UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

AGH

On-demand provisioning of educational cloud-based services

BACHELOR'S DEGREE IN COMPUTER ENGINEERING

ERASMUS+

Semester: Spring 2020

AUTHOR:

Raúl Llopis Gandía

UPV Supervisor: Igual García, Jorge **AGH Supervisor:** Zieliński, Sławomir

Kraków, July 2020

Resumen

El objetivo de este proyecto es facilitar la creación de nuevos entornos de capacitación en las aulas de las instituciones educativas, lo que permite a los docentes e investigadores aprovechar los medios tecnológicos al facilitar rápidamente el acceso y la organización de la información. Por esta razón, se presenta la implementación de un orquestador de servicios en la nube basado en el enfoque de la arquitectura dirigida por modelos (MDA), proporcionando así un acceso rápido a recursos y servicios basados en lecciones educativas interactivas.

Los resultados del proyecto se utilizarán en la Nube Educativa de Małopolska (MCHE), el cual es un entorno educativo que conecta aproximadamente 150 escuelas con 10 universidades.

Palabras clave

Computación en la nube; SaaS; integración tecnológica; servicios educativos; acceso a la red bajo demanda.

Abstract

The purpose of this project is to ease the creation of new training environments in the classrooms of educational institutions, allowing teachers and researchers to take advantage of technological means by promptly facilitating access and organization of information. For this reason, the implementation of an orchestrator of cloud services based on the Model Driven Architecture (MDA) approach is presented, thus providing quick access to resources and services which are based on interactive educational lessons.

The results of the project will be used in the Małopolska Educational Cloud (MCHE), which is an educational environment connecting approximately 150 schools with 10 universities.

Keywords

Cloud Computing; SaaS; technology integration; educational services; on-demand network access.

Table Of Contents

Chapter 1. Introduction

Over the past decades, exponential growth in the use of Information and Communication Technologies (ICT) has been originated due to the need for addressing solutions to the problems that arise in areas such as *education, medicine or science* (Acemoglu, Daron, Restrepo, & Pascual, 2018). Therefore, this growth in demand leads to a necessary evolution of traditional technologies, leading them to a new understanding of efficient and reliable architectures that can be adapted to all circumstances.

The project presented in this thesis is focused on applying these new technological advances against a typical problem in the educational system, the huge educational gap that exists between secondary schools and universities (Czekierda, Zielinski, & Szreter, 2017). This problem is preceded by the fact that before entering high school, students do not have enough information about the specific field study program and even less about the base content they will face during their studies. In many cases, students realize quite late that their career choice was not the right one because it did not meet their expectations due to this lack of information.

For this reason, it is necessary to establish a bond of union between secondary schools and universities so that, through new teaching methods based on digital educational collaboration through ICT technologies, the knowledge of higher education is disseminated in an easily accessible and dynamic way to younger students. Thus, students will be more encouraged to make decisions based on facts like the content learned with this learning methodology and, at the same time, on their vocational preferences.

My contribution is to implement a cloud service orchestration process to offer high school teachers on-demand interactive and cooperative learning platforms based on educational lessons that include a variety of generic content from the field of study in question. At the hand of this solution, secondary school teachers can request services in an effortless and intuitive way through a web form in which they simply specify the requirements and characteristics they need in a declarative way.

The most significant advantage this solution attempts to provide to institutions is the immediate availability of services from the moment the service is requested until the service is ready. That functionality is provided by building these services on virtualization platforms which are advantageous to achieve this prompt availability sought.

The results of the project will be used in the Małopolska Educational Cloud (MCHE)[1], which is an educational environment intended for high school students to take advantage of the potential of higher education institutions in the Małopolska Region (Lesser Poland), through the implementation of innovative joint research projects and the conduct of virtual classes and laboratory classes based on infrastructure and modern technology (Zieliński, Czekierda, Malawski, Straś, & Zieliński, 2017).

The interest in the subject of this project is given by the personal satisfaction of knowing that secondary school students will be brought closer to what the University will be like, as well as showing them first-hand educational content on the degree they wish to pursue. This project is focused principally on IT-based lessons, although it is designed to cover a wide assortment of disciplines of knowledge, therefore in an accessible and interactive way, each student will be able to learn the basics and practical applications of programming languages, networks, security, neural networks, etc., in their respective schools, not having to visit the universities.

By the accomplishment of this project, it is intended that future students of secondary schools of Małopolska Region have at their disposal, when needed, an interactive platform that is tailored to common interests. It is intended to help them improve transversal skills such as association in a group or lifelong learning through the acquisition of basic concepts that will be key when making the decision to choose a bachelor's degree, in addition to having a broader conception of what university degrees look like.

