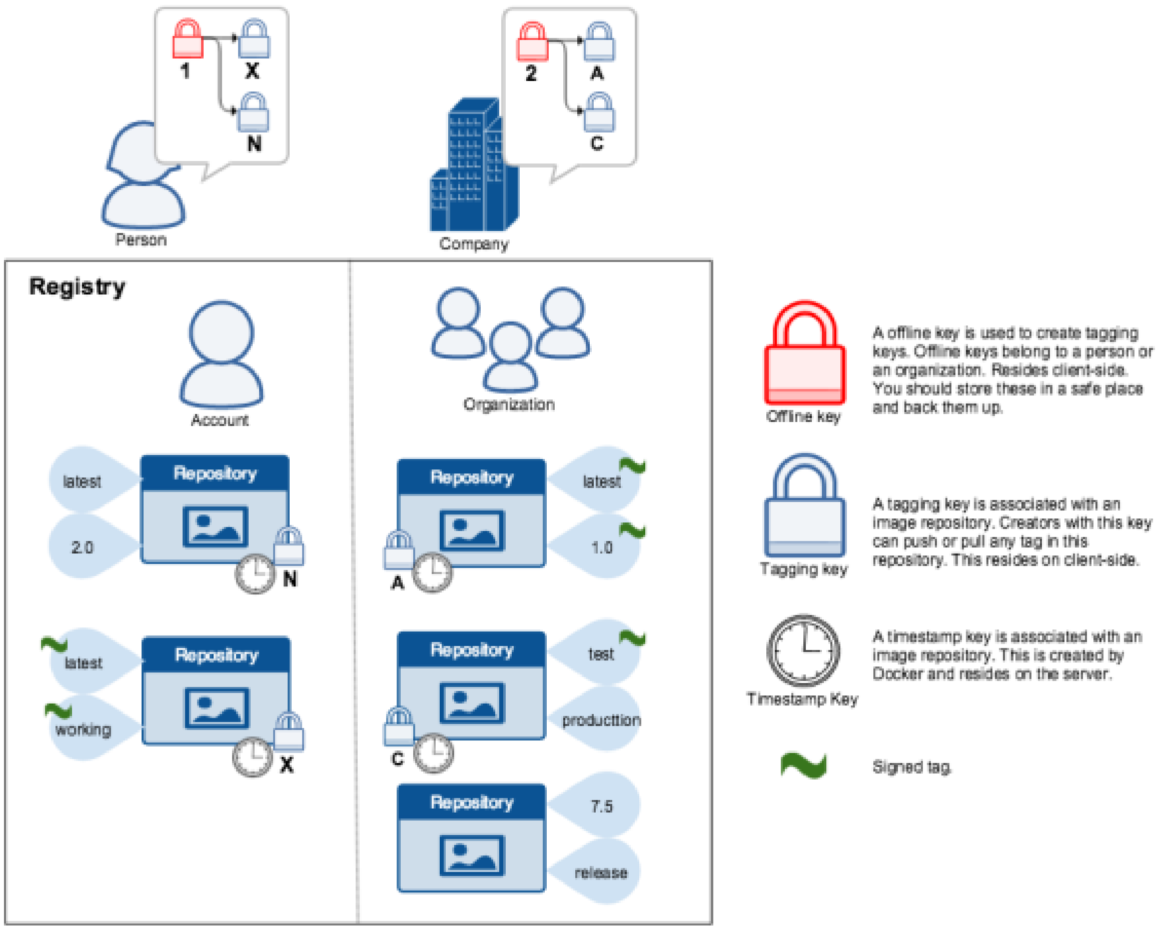
	VIRTUAL MACHINES	CONTAINERS
Guest Operating System	Each VM runs on top of hypervisor and kernel loaded into its own memory region.	All guests share the same kernel. Kernel image is loaded in its physical memory.
Performance Efficiency	Suffers light overhead as the machine instructions get translated from guest to host OS.	Almost native performance as compared to the underlying host OS.
Security	Complete Isolation.	Isolation using namespaces.
Storage	Takes more storage.	Take less storage as the base OS is shared.
Isolation	Higher level of Isolation. Need special techniques for file sharing.	Subdirectories can be transparently mounted and can be share.
Networking	Can be linked to virtual or physical switches. Hypervisor has its own buffers to improve I/O performance.	Leverage standards like IPC mechanisms such as signals, pipes, sockets, etc. Advanced features like NIC are not available.
Bootup Time	Take a few minutes to boot.	Boot up in a few seconds.

Using a client/server paradigm, the server only accepts data from a Unix socket, isolating the execution from other processes and even from outside the network. This is a security enhancement as if someone will take control of the daemon will be able to control the host. A simple attack to a running container will result in a breakthrough.

- **CONTAINERS**
  A container is a lightweight package, portable and selfsufficient which can be executed in cloud, public or private, or locally. Containers are, essentially, a collection of data, structures and files an application need to be executed: libraries, dependencies, more exec, filesystem portion, configuration files, scripts and so on.

- **IMAGES**
  Images are the building blocks of Docker. They are built layer by layer using union file systems. Similarly to what happens in object-oriented programming languages, an image is like a class object, while a container is an instantiation of that image.

- **DOCKER REGISTRY**
  This is a repository for Docker images. Figure illustrates a Docker registry ecosystem. As shown, all the services are packed in a container. Application containers are in a way similar to microservices where only the necessary features are packed as one to deploy the feature, in this way every image will result in a very light package if compared to an entire virtual machine.



A offline key is used to create tagging keys. Offline keys belong to a person or an organization. Resides client-side. You should store these in a safe place and back them up.

A tagging key is associated with an image repository. Creators with this key can push or pull any tag in this repository. This resides on client-side.

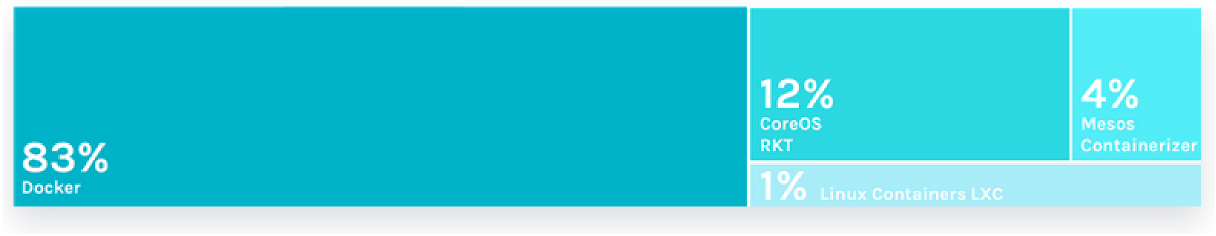
A timestamp key is associated with an image repository. This is created by Docker and resides on the server.

Signed tag.

- **UNION FS**
  Once a container is executed, Docker mount a layered filesystems using UnionFS to aggregate different mounting points and presenting them as unique filesystem. When a docker image is built up it could run internally different services. For example a voting application could be structured as different layers: a database server to store the information, a web server to give the users the interface and a PHP interpreter. Every layer came within its own filesystem and the resulting one is a collection of various branches all packed together.[58]
  Data volumes do not follow those rules as they are mounted from the host filesystem and, unlike images that are non-persistent, a volume is.[59]

2.5 Other container engines

Although Docker still made up 83% of containers in 2018, that number went down from 99% in 2017, and different container runtime environments are on the raise, such as CoreOS, Mesos, LXC just to cite a few and the availability of different options is growing.[60]



Docker is gaining a lot of consideration among developer's community but it isn't the unique container option. Some options are illustrated here after just to exemplify the distinctive , discovering their differentiating features, advantages and even weaknesses.

58 *V. Jurenka, "Virtualization using Docker Platform", 2015*
59 *R. Dhar, "Comparative evaluation of Virtual Environments: Virtual Machines and Containers", 2016*
60 *https://sysdig.com/blog/2018-docker-usage-report/*

- **CoreOS rkt**

  Pronounced “Rocket”, it runs both Docker and appc images and it works out of the box with Kubernetes. Even though it support Trusted Platform Modules (TPMs) compared to Docker, it doesn’t have much third-party developments. Unfortunately, major drawbacks including the absence of OCI[61] standard adoption and the fact that it is still under development, doesn’t give rkt much visibility.

- **Mesos Containerizer**

  Developed by Apache in 2008, Mesos stands out for good performance and for supporting both Docker and appc images.[62]

  The integration with Spark and Flink frameworks for big data software makes of Mesos a starting point for such environment, but the worst limitation is that a user cannot execute containers independently but needs to run the entire Mesos framework.

- **LXC Linux Containers**

  While not that popular as Docker, Linux containers are gaining attention due to the ease of use they offer and have now become an important part of IT security. The LXC container platform is often used to isolate different processes from each other and from the rest of the system.

  A crucial disadvantage of LXC is related to the memory management because, although several memory backends are supported (Ivm, overlayfs, zfs and btrfs) the storage method rely, by default, on the Rootfs, and other container platforms offer a better and more flexible solution for both container storage and image management.

61 *Open Container Initiative - https://opencontainers.org/*
62 *See https://iamlinops.blogspot.com/2019/08/docker.html*

- **OpenVZ**

  OpenVZ has been available to users since July 2016 as a standalone Linux distribution and expansion of the Linux kernel. The software itself is based on a Red Hat Enterprise Linux 7 (RHEL) and allows guest operating systems both like virtual machines or containers, needing a lower memory usage if compared to other container frameworks.[63]

- **containerd**

  Described as "an industry-standard container runtime with an emphasis on simplicity, robustness and portability",[64] containerd is an early-stage project of the Cloud Native Computing Foundation with supporting for OCI images[65]. As previously part of Docker project, it then was unlinked from the container runtime environment to give birth to this new project. This includes Docker functionality for running containers, managing low-level storage and managing images.

2.6 Security in containerized environments

Root privileges are mandatory to execute Docker containers, for that reason, if a container gets exposed, the entire host execution will be jeopardized. For what images are concerned, a cryptographic signature is calculated and added in order the image to be verified before execution.

Another likely threat to container based virtualization rely on the implementation of the Linux namespaces. As the host operating system uses namespaces to limit the access to certain data structures, a bug on that could result in an unauthorized access that could potentially give permissions to filesystem entries. Looking at the various vulnerabilities detected in the last years, and even if it seems practicable

63 *See https://openvz.org/*
64 *See https://containerd.io/*
65 *Open Container Initiative - https://opencontainers.org/*

controlling privilege authorization, the chance of new unknown exploits is always behind the corner.[66]

- **Network Security**: isolation is key to prevent network-based attacks, such as Man-in-the-Middle (MitM) and ARP spoofing/poisoning. Docker image repositories manages the distribution of image layers to Docker clients. The link to the registry is secured using TLS and an hash key is generated to check the image integrity. Each container expose its own IP address and has IP routing tables, network devices, granting containers the permission to exchange data to each others via their network interfaces, and that is exactly like connecting with external hosts. The interface is then associated to a bridged network and Docker is responsible of the creation and management of the virtual Ethernet bridge in the host machine that forwards packets between them. Giving that, usual best practice for network security should be arranged even for a containerized environment.

- **Denial-of-Service (DoS):** one of the attacks could exploit the lack of resource management ending in a DoS for which a process try and tackle the system in order to drain all of the system resources. Preventing that is possible using resource limitation mechanism offered by cgroups.

2.7 High availability and fault tolerance issues

Despite the benefits of containerization, there are a couple of aspects relating to high availability and fault tolerance that must be taken into consideration.

Instance redundancy: one of the benefits of deploying images like Docker is that a user can specify a number of replicas to be executed and the container management system will take care of it, without the user intervention, trying to match the

66 *Bui, Thanh. (2015). Analysis of Docker Security.*

desired number of instances the user asked.[67] [68] [69] And that is true even for single physical computers and for clusters.

Resource redundancy: as commonly different services have different resource requirements, one of the strategies used to balance the load is automatic mixing services that have different resource allocation profiles so that the system always try to accomplish the best resources utilization degree.

67 *W. Li and A. Kanso, "Comparing Containers versus Virtual Machines for Achieving High Availability," in IEEE International Conference on Cloud Engineering, Tempe, AZ, Mar. 2015, pp. 353–358.*
68 *W. Li, A. Kanso, and A. Gherbi, "Leveraging Linux Containers to Achieve High Availability for Cloud Services," in IEEE International Conference on Cloud Engineering, Mar. 2015*
69 *W. Liu, "High Availability of Network Service on Docker Container," in 5th International Conference on Measurement, Instrumentation and Automation, Nov. 2016.*

Chapter 3

Orchestration

In this chapter we will deal with Kubernetes which is an orchestration tool born to help IT organizations managing containers. In fact after a period of explosion in the use of containers, a need for a comprehensive management tool was felt. That's where Kubernetes come in handy.

We will look over the change occurred starting from classical devop infrastructure (a single machine with a single operating system) to virtualization where more virtual machines run over a single server to a containerized environment, for which orchestration tools like Kubernetes are becoming essential.

3.1 Brief introduction

The CNCF (that stands for Cloud Native Computing Foundation) is a Linux Foundation project that was born in early 2000 to help the improvement of container technology and organize the technology and its evolution.[70] CNCF as part of the nonprofit Linux Foundation[71].

Kubernetes was the first project of CNCF and the foundation part of the Kubernetes infrastructure. Kubernetes is a portable, extensible, open-source platform for controlling containerized applications aimed to help managing container images like Docker but not limited to that.

Using a traditional deployment, applications are directly run on physical infrastructure and one of the difficulties is to plan resource resources allocation otherwise ending in performance issues.

In virtualized environments, various virtual machines can be executed on a single physical machine where each of them is isolated from the rest and because of that the security level of each application is incremented. Another fact to consider is as those virtual machines are executed over the same hardware, better resource usage can be reached and also scalability is enhanced because when needed, is easy to provide more virtual resources, and is indeed quite simple to vertically scaling a virtual machine.

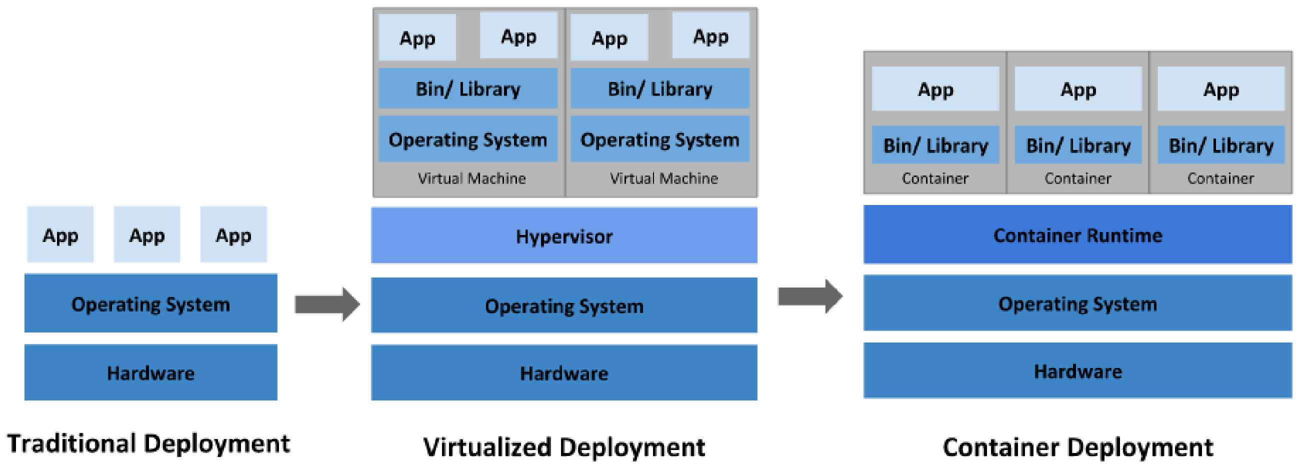
Containers are like virtual machines, with mild isolation properties permitting containers to be called lightweight and comparable to a virtual machine. As container are not linked to the physical infrastructure, portability across different operating system or cloud is achievable, and they come within other improvements: ease and efficiency of container image creation compared to virtual machines, continuous development, integration, and deployment, reliability, fast roll-backs, automatic separation of concerns (SoC)[72], decoupling applications from the underlying infra-

70 *Cloud Native Computing Foundation. - https://www.cncf.io/*

71 *Linux Foundation. - https://www.linuxfoundation.org/*

72 ***separation of concerns (SoC)** is a design principle for separating a computer program into distinct sections. Each section addresses a separate concern, a set of information that affects the code of a computer program. A concern can be as general as "the details of the hardware for an application", or as specific as "the name of which class to instantiate". Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a*

structure, environmental stability across different platforms as the same behavior is expected in different situations, avoiding the "it worked on my computer" problem, portability, distributed and flexible micro-services and resources isolation.[73]



Containers are a great solution to bundle and runs applications, considering a production environment there are many other aspects that must be taken into consideration, such as managing downtime for example, in such case if a container fails another one replace it, using a deployment strategy such as green blue deploy, canary, etc. Kubernetes is a framework to run distributed systems and it takes care of scaling and failover for applications providing deployment patterns.

Blue-green deployment is a software release pattern that gradually switch from an older version of an application or microservice in production to a new one. The first version is called the blue environment, while the new version is called the green environment. After the migration has been completed, the old version is no more accessible, but still in place for possible rollbacks.

well-defined interface. Encapsulation is a means of information hiding. Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).

73 *See https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

In software engineering, canary deployment offers a way of publishing periodic releases. A software update is branched out to a narrow group of the customers at the beginning, so they can evaluate it and give back a feedback. When the corrections have been tested, the new software features are distributed to the remaining users. Canary deployments can demonstrate the way users work with software in a realistic environment. Compared to blue-green deployments, the canary approach provide no-downtime to upgrade the software and easy rollbacks in case of need. On the contrary, canary deployments are easier, and breakdowns are simpler to handle. A canary release could be seen as a beta version off a specific software function, giving the entire project a stable and non-stable versioning.

The main features provided by Kubernetes are:

- **Load balancing and service discovery,** Kubernetes can broadcast a container using the DNS name or using its own IP address. If a container generates a huge amount of data, Kubernetes is capable of load balancing and spread the network traffic so that the system as a whole result stable. Software in a Kubernetes cluster are deployed by using pods, the building block of this architecture, which is quite diverse compared to the traditional approach of deploying applications on target machines. Because during a POD lifecycle its IP address changes continuously, that makes unfeasible to deploy a software using a traditional approach. Software need to expose somehow a way external users or other applications can reach it. For example via API. For what PODs are concerned, as they could be spread in different network segments, a way to overcome this limitation is Kubernetes service discovery and load balancing which are provided as a builtin service.

- **Storage orchestration,** Kubernetes is able to use a storage system of pretty any kind, for example local disks, public/private cloud storage, NAS or SAN. Due to the fact that containers' lifespan could be very limited, the information saved within are canceled after a while, creating many complications for the kind of application we are used to handle with. Though, Kubernetes offers a method to preserve data and overcome this issue using stateful storage in a containerized environment. All phases of a container, not to

mention creation, automation, load balancing, management and even I/O interfaces. Kubernetes propose the idea of Persistent Volumes, those live autonomously with respect of containers, and they will be operational even after a container is restarted or powered down.[74]
Volumes could be both persistent or non persistent, Kubernetes offers containers to ask for dynamically-created storage, adopting a trigger named "volume claims".

- **Automated rollouts and rollbacks**, adopting Kubernetes, a manifest of whished services can be declared in order to be run and automatically managed by Kubernetes itself without any external intervention. Kubernetes will autonomously check the services and running components to guarantee the required standards of execution are met. For example, if a 3 replica of a specific container is published in the manifest, Kubernetes will run a strategy to maintain those 3 replicas in production even in case of a node fails.[75]

- **Self-adapting containers,** when Kubernetes cluster is created and nodes have been set to use a capped amount of resources, the orchestration mechanism will manage the best way to distribute or redistribute the containers among available PODs to maximize CPU usage.

- **Auto-healing**, in case a running container fails, Kubernetes re-executes it without a manual intervention. Cases of failure can be when a container does not respond to user-defined health check.[76] A valuable improvement introduced in Kubernetes is its auto-healing ability that meets the "desired state" functionality according to which the architecture will try to fit the requested standard of execution at its best, even if an entire node breaks or is no more reachable.

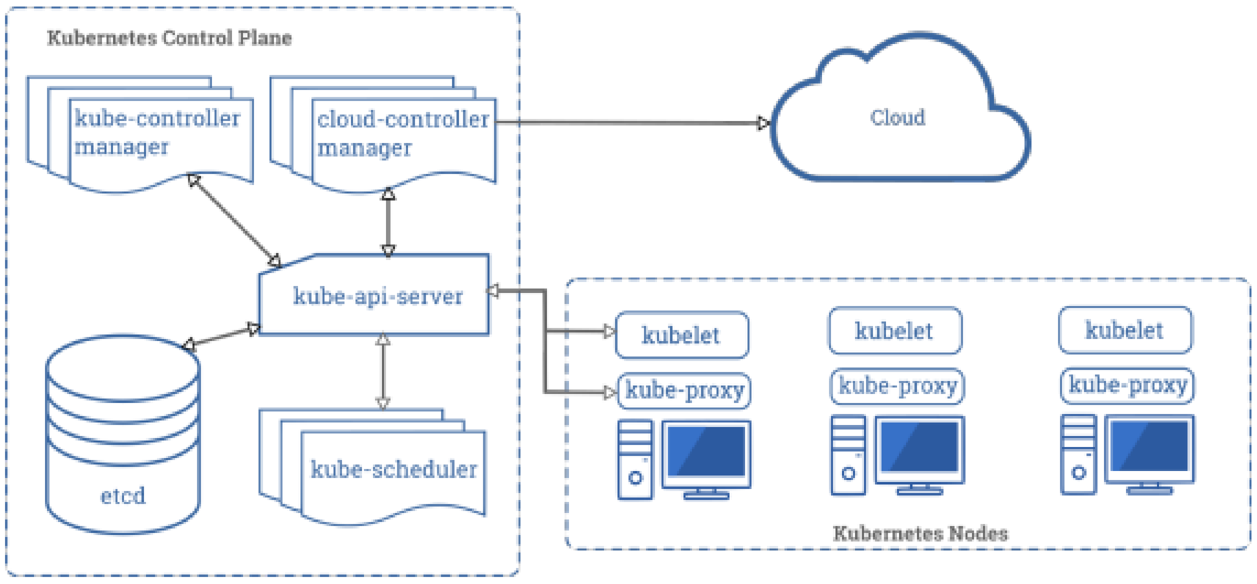
In the event a software or functionality is properly containerized and the Pod where the container is placed goes down, Kubernetes will reschedule it as soon as possible, as long as there are sufficient available resources. A second

74 *See https://cloud.netapp.com/blog/cvo-blg-kubernetes-storage-an-in-depth-look*
75 *See https://learnk8s.io/kubernetes-rollbacks*
76 The Kubernetes Book, Nigel Poulton and Pushcar Joglekar, 2019

stage of auto-healing mechanism rise up when a Kubernetes component fails. That includes kubelets, kube-proxies, and the container runtime. When one of these components breaks, the POD might be non responding, so Kubernetes will take care of the situation restarting the broke service. At last, if a physical machine itself is not responding, Kubernetes can spin up by itself a completely new POD in a completely new machine, if any available in the cluster, to restore the faulty situation.

3.2 Kubernetes architecture

When Kubernetes is deployed a cluster is fetched, a K8s (read Kubernetes) cluster consists of a set of workers machines, called nodes, that run containerized applications and various services. The control plane manages the nodes and all the running container in the cluster, in a production ready environment usually the control plane is run on multiple nodes providing fault tolerance and high availability.[77]



77 *See https://v1-20.docs.kubernetes.io/docs/concepts/overview/components/*

3.2.1 etcd

It is an highly-available key-value storing method used by Kubernetes to backup for every cluster information, etcd is an open-source, distributed and robust repository of key-value pairs used to handle distributed configurations, service discovery for clustered applications. The etcd database assures a safe automatic updates. Etcd stores an exact copy of the cluster state and for the sake of high availability and redundancy. Even though it could seem that a key-value database is something simple, since etcd is a crucial component of a Kubernetes cluster and etcd is a consensus-based distributed system, the cluster configuration can be quite complex.[78]

3.2.2 kube-scheduler

It checks for new Pods without designated execution node, and selects it for the Pod to be executed, according to the execution policy and resource needs.

3.2.3 kube-controller-manager

The Kubernetes controller manager has the entire control over Kubernetes that balances the state of the system. It can control the replicas, the termination of an execution. There are various type of controllers, like: Node controllers responsible for checking health status of the nodes, Replication controllers that keep the requested number of pods, Service Account and Token controllers, Node controller when a node has been deleted in the cloud after it stops responding, Route controller set up traffic routes on cloud infrastructure, Service controller for load balance.

78 *Lars Larsson, William Tärneberg, Cristian Klein, Erik Elmroth, Maria Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance".* 2020

3.2.4 kubelet

An agent based service that runs on each node in the cluster to ensure the specification requirements in term of resource utilization, container replicas, memory usage are met. Those specifications are defined in YAML or JSON format.

3.2.5 Container runtime

The container runtime is the software responsible for containers execution. Kubernetes accept Docker, containerd, CRI-O any implementation that meets CRI (Container Runtime Interface) criteria.

3.2.6 API Services

This service broadcasts the Kubernetes REST API permitting the nodes to connect with the master and the users to manage the cluster.

3.2.7 PODs

Pod is the smallest runnable unit that Kubernetes can handle and it represents a collection of containers having the same filesystem and networking resources in common. That means having the same namespace, cgroups and the same isolation context. A Pod in a Kubernetes cluster can:

- **run a single container**. The "one-container-per-Pod" schema is the most used in Kubernetes and it can be tought as a wrapper for a single container;

- **run multiple interconnect containers**. A Pod can envelop a software split in a collection of coordinated containers needing to access the same resources. And if a Node fails, a controller get a warning so it can take measure to restore a previous working situation.

- **deploy a StatefullSet:** a StatefulSet is thee application program interface adopted to handle stateful executions. These kind of pods are created the same way as other Pods but instead of being equivalent, due to the fact that there is a state to be maintained, every one has a unique identifier kept even in an event of rescheduling.

- **Jobs:** A Job is responsible for the creation of Pods and also their termination or rescheduling in case of failure. When a Pod end with a success exit state, is the Job that keeps track of the conclusion and if the initially started Pods have ended, the entire task is marked as completed.

- **CronJob:** are used to execute recurring tasks, as bulk email campaign or automatic warning messages, even backups.

- **Storage in Pods:** a pod doesn't have long-lasting storage so that if the pods fails or is re-executed even the data saved until re-execution is lost.

- **Networking in Pods**: every Pod is given a unique network identifier and each container within that Pod shares the network namespace. Containers can communicate with each other but when a message must be sent outside the Pod network, it needs to be addressed to the Services Kubernetes abstraction that manages all the network load among Pods.

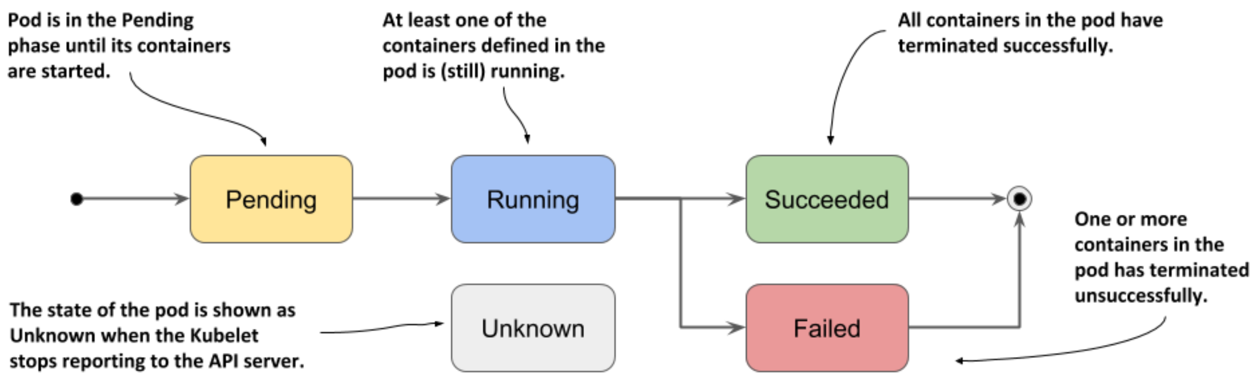
Pods Lifecycle: like containers, Pods are meant to live for a limited period of time and when built they are given a unique ID (UID) and scheduled to physical machines where they run until completion or deletion. Kubernetes uses a controller to monitor the Pod's status change. Pod states are:

- **Pending:** Kubernetes has approved the execution of the Pod in the cluster but it is still waiting to be assigned to a targeted node.

- **Running:** The Pod has been assigned to the node and is in execution.

- **Succeeded:** Every container mapped into the Pod has successfully terminated its execution.

- **Terminated:** Unfortunately one or more containers resulted in a failure state. That could be a failure of one of the containers or an issue linked to the communication between Pod and Kubernetes management, so that a successful state cannot be presumed.


Pod is in the Pending phase until its containers are started.
At least one of the containers defined in the pod is (still) running.
All containers in the pod have terminated successfully.
Pending
Running
Succeeded
One or more containers in the pod has terminated unsuccessfully.
The state of the pod is shown as Unknown when the Kubelet stops reporting to the API server.
Unknown
Failed

In addition to the Pod health status, Kubernetes records the state of each container inside each Pod in order to reschedule in case of need.

3.2.8 Configuration And Secrets

Because accessing every Pod/container is not that easy as we do not normally know where they will be scheduled, it is far complicated sharing sensitive information. The solution adopted by Kubernetes is a shared secret infrastructure accessible by Pods/containers no matter where they are executed. Those kind of secrets could be passwords, SSH keys, tokens, even specific configurations[79].

79 *https://kubernetes.io/docs/concepts/configuration/secret/*

3.2.9 Resource Management

While a Pod is scheduled, resource usage limits can be set to reduce for example CPU or RAM consumption and, according to that, Kubernetes can implement the best allocation strategy and it can even be decided to temporarily assign at run-time more resources to a specific container if the node is idle. All those operations are in charge of Kubernetes resource manager.

Chapter 4

From virtualized hosts to containerized applications

In this section we are going to explore two significant examples: a webserver and a fileserver that are two of the typical scenarios in a datacenter.

Stress test have been performed to measure the performance in both cases: using a virtual machine and a Docker container.

Afterwards a cost/benefit analysis has been performed over different situations: on-premise, on cloud and hybrid.

4.1 Datacenter context

Everyday virtualization is all over the world, from small projects to large companies like Google, Facebook, Amazon or Microsoft, and a lot of people use it on their personal computers as well. Also security is a field where is sometimes needed to run software in a environment for which a container would do. The benefits offered by the Docker and Kubernetes made virtualization technologies extremely valuable in datacenter scenarios and many companies are adopting their own datacenter and cloud infrastructure. On the other hand a company having a large computational capacity may decide to even outsourcing part of the workload externally renting products from the cloud computing business available on the market. In such scenarios, where a cost/benefit approach is particularly considered, an analysis of the hardware requirement is key to save costs.

4.2 Tests and benchmarking

We focused our analysis on the comparison between a containerized environment and a virtual machine. First using a simple web server, then a file server. The experiments setup was done over a DL380 Gen8 server mounting 2 Intel Xeon E5-2680 v2 @ 2.80GHz CPUs each having 10 physical cores and 2 threads per core, 128 GB RAM. Ubuntu 20.10 version (CODENAME: Impish) and Docker version 20.10.12, build e91ed57.

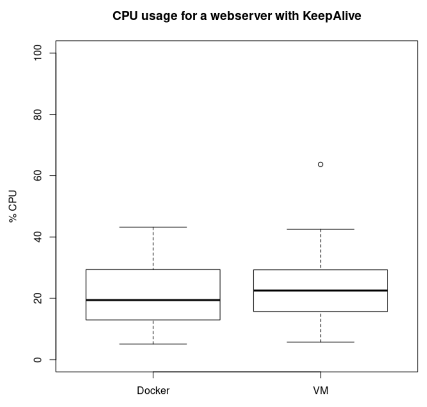
4.2.1 Apache webserver

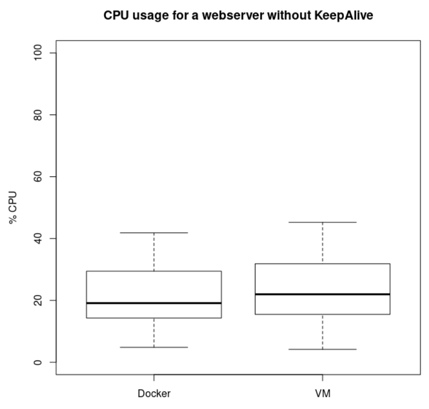
To test the web server (Apache – version 2.4.0) performance “Apache Service ab stress test” was used to generate concurrent multiple HTTP requests using:

```
ab [-k] -c 1000 -n 20000 http://192.168.1.5:8080
```

–k	KeepAlive header for HTTP requests enabled/disabled – if active a single TCP connection will be kept up to serve multiple HTTP requests.

-c 1000	concurrent sessions will be activated to mimic a 1000 user workloads
-n 20000	20000 HTTP requests will be generated for each of the 1000 sessions

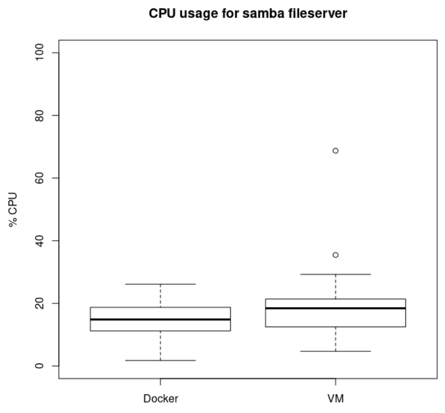




with KeepAlive activated KeepAlive disactivated

4.2.2 Fileserver

To test the fileserver (SAMBA – version 4.13.17) a file transfer over ethernet cable of a single 3.1 GB file was performed multiple times.

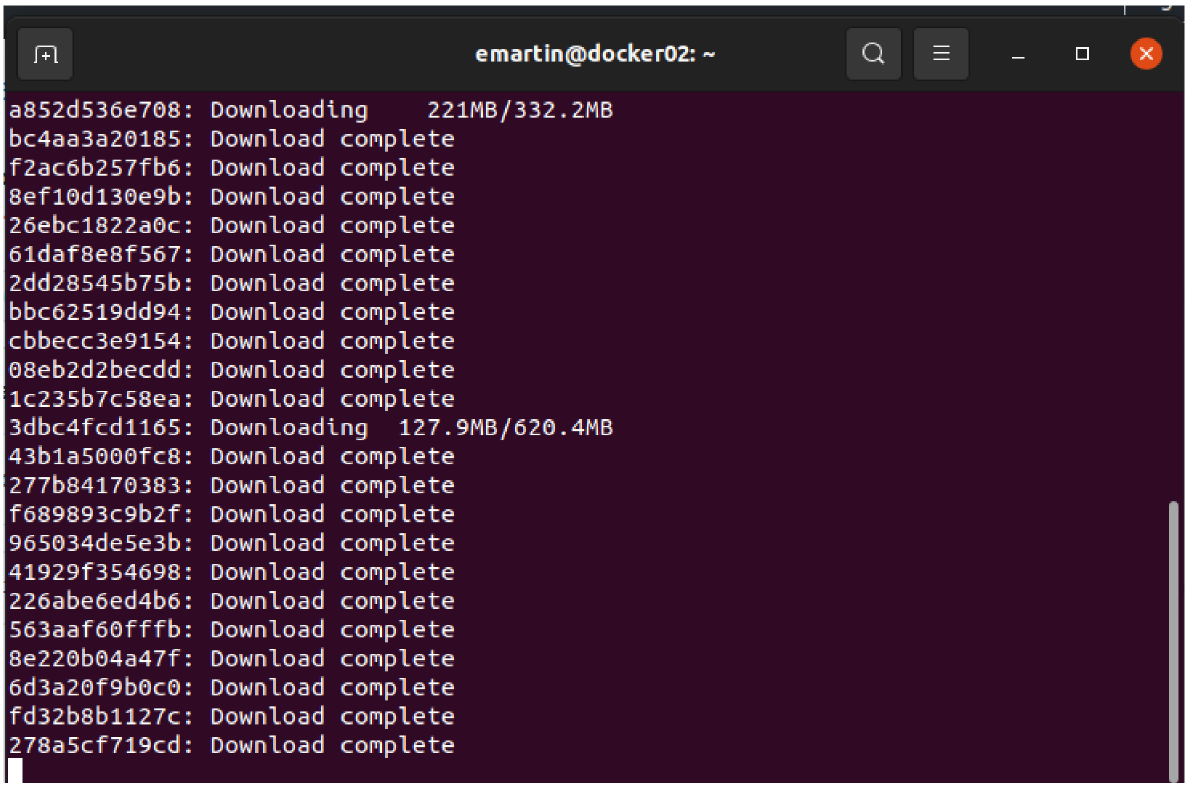


A slight benefit can be seen when using Docker over a virtual machine. Even though not extreme, a performance improvement is still visible and, thinking of complex datacenters with hundreds of concurrent containers, this could translate in a cost saving that must be taken into consideration.

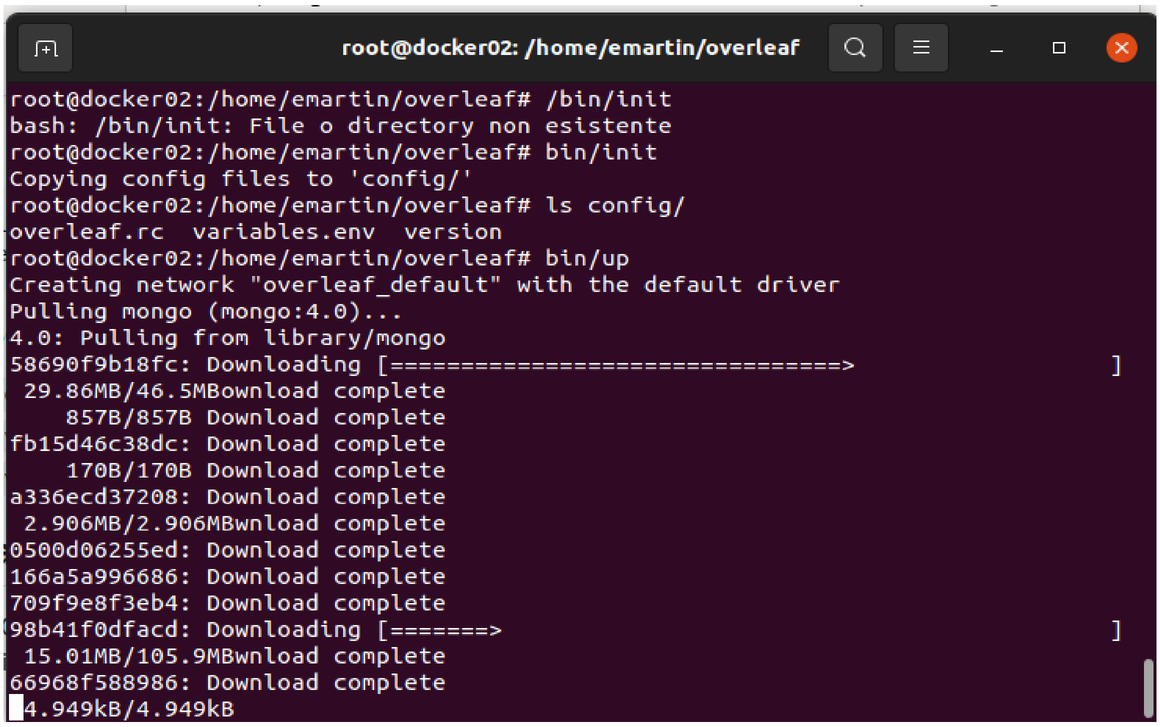
4.2.3 OverLeaf

In order to experiment different solutions for datacenter scenarios, we tried to install and run a open-source online real-time collaborative LaTeX editor called OverLeaf.
First we check if the prerequisites for installing the application: bash, docker and docker-compose, then we clone the software repository (that will also download and install mongodb and redis)

```
git clone https://github.com/overleaf/toolkit.git ./overleaf
```

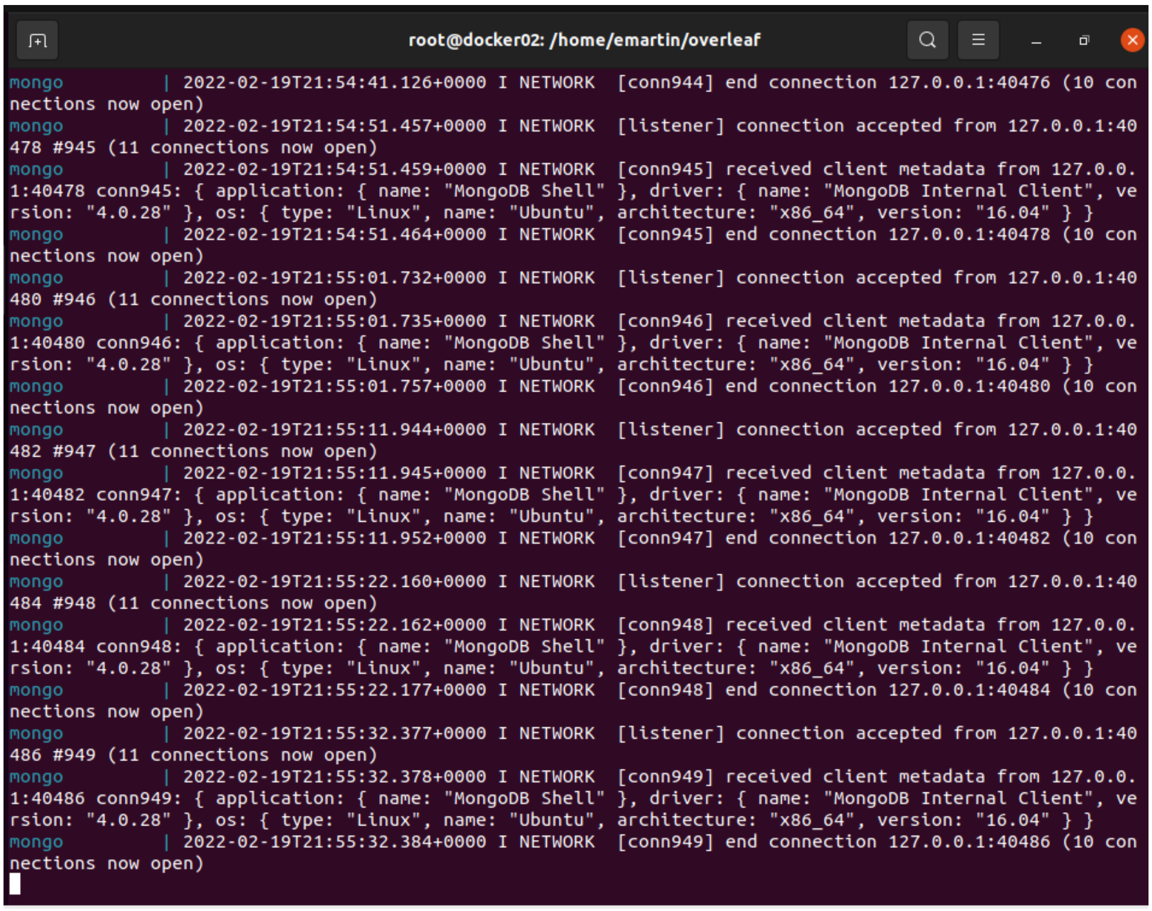


```
a852d536e708: Downloading    221MB/332.2MB
bc4aa3a20185: Download complete
f2ac6b257fb6: Download complete
8ef10d130e9b: Download complete
26ebc1822a0c: Download complete
61daf8e8f567: Download complete
2dd28545b75b: Download complete
bbc62519dd94: Download complete
cbbecc3e9154: Download complete
08eb2d2becdd: Download complete
1c235b7c58ea: Download complete
3dbc4fcd1165: Downloading  127.9MB/620.4MB
43b1a5000fc8: Download complete
277b84170383: Download complete
f689893c9b2f: Download complete
965034de5e3b: Download complete
41929f354698: Download complete
226abe6ed4b6: Download complete
563aaf60fffb: Download complete
8e220b04a47f: Download complete
6d3a20f9b0c0: Download complete
fd32b8b1127c: Download complete
278a5cf719cd: Download complete
```



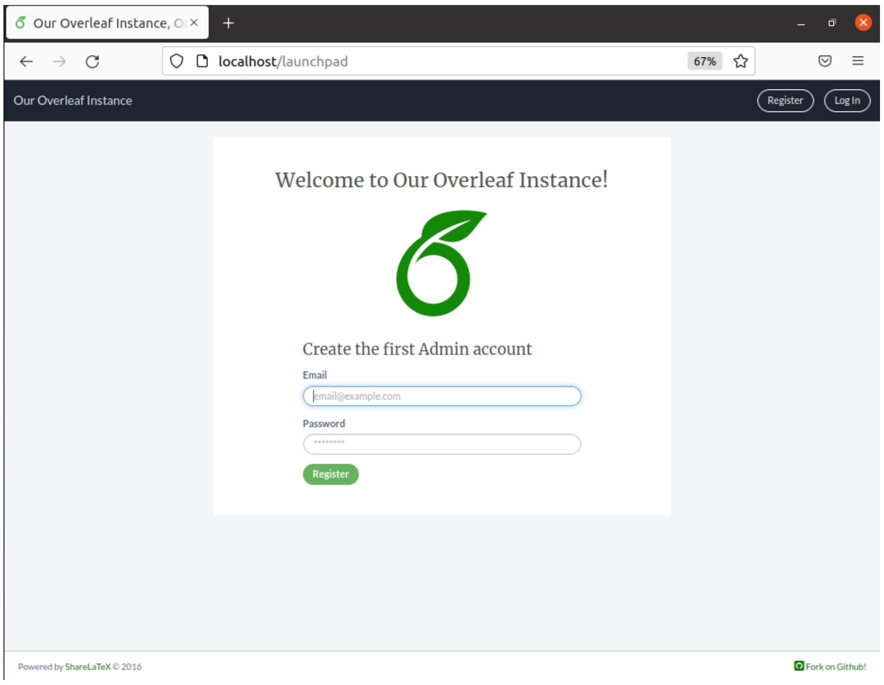
```
root@docker02:/home/emartin/overleaf# /bin/init
bash: /bin/init: File o directory non esistente
root@docker02:/home/emartin/overleaf# bin/init
Copying config files to 'config/'
root@docker02:/home/emartin/overleaf# ls config/
overleaf.rc  variables.env  version
root@docker02:/home/emartin/overleaf# bin/up
Creating network "overleaf_default" with the default driver
Pulling mongo (mongo:4.0)...
4.0: Pulling from library/mongo
58690f9b18fc: Downloading [================================>                 ]
 29.86MB/46.5MBownload complete
    857B/857B Download complete
fb15d46c38dc: Download complete
    170B/170B Download complete
a336ecd37208: Download complete
 2.906MB/2.906MBwnload complete
0500d06255ed: Download complete
166a5a996686: Download complete
709f9e8f3eb4: Download complete
98b41f0dfacd: Downloading [=======>                                          ]
 15.01MB/105.9MBwnload complete
66968f588986: Download complete
 4.949kB/4.949kB
```

Once the installation is completed, the database is ready to accept connections



```
mongo          | 2022-02-19T21:54:41.126+0000 I NETWORK  [conn944] end connection 127.0.0.1:40476 (10 con
nections now open)
mongo          | 2022-02-19T21:54:51.457+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40
478 #945 (11 connections now open)
mongo          | 2022-02-19T21:54:51.459+0000 I NETWORK  [conn945] received client metadata from 127.0.0.
1:40478 conn945: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ve
rsion: "4.0.28" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version: "16.04" } }
mongo          | 2022-02-19T21:54:51.464+0000 I NETWORK  [conn945] end connection 127.0.0.1:40478 (10 con
nections now open)
mongo          | 2022-02-19T21:55:01.732+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40
480 #946 (11 connections now open)
mongo          | 2022-02-19T21:55:01.735+0000 I NETWORK  [conn946] received client metadata from 127.0.0.
1:40480 conn946: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ve
rsion: "4.0.28" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version: "16.04" } }
mongo          | 2022-02-19T21:55:01.757+0000 I NETWORK  [conn946] end connection 127.0.0.1:40480 (10 con
nections now open)
mongo          | 2022-02-19T21:55:11.944+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40
482 #947 (11 connections now open)
mongo          | 2022-02-19T21:55:11.945+0000 I NETWORK  [conn947] received client metadata from 127.0.0.
1:40482 conn947: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ve
rsion: "4.0.28" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version: "16.04" } }
mongo          | 2022-02-19T21:55:11.952+0000 I NETWORK  [conn947] end connection 127.0.0.1:40482 (10 con
nections now open)
mongo          | 2022-02-19T21:55:22.160+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40
484 #948 (11 connections now open)
mongo          | 2022-02-19T21:55:22.162+0000 I NETWORK  [conn948] received client metadata from 127.0.0.
1:40484 conn948: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ve
rsion: "4.0.28" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version: "16.04" } }
mongo          | 2022-02-19T21:55:22.177+0000 I NETWORK  [conn948] end connection 127.0.0.1:40484 (10 con
nections now open)
mongo          | 2022-02-19T21:55:32.377+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40
486 #949 (11 connections now open)
mongo          | 2022-02-19T21:55:32.378+0000 I NETWORK  [conn949] received client metadata from 127.0.0.
1:40486 conn949: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ve
rsion: "4.0.28" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version: "16.04" } }
mongo          | 2022-02-19T21:55:32.384+0000 I NETWORK  [conn949] end connection 127.0.0.1:40486 (10 con
nections now open)
```

first of all we are going to create an admin user-defined by connecting to `http://localhost/launchpad`



Welcome to Our Overleaf Instance!

Create the first Admin account

Email

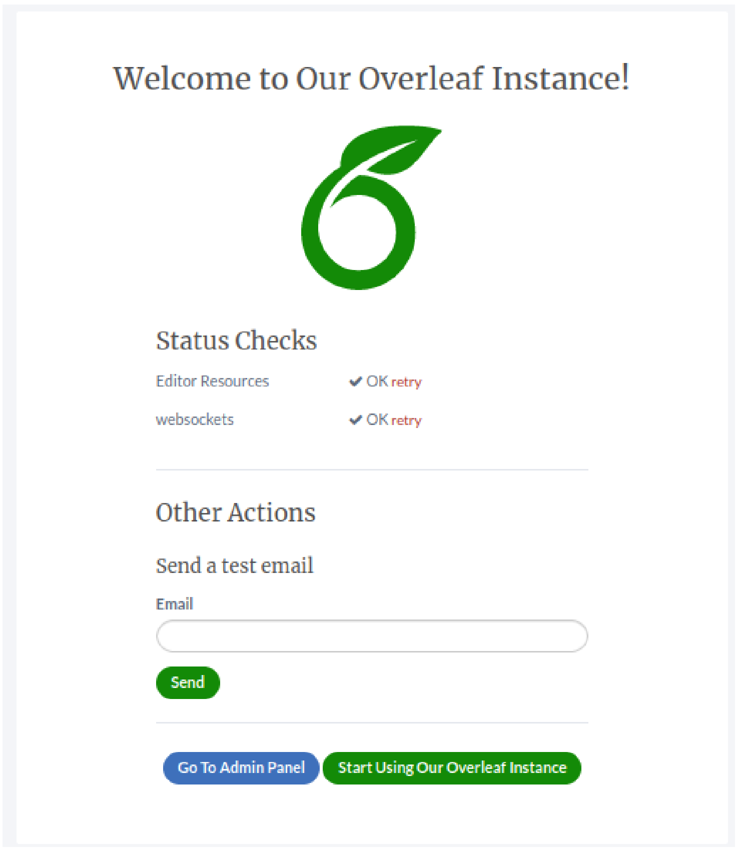
email@example.com

Password

Register

Powered by ShareLaTeX © 2016

Fork on Github!



Welcome to Our Overleaf Instance!

Status Checks

Editor Resources	✓ OK retry
websockets	✓ OK retry

Other Actions

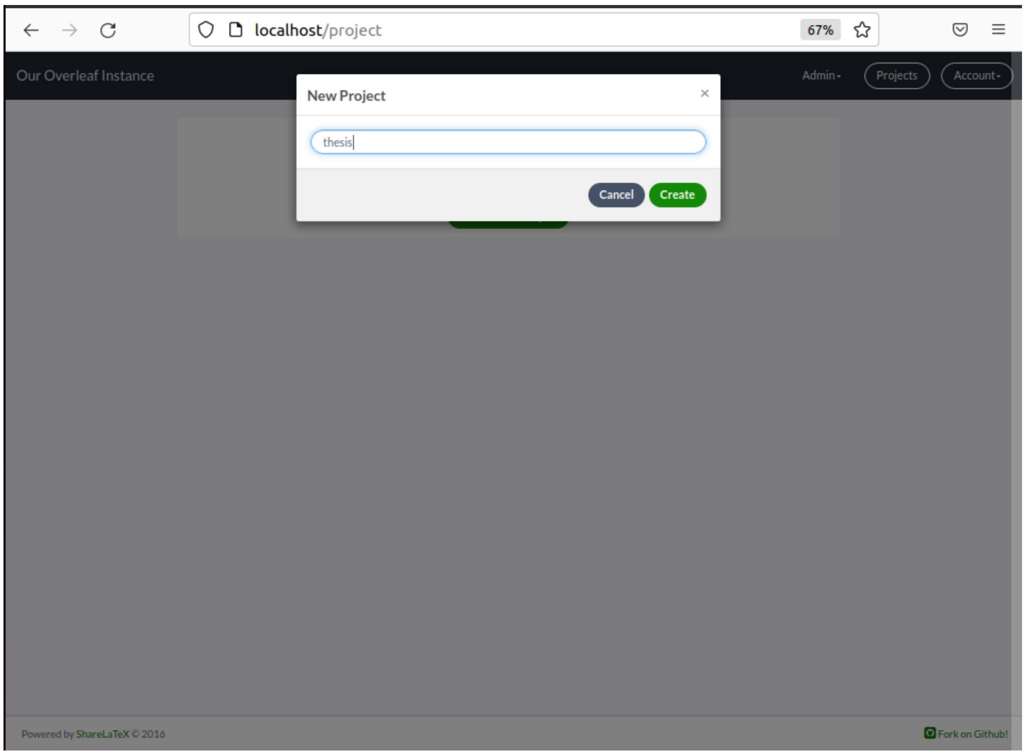
Send a test email

Email

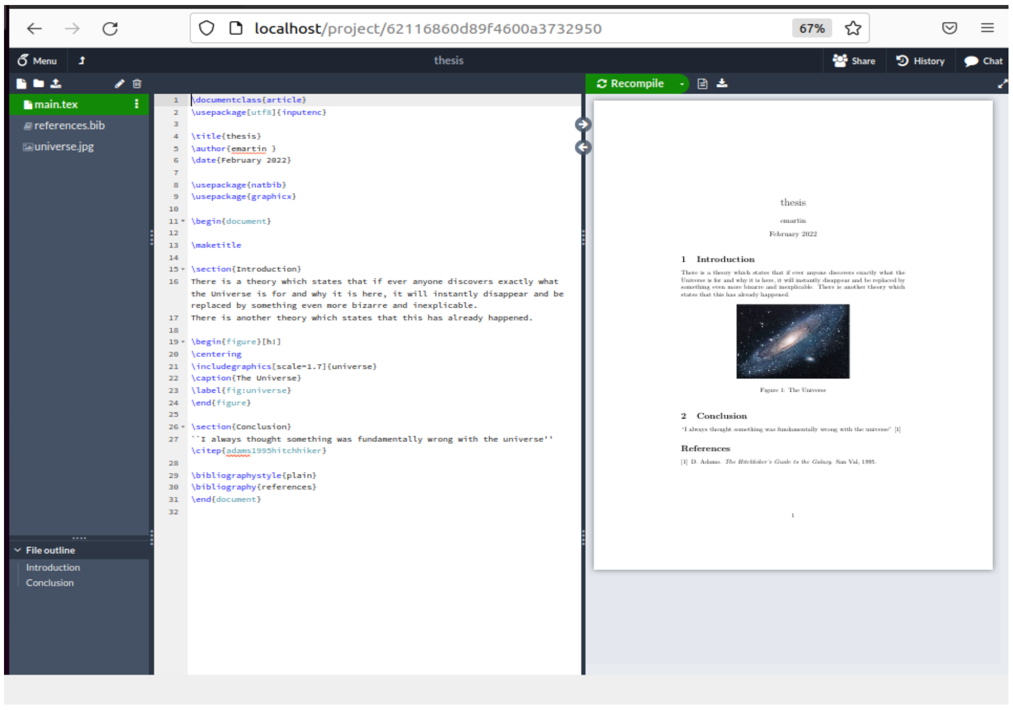
Send

Go To Admin Panel

Start Using Our Overleaf Instance



We can create a new project as shown below



```
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{thesis}
\author{emartin }
\date{February 2022}

\usepackage{natbib}
\usepackage{graphicx}

\begin{document}

\maketitle

\section{Introduction}
There is a theory which states that if ever anyone discovers exactly what
the Universe is for and why it is here, it will instantly disappear and be
replaced by something even more bizarre and inexplicable.
There is another theory which states that this has already happened.

\begin{figure}[h!]
\centering
\includegraphics[scale=1.7]{universe}
\caption{The Universe}
\label{fig:universe}
\end{figure}

\section{Conclusion}
``I always thought something was fundamentally wrong with the universe''
\citep{adams1995hitchhiker}

\bibliographystyle{plain}
\bibliography{references}
\end{document}
```

thesis

emartin

February 2022

1 Introduction

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.

Figure 1: The Universe

2 Conclusion

"I always thought something was fundamentally wrong with the universe" [1]

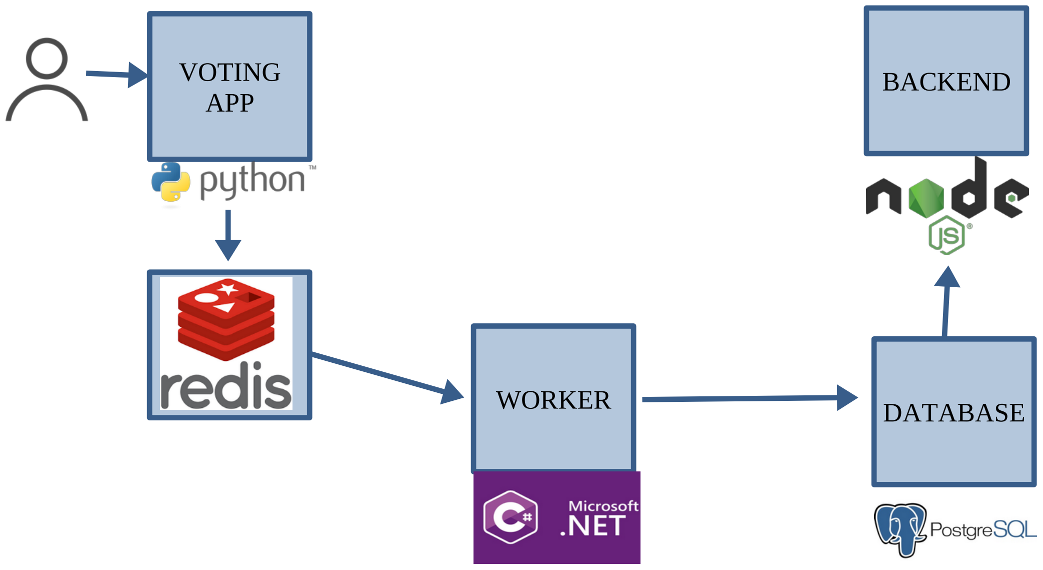
References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.

1

4.2.4 Voting webAPP

Another way containers could be used in a company is via webApp. Here a simple voting web application will be containerized and installed in a cluster of 3 nodes. The application is composed of a web frontend written in Python, a redis key-value store that keep track of the votes, a worker application written in .NET that calculates the votes and the percentage, a PostgreSQL database and a backend web application written in Node.JS needed to summarize all the votes.

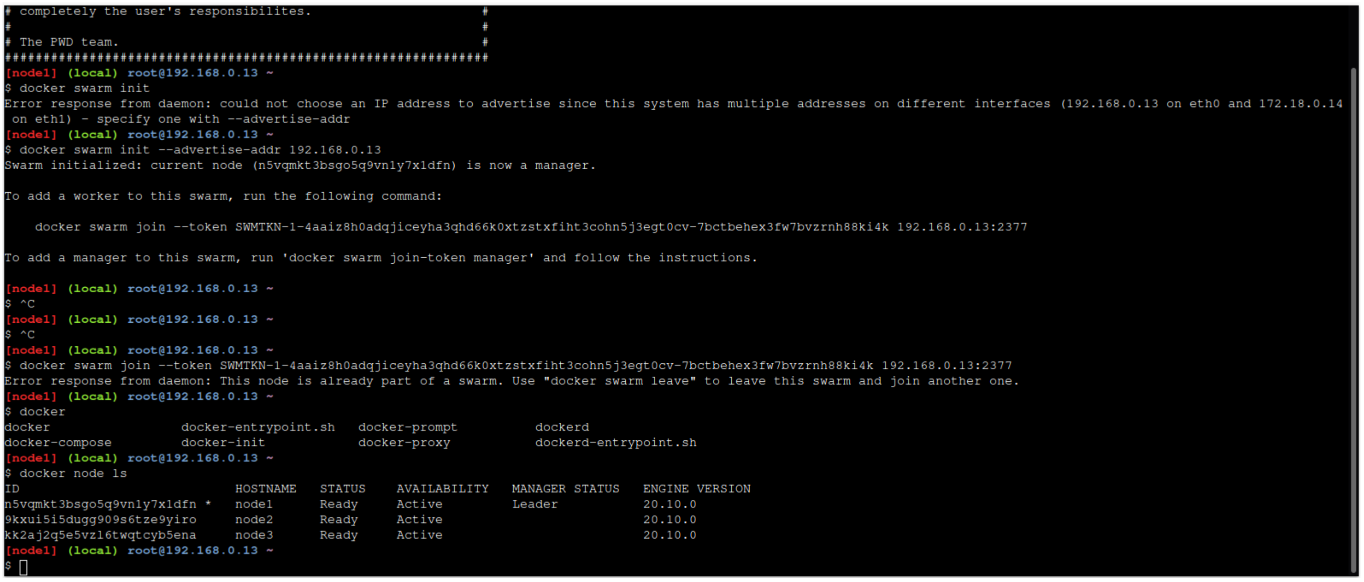


First of all a cluster has been created using Play-With-Docker infrastructure.[80] On the first node the cluster has been initialized and the first node become the manager of the cluster, then the two other node have been added to the cluster using the token initially generated (note: sometimes the command will be split just to be more easy to read)

```
docker swarm init --advertise-addr 192.168.0.13
```

```
docker swarm join –token
SWMTKN-1-4aaiz8h0adqjiceyha3qhd66k0xtzstxfiht3cohn5j3egt0cv-7bctbe
hex3fw7bvzrnh88ki4k 192.168.0.13:2377
```

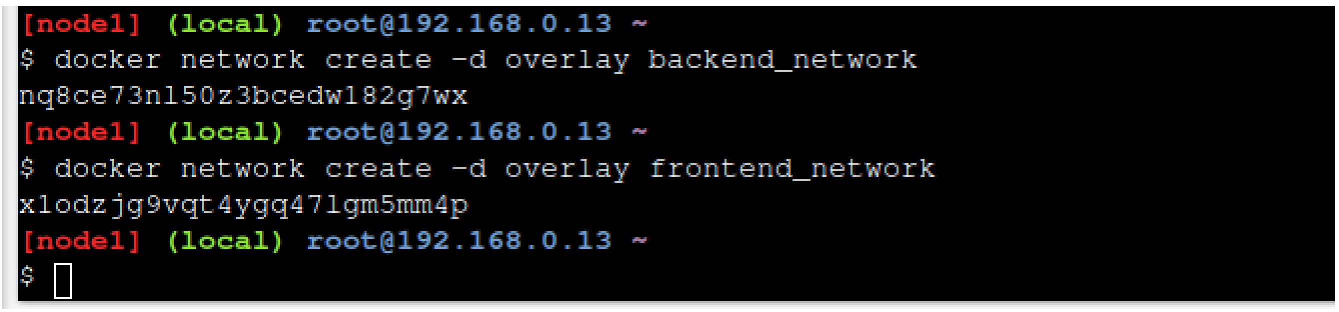
80 *See https://labs.play-with-docker.com*



```
# completely the user's responsibilites.                        #
#                                                               #
# The PWD team.                                                 #
#################################################################
[node1] (local) root@192.168.0.13 ~
$ docker swarm init
Error response from daemon: could not choose an IP address to advertise since this system has multiple addresses on different interfaces (192.168.0.13 on eth0 and 172.18.0.14
 on eth1) - specify one with --advertise-addr
[node1] (local) root@192.168.0.13 ~
$ docker swarm init --advertise-addr 192.168.0.13
Swarm initialized: current node (n5vqmkt3bsgo5q9vn1y7x1dfn) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4aaiz8h0adqjiceyha3qhd66k0xtzstxfiht3cohn5j3egt0cv-7bctbehex3fw7bvzrnh88ki4k 192.168.0.13:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[node1] (local) root@192.168.0.13 ~
$ ^C
[node1] (local) root@192.168.0.13 ~
$ ^C
[node1] (local) root@192.168.0.13 ~
$ docker swarm join --token SWMTKN-1-4aaiz8h0adqjiceyha3qhd66k0xtzstxfiht3cohn5j3egt0cv-7bctbehex3fw7bvzrnh88ki4k 192.168.0.13:2377
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave this swarm and join another one.
[node1] (local) root@192.168.0.13 ~
$ docker
docker                 docker-entrypoint.sh   docker-prompt          dockerd
docker-compose         docker-init            docker-proxy           dockerd-entrypoint.sh
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
n5vqmkt3bsgo5q9vn1y7x1dfn *   node1      Ready     Active         Leader           20.10.0
9kxui5i5dugg909s6tze9yiro     node2      Ready     Active                          20.10.0
kk2aj2q5e5vzl6twqtcyb5ena     node3      Ready     Active                          20.10.0
[node1] (local) root@192.168.0.13 ~
$ 
```

Then 2 separated networks will be created, one for the frontend and one for the backend to isolate the communications. The worker, acting as an intermediator between the two parts of the application, will be in both networks.

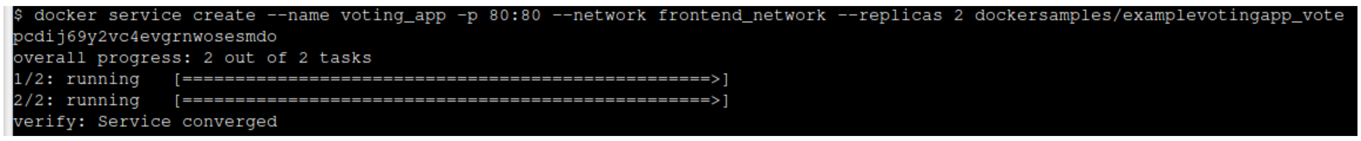
```
docker network create -d overlay backend_network
docker network create -d overlay frontend_network
```



```
[node1] (local) root@192.168.0.13 ~
$ docker network create -d overlay backend_network
nq8ce73nl50z3bcedw182g7wx
[node1] (local) root@192.168.0.13 ~
$ docker network create -d overlay frontend_network
xlodzjg9vqt4ygq47lgm5mm4p
[node1] (local) root@192.168.0.13 ~
$ 
```

Now it is possible to download and install the component parts of the application

```
docker service create --name voting_app -p 80:80 --network frontend_network
--replicas 2 dockersamples/examplevotingapp_vote
```



```
$ docker service create --name voting_app -p 80:80 --network frontend_network --replicas 2 dockersamples/examplevotingapp_vote
pcdij69y2vc4evgrnwosesmdo
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
```

In this example the fronted webapp has been installed in the cluster and two replicas have been created.

```
docker service create --name redis --network frontend_network redis:3.2
```

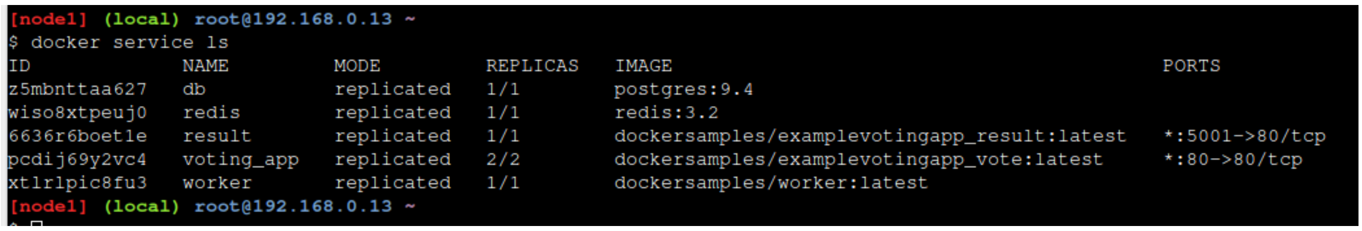
```
docker service create --name worker --network frontend_network
--network backend_network dockersamples/worker
```

```
docker service create --name db --network backend_network
--mount type=volume,source=db-data,target=/var/lib/postgresql/data
-e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker service create --name result --network backend_network
-p 5001:80 dockersamples/examplevotingapp_result
```

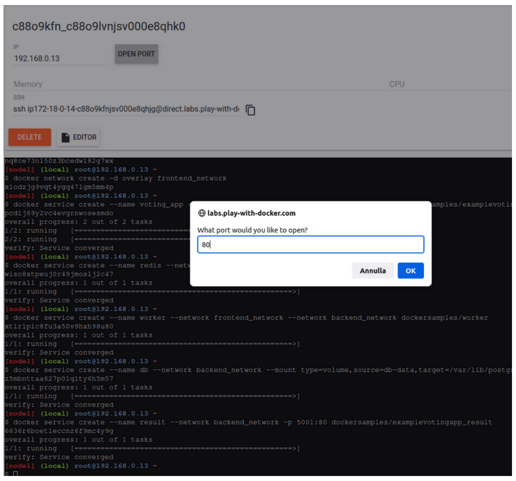
Now, if we want to check if all the requested services are running correctly, we can execute "docker service ls" (note the 2 replicas of voting_app)

```
docker service ls
```



```
[node1] (local) root@192.168.0.13 ~
$ docker service ls
ID             NAME         MODE         REPLICAS   IMAGE                                            PORTS
z5mbnttaa627   db           replicated   1/1        postgres:9.4
wiso8xtpeuj0   redis        replicated   1/1        redis:3.2
6636r6boet1e   result       replicated   1/1        dockersamples/examplevotingapp_result:latest     *:5001->80/tcp
pcdij69y2vc4   voting_app   replicated   2/2        dockersamples/examplevotingapp_vote:latest       *:80->80/tcp
xtlrlpic8fu3   worker       replicated   1/1        dockersamples/worker:latest
[node1] (local) root@192.168.0.13 ~
```

If we expose to internet the port 80 that is used by the voting app, we will get:



c88o9kfn_c88o9lvnjsv000e8qhk0

IP
192.168.0.13

Memory
CPU

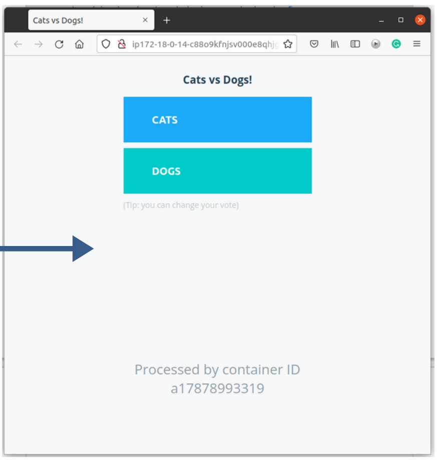
SSH
ssh ip172-18-0-14-c88o9kfnjsv000e8qhjg@direct.labs.play-with-d

```
nq8ce73nl50z3bcedw182g7wx
[node1] (local) root@192.168.0.13 ~
$ docker network create -d overlay frontend_network
xlodzjg9vqt4ygq47lgm5mm4p
[node1] (local) root@192.168.0.13 ~
$ docker service create --name voting_app                                    amples/examplevoti
pcdij69y2vc4evgrnwosesmdo
overall progress: 2 out of 2 tasks
1/2: running   [===========================
2/2: running   [===========================
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service create --name redis --netw
wiso8xtpeuj0r49jmoslj2c47
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service create --name worker --network frontend_network --network backend_network dockersamples/worker
xtlrlpic8fu3a50v9hxh98u80
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service create --name db --network backend_network --mount type=volume,source=db-data,target=/var/lib/postg
z5mbnttaa627p01qlty6h5m57
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service create --name result --network backend_network -p 5001:80 dockersamples/examplevotingapp_result
6636r6boet1eccnz6f9mc4y9g
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$
```

labs.play-with-docker.com

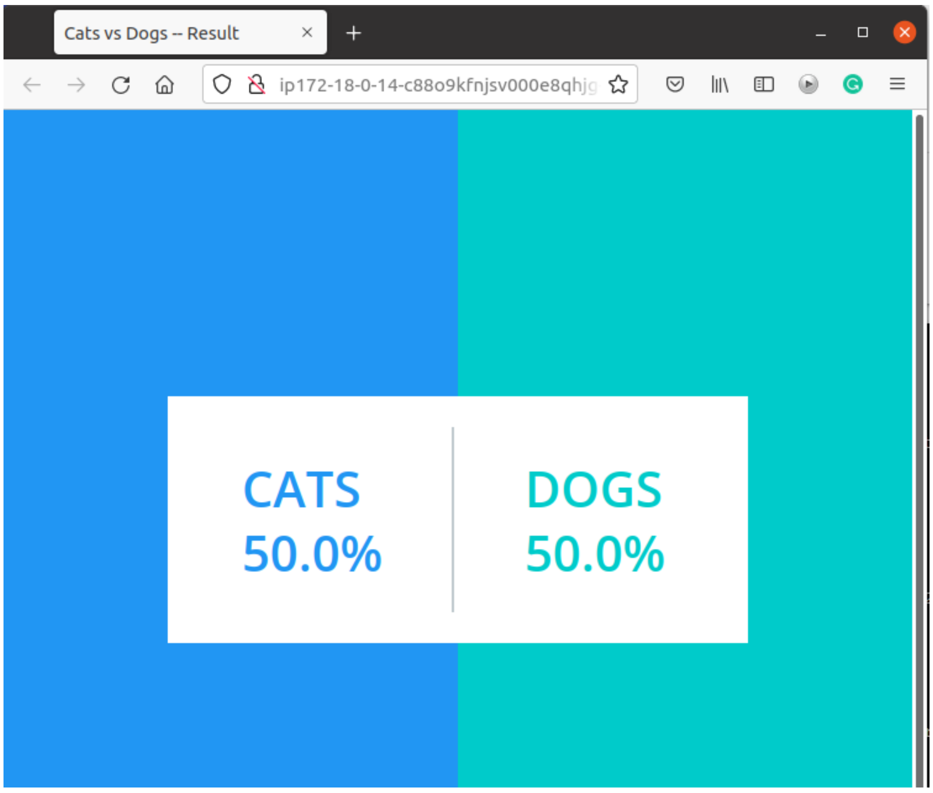
What port would you like to open?

80



Cats vs Dogs!
ip172-18-0-14-c88o9kfnjsv000e8qhj
Cats vs Dogs!
CATS
DOGS
(Tip: you can change your vote)
Processed by container ID
a17878993319

On the other hand, if we expose the port 5001 that is used by the backend



4.3 A cost/benefits analysis

Kubernetes and containers are spreading among the IT world and Docker actually starts to take since Google, among other big companies, invested so much on the project. Since then, both technologies have grown exponentially. An S&P Global Market Intelligence[81] research predicts that containerization will grow by 30%, while a survey of over 500 IT professionals conducted by Portworx and Aqua Security in 2019[82] showed that 87% used containers in production, in the meanwhile, membership of the Cloud Native Computing Foundation have been continuously growing since.

Most organizations have valid motivations to adopt containerization, and even if the specific motivations of each organization are different, there are points in common like:

81 *https://451research.com/451-research-says-application-containers-market-will-grow-to-reach-4-3bn-by-2022*
82 *https://portworx.com/wp-content/uploads/2019/05/2019-container-adoption-survey.pdf*

- **Costs reduction** - Containers can be much more efficient compared to software running on virtual machines and can be bundled more densely on the same physical machine reducing the amount of resources needed to run the same application, whether it is in a data center or in the cloud. As containers uses the operating system as a common base, those are smaller compared to a virtual machine and demand fewer memory and limited storage capacity, less electric power. For an company the cost savings can be significant.

- **Portability** - Containers run exactly on various hardware, meaning there is no need to worry about compatibility or library dependencies, helping the productivity and, coincidentally, containers can also make it simpler to demote a part or even all of the services to the public cloud, making cloud strategies feasible.

- **Transparent updates** - As containers are built using different layer of independent software, all the updates or even a complete substitution of a single layer became much much easier and faster. In addition to that, is reasonable to estimate a lesser error rate is likely to happen. And, due to segmentation of layers and isolation, an issue regarding a single application is not expected to effect the entire application or software.

Kubernetes has emerged as the standard platform for orchestrating container, giving companies the tools they need to automate tasks such as scaling and recovering from failure.
The whole point of Kubernetes is that it makes containers possible. Without one container orchestration platform, organizations would not be realistically able to achieve the advantages of containers, because the trade-offs would be too high. Kubernetes enables organizations to automate load balancing, self-healing, orchestration of archiving, configuration management, and implementation and the automated rollbacks, including advanced implementation strategies such as test implementations. Without this level of automation, containers would not be feasible in production. The main challenges of adopting Kubernetes in the company are :

- **Vendor lock-in** - The simple adoption of Kubernetes is not realistic for every company, and many of them prefer to rely on commercial distribution or one Kubernetes platforms owned by a cloud provider. That could result in a vendor lock-in, obliging companies to accept the conditions and limitations imposed by others.

- **Migration costs** - Shifting all or a part of the services to Kubernetes needs time and skills, and often organizations underestimate the total cost for the conversion.
  Kubernetes is still a rather new technology, is also very rich in functionalities that even after a while, one can have the feeling of being in a continuous learning phase. The framework also depends on an extensive ecosystem of open source and proprietary tools and solutions. On the other hand, companies are facing two big challenges related to tools:

- **Tool selection** - just choosing the right selection of tools means accepting numerous trade-offs. How well do the tools integrate with the rest of the softwares? Deciding among flexibility of open source or a more costly proprietary tool?

- **Tool management** - it kind of normal for an enterprise to have a lot of different tools, which result in a steep learning curve and also creates further complexity that must be properly handled.
  So shifting to Kubernetes needs a company to rethink the way they manage security, networking and storing. While containers and Kubernetes are not inherently less secure than traditional virtual machines, they require a different approach to security that relies more on configuration management and perimeter security. Even proper load balancing setup can also be tricky and it is essential to ensure that the application remains available and performing no matter the infrastructure had been chosen.

However, there are ways to solve these challenges, as well as other challenges that might arise during production operations and that depends on the path a company wants to undertake.

4.4 From on-premise solutions to cloud platforms

As Kubernetes and containerized applications are gaining more and more popularity among companies and institutions, because they offer many advantages in terms of development, agility and cost-effectiveness, it’s true that they pose a series of new challenges. In the next two sections we are going to describe two scenarios the companies may face when adopting this technology. Even though there are common aspects for every company to be taken into consideration, there isn’t a unique approach to follow.

A pure, open source Kubernetes is always an option, and it is probably the first type of Kubernetes that someone will be experiencing. Kubernetes plain installation is extremely flexible and extensible, but it also lacks enterprise-grade functionality regarding monitoring, management status, availability, lifecycle operations, and more.

It is true that there are no costs of license for the use of Kubernetes, organizations have almost infinite control over configurations and extensions and can be installed on site or on any cloud provider, on any operating system. On the other hand, there are disadvantages like lack of skills as open source distribution is the option most technically challenging, requiring a group of very expert technicians to make it ready for the company’s needs.

The only organizations should take into consideration the use of pure open source Kubernetes are highly techniques that consider to have the ability to build their own tools and customized platform because they think that could be a competitive advantage. Companies that predict to use open source Kubernetes should have a team of experts already that is able to provide the support necessary to meet business needs.

For those who are not able to access a pure Kubernetes option, other solution could come in handy, for example Platform-as-a-Service or Kubernetes Enterprise Platforms. Kubernetes Platform-as-a-Service (PAAS) offering products by a vendor that creates ready-to-use Kubernetes packages. The platform usually includes preconfigured Kubernetes and associated tools. The PAAS solutions generally include security, networking and storage as part of the platform and it is usually much easier for organizations to get going with a PAAS version of Kubernetes. PAAS creates a coherent organization achieved by reducing the configuration options available to users and by presetting tools and services, which means PAAS solutions are significantly less flexible and sometimes difficult to update.

A great advantage is short learning curve as PAAS solutions help reduce the skills deficit by lowering the barrier to entry for individuals end organizations to start work with Kubernetes, also reducing the learning curve means the organizations are capable of get services in production faster. The adoption of preconfigured Kubernetes platforms gets better security standards above all because it is easier to guarantee adherence to the security best practices. The price to pay involves high licensing costs, plus additional costs for support service subscriptions or additional features, moreover third-party PAAS vendors will make it difficult the passage in the future to a different vendor, not to mention that it could be hard to migrate from one environment to another, leading to further lock-in.

But for very big companies with complex implementations e multi-environments and with sufficient technical skills to exploit the flexibility that Kubernetes platforms offer, that could be a great option as managed Kubernetes has a short learning curve, and the IT engineers can start delivering valuable applications in Kubernetes without having to become experts in the administration of the cluster, and of course will be a supplier’s duty to deals with day 2 operations, such as updates and patches, simplifying things further more for the in-house engineering team. For what security is concerned, the service provider takes care of patching security, hiring one much greater responsibility for security with respect to suppliers of cloud services compared to the cloud-hosted distribution. It is true that Kubernetes managed services are more expensive than distributions in cloud hosting, but cheaper of PAAS offers. The organizations will not be able to choose tools or features outside of what is provided by the managed service provider. This can reduce stress related to tool selection, but it can also mean that the organizations may not be able to get all the functionality they need.

4.5 Hybrid solutions

Hybrid solution that mixed on-premise solutions with enterprise platforms offer a Kubernetes infrastructure that help organizations managing the entire life cycle applications, services and generally focus on providing a centralized platform for controlling multiple clusters and multiple environments. These platforms make easier for centralized teams to control configurations and manage access for the entire organization. Particularly for those distributed around the world. Such Kubernetes platforms offer considerably more flexibility than any other option.

Those platforms blend particularly well ease of use for dev-oriented teams, while operations-focused platforms generally provide more control focused on ensuring the reliability and availability for businesses need because they come with a suite complete with tools so that the organizations must not lose time to select them. At the same time, it is easier to integrate them with new tools, if necessary, rather than with other options. The single and centralized control for the cluster management allows the organizations to pursue an approach multi-cloud or hybrid, without increasing drastically complexity and guaranteeing at the same time adherence to the policies of organizational governance on all environments. Corporate platforms make even workloads easy to move from one environment to another. The Enterprise Kubernetes platforms are the most sensible for large companies that manage multiple environments and hundreds of clusters and often make it easier to move workloads between environments.

It is true that increased flexibility and control means that the organizations must have a higher level of IT skills for successfully use the Enterprise Kubernetes platforms. While only a small central team must learn about Kubernetes from inside out, the organizations need at least some experience internally and depending on the supplier, license fees and contracts assistance can be considerably expensive, especially for smaller companies and/or for smaller use cases.

For large companies with complex implementations and multi-environment and with sufficient technical skills to exploit the flexibility that Kubernetes platforms offer are the most suitable for this option. This is often the best choice for organizations that are further along on their Kubernetes journey.

When organizations rush to adopt Kubernetes they cannot ignore the real business needs for security, availability and disaster recovery. Every organization must decide what’s the right balance between ease of use, time-to-market and flexibility, and if you prefer to pay more for the time for internal Kubernetes experts or for licensing. The point is, if the enterprises want to use Kubernetes for production workloads, especially the mission-critical ones, cannot use Kubernetes out-of-the-box. The right way to get Kubernetes depends on the priorities and capabilities of each company.

Chapter 5

Conclusions and future works

Virtualization has been an active field of research in computer science for decades. Starting from the work of Gerald J. Popek and Robert P. Goldberg, hardware virtualization and hypervisors became popular around the 1970s because of their ability to exploit hardware resources of mainframes.
Nowadays, type-2 hypervisors are still in use in computers for many scopes, e.g. to overcome incompatibility problems and to create isolated environments for security-critical applications. Type-1 hypervisors, on the other hand, are more generally used in data centers, where the broad hardware resources need to be split into smaller chunks where isolation is a constraint.

In the late 1980s, a new lightweight virtualization approach, named operating-system-level virtualization, marked the start of a new era. Today, the most popular implementations of this new kind of virtualization are container technologies. Despite being often considered the successors of hypervisors, container technologies are actually not meant to replace them. In fact, the virtual machine abstraction introduced by hypervisors is based on the concept of virtual hardware, on top of which an entire operating system is executed. Containers, on the other side, do not involve virtual hardware and run the same kernel of the underlying host. Moreover, some container technologies, especially Docker, shifted the focus from virtualizing an entire system to isolating a single piece of software. Application containers, in fact, are increasingly growing in popularity, especially among software developers, as a fast and easy way to pack and distribute software. Virtual machine and containers also provide different levels of isolation, with containers being intrinsically less secure than virtual machines because of their architecture. In the near future, containers might be able to guarantee even higher levels of isolation thanks to the advancements in underlying containment features of the Linux kernel that are being introduced at each release. With regards to performance, containers have proven to be really fast. Benchmarks actually showed that Docker introduces very low overhead and thus containers are to be considered a good solution when dealing with high performance requirements. Because of all the aforementioned reasons, the choice of using virtual machines or containers must be made on a case-by-case basis, depending on the specific isolation and performance needs.

With that in mind, many datacenter services are moving to containerization. This work wants to quantify the advantages introduced by those technologies taking as example two well known scenarios: a fileserver and a webserver. Expected results have been supported by the tests that have been executed, which demonstrate that a slight performance increase does exist, switching from a virtualized environment to a containerized one. Those results suggest that in a datacenter environment, the increment in terms of performance of a set of simultaneously running processes, might be significative. In addition to that, a cost/benefit analysis have been done for a cloud solution and an hybrid solution to study the iter for switching from an on premise solution to a cloud or hybrid one.

Considering the attention that Docker and Kubernetes are gaining in the last few years, a great effort will be dedicated to security topics. In fact, dealing with security challenges is one of the greatest concerns of IT operation and Kubernetes is not meant to enforce security policies, for example finding vulnerabilities in the images. So companies using Kubernetes will be obliged to find a different way to secure datacenter services. Just to cite an example, based on the default network policy, Kubernetes pods can communicate with each other and external endpoints. Due to application or infrastructure security issues, if one container or pod is breached, all others can be attacked. As cybersecurity is a problem that every organization will be facing in their life, a future work could be focusing on the integration of those two aspects.