UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Live Migration of Containers in Cloud Computing and Multicloud

Master's Thesis

Master's Degree in Informatics Engineering

AUTHOR: Yerle , Carlos Martín

Tutor: Bernabeu Aubán, José Manuel

External cotutor: GERA, ZOLTAN

ACADEMIC YEAR: 2021/2022

Escola Tècnica Superior d’Enginyeria Informàtica

Universitat Politècnica de València

Eötvös Loránd University

Faculty of Informatics

Live Migration of Containers in Cloud Computing and Multicloud

Master's Thesis

Master's Degree in Computer Engineering

Author: Carlos Martin Yerle

Tutor: Zoltán Gera

Cotutor: Jose Manuel Bernabéu Auban

2021-2022

Acknowledgment

I would like to dedicate the following work to all the people who supported me or who contributed to my being able to carry out this project, with special mention to my mother and my sister, who are the fundamental pillars of my life and who have supported me throughout my bachelor and master's degree.

I would also like to thank my two tutors, Zoltan Gera, my tutor at the University of ELTE, for his great willingness to help me always, and my tutor at the Polytechnic University of Valencia, Jose Manuel Bernabéu Auban, who offered to be my tutor even though we were far away and with little time to spare.

Finally, I do not want to forget Michael Ogbuachi, who was my supervisor at Ericsson, a great help on the subject, a great support to be able to carry out this project and he was very patient with me.

Resumen

En la computación en la nube y los recientes paradigmas de computación distribuida (por ejemplo, sin servidor), los contenedores son clave para proporcionar la agilidad necesaria en el equilibrio de carga altamente flexible, la gestión de fallos de la máquina, el escalado y la gestión de recursos de un servidor. Sin embargo, cómo mover (escalar, redesplegar) estos contenedores, puede seguir siendo problemático, y por ello investigaremos qué tecnologías existen para el proceso de migración en vivo de contenedores en la computación en la nube y más precisamente en multicloud.

Este proceso se analizará tanto para el caso trivial de los contenedores sin estado, donde la única información adicional se almacena en el cliente (su navegador), como las cookies o el almacenamiento local, como en el caso de la migración de contenedores con estado, donde tenemos información adicional almacenada en el servidor que deberá ser tratada de forma especial al migrar (por ejemplo, volúmenes, almacenamiento persistente replicado y similares).

Una vez que descubramos cómo funciona este proceso en la nube tradicional, investigaremos cómo se aplican estas técnicas a la multicloud. En concreto, buscaremos que desafíos se nos plantean en dicho proceso y analizaremos cómo funciona en concreto la plataforma OpenNebula. Plataforma de código abierto que ofrece servicios de migración en vivo que funcionan en la multicloud, pero ninguna solución es perfecta. Esta evaluación también tratará de abordar algunas de las deficiencias de las tecnologías.

Palabras clave: Migración en vivo, Contenedores, Cloud Computing, Multicloud, OpenNebula.

Abstract

In cloud computing and recent distributed computing paradigms (e.g., serverless), containers are key to provide the necessary agility in highly flexible load balancing, machine fault management, scaling and resource management of a server. However, how to move (scale, redeploy) these containers, can still be problematic, and therefore we will investigate what technologies exist for the live migration process of containers in cloud computing and more precisely in multicloud.

This process will be analyzed both for the trivial case of stateless containers, where the only additional information is stored on the client (your browser), such as cookies or local storage, and in the case of stateful container migration, where we have additional information stored on the server that will need to be treated in a special way when migrating (e.g., volumes, replicated persistent storage and the like).

Once we discover how this process works in the traditional cloud, we will investigate how these techniques apply to the multicloud. We will look at what challenges we face in this process and analyses how the OpenNebula platform works in particular. Open-source platform that offers live migration services that work in the multicloud, but no solution is perfect. This assessment will also seek to address some of the shortcomings of the technologies.

Keywords: Live Migration, Containers, Cloud Computing, Multicloud, OpenNebula

Resum

En la computació en el núvol i els recents paradigmes de computació distribuïda (per exemple, sense servidor), els contenidors són clau per a proporcionar l'agilitat necessària en l'equilibri de càrrega altament flexible, la gestió de fallades de la màquina, l'escalat i la gestió de recursos d'un servidor. No obstant això, com moure (escalar, redesplegar) aquests contenidors, pot continuar sent problemàtic, i per això investigarem quines tecnologies existeixen per al procés de migració en viu de contenidors en la computació en el núvol i més precisament en *multicloud.

Aquest procés s'analitzarà tant per al cas trivial dels contenidors sense estat, on l'única informació addicional s'emmagatzema en el client (el seu navegador), com les cookies o l'emmagatzematge local, com en el cas de la migració de contenidors amb estat, on tenim informació addicional emmagatzemada en el servidor que haurà de ser tractada de manera especial en migrar (per exemple, volums, emmagatzematge persistent replicat i similars).

Una vegada que descobrim com funciona aquest procés en el núvol tradicional, investigarem com s'apliquen aquestes tècniques a la *multicloud. En concret, buscarem que desafiaments se'ns plantegen en aquest procés i analitzarem com funciona en concret la plataforma *OpenNebula. Plataforma de codi obert que ofereix serveis de migració en viu que funcionen en la *multicloud, però cap solució és perfecta. Aquesta avaluació també tractarà d'abordar algunes de les deficiències de les tecnologies.

Paraules clau: Migració en viu, Contenidors, Cloud Computing, Multicloud, OpenNebula.

Contents

Figures

1. Introduction

For some time now, containers have become a fundamental part of cloud and multi-cloud computing, fog computing and edge computing. This is partly thanks to the ability of containers to isolate resources between different containers within the same kernel and because of their potential, servers and cloud architecture is becoming application-oriented rather than machine-oriented. The latter allows developers to focus solely on application development without having to keep an eye on the hardware and operating system for data on CPU, memory, and other resource management.

Considering the increasing popularity of containers, the demand for availability and scalability that they have to offer is growing. This problem is addressed by container migration, i.e., moving a container from one machine to another in real time without affecting application operations. The simplest approach would be a cold migration, where the container is stopped at the source, copied to the destination, and restarted at the destination. Clearly this option is not the most sought after due to the downtime provided by such a method. Therefore, live migration is used, where there is minimal downtime on the service while the container is being migrated. But this live migration that seems as trivial as copying/replicating a container presents several questions that we intend to answer with this research. How to scale up, redeploy these containers again? Are there tools on the market that handle this process? Do they cover all the needs of live migration?

1.1. Motivation

During the academic exchange in Budapest, I had to find a project that would be enthusiastic and motivating for my master's thesis. So, I started looking for possible projects offered by professors at my university. I contacted many professors and they all offered different projects and ideas, giving me several thesis projects to choose from.

Within this range of options, there were projects that were outside my field of knowledge and that did not offer the motivation I was looking for, or that would arouse great interest. After much evaluation and analysis of the options I had, the current project was the one I decided on; the two main reasons for choosing it were: that it offered the opportunity to do an internship in a prestigious company like Ericsson, with all the advantages and opportunities that this represents, and it also offered the possibility of doing research in a group with a project with an emerging topic such as Multicloud, which aroused in me great curiosity.

Therefore, once I found the project that most fascinated me and I had the opportunity to be part of a community as important as Ericsson's, I just had to get to work.

1.2. Goals

There are several objectives that in this work I have to fulfill, or I will try to answer the different questions that arise in the current situation of live migration of containers. I will focus this work mainly on Multicloud.

The specific objectives I seek with this research are:

- What is the current situation of live migration of containers for the traditional cloud?
- How does this situation (live migration of containers) translate to the multicloud?
- To investigate how this process is carried out for stateless and stateful applications.
- What tools are offering this service and how do they do it?
- What needs do these applications not meet, or where can they be improved?
- Provide a practical example using an open-source platform with data, time, and references on a migration example. and references on a container lifecycle migration example.

1.3. Memory Structure

Before continuing with this research, the structure of this document is presented, including a brief description of each chapter that follows.

- **Background:**
  Always within the introduction, this sub-section provides the information that has been deemed appropriate to familiarize the reader with the technology.
- **Chapter 2: State of the art**
  In this chapter we will talk about the current situation of our subject, the applications that carry out a similar task to the one we are investigating. We will also look at what options they offer the user with respect to the live migration process of containers. Both for the reader to master the terminology and to know the object of our research.
- **Chapter 3: Problem analysis:**
  Within this chapter, the challenges or problems that have been arising that are related to the container live migration process will be stated. These

problems will be the result of research on the subject and contact with different platforms.

- **Chapter 4: Proposed solution**

  Here will be presented the solutions that I believe can answer the challenges encountered in chapter 3.
- **Chapter 5: Practical Experiment**

  This chapter will highlight everything related to the practical experiment that we will try to conduct. I will try to carry out: Initial idea, technologies used, steps to follow, results and problems.
- **Chapter 6: Conclusions**

  The last chapter of our document, we will present our opinion about the work done and the work that we still must do regarding this topic. In the last chapter of this document, I will give my opinion about the work done and the work done and the work still to be done on this topic.

1.4. Background

this section we present several topics related to the subject to facilitate the understanding of our research. understanding of my research. I will try to ensure that the user can get to the main part of the research and is already familiar with the topic, both with the concepts I will use, as well as with the terminology mentioned. The topics that will appear next are Cloud Computing, Edge Computing, Fog Computing and Multicloud.

Also, I will introduce the topic of live migration of containers, which is the main part of this paper.

1.4.1. Cloud Computing

Few terms have become as popular in recent years in the world of technology as cloud computing. It seems that nowadays everything happens in the "cloud", because without realizing it, we use this type of technology in our daily lives much more than we think we do. Because cloud computing (1) consists, as can be seen in Figure 1, of offering computer services over the network. This enables a business model in which companies pay only for what they use and do not have to incur the expense of buying all the necessary equipment and infrastructure to have it themselves. For ease of understanding, some examples of cloud computing are email, Spotify, Netflix or even Google Docs, etc., where there is no need for any downloading, installation, and access of any software on your PC. You just need a computer and a good Internet connection to enjoy them.

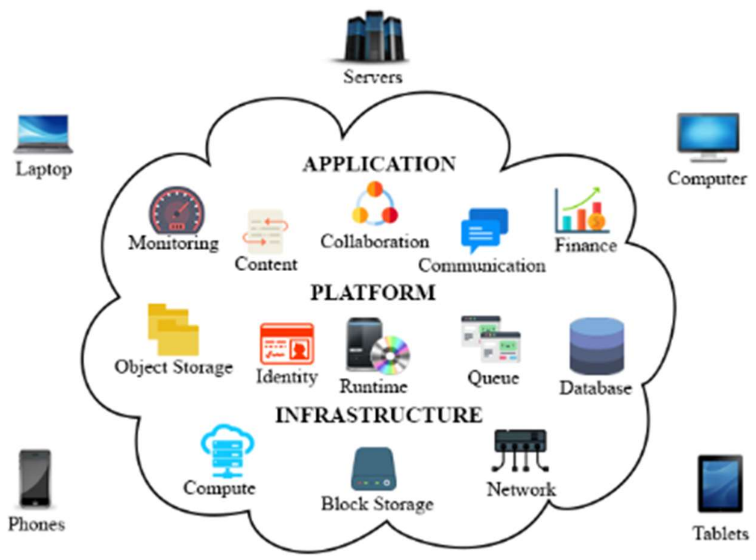


Figure 1 Cloud Computing Example

As mentioned above, cloud computing aims to help people or companies to avoid having to put all the workload on the computer when running applications. Cloud computing offers them:

- Infrastructure: it eliminates the cost of purchasing hardware, software, installation, and maintenance. It also satisfies the need for scaling the infrastructure for each organization. Reduces space savings as resources are stored online.
- Recovery from failures: companies make use of cloud computing to guarantee the continuity of services during possible failures.
- Information: it makes data more accessible, facilitates analysis and backups. This means that information is no longer stored and accessible on a single device but can be accessed by any device with internet access.
- Maintenance: cloud computing service providers are responsible for server maintenance.
- Big Data analysis: Cloud computing has the almost infinite power to process large amounts of data to streamline investigative processes and thus shorten processing time.
- Remote working: it provides the option for a company's employees to access information/data via any computer, tablet, or smartphone with an internet connection.
- Ecology: cloud computing only uses the space/resources on the server.

To simplify how it works, cloud computing systems are divided into two:

- Frontend: here we will find the application with which the user interacts to access the cloud services, and we will also include the computer network.
- Backend**:** this part includes all the systems used by the cloud to offer the service, whether servers, computers, etc.

Cloud computing offers three different types of computing each with different characteristics such as cost, availability, performance, and expectations.