From a methodological perspective, the aim is to achieve the previous results by carrying out the design and implementation of the service orchestration system, in which each of the main process workflows that are automatically performed when requested will be consequently orchestrated with the slightest human interaction. At the moment after the request is produced, it results in the specific platform provided and quickly accessible to users through online web access. This orchestration process is

[1] Official website of the MCHE project https://e-chmura.malopolska.pl/

designed including the principal concepts of scalability and modularity, in order to reduce the complexity of the system, making it easy to incorporate several specific platforms as modules independent of the central process.

The remaining chapters of the presented project are structured as follows:

- Chapter 2 presents fundamental definitions and terminology correlated to the subject of the project.
- Chapter 3 analyzes functional requirements and behavioural aspects related to the conceptual modelling of the solution.
- Chapter 4 presents in detail the aspects regarding the design and architecture of the ad-hoc Model Orchestration System, and describes the technologies chosen to carry out the implementation of the solution.
- Chapter 5 describes in detail the structure and logical operation of the implementation.
- Chapter 6 compiles and explains the different evaluation experiments carried out with the implementation along with an analysis of the results.
- Chapter 7 contains the conclusions derived from this project and possible future improvements that can be made to the current implementation.
- Eventually, the final chapter exposes the literature derived as a result of the contributions of this project.

Chapter 2. State-of-the-art

In the past, for data infrastructures, the most common practice was to acquire physical computer systems, mainly servers, which entail a large economic investment, both in acquisition and maintenance. The scalability of the system was based on adding hardware vertically, so the process often involved downtime for production. Furthermore, the electricity consumption of those servers could lead to unwanted carbon emissions at a large scale.

The evolution of the computing power and the evolution of infrastructures have converged with the generalization of broadband Internet access, which led to a notable change in the paradigms of infrastructure architecture and design as low-cost commodity components. Those factors are changing the way we do computing, resulting in a new working model called Cloud Computing. In this section, the state of the art implementations of Cloud Computing along with the key technologies currently used are presented.

2.1. Cloud Computing Paradigms

In general terms, Cloud Computing is a computing paradigm that consists of offering a shared pool of computing resources (E.g., storage and CPU) of a conventional computer system as a service available on user's demand. This concept is not at all new, in fact, it has been forged over the years since in 1960, John McCarthy named the use of computing facilities as a utility. In addition, most of the technologies used in cloud computing today, such as virtualization, are not recent technologies but rather are used appropriately to meet the high demand in Information and Communication Technology (Zhang, Cheng, & Boutaba 2010).

A very simple way of thinking about cloud computing is to consider the case in which a task that requires a huge amount of computational data to be processed, such as genome analysis, which requires to be performed and searched at the same time. A simple way of conceiving cloud computing is to consider the case of a task that requires a considerable amount of computational data to be processed, such as genome analysis.

A traditional solution would be to acquire a supercomputer for this specific task, which is extremely expensive to maintain over a long term in addition to the initial cost.

Instead, an optimal and energetically sustainable solution would be to benefit from several cloud service providers in order to request the essential resources on-demand, thus paying an infinitesimal amount of the estimated costs with the traditional solution. In addition, this solution provides an indispensable benefit for research such as the immediate availability of the service, which is pursued as the main objective of this project.

The main characteristics cloud computing offers are the following defined by NIST: National Institute of Standards and Technology (Mell & Grance, 2011):

- ***On-demand self-service*:** End users can decide for themselves what services to provide and how much to invest, based on a 'pay-as-you-go' payment model, in which the user is billed only for the amount of resources used or subscribed. This characteristic provides a very important benefit for users since it facilitates access to services through a control panel in the browser, thus reducing the learning curve as the user does not need to learn how to install the applications.
- ***Ubiquitous network access*:** Cloud services depend inherently on distributed communication infrastructure and as such provide pervasive availability of services as long as there is a connection to maintain continuous connectivity. In this way, users can make use of the services wherever they go, regardless of the geographical location, in which the resources are housed.
- ***Resource pooling*:** This concept combines the computational power of modern scalable systems, such as physical and virtual servers, that are involved in cloud computing to create a sense of unlimited resources. Thus, as discussed before, users should only keep in mind the resources they need, abstracting from the provider implementation.
- *Rapid elasticity*: This concept implies dynamic provisioning of services, self-service on-demand and ability to increase the amount of resources used in case of an anomaly in the load of user requests in the service. Additionally, tenants can make almost instantaneous modifications without contacting a service provider.
- *Measured service*: The idea behind this concept is the automated optimization of metrics on cloud systems, in order to control the use of resources by the user, thus mitigating the need to forecast and plan resources, and reduce the waste of expenses generally.

