

Cloud based Video Processing and Transmission

Ricardo Jorge da Silva Ramos

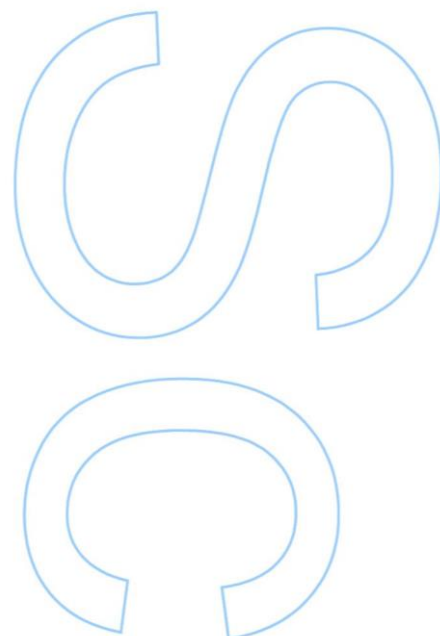
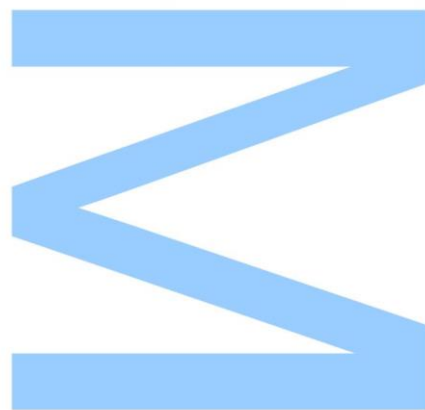
Mestrado Integrado Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2021

Orientador

Ivone Amorim, Professora Doutora
Faculdade de Ciências da Universidade do Porto

Coorientador

Sérgio Crisóstomo, Professor Doutor
Faculdade de Ciências da Universidade do Porto

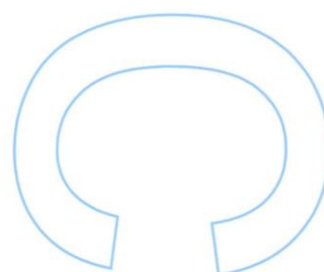
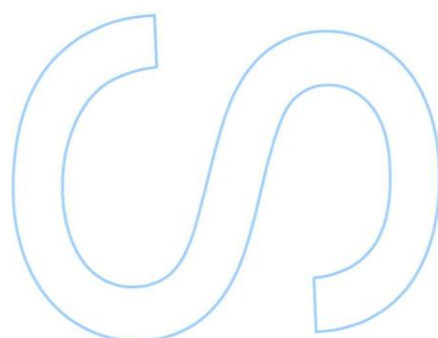
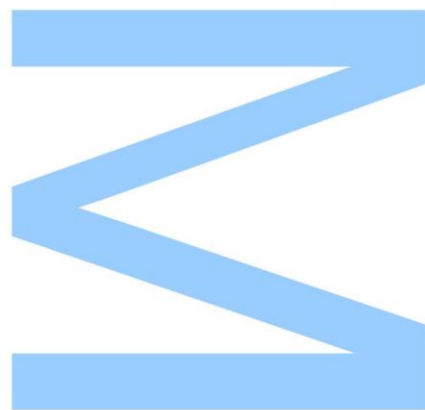




Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ____/____/____



Abstract

In recent years, with the increasing need for digital services, the cloud computing market has seen a wave of mass adoption and growth. The cloud technology is based on virtualization methods capable of transforming one physical resource into multiple virtual ones. The cloud platforms, as well as the virtualization methods, are capable of providing advantages to many different industries. However, some industries are still reluctant to adopt this type of technology. As a consequence, they do not benefit from all its advantages and cannot offer the best possible solutions to their customers. One example of this kind of industry is the media sector, where some of the most used methods are outdated which makes them unable to keep up with. In this thesis, the drawbacks of the current methods for processing media are identified and a new media manager, **CMAM4PRO**, is developed. This new tool is cloud-based, which means it provides the media industry with an approach capable of using the cloud technology as well as virtualization. This upgrades the current methods by offering new essential attributes for their users. Among those attributes are scalability, easier and faster deployment, better cost-effectiveness, and robustness.

Resumo

Nos últimos anos, com a crescente necessidade por serviços digitais, o mercado de computação em *cloud* tem assistido a uma onda de adoção e crescimento em massa. A tecnologia da *cloud* é baseada em métodos de virtualização capazes de transformar um recurso físico em múltiplos recursos virtuais. As plataformas *cloud*, assim como os métodos de virtualização, são capazes de proporcionar vantagens a muitas indústrias diferentes. No entanto, algumas indústrias ainda estão relutantes em adotar este tipo de tecnologia. Como consequência, não beneficiam de todas as suas vantagens e não podem oferecer as melhores soluções possíveis aos seus clientes. Um exemplo deste tipo de indústria é o sector dos media, onde alguns dos métodos mais utilizados estão obsoletos. Nesta tese são identificadas as desvantagens dos métodos atuais de processamento de media e é desenvolvido um novo gestor de media, **CMAM4PRO**. Esta nova ferramenta é baseada na *cloud*, o que significa que fornece à indústria dos media uma abordagem capaz de utilizar a tecnologia da *cloud*, bem como a virtualização. Isto melhora os seus métodos atuais, oferecendo novos atributos essenciais para os seus utilizadores. Entre esses atributos encontram-se a escalabilidade, uma implementação mais fácil e rápida, uma melhor relação custo-benefício, e robustez.

Acknowledgements

Quero começar por agradecer aos meus orientadores, a Professora Ivone Amorim e o Professor Sérgio Crisóstomo, por toda a vossa orientação e ajuda nesta conclusão do meu percurso académico na FCUP. Com ênfase para todo o tempo que a Ivone dispendeu a ajudar-me com a escrita deste documento. Muito obrigado por toda a paciência e persistência, foi um prazer trabalhar com vocês.

Quero também agradecer a toda a equipa da MOG que me acompanhou durante todo este processo e que, em inúmeras ocasiões, foram essenciais para a conclusão deste projeto. Com uma menção especial para o Alexandre Ulisses e o Vasco Filipe. Ambos se mostraram sempre disponíveis para ajudar no que fosse preciso, e tiveram contribuições muito valiosas que enriqueceram este projeto e também a minha experiência na MOG. Quero também agradecer ao Hugo Cardoso e ao João Estevinho, sem a vossa ajuda seria impossível concretizar todos os objetivos práticos desta tese. Trataram-me sempre com imensa simpatia e estiveram sempre disponíveis para me fornecer explicações detalhadas e ricas. Forneceram-me também conhecimentos que me permitiram desenvolver o trabalho desta tese e que fazem de mim um melhor profissional.

Quero agradecer a todos os meus amigos que me acompanharam durante este processo. Obrigado por todos os votos de confiança e encorajamento ao longo dos anos, sem vocês esta caminhada não teria sido a mesma.

Quero agradecer à minha família por toda a ajuda e apoio dado durante estes anos. Com especial menção para os meus pais e irmãos que me acompanharam nesta etapa desafiante.

Por fim quero agradecer à pessoa mais importante neste processo, a minha companheira de uma vida, Alexandra Silva, que me forneceu apoio e motivação incondicionais. Alcancei este objetivo muito graças a ti, ajudas-te me a enfrentar os meus problemas sempre com a cabeça levantada e acreditaste sempre no meu sucesso mesmo quando eu não acreditava.

O meu profundo e sincero obrigado a todos. Posso afirmar, sem dúvida nenhuma, que entrego esta tese como um aluno e pessoa completamente diferente graças ao processo de aprendizagem que este projeto significou para mim desde o primeiro até ao último dia.

Ricardo Ramos

Contents

Abstract	i
Resumo	iii
Acknowledgements	v
Contents	ix
List of Tables	xi
List of Figures	xiv
Listings	xv
Acronyms	xvii
1 Introduction	1
1.1 Context	1
1.2 Motivation	2
1.3 Objectives	2
1.4 Cloudsail project	3
1.5 Document organization	3
2 State of the art	5
2.1 Virtualization	5
2.2 Virtual Machines	6

2.3	Containers	7
2.4	Docker	9
2.5	Orchestration	11
2.5.1	Kubernetes	13
2.6	Cloud technology	14
2.6.1	Cloud architecture	15
2.6.2	Cloud computing advantages	16
2.7	MAM4PRO	17
2.7.1	MAM4PRO architecture	17
2.7.2	MAM4PRO user interface	23
2.8	Monitoring Software	24
2.8.1	Grafana	24
2.8.2	NetData	25
2.8.3	Windows server monitoring architecture	26
3	Problem statement and proposed solution	29
3.1	Problem statement	29
3.2	Requirements Specifications	31
3.2.1	Users Identification	31
3.2.2	Functional requirements	32
3.2.3	System requirements	33
3.3	Solution proposal	34
3.4	Advantages of the proposed solution	36
3.5	Cloudsail integration	38
4	Solution Development	41
4.1	Solution overview	41
4.1.1	Container cluster architecture	42
4.1.2	MAnnouncer and RabbitMQ	44

4.1.3	SCapture container	44
4.1.4	HOperation container	46
4.1.5	RPlayout container	46
4.1.6	RPlayerWeb container	47
4.1.7	Scalability	48
4.2	Cluster creation	50
4.2.1	Kubernetes	55
4.3	Cloud Deployment	56
4.4	Monitoring software	58
5	Test and benchmark	61
5.1	Functional testing	61
5.2	System requirements	62
5.2.1	Performance	62
5.2.2	Live stream tests	68
5.2.3	Scalability	70
5.2.4	Diversity	72
5.2.5	Robustness	72
5.3	Tests conclusions	75
6	Conclusion	77
6.1	Future Work	78
	Bibliography	81

List of Tables

2.1	Codecs supported by MAM4PRO	22
2.2	Supported formats	23
3.1	Functional requirements	33
5.1	Functional requirements	61
5.2	Test workflows	63
5.3	Metrics analysis from the Grav - 2160p workflow	65
5.4	Metrics analysis from the Beach - 720p workflow	66
5.5	Average time (in minutes) to run the test workflows	67
5.6	Non-virtualized solution	67
5.7	Average RAM used in GB	71

List of Figures

2.1	The architecture of a system running virtual machines [5]	7
2.2	Docker vs VMs	8
2.3	Container images	10
2.4	Docker Layers	11
2.5	Docker Compose	12
2.6	Docker Swarm	12
2.7	List of Kubernetes Tools [21]	14
2.8	Different services in the cloud ecosystem [24]	15
2.9	Advantages to a cloud-based approach [31]	17
2.10	A stack representing the MAM4PRO architecture	18
2.11	Detailed method of operation of a media workflow	20
2.12	A graph of the architecture performing simple video and audio operations.	21
2.13	MAM4PRO GUI	24
2.14	Grafana GUI	25
2.15	NetData GUI	26
2.16	Data monitoring architecture	27
3.1	Use case representing a media workflow of a live event	30
3.2	Container cluster creation	34
3.3	Deploy to the cloud	35
3.4	Media flow through the proposed solution	36

3.5	Cloudsail work plan	39
4.1	Overview of our solution	42
4.2	Architecture of the container cluster	43
4.3	SCapture container	45
4.4	HOperation container	46
4.5	RPlayout container	47
4.6	RPlayerWeb container	48
4.7	Scalability example	49
4.8	Docker process of container creation	50
4.9	The inheritance of the images	51
4.10	Docker Compose steps to build a cluster	53
4.11	Docker Compose command to initialize the cluster	54
4.12	Master/nodes relation	55
4.13	Final overview of the solution deployment	57
4.14	MAM4PRO log in menu	58
4.15	NetData dashboard example	59
5.1	MAM4PRO workflow example	64
5.2	Stream protocol change workflow	69
5.3	CPU % usage at and idle state	73
5.4	CPU usage Workflow Grav - 2160	73
5.5	RAM usage while applying the Grav - 2160p Workflow	74

Listings

4.1	Dockerfile example	51
4.2	Dockerfile of the SCapture module	52
4.3	Exert of the YAML file used in this project	54

Acronyms

API	Application Programming Interface	SaaS	Software as a service
CLI	Command Line Interface	VPN	Virtual Private Network
MIT	Massachusetts Institute of Technology	CPU	Central Processing Unit
REST	Representational State Transfer	RAM	Random Access Memory
OS	Operative System	OB	Outside Broadcasting
VM	Virtual Machine	GUI	Graphical User Interface
AWS	Amazon Web Services	AMQP	Advanced Message Queuing Protocol
MOG	MOG Technologies LDA	CMAM4PRO	Cloud MAM4PRO
CTSS	Compatible Time Sharing System	SSD	Solid State Drive
IaaS	Infrastructure as a service	HDD	Hard Disk Drive
PaaS	Platform as a service	SDI	Serial Digital Interface

Chapter 1

Introduction

This chapter provides the introduction to this thesis and it contains: the context of this project, the motivation behind this work, a description of the objectives to achieve and the relation with the European project Cloudsail.

1.1 Context

The cloud technology is a growing market that is based on the leasing of hardware, which is made possible by dividing it into smaller virtual partitions. In the computational paradigm, one calls virtualization [28] to the method used to create these partitions. This concept first appeared in 1959 when the British computer scientist Christopher Strachey published a paper about the potentialities of sharing big and fast computers [32]. However, the creation of what we today call virtualization begun in the '60s where the need to share computers was very high, and at that time a computer was a technology that was extremely expensive and occupied a lot of space. For example, a single computer could be as big as an entire room and companies did not have the financial ability to buy one computer per employee. At this time the first time-sharing concepts emerged, which are considered to be the first steps to virtualization. The first time-sharing algorithms were as simple as making the computer change between tasks periodically. The computers changed between tasks so fast that the users said it was like the machine was not shared at all.

Those methods evolved and virtualization arose, nowadays, one of the methods most used by the cloud providers employs what is called Virtual Machines (VMs), which is a process that emulates one or more machines inside a single physical one. In this context, the physical machine is called the host and the simulated machines are called guests. Besides the VMs other methods exist that provide virtualization, one of the most recent and popular is containerization. Which is the process called to the creation of containers, standard units of software that package code and all its dependencies, libraries, and binaries so that the application runs quickly, reliably, and predictably from one computing environment to another.

Despite the promising evolution of these technologies, several industries are still very traditional and do not seize the advantages inherent with it. For example, in the media broadcast industry, the live broadcasting of events is done using the same physical resources that were used before the appearance of this technology.

1.2 Motivation

Nowadays, there are several businesses that require media processing services, the most important being the broadcasting industry. The number of different kind of content creators that need these services is growing everyday. As a consequence, more flexible solutions are required to fulfill their needs. A flexible approach is able to adapt to better suit each user specific needs. The current solutions do not provide the required flexibility. For example, the traditional content creators often deal with operations that demand huge computational power and therefore require expensive hardware. Even if in some cases the hardware is only required sporadically, they still have to buy that hardware, meaning the costs are the same for a sporadic and recurrent user. Furthermore, in the use case of live broadcasting, there is a inherent logistic complexity to the deployment of this hardware. First of all, qualified human resources are required at event's venue, which can be very complicated if they are also needed in the broadcast company headquarters. Then, a well prepared and configured workspace is required for the deployment of this kind of software. The recent emerged amateur content creators, such as "YouTubers" and "influencers", also gave origin to new use cases for the media services. These use cases have some requirements that the current services are unable to provide, such as being portable, cost-effective, scalable and easy to obtain.

Cloud technology and the use of containers seems to be a promising approach to provide a solution with properties such as scalability, easier logistics implementation, cost-effectiveness, and portability. This will enable the media services to expand the use cases they support with the traditional content creators, offering them a new way of processing media, as well as support the new type of creators.

1.3 Objectives

The main goal of this project is to develop a flexible and scalable cloud-based virtualized media management solution. This solution must be able to fulfill the current needs of the media market which are not provided by the existing solutions. To reach the above-mentioned goal, the following objectives were defined:

- Develop a study of the state of the art in virtualization with a specific focus on containers.
- Identify the requirements of the solution.

- Apply the container technology (using Docker) to media services, creating a cluster of containers as a result, and, in this way, to enable its deployment to the cloud.
- After the virtualized solution is created and deployed to the cloud, the next objective is to benchmark different cloud providers. Nowadays there are many cloud services offered in the market by different providers, but the major providers are the Amazon Web Services (AWS), Microsoft Azure, and Google Cloud which will be the ones used in this project besides MOG Technologies LDA (MOG) own private cloud.

The end goal is to create a new cloud-based media management tool which provides the media industry with a improved solution capable of fulfill demands that the current approaches do not.

1.4 Cloundsail project

This dissertation is enclosed in a bigger European project called Cloundsail. Cloundsail is an acronym for cloud with Self Administering Intelligence Logic and is a project being developed by **MOG** and two other companies: OvationData Ltd and Inca Cloud, both of them being from the United Kingdom.

The aim of the Cloundsail project is to maximize ingest speed and minimize processing time when working with clouds. This will be achieved by providing a single interface that gathers multiple cloud facilities and enables a dynamic and seamless switch between cloud providers while also providing its users with an optimal choice based on prices and performance. This will also improve cloud utilization and get greater levels of efficiency with any given hardware. The project of this thesis is intended to provide **MOG** with a cloud-based solution of a media workflow. Thus giving Cloundsail the component needed to test the deployment of a service using their platform.

1.5 Document organization

This dissertation is organized as follows, chapter two provides an overview of the state of the art. The third chapter explains the requirements for this project and introduces the proposed solution. The fourth chapter details the process of development, providing a walkthrough of the steps taken to build the cloud-based services. The fifth chapter focuses on the tests performed on the solution. It explores the correct deployment of the solution and measures the performance of the cloud providers, enabling their performance benchmark. Finally, on chapter six the outcome of this project is presented, there is also a brief look into the future work that can be done with this project.

Chapter 2

State of the art

In this chapter, there is an exploration of the current state of the art for the virtualization methods, more specifically the containers. We explore the history of virtualization, the needs behind it and its evolution until today. There is also an introduction to the cloud technology, explaining the different services offered by the public cloud providers. We also dive into the advantages of choosing the cloud as a platform to build upon. This is followed by a description of MAM4PRO, offering a description of how it works and the architecture of the software. Always considering that this is a proprietary software so naturally there is a limit for what we can include in this thesis. The last part of this chapter covers monitoring software used to benchmark the solution. This software is crucial in order to test the end product of this project and analyze its behavior.

2.1 Virtualization

Nowadays, virtualization is one of the most important areas of the technological industry, and even though its utilization grew in a more significant way in the last years, its development began in the early 60's. The first time-sharing concept was implemented by the Massachusetts Institute of Technology (MIT) with the first patent being filed by Christopher Strachey, the first computation professor at Oxford University [32]. At that time, and given the computational limitations, the idea of sharing a computer was as simple as making it automatically switch among tasks. Every task received a period of time to be performed and, if the processor did not finish it in that time, it would stop that task and start a new one. Only after running through all other tasks, he would resume the initial one. An interesting fact about this method was that the switching was performed so fast by the computers that the scientists, at the time, said that it seemed like the computer was not shared at all. This methodology is still in use today by the Operative System (OS) schedulers [23] and is called multi programming [20]. One of the first computers to have a time-sharing system deployed was the IBM 709 [22] that ran the Compatible Time Sharing System (CTSS) developed by the MIT [26].

With time, the virtualization methods evolved, and the ability to turn a physical resource into multiple virtual ones got more demand. But, when computer hardware improved significantly in terms of performance, size, and cost, the need for sharing a machine dropped. Computers got so cheap and powerful that companies could afford to buy a workstation for every single worker, and sharing resources started to lose the big meaning they once had. Nevertheless, the same reason that took the need for virtualization also gave it a new purpose. Because as the price of a computer got cheaper and cheaper, the number of internet-based services grew. And virtualization is a primary tool when working with hardware that hosts internet-based services. The more companies and people had computers the more internet based services grew in need and diversity. Nowadays, the internet is easily available to almost everyone and everywhere. For instance, in 2010 the number of registered websites was around 206 thousand, while in the year 2020 that number was 1.2 billion [14]. In fact, the number of new Websites increases exponentially every year. The rise of these and other internet-based services led to an increase on the need to host those services. A new market appeared, based on the lease of the hardware running those services. Hardware leasing can be done by applying virtualization technology to physical hardware, creating multiple virtual instances that can be individually leased. But this hardware was not cheap, while personal computers and workstations were getting cheaper this hardware was still too expensive to some companies. Since big companies had the economic means to buy this hardware, and host their internet-based services, they started leasing the unused capacity of the hardware to smaller companies, or individuals.

The market of leasing hardware capacity created a whole new business. That would be the beginning of the cloud data centers that many companies own today and that consist of a warehouse filled with hardware to lease. This market is called Cloud computing and offers a high number of different operations within the lease of hardware capacity ideology. Cloud computing fulfills many purposes like permanent storage, machine learning pipelines, or server hosting.

2.2 Virtual Machines

Virtual Machines (VMs) are an emulation of an entire computer system, from the OS to the libraries and applications. A VM can emulate most environments, enabling the possibility to run multiple separate environments on the same hardware [30].

The OS and their applications share the hardware resources from a single host server or a pool of host servers. The VMs always need a manager to create and run them, called hypervisor, which is located between the VMs and the hardware (see Figure 2.1). This hypervisor can either be a firmware, software or hardware.

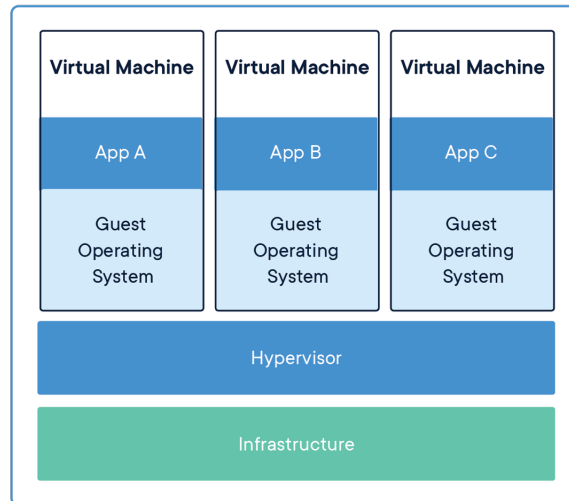


Figure 2.1: The architecture of a system running virtual machines [5]

The VMs have been the go-to technology to virtualize hardware in the past decades and are the main tool for cloud computing services, IT departments, and freelancers as a way to lower costs and increase the efficiency of their hardware.

These machines provide several advantages, some of them are:

- Isolation - The VMs provide isolation between other VMs and the host machine. This is very helpful in many scenarios, for example, when testing potentially infected files. In this case, if a virus infects the system, it will only infect the VMs and not the host system.
- Encapsulation - They provide encapsulation and the software inside the VMs can be controlled and modified. This attribute is helpful when developing software. In this use case, the VM is used as a testing environment for new software versions that are not ready for a stable release.
- Diversity - With VMs it is possible to run multiple OS in the same physical machine, sharing its resources, and without the need to dual boot or replace the host OS.
- Scalability - By using VMs it is possible to increase the number of machines performing a task or increase the capacity of an existing VM to better suit the needs of the user. In a company, for example, they can help in the scaling process alongside with the expansion of their business. So the number of VMs can grow in relation to the growth of the business.

2.3 Containers

As the development of virtualization advanced a new way of virtualization emerged, the containers. Containers are a standard unit of software that encapsulates all the code, dependencies, libraries, and binaries needed to execute the application within the container. What this means

is that containers are an isolated environment where we can deploy a single application without needing an **OS**. A given container can run an application or even just a small task, as long as we provide the container with the code for that application/task and all its dependencies.