1. Public Cloud: The public cloud offers computational resources such as servers and storage that are provided by third parties and are available to any person or company that wants to hire them. In this type of cloud, the client oversees what will be sent to the cloud, an application, files, backups, while the cloud provider will only have to take care of maintenance, security, and resource management. In this format everything is available on the cloud and shared among several users who use them simultaneously. Examples: Google AppEngine, Sun Cloud, Amazon Elastic Compute Cloud, Windows Azure Service Platform.
2. Private Cloud: In the private cloud model, the company purchasing the service keeps the infrastructure in-house and gives restricted access to selected users, e.g., employees, partners, etc. This type of cloud allows customization of functions and support according to your needs. As a more corporate cloud, it should offer a higher degree of security and privacy than a public cloud, as it will have firewalls and company credentials that will be part of the control process.
   Examples: Cisco CloudCenter, Vmware Cloud Foundation, Azure Private Cloud, Oracle Cloud Platform, Red Hat OpenShift..
3. Hybrid Cloud: When referring to the hybrid cloud, we refer to a combination of the two previous clouds, which makes it better. It allows data and applications to be shared between them. It offers customization of costs according to the needs of each company.
   Examples: Amazon Web Services, Microsoft Azure, Google Cloud.

Cloud computing offers users different types of services (2), which we will mention below.

1. SaaS (Software as a Service): The SaaS service provides access to a software without having to purchase a license, it allows its use through the cloud, which is managed by a provider through the browser. There is no installation of the software on the machines, the user connects through an API. The major benefits of SaaS are

time savings and profit, but it can present certain problems with control, security and performance, which makes it very important to choose the right provider.
Some examples of SaaS are Dropbox, Google Apps, Office 365, etc.

2. PaaS (Platform as a Service): it is the service that provides the hardware and a software platform, mainly this gives developers the option to work, run and manage their projects without having to oversee the infrastructure. PaaS gives you the possibility to avoid the expense and complexity of buying and managing software licenses, infrastructure, and middleware, etc., we only must manage the apps and services we develop while the provider manages all the rest.
   Examples of such services are Microsoft Azure, Heroku, the SAP cloud, etc.
3. IaaS (Infrastructure as a Service): In this case the service offered by IaaS is the payment for infrastructure such as storage or virtualization. We as the user would oversee the operating system, app data, middleware while the provider gives us access and administration to the network, virtualization, servers, and storage. This service allows us to have the flexibility to buy only what we need and to scale as our project requires.
   Examples we can find are Amazon Web Service, Google Cloud, IBM Cloud.

1.4.2. Fog Computing

Although it may seem that nowadays it is all about cloud computing and that this is the solution for everything, we also find some challenges in terms of communication and security that fog computing (3) aims to solve. If we think about applications related to health monitoring, for example, we know that we need low latency, as the delay in response can seriously affect performance. Therefore, the emergence of fog computing places a decentralized infrastructure where computing resources are between the data and the cloud.

This new infrastructure offered by cloud computing makes it possible to place the services that are responsible for working with the data closer to that origin, as can be seen more clearly in Figure 2. This translates into a reduction of the distance that data must travel in the network and thus achieve greater efficiency. We could consider fog computing as an extension of cloud computing, as it is still a paradigm with data, storage, and applications on a distant server and not locally.

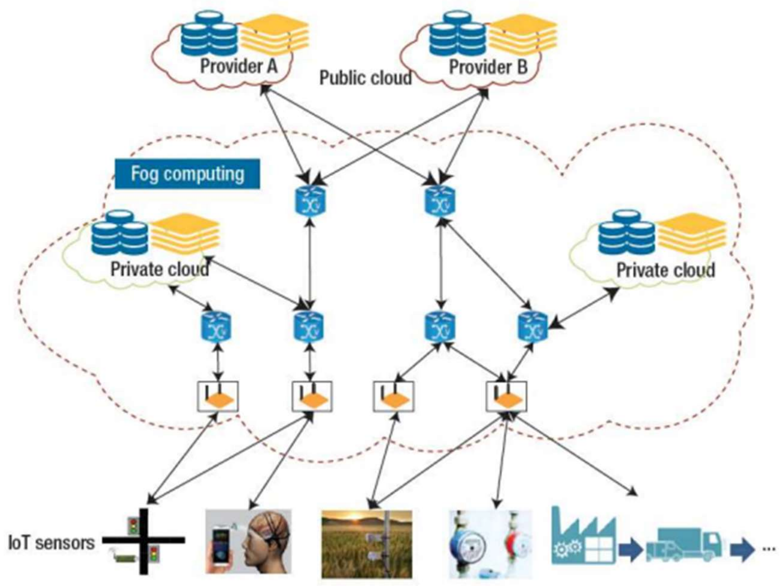


Figure 2 Fog Computing Infrastructure

Within fog computing, different uses or different scenarios can be found in which we can implement this solution:

1. When we have data that needs to be analyzed in very few seconds, i.e., minimal latency.
2. Some devices that are tied to strict computation and processing will use fog computing.
3. Having specific data to leave on the nine, as it will be stored for the long term and will not be consulted very often by the host.
4. Often it will be necessary to offer several services over a large area where the geographical location is not the same.

In addition to the above mentioned, for fog computing the following applications can be found:

1. Monitoring and analysis of patients' condition. Alerts can be used in case of emergency.
2. Possibility to provide rapid response in the railway world with real-time monitoring.
3. Processing of the large amount of data generated by oil and gas pipelines would provide ideal optimization.

1.4.3. Edge Computing

One of the most common situations today within a modern manufacturing plant is the need to process data locally and immediately. This is because you can't wait for data to be transmitted to a data center to be processed as the latency generated could cause major problems. This type of problem is a common situation in Industry 4.0, where IoT sensors create a large stream of data that is often used to prevent breakdowns and improve the process.

So as a solution to the above problem, which is accompanied by many more, edge computing (4) arises, running processes or managing data on the same device or at the same data source. The initial idea is simple, if you can't bring the data closer to the data center, the data center will be the one to bring the compute closer to the data, this can be clearly seen in the example shown in Figure 3. So basically, by bringing the processing power as close as possible to where the data is generated, we speed up the processing and access to the information, so that it can be made available more quickly, reliably, and efficiently.

The reason why edge computing should be implemented is often overlooked, so here are some advantages of edge computing, which have been found in several articles where edge computing was used:

1. More stable, cheaper, and faster service. This is because it offers a faster user experience. On the business side, we have a service with lower latency and high availability that is constantly monitored. It also saves a lot of money in bandwidth to move data back and forth.
2. Reduction of errors in the service thanks to the presence of Big Data in the facilities and allows better control in the transfer of confidential data.
3. Businesses benefit from the flexibility and cost savings that cloud computing provides. This is because we have the processing power close by and regional locations have some independence from the central site in case the central site fails.

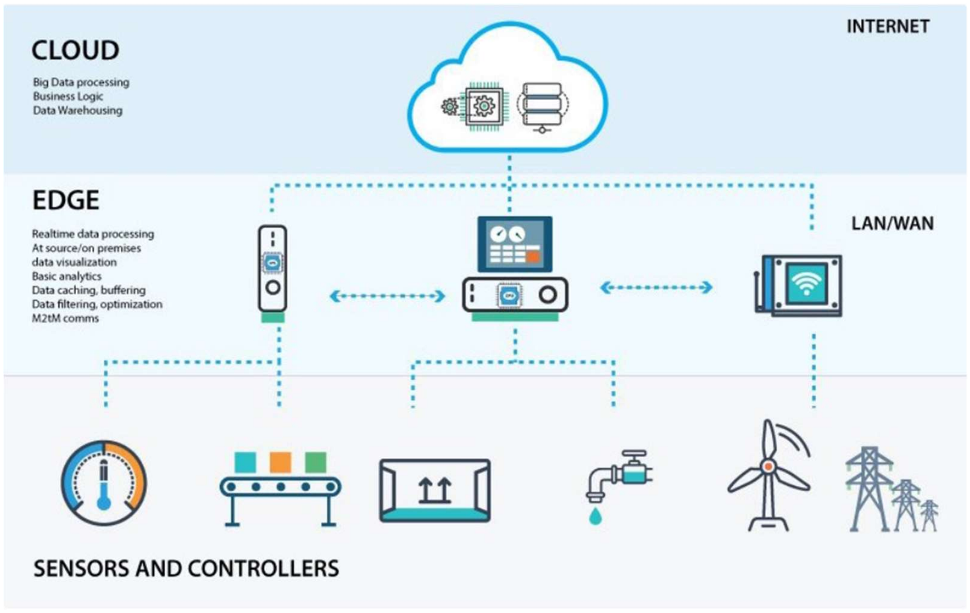


Figure 3 Edge Computing Example

As mentioned above, edge computing techniques are used to filter, process and analyses data "in situ" at or very close to the edge. This is a great weapon for working with large volumes of data that might be too difficult or expensive to move to a central location. In the following, we will look at just a few of the many examples and uses of how edge computing is used:

1. Agriculture: Companies use sensors to monitor water use and nutrient density to calculate the optimal harvest. This data is compiled and analyzed at the edge to further improve cultivation techniques for better yields.
2. Safe workplace: by making use of security cameras, security devices and other sensors, cloud computing helps to set optimal security protocols and to monitor those workers follow these protocols.
3. Manufacturing: perhaps the best-known example is Industry 4.0, where sensors and data are used to control production, predict failures and estimate when machines need to be serviced. All of this is made possible by collecting data and analyzing it at the edge.
4. Healthcare: Healthcare is changing lately, as the amount of data being collected from patients has increased dramatically in recent times thanks to the improvement of devices, sensors and equipment used. All this data needs to be analyzed immediately, and this is where edge computing comes in, offering speed, aiding machine learning and automation.

1.4.4. Difference between Cloud, Fog and Edge Computing

At this point we can establish what the clear differences are between the three types of computing (5). After the above definitions, we can get an idea of what the infrastructure would look like, as shown in Figure 4. Basically, between cloud computing and edge computing the fundamental difference is where the data processing takes place. While in edge computing the processing is done on the devices themselves, fog computing moves the processing activities to LAN hardware that is generally further away than the sensors. Finally, the cloud, an infrastructure completely removed from the user, will be where data that has already passed through one or both two previous parts (edge and fog) will be sent. In the cloud the data will be stored and can be processed as well.

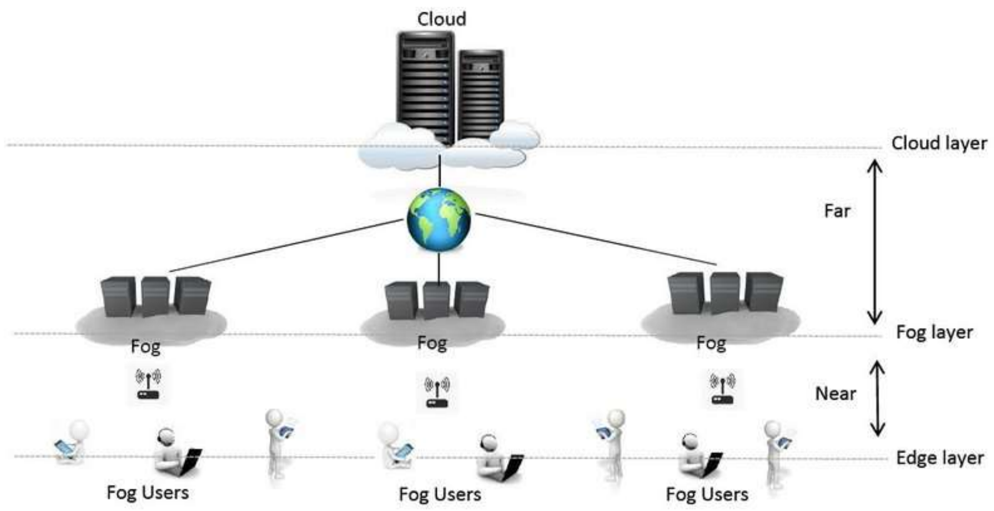


Figure 4 Cloud, Fog and Edge Computing

1.4.5. Multicloud

At this point, it is time to introduce the concept of multi-cloud (6), which refers to the use of cloud services from more than one cloud provider to meet the needs of an organization in terms of infrastructure and services. Within the multi-cloud we can find different strategies to combine public and private clouds as we can see in Figure 5 below. For example, a company can use Google Cloud for development, AWS to take care of data recovery before possible failures and Microsoft Azure for business analytics data processing.