The principal focus of this bachelor's project is that the designed system complies with the first three characteristics, although the others are not the primary objective, however, they are also taken into account during development and future work.

Cloud computing offers its benefits through three main types of service models:

- *Infrastructure as a Service (IaaS):* It is mainly based on offering on-demand all the physical components of a system in a virtualized way, such as servers, network connections, bandwidth, addressing and load balancers. All of this is accomplished by providing some of the processing power of physical servers to users in a friendly way. The main advantage of this type of services is the scalability, it allows the possibility of paying based on consumption, speed to access the infrastructure in a matter of minutes, disaster recovery infrastructure thus reducing costs and increasing manageability.
- *Platform as a Service (PaaS):* It is based on offering the customer an environment to develop and run their applications. The PaaS provider offers access to a machine already configured with an operating system, libraries, etc., with all the dependencies satisfied so that the client does not have to manage the abstract part of the system, since that task is performed by the provider.
- ***Software as a Service (SaaS):*** As the name suggests, this type of service is based on offering applications as a service, so that the end-user only has access to the application, usually from a web interface on the client-side. The provider is in charge of offering the ready-to-use service to the end-user.

  Some of the advantages offered by these services are that it eliminates the need to install and run applications on individual computers, reduces infrastructure and maintenance costs, is scalable according to the needs of the provider, the software is always up-to-date and ready to use from anywhere.

  On the other hand, this model has some limitations such as the security of the data that must be exchanged between the backend servers of the services in the public clouds, so it could happen that the data could be compromised.

In this project, the SaaS model is used to provide services on-demand with minimal user interaction using a web interface which will be the main focus of user interaction. The benefits of this model to alleviate the inconveniences of traditional learning are positive for all parties; Teachers have control over the type of service that is demanded

to meet the needs of students, in addition to achieving learning results in a manageable and practical approach by gathering data through the service.

Furthermore, Cloud Computing offers its service through four implementation models:

- *Public cloud:* In such implementation, the cloud infrastructure is accessible to the general public through the Internet and is shared in a pay-as-you-go payment model. In this model, the provider is responsible for managing the shared infrastructure, thus providing a high availability ratio for which customers will consider when choosing the service. From a tenant's perspective, the main advantages of this implementation are the low infrastructure costs due to the fact that you only pay for the resources that you use, in conjunction with a vast distributed network of servers that guarantees tolerance against failures and on-demand resources are available to scale when needed. Instead, absence of privacy is suffered because servers are public and therefore visible throughout the Internet, thus leading to greater opportunities for attackers to perform remote scanning for vulnerabilities.
- **Private cloud:** In this model, cloud resources are usually located within the client's organization (on-premises) or hosted by a third-party service provider (off-premises), but they can never be accessed by the general public, but only by internal users, and it should be noted that in this model the organization will be incapable to use the public cloud. Services and infrastructure are perpetually kept in a private network that can be accommodated to the needs of the consumer to accomplish the particular conditions coveted. This implementation criteria adjusted to the project objectives is discussed further below.
- *Community cloud:* In this model, the cloud infrastructure is shared by several organizations with common concerns, such as security and compliance considerations. This type of cloud can be managed internally or by a third party and can be hosted on-premises or off-premises. Besides, all community members have access to cloud services.
- *Hybrid cloud:* This model implements the combination of different clouds, typically private and public clouds, which is beneficial during an anomaly in the demand concerning services, so these events can be thoroughly automated for scaling resources and for balancing the workload of resources between various environments while maintaining absolute control over the data.

The principal motivation for choosing the private cloud as the implementation method is given by the fact that the present project is developed and implemented entirely upon the educational cloud MCHE which is a private cloud created by the Małopolska Voivodeship in cooperation with leading universities from the Małopolska Region and secondary schools. However, given the fact that teachers are the ensembles who demand the required services, it is crucial that each application does not result in additional expenses for schools, in such a manner that the usage of the private educational cloud is the most reliable approach to deploy the services.