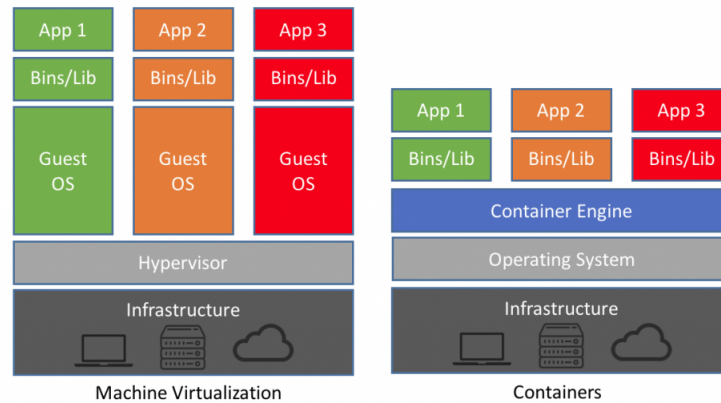


Figure 2.2: VMs vs Containers [7]

The containers, like **VMs**, run in an isolated way within the **OS** but, unlike **VMs**, they are not completely isolated since they share the **OS** Kernel. The comparison seen in Figure 2.2 shows that the **VMs** always require an **OS** while the containers can host an **OS** or not. This enables containers to just run an specific application or feature from an application. But, without an **OS**, the containers do not have a kernel, and since the kernel is essential to perform any task, they use the host machine kernel.

In the containers paradigm, the hypervisor that creates and manages **VMs** is replaced with the container engine. The container engine sits above the host **OS** and acts as a bridge between the containers and the kernel.

This less abstraction also offers some disadvantages, for example in the use case stated before of testing potential infected files. Containers can not offer the full isolation that **VMs** do, so they are not suited for this use case. Nevertheless, the containers also have advantages when compared with the **VMs** [18]. They are easier to configure, faster to create and deploy, require less computational power from the host and are extremely portable. But their main advantage is the ability to run, the application they contain, in a predictable way. Since they are almost completely abstract to what **OS** the host machine is running, it is irrelevant what kind of libraries and dependencies the host does or does not have. So the software inside the containers will behave the same across different hosts. This happens because containers encapsulate all their needs (like libraries and dependencies) to run an application, inside them. Therefore, a container has everything needed to properly run the desired software and is unaffected by the host.

In software development, the predictable behavior is significantly important since it keeps the application running stable and, as expected, through the entire process of developing, testing, and deploying. This attribute is also helpful when the software needs to be distributed by ensuring

that it will work as predicted in every computer where it is deployed. Since the containers share the kernel, they are able to communicate with processes running in the host. This allows the software inside the containers to access and modify information from the host and vice-versa. In some use cases this presents an advantage over VMs.

Moreover, they can also communicate with other containers in the same host through networks created by the container engine. This characteristic of containers opens the door to the possibility of distributing an application in multiple containers (called a cluster) instead of just one. This provides easy scalability, in terms of computational power, to the containerized software. It also allows the deployment to follow a modular architecture that is based on separating the multiple features of an application into modules (containers in this case). The modular approach provides an advantage in identifying and correcting bugs, without having to replace the entire application, and only affecting the containers that need to be fixed. Another advantage of this modular approach is that it provides a new way to scale the solution by allowing to scale just a single container inside a cluster and amplify the capacities of a certain feature. The ability to launch copies of containers is also helpful when updating. It makes possible to update a single container without affecting the entire solution. That can be done by launching new container(s) with the updated code and features, in the place of the old container(s) while keeping the remaining features of the solution working.

2.4 Docker

Docker is a company that developed an entire system around containers [1], and transformed the container ecosystem by developing and providing tools for a better and simpler creation of containers. They developed a container engine that reads through files, called DockerFiles, in order to build an image of a container. One container can be created from one or more images. And from one image we can launch as many containers as we want.

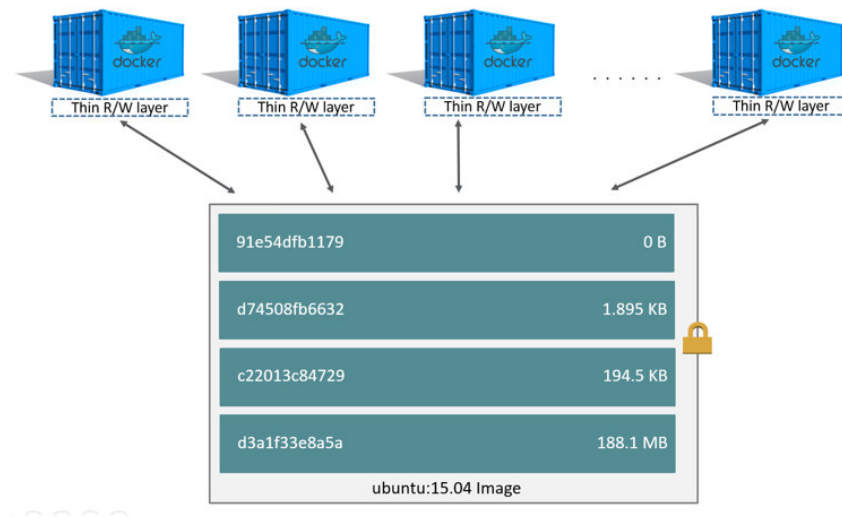


Figure 2.3: Multiple containers sharing the same Ubuntu base image [4]

The process of building an image is done in what is called a layered process: for each instruction in the Dockerfile, the Docker engine will create a layer in the image containing the output of that instruction. As seen in Figure 2.3, images are a set of layers that gather everything needed (binaries, libraries, third-party applications) to deploy an application. They can be defined as a read-only template with detailed instructions for Docker to build a container. It is important to notice that one application might need multiple stacked images to function properly. Moreover, the last layer created by the Docker engine is a writable layer unique for that container where the container performs write/read operations during its run time. Figure 2.3 shows that multiple containers can share the same base image, but each container has its own writable layer and the data in this layer is deleted after a container is destroyed.

Docker offers a tool called Docker Hub for image sharing. This hub is Docker's registry for images and there are millions of images[33] publicly available in it. The software companies can create images for their software and provide them to the public through this hub. They can also be private images and only provide access to users with the right credentials.

Docker is a client-server application [27] based in 3 main components:

- A server in a daemon process.
- A Representational State Transfer (**REST**) Application Programming Interface (**API**) which indicates the interfaces that the programs can use to communicate to, and instruct, the daemon.
- A Command Line Interface (**CLI**) client.

The **CLI** uses the **REST API** to control and interact with the daemon through commands, and the daemon creates the multiple components of Docker such as images, containers, networks, and volumes.

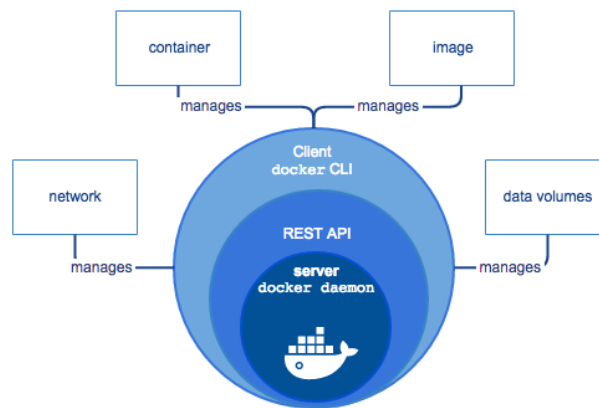


Figure 2.4: The multiple layers of the Docker architecture [2]

Figure 2.4 represents a simplified version of the Docker main components hierarchy. The **REST API** receives requests and forwards them to the daemon that executes them. The daemon can communicate with other daemons on the host machine to manage Docker services that might need intervention from external daemons or processes. The Docker Client is the tool used to interact with the Docker ecosystem and is this client that sends the requests to the API.

2.5 Orchestration

One of the biggest advantages of containers is the ability to deploy an application in multiple containers. This is helpful in many different ways but it also adds complexity to the management of the application. In order to help in the management aspect of multi-container applications, Docker has developed a tool called Docker Compose. Compose is a service based on YAML files, and just like the images mentioned before, the YAML files act as a template that contains all the instructions needed for Compose to create and deploy a cluster of containers (see Figure 2.5).

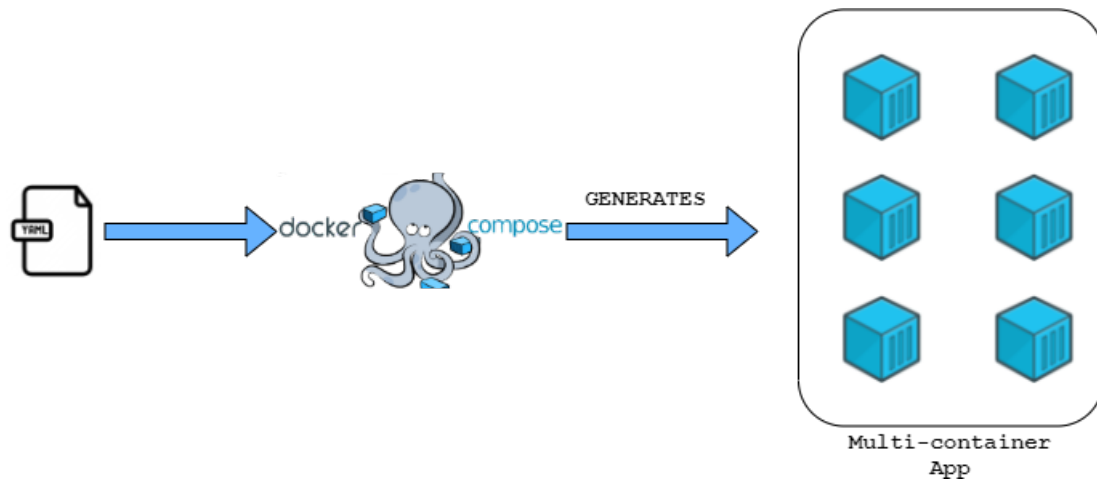


Figure 2.5: Docker Compose

Figure 2.5 illustrates, in a simplified way, the steps of creating a cluster with Docker Compose. Compose makes the process of running and managing multi-container applications easier because it gathers in a single file the entire information about persistent volumes, networks, and images needed by the cluster. Thanks to Compose it is possible to deploy a cluster of containers within seconds.

Docker also develops a tool called Swarm that manages multiple machines running Docker. When we get multiple machines running containers, whether they are single containers or clusters, and they are working together in a cluster of machines it is called a swarm.

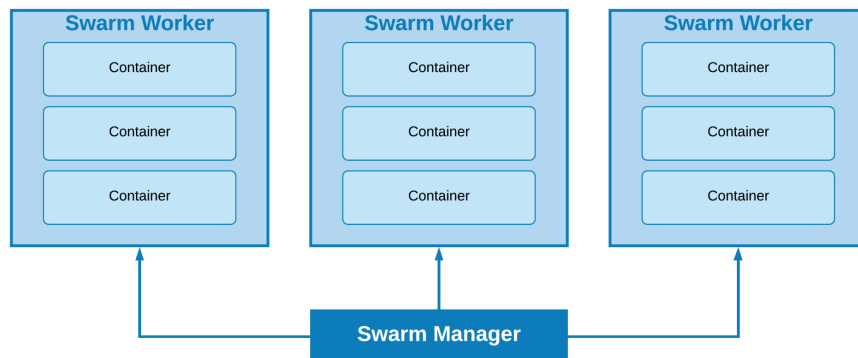


Figure 2.6: Docker Swarm mode

That cluster of machines can be composed of multiple physical machines, VMs or both. The activities in this cluster are controlled by a swarm manager and the machines that compose the swarm are the workers as can be seen in Figure 2.6. In this swarm mode of Docker, we can run all the regular commands and operations but they will be executed by a cluster of machines instead of just one.

2.5.1 Kubernetes

There are other software options capable of this management of clusters and the most popular is Kubernetes. Kubernetes is an open-source platform, developed by Google, that automates many of the manual processes required to create and manage a cluster of containers. Kubernetes enables managing clusters across multiple clouds, whether they are public, private, or hybrid, as well as physical machines. Basically providing both the features of Docker Compose and Docker Swarm all in one place. The Kubernetes popularity is mainly due to the advantages that this technology brings when working with cloud-native applications. Some of those advantages are: rapid scaling, compatibility, and simple methods to manage and deploy the clusters in the public cloud providers. By using Kubernetes it is possible to [17]:

- Orchestrate containers across multiple hosts, whether they are in a cloud, a physical machine, or a VM.
- Control and automate deployments and updates.
- Mount and add/remove storage to run stateful applications.
- Use a set of tools that automate functions to ensure that the applications keep running has expected. Some of these tools are: auto-placement, auto-restart, autoscaling, and auto-replication.



Figure 2.7: List of Kubernetes Tools [21]

Figure 2.7 represents the multiple services created by the Kubernetes software or third-party software and that Kubernetes offers direct compatibility. They are organized by different categories, which are also explained in the Figure 2.7. These services are used to work with the different aspects of managing a cluster of containers. From the deployment, to management, monitoring, and security of the clusters, Kubernetes offers multiple tools to enhance its purpose.

2.6 Cloud technology

The technology provided by AWS, Google Cloud, Microsoft Azure, and many other cloud providers, is based on a cluster of machines located in a data center that are accessible via internet and serve as loan machines to perform actions such as compute certain tasks, deploy servers or data storage [25]. The services offered by the cloud ecosystem can be divided in three main service models: Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS) [29]. These three categories are related to each other as seen in

Figure 2.8.

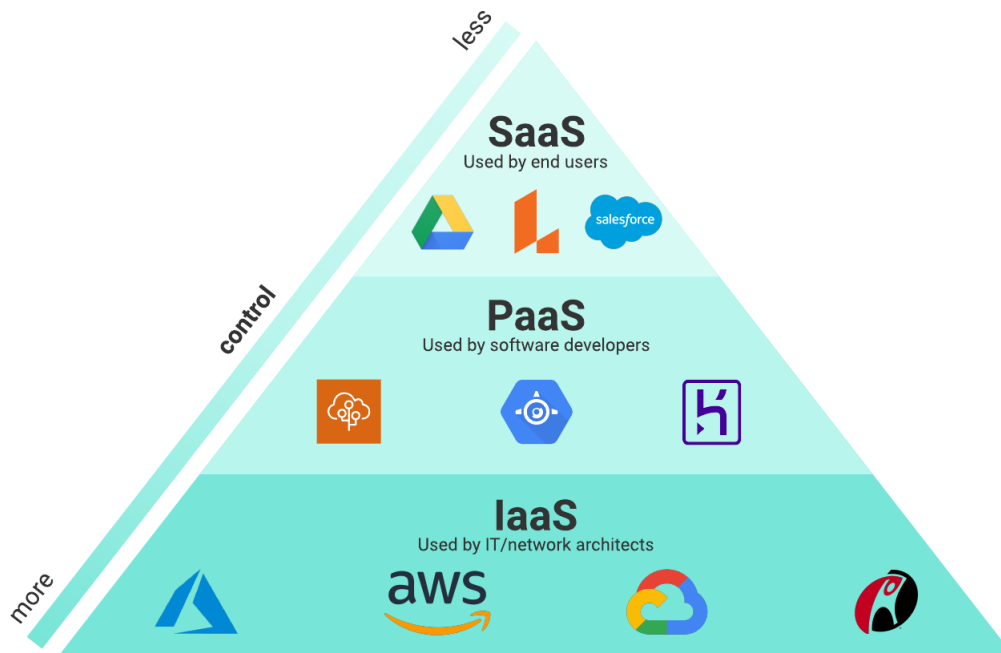


Figure 2.8: Different services in the cloud ecosystem [24]

IaaS is the most common service model, it offers the fundamental tools in terms of virtual servers, networking, operating systems, and data storage. This is helpful when companies need a cost-effective IT solution to support their business growth, and since this is a "pay for what you use" system, it provides full flexibility to the users. This is the service model that we often associate with the cloud services such as AWS, Google Cloud, and Microsoft Azure. In this service, there is a complete control of the resources and their configuration. **PaaS** represents cloud services as a platform, with the main purpose being to provide a framework for developers to build their applications upon. All servers, storage, networking, and configurations are of the cloud provider's responsibility leaving the management of the applications to the providers. It offers less control than **IaaS**, since **IaaS** offers a full customization experience. **SaaS** is the service model that engulfs all the cloud application services and which represents the most commonly used option in the cloud. These services are used to deploy applications on the cloud to distribute them through the web. The control and configuration are limited to only a few basic aspects, the reason for this is to provide an easy approach to deploy and scale an application.

2.6.1 Cloud architecture

The architecture of the cloud product has two main components: the user interface platform, used to access and configure the resources in the data center, and the physical data center. The user platform can be as simple as a website, or more complex like a dedicated program or mobile

application. A data center is a physical place where all the machines that compose the cloud system are located. Usually, these data centers have a huge amount of hardware running and available at any given time. In this way, it is possible to access a computational power that is, by far, greater than the one offered by a personal computer or a single workstation. In these data centers there is a central server that administers the system, by monitoring traffic and client demands[19], to ensure that it runs smoothly. Cloud computing uses the technique of virtualization for everything, this way a single piece of hardware can be shared by multiple users and its usage is at maximum capacity.

The data centers need to have available at least double the storage that is being used, because in the event of a break down they need to ensure that no data is lost. They do that by following a protection protocol where all the client's information and data is backed up in case a break down occurs.

2.6.2 Cloud computing advantages

The cloud technology presents many advantages when compared to working on a physical machine. Figure 2.9 presents some of the benefits that the clouds offer. Every program that can be executed on a regular computer can also be executed in the cloud. If all the requirements needed to run a program are present in a VM deployed on a cloud, then the software can be deployed. From generic word processing software to video games or video editing software, the cloud offers enough compatibility, scalability, and flexibility to run anything. But, instead of running those programs in a powerful local machine, their computation is done in the cloud, which makes possible to use computers with an older and slower hardware to run demanding programs. This approach brings the hardware cost down on the client side [3] since all they need is a computer terminal with access to the internet and, in some cases, input/output hardware. There is also the advantage of accessing the programs everywhere if an internet connection is available. Finally, and since the cloud can keep our programs running at all times, we can start a task that takes a long time to be completed and have our personal computer available for use while that task is being performed in the cloud.

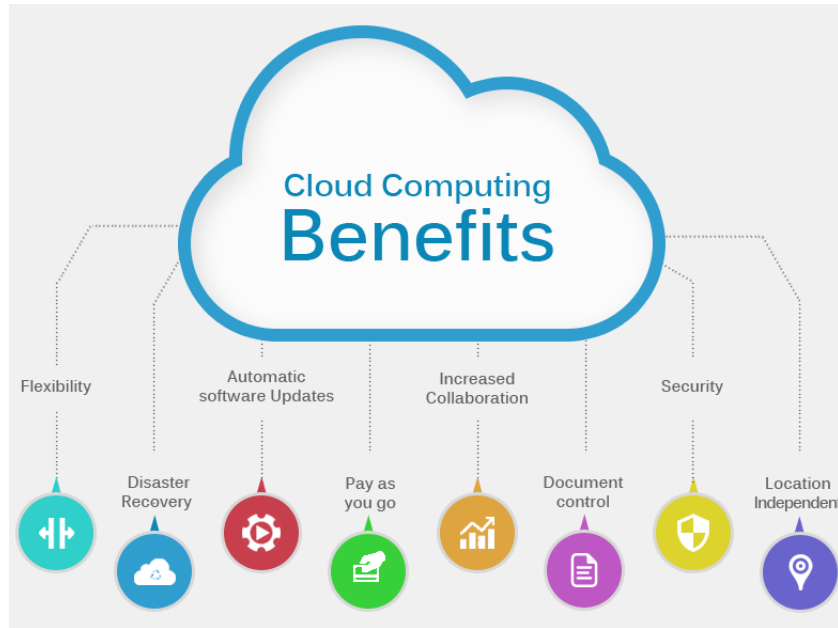


Figure 2.9: Advantages to a cloud-based approach [31]

2.7 MAM4PRO

MAM4PRO is an enterprise-level digital media manager that offers a wide variety of operations with media, such as encode, transcode, decode, and all types of transformations. This solution is developed by **MOG**, which has customers spread all around the world like ABC, AMC, Antena 3, CNN Indonesia, among many others. Solutions like MAM4PRO are needed by many different type of customers that need to manipulate media.

MAM4PRO is a tool that allows content creators the ability to transform raw media into the final product that is delivered to their audiences. Some use cases of MAM4PRO are: Live events like football matches or concerts, movies, television shows, television programs and live programs. MAM4PRO already offers a high number of diverse transformations enabling it to work with many different formats of media. Its constant development ensures that he is always up-to-date in format compatibility and keeps on improving the number and quality of the offered features.

2.7.1 MAM4PRO architecture

MAM4PRO is based on a modular architecture where the features are distributed as independent modules that work as a single unit in a larger connected environment. Modular architecture is a software design based on making each set of related features a module, and each module contains all the necessary dependencies to operate one or more functionalities. This approach brings a lot of advantages from a software development point of view, mainly in error detection and code re-usability, which results in faster development. This modular approach

also allows to have different installations of MAM4PRO considering the client needs. We can deploy a full installation with all the modules present and available, but we can also install only a portion of the modules, leaving out unnecessary modules for a certain task. This flexibility helps lowering the size of the program and provides faster and simpler installations.

The modules can be divided into three main categories, the Graphical User Interface (**GUI**), the managing service composed by the MCore module and the media processing services. The MCore module is the responsible for managing all the other modules. Therefore it is responsible for receiving the requests from the **GUI** and, as a consequence, order the media processing services to perform those requests. This module is the center of the architecture, he is the only module able to order the processing services to start and stop their tasks. Succinctly he is a manager of user requests and media processing modules. The communication between the MCore and the other modules is made through a third-party message queue software called RabbitMQ. The MCore acts as a broker which is connected to every module and the modules are connected to each other through the MCore, as represented in Figure 2.10.

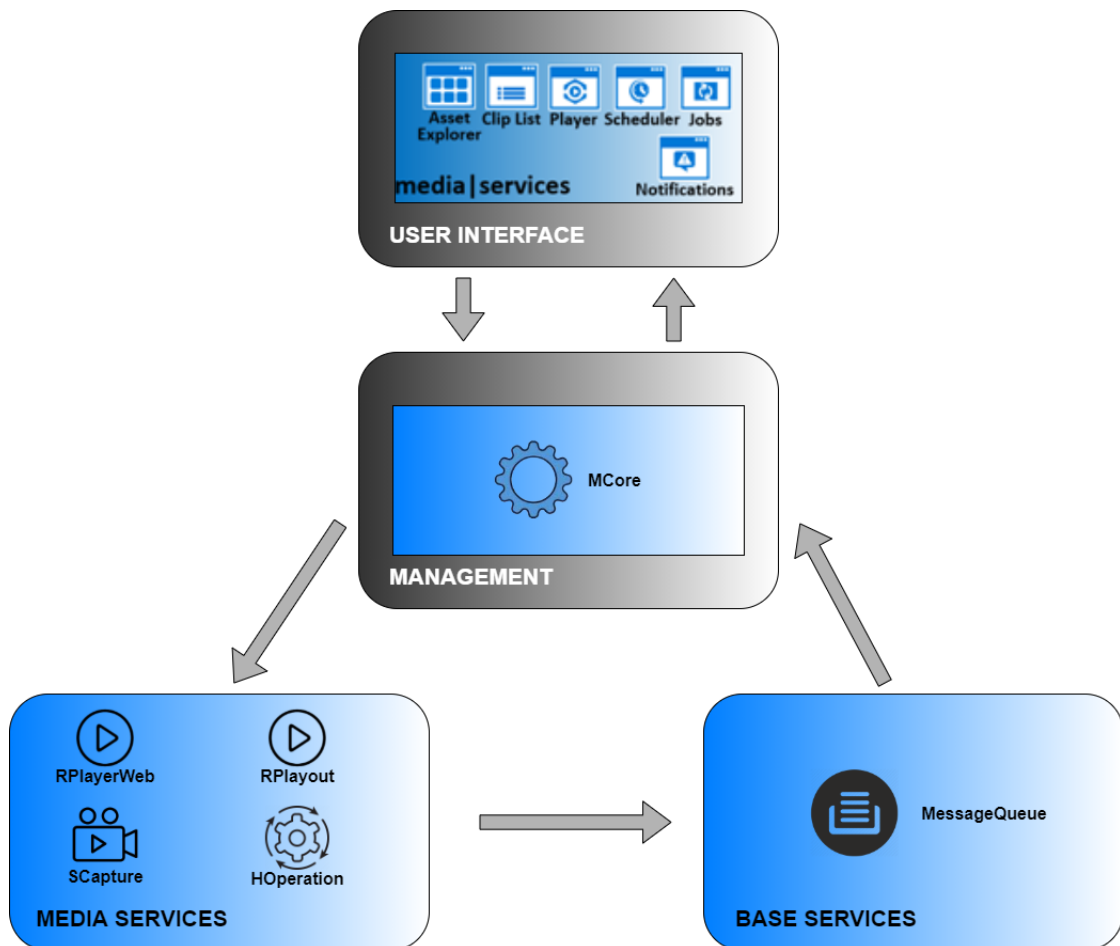


Figure 2.10: A stack representing the MAM4PRO architecture

In Figure 2.10 is possible to visualize the features offered by the different services. In the GUI we have features such as notifications, media players and an asset explorers. In the media services, responsible for processing and transforming media, we have depicted multiple modules of the MAM4PRO architecture. Each of those modules offers, one or more, features, the SCapture module is the responsible for applying the transformations to live streams. The HOperation module performs the same transformations but for pre-recorded media (files instead of live-streams of data). The RPlayerWeb is a player designed to work with pre-recorded files and, given that, he is able to perform seeks on the data and access any part of the video. Unlike the RLayout that is designed to work with live streams, so he does not need to perform seeks on the data since the data is being received in real-time. He only needs to ensure that there is a fluent flow of media output to the user. The MessageQueue module, present on the base services, is composed by two main components, the MAnnouncer and the RabbitMQ message queue software. The MAnnouncer is tasked with receiving announcements from the service modules with information from their tasks, usually a confirmation of a successful task or an error, and redirecting them to the RabbitMQ message queue. That queue is then accessed by the MCore that reads the responses from the media processing modules ensuring the correct management of the solution and redirecting them to the GUI.

Figure 2.11 offers a detailed vision over the interaction between the MCore, SCapture and MessageQueue modules. It is possible to see the logic working behind each module and how the software performs a given task.

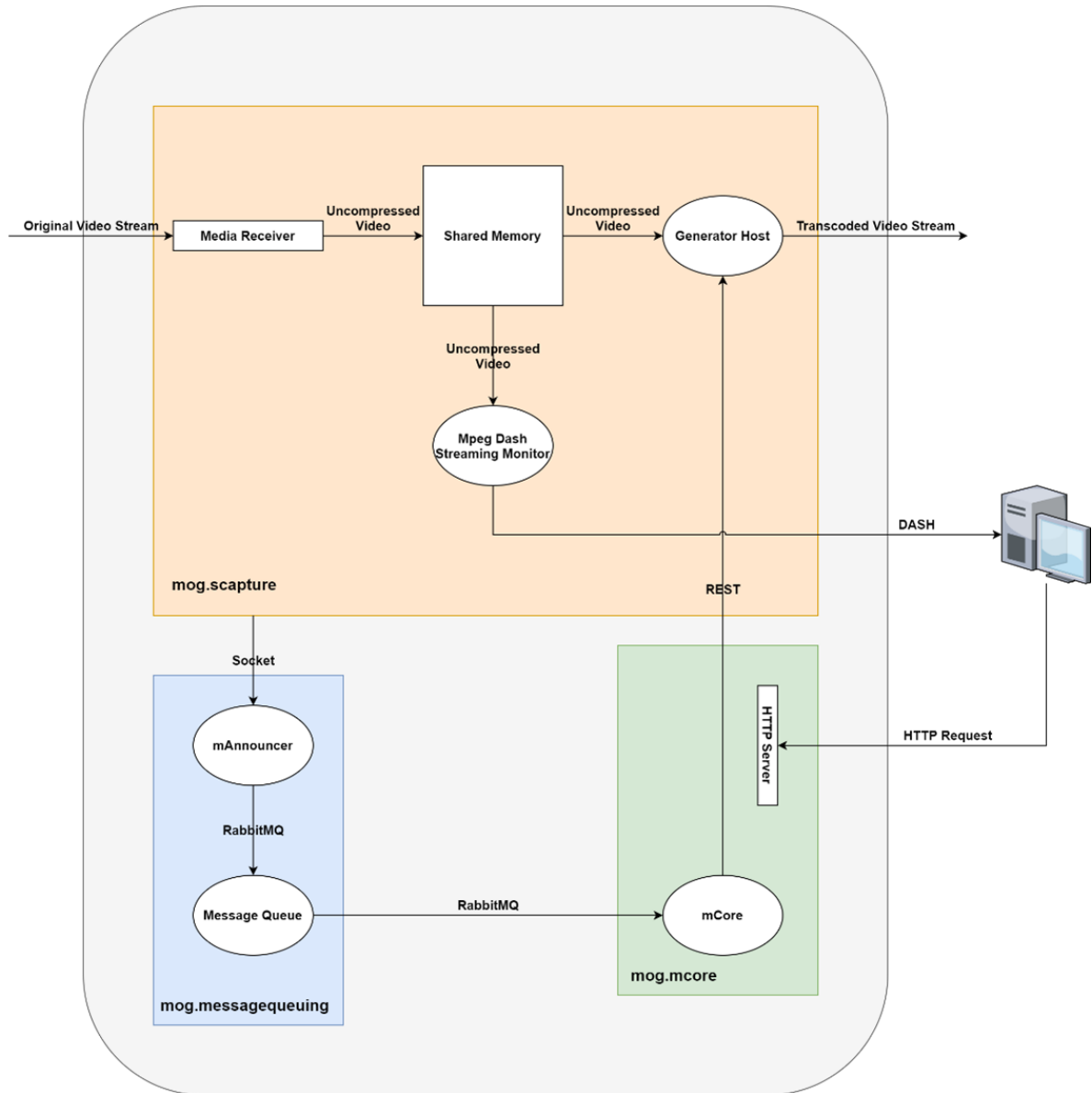


Figure 2.11: Detailed method of operation of a media workflow

Considering the use case of capturing a stream of media, the entrance into this media flow is placed at the SCapture module under the media receiver process. The content of that stream is then saved in the Shared Memory in the same format received, which can be uncompressed video from a media source such as a camera, or compressed video in the desired format. That data can then be displayed to the user using the Mpeg Dash Streaming Monitor process, this allows the user to define transformations to that data which are sent as a request to the MCore HTTP Server. This request is then forwarded to the Generator Host that accesses the data from the Shared Memory and applies the transformations frame by frame. When the final essence is ready for extraction and transmission the SCapture module sends the a message of confirmation via socket to the MAnnouncer feature that lays inside the MessageQueue module. Which then converts and stores that message in a RabbitMQ message queue to be available to the MCore.

This allows MCore to learn that the task was successful, this is essential when managing multiple modules performing different tasks that can require a certain order to be followed.

The method used by MAM4PRO to apply transformations to the media is based on a graph structure, as represented in Figure 2.12, where each node applies a desired transformation to the video, audio, or subtitles.

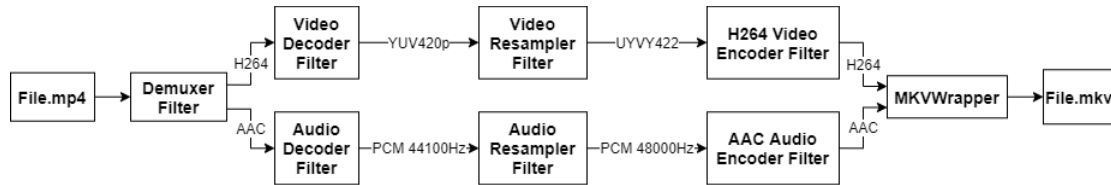


Figure 2.12: A graph of the architecture performing simple video and audio operations.

These nodes are called filters where media passes through and comes out with the desired filter applied. For example, MAM4PRO can receive uncompressed video and pass it through a node where it comes compressed in AV1 format. It can also receive a compressed video in a supported format and use a decoder node to decompress that video. The decompressed video can then be used to apply the desired transformations before compressing again. With the media's industry, most-used codecs and formats, supported by MAM4PRO, the possibilities are huge.

MAM4PRO also uses some third-party libraries for bitstream transformation, encoding and wrapping, besides their own libraries. One of them is FFmpeg, which is a cross-platform solution to record, convert and stream audio and video. Given the market where MOG operates this is the perfect tool for them to explore and to use in a solution like MAM4PRO. FFmpeg is very popular because of its free license, excellent performance, and high variety of compatibility with multiple audio and video formats. The development of this solution started in the year 2000 and there is a release of a new version every 3 months (on average). The developers keep adding support for newer codecs and formats. Nowadays, it is the most popular multimedia framework and is used by a big number of companies that work with huge amounts of media, like Youtube.

MOG also develops and uses its own libraries for some of these operations, and the support for more formats and codecs is a continuous job at the company. It is critical to keep the supported formats and codecs always up-to-date in order to maintain the software as compatible as possible with the media market. The development of their own libraries is a huge advantage that allows MOG to be ready to adapt to a new format or codec as fast as possible.

CODECS	Nº OF PROFILES
MPEG1	1
MPEG2	6
MPEG4	2
H264	53
H265	13
IMX	3
DVCAM	1
DVCPRO	3
DNxHD	12
DNxHR	5
DNxUNCOMPRESSED	1
PRORES	12
XAVC INTRA	7
XAVC LGOP	9
AVCI	3
AVC INTRA	2
AVC LGOP	4
NEWTEK NDI	1
JPEG2000	4
VP8	8
VP9	8
AV1	4
UYVY	1
V210	1
RGBA	1
GIF	1
PNG	1
JPG	1
BMP	1

Table 2.1: Codecs supported by MAM4PRO

As explained before, MAM4PRO applies transformations using filters. Each filter provides a different transformation to the media. The filters can transform multiple fields in the codec, for example the resolution, the library for the transformation or the bitrate are all fields configured in each filter. So there are filters that use the same codec but apply a different transformations. In Table 2.1 is possible to see all the codecs supported by MAM4PRO and also the filters it has based on each codec.

	FORMATS
1	AVID
2	OP1A
3	MP4
4	FILE
5	QUICKTIME
6	MPEG
7	MPEGDASH
8	LLDASH
9	FLV
10	HLS
11	ISMV
12	MPEGTS
13	NDI
14	NDIHX
15	XDCAM
16	XAVC
17	XDCAMEX
18	MATROSKA
19	WEBM
20	MPEGDASH_WEBM
21	MXF_IMF
22	CMAF
23	P2
24	RTP
25	RTSP
26	MOGPLAYLIST
27	MOGPROXY
28	WAV

Table 2.2: Supported formats

In order for MAM4PRO to be a good media manager it needs to support has many formats as possible. This makes the software diverse and an possible option to the media market producers. Table 2.2 provides all the supported formats it has as of writhing of this thesis.

2.7.2 MAM4PRO user interface

MAM4PRO GUI, as seen in Figure 2.13, operates on a HTTP Server and enhances the user experience by simplifying the way the transformations are made to the desired data. This GUI also offers customization to better suit the needs of each customer.

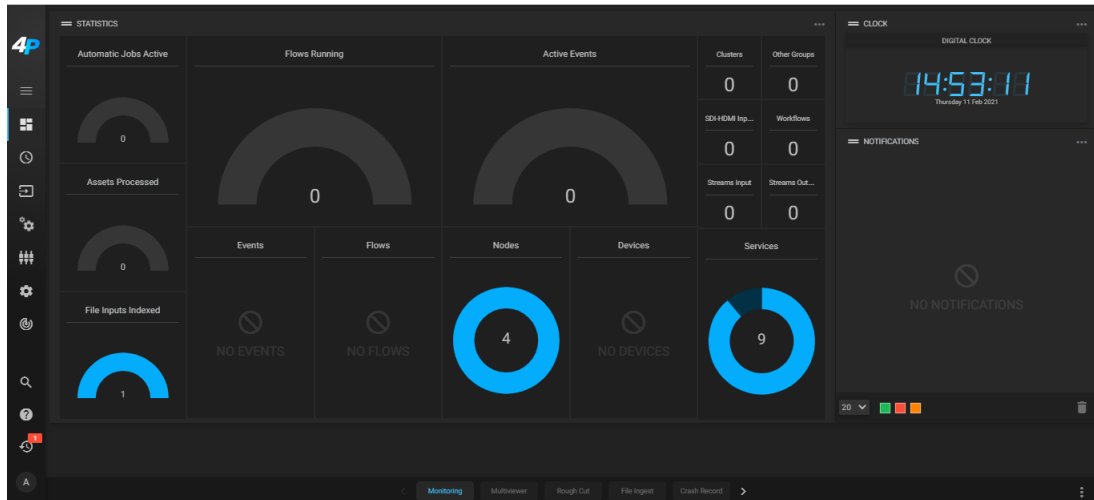


Figure 2.13: MAM4PRO GUI

Figure 2.13 only shows the dashboard of the MAM4PRO. The GUI offers many features separated in the menu, such as workflows configurations, input/output configurations, storage configuration, among many others. The dashboard present on Figure 2.13 is fully customizable with the option to remove the widgets seen and add different ones. MAM4PRO offers a wide variety of widgets allowing for the users to customize the solution to better suit their tasks. MAM4PRO also offers a task scheduler that allows the user to define a day and time to perform a certain task and it can even be defined as a weekly or monthly process.

2.8 Monitoring Software

To test and benchmark the solution it is required to use monitoring software. This type of software provides ways to analyze and study a big number of metrics from the solution resource usage. By studying these metrics, it is possible to infer whether the solution is performing as it should or not, and compare the different public cloud providers with each other and with the private cloud at MOG.

2.8.1 Grafana

Grafana [6] is an open-source solution that provides tools to run data analysis. It does this by pulling metrics from a data source and then presenting that data in useful ways through customizable dashboards. These kind of dashboards allow to easily analyse and track application and user behavior, frequency of errors, and the bottleneck of the resources used.



Figure 2.14: Grafana GUI

As seen in Figure 2.14, the Grafana dashboards can be customized to show information like graphics, histograms, heat maps, and a wide variety of charts.

Grafana gained popularity in recent years and is used by big companies like Bloomberg[8], Uber, and Booking[9]. It offers support to both Windows-based OS and Linux-based OS usage, which is a huge advantage since nowadays Windows servers are as common as Linux servers. It also offers compatibility to most of the data sources used today such as Prometheus[16], Graphite[10], and Influx DB[11]. Although Grafana is open-source, the full solution with all the functionalities, called Enterprise Edition, is paid. They offer a functional but simpler version for free.

2.8.2 NetData

Just like Grafana, NetData is a Linux-based open-source solution that performs data analysis from a wide variety of data sources. NetData can work with a wide variety of data sources such as Postgre SQL[15], MongoDB[12], and MySQL[13]. It can perform the same functions as Grafana, but it is better optimized so it offers better performance when monitoring big amounts of data.

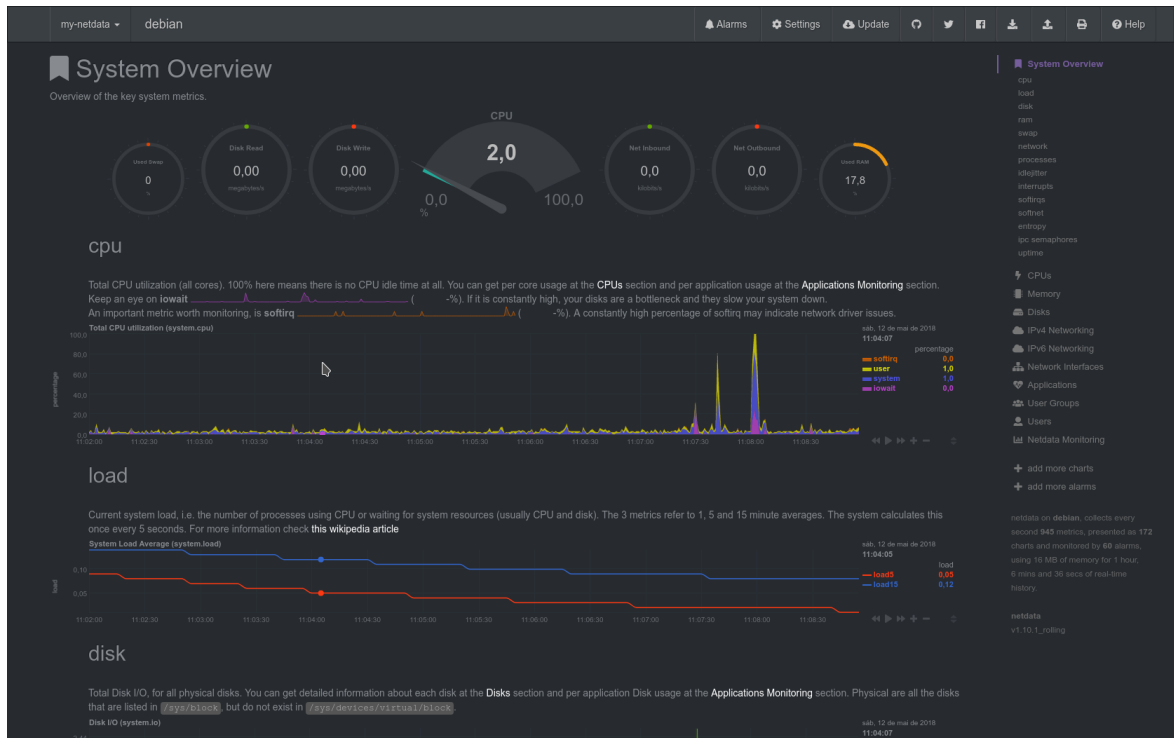


Figure 2.15: NetData GUI

NetData (Figure 2.15) also offers the possibility of customizing the dashboards in order to present the data as needed. Also, the user can define alerts for when a determined resource reaches a certain level. It can also take snapshots from the dashboard by choosing a period to capture the data received and save it as an image. NetData is a free and open-source software, which is a big advantage over Grafana, since it offers all the functionalities of the software for free and without limitations.

2.8.3 Windows server monitoring architecture

Both Grafana and Netdata are tools to display the received data. They do not perform data extraction from the source, they just receive the data and present it. Their advantage is that they support many different data sources and they offer high customization to the way the data is presented. In Figure 2.16 we can see an example of how data is gathered and exposed in a Windows server machine. The WMI Exporter service will expose the data about the machine resources to the port 9182. Afterwards, Prometheus will periodically scrape that data. Netdata and Grafana will send queries to Prometheus asking for the data. Prometheus can be thought as the database where all the data is stored, while Netdata and Grafana are the interfaces for visualization and analysis.

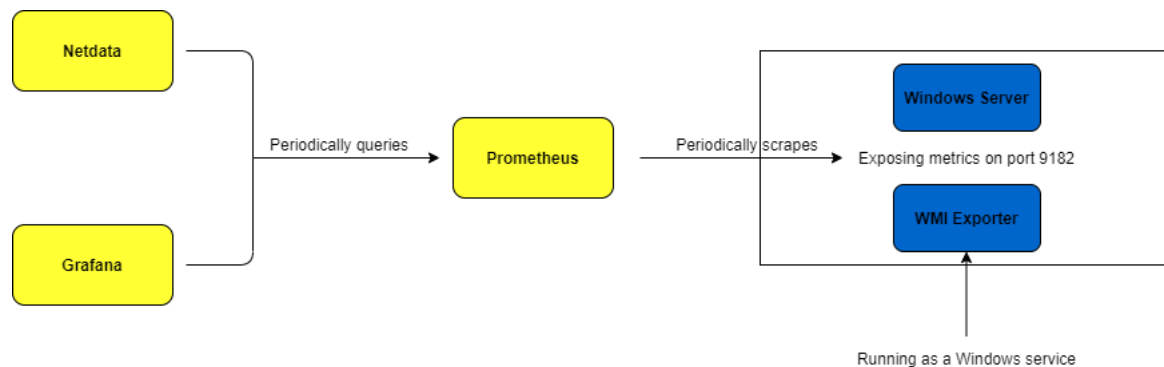


Figure 2.16: Data monitoring architecture

Prometheus is contained within the NETDATA distributions but it can be an independent tool. Although the media solution presented in this work is based on Windows, the chosen software for monitoring the solution and benchmark the different public and private clouds is NetData. This choice is based on the fact that NetData is an open-source software that provides its entire features for free. Grafana also provides an open-source free distribution of its software but it is a very limited option and it requires a paid subscription to unlock its entire features.

Chapter 3

Problem statement and proposed solution

This chapter focuses on the details about the scope of the problem and how our proposed solution aims to solve it. The chapter starts by stating the problem, explaining what makes it a problem and the challenge involved in solving it. The problem statement is followed by information about the different types of users, specifying the different interactions each user has with media processing services. Then the chapter continues by presenting the requirements of this project, which are divided into two categories: functional and system requirements. Afterwards, there is the proposed solution for this project. The architecture of this proposal is presented alongside details on how it will work. There is also a smaller section highlighting the advantages of this approach and how will the users benefit from it. To conclude the chapter there is a section about the integration that this project has with the Cloudsail European project.

3.1 Problem statement

The media industry is still very traditional in the way that the content is prepared and produced. This is especially true when considering live broadcasting. The huge amount of data being processed and the past investment in expensive legacy systems has been a huge obstacle when considering the implementation of new technologies. One of the most important aspects when considering a live media production workflow is the transcoding of the captured content for distribution/editing friendly formats. MAM4PRO, as referred in the previous chapter, is a digital media manager owned by **MOG** which performs this kind of operations.

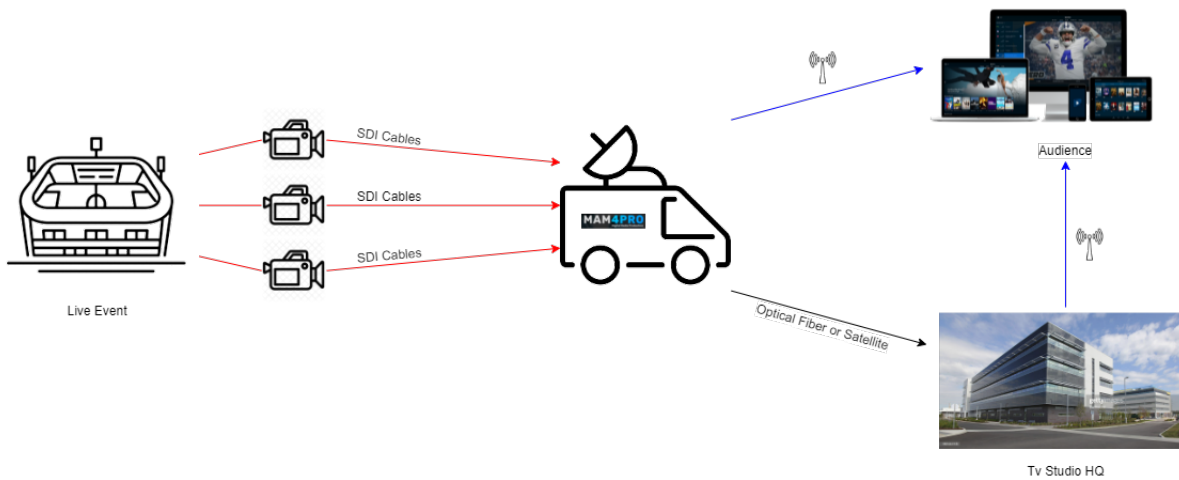


Figure 3.1: Use case representing a media workflow of a live event

In a traditional live event, the common media sources are cameras and microphones, as represented in Figure 3.1. Common examples of such live events are concerts, football matches, and festivals. The captured media is delivered to what are called Outside Broadcasting (OB) vans through Serial Digital Interface (SDI) cables that are directly connected to the media sources. These OB vans are specially built to transport all the necessary equipment to process and edit media, such as MAM4PRO. Therefore, they are, basically, portable television studios. In fact, in some events, all the processing and editing operations are done at the vans which then broadcast the final product. This final product, usually a live stream, is the content consumed by the spectators on their devices. In other cases, the vans only apply some pre-processing and edition operations to the content before transmitting it to a studio. The transmission to the studio is usually done either by satellite or by optical fiber. The final product is created at the studio and from there it is broadcasted to the audience.

The traditional approach to cover a live event, happening outside the TV station, has some drawbacks. For example:

- It does not allow to easily scale the media processing capacity. This is because the higher the amount of media to be processed, the more vans are needed, resulting in higher costs.
- Another issue is the need to deploy the vans at the event, which in some cases might be a complex logistic issue. Notice that, in some cases, the location of the live event is of difficult access and, in other cases, the necessary space, to park the vans in a close enough location to operate, is limited.
- This approach does not allow to have a cost-friendly fail-safe plan for hardware/software errors that might happen. Such a plan can be obtained if the event organizers ensure that they have at least twice the number of initially required vans. In this way, they would have enough vans which can act as backup in case of a failure. However, this is a very expensive solution.

- The live events also need to have the right hardware infrastructures in order to allow the vans to be integrated into the event, otherwise, it is not possible to use them.

In this context, and given the diversity of live events that are nowadays produced, a system capable of exceeding these limitations is needed. More specifically, a system that offers scalability, easier and faster deployment, better error recovery systems and higher compatibility with the current infrastructures. The main challenge of this project is to obtain such a system using virtualization technology.

3.2 Requirements Specifications

Considering the problem previously stated, and the main drawbacks of the traditional approach to cover live events, it is now important to identify the potential users and the set of functionalities and features expected from a solution capable of overcoming such drawbacks.

This section starts by identifying the set of potential users this solution may have. It then details a list of functional requirements, followed by a set of system requirements. In the functional requirements the emphasis is given to the user experience using the proposed solution, while the system requirements focus on the expected features of the solution.

3.2.1 Users Identification

There are two types of users that were identified in this system: the media operators and the administrators of the solution. The administrators can either be someone from the solution provider or from the cloud provider. The media operators can be comprehended by all the companies and individuals that need to work with media processing and editing at an enterprise scale. In this section, a list of some potential users for the end product of this project is presented.

- **Television broadcast companies** that need to perform media transformations to their previously recorded content or to their live content. Currently, **MOG** works with a variety of companies such as Fox, ABC News, AMC, BBC. Clients like these would greatly benefit from the cloud-based solution proposed by this project.
- **Streaming services** that require media transformations in order to deliver the media to the client, or to work on the media, for example, to add subtitles or manipulate the media formats to a more user-friendly version.
- **Internet streamers:** individuals who create content in a wide variety of scenarios, for example: recording themselves playing video games, or creating content about different subjects such as fashion, cars, and movies. The number of streamers rises every year, and in 2020 the number of active streamers on the platform Twitch.tv reached 9.24 million. These numbers show that there is a significant number of customers that fit in the target

market for this cloud-based solution in the streamer community. There are other streaming platforms and it is a market that is expected to keep growing. In this case, a streamer would benefit by using this cloud-based solution to operate the media transformations to his content.

- **Marketing companies** that need to create advertising campaigns and use media content for that purpose, whether it is a visual advertisement like the commercials seen on television, or a song related with a specific product. All the media content production and distribution requires a tool able to perform these media operations.
- **Live events** where a media manager solution is required to provide the tools for media transformations to the content captured and created through the live event. For example, a football game where the content needs to undergo a big number of transformations like cuts, slow motions, zooms, in order to provide higher quality replays. Besides the more usual transformations needed to distribute the content from the source (the cameras in the event) all the way to the audience devices.
- All the companies or individuals that work in a market related with **digital media production**, whether it is audio or music, are potential users of this solution. This solution focuses on media production at an enterprise level but can still be used to perform lighter and simpler tasks. Being cloud-based also helps to increase the number of potential clients since the cloud flexible characteristics offer a higher customizable experience to better suit the user's needs. So, the solution in the cloud will be different considering the needs of the specific user. This kind of approach can accommodate much better the users that do not need an enterprise-scale solution.

3.2.2 Functional requirements

The functional requirements are the expected actions a user should be able to perform when interacting with the solution. These requirements not only assure that the solution does what it proposes, but it also provides a smooth interaction between the user and the interface.

Table 3.1 presents the functional requirements identified for each type of user: media operators or solution administrators.

Type of user	Requirement
Media operator	Select an IP-based live stream as input to the media operations.
Media operator	Select a file as input to the media operations.
Media operator	Select the encoder used during the media processing, amongst those most used by the media market.
Media operator	Select the media format input/output from the most used ones in the market.
Media operator	Configure the encoding settings used by the encoder during the media processing.
Media operator	Store the result of the media operations in a file-based format.
Media operator	Export media from the solution to a storage or broadcast it.
Media operator	Be able to increase/decrease the solution's modules to increase/decrease its processing capacity.
Media operator	Access the service everywhere where a connection to the internet is available.
Administrator	Be able to increase/decrease the capacity of the VM running the solution.

Table 3.1: Functional requirements

3.2.3 System requirements

The system requirements are global characteristics of the system that need to be achieved in order for it to run as expected. By following these requirements the solution proposes to deliver a product with market-ready quality. The identified system requirements are:

- **Scalability** – the solution should be able to scale its resources, up or down, as needed. This is a high priority attribute since the work load may vary in some cases. Scalability is achieved by increasing/decreasing the number of processing modules of the solution.
- **Diversity** – the solution should offer diversity in the media workflows available. A media workflows can be categorized as a pipeline that operates media transformations. These workflows should provide a high diversity of encoders and formats among the most used in the market. This is a very important requirement since different users require different workflows. Another important aspect is the ability of the solution to be deploy in different cloud providers. Ensuring that it is compatible with a diverse set of providers is a critical requirement since different users will probably use different cloud providers.
- **Performance** – taking into account the different user needs the solution needs to perform accordingly. This is a very important aspect since some users will not be eligible to benefit from the solution because of the higher latency. While other users do not require low latency but a require a more demanding resource usage or flexible scalability, making this solution a better tool for them. Although some latency might be constant since the solution is cloud-based and there is inherent latency in the process of sending/receiving data to/from the cloud, the solution should be able to perform the media operations in a reasonable time period.

- **Robustness** – the solution should be robust during its usage. This is achieved by assuring that the solution is error prone, in order to provide that, a periodic monitoring of the container logs and the VMs resources is needed. Doing regular updates is also a procedure that helps to ensure the robustness of the solution. This is a very important aspect to consider because the market where it is situated can be very demanding in the timing of media creation.

3.3 Solution proposal

In the context of live events media coverage, as presented in Section 3.1 and represented in Figure 3.1, the most common approach is to use expensive OB vans to ensure that all the media processing services are available at the event venue. MAM4PRO, as referred before, is a digital media manager that is fit for the enterprise-level usage required in these scenarios. In this context, it will perform all the transcoding operations required for the captured content to be edited and distributed. This product is already well established in the market and, therefore, it seems to be a good starting point to overcome the challenge of developing a cloud-based system which fulfills all the requirements identified in the previous section.

Therefore, given the paradigm of the problem and the requirements of the system, the approach we propose is to apply virtualization methods to MAM4PRO, in order to enable its cloud deployment and, with this, to create a cloud-based media processing service. From this point forward, this new solution will be referred to as Cloud MAM4PRO (CMAM4PRO).

Figure 3.2 depicts, in a simple way, the process to obtain the cluster of containers for our solution. There is a representation of the MAM4PRO architecture stack present in Figure 2.10 on Chapter 2, and after applying the container technology the output will be a container cluster.

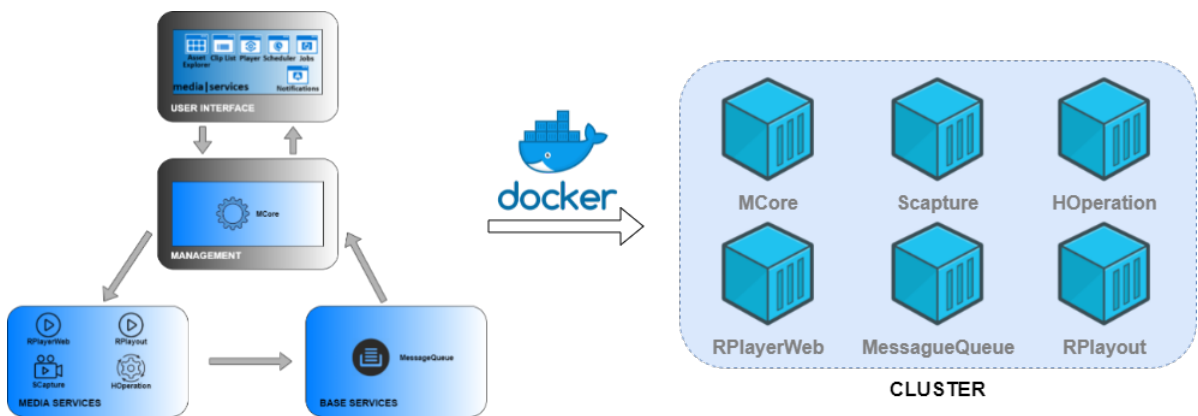


Figure 3.2: Container cluster creation

The resulting container cluster should be a perfect emulation of the containerized software within. The cluster divides in to multiples containers the main modules of the MAM4PRO architecture. This results in a cluster with 6 containers: MCore, SCapture, HOperation,

RPlayerWeb, MessageQueue, RPlayout. Each one of these containers emulates the behaviour of the corresponding module in the MAM4PRO architecture. Therefore, the resulting containers will have the following main functionalities:

- MCore - Central GUI and user access point, task management, database and REST API.
- MessageQueue - Service responsible for ensuring communication between containers.
- SCapture - Process services (transcoding, re-wrapping, metadata, among others) that work with real-time operations (live streams).
- HOperation - Process services (transcoding, re-wrapping, metadata, among others) that work with pre-recorded files (not real-time processing).
- RPlayerWeb - Player server with web monitoring for pre-recorded files.
- RPlayout - Playout solution with support for all common media formats and playlists that works with real-time data (live streams).

After the creation of the cluster the virtualized solution is deployed to the cloud. This operation requires the right configuration of the VMs that host CMAM4PRO.

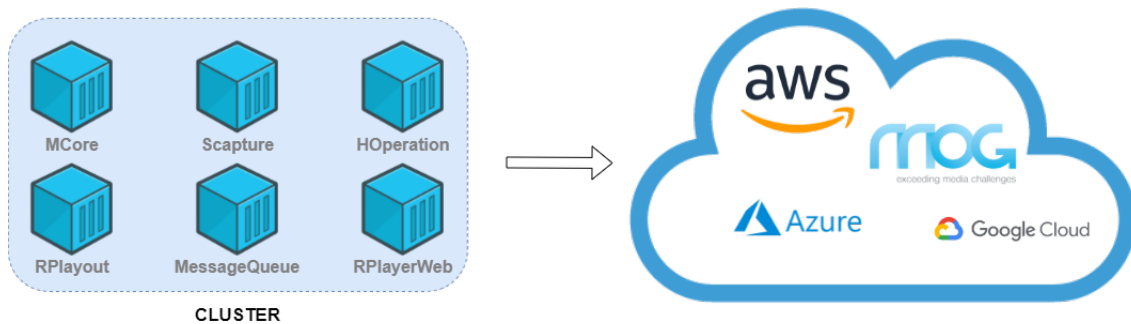


Figure 3.3: Deploy to the cloud

After the cloud deployment, CMAM4PRO is now ready to process media. This approach will change the flow of media from the workflow previously used as an example on Figure 3.1.

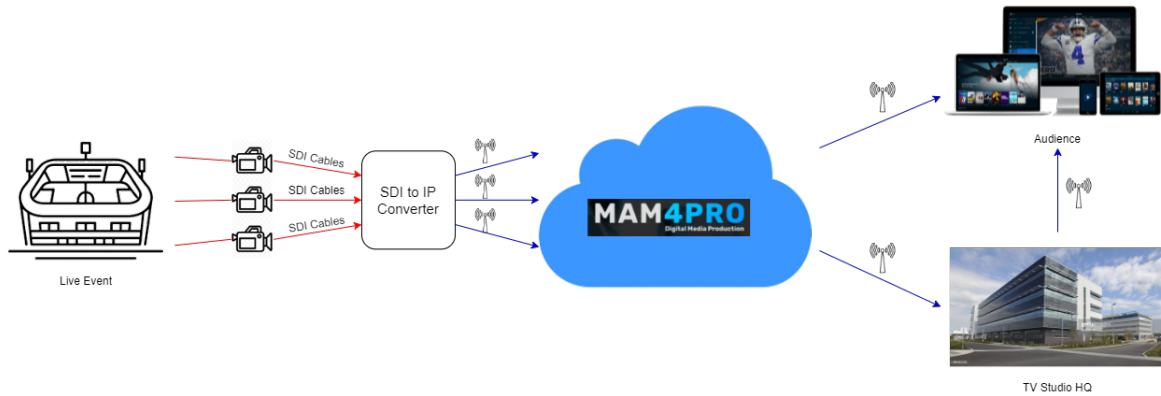


Figure 3.4: Media flow through the proposed solution

Figure 3.4 represents a new media workflow that now uses CMAM4PRO. It is possible to see the flow of media during a cloud transcoding process, from being generated, sent to the cloud, and outputted by the cloud services.

This kind of approach will eliminate the need to have the media processing services running in physical machines inside the OB vans. Nevertheless, for this to be possible, it is necessary, as seen in Figure 3.4, to ensure that an SDI to IP converter is available and connected to the media source, which allows sending the media to the cloud.

In this scenario, the OB van represented in Figure 3.1 could still be used as the SDI to IP converter. The van would supply with the SDI to IP conversion and the transmission to the cloud, where the media processing services would be running. This means that it would not process or edit any of the media, that task would be entirely performed by the services on the cloud. But, as long as there is equipment performing the SDI to IP conversion, there is no need to deploy the OB vans since there are simpler, easier to deploy, and cheaper ways to perform the signal conversion.

3.4 Advantages of the proposed solution

The approach presented in the previous section is expected to provide abstraction from the need to use expensive hardware while profiting from the cloud-based advantages. Migrating this process to the cloud will enable not only an easier deployment of the workflows, in different type of scenarios, but also cost reductions as there is no need for broadcasters to invest in physical on-premises equipment such as the OB vans or similar solutions.

The proposed solution will solve the main issues that the current solutions have. As mentioned before, our solution will be scalable, robust, diverse, and ensure reasonable performance. This is achieved by benefiting from the cloud-based nature that allows us to create a fully scalable solution, able to scale, up or down, the resources, anytime as needed. The cloud also helps in ensuring robustness by providing resources that are ready and available at all times while

ensuring their correct maintenance and operation. The diverse aspect of our solution consists of two different aspects. One of them is by offering a wide number of media operations in a effort to cover the needs of every customer. This aspect is fulfilled by the MAM4PRO software, a well-established and reliable solution that operates media at all levels, from simple jobs up to enterprise-level ones. The other diversity aspect is compatibility with different cloud providers. This will allow the customers to chose the best-fitting provider for their use case. At last, the performance is optimized by using containers as the virtualization method. By developing a container-based solution we pivot on their advantages of being inherently faster and simpler to deploy, easier to manage, faster to detect errors and bugs, easier to update the software contained within, and predictable behavior abstract to the host running them.

Below are presented some examples of how the proposed solution will change the way customers interact with the media services they need:

- **Football games or live events**

The current situation requires that a TV station that wishes to broadcast a live football game has to buy **OB** vans that contain all the required hardware and services to broadcast from the stadium cameras to the spectators' televisions. The **OB** vans supply the services to receive the signal from the cameras on the stadium, edit the images and broadcast them. These vans are very expensive and require human resources to be operated and a physical connection to the cameras in the stadium.

Future scenario using this project would enable that all the work and services offered by the **OB** vans was instead offered by a cluster of containers running on the cloud. In this particular case it would be required that the content from the stadium cameras was sent to the cloud where it would be processed. This eliminates the need to buy and deploy the **OB** vans. It also requires less human resources, so this solution has an economic advantage when compared with the current situation. In the current situation, if there is a need to increase the amount of resources the broadcasting station needs to buy more hardware and allocate more human resources to operate the new hardware. With this work, a more adaptable solution is provided. If more resources are required, then the cloud infrastructure can be configured to meet those needs, scaling up or down as needed.

- **Sporadic resource usage**

In the current situation, a company that needs to use these media services (such as encoding, decoding, wrapping, etc.) needs to buy a set of hardware and software tools. Even if they only plan to use those tools a few times per year, they need to buy and pay the same as a company that uses the same tools multiple times a day.

In the future, they can use the cloud-based solution that better fulfills their needs. In the proposed solution they only pay for what they use, both in terms of time and in terms of computational power. Let's imagine an example where a company called "Portugal Ads" needs to create a TV commercial for another company. To complete this task, the content creators will need to transform their media and, instead of buying the hardware tools, they

can buy a license to access the proposed solution, which can be configured accordingly. This brings advantages since it is cheaper for the "Portugal Ads" company, and also easier, to configure the solution to better suit their needs. Without having computational power in excess, which might be the case in the current situation, the "Portugal Ads" company can lower their costs.

- **Freelancers that work with media (streamers, content creators)**

Current situation. John is a streamer and youtuber that creates live content, as well as videos that he edits and posts on YouTube. John needs to buy a lot of hardware to perform these tasks. He needs cameras, capture cards, decks and other similar equipment that acts as a studio where he can manipulate the content he creates. John records and live streams from his house where all the equipment is. If John wants to visit another country and record that experience, he then needs to wait until he is back home to use his equipment. Only then can he process all that content and post it to his social networks. He also needs powerful enough equipment to perform these media manipulations, since working with media requires a great computational power.

In the future John can benefit from the cloud-based solution proposed in this project. In the example given above (travelling to another country), if John could use the cloud-based solution, that would mean that wherever he was he could use our solution to process and manipulate the content. All that he needs is a connection with the Internet and he will be able to live stream and edit his content, and post it to his social networks. Without having to carry all of his hardware with him, even a small laptop with a low computational power is enough for this use case. This provides an affordable way of managing media, since all the processing is done by the machines in the cloud center.

3.5 Cloundsail integration

The project of this thesis is related to the European project Cloundsail. As mentioned in Section 1.4, its main goal is to maximize cloud resources efficiency, namely in two specific scenarios: Media and Oil & Gas. This will be done by building a platform that is intended to provide easy access to deploy services on multiple cloud providers. This platform will also gather information about the relation between cost/performance for multiple cloud providers.

Figure 3.5 shows an overview of the several components of this project and how they will work together. These components are aligned with the Work Packages (WP) of the project.

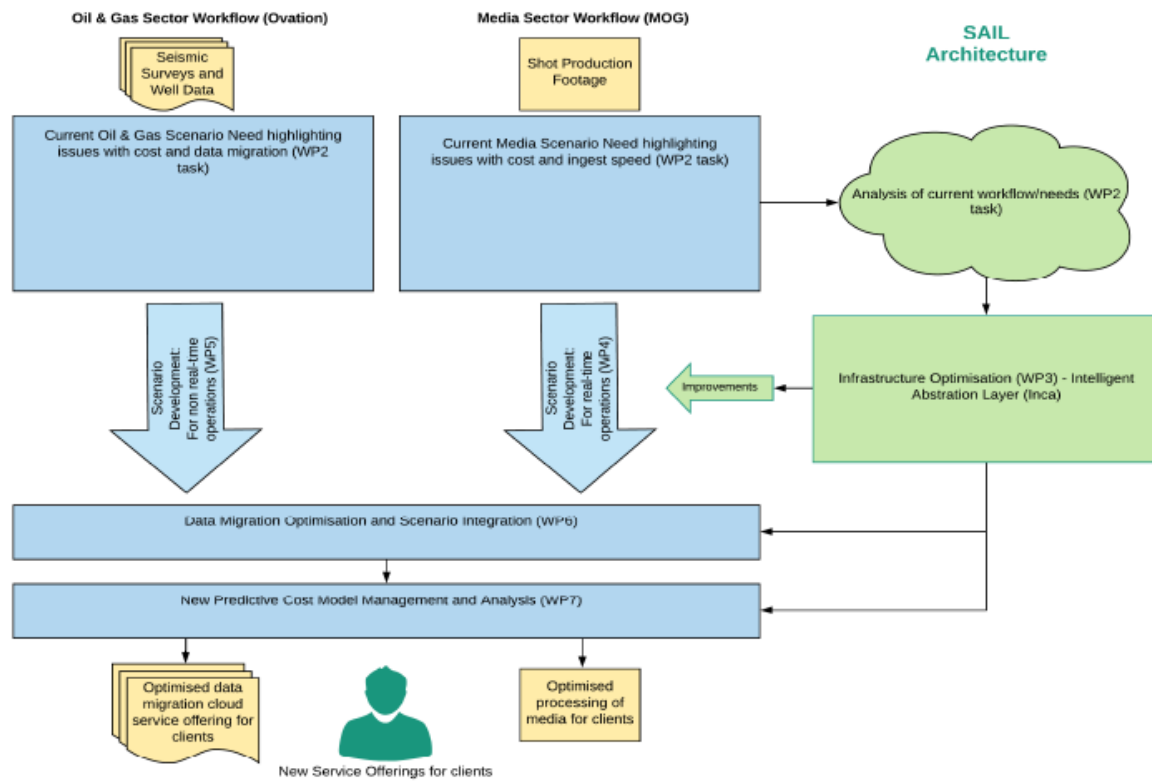


Figure 3.5: Cloudsail work plan

As it can be seen from the figure, there are two scenarios in the project: Media and Oil & Gas. The workflows of these scenarios will be analysed to define the specifications and design the infrastructure (WP2). Then, using the results of the previous analysis, and by doing some research, the infrastructure will be optimized for the two identified scenarios (WP3). Each of these scenarios will be developed for their deployment in the cloud. In this context, our project is responsible for the development of the Broadcast Sector Media Data Scenario (WP 4 in Figure 3.5). Finally, the movement of vast quantities of data into and out of the cloud will be optimised (WP 6) and a new predictive cost model management system will be developed (WP 7), to allow potential users to know what the real cost of their cloud usage, where and why.

Our work will not only act as a bridge between **MOG** and the Cloudsail project, but also as a new development in the media sector. The research and solution developed by our project provide a new and improved system to work with media that fulfils the some of the most critical gaps of the current solutions.

Chapter 4

Solution Development

This chapter explores the development of the proposed approach, starting with an overview of the entire solution followed by a more detailed description of each of its components. The choices made are emphasized as well as the reasoning behind them. More precisely, this chapter offers a walk through the creation of the **CMAM4PRO** solution. From the creation of the cluster of containers to its deployment in the cloud, including the setup of the monitoring tools. This chapter starts by providing an in-depth explanation about the creation of the cluster, how it is managed and what kind of configurations are needed to create the containers that host our media management software. This is followed by a description of the necessary configurations made on the Virtual Machine (**VM**)s hosting our containers. At last, there is an explanation about the configuration and deployment of the monitoring tools used to analyze and study the behavior of the solution, as well as, later, benchmark the different cloud providers' performance.

4.1 Solution overview

This project solution consists of a cloud-based media service capable of processing video, audio, or both. The media source can either be a stream of data (live content) as well as pre-recorded data (not live content). The complete stack of the solution consists of **VM**s hosting our media processing services and a set of monitoring tools. The media processing services are supplied by the MAM4PRO, a media manager from **MOG**. The major challenge of this project was to apply the container technology to MAM4PRO, which was necessary in order to deploy it to the cloud. As explained before, MAM4PRO is divided into modules so, the first step when creating its' containers was to plan how the division of the MAM4PRO modules to containers would occur. After that, there was a configuration and creation of the cluster, and after it was operational, the hosting cloud environment was created. The cloud providers chosen in this project were the three leading companies in the market of cloud computing services, Amazon, Google, and Microsoft. Besides those three, there were also machines from **MOG** that represented a private cloud. The next step was to deploy the cluster to the virtual machines on the cloud. At last, there was the configuration of the monitoring stack. This monitoring stack is composed

of data extraction and visualization tools. The monitoring tools used were the WMI-Exporter for Windows Server (data extraction) and NETDATA (data visualization).

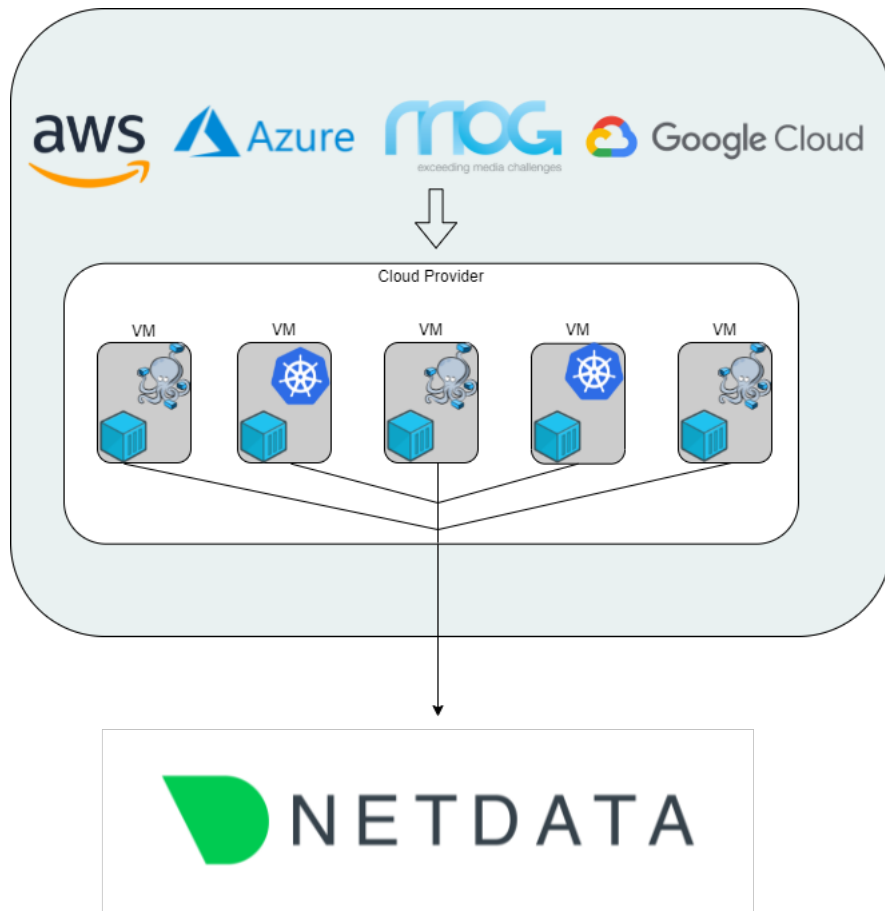


Figure 4.1: Overview of our solution

Figure 4.1 represents the multiple components of this projects' solution. Starting with the logos of the cloud providers that represent the cloud hosting the media processing services. Followed by the media services, represented by the cluster of containers inside the VMs. To be able to create a cluster of containers, cluster management software is required. In this project, two cluster managers were used, Docker-compose and Kubernetes. Both of them are represented in Figure 4.1 inside the VMs. Either one of them can be used in a deployment. All the VMs hosting the cluster are connected to the monitoring stack, represented by the NETDATA logo.

4.1.1 Container cluster architecture

A cluster is a group of containers inside the same environment, working together as a multi-container application. These containers are all inside the same virtual network so they can communicate with each other and, if needed, use the same permanent storage. Although each container has its temporary storage that is only kept during its runtime. As mentioned before, MAM4PRO follows a modular architecture. This means that its features are divided into different

modules with some modules containing multiple related features. While building the cluster, the fastest and simplest way to divide these modules into different containers is to follow a similar approach and create containers separated by features. Inside those containers can be one or more modules from MAM4PRO. Our cluster kept the same organization of the MAM4PRO software with some necessary changes to allow communication between the modules. These changes will be explained further ahead. This modular approach is helpful when identifying bugs or scaling the solution. It is also helpful when updating the cluster. This approach allows developers to update certain containers without affecting the remaining containers. This allows for updates to occur without affecting the functioning of the entire solution.

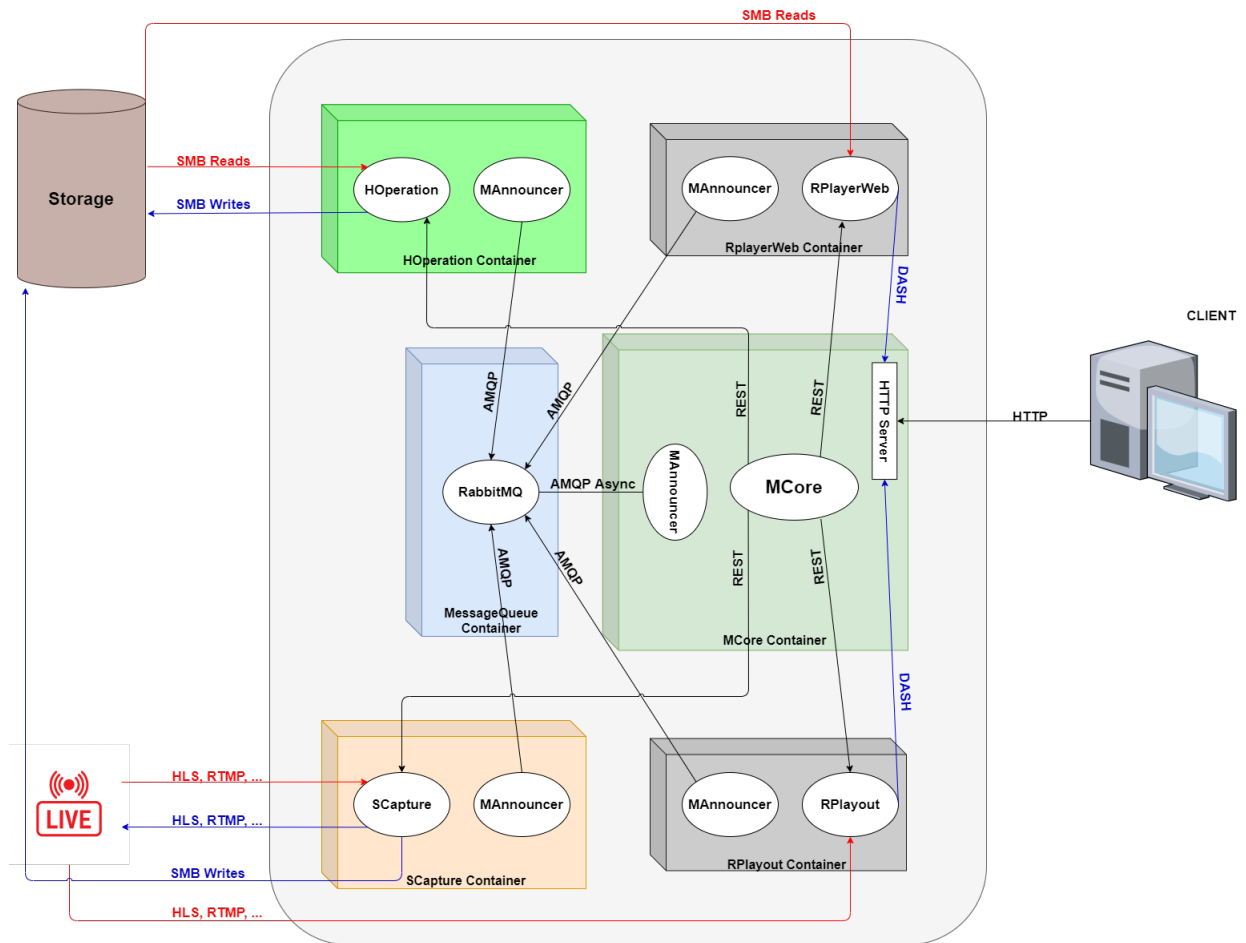


Figure 4.2: Architecture of the container cluster

Figure 4.2 represents the cluster of containers hosting our solution with a focus on the interactions between the multiple containers. The interactions, represented by the arrows, contain the protocols used to transfer information between each module. Besides the containers, there is also the representation of a storage, that represents the pre-recorded content, and a representation of the live content. These two are the different input/output options of media. The organization of the containers in the image was done by grouping them together with containers that work with the same type of data, this will be further explaining in the following sections

about each one of the containers. The *MessageQueue* container is at the center of the operations since it is the bridge between the *MCore* container and the remaining modules. The *MCore* container is also in the center of the operations, this container has the module responsible for managing the media operating module's, *MCore*, and the service that launches an HTTP Server to host the **GUI** interface for the users. This container acts as a **REST API**, where it receives the requests from the users and re-routes them to the necessary modules in the other containers.

4.1.2 MAnnouncer and RabbitMQ

In Figure 4.2 is possible to see that the *MAnnouncer* module is present in more than one container. This is because of the necessity to ensure communication between modules. When working with containers the communication between them needs to be configured, since they are single isolated environments. By creating a cluster with all the containers inside, they can recognize each other and communicate but, as stated in chapter two, the MAM4PRO services are designed to communicate using the Rabbit message queue software. The *MCore* is responsible for receiving requests from the users, when that happens, he orders the necessary modules to start the desired operation. The modules then need to keep *MCore* updated on their status of operation. This is where the *MAnnouncer* feature is needed, it provides the modules with an announcing functionality so they can communicate that state to the message queue. In the non-virtualized version of MAM4PRO, these modules launch processes that can interact and call for each other. Here in the virtualized version, they are isolated from each other, which makes it impossible for the processes launched to call for each other. So, since every container, besides the *MessageQueue*, needs the *MAnnouncer* module to "speak", the solution was to replicate this module in every container that needs it.

The *MAnnouncer* communicates with the *RabbitMQ* module that is inside a container with the same name. The communication is done through a message queue protocol called Advanced Message Queuing Protocol (**AMQP**) 0-9-1 that is based on publishers (applications that create messages) communicating with brokers, which then route the messages to consumers (applications that process those messages). In the scope of this project, the *RabbitMQ* is the broker, the media operating modules are the publishers and the *MCore* is the consumer. The *MCore* creates a persistent connection to the queue and receives the messages asynchronously. The *RabbitMQ* ensures that the messages are stored in a queue without losses of messages and that the *MCore* receives the messages in the same order they were sent.

4.1.3 SCapture container

The *SCapture* container hosts the *SCapture* module responsible for media operations to live data, besides that it also has the *MAnnouncer* module for communication. The *SCapture* module is a multi-process feature that needs to work with different protocols of streaming. It also supports different formats of data content. This feature needs to have near real-time performance

since it is working with live data. Near-real time performance is when the processing of the frames of media is done at the fastest speed possible. The fastest speed possible means that this module can process the frames as it is receiving them, which makes it real-time performance. This is the goal to be achieved every time. But this module is not able to offer this performance every time. When the transformations being applied to the media are more complex (harder to encode/decode) or when the media received is more demanding to transform (higher resolution of video for example), the performance decreases and stops being real-time. It is still very close to real-time so it is considered that this module offers near-real-time performance.

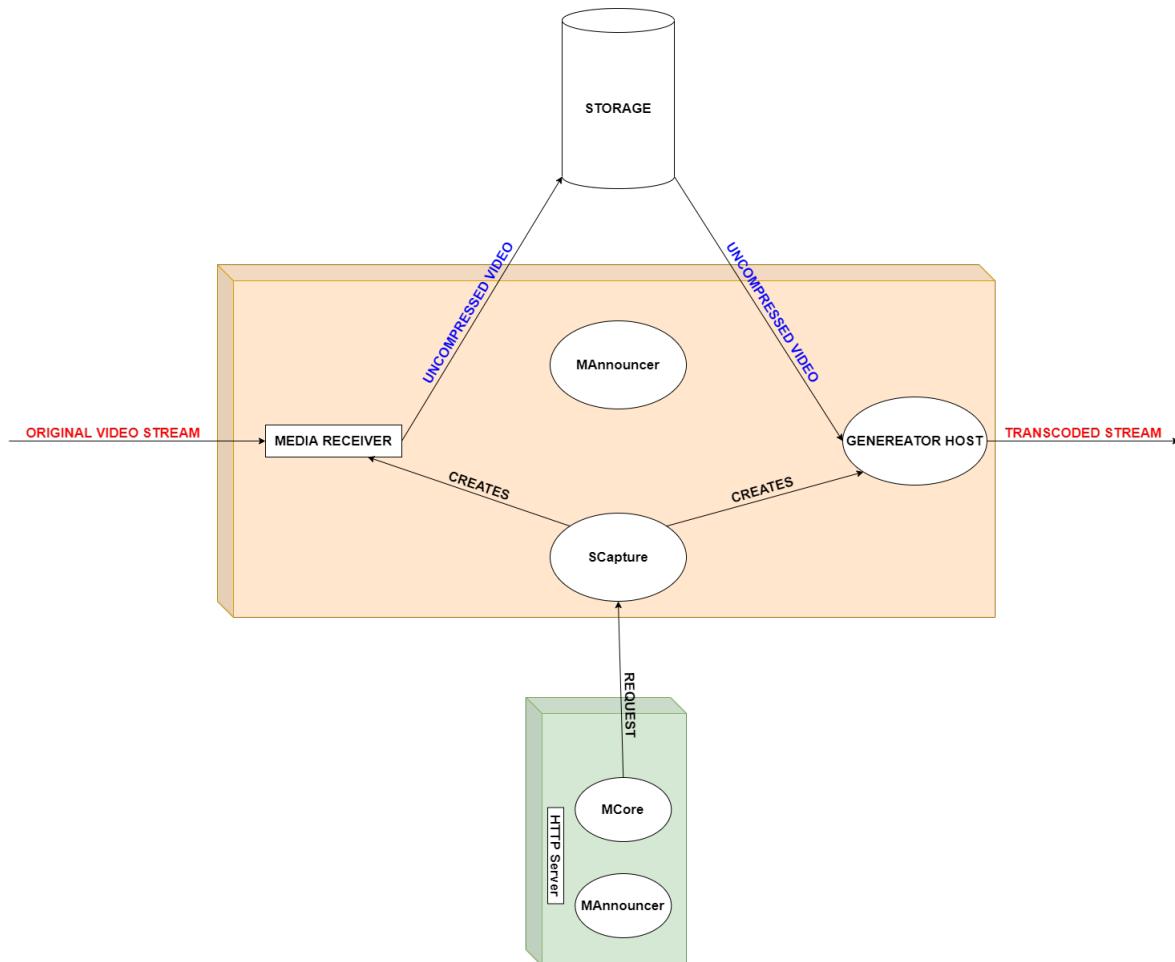


Figure 4.3: SCapture container

Figure 4.3 provides a detailed look at what happens inside the *SCapture* container when processing media. After it receives the request from the *MCore* it launches 2 processes called *media receiver* and *generator host*. What these processes do is receive media and transform it respectively. The *media receiver* receives a live stream, transforms it to uncompressed video, and stores it in the storage of the VM. The *generator host* receives the uncompressed video from the storage and applies the desired transformations before either transmitting it again or saving it in the storage as a file.

4.1.4 HOperation container

The *HOperation* container hosts the *HOperation* module responsible for media operations to pre-recorded content, and the *MAnnouncer* module for communication. The *HOperation* module works with different formats, codecs, and wrappers of video. The amount of supported content keeps increasing thanks to the continuous development at **MOG**.

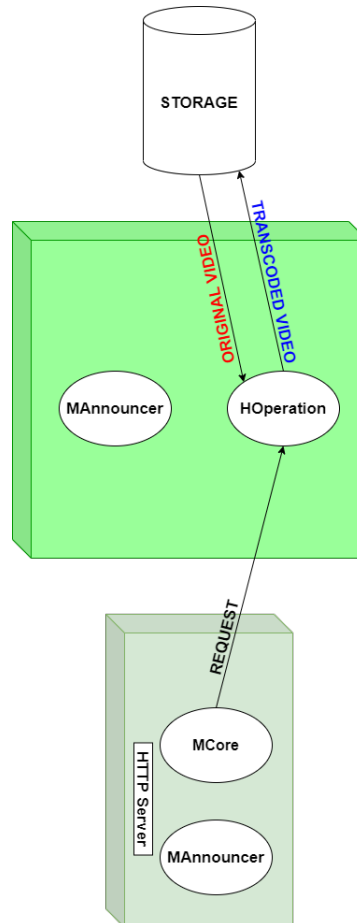


Figure 4.4: HOperation container

The *HOperation* module is a simpler module when comparing to *SCapture*. It requires only one process to transform media, and this process does not have to perform near real-time actions. This is because it is working with files that are pre-recorded. Naturally a video file with a 4k resolution will take longer to encode/decode and transform than the same video file but with 720p resolution, even if they have the same file extension and codecs associated to them.

4.1.5 RLayout container

The *RLayout* container hosts the *RLayout* module, the *rplayersdi* module, the *casparcg* module and the *MAnnouncer* for communication. The *RLayout* container is responsible for

providing our solution with a media player for live data. This enables the user to see the live streams that are being received in real-time.

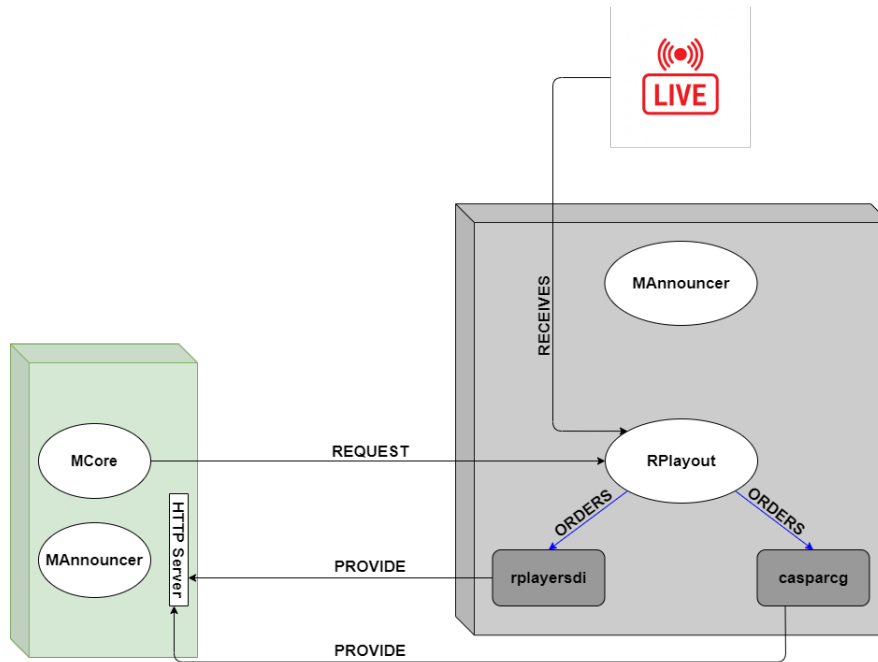


Figure 4.5: RPlayerWeb container

The *RPlayerWeb* module is just a manager that will receive the requests from the *MCore* and choose which one of the two processing modules can provide the HTTP Server hosting the GUI with a player to see the live stream content. The two processing modules, *rplayersdi* and *casparcg*, offer different support to streaming formats. The *rplayersdi* is developed at MOG and the *casparcg* is a third-party player included in order to support different formats that the *rplayersdi* does not support. By including a third-party player that complements their player, MOG is increasing the number of supported streaming protocols without having to add redundancy to their player.

4.1.6 RPlayerWeb container

The *RPlayerWeb* container hosts the *RPlayerWeb* module as well as the *MAnnouncer* module for communication. The *RPlayerWeb* module is responsible for providing our solution with a media player for the pre-recorded content. This means that it works with files and does not have to ensure a near real-time performance.

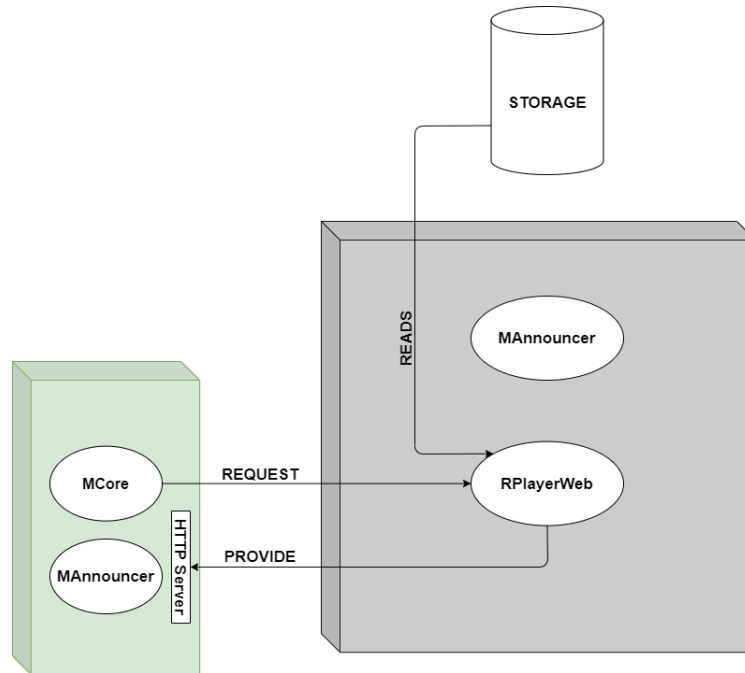


Figure 4.6: RPlayerWeb container

This player needs to support a wide array of codecs, formats, and wrappers. It is a simple singular process container where the *RPlayerWeb* module performs all the operations. This player supports media features that the *RLayout* does not have such as video seeks.

4.1.7 Scalability

As said before it is possible to launch more than one container to amplify that container capacity. For example, the container responsible for applying the media transformations to live data is SCapture. This container was designed to process a single stream simultaneously. When using the non-virtualized solution, to handle a use case where there was a need to process multiple streams at the same time, the clients needed to buy more hardware to process all the streams. Thanks to this project's solution it is now possible to launch new copies of the SCapture container, which are added to the cluster environment and can process one stream each. This increase might need an increase of the VMs resources to prevent a bottleneck.

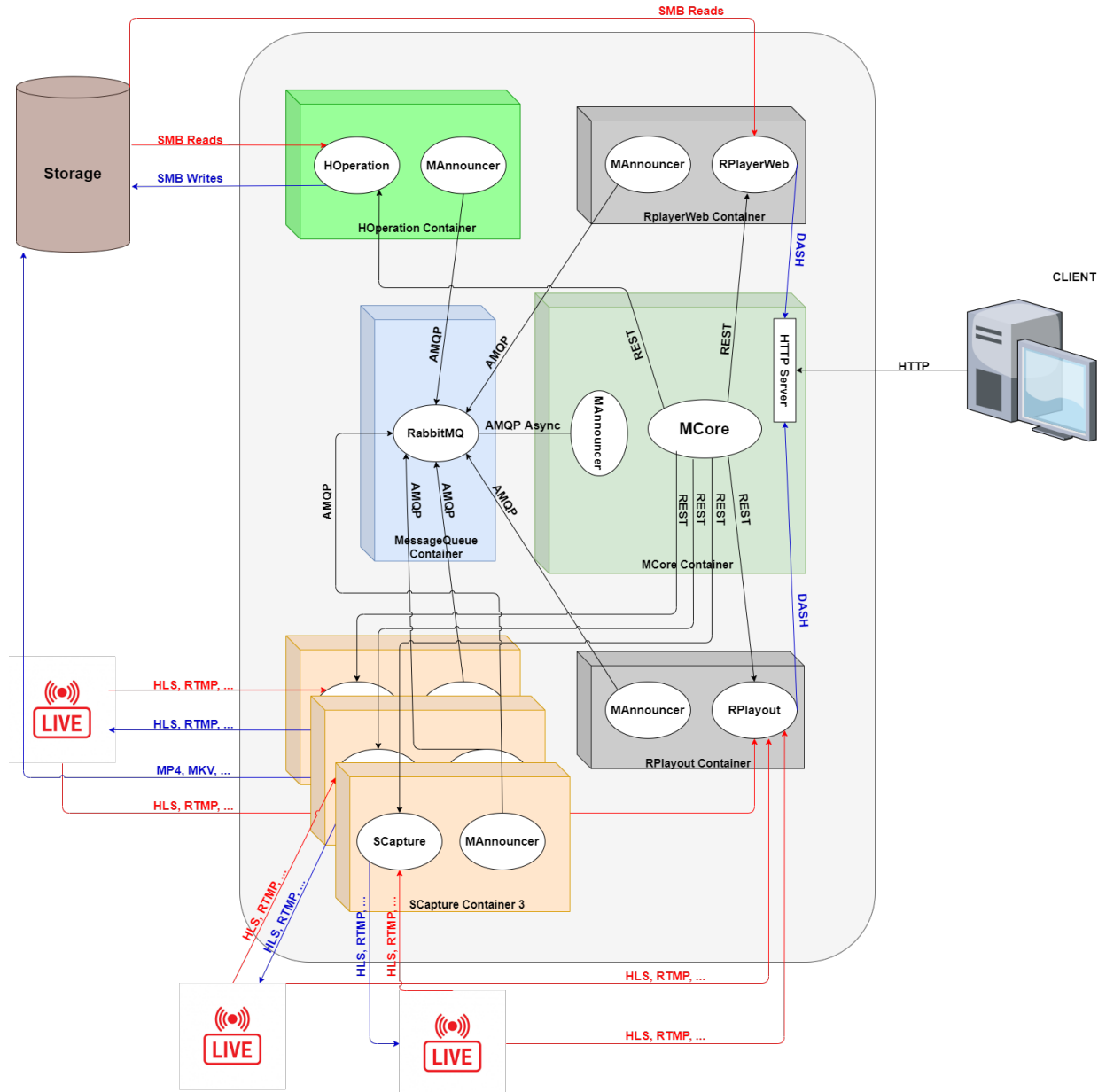


Figure 4.7: Scalability example

Figure 4.7 illustrates a use case where three streams are being transformed by the solution. The number of SCapture containers increases and each one of them works with a different stream. It is possible to scale every container in the cluster, but not all of them need to be scaled. In Figure 4.7 the RPlayer container is not scaled even though it is working with three streams instead of one. This container is able to work with more than one stream simultaneously without scaling.

To manipulate and change the cluster of containers, a cluster managing tool is needed that launches, deletes, and edits the cluster. Docker offers a managing tool called Docker compose, this is an extension developed by Docker to work with their containers. But, there are other tools capable of this task, the most complete and widely adopted one is Kubernetes. In this project,

both tools were used, although at first, only the Docker compose was compatible with our solution since MAM4PRO uses a third-party technology that made the Kubernetes technology integration impossible. Recently this issue was solved granting us the possibility to use Kubernetes as well. Given the small amount of time left, the full list of Kubernetes features was not fully explored.

4.2 Cluster creation

As explained before in chapter 2, Docker works with files called Dockerfiles that when built give origin to images. Those images when run give origin to containers, this process is illustrated in Figure 4.8. For this project a cluster of containers was needed. To create the cluster of containers there was a need to divide the modules of MAM4PRO into different images/containers. As seen in the previous section, the division of containers in CMAM4PRO followed an almost identical organization to the module division in MAM4PRO. The first step to build the cluster is writing the Dockerfiles for CMAM4PRO.

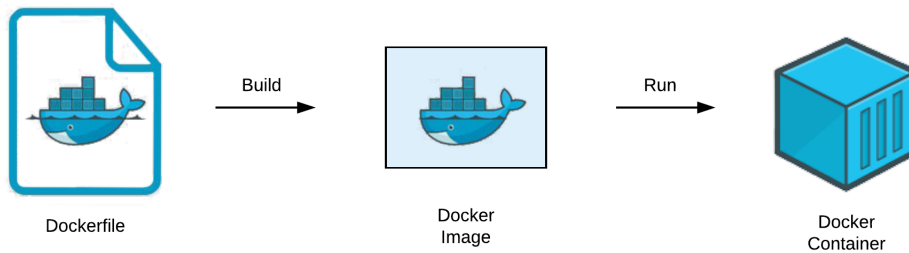


Figure 4.8: Docker process of container creation

When writing the Dockerfiles it was noticed that some configurations and dependencies were common to all the containers. As explained before, a image can be built on top of other image that works as a base. This is used to prevent redundancy in the Dockerfiles. So, in this project, instead of writing the same configurations and dependencies in all the Dockerfiles, a Dockerfile named *Common* was created. The image created by that Dockerfile is responsible for providing the runner for the CMAM4PRO, the base operative system, and configuration files. Although this image is essential it does not result in a container in the final cluster as the others do, as it is illustrated in Figure 4.2. This happens because it will only serve as a base for other images and it is not run as an individual image.

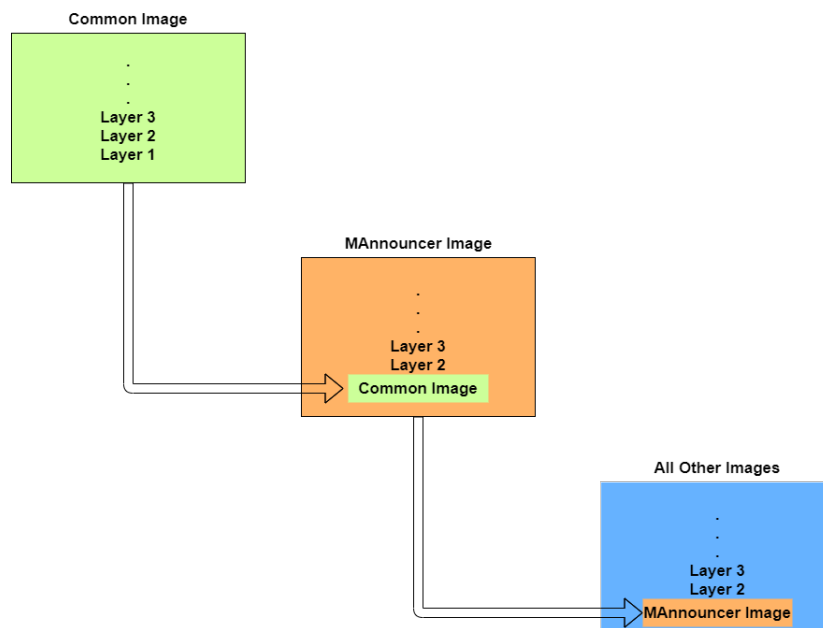


Figure 4.9: The inheritance of the images

This is not the only image used by multiple containers. As explained before, the *MAnnouncer* image is used by every container to communicate. So, the organization of images applied here was to use the *Common* image as a base for the *MAnnouncer* image, which itself is the base for the other images. So, as illustrated on Figure 4.9, every container (besides the *MessageQueue*) receives the *MAnnouncer* as a base, which contains the *Common* as its base.

```

FROM mcr.microsoft.com/windows/servercore:2004
LABEL maintainer="MOG"

ENV PROGRAM_FILES "C:/Program Files"
ENV LICENSE_SERVER "MOG-MONETIZE"
ENV RABBITMQ_SERVER "MOGRABBITMQ"
ENV GROUPID "920b2851-78a1-42bf-b74c-556dda728ed6"
ENV NODE_ADDRESSCOMPOSITION "IPAddress"

COPY setup/common/docker/Mog.DockerRunner.exe C:/Mog.DockerRunner/
COPY setup/common/docker/Mog.WebServicesProxy.exe C:/Mog.DockerRunner/
COPY instdir/common/cfg/ ${PROGRAM_FILES}/MOG/MAM4PRO/common/cfg/

RUN "mkdir C:\ProgramData\MOG\"
RUN "sc.exe config NetTcpPortSharing start=demand"

WORKDIR ${PROGRAM_FILES}/MOG/MAM4PRO/
ENTRYPOINT [ "C:/Mog.DockerRunner/Mog.DockerRunner.exe",
  "C:/Mog.DockerRunner/runner.json", "C:/Mog.DockerRunner/setup.cmd"]

```

Listing 4.1: Dockerfile example

Figure 4.1 represents the Dockerfile for the *Common* image. When analyzing its content it is possible to see, on the first line, an instruction for Docker to set the base image as the Microsoft Windows Server Core build 2004. This **OS** is chosen since it offers compatibility to the MAM4PRO software that is developed in Windows, and is the lightest option, comparing with other Windows builds, that offers all the tools needed for the software to run properly. The Dockerfile continues to configure the needed requirements to define the image. Some of those requirements are the execution of commands to create directories, copy files (like the MAM4PRO runner), enable port-sharing, and define entry points to the application. Since this image serves as a base to the MAnouncer image, everything contained here will also be present in the MAnouncer image. So the base **OS** will also be the same for the MAnouncer, and every container that receives the MAnouncer image as a base image will also have this as a base **OS**.

```
FROM mog.mannouncer
LABEL maintainer="MOG"

COPY instdir/sCapture/ ${PROGRAM_FILES}/MOG/MAM4PRO/sCapture

COPY hasp_rt.exe ${PROGRAM_FILES}/MOG/MAM4PRO/sCapture/bin/

COPY NodeServicesConfiguration.xml C:/Mog.DockerRunner/
COPY runner.json C:/Mog.DockerRunner/
```

Listing 4.2: Dockerfile of the SCapture module

Figure 4.2 is from a different Dockerfile, that is responsible for creating the image for the SCapture module. This module is responsible for processing, frame by frame, media transformations such as *encode*, *decode*, and *transcode*. As expected, this image receives the MAnouncer image as a base, and the installation of the SCapture module on top of that.

After all the Dockerfiles are defined the images can be built. This process is done at **MOG** with a machine that has access to the source code of MAM4PRO code and all its dependencies. Without this access it is impossible for Docker to build the images. These images are constantly being updated to keep up with the MAM4PRO versions that are released. All the images are stored in **MOG**'s registry, basically a private online storage only accessible through **MOG**'s network. To establish the connection from the **VMs** hosting the solution to the **MOG**'s network a Virtual Private Network (**VPN**) software is used. This is useful because the images can be accessed through the internet as long as there is this connection and Docker is provided with credentials to access this registry. In the registry, there are more files besides the images from this project, so there was a need to generate credentials that would only provide reading access to the chosen resource, in this case, the container images. This provides a layer of security by having credentials that only provide reading access to the specified resources.

The **VMs** hosting **CMAM4PRO** only need the images to create the containers. This is also helpful because it prevents sharing the source code from the MAM4PRO software with the public cloud providers, adding a layer of security to our solution in the deployment process. They only

received the already built version of the software in the form of images, which can be compared to a binary after the compilation of code.

When all the images are built it is possible to run them and create the respective containers. But they will not be inside a cluster which means they will be isolated from each other. The cluster represents a group of containers that is connected and working as a single application, in this case **CMAM4PRO**. It is possible to manually create a cluster by using Docker commands to configure the environment that a cluster needs and add the containers to that environment. There are tools that automatize the cluster creation simplifying this process. Docker provides one of those tools called Docker-Compose.

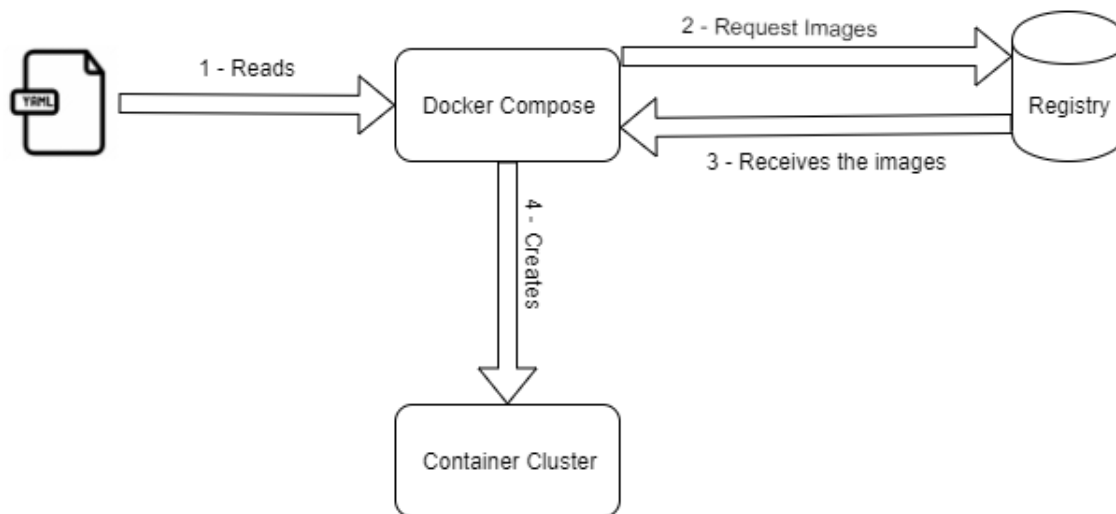


Figure 4.10: Docker Compose steps to build a cluster

The process of downloading the images from the registry is only required the first time the deployment happens and when the images are updated. Otherwise the Docker engine will have the images available locally and run them from there. As seen in Figure 4.10 to create a cluster, Compose needs a YAML file. This is a configuration file that contains all the needed images and configurations to create the cluster.

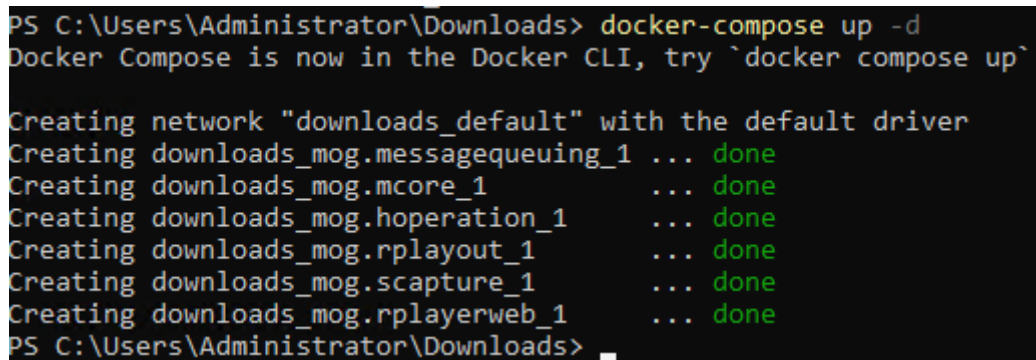
```

...
version: "2.4"
services:
  mog.messagequeuing:
    image: /common/mxfspeedrail/mog.messagequeuing:latest
    hostname: MOGRABBITMQ
    mem_limit: 512M
    environment:
      RABBITMQ_SERVER: MOGRABBITMQ
  mog.mcore:
    image: /common/mxfspeedrail/mog.mcore:latest
    ports:
      - "80:80"
      - "8731-8737:8731-8737"
      - "48731:48731"
    mem_limit: 2048M
    depends_on:
      - "mog.messagequeuing"
    environment:
      LICENSE_SERVER: 10.20.10.52
      RABBITMQ_SERVER: MOGRABBITMQ
...

```

Listing 4.3: YAML File

Listing 4.3 is an excerpt of the YAML file used in this project. Some configurations are container-specific such as memory limits, open firewall ports, and licensing permits. But there are also configurations related to the cluster itself such as the network where the containers are all placed (used by Docker for container communication) and if needed, persistent volumes (used to store persistent data from the containers configurations).



```

PS C:\Users\Administrator\Downloads> docker-compose up -d
Docker Compose is now in the Docker CLI, try `docker compose up`

Creating network "downloads_default" with the default driver
Creating downloads_mog.messagequeuing_1 ... done
Creating downloads_mog.mcore_1 ... done
Creating downloads_mog.hoperation_1 ... done
Creating downloads_mog.rplayout_1 ... done
Creating downloads_mog.scapture_1 ... done
Creating downloads_mog.rplayerweb_1 ... done
PS C:\Users\Administrator\Downloads>

```

Figure 4.11: Docker Compose command to initialize the cluster

Figure 4.11 shows the output of the Compose command that creates the cluster. This command needs to be executed in the same directory of the YAML file to receive it as input. The output contains the name of each container created and attached to the cluster. The status of the creation operation is presented in front of the name of the container. It is possible to see Compose creating the container network that in this case is called "downloads_default".

4.2.1 Kubernetes

Kubernetes is another cluster managing tool for containers just like Compose. During most of the time in this project Kubernetes were not able to be used due to an incompatibility between Kubernetes and MAM4PRO. Nearing the end of this project that problem was solved, by the team developing MAM4PRO, and the integration of Kubernetes was possible.

Kubernetes provides a wider and more complete set of tools when comparing with Compose. Among those tools is the self-provisioning of more resources (containers) when a bottleneck occurs. This is one example of a powerful tool that automatizes the process of increasing or decreasing the capacity of the solution as needed. This, and other features, make Kubernetes an improvement over Compose. But, it also brings more configuration complexity when deploying the solution.

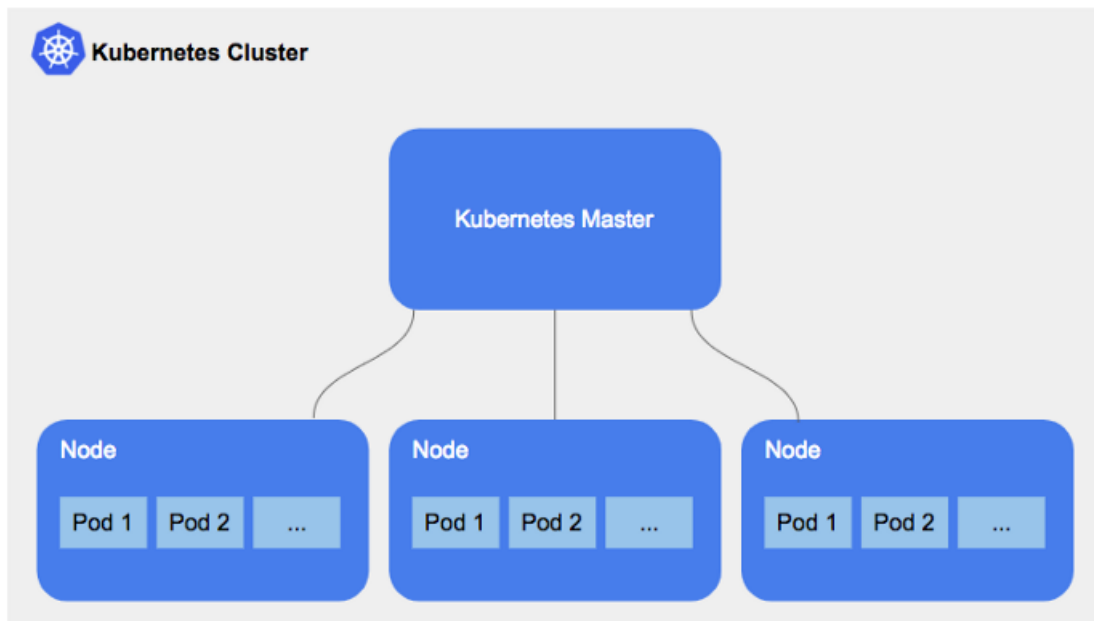


Figure 4.12: Master/nodes relation

The clusters created by Kubernetes use a different organization from the Docker Compose clusters. In Kubernetes at least one Linux machine is needed to reprise the role of master. The master in this project was a physical machine at the MOG facilities. This machine runs the master service of Kubernetes which is responsible for managing other machines in the cluster. The other machines in the cluster are called worker nodes. These are the machines responsible

for performing the determined tasks. In the case of this project these are the **VMs** hosting **CMAM4PRO**. These work nodes run the kubelet service of Kubernetes. This is a service that allows communication with the master service. The worker nodes can be physical or virtual machines. The master is capable of monitoring all the worker nodes as well as configure the containers they are hosting.

Creating a Kubernetes cluster is a process that requires higher degrees of complexity when compared with Docker Compose. There is a higher number of configurations needed in order to be able to create a Kubernetes cluster. Kubernetes provides a set of tools that help in this process. One of the most useful ones is a command line tool called Kubectl. This tool enables us to create/manipulate the cluster in any machine, even if it is not part of the cluster or even has other Kubernetes services installed. As long as the right authentication (token) is provided, Kubectl can operate the cluster. Other useful tool is the Kubernetes dashboard. This service supplies a visual tool to check the status of the entire cluster. This dashboard can be configured to present different metrics in multiple ways, allowing for the solution administrators to perform a better job at managing and monitoring the cluster.

Since time was scarce, the Kubernetes features were not fully explored. The deployment made was enough to test the integration of the Kubernetes technology in our project. The Kubernetes configurations included all the container images, the connection to the **MOG** registry to access and download said images, the license permits to operate MAM4PRO, the persistent volumes used by the worker nodes, and the node's configurations such as open firewall ports and memory limits. The final result was a fully functional solution.

4.3 Cloud Deployment

After the creation of the cluster of containers, the next step for our solution is its deployment to the cloud. The cloud provider can either be public, private or a hybrid of both. The number of resources needed by the solution can vary alongside the capacity needed for a certain task. As stated before, scaling the number of containers for each one of the modules is possible, but the resources of the host must scale as well so a bottleneck does not occur. Fine-tuning the necessary resources for each task is essential to lower the costs as much as possible. For this project, four cloud providers were available to deploy and test the solution. Three public providers with Google, Microsoft, and Amazon respectively, and **MOG** as the private option.

A physical machine with Windows 10 as the **OS** was monitored while hosting **CMAM4PROs**. Some functional operations, as well as some media workflows, were performed in order to extract the amount of resources used. The results provided a general notion of the amount of resources needed by a **VM** to host our solution. The recommended requirements were:

- 8 virtual Central Processing Unit (**CPU**)s;
- A fast/high (1 Gbps) network performance (some providers offer this as a customizable

feature with the "Fast/High" option normally associated with 1Gb of bandwidth per second);

- 32GB of Random Access Memory (**RAM**);
- 200GB of Storage (Either Hard Disk Drive (**HDD**) or Solid State Drive (**SSD**));
- The Microsoft Server Core 2004 build **OS**;

The eight virtual **CPUs** provide enough power for the solution to operate smoothly. Usually the calculation of the virtual **CPUs** is done with the formula (Threads x Cores) x Number of **CPUs**. The network bandwidth chosen is one Gbps because **CMAM4PRO** works with enterprise-level media. In certain uses cases it requires high amounts of data to flow trough the solution. The **RAM** amount was defined after testing the physical machine with a live stream workflow. In this use case the solution is more reliant on this type of memory than other media operations. The physical machine used 24GB of **RAM** when processing just one live stream. The amount of 32GB offer more than enough **RAM** to ensure a operational solution. Later after testing the solution this amount will be subject of debate. The 200GB of storage is just a comfortable amount to ensure that the **VM** has enough space for the containers (they amount to a total of 25GB) and for the tests inputs/outputs that were used after the development. When working with media, specially with high quality media, the file sizes are often several GB.

Besides installing all the software needed by our solution, the **VMs** also need network configurations. More specifically, they need to have firewall ports open to internet traffic. The ports in question are ports 80 and 9182, and both use the TCP protocol. Port 80 hosts the HTTP Server for the **GUI**, and port 9182 is used by a monitoring tool to expose resource usage metrics. This will be explored in greater detail in the next subsection.

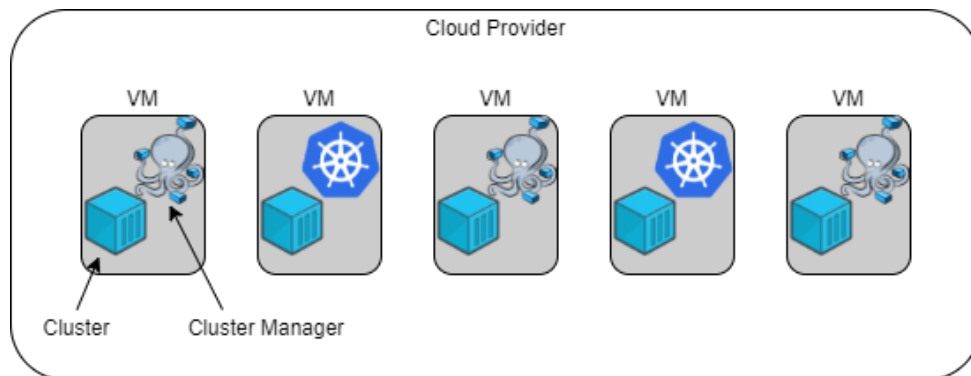


Figure 4.13: Final overview of the solution deployment

The final look of the interaction between the cloud and the virtualized media software is represented in Figure 4.13. It is possible to identify the cluster of containers running inside a **VM** along with the cluster manager software. As said before, the cluster manager software varies because the project started by only having the Docker-Compose tool available, and later it was also possible to implement the use of Kubernetes. This provides flexibility in the type of

deployment a certain use case might need, whether it is with Compose or Kubernetes. Figure 4.13 also illustrates that naturally, it is possible to have multiple VMs in the same provider. These VMs can have different amounts of resources to better suit each use case.

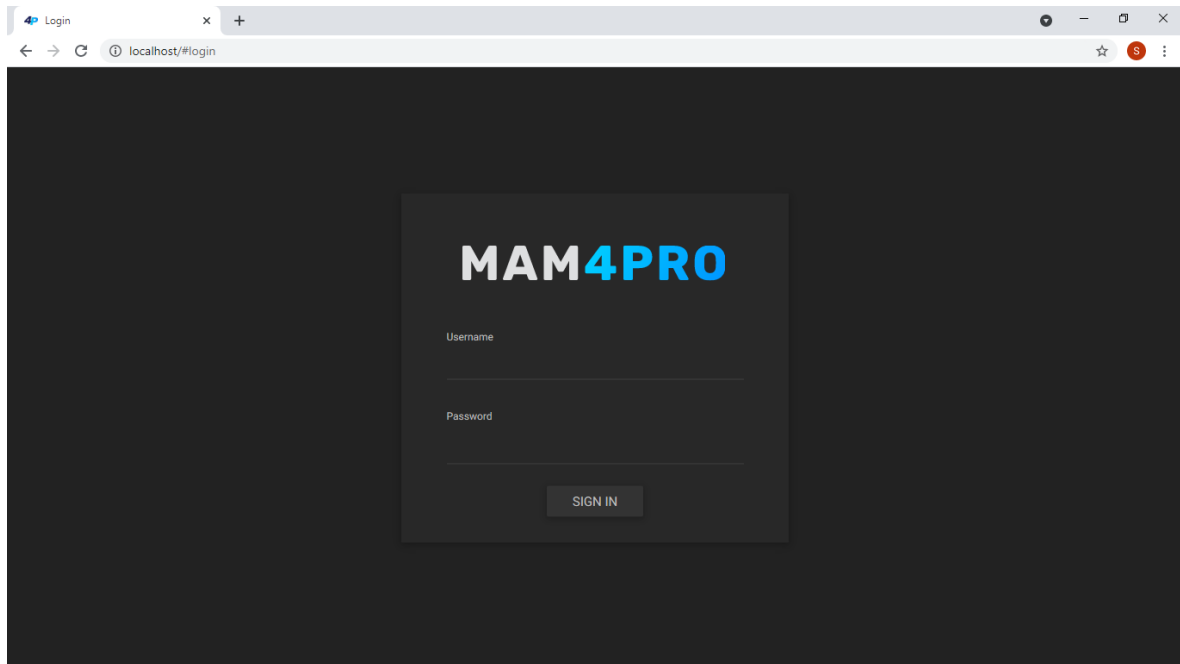


Figure 4.14: MAM4PRO log in menu

At this point, the solution was ready to receive and process media. The entire deployment and configuration of the virtualized solution was done and it was fully operational. The public providers generate a public IP address that give access to the user interface seen on Figure 4.14. When inside the VM environment this interface is accessible on the localhost. A web browser can be used to access and interact with it.

4.4 Monitoring software

The last component of our solution is the monitoring stack that captures and analyzes the resources of the VMs. In the scope of this project, NETDATA was one of the tools chosen for that stack, since it provided us with a dashboard interface to visualize data. Another advantage is NETDATA's ability to provides us with data, in a CSV format, from a certain time frame. This allowed us to build graphics to better understand the VMs performance. The fact that it is open-source is also weighted in the decision. For the data to reach NETDATA, the VMs need to be configured to collect and expose their metrics. The last component of the monitoring stack is a tool designed to capture and expose resource metrics from Windows Server, called WMI-Exporter. Netdata can access the data in port 9182 and present it in a user-friendly dashboard. The WMI-Exporter is not a native tool included in this version of the Windows Server so it needs to be downloaded beforehand. The IP of the VMs is then provided to the

NETDATA software which then begins to periodically scrape that port for the metrics.

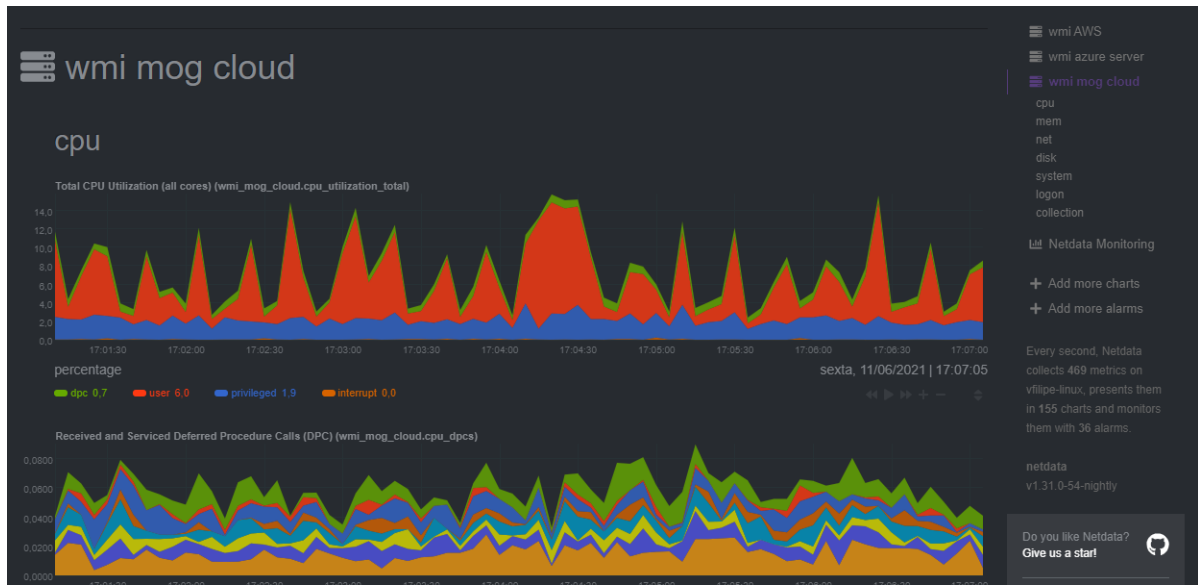


Figure 4.15: NetData dashboard example

Figure 4.15 shows an example of the dashboards available in NetData that offer customization in the way data is presented. The resources that required more focus in this project were the CPU percentage usage, RAM usage, Disk operations (reads and writes), and Network bandwidth, the latter used only when handling live streams. This analysis helps to understand the behavior of the VMs while hosting the virtualized MAM4PRO solution. While also being useful when trying to find differences in performance from one VM to another. The in-depth analyses of the performance from the different VMs are presented later in this thesis. With the last component added to our solution, the entire solution is now deployed and running.

Chapter 5

Test and benchmark

This chapter presents the tests which were performed to check the fulfillment of the requirements presented in Chapter 3, as well as an analysis of the solution behavior in the multiple cloud providers. Since the performance changes from provider to provider, a comparison of the results obtained in the different VMs is also made. This comparison is based on resource usage and performance related metrics.

5.1 Functional testing

CMAM4PRO was tested to ensure that the functional requirements defined in Chapter 3 are all met. The tests were performed as the development progressed and repeated in the end. Table 3.1 provides the test results and descriptions of each functional requirement.

Requirement	Expected Result	Actual Result	Test Result
Input of live streams	Select an ip-based live stream as media input	As expected	Passed
Input of files	Select a file (pre-recorded content) as media input	As expected	Passed
Select the encoder	Chose the encoder used in the media operations	As expected	Passed
Configure the encoder	Configure the encoder settings	As expected	Passed
Select the input/output format	Chose the input/output format of media	As expected	Passed
Store the input as a file	Store the output of media operations in a file	As expected	Passed
Store or broadcast media	Store the media output or broadcast it	As expected	Passed
Scale the solution	Increase/decrease the solutions' modules to increase/decrease its capacity	As expected	Passed
Solution hardware resources	Increase/decrease the hardware resources to better fit each use case	As expected	Passed

Table 5.1: Functional requirements

The results of these tests show that the creation of **CMAM4PRO** was successful since it was able to perform the same functionalities as the non-virtualized approach. The tests were designed to check if all the containers were producing the right output without errors.

5.2 System requirements

The system requirements, as referred to in Chapter 3, are global characteristics that **CMAM4PRO** should offer for it to run as expected. These requirements are related with the performance, the scalability, the diversity, and the robustness of the solution.

5.2.1 Performance

The performance of a new solution is always one of the key points that needs to be analysed. Although the main objective was not to increase the performance of the non-virtualized approaches, it was still mandatory to achieve a reasonable performance (within an acceptable range from the non-virtualized ones). This was tested by indulging the solution in a series of tests that mimicked common use cases.

The chosen test scenarios are based on the transcoding of video files from an MP4 format to an MKV format. More specifically, three different test workflows were defined for each file of a total test set of three files. Therefore, a total of nine workflows of video transcoding were used.

The three chosen files possess different resolutions and lengths to ensure enough diversity, and are presented below

- Beach: a sample video from a beach, 15 seconds long with a resolution of 1920x1080p and a file size of 78,5 mb.
- Grav: a movie trailer from the film "Gravity" (2013) with a duration of 2 minutes and 27 seconds, 2048x858p resolution, and a file size of 355 mb.
- Blue: a movie trailer from the film "View from a blue moon" (2015) with a duration of 3 minutes and 3 seconds, 1280x720p resolution, and a file size of 68,4 mb.

For each video there are three different workflows which output a file with a different resolution. In the case of the videos Beach and Blue, the original resolution is increased, decreased and maintained, depending on the workflow. For the video Grav, the three defined workflows originate an output with a resolution which is different from the original one.

Video name	Workflow name	Input resolution	Output resolution
Beach	Beach - 720p	1920x1080p	1280x720p
	Beach - 1080p	1920x1080p	1920x1080p
	Beach - 2160p	1920x1080p	3840x2160p
Blue	Blue - 480p	1280x720p	854x480p
	Blue - 720p	1280x720p	1280x720p
	Blue - 1080p	1280x720p	1920x1080p
Grav	Grav - 1080p	2048x858p	1920x1080p
	Grav - 1440p	2048x858p	2560x1440p
	Grav - 2160p	2048x858p	3840x2160p

Table 5.2: Test workflows

Table 5.2 lists all the workflows. Notice that, there is an identifier to each video in the "Video name" column, followed by the workflow used for that video. So, for example, the "Grav - 2160p" row represents the workflow that upscales the Grav video to an 3840x2160p resolution.

The reason for choosing different resolutions comes from the need to have diverse test scenarios to better compare the performance of each cloud provider. This analysis will help to better decide which provider better suits each use case scenario. While some users required a certain performance to be achieved, others do not. So it is important to discover which provider offers the best performance.

In Figure 5.1, is possible to see one example of a workflow, defined in CMAM4PRO, that applies the format and resolution transformations to these files. The presented workflow creates a 1920x1080p MKV file as output. In the figure, it is possible to see the different fields configured in order to apply such transformations. Such fields include the categorization of operation type, the format of the input files, and the configuration of the output files. The output configuration combines multiple fields in itself such as the format family, format sub-family, and wrapper of the output. It is also specified where to store the output.

Figure 5.1: MAM4PRO workflow example

In the tests performed in this work, the solution always reads and writes the input/output locally, more specifically to a shared folder using the server message block network protocol on the **VM** storage.

Moreover, all three files and all the workflows are the same in the **VM** to provide a fair and reliable test scenario. The input files are located in the **VM** storage and the output is written to the same location. So this operation does not involve any kind of network latency to access the files or to write the outputs. The **VM** from the public providers are all in the same conditions, while the conditions from the **MOGs VM** were as similar as possible to the public providers, but not equal. The reason behind this is explained further ahead. The hardware resources used by the public providers **VMs** are:

- 8 virtual **CPUs**
- 32 GB of **RAM**
- 200 GB Hard Disk Drive (**HDD**)

Although the setup of resources for the **MOG VM** was the same as described above, their amount of resource sharing is a lot higher. This makes the conditions different because the amount of **VMs** per resource is a lot higher at **MOG** than it is on the public providers. This applies a higher workload on the hardware resources and affects the performance of **CMAM4PRO**. Opposite to the public providers, **MOG** supplied the information about the hardware used by their **VMs**. The server responsible for virtualization was the Dell PowerEdge R610, the **CPUs**

was the Intel Xeon CPU X5660, the **RAM** was the Samsung M471A1K43DB1-CTD and the memory storage was the DELL Equallogic PS6100.

The performance of the **VMs**, while hosting **CMAM4PRO**, can be compared using the recorded times of each test workflow. Each workflow was applied ten times and the execution time of those runs was recorded. This was repeated multiples times over the course of two weeks. The tests repetitions were made on different periods of the day and during the weekends as well. This was a decision made to have a better sample of data, which leads to a more accurate performance measurement of each workflow. The data centers hosting the solution can have higher workloads on different periods of a day, from there derives the need to perform the same test on different days and periods of a day. Then, all the results were gathered and the average values were calculated as well as the standard deviation for those values. Table 5.3 is an example of this data from the Grav - 2160p workflow.

	AVERAGE TIME	STANDARD DEVIATION
MOG	5.01	0.2837
AWS	3.44	0.0097
AZURE	3.21	0.0077
GOOGLE	4.08	0.0063

Table 5.3: Metrics analysis from the Grav - 2160p workflow

From the values presented in Table 5.3, is possible to compare the performance of the **VMs** while **CMAM4PRO** is upscaling the video Grav to a 3840 x 2160p resolution. There is a clear advantage of the public clouds against the private cloud from **MOG**, and even between the public providers there are noticeable differences. Notice, however, that the information about the specific hardware used by the public cloud providers is not disclosed. Therefore, the reason behind the difference in performance can not be fully explained here. Nevertheless, **MOG's** cloud is not only the slowest but also the more volatile, as it is displayed on Table 5.3 by the standard deviation column. This column provides information about the fluctuation in performance from the all multiple executions of this workflow. This difference between the public providers and **MOG** is very high in this specific test case. Besides the type of hardware used, other variables can explain why the difference is so high. It can be due to the fact that there is a higher number of **VMs** per resource at **MOG**. This causes a constantly higher usage percentage of the resources from **MOG** than the resources from the public providers. One of these resources is the memory storage. In the public clouds abstraction is used to divide the multiple physical memory hardware into smaller virtual partitions that are independent of each other. While, at **MOG**, the memory hardware is composed of a single physical resource that is shared, through virtual partitions, with all the **VMs** running in their cloud. That shared single physical memory provides a smaller availability of resources when compared to a public cloud. This is the main reasons that could be identified for the difference both in performance and stability.

Its possible to determine that the Microsoft Azure **VM** is slightly faster than the **AWS** and

Google VMs. All three public VMs have the same resource specifications but the type, brand, and power of the hardware in each one of them are undisclosed by the providers. Besides hardware, other factors might explain the difference in performance, for example, a higher usage from one data center to the other. These variables can not be measured so it is impossible to state that they are the reason behind the difference in performance. Azure and AWS are still the two most efficient providers and their performance is very similar. Both in time, as well as consistency, since their time values do not present big variations. These two providers are considerably faster than Google who is also a huge competitor in the cloud market. But, Google is the more stable provider out of them all. Offering a slower but more predictable service that can suit some client's use cases better than others.

Next, there is the same comparison but for the lightest workflow, whose values are shown in Table 5.4. This workflow is the one that downscales the Beach video to a 1280x720p resolution. This video is just 15 seconds long, which makes him the shortest test video of all three, and this helps to lower the complexity of the process. This is the fastest workflow in every VM from all the tested ones.

	AVERAGE TIME	STANDARD DEVIATION
MOG	0.10	0.0046
AWS	0.10	0.004
AZURE	0.09	0.003
GOOGLE	0.11	0.0067

Table 5.4: Metrics analysis from the Beach - 720p workflow

In this simpler test case, it is possible to see that now MOG has a very similar performance to the public clouds. It even outperforms Google being faster, or equal, in all runs, which results in an average time lower than Google. Azure keeps the greatest performance of them all but is only slightly faster (1 second in average) than AWS and MOG. In terms of time variations, the data reveals that all the providers, except for Google, get increased stability in this simpler workflow. Google keeps almost the same standard deviation it had in the most demanding workflow. This means that the complexity of the workflow does not affect the stability of the solution on Google. This can be thought of as an advantage because it proves that the solution has a predictable behavior. But, it is also a disadvantage when comparing to the other providers, given the fact that the other providers offer a much better stability in this workflow. Even MOG, which had the worst performance in the previous test, here provides a faster and more stable option than Google.

As the workflows get simpler and the files get smaller, the gap between the VMs performance decreases and, eventually, the difference between them is close to being unnoticeable. But, the more demanding the workflow the bigger the gap will be, with Azure always displaying better performance.

By combining the entire data from all the workflow executions is noticeable that, in general,

the public clouds offer better performance compared to **MOG**. In the most extreme case, the performance from **MOG** was 65% slower than the performance from Azure, which is the best option at every workflow. In time-critical operations, this can become a problem, so the information of which is the best performing provider is very useful. Table 5.5 has all the average times (in minutes) from each test workflow on each cloud provider, where is possible to see each **VM** individual performance in every workflow, and it is possible to compare the performance's of all the providers.

	WORKFLOWS								
	Grav - 2160p	Grav - 1440p	Grav - 1080p	Blue - 1080p	Blue - 720p	Blue - 480p	Beach - 2160p	Beach - 1080p	Beach - 720p
MOG	5:01	2:17	1:08	1:08	0:36	0:25	0:40	0:14	0:10
AWS	3:44	1:46	1:05	1:07	0:36	0:24	0:33	0:13	0:10
AZURE	3:21	1:33	0:58	1:00	0:33	0:22	0:30	0:12	0:09
Google	4:08	1:59	1:12	1:17	0:44	0:29	0:38	0:14	0:11

Table 5.5: Average time (in minutes) to run the test workflows

Besides the comparison between the cloud providers, previously presented, a comparison with the current/non-virtualized solution was also made.

MOG supplied this project with hardware that had the non-virtualized solution installed. The hardware resources of this machine are different from the **VMs** used for this project. This hardware setup was chosen because it is one of the most common one's built and sold at **MOG**. The configurations of the machine are:

- 1 processor Intel i9-9900
- 2x 8GB DDR4 2666 1R x 8 SODIMM (Samsung M471A1K43DB1-CTD) **RAM**
- 1 Solid State Drive (**SSD**) 256GB 3D NAND NVMe PCIe Gen3 x2 (WD SDAPMUW-256G)

Given these resources and the non-virtualized version of MAM4PRO installed in this machine, the results of the same tests are presented in Table 5.6.

	WORKFLOWS								
	Grav - 2160	Grav - 1440	Grav - 1080	Blue - 1080	Blue - 720	Blue - 480	Beach - 2160	Beach - 1080	Beach - 720
Current	2:50	1:19	0:46	0:52	0:28	0:20	0:25	0:09	0:08

Table 5.6: Non-virtualized solution

This machine outperforms **CMAM4PRO** in every test. In the less demanding tests, the time is only slight faster (one second in average) than the best-performing cloud provider (Azure). It is important to keep in mind that this setup has at least one component better than the **VMs**

which is the **SSD** as storage. This type of storage provides faster reading/writing times than the **HDD** technology used by the **VMs**

The cost of performing these tests on the **VMs** is much smaller than it is to buy the machine here used. This derives from the fact that to perform the tests on the **VMs** the users only need to lease the equipment for the duration of the test opposed to the non-virtualized solution where they have to buy the equipment. Buying this type of equipment is several times more expensive than the lease of a **VM** on a cloud provider for the duration of our tests.

Nevertheless, it is important to highlight that **CMAM4PRO** is not aiming to be faster than the non-virtualized approach. As previously stated, this project aims to correct some of the disadvantages that the current solution has, without significantly decreasing the performance. And, besides that, offer a lighter and faster deployment that is capable of performing the same tests with less complexity and resources.

In conclusion, **CMAM4PRO** is a solution that successfully passed the tests performed, while replicating the behavior of the non-virtualized solution with a similar performance (if the best cloud provider is considered). Moreover, **CMAM4PRO** is easily scalable, meaning that at any given time it is possible to increase the hardware resources, which would most likely increase the performance as well.

5.2.2 Live stream tests

Another important use case is the live broadcast of content. To test this a workflow was created that simulates the transformation and broadcast of a live stream. This test serves the purpose of studying the behavior of our solution when working with time-critical operations. The workflow used is similar to the previous ones but, in this test, it changes the streaming protocol from a test live stream. This live stream was a looped video from a French national television broadcast. This stream is used for testing purposes and is transmitted non-stop, 24 hours a day, with a 640x480p resolution. The stream is being transmitted from a server using the HLS protocol and the workflow designed for this test changes the protocol from HLS to FLV. The FLV protocol was chosen since our output is a Youtube channel, created for this test, and FLV is the protocol Youtube works with.

Figure 5.2: Stream protocol change workflow

In Figure 5.2 is possible to see the workflow configuration used to define this test. First, for the system to receive the stream it was necessary to provide **CMAM4PRO** the link for the transmission. Afterwards, all the workflow fields were defined, namely the output of the operation which is the FLV protocol. In this case, the stream is being redirected to an Youtube channel. In that channel, a live stream event was created which generates two links for the transmission, one private (for sending the content to the stream) and a public (that offers access to the player showcasing the streams content). After applying the workflow to the stream, the Youtube live stream event receives the stream and displays it in a player which is accessible to everyone with the public link. Youtube simply acts as a live stream platform to ensure that the transformation and redirection are correctly applied to the stream. If an operation of the workflow fails, it would not be possible for Youtube to receive and display the stream. Notice that this process has inherent network latency, since the media has to travel between multiple servers from its origin point to Youtube.

Beside network latency there is also the latency related with the operation of altering the streaming protocol. In this context, the entire latency of the workflow is comprehended by the time it takes for the stream to be received, modified, and sent to Youtube. This latency is obtained by measuring the time that the stream takes to reach the Youtube player after the workflow is initialized in **CMAM4PRO**.

The data centers hosting these **VMs** are located in different countries and they were chosen by proximity to Portugal.

- The **AWS VM** is located in London, United Kingdom and the measured latency was 17 seconds.

- The Azure VM is located in their Northern Europe datacenter, which is in Ireland, and the measured latency was 14 seconds.
- The Google VM is also located in London, United Kingdom and the measured latency was 15 seconds.
- Lastly the MOG VM is located in Portugal, at the MOG facilities in Maia, and the measured latency was 20 seconds.

The latency recorded is mostly affected by two variables: the internet connection stability, both from the cloud providers end as well as from the server/machine streaming the test stream, and the user/datacenter location. With the results obtained is possible to state that, in this specific test, the provider with less latency was Microsoft and the one with more latency was MOG. In MOGs case, a internet bandwidth bottleneck occurred meaning that the available bandwidth was not enough to always keep the stream running smoothly. This is caused by the high amount of hardware sharing done at MOG where several VMs share the same physical resources and network infrastructure.

5.2.3 Scalability

Scalability represents the ability of our solution to increase or decrease its processing capacities to better suit the user's needs. Scalability is offered in two aspects, by the cloud in terms of hardware resources, and by CMAM4PRO in terms of media processing capacity. As explained before, it is possible to easily launch multiple instances of a certain feature, increasing that feature's processing capacity. This is a very relevant feature that our solution has that is not present in the current one.

To test the scalability of the solution it was necessary to create tests that would require scaling a certain container. Since the SCapture container can only process one stream simultaneously, to simulate a use case that works with multiple streams simultaneously is necessary to scale this container. A test to process three streams simultaneously was created. A deployment with three instances of the SCapture container was launched and each one processed a different stream. CMAM4PRO was successful at processing the three streams without having to change the VMs resources.

When the system is just receiving streams (and not processing them), the most affected resources were the RAM and the internet bandwidth. The RAM rose but it never reached a high percentage of usage. Remember that if the solution is just receiving the streams simultaneously, then the only container being used is the RLayout that provides the user with a player to see the received streams' content.

	Nº STREAMS		
	1	2	3
AVG. RAM	4,6	5,2	5,9

Table 5.7: Average RAM used in GB

Table 5.7 represents the average RAM, used from the Azure VM, when receiving low quality streams. The three streams had different resolutions. Two of them had a resolution of 640x480p and the other had a 256x144p resolution. Besides the RAM, another heavily affected resource is the bandwidth. In this test, when just receiving the streams it was kept under high percentages of usage with the average value being 250 Mbps of the 1Gbps available.

Until now CMAM4PRO was just receiving the streams but, it was also tested while recording the content of all the streams simultaneously. Notice that this process already requires three SCapture containers to transform and record the streams. The RAM and internet bandwidth remained within the same values, so the transformation necessary to record the streams into a file does not affect these resources. Opposite to that is the CPU usage which changed drastically from just receiving the streams to recording them. In this case, its usage raised to an average of 80%.

In order to provide a more diverse test, this recording operation was also done with three high quality streams with 1920x1080p resolution. This led to a bigger resource usage and even provoked a bottleneck of the CPU, which means that the amount of CPU was being totally used and it even compromised the usage of the solution. It partially affected the ability of the solution to maintain a good flow of the streams to the players on the GUI dashboard, but did not affect the recording of the content. This demonstrates that the amount of CPU was barely enough for this operation since some modules were not functioning properly. More precisely, the SCapture containers were able to perform their operations but the RPlayout container was not delivering the streams with the expected flow.

The amount of used RAM rose to an average of 9,1 GB, which is still far from the 32 GB that the machines are configured with. When testing with the non-virtualized solution the amount of RAM used to record just a single 1920x1080p stream was nearly the triple (24 GB) of the amount needed by our solution. Even with a lot of RAM available, our solution always used significantly less.

The maximum number of simultaneous, full high definition (1920x1080p) streams, that CMAM4PRO can operate is three. This is already a significant achievement since the tested non-virtualized solution was only capable of processing 2 streams simultaneously, besides not being able to scale.

5.2.4 Diversity

The next system requirement is the diversity feature, which is the ability to ensure that our approach offers compatibility to a wide variety of options. In this project there are two different aspects in which **CMAM4PRO** offers diversity. One of them being the media processing services compatibility with hundreds of different codecs, formats, and streaming protocols, allowing the solution to offer its user's the capacity of creating thousands of different media processing workflows. The other aspect is related with the compatibility with different cloud providers. The solution is designed to offer compatibility with providers that offer **VM** deployment with the necessary **OS**, which is the Windows Server build 2004. This increases the options of the users when choosing the best provider for their use cases.

The tests previously done for the performance and scalability requirements also tested the ability of the solution to work with a diverse range of codecs and formats. Naturally, not all of the supported codecs and formats were tested because of their high number. But from those tested, the virtualized solution excelled and kept the expected behavior. Table 2.1 and Table 2.2 represent the supported codecs and formats respectively.

5.2.5 Robustness

The last requirement is robustness, this is the ability of our solution to be as error-prone as possible. This is achieved by including the ability of monitoring the state of the solution to ensure it has enough resources and is behaving as expected. This is offered by the monitoring component of our approach that grants the users with resource data extraction from the cloud, as well as customizable dashboards for that data. In this way, the user can monitor the state of the solution to ensure it has enough resources and is behaving as expected. Constant use of these tools and a periodic update of the virtualized software are critical practices that will ensure the availability of the solution and, inherently, its robustness.

The monitoring tools were essential in providing a way to monitor the resource usage of the **VMs** while the tests were executed. This analysis also provides an effort to better understand where the performance gap comes from. As presented before, this was done by using the monitoring software Netdata and the data extraction tool WMI-Exporter. The analyzed metrics provide quantifiable data that allows us to understand the different **VMs** behavior and learn about their resource usage. The metrics gathered were the **CPU** percentage of usage, **RAM** usage, internet bandwidth.

Some reference values were used for comparison, which were taken with the **VMs** at an idle state, that is, not performing any action (see Figure 5.3). This is helpful because the **VMs** can have different workloads at an idle state that might cause a bottleneck of resources when applying a demanding workflow. This can be caused by a different number of processes being executed in the background. The reason behind this could be **OS** updates or security updates to the **VM** itself. But, it can also be linked with the creation process of the **VM**. This process can differ

from provider to provider, causing a slightly different output with different performance.

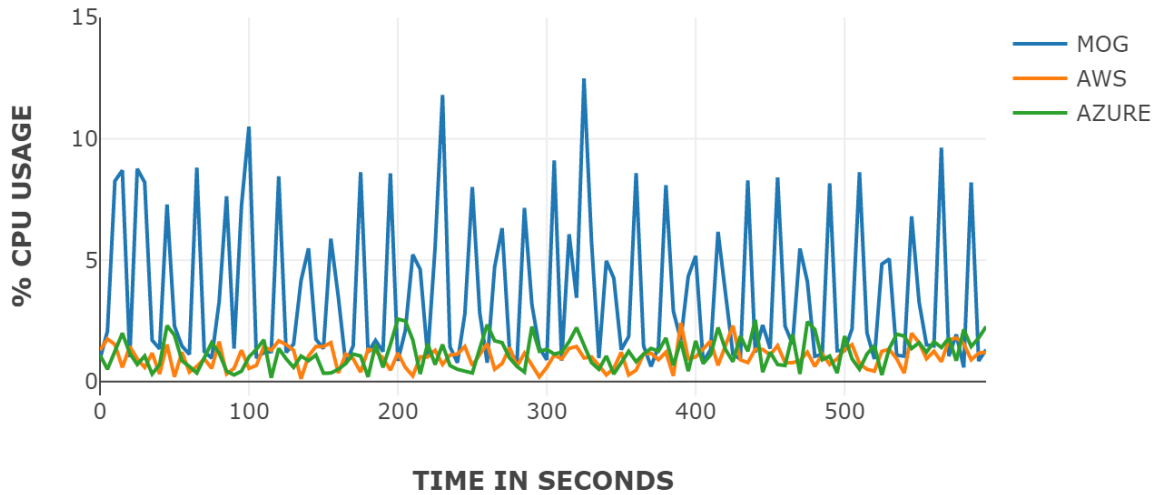


Figure 5.3: CPU % usage at and idle state

Figure 5.3 provides an insight into the CPU percentage of usage that is already useful when identifying possible causes for the performance gap. This insight comes to prove what was previously stated, MOG has more VMs per resource than the public providers. Which effectively results in a higher workload for the hardware hosting the VMs. This can be observed by the higher fluctuation and percentage of usage, even at an idle state.

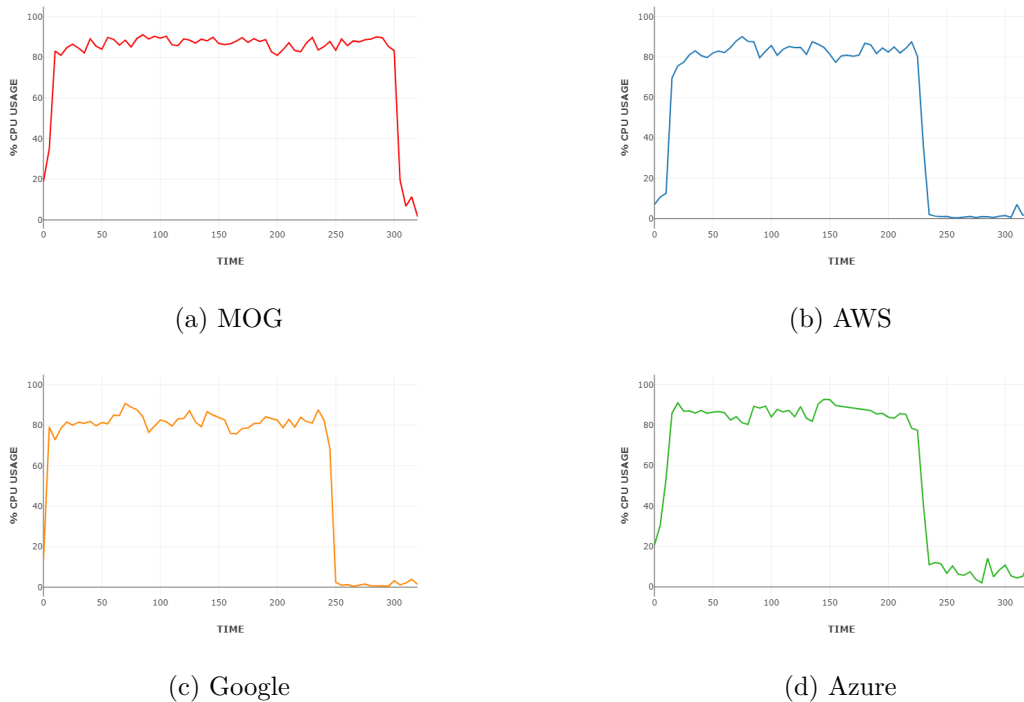


Figure 5.4: CPU usage Workflow Grav - 2160

Figure 5.4 has graphics representing the percentage of **CPU** usage. All the values vary around the same amount, with **MOG**s average values being slightly higher but still within the registered values from the public providers. The **MOG VM** takes a longer time to perform the workflow as expected by the values present on Table 5.5. The maximum value registered for each one was: **AWS** - 90.17%, Google - 90.79 %, **MOG** - 91.17%, and Azure - 92.80%. These values are almost constant for all the workflows, which means that the complexity of the workflow does not influence the amount of **CPU** usage.

After the **CPU** analysis, the **RAM** usage was also compared. Figure 5.5, represents how the amount of **RAM** used changes when the Grav - 2160p workflow is tested.

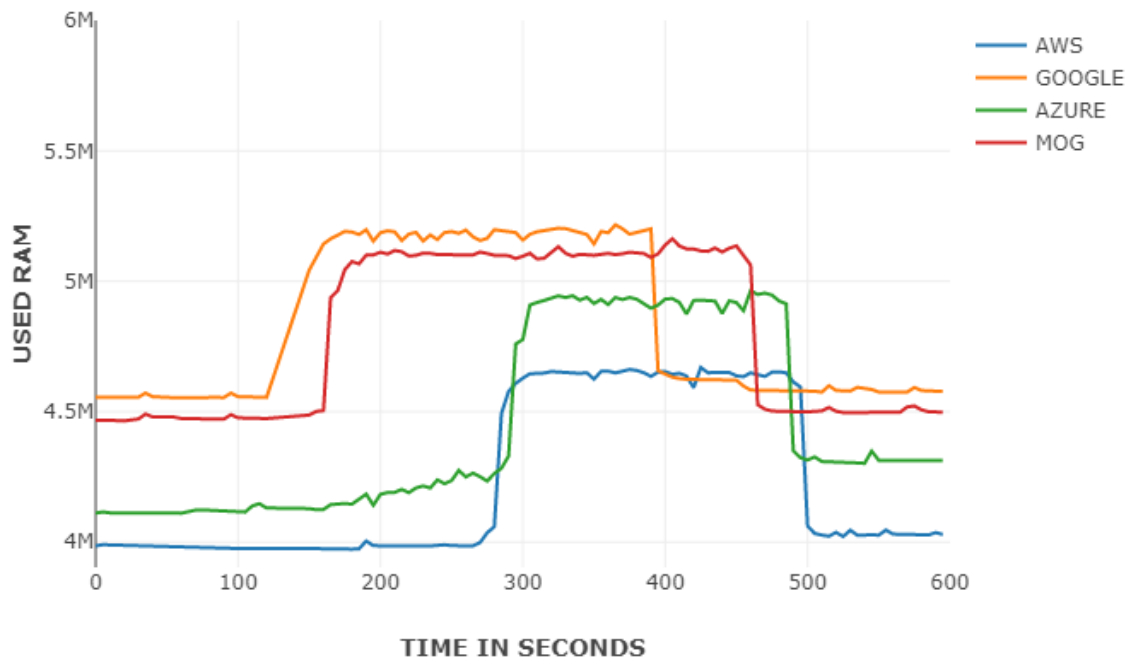


Figure 5.5: **RAM** usage while applying the Grav - 2160p Workflow

The difference in **RAM** usage it's not much. All of the **VMs** are equipped with 32GB of **RAM** and as Figure 5.5 illustrates, the amount of **RAM** used by our solution to perform this test is approximately 600MB. This value is the same for all the providers. It is also possible to see that the **VMs** have different usage when idle. This does not affect the ability to perform our workflows because in these specific tests they did not need a high amount of **RAM** to be executed. But, this varies depending on the type of data being handled. As previously seen, in the live stream tests **CMAM4PRO** is much more reliant on **RAM**. This means that there are use cases where this difference when idle can be significant.

5.3 Tests conclusions

After performing the tests it is possible to conclude that **CMAM4PRO** performed the proposed task successfully. The end product of this project is a fully virtualized media workflow capable of performing the same media operations that the current ones do, while also providing a lightweight, simpler, more accessible, and scalable approach.

The best performing public provider (from the ones tested) is Microsoft Azure, which provided the best performance results in every single test, while the creation and configuration of the **VM** in this provider was also a fast and smooth operation. On the opposite side is Google which was the worst public cloud provider. With the worst performance in all tests although sometimes it had a better stability. Google also had a worst experience when creating and configuring the **VM**. The **VM** would often freeze and had a slower response when working with it.

Recalling that the **VM** from **MOG** was not entirely comparable to the **VMs** from the public providers since it had less available resources and not the same amount of abstraction between the shared hardware. This led to a worst user experience and performance in all tests. Nevertheless, our tests show that even though it had these limitations, it is still possible for a company to host **CMAM4PRO** without recurring to a cloud provider.

Chapter 6

Conclusion

The main goal of this project was to create a cloud-based media processing service that would fill the needs that the current services could not. Receiving, manipulating, and sending media is a process that can require a big investment in hardware, software, and human resources. Migrating this process to the cloud cuts the need to have expensive hardware - like the OB vans explored in this thesis - to perform these actions. The cloud hosts these services on a "pay for what you use" approach that greatly decrease the amount of financial resources needed. It also reduces the amount of human resources and complex logistics inherent with them, since it is available everywhere where an internet connection exists and does not require a physical on-site deployment. The cloud platform also offers scalability, which is a very important feature in the media industry because nowadays the amount of different use cases require the solution to be ready to increase or decrease capacity.

The above mentioned goal was successfully achieved by deploying a media management software in a cloud-based environment. The media software supplied to this project was MAM4PRO, an already well established and widely adopted media manager developed by MOG. To enable its deployment to the cloud, MAM4PRO required virtualization, a task that was done through the container virtualization method. This virtualized version, called CMAM4PRO, was then deployed to multiple cloud providers, and successfully fulfilled all the requirements set for the solution.

Another goal of this project was to test and benchmark the solution in the chosen set of multiple cloud providers. This was performed successfully by creating test scenarios that mimicked real use cases. These tests were then executed on CMAM4PRO and its performance was measured. It was then possible to compare the performance between the deployments in different cloud providers. The resource usage of the VMs hosting CMAM4PRO was measured and their behavior analyzed. To measure the resources and analyze the data it was necessary to add monitoring tools to the solution. These tools not only provided a way to capture and store the resource usage metrics but also added a way for the users to monitor the solution. This is essential when detecting and solving errors. The results obtained proved a great success in creating a working media processing service based on the cloud while also offering a strong

performance. **CMAM4PRO** was able to keep the same performance as MAM4PRO in some of the tests while exceeding the later in others. Specially when working with live streams, where **CMAM4PRO** was able to outperform MAM4PRO even with less hardware resources available.

According to the established requirements, all the set objectives were accomplished. Therefore the result of this project was a solution capable of eliminating the drawbacks that the current approaches have. Succintly this means that **CMAM4PRO** is cost-effective, scalable, more flexible, better in error resistance and detection, and capable of serving a wider range of customer needs, when comparing to the previous solutions.

6.1 Future Work

This section explores the concepts that can, in the future, improve the work done on this thesis. Some of the ideas explored here were not implemented due to the lack of time. Otherwise they would already be part of this project, since they bring either new features to our solution or simplify the way we interact with it. For anyone picking up from this project, this can be seen as the planning for the next steps to take.

Since the solution is designed to be flexible and scalable, it could be added as future work a better management of the multiple instances running our solution. Both inside the same cloud provider as well as across different cloud providers. This could be achieved by taking more advantages of the features offered by Kubernetes. This would mean better control of the resources needed and used by the users and also a better failure recovery system. As stated before, Kubernetes offers auto-recovery systems that automate the launch of new containers in case of failure. Furthermore, there is also the auto-scaling feature that monitors the usage of resources and increases/decreases them as needed. This allows our solution to be even more cost-effective.

Besides the exploration of the Kubernetes features, we could also consider using other cloud providers to deploy our solution. This would give us a bigger sample when judging which provider performs best in the different scenarios of media processing for different client needs.

When managing the cloud resources, there is also a tool that could be added which simplifies this process. This tool is Terraform, a software that enables its users to launch all the cloud resources they want, in every cloud provider of their choice (chosen from a pool of supported providers). Currently, Terraform offers support for hundreds of providers and the integration of this tool in the solution will transform the interaction with the cloud, making it easier, faster and more central, since all the deployments and management would be done in a single platform. Terraform would be integrated with the use of the auto-scaling/auto-recovery features from Kubernetes complementing those features with a all-in-one platform for cloud resource creation and management.

The integration of these features into **CMAM4PRO** in a user-friendly way is very important.

This would require the development of new interactions inside the **CMAM4PRO GUI** that would enable the user to scale the solution in a intuitive way.

Bibliography

- [1] C. Anderson. [Docker](#) [software engineering]. *IEEE Software*, 32(3):102–c3, 2015. doi:10.1109/MS.2015.62.
- [2] Arvind. Docker Architecture: Why is it important? <https://www.edureka.co/blog/docker-architecture/>, 2019. Accessed: 2021-04-22.
- [3] M.G. Avram. [Advantages and challenges of adopting cloud computing from an enterprise perspective](#). *Procedia Technology*, 12:529–534, 2014. ISSN: 2212-0173. The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania. doi:<https://doi.org/10.1016/j.protcy.2013.12.525>.
- [4] Barten, Ben. Docker Hands-On Start for Developers - Part I. <https://benbarten.com/docker-hands-on-start-for-developers-part-i/>, 2019. Accessed: 2021-04-22.
- [5] BRÜNDL, Thomas. Comparing Containers and Virtual Machines. <https://it-infrastructure.solutions/comparing-containers-and-virtual-machines/>, 2020. Accessed: 2021-04-20.
- [6] Mainak Chakraborty and Ajit Pratap Kundan. *Grafana*. Apress, Berkeley, CA, 2021.
- [7] DEĞER, Tayfun. What is Docker? <https://www.tayfundeger.com/what-is-docker.html>, 2019. Accessed: 2021-04-20.
- [8] Grafana Enterprise. [How bloomberg tracks hundreds of billions of data points daily with metricktank and grafana](#). In *How Bloomberg Tracks Hundreds of Billions of Data Points Daily with MetricTank and Grafana*, 2019.
- [9] Grafana Enterprise. [Featured customers](#) (<https://www.featuredcustomers.com/vendor/grafana/customers>). In *Featured Customers*, 2021.
- [10] Graphite Enterprise. [Overview](#) (<https://graphiteapp.org/overview>). In *Overview*, 2021.
- [11] Influx Data Enterprise. [What is influx?](#) (<https://www.influxdata.com/products/influxdb/>). In *What is Influx?*, 2021.
- [12] MongoDB Enterprise. [Mongodb manual](#) (<https://docs.mongodb.com/manual/>). In *MongoDB Manual*, 2021.

- [13] MySQL Enterprise. [Mysql 8.0 reference manual](https://dev.mysql.com/doc/refman/8.0/en/introduction.html) (<https://dev.mysql.com/doc/refman/8.0/en/introduction.html>). In *MySQL 8.0 Reference Manual*, 2021.
- [14] NetCraft Enterprise. [Netcraft web services auditor: Active websites](https://www.netcraft.com/active-sites/) (<https://www.netcraft.com/active-sites/>). In *Netcraft web services auditor: Active Websites*, 2020.
- [15] PostgreSQL Enterprise. [About postgresql](https://www.postgresql.org/about/) (<https://www.postgresql.org/about/>). In *About PostgreSQL*, 2021.
- [16] Prometheus Enterprise. [Overview](https://prometheus.io/docs/introduction/overview/) (<https://prometheus.io/docs/introduction/overview/>). In *Overview*, 2021.
- [17] RedHat Enterprise. [What is kubernetes?](https://www.redhat.com/en/topics/containers/what-is-kubernetes) (<https://www.redhat.com/en/topics/containers/what-is-kubernetes>). In *What is Kubernetes?*, 2020.
- [18] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. [An updated performance comparison of virtual machines and linux containers](#). In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, 2015. doi:10.1109/ISPASS.2015.7095802.
- [19] Noha Hamdy, Amal Elsayed Aboutabl, Nahla ElHaggar, and Mostafa-Sami Mostafa. Resource allocation strategies in cloud computing: Overview. *International Journal of Computer Applications*, 177(4), 2017.
- [20] Per Brinch Hansen. [The nucleus of a multiprogramming system](#). *Commun. ACM*, 13(4): 238–241, April 1970. ISSN: 0001-0782. doi:10.1145/362258.362278.
- [21] Hrishikesh. List Of Essential Kubernetes Tools. <https://www.totalcloud.io/blog/list-of-essential-kubernetes-tools>, 2020. Accessed: 2021-04-24.
- [22] Nectarios Koziris. [Fifty years of evolution in virtualization technologies: From the first ibm machines to modern hyperconverged infrastructures](#). In *Proceedings of the 19th Panhellenic Conference on Informatics, PCI '15*, page 3–4, New York, NY, USA, 2015. Association for Computing Machinery. ISBN: 9781450335515. doi:10.1145/2801948.2802039.
- [23] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20:46–61, 1973.
- [24] Lucidchart. Understanding the basics of cloud computing. <https://www.lucidchart.com/blog/cloud-computing-basics>, 2020. Accessed: 2021-04-28.
- [25] Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. [IaaS cloud architecture: From virtualized datacenters to federated cloud infrastructures](#). *Computer*, 45(12):65–72, 2012. doi:10.1109/MC.2012.76.
- [26] Massachusetts Institute of Technology. Computation Center and F.J. Corbató. [The Compatible Time-sharing System: a Programmer's Guide](#). *The M.I.T. Computation Center: F.J. Corbató [and Others]*. Massachusetts Institute of Technology, 1964.

-
- [27] Docker official documentation. [A docker overview](#). In *A Docker overview*, 2020.
 - [28] Matthew Portnoy. *Virtualization Essentials*. Indianapolis, IN : John Wiley & Sons, 2012.
 - [29] Dimpi Rani and Rajiv Kumar Ranjan. A comparative study of saas, paas and iaas in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6), 2014.
 - [30] J. E. Smith and Ravi Nair. [The architecture of virtual machines](#). *Computer*, 38(5):32–38, 2005. doi:10.1109/MC.2005.173.
 - [31] Srikanth. 3 Top Benefits Of Cloud Computing For Enterprises Can Expand Their Investment. <https://www.techiexpert.com/3-top-benefits-of-cloud-computing-for-enterprises-can-expand-their-investment/>, 2021. Accessed: 2021-05-3.
 - [32] Christopher Strachey. [Time sharing in large fast computers](#). In *Time sharing in large fast computers*. National Research Development Corporation of London, 1959.
 - [33] Docker Website. [Docker hub images](#). In *Docker Hub images*, 2021.