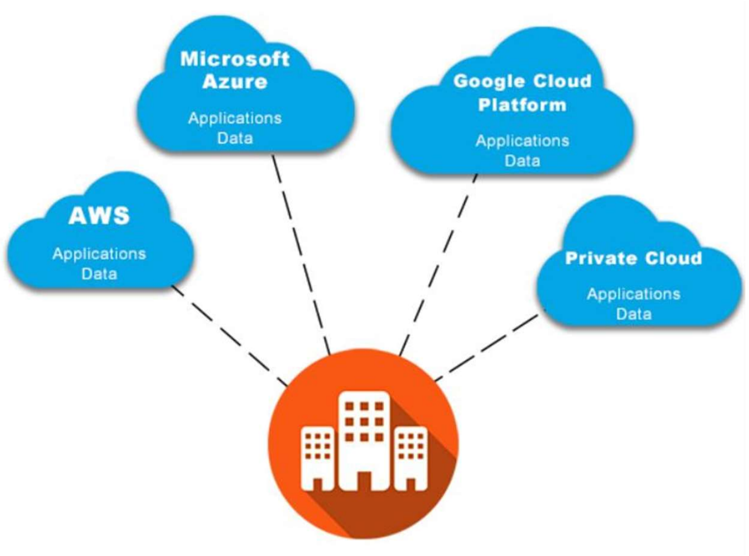


Figure 5 Multicloud Infrastructure

In the following we will see what are the advantages and disadvantages that one could come to have when using multicloud in your projects. Most organizations use multi-cloud architectures to take advantage of the combination of public clouds with private cloud implementations. Some of the advantages found with a multi-cloud architecture include:

1. <u>Reliability:</u> by talking about a multi-cloud architecture, companies ensure that if any cloud fails, some functionality will still be available to users when deployed in other clouds. In addition, you can use other public clouds to back up another cloud.
2. <u>Reduced vendor lock-in:</u> by using a multi-cloud strategy, systems and storage are not all on the same provider, which makes it easier to switch between providers. Also, during the changeover, the entire application infrastructure is not affected.
3. <u>Cost savings:</u> companies have the option to choose the provider that suits them best in terms of price, performance, and security.

But, having seen the above advantages, some possible disadvantages that might be encountered in this type of strategy (multi-cloud) should be mentioned:

1. <u>More difficult to manage:</u> In a multi-cloud strategy, an implementation will have to interact with several different providers, each with different processes and technologies.
2. <u>Increased latency:</u> communication between multi-cloud services can increase latency in the process.

3. Increased vulnerability: in a multi-cloud organized service, there is a larger hardware and software surface to receive possible attacks.
4. Reliability: correct load balancing can often be difficult in a multi-cloud strategy, especially in situations where data is located far apart.

1.4.6. Differences between hybrid and multicloud clouds

The first separation that must be made is that multi-cloud refers to the presence of at least two cloud implementations of the same type (public or private) that come from different providers. While when talking about hybrid cloud, it is referring to the presence of several types of cloud (public or private) that have some level of integration between them.

Within a multi-cloud strategy, it is possible to find the use of two public clouds or two private clouds, but if related to a hybrid approach, one could have a public cloud and a private cloud. Both approaches are mutually exclusive, as it is impossible to implement both at the same time. If the clouds are interconnected, we would be talking about a hybrid cloud, while if, on the contrary, the clouds are not interconnected, we would have a multi-cloud.

1.4.7. Live migration of containers

In this section of the background, the concept of live container migration (7) will be introduced along with its role in the cloud or multi-cloud. The goal is to try to give the reader a clear understanding of the container migration process, what it is for and how important it is today. Furthermore, my research is based on how this process works in the traditional cloud and how it adapts to the multi-cloud.

The first thing that will be explained is that the use of containers to host applications is growing because of the ability of containers to isolate resources between different containers within the same kernel. Due to the popularity of containers and the use that is being made of them, there is a demand for greater availability and scalability. To solve this problem, live migration of containers from one machine to another without stopping the service they provide, or at least with very little downtime, has emerged. The process of moving a container from one server to another is known as migration, which serves to counteract system failures, balance the load, scale, and redistribute resources. When alluding to moving a container application between different physical machines or clouds, we are talking about moving the memory, file system and network connectivity of the container where it runs without the operating system.

Next, with the help of this picture I will try to represent how live migration works from a technical point of view:

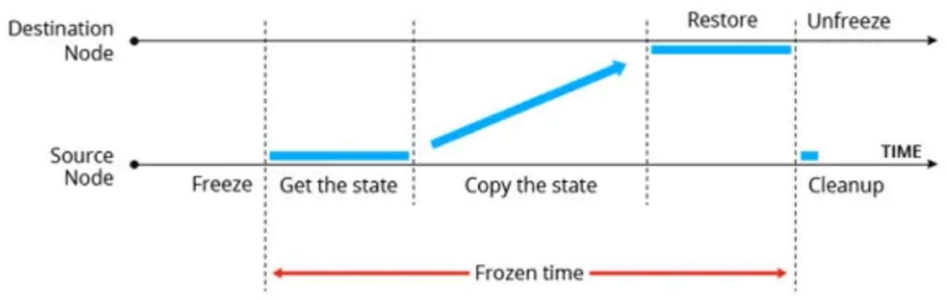


Figure 6 Node Migration Process

As it can be seen in the figure above, the Figure 6, we have two nodes, the source node that will hold the container that will perform the live migration and a destination node that is where we will have the container after the live migration.

The migration process performed by most platforms is very similar, the container is frozen on the source node, blocking memory, processes, file system and network connections, and then recovers its state. The information is then copied from the source node to the destination node. At this node, the platform recovers the state and unfreezes the container. The source node then undergoes a brief clean-up step. You can imagine that the process is simple, get the state, copy the state, and restore the state. However, it is by no means trivial, and it must be taken into account that it will have* a pause or freeze period and that we will have to plan the application architecture with this in mind.

Within live migration several solutions can be found, but the most used ones are pre-copy memory and post-copy memory. We will now look at a simple approximation of how each solution works to understand the process.

Pre-copy memory: For the Pre-copy we have Figure 7 which represents the copying process, when the memory of the source node is tracked and copied to the target node in parallel until a minimum difference between them is obtained. Once this is done, the source container is frozen, the missing state is obtained, migrated to the target node, restored and unfrozen.

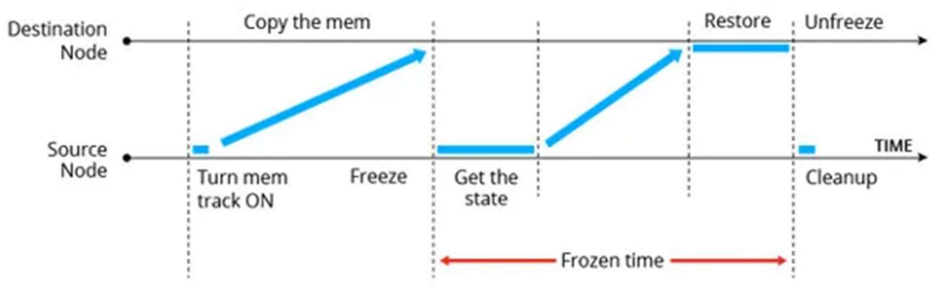


Figure 7 Pre-copy Memory Process

Post-Copy Memory: In this solution, the system freezes the container on the source node in order to obtain the state of the most used memory pages. It moves this state to the destination node, restores it and unfreezes the container. We can see that there is still some memory left to be copied in the source state, this state will be copied to the target in background. All this can be seen in Figure 8 below.

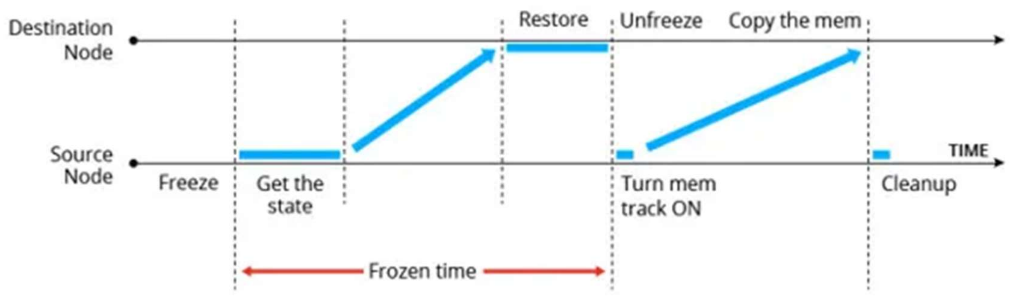


Figure 8 Post-Copy Memory Process

For us to understand what the uses of live migration could be, we have the following examples:

Hardware maintenance: when maintenance is required, containers are migrated from one hardware node to another within the same data center, so there is no downtime.

Load balancing: with live migration we can obtain load rebalancing by live migrating containers from one to another. This action is also offered by several platforms in the form of an automated algorithm.

Cloud provider switching applications can migrate from one service provider to another without reconfiguration or strange changes during the process.

2. State of Art

This section will show some of the platforms that offer live container migration services. Each of them will be presented separately, looking at how they work, what services they offer and how they cover the migration need. But before the platforms, the project that performs the low level migration, and that is used by the different platforms with some variations, will be exposed, and finally, a comparison will be made between them, to determine the common factors, the gaps that still exist within the migration process, etc.

2.1. CRIU

The first of the options to be presented is CRIU, as mentioned above, CRIU is a Linux project that implements the "Checkpoint/Restore In Userspace" functionality which will be used by various platforms as a basis for performing container migration. This Linux software can freeze a running container, check its state on disk and save the necessary data so that the application can be restored in another container. This process can be seen more clearly in Figure 9. Linux thanks to CRIU can perform live migration of containers, snapshots, etc.

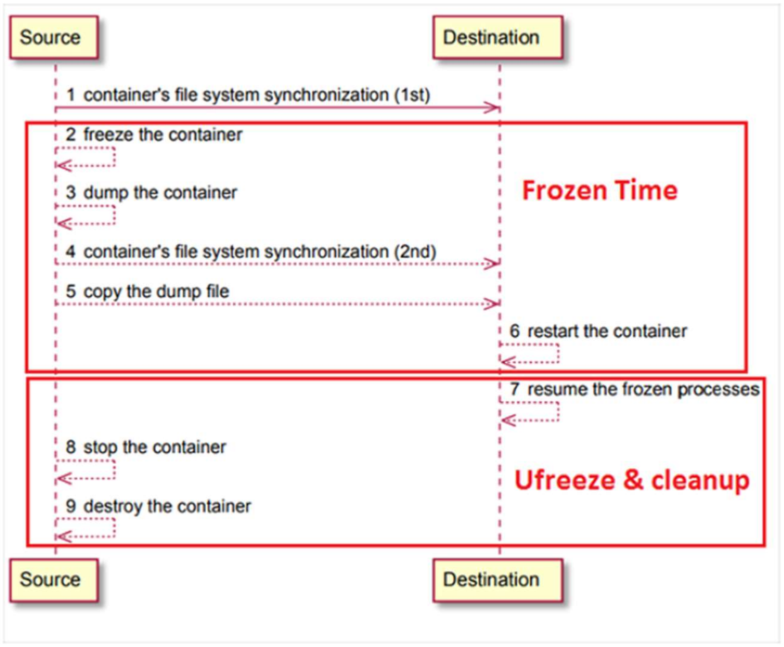


Figure 9 Live Migration in CRIU

When it comes to live migration, CRIU offers the implementation of several algorithms, pre-copy, post-copy, and hybrid-copy, which I have already described in the previous chapter, so now we know how they work in general. In addition to these three, there is also the so-called "image cache/proxy". CRIU has several peculiarities that led to its success in performing this checkpoint/restore process, one of them is that it works at user space level and does not perform its approach inside the kernel. Also, when it comes to restoration, CRIU sets the same PID for the process, both before being frozen and at the time of restoration.

1. Pre-copy migration with CRIU:
   One of the most reliable live migration algorithms for virtual machines and containers are iterative pre-copy live migration. Pre-copying occurs in rounds, with the memory pages to be transferred in round n being those that have been updated after round n - 1 (all pages are copied in the first round). The pre-copy capability in CRIU is implemented as an incremental pre-dump in a Page Table Entry (PTE) using the idea of "soft-dirty bit". The Linux kernel implements the soft-dirty bit function to reduce memory change tracking. By writing the integer 4 to /proc/$PID/clear references, CRIU starts memory tracking. This action tells the kernel to clear the soft-dirty and readable bits of all PTEs associated with the specified process. After that, the soft-dirty flag will be set on the first write operation to a memory page connected to this process. In a subsequent pre-copy cycle, modified PTEs are identified by reading /proc/$PID/page-map, where $PID is the process identifier of the checkpoint process. Modified PTEs are those that have a soft-dirty bit (the 55') reported. The pre-dump action in CRIU allows pre-copy iterations. This allows CRIU to extract only a portion of the information associated with a container (i.e., the memory pages).

2. Post-copy migration with CRIU:
   During live migration, post-copy reduces application downtime. Unlike pre-copy, this approach does not transfer any memory pages until the CPU state has been transferred and resumed on the target host. When the migrated application accesses a missing memory page, CRIU handles the page fault by transferring the necessary page from the source node and injecting it into the memory address space of the running task. This method of on-demand paging ensures that each memory page is only transferred once across the network. However, network latency can degrade the performance of the transferred process and introduce additional high-priority network traffic. By proactively pushing excess memory pages to the destination, residual dependencies are removed from the source host as quickly as possible.

3. Combination of pre-copy and post-copy (Hybrid-copy):

   Pre-copy and post-copy migration algorithms have advantages and disadvantages. Application downtime can be significantly reduced. By copying the appropriate memory pages from the address space of the migration process hierarchy before the final freeze. However, for write memory intensive applications, these pre-copy rounds have a detrimental impact, increasing the overall migration time while reducing application downtime. Post-copy migration offers a solution to this problem. During downtime, the post-copy technique simply transfers the minimum application state necessary to resume execution on the target host. This method reduces application downtime and ensures that each memory page is only delivered once across the network. This strategy, however, has two main shortcomings. First, network latency, which delays memory access, has a substantial impact on application performance during migration time. Secondly, the reliability of this migration procedure is impaired by the inability to recover the running state of the application on both the source and destination host in case of network failure.

Fortunately, the pre-copy and post-copy algorithms can be used together CRIU, and as hybrid-copy is the name for this method. The following are some of the advantages of this combination:

- By streaming memory pages that are periodically updated on demand, application downtime is reduced.
- Once the application has been relocated to the target host, performance degradation is minimized by supplying as many memory pages as possible on the target side before the start of the post-copy phase. For example, before the post-copy phase, read-only regions of the application's memory address space are transferred.
- Compared to Post-copy, reliability is significantly increased by reducing the number of pages transferred on demand.

4. Migration using image cache/proxy with CRIU:

   CRIU was developed to store the operational state of a checkpoint process in persistent storage as a collection of image files. This approach has two key drawbacks in a live migration scenario. For starters, all image files are written to the persistent storage twice: once when the container is dumped on the source host and again when the images are received on the destination host. Secondly, these image files are read from disk twice: once when the images are sent to the destination host and once when the restore procedure is performed. The CRIU manual describes a simple solution to this problem called disk migration, which saves the image files in a temporary file system instead of persistent storage. In the article "ALMA – GC-assisted JVM Live Migration for Java Server Applications" (8) a better approach is proposed by adding two new components to CRIU: image-cache and image-proxy. These components allow the storage and reading of image files to be separated from the dump and restore of the process tree. The execution state of the verification/restore process is transferred via Unix sockets from the CRIU process, and the communication between these two components is done via a TCP socket. By putting the image files in cache rather than in persistent storage, this strategy reduces the overall migration time and downtime for live container migration. Another significant benefit of this strategy is the automated transmission of image files, which simplifies the implementation of live migration with CRIU.) proposes an improved approach by adding two new components to CRIU: image-cache and image-proxy. These components make it possible to separate the storage and reading of image files from the dumping and restoring of the process tree. The execution state of the verification/restore process is transferred via Unix sockets from the CRIU process, and the communication between these two components is done via a TCP socket. By putting the image files in cache rather than in persistent storage, this strategy reduces the overall migration time and downtime for live container migration. Another significant benefit of this approach is the automated transmission of image files, which simplifies the implementation of live migration with CRIU.

2.2. OpenNebula

OpenNebula is the second platform that is going to be introduced and analyzed. OpenNebula is a very powerful open-source platform for building and managing virtualized enterprise clouds and data centers. To deliver these services OpenNebula relies heavily on the use of virtualization and container technology combined with multi-tenancy and elasticity. All this allows us to enjoy in our cloud's flexibility and a total unification between infrastructure management and IT applications. The aim is to avoid vendor lock-in, high resource consumption and operational costs, and to reduce complexity as well.

After a deep study of this platform, it was found that OpenNebula can manage:

- Applications**:** Enables containerized applications by combining Kubernetes and Docker Hub with virtual machine workloads in a similar shared environment to offer the best of both worlds: mature virtualization technology and application container orchestration.
- Infrastructure**:** Combines the expansion of its private cloud with infrastructure resources from third-party bare-metal and public cloud providers create a true hybrid, edge (9) and multi-cloud (10) platform (Equinix Metal).
- Virtualization**:** Use VMware and KVM virtual machines for fully virtualized clouds, LXC system containers for containerized clouds and Firecracker microVMs for serverless deployments to meet your workload needs.
- Time**:** Automatically add and remove new resources to accommodate peak demand, develop fault tolerance techniques or meet latency needs.

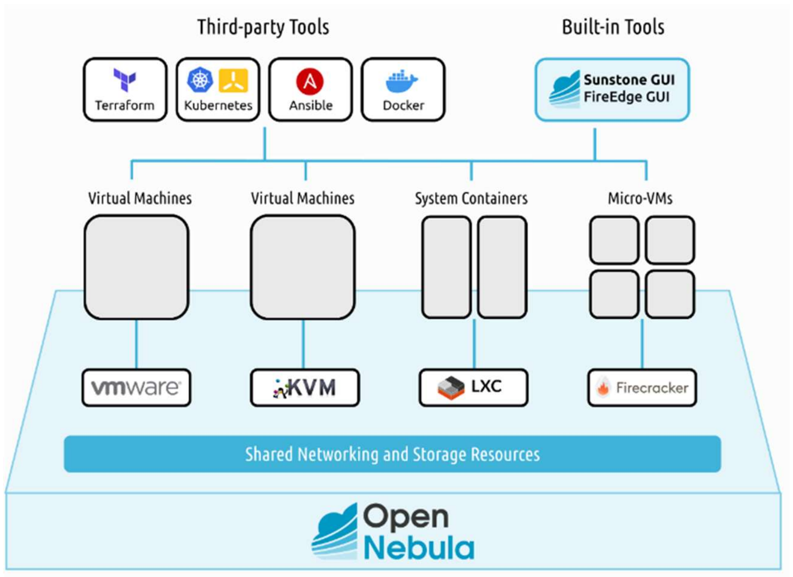


Figure 10 Technologies and Structures offered by OpenNebula

In this research I did not focus on the virtualization part but, to be more precise, on the use of containers (11). Where OpenNebula, as shown in Figure 10, offers containerized applications a natural approach to running containerized applications and workflows by using official Docker images from the Docker Hub and running them as LXC system containers or lightweight Firecracker microVMs

When compared to Kubernetes or OpenShift, this solution combines all the benefits of containers with the security, orchestration, and multi-tenancy features of a robust cloud management platform without introducing unnecessary layers of management, reducing complexity and cost. In those circumstances where Kubernetes is necessary or the best option, OpenNebula also provides support for deploying Kubernetes clusters via a CNCF-certified virtual machine.

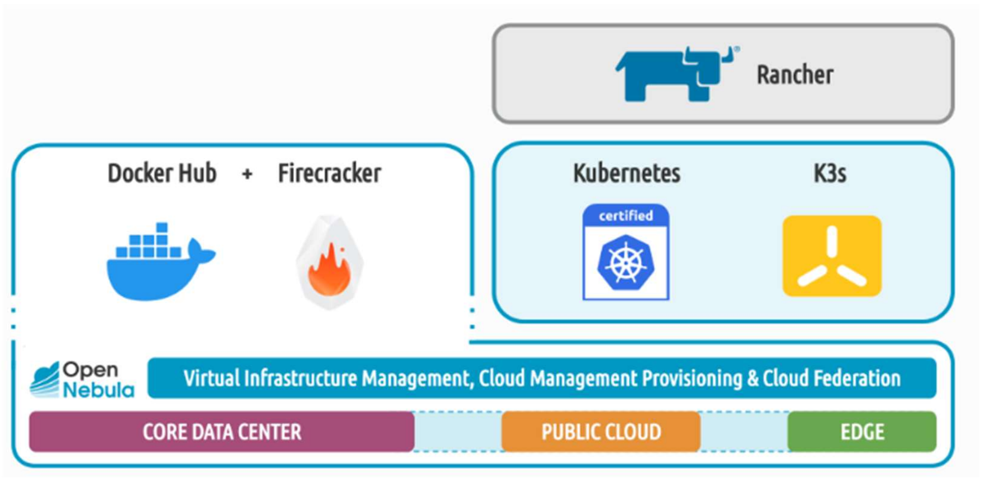


Figure 11 Container Management Structure

As mentioned above, OpenNebula makes use of the LXC system to manage containers, so we will see how it performs live migration of LXC containers.

LXC is a user space interface that through a more than powerful API and some tools, gives Linux users the possibility to create and manage system or application containers in a simple way. One of the peculiarities of LXC is that containers are considered as something in between a chroot and a full virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation, but without the need for a separate kernel. LXC offers a live migration process using lxc-checkpoint, this process is based on the Checkpoint/Restore in Userspace offered by CRIU. The steps involved in the migration process in LXC are:

- Synchronize the file system.
- Dump of all processes using CRIU
- Transfer CRIU dump
- Final file system synchronization
- Restart the container at the destination

In this process the idle time we will have will depend on the memory size of the processes and the rate of change of the file system.

LXC uses the same memory copy algorithms but performs some small optimizations on each of them. The final algorithms that OpenNebula works with using LXC are the following:

Pre-copy migration:

Use the soft-dirty bit of the page table entry.
Freezes the process
Dumps the memory
Process continues to run
Memory is transferred to the destination
Dumps only the memory changes that have changed
Process continues to run
Memory is transferred to the destination
Dump only the memory changes that have changed
Final dump

Post-copy migration (lazy migration):

Based on userfaultfd
Freeze the process
Transfer everything except memory pages
Restart process
Transfer memory pages in case of page fault

Combination of pre-copy and post-copy migration:

First (multiple) pre-copy iterations
Migration to destination
Restoration of lazily missing pages

2.3. Cloudify

Cloudify[1] is the next application that I will analyze and showcase. Cloudify is a TOSCA-based open-source perimeter, cloud and multi-cloud orchestration platform that allows users to model application services and automate their entire lifecycle. Figure 12 gives an understanding on how cloudify enables enterprises to automate their existing infrastructure along with cloud-native and distributed edge resources, enabling a seamless transition to public cloud and cloud-native architecture. It includes deployment in any cloud or data center environment, monitoring all aspects of the deployed application, detecting problems and failures, repairing assets manually or automatically, and handling maintenance tasks. Users can also use Cloudify to manage multiple orchestration and automation domains as part of a single CI/IP pipeline.

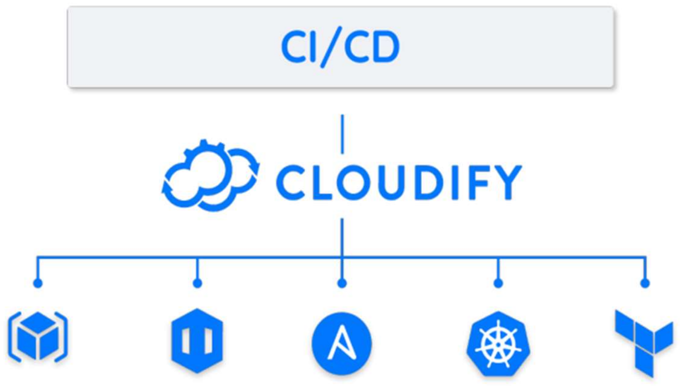


Figure 12 Cloudify Structure

The main features that can be stated about Cloudify after my research are:

- Everything as a Code: the composition of the service is based on DSL (Domain Specific Language), which allows modelling a service composed of several components as shown in Figure 13. Whether they are shared resources such as a database or an exclusive node that makes up a service such as a load balancer. Cloudify specializes in modelling the relationships between services, as well as cascading processes, shared resources, and distributed lifecycle management.

[1] https://cloudify.co/

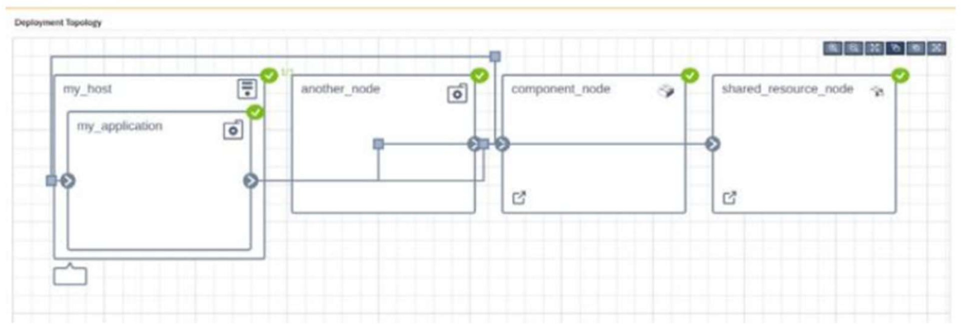


Figure 13 Example of Service Modelling

- Integration with infrastructure orchestration domains is built in, e.g., AWS Cloud formation, Azure ARM, Ansible, Terraform.
- Kubernetes Manager: incorporates the native Kubernetes orchestration manager for cluster management, such as AKS and KubeSpray. Cloudify also offers a built-in blueprint to automate cluster setup and configuration.

In addition to the features mentioned above, Cloudify presents a wide range of possibilities or features that help the user manage their infrastructure. But as this research focuses on live migration of containers, I have decided to limit it to my main topic. Despite researching on the internet and the Cloudify website and requesting concrete information via their website, I was only able to obtain how Cloudify performs the migration of Pods on the same Kubernetes cluster (12).

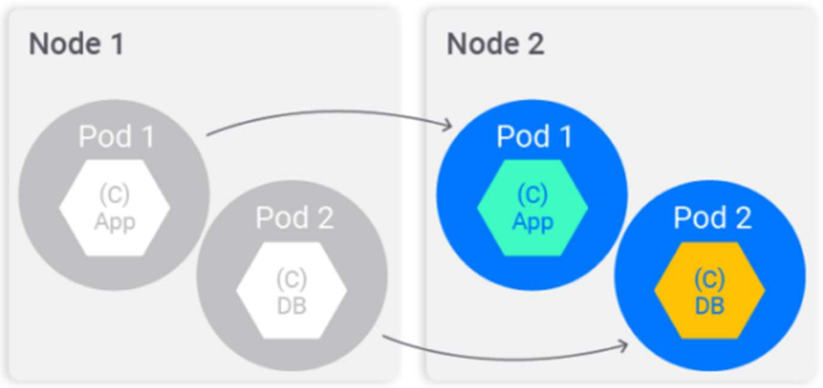


Figure 14 Live Migration of two Pods on Kubernetes

The image above (Figure 14) shows what the objective of the migration is, to move the pods containing the application and the database from one node to another. Cloudify performs the following process using Kubernetes:

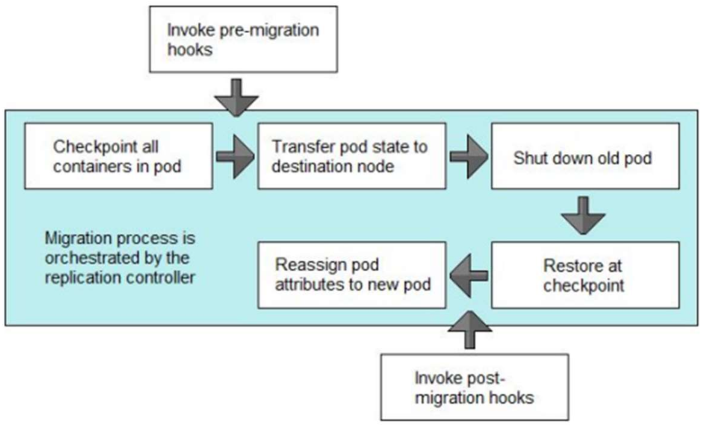

Invoke pre-migration hooks
Checkpoint all containers in pod
Transfer pod state to destination node
Shut down old pod
Migration process is orchestrated by the replication controller
Reassign pod attributes to new pod
Restore at checkpoint
Invoke post-migration hooks

Figure 15 Migration on Kubernetes

In Kubernetes the migration process is based on CRIU's Checkpoint Restore in Userspace which we have mentioned in the previous platforms as well and which also uses Linux. In Kubernetes the pod management (create, delete or update) is handled by kubelets. In Figure 15 you can see the process being carried out and we will briefly explain its steps below.

- Capture container status

  The process of getting the container state is the same as the original CRIU process, the admin must create a helper function to call the Docker checkpoint command, which dumps numerous image files representing the live state of a container. It returns the file handles for further processing once it has located these files.

  But the admin will also have to have another function that does the reverse procedure by calling the restore Docker command, which, given the images representing the container state, relaunches the container in the same state it was in at the time of the checkpoint.
- Transporting Pod State

  A means to transport this data between nodes once we get the container checkpoint images for a pod will be needed. Transferring minor configuration data, such as pod templates, is currently the only example of internodal data communication within Kubernetes. In addition, nodes in the orchestration layer do not currently communicate with each other.

A process at the Kubelet level is required for achieving the following to make the movement of states possible:

1. Within the source pod/node, check all containers, considering any dependency order between containers.
2. Creates a communication channel to a remote destination node, based on the address provided in the input.
3. Send the data to the target node in stages.
4. Continue receiving data until you have obtained the full set of checkpoint capture.
5. Restore each of the containers in the order in which they were created, considering the dependency order.
6. Remove the pod and any containers from the source node once the migration is complete.
7. If a communication problem occurs in the middle of the transfer, we will try to retry several times. We only abandon the process in case of irreversible failure, and the old pod continues to function normally. It is not the task of the Kubelet to ensure the final success of the migration.

Shared pod volumes, in addition to the container state, must be transferred using a similar data transfer technique. At a high level, this should be a less difficult transfer than the operational state because we can simply copy the directory.

- Reassignment of Addressable Properties:
  All accessible attributes, such as hostname, IP address, and active ports, must be identical on the source and destination pods when a pod is relocated. Fields that are not addressed, such as internal Kubernetes IDs and names, are not transferred.
  Currently, Kubernetes lacks a way to transfer IP addresses between pods. Instead, each node is given a set of IP addresses that its pods can use. It is up to the developers to implement a network driver that maintains the logic of that process.
- Orchestration and Coordination of Migration:
  The replication controller on the Kubernetes master node(s) is currently responsible for maintaining the lifecycle of pods, creating, or destroying pods to ensure that there are always a specified number of replicas. We need to extend the controller to also coordinate migration between two nodes.

The tasks of the replication controller are as follows: Since the Kubelets on the nodes already provide an interface to initiate checkpoint, restore and state transfer, the responsibilities of the replication controller are as follows:

1. On the source and destination nodes, start the pod snapshot and transfer procedure.
2. As with building a new pod, ensure that no requests are forwarded to the pod on the destination node before the migration is completed.
3. Monitor the migration process and detect problems or the completion of the state transfer.
4. Once the target pod is up and running, the replication controller calls the network service mentioned above to migrate the IP from the old pod to the new one, thus routing all traffic to the new pod. The old pod is then discarded. During this final transition phase, there may be a small service interruption.

Before continuing with the study of the following platforms, I would like to point out that I know that Cloudify offers the service of live migration of containers between different clusters, from one machine to another and from one provider to another. The only problem is that I was not provided with information about the algorithms it uses or how it performs this process. That is the reason why the study of pod migration in the same cluster is presented. Now I can only wait for the practical part of the research, where the service offered by Cloudify will be tested.

3. Analysis of the problem

The upcoming part of this thesis, it will be focused on analyzing the challenges I have encountered in live container migration or the questions that arose when analyzing previous platforms. The first approach leads to make a separation between the possible challenges that could be encountered in stateful and stateless. Although this research is mostly based on stateful, when talking about stateless, a less problematic process or fewer unknowns will not be found.

Clearly when talking about live migration in containers from a stateless point of view, the first thing that comes to mind is that it would be a trivial, simple process. For example, we copy the container to move, we move it, we start the new container, and the process is finished, but this is far from reality. Because, even if you don't have to deal with the problems about data storage in the container as in stateful, both situations (stateless and stateful) share, there are some factors that have to be taken into account: Does the architecture of the new "receiver" node match the architecture of the container that is going to migrate, it will be necessary to make sure that the migration fits the requirements and that there will be no consequences or at least that those consequences can be dealt with. Another challenge to consider is how does the migration process handle the IP tables? What happens to the DNS domain in the migration from the source node to the destination node?

But as mentioned above, there is another part, when the container has state (stored information), then the problem or the number of challenges increases. Moreover, the problems to be tackled are not purely IT or technical, but will also involve predicaments of various natures, such as ethical (different data protection policies in different countries) or energetic (resource expenditure that may be involved in migrating to the new host) problems.

To make it easier to read and organize, I have decided to list the challenges or problems I have faced, whether they are stateless or stateful. Each challenge will be presented with an explanation, as well as the questions that in the following chapters I will attempt to answer.

3.1. Data Protection

In recent years, one of the most controversial and important issues is the data processing used in each process. And of course, the live migration of containers does not escape this problem. Sometimes the migration process will place the container in a region with the same data protection policy. Such as a migration within the same country or within the European Union, for example. But now the question arises, what would the situation be like if the container were to migrate, for example, from the United States to Europe or vice versa?

Because to get an idea, data protection in the United States is different from that in Europe. The CLOUD Act allows Internet service providers and other electronic media companies based in the US to store and transmit your sensitive data. In Europe, by contrast, there are public bodies dedicated to the protection of your personal data. European data protection legislation protects you whether you are European or not, and only collects your essential data, or so they say.

So, should we continue to treat the data with the data policy where the container comes from? Should the data owners be informed of the change and decide what to do with their data? Should we create a global law to be used in case of a change of data location? Could we create something to check if the migration respects the data policy?

3.2. Individual or group migration

When thinking about live migration of containers, the first thing that comes to mind is to try to minimize service downtime, and that is the goal, and clearly when our service to be migrated consists of a single container, we have no major challenges or options. We migrate the container that hosts the service and that's it. But what happens when our service consists of more than one container. Because it is well known that nowadays many applications that offer a service are made up of microservices that, all together, make up the entire app. All these microservices must be interconnected with each other and clearly know how to access or communicate with the rest of the parts of the application.

At this point, then, there are two different possible scenarios:

- When we have a service composed of microservices (e.g., user interface, ordering, payment, shipping) with the need to do live migration, how do we migrate the containers of the microservices, is the process done separately one by one, would it be given a group treatment, for which microservice should we start the migration, for which microservice should we start the migration?
- There is a service composed of different microservices, where for different reasons, one of those microservices is replicated because it receives many requests or because the processing of that information is slow and expensive

(image recognition, neural network, etc.). Now, let's suppose that a live migration of the service must be performed, how would the process be with these two equal microservices? Would the migration end when the two containers have migrated, or would the service be offered again when one of the containers is available?

3.3. Energy costs

Another of the big points to consider in any project or process is the cost of this activity. As could not be otherwise, live migration of containers raises doubts about the cost that it would have in different scenarios. We know that when we talk about small applications or services, the cost will be affordable and not unreasonable. But when we talk about large applications, large-scale services, how much does the cost of the process start to weigh? Is it still a viable and profitable process?

3.4. Source Node Failure

During the live migration process many applications make use of the CRIU (Checkpoint/Restore In Userspace) system where there are several memory copying algorithms. These algorithms, which have already been detailed in previous chapters, raise some questions about possible situations. For example, what would happen if in the middle of the copy the start container stops working, and the end container cannot access the memory addresses? For example, in the post-copy algorithm, the new container is accessing the memory pages once the migration is done, what would happen there, if it could not access those pages?

On the other hand, there is the pre-copy algorithm, where the memory pages are obtained before performing the migration, it is imaginable that in this case the migration would not take place, but is the process aborted? Are there measures that control, prevent, or contemplate this error?

3.5. Architecture, Networking and Filesystem

Another question I was asked was about the things that should be checked before the migration. For example, CPU compatibility between both nodes (source and destination), checking and loading the necessary kernel modules such as filesystems. Non-shared filesystems must be copied as well. In addition, we cannot forget about the network aspect, such as iptables and the DNS domain. How does the destination node make the DNS of the source node available? How does the live migration process handle the redirection of network traffic? How could we solve this in such a way that it is sure that at the end of the migration the destination node is completely the same and reachable as the source node?

4. Proposed Solutions

Within this chapter, I will present the possible solutions I have thought of for some of the challenges mentioned in chapter 3. The proposals that follow are all theoretical; due to time, resource and logistical problems, no implementations were carried out. As a matter of organization and structure, I have decided to present each of the possible solutions at different points within this chapter, offering for each solution a detailed explanation as well as the necessary diagrams to clarify my ideas.

4.1. Live Migration based on Privacy Certificates

As mentioned in the problem analysis chapter, data protection is an issue to be considered when accepting or not a live migration, or when choosing a target host. To overcome this, I thought of performing a live migration based on trust verification of the host machines. In other words, that the host that will receive our container complies with the same data protection standards as the current host. This chain of trust between the different machines of a multicloud infrastructure is obtained thanks to the attribute certificates issued by trusted entities. These entities are the PCA (Privacy Certificate Authority)4. Once the security of the data in the movement between the host machines (Host source, Host destination) has been confirmed, the live migration process is carried out.

As you could see in our idea, we are adding new elements to the formula. So, to get to a better understanding of what my idea is, Figure 16 shows the new elements inside the Host.

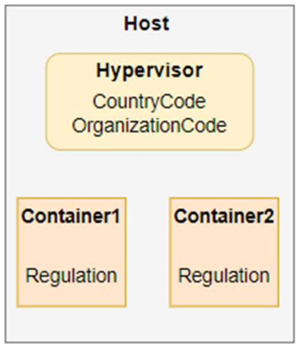


Figure 16 Overview of the Host

As it is shown, each container has a new attribute which is Regulation, this has a list of:

- Country Code: indicates which countries the container can be moved to.
- Organization Code: indicates which organizations make use of the container.

The container must acquire all this information to be executed and then migrated. Clearly this new information will have to be beaconed and secured with an attribute certificate.

On the other hand, we see that the Host Hypervisor also has two new components:

- Country Code: indicates the country code of the country in which the Host is located.
- Organization Code: list of organizations that will use the container.

Now that I have presented what the new Host and container structure would look like, we can move on to see what our proposed multi-cloud infrastructure implementation idea would look like. For that we must look at Figure 17 where we can see that the data protection is done in two parts:

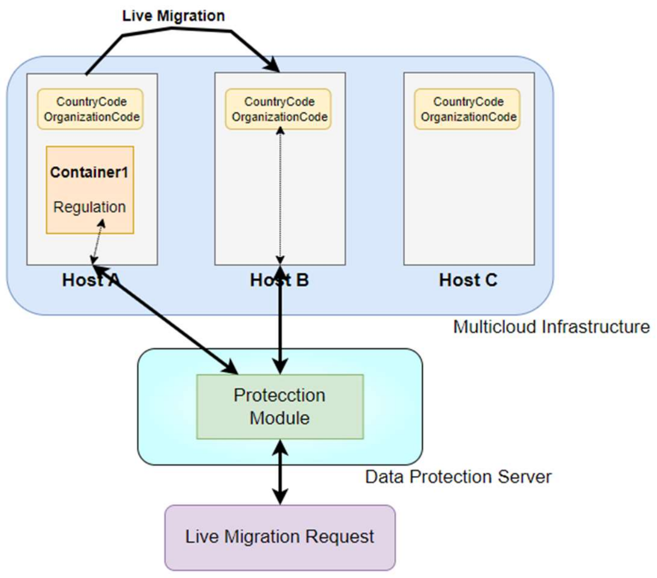


Figure 17 Implementation Proposal

Protection Module: it oversees receiving the migration request and deciding whether the migration can be carried out or not. It can do this because it will oversee requesting the necessary information from both the container and the hosts.

Multicloud infrastructure: here there will be almost no changes with respect to the multicloud infrastructure we are used to. Only that, as we saw before, the hosts where the containers and the containers will be hosted have some more attributes.

In order to clarify a little more how we think our idea should work, in Figure 18, we set out a data flow of what the live migration would look like using our data protection process.

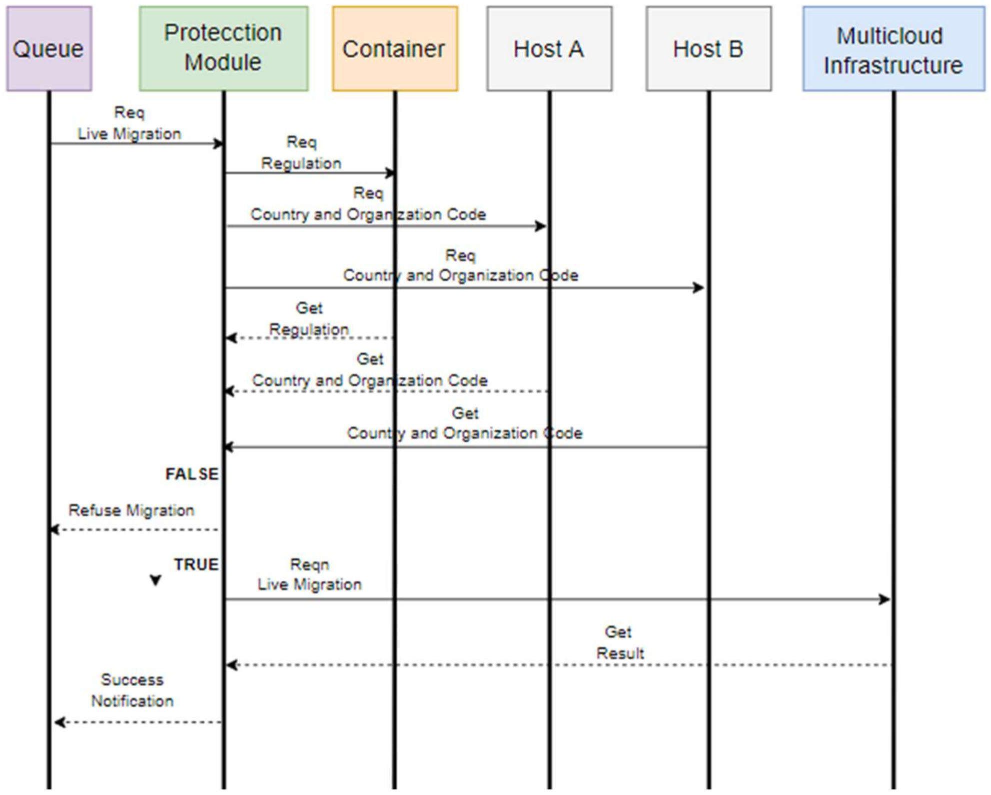


Figure 18 Live Migration Data Flow

Looking at the figure above one can see that what we have in the first step is the migration request that arrives to the protection module. Then, this module sends a request to the container to be migrated to obtain its Regulation attribute, another request to the Host source and to the Host destination to obtain the Country and Organization Codes. Once the protection module has received the response to its requests, it will be able to decide whether the migration can be performed or not. If the migration is not allowed, it responds to your request with an error notification. But if the result is positive and the migration can be performed, then the module sends the command to the multicloud infrastructure to perform the migration live. Once the migration is finished, the infrastructure will send you the migration results and the protection module will send a notification of successful migration.

Finally, it is important not to forget about security, based on the assumption that we have all trusted machines in the infrastructure. In other words, malicious attacks on hosts belonging to the infrastructure are not contemplated in this proposal. For communications we could use TLS connections. It is also considered that the security policy of the countries and organization is complied with. All the attributes I have included are protected by the electronic signature of the attribute certificate.

4.2. Preservation of the DNS domain

When in chapter 3 the challenges of live container migration were explained, I mentioned the DNS domain as a possible unknown. How would the process be able to maintain the DNS domain of the migrated application, or how would it adapt it, in short, what would be the DNS strategy to keep everything working correctly after the migration. For the following solution, I have relied on the knowledge previously learned in some networking courses at university, with some videos from the internet and with the information I found on the internet (13) that I could use to implement my idea.

The DNS domain of the destination node is different from the domain of the source node. The applications automatically receive the FQDNs of the destination node after migration. What I thought, is that in order to keep the source DNS domain of the container/applications performing the live migration I could use one of the two options below. The two strategies proposed to maintain the DNS domain are:

1. <u>The DNS domain of the destination is isolated from the clients:</u>

Without exposing the destination node to clients, you can allow queries sent by clients to the DNS domain of the source node to reach the DNS domain of the destination node. The steps that should be followed would be as follows:

A. Locate between the clients and the target cluster a component such as a load balancer or another component that can be placed in the external network.
B. In order to return the IP address of the load balancer, update the FQDN of the application on the source node in the DNS server.
C. Configure the network component to route any request from the source domain application to the load balancer of the destination node domain.
D. Make a new DNS record (we can call it a helper) for the domain pointing to the load balancer of the source cluster IP address. *.apps.source.example.com
E. For each application, set a DNS entry pointing to the IP address of the external network element in front of the destination node. When resolving the application's FQDN, there is no conflict, as a particular DNS record has higher priority than a wildcard record.

All secure TLS connections must be closed by the external network component. The FQDN of the target application is made available to the client and we will have certificate failures if connections reach the load balancer of the target node. Clients must not receive connections from applications pointing to the target node's domain. Otherwise, the program may not load or function correctly in some areas.

2. <u>Configure the acceptance of the source DNS domain in the destination node:</u>

We need to configure the destination cluster to recognise requests coming from the migrated applications in the DNS domain of the source cluster. Clearly there are several ways to do this, but we believe it could be done in the following way for both HTTP and HTTPS access:

A. Create a route on the destination node that accepts requests addressed to the FQDN of the source node. What we are looking for is for the server to accept any request from the FQDN and send it to the necessary nodes. Also, when the migration is performed, a new route will be established in the domain of the destination node.
B. To route the FQDN of the application on the source node to the IP address of the default load balancer on the destination cluster, we create a DNS record with your DNS provider. By doing this, traffic will be diverted from our source node to our destination node. The load balancer of the destination node can be reached using the FQDN of the application. Due to an exposed route for that hostname, the default ingress controller router will accept requests for that FQDN.

We think this is enough for an HTTP connection, but for secure HTTPS connections, we should take a few more steps:

- We should update the certificate of public and private keys, etc. (x.509) of the entry controller that was configured at the beginning of the installation. To the certificate we should add the DNS domains created for the migration process, both the DNS of the destination node and those created to manage the process.

Once the above certificate has been created, our application is ready to protect all connections to any DNS.

4.3. Network traffic redirection

Another problem presented and analyzed is how to re-establish the network traffic with the new node at the end of the migration process. Therefore, we will present different strategies for network traffic redirection in the following.

Before presenting our options, we should mention that we start from some firsts, such as that:

- We have the applications already running on both the target node and the source node.
- The applications have the path to access the source node host.
- We will have already processed the CA certificate so that the route to the source node host already has it. In addition, the CA certificate of the destination router will already have the DNS record of the source node in its possession.

Having established the basics, we will now turn to the possible strategies that we believe could be put in place to redirect network traffic:

1. Synchronized redirection of network traffic for all applications. Modify the virtual IP address of the destination node's router in the DNS wildcard (VIP) record of the source node. We believe that this strategy is best suited for simple applications with small migrations.
2. Redirect network traffic for specific purposes. Make a DNS entry for each application pointing to the VIP of the destination node's router from the source node's hostname. The DNS helper record (record created specifically for this purpose, as a wildcard) of the source cluster is replaced by this DNS entry.
3. Make use of a proxy to direct traffic to the source node's router and the destination node's router. Create a DNS record for all applications pointing to the proxy with the name of the source node. Configure so that traffic arriving at the proxy entry is directed to the source node and not to the source. But clearly, we will do this gradually, until we reach the totality of the traffic. This could be called a gradual redirection of network traffic.

These have been the three strategies we have come up with to carry out the network re-routing process, but we are aware that there are others, and that some of ours are applicable only in specific situations.

4.4. Live migration of multiple containers

As mentioned in the previous chapter, most services today are composed of microservices. Usually, these microservices are each in a different container. Often these containers are hosted on the same host or not. So, this proposal is to try to solve the challenge of migrating those containers, whether to do it sequentially or in parallel. Opting, in the end, for the parallel migration solution. This would be possible thanks to an application migration middleware that would manage the parallel migration of the containers between the source node and the possible destination node. It was also mentioned that this would be done by making use of multiple TCP (parallel TCP) connections, which would allow more than one container to be sent at the same time.

Next, in Figure 19 it is described an example of what the inclusion of our middleware in an application would look like.

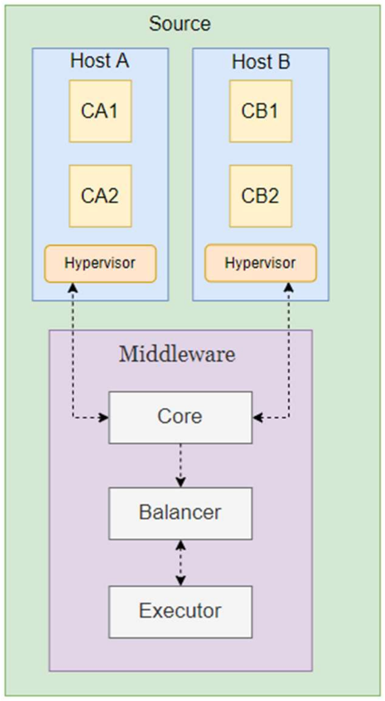


Figure 19 Structure with the inclusion of our Middleware

As we can see in the previous figure, there are two different source hosts (Host A and Host B) that each one has two containers (CA1, CA2, CB1, CB2) that we will try to migrate to Host C and Host D. We can also see that our middleware appears, where we can see which are the elements that compose it. Now, before explaining the process of this idea, there is an enumeration and explanation of what the (instead of ours) middleware will be composed of:

- Middleware core:
  The middleware has all the requirements of the application (resources, locations) and the hosts it has. When the live migration action is executed, the middleware core will communicate with another middleware core to get the information of the target resources. If the requirements are met, the middleware core will launch the migration requests.
- Load Balancer:
  Our middleware will have a load balancer that will have two tasks:
  - Calculate according to the bandwidth of the connections how many containers can be sent in parallel. The start will always be 1 container. But seeing the size of the connection, the load balancer will change that number.
  - Select which containers will migrate in parallel. Because if we have the same replicated container, there is no challenge, it can be done in rotation or depending on the algorithm. But if we are talking about something more complex, where each container is different, then the balancer will have to use another kind of tactic to choose which container to migrate.
- Executor:
  This will be in charge of carrying out the migration, creating the snapshot of the container and migrating the container. All this with the help of the software chosen to manage the virtualization and migration.

Previously, it has been seen that the core of the middleware is in charge of launching the migration request, for this purpose it would be necessary to have a registry with the available containers and their states. These states can be 5:

A. Start, Start phase of the request
B. Screenshoot, when the snapshot of the container is being taken.
C. Ready, when the container is ready to be migrated.
D. In Process, when the live migration process is in progress.

E. Migrated, when the container is already in the destination node.

In Figure 20 below, there is an example of what the live migration process would look like using my solution.

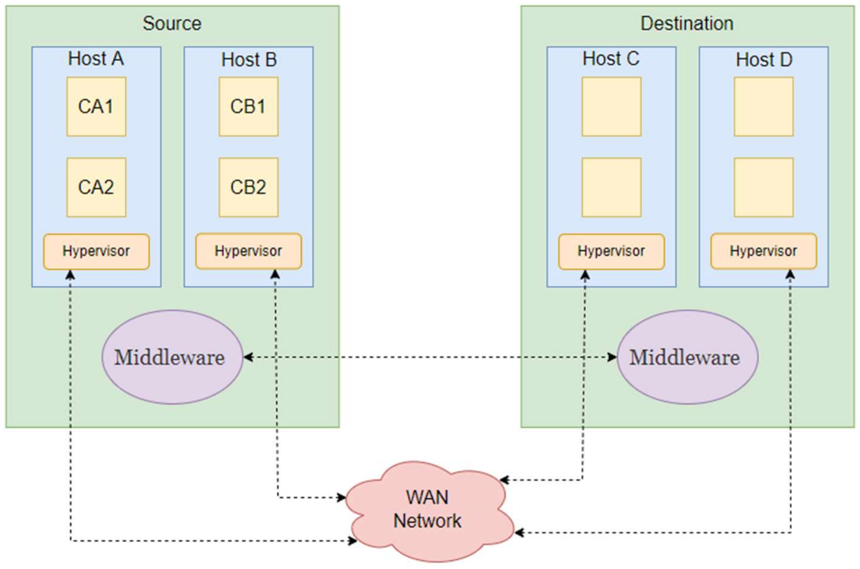


Figure 20 Example of live migration process with our middleware

To finish with my idea, and to explain the previous figure, I will also explain how the live migration process would be using the middleware:

1. We start by establishing communication between the different middleware cores. The objective is to identify if the target resources meet the needs to perform the migration.
2. Once the core knows that it can perform the migration, it creates a request for each container to be migrated and places it in the queue of the balancer.
3. The balancer will start selecting requests according to the selection algorithm it has and the number of containers it can send in parallel (bandwidth) and will send them to the executor to perform the migration.
4. The executor carries out the migration process with the help of the chosen software. It updates the status log of the containers.
5. Each time a migration is finished (either individual or group) the balancer checks the bandwidth to re-estimate the number of containers that can be sent in the next iteration.
6. This process is repeated as long as we have requests in the balancer queue.

So, this has been my idea to solve the multi-container migration challenge. It wouldn't be possible to know the feasibility of my idea until we implement it. But we are confident that we are looking at a viable and implementable proposal.

5. Practical Experiment

After having seen the exposed solutions, the aim is to present in this chapter everything related to the practical experiment that I will do to carry out to check first-hand how live migration works in one of the platforms mentioned in chapter 2. It is going to present the idea of the experiment, explaining what is sought, the technologies used to carry it out, the steps that would have to be performed in the experiment, and will finish with the presentation of the results and/or the possible problems encountered during the execution of the experiment.

5.1. Live Migration Test Proposal

The first thing to be mentioned is that the chosen platform to carry out our experiment was Opennebula[2]. Our experiment is based on developing a small service composed of microservices, where there will be an attempt to perform the live migration process using Opennebula in order to obtain data that will be studied at the end of this chapter. The service will be deployed as shown in Figure 19, initially on Google Cloud Platform[3].

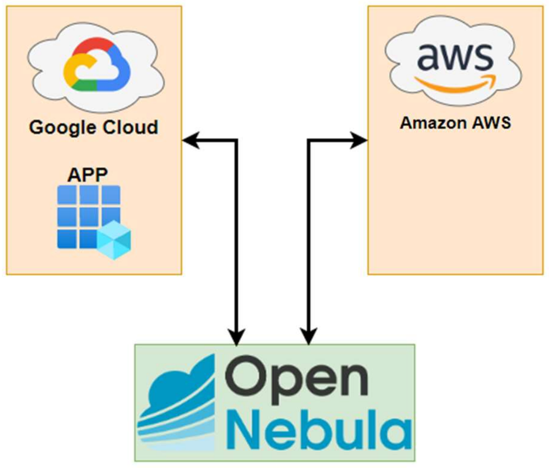


Figure 21 Initial phase of our deployment

Then, using OpenNebula we will look to migrate our server to move it to Amazon Web Services. The next two stages can be seen in figure 20 and figure 21, where we can see that our server is idle due to the migration, ending up running on the final provider.

[2] https://opennebula.io/
[3] https://cloud.google.com/

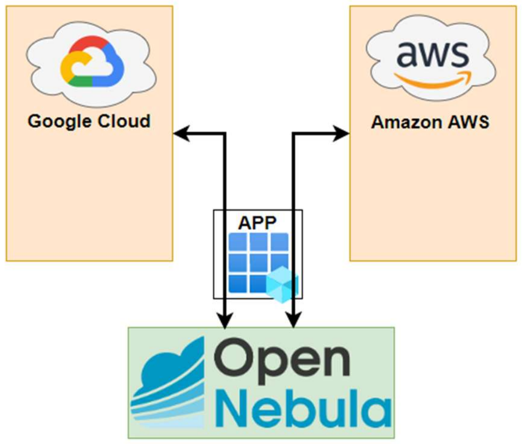


Figure 22 Application migrating and inactive

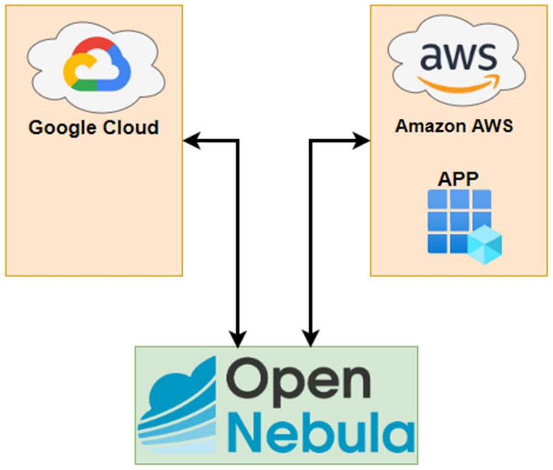


Figure 23 Final phase, application deployed on AWS

The objective of this experiment is to be able to determine, thanks to the data that will be obtained and some metrics that will be explained later, if the live migration process is correct, if it does indeed present some of the challenges that have been raised previously, if it is profitable, how long it takes, etc. In addition, we will try to familiarize ourselves with the platform, as it is one of the most popular platforms offering such a service.

The service that is going to be used to perform this migration can be seen in Figure 22, where we can see that it is made up of a server that obtains the time every second and stores it in a database (GCP) in a text file. The purpose of this information could be to verify the inactivity gap when the migration is performed. Check the time it takes to perform the process.

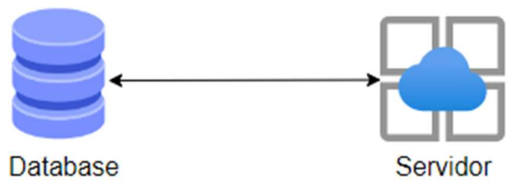


Figure 24 Structure of the service to be migrated

5.2. Technologies used

In this part of the document, we will list the technologies we will use for our experiment. Many of them we have already introduced in the previous subsection, here, we will introduce some more and explain the use we have made of them.

- OpenNebula: for our experiment the free version of OpenNebula (Community Edition via miniONE[4]), has been used, as the Enterprise Edition is subject to an annual subscription fee. A simple deployment script called miniONE is used to set up an OpenNebula Front-end. This tool can serve as a platform for more substantial short-term deployments or project, but its primary purposes are assessment, development, and testing.
- Google Cloud Platform: this is where our service will be hosted at the starting point. Google Cloud Storage is used for the database and Google Compute Engine for the service.
- Amazon Web Services[5]: this is where our service will be hosted at the end of our experiment. Where we will place the final instance of our container.
- Docker: taking into account the advantages offered by Docker[6] and the familiarity we have with this technology; this is the one that has been chosen to create our container.

5.3. Experimental Procedure

To start the experiment, the first thing that must be done is the implementation of our service. To do this we will create a server with express that gets the time every second and using Google Cloud Platform we save this information in a .txt file in Google Cloud Storage.

[4] https://github.com/OpenNebula/minione
[5] https://aws.amazon.com/
[6] https://www.docker.com/

Clearly, before it is possible to include the use of GCP in our service, we must create our account and make the necessary configurations in order to have our project ready to use GCP.

In addition to GCP, we must configure our Amazon Web Services account. This involves creating the project and everything we need to migrate. Once we have the server created, we create the Docker container and upload it to our repository to be able to access it from OpenNebula.

Now it's time to move on to OpenNebula, taking into account that it is an academic project, we have used the Community Edition version because the Enterprise Edition is subject to an annual subscription. For the installation of OpenNebula, we decided to try the two ways offered by the platform. The first option is to use a script (miONE) prepared by the OpenNebula team. This script is ideal for testing and small deployments. The second option we have also used is to follow step by step the instructions offered by OpenNebula. Where we made a complete installation of the Community Edition frontend.

The first thing we are going to do is to add the providers that we will use to the platform, in Figure 23 we can see how the environment is configured.

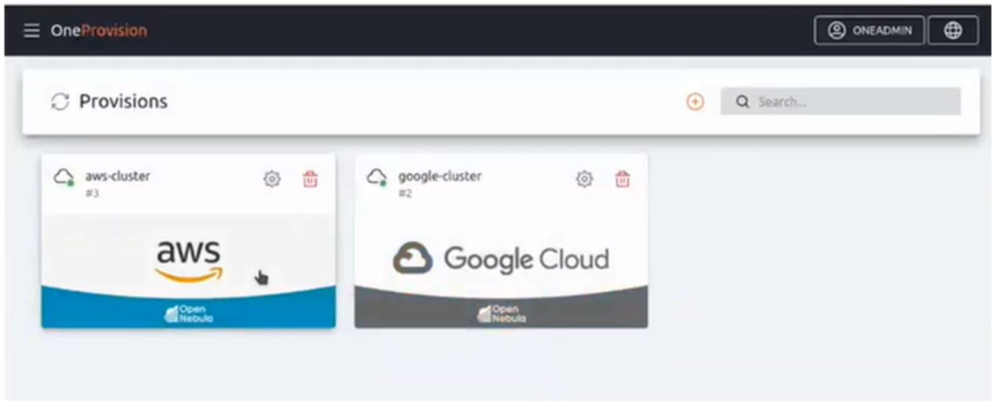
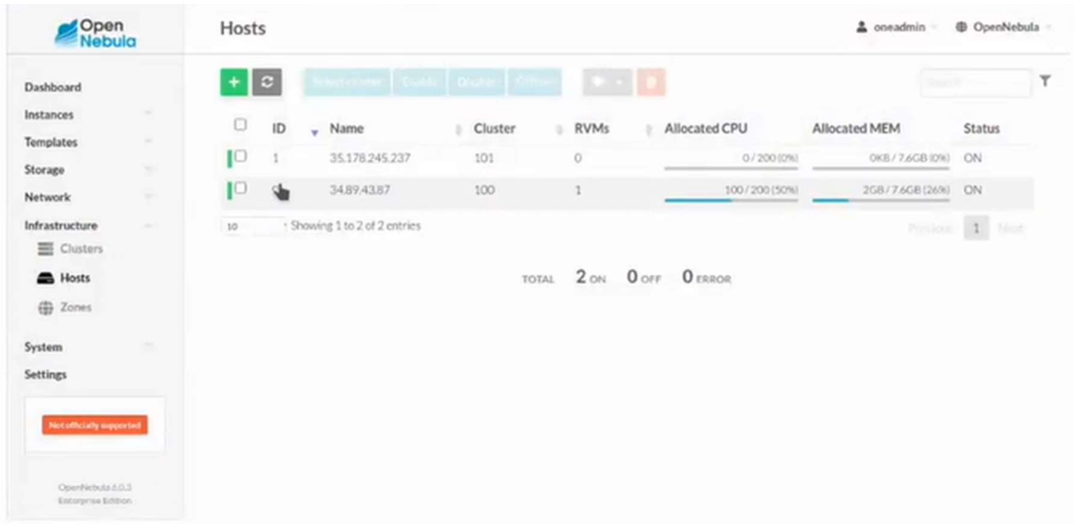


Figure 25 Providers available for use

Next, I will take care of creating the two hosts that will be in charge of storing my containers. In the following Figure (24) it is described how the two available hosts appear. The name of each host is the IP they have access to in each provider.



ID	Name	Cluster	RVMs	Allocated CPU	Allocated MEM	Status
1	35.178.245.237	101	0	0 / 200 (0%)	0KB / 7.6GB (0%)	ON
	34.89.43.87	100	1	100 / 200 (50%)	2GB / 7.6GB (26%)	ON

10 Showing 1 to 2 of 2 entries

TOTAL 2 ON 0 OFF 0 ERROR

Figure 26 Host created

Once the hosts are ready, it is the time to load our Docker image to OpenNebula. After having the image loaded on one of the OpenNebula Datastorage we deploy our app on the host we want.

After the application is running on the GCP host, it is time to move on to the live migration to AWS. When we select the live migration option, OpenNebula starts the process where it will take our container to AWS. During this process OpenNebula takes care of removing the connection to GCP and switching it to AWS, places the container in the "PowerOFF" state while it performs these tasks, and finally, reactivates the container and gets our service back up and running.

To finish with our experiment, we just need to check that the information that is being stored in our database (Google Cloud Storage) is being updated as it should. And that the service is still active after the migration. Then we will check the stored data, which will serve as metrics to evaluate the migration. Since we will be able to observe for example the time that the service was not active during the migration. We will also evaluate the process of the OpenNebula platform, passing the installation process, the usability of the application and the final result of the migration process..

5.4. Results and Problems

Once I have reached this part of the thesis, I will proceed to explain my experience with the live migration of my service using the OpenNebula platform. The first thing I am going to expose are the predicaments I have had during the experiment; I have suffered some problems and of different types. Before starting to talk about the problems, I would like to separate them into two groups, the problems that appeared using the installer script, and the problems that arose using the step-by-step manual installation version.

Installer script: The biggest problem I had with the platform when I installed it with the installer script was that it did not allow me to add my suppliers. The creation process generated an error with Ansible, which I could not solve by my means. Also, in the Community Editionversion of OpenNebula, technical support or questions are handled through a forum. Which is quick to respond in many occasions. But in our case we did not get answers that solved our problem.

Therefore, since I could not perform the experiment with the platform installed through the script, we decided to go to the manual, step by step version found in the documentation.

Step-by-step manual installation: (14) Unlike the previous installation method, this one allowed us to add our providers, add the hosts on both providers, load our Docker image on the platform, but we could not run our application on the host after deploying. OpenNebula was deployed to the GCP host, but when we ran our container, nothing happened. It never got to run our service on the provider. Therefore, after the problem with the automatic scripting option we found that we were also unable to perform the migration process using the manual OpenNebula installation.

In addition to the above problems, my biggest drawback was again that when using the Community Edition version, I had no technical support option. I contacted OpenNebula via email, and they told me that errors or problems with this version are resolved through the application's forum[7]. It should be noted that all the problems I had throughout the installation and configuration were solved satisfactorily through the forum. But these last two, both the installer script and the step by step installation could not be solved by any of the forum users.

[7] https://forum.opennebula.io/

So, at this point, I can say that I were not able to perform the migration process on the chosen platform. Therefore, I have no experimental results to offer, nor will I be able to use the metrics I had prepared for the assessment of the process.

6. Conclusions

Having reached this chapter, it is time to look back and see how my project, my research, developed. I can talk about both the overall objective and the journey I have made throughout the research. The overall objective was partially achieved, but I was not able to achieve all the objectives previously set at the beginning. As mentioned in earlier chapters, the problems encountered when using OpneNebula did not allow me to test the live migration process in a practical way. Many of the problems I encountered regarding the installation or use of the platform were solved thanks to all the documentation offered by OpenNebula and the speed with which the platform's forum community responds. But at a certain point, I had run out of answers, resources, and time perhaps.

Also, I don't know whether to call it a mistake or an unexpected surprise, but what seemed at the beginning that OpenNebula works with containers with a natural approach using Docker with the LXC system, I came to realize during the experiment that the migration is managed with microVMs with Firecracker. Which I know is a valid way because they are still containers, each platform can manage the live migration of containers as they want. But the duality that options with containers that I saw at the beginning of this document, in the end it was only the microVMs option, and that caused a great deal of damage.

On the other hand, I have been able to offer several theoretical solutions to the different challenges that arose at the beginning of the project. This could be done thanks to the research on how the different platforms currently work, what are the market options in other processes and combine them to be able to offer our contribution to continue improving in that aspect. All of them are presented in a theoretical way, without doing any implementation, but I tried to provide as much information and resources (diagrams, data flow, installation steps, components, etc.) as possible in order to understand what we are looking for and what we offer.

Finally, I was also capable of enjoying the journey I went on during the research, because as far as I was concerned, this was a completely new subject, everything related to the multicloud, the challenges it presents when working with it. I have become familiar with the technology, discovered the virtues, the problems that arise, all this thanks to a number of papers published by researchers. Also, the different informative videos offered by many platforms that work with multicloud.

So, if we stop to think about how I have completed my project, I can say that I am satisfied, because despite the little initial knowledge of the subject, the time available to fulfill the goal and the problems encountered throughout the research, a large part of the objectives has been met. I was able to understand the technology, the advantages it offers and the possible challenges it poses, and to propose possible solutions to current problems in the field.

6.1. Relationship Project - Studies

This section will be devoted to explaining how and where all the knowledge acquired during the completion if this project can be put into practice. I will start by pointing out once again that technology was almost new to me. But during my first year in Valencia, I had two courses that helped me, one of them was the one that familiarized us with clustering, node, Docker and kubernetes. The other was the one that introduced me to the topic of microservices, how to store them in the cloud. Although it was about the traditional cloud, I was qualified enough to apply it. Then, during my stay in Budapest, in the first semester, two of the courses were software development, where my part was just the realization of the server and deployment in the cloud and on a server.

6.2. Future Work

To conclude my thesis, I will, of course, present those things that I was not able to do in the way I first wanted to do them and some new points of interest that were discovered in the process.

- The first aspect I would like to work on further is clearly the experimental part. To be able to finalize the live migration process in OpenNebula.
- Of the ideas put forward as solutions, I am personally seduced by the idea of implementing live migration based on privacy certificates. Because I believe that data protection is a fundamental aspect.
- Being able to perform the same migration process on more than one platform, such as Cloudify or Crossplane.
- To carry out the live migration process with a larger project, so I can observe the differences or truly quantify the process.

7. Bibliography

1.**Red Hat.** Understanding cloud computing. [Online] https://www.redhat.com/es/topics/cloud.

2.**Red Hat.** What are cloud services? [Online] https://www.redhat.com/es/topics/cloud-computing/what-are-cloud-services.

3.**Stackpath.** What is fog computing? [Online] https://www.stackpath.com/edge-academy/what-is-fog-computing.

4.**Red Hat.** Edge Computing. [Online] https://www.redhat.com/en/topics/edge-computing.

5.**Cepymenew.** Diferencias cloud fog edge computing. [Online] https://cepymenews.es/diferencias-cloud-fog-edge-computing.

6.**Cloudflare.** What is Multicloud? [Online] https://www.cloudflare.com/es-es/learning/cloud/what-is-multicloud/.

7.**Infoq.** Container live migration. [Online] https://www.infoq.com/articles/container-live-migration/.

8.**Bruno, Rodrigo and Ferreira, Paulo.** ALMA – GC-assisted JVM Live Migration. [Online] https://rodrigo-bruno.github.io/papers/rbruno-middleware16.pdf.

9.**OpenNebula.** Edge Cloud. [Online] https://opennebula.io/edge-cloud/.

10.**OpenNebula**. Multicloud. [Online] https://opennebula.io/multi-cloud/.

11.**Opennebula.** Mastering Containers. [Online] https://opennebula.io/mastering-containers/.

12.**Cloudify.** Migrating Pods Same Kubernetes Cluster. [Online] https://cloudify.co/blog/migrating-pods-containerized-applications-nodes-kubernetes-cluster-using-cloudify/.

13.**Red Hat.** Network considerations. [Online] https://docs.openshift.com/container-platform/4.7/migrating_from_ocp_3_to_4/planning-considerations-3-4.html.

14.**OpenNebula.** Single Front-end Installation. [Online] https://docs.opennebula.io/6.4/installation_and_configuration/frontend_installation/install.html.

15.**Paul, Subharthi , et al.** Application delivery in multi-cloud environments using. [Online] https://reader.elsevier.com/reader/sd/pii/S1389128614000826?token=2680C78F80A2E827DD207AF973AC8936F5B756AE26210095C5C557AAE26794F863D4CC1A6B967707F431374B764BFEFD&originRegion=eu-west-1&originCreation=20220627000317.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

8. Annex

8.1. Sustainable Development goals

Degree to which the work is related to the Sustainable Development Goals (SDGs)

Sustainable Development Goals	High	Medium	Low	Not applicable
SDG 1. **No poverty.**				X
SDG 2. **Zero hunger.**				X
SDG 3. **Good health and well-being.**				X
SDG 4. **Quality education.**				X
SDG 5. **Gender equality.**				X
SDG 6. **Clean water and sanitation.**				X
SDG 7. **Affordable and clean energy.**				X
SDG 8. **Decent work and economic growth.**				X
SDG 9. **Industry, innovation, and infrastructure.**	X			
SDG 10. **Reduced inequalities.**				X
SDG 11. **Sustainable cities and communities.**				X
SDG 12. **Responsible consumption and production.**		X		
SDG 13. **Climate action.**				X
SDG 14. **Life below water.**				X
SDG 15. **Life on land.**				X
SDG 16. **Peace, justice, and strong institutions.**				X
SDG 17. **Partnerships for the goals.**				X

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

ETS Enginyeria Informàtica
Camí de Vera, s/n. 46022. València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es -
www.inf.upv.es

Euro-Inf Bachelor awarded by EQANIE

Reflection on the relationship of the thesis with the SDGs and with the most related SDGs.

I have related several of the SDGs to my thesis. I do anticipate that this project will be relevant to 2 of the suggested overarching objectives. All the project goals will be discussed, along with an explanation of how and why each goal is connected to the others.

The first SDG that my thesis may be related to is SDG 12. Responsible Consumption and Production, with a medium level relationship. This is because my project is based on the study of the migration process of containers between different Clouds, where either the reason for the migration process or just the outcome of the migration can be related to that SDG. When talking about the relationship with the reasons for migration, we refer to the fact that what we are looking for with the migration is to satisfy responsible consumption needs. Moving our container to a provider that offers services more in line with our needs (dedicated CPU, dedicated GPU, etc.). On the other hand, it is also mentioned that migration can result in a closer relationship with this ODS. This may be that, when migrating for whatever reason, scalability, maintenance, error prevention, we find that the new container is hosted on a host that has a much more responsible resource allocation than the previous one.

Lastly, and the one that I consider to be the one that is most closely related to my work is SDG 9. Industry, Innovation, and Infrastructure. Considering the nature of the process being studied in my work, it is clearly related to infrastructures, since it talks about the migration of a container that is in a Cloud to another, which may be in a different region or country than the one it was previously in. It is also worth highlighting the relationship it has as it is an innovative process. Of course, it is a process that has been going on for some time, but it is still expanding and improving due to the number of unknowns or problems it presents as discovered throughout this work.