Aside from the aforementioned purpose, there are other reasons based on the premises that support it:

- **Adaptability:** Thanks to virtualization, multiple applications can be nested to operate on the same physical machine or server to achieve a great performance sharing physical resources. This makes it tend to offer great versatility if the requirements of an application change, such as increasing the amount of RAM for greater input of users without application downtime, thus ensuring application performance at the same time it reduces costs by getting the most out of the physical servers.
- **Enhanced protection:** It provides several security benefits over a public cloud. First of all, given the fact that it is accessed through private and secure network links like VPN, rather than the public internet, it provides enhanced confidence regarding where the data is passing through. Aforementioned peculiarity is accomplished by the fact that the private cloud is primarily hosted on proprietary servers that the organization owns them has exclusive access to and use of, which makes the physical security more straightforward to ensure. In contrast, in the public cloud, reliable implementation of a network firewall or a web application firewall (WAF) would be compelled to prevent undesired access.
- **Eminent scalability and resource availability:** Through the redundancy of physical and virtual components such as the storage area network (SAN), the power supplies or virtual networks in the servers, enables the accomplishment of service activities without causing downtime, and can further significantly reduce the recovery time before failures (RTO). Those capabilities allow the stipulation of service level agreements (SLAs) in private clouds with guarantees higher than 99 percent of availability.

2.2. Cloud Adoption in Education Sector

Education is increasingly embracing advanced technology because students are already doing so. Hence, in an effort to modernize classrooms, Educators have introduced new e-learning environments in order to innovate the way in which it is taught today. These environments are linked with the rapid evolution of new technologies, adapting them to specific case studies such as virtual classes or collaborative lessons in real-time, in order to create a close link between the teacher and the students, as well as awakening in the students the interest in lifelong learning.

As stated in (Ercan, 2010), the educational sector must benefit from the consumption of the most advanced technological services that are tailored to the particular learning needs to improve the technological literacy of students with the contributions from the Universities to detect emerging technologies in addition to providing equality in access to technologies.

The technology that may have significant importance in the educational revolution is cloud computing. This is because it provides beneficial innovations for education, such as on-demand self-service that promotes to reduce waiting times for providing services. Furthermore, this technology induces a significant reduction in the economic cost of infrastructure and maintenance for institutions, making it a key technology for education.

The adoption of this technology enriches education in general, but the reality is that not everyone has the same privileges or easy access to technology *(Masud & Huang, 2012)*. Above all, the main focus of the problem is mainly in developing countries where the imbalance between areas hardest hit by the lack of resources makes it impossible to implement e-learning tools. Moreover, the absence of expertise and technological knowledge on the role of the educators in the most disadvantaged educational centres that make adoption further impracticable.

Given this problem, cooperation between universities and institutes is necessary to mutually benefit from the use of cloud computing. This benefit derived from the cooperation allows both to overcome the maintenance costs of the IT infrastructure in the long term (Katiyar, Nishant & Bhujade, 2018) thanks to the Virtualization concept that is linked to the term Cloud Computing, resulting in immense savings in energy and hardware costs.

By integrating resources through cloud computing, it is possible to meet the high demand for institutions thus ensuring the development of digital education through the trend of Software as a Service. For this trend, according to Figure 1, it can be predicted that by 2023 institutions will have invested more capital in SaaS, thus replacing progressively on-premises educational applications.

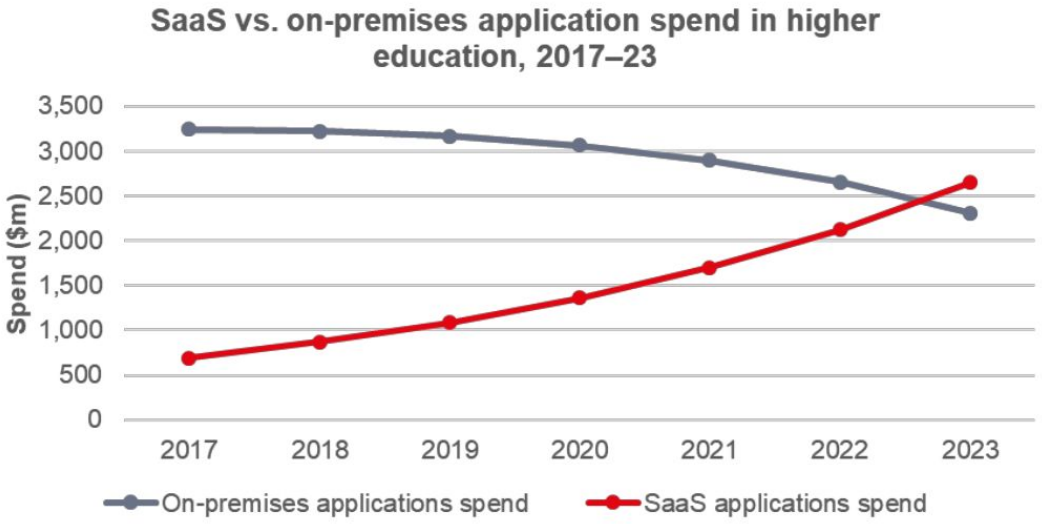


Figure 1. Prediction of SaaS adoption for high schools for the upcoming years[2]

This prediction becomes stronger with the statistics shown in Figure 2, which demonstrates the growth that Cloud Computing has experienced to date. It can be seen that 81% of the respondents have at least one application in the cloud, which represents a growth of 24% compared to 2012, according to IDG.

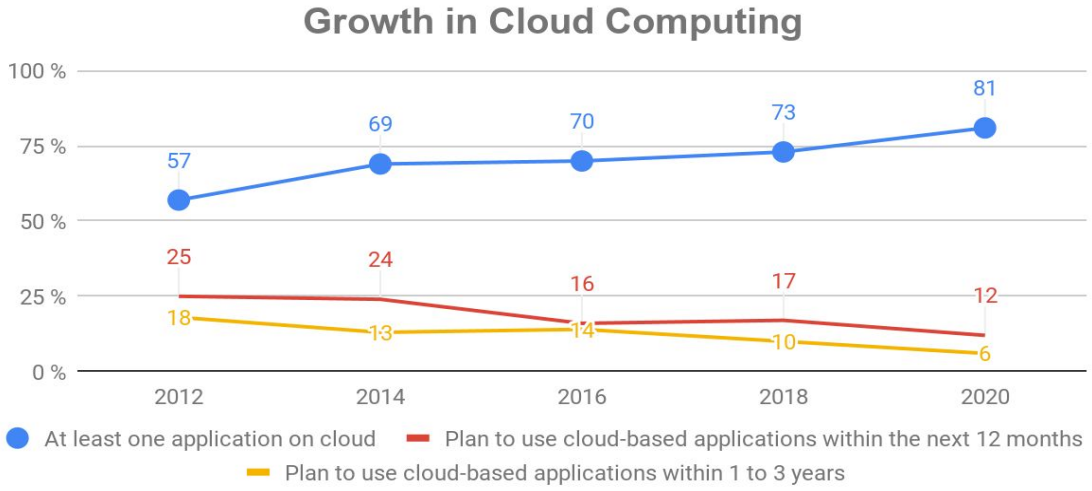


Figure 2. Survey on the growth of Cloud Computing till 2020 by IDG.[3]

[2] Source: https://d1.awsstatic.com/analyst-reports/WP%20AWS%20Reaching%20for%20the%20Cloud.pdf

[3] Source: https://www.idg.com/tools-for-marketers/2020-cloud-computing-study/

2.2.1. Impact of COVID-19 in Education

All these efforts to bring the technology directly into classrooms have become more important in recent times during which the present project is developed due to the global pandemic caused by the infectious respiratory disease COVID-19, which has paralyzed most sectors due to the need for social distancing between people by implementing lockdowns and community quarantines to abate the growth of global infections.

During this unwelcome and unprecedented situation, the vast majority of non-essential sectors have been forced to implement new digital methodologies to maintain the optimal functioning of daily work. For many of them it has not been so easy to adapt to the change for what has turned out to be a hasty action that has gone accompanied by uncertainty and lack of previous preparation time.

Suddenly, cloud computing has become the backbone of global virtual collaboration and learning experiments where millions of people use online services during confinement on a scale never experienced before in such a way that it has truly become a real test to the infrastructure of today's Internet. As can be seen in Figure 3, the use of online services from the months before confinement is compared to what is measured after governments lockdowns are applied in Europe.

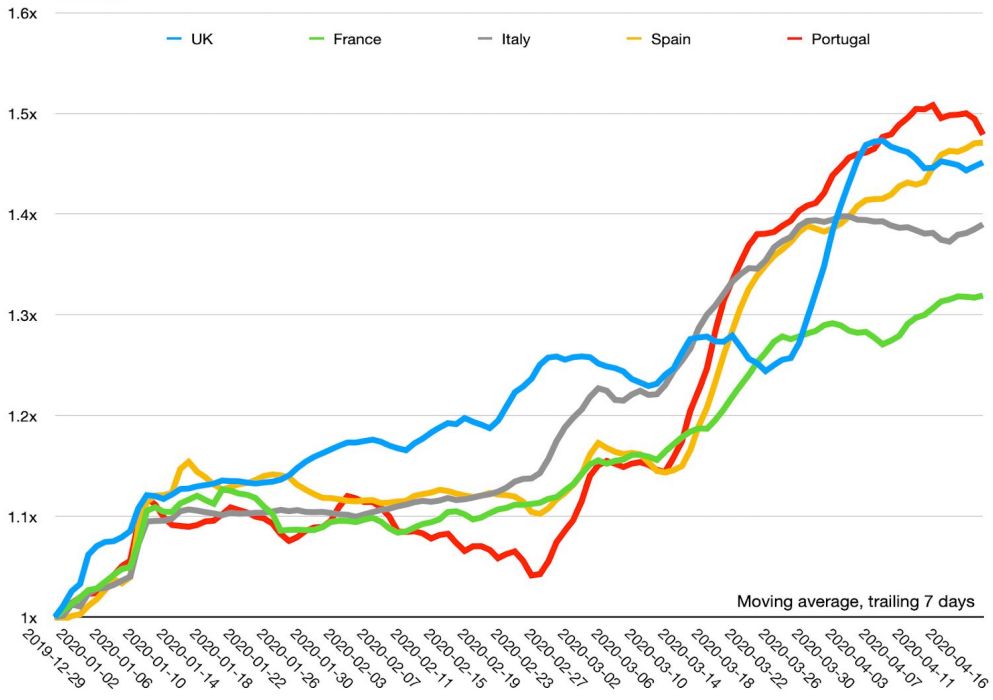


Figure 3. Variation of Internet traffic in Europe seen by Cloudflare[4]

[4] Source: https://blog.cloudflare.com/recent-trends-in-internet-traffic/

However, it is clear that everyone is beginning to realize the benefits and value of cloud computing, and that is why education has not wanted to be left behind in adaptation. The cloud computing model that is prevailing in the education sector and that is now gaining strength is Software as a Service, which can be seen in different ways, such as digital learning management systems or communication tools.

In addition to rapid adaptation to change, the biggest and increasingly evident hurdle in this situation are the equality gaps in the multiple layers of access to the physical and virtual infrastructure required to access cloud services, particularly for students and low-income families and poorly connected regions suffering from a form of digital inequality where they lack the connections and devices to learn remotely. In this situation, it happens that digital access is not the same for everyone and inequality increases, despite the fact that educational centres strive to keep them included.

As a result, it is likely that many sectors and especially education will begin to strive to invest in the digital transformation through cloud resources in the coming years.

2.3. Model-Driven Architecture

The Model Driven Architecture (MDA) is an Object Management Group (OMG) standard, which establishes the idea of separating the specification of the system's operation from the specific details of the platform by adopting models as the principal line to design and understand a system (Amar, I. and Hany, 2015). It reveals a detachment of affairs by dividing business functionality from specific implementation technology.

The fundamental basis of the MDA concept is the fact that the programmer who develops the software applications should not focus on the functionality of the source code but rather on separate models and their interaction (Deeba, F., 2018). In the concept of model, it is where the entire construction of the application is systematically defined without having any dependence on a specific language, therefore these models are described according to a standard such as Unified Modeling Language (UML).

This mode of operation of MDA using models provides important features such as the portability of applications to any platform, whether written in the Java or C programming language, because the system is independently specified from the

platform that supports it, therefore it promotes the reusability of the system being able to be applied to brand-new specific platforms.

The concept of separating and abstracting parts of a system is a term that is not presented as a novelty with MDA, but it dates back to the year 1976 when it was credited to Edsger W. Dijkstra.

"[...] study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. [...]" (Dijkstra, E. W., 1976)

Through this citation, Dijkstra notes his support for the need to analyse problems independently in order to preserve them consistently.

The Model Driven Architecture (MDA) standard describes three abstraction models with distinct functionalities (Miller, J. and Mukerji, J., 2003), these are the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM).

- **Computation Independent Model (CIM)**

  First, The Computation Independent model is a representation of a system from the independent point of view of computation, that is, the model focuses on describing the external functionality of the environment with which it will operate, providing the platform external aspects thus abstracting the details of the internal structure of the system.

  In this way, the primary user of CIM does not have knowledge about the technicalities of the system or the platform where the application is developed, the user will simply interact with the CIM. It is assumed that this user does not have advanced knowledge on the subject and therefore should solely focus on the aspects that are especially well-known for its global operation.

  In this project, as discussed hereinafter, this model is replaced by the Teacher Model which follows the same philosophy as CIM but is focused on the specific system developed for the project purpose.

- **Platform Independent Model (PIM)**

  The Platform Independent model is primarily focused on defining the logical operation of the system at a low level, representing the generic values of a system which are applicable to any system independent of the platform where

the code is generated, furthermore, a degree of undependability is obtained by not altering its functionality from platform to platform, thus hiding the specific details of the platform.

In order for this model to be applicable, each platform must be of a similar type, that is, they must share the same objective and functionality pursued as in the global system.

- **Platform Specific Model (PSM)**

  Subsequently, the Platform Specific model is the outcome of the transformation of a PIM. This model has similar functionality as the one specified in PIM, reflected in the specific platform representing the same concepts, but with the previous transformation of PIM methods to platform-specific values.

  The significant concern about these two models is that both specify identical functional concepts but these are extended from PIM to PSM. By means of the transformation carried out, details necessary for the platform will normally be incorporated in PSM.

In order for a MDA specification to be thoroughly comprehensive, it must consist exclusively of a Platform Independent Model (PIM), one or more Platform Specific Models (PSM), a transformation and interface models, which will determine how to implement the model on a heterogeneous platform (Sharma, Ritu, and Manu, 2011).

The turning point in this model is in the idea of performing the transformation model to model and model to source code automatically, an objective which is pursued along with the final implementation of this project (Sharma, Ritu, and Manu, 2011). These transformations occur at different levels, in which a transformation from the CIM to PIM model occurs first, the general data specified by the user is converted to technical concepts related to the underlying system functionality.

Next, the subsequent transformation process occurs between PIM and PSM in which detail is added to the independent specification based on the information required by the specific platform resulting in the PSM model, finally, the last transformation happens with the conversion of the data from the specific platform to the source code according to the specified programming language.

2.4. Related work

This chapter surveys the achievements of remarkable researchers on cloud-based educational methods, presenting several approaches in which embrace related solutions to the dilemma manifested in this project.

Glancing at the currently existing papers, remarkable similarities can be found with this solution. (Pena, J., 2006) proposes an approach quite comparable to the one carried out in this project, in which, through the Model Driven Architecture (MDA) philosophy, they approach an autonomous system based on policy management to carry out transformations until the code is generated. The research has a high degree of similarity because previously stipulated policies are used for each model transformation, however, the approach is oriented to space projects in which the policies specify autonomous behaviours for a spacecraft in order to avoid collisions, reason why in this aspect it differs from the approach made in this project.

(Sabiri, K., 2015) and (Zhang, X., 2012) provide different approaches to the MDA model by combining different models. First, Sabiri uses the ADM specification, which is based on MDA, but it is that an inverse engineering model is first applied to the process, so the source code can be transferred from a legacy on-premises application to a cloud application with newly added functionality to the source code. In the educational context, this concept could improve the system proposed in this project by adding the functionality of being able to convert existing monolithic applications to SaaS-based applications.

Secondly, Zhang takes advantage of the MDA philosophy but combines it with the Service Oriented Architecture (SOA) to perform model transformations based on the Role & Goal-Process-Service (RGPS) framework, that is, role-based rules are applied in order to specify the requirements of business logic. In this work, similarities are found in the way of transforming the models, because they are governed by policies.

(Mohssen MA, 2011) proposes the use of the Complete Cloud Computing Formations (C3F) based on the Cloud Cube Model (CCM) to emphasize the concept of service by specifying a quadrant to determine Education and Learning as a Service (ELaaS) using different criteria. This project introduces a conceptual framework that specifies the requirements that an institution needs to be able to move its IT resources to the cloud.

Focusing on the educational domain, the research of (Gaur, A., and Manoj M., 2014) and (Zhu, Z., and Xiao Fei, 2009) promote the objective of providing educational institutions with public cloud based tools to evolve the traditional way of learning by bringing new cloud-based technologies closer together to facilitate learning for students. In this way, these approaches are in the hands of the companies that provide the infrastructure resources, which may not be an adequate solution for educational institutions where privacy remains an important factor.

Eventually, in my personal opinion, concerning the works that have been associated in this chapter, each of them follows methodologies that differ from the one used in this project, except Pena, Sabiri or Zhang whose works support the MDA methodology with a slight variation in design although always being model-oriented with their respective transformations applying policies. However, it should be noted that each work offers a characteristic that is pursued as an objective in this project, which has been of noble inspiration to develop the present final degree project.

Chapter 3. Problem analysis

This section aims to analyze in-depth the inherent challenges of the problem that this project faces, outlining the primary requirements that the solution needs to implement in order to achieve uniform and positive results.

3.1. Requirements analysis

As described in the previous chapters, the problem faced by this project is to minimize the waiting time from when a service is requested until it is available, depending on the requirements that will be described in the following analysis thus the overall process is consistent and meets the stipulated requirements.

3.1.1. Use case

First, the actors will be identified for the analysis of use cases, these are each type of external entities with which the developed system should interact accordingly. Four main actors are involved: the teacher, the students, the leading teacher and the administrator.

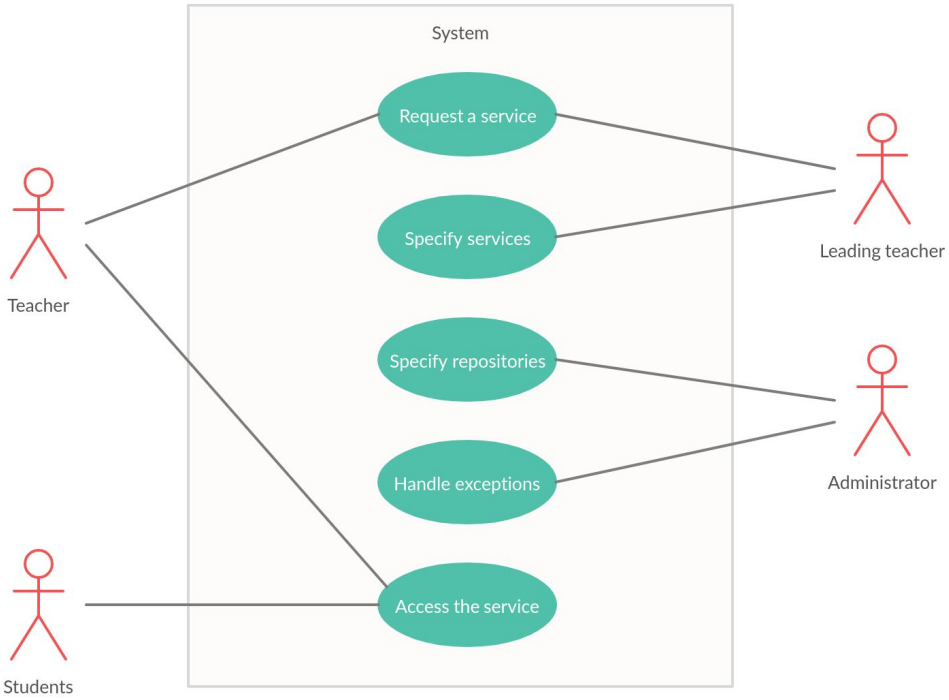


Figure 4. System use cases - Actors

As represented in Figure 4, It can be observed that there are actors that have a direct relationship between them. Following, the actor roles are described in further detail:

1. **Teacher**

   The role of the teacher's actor is essential. The secondary school teacher is in charge of specifying the CIM of an application, so that the system orchestrates an appropriate application satisfying the request. Considering the role of the teacher, he does not have to know technical details. Eventually, when the system is ready and receives the notification with the access instructions and data, the teacher will be in charge of informing the students.

2. **Leading teacher**

   There is another significant actor that has a similar role but with more distinguished privileges than the previous one, this is the leading professor at the University. They can request the service and even support the high school teacher in the same process. In addition, it will be in charge of generating the content that the secondary students will consume. This content is based on interactive lessons that will be adapted to the field in question.

3. **Administrator**

   Another essential actor who has a more secondary role than the rest but not less important is the system administrator. He is in charge of all the technical parts of which the system in question is composed. This actor will be in charge of adapting the repositories of policies, templates and inventories to changes according to the needs specified by the leading professor in the services offered.

   Further, it will be in charge of maintaining the system environment where the implementation is hosted, taking into account high availability and load balancing aspects according to the requirements of the project to which it is adapted. In the situation that some type of exception could occur in the system, it will be in charge of trying to resolve it to resume the process.

4. **Students**

   In general, students have the minimum interaction necessary with the system, this is in order to access the service and make use of it. By the advantage of Software as a Service technology, this actor will only need a digital device with a connection to the service network provided by the tutor and aforementioned could be accessed directly through any current web browser.

3.1.2. Security and data protection analysis

Any implementation approach to potential solutions to this problem is inherently concerning managing and storing personal information about users. That is why given the nature of the system it is essential to analyze how data must be protected and how secure is against possible data leaks.

First, any system implemented in a private cloud has a greater degree of security than the different potential cloud computing implementation models, as long as the private cloud does not have dependencies on third parties for infrastructure.

Despite being designed to reside in the private cloud, data is managed by the corresponding organization, hence a security strategy with multiple layers of defence such as the concept of Defence in Depth should be applied in order to allow the organization to have time to detect and act against cyber attacks to mitigate the possible damage that may be caused to the data *(Vacca, 2017)*.

These defence layers must be distributed around the entire perimeter of the cloud system used to implement this project so as not to leave any fissures in the SaaS environment, some of these layers that can be applied are the following:

- Firewalls (software)
- Intrusion Detection Systems (IDSs)
- Intrusion Prevention Systems (IPSs)
- Virtual Private Networks (VPN)
- Auditing logs
- Restrict access to application services

These previously mentioned layers are of vital importance to keep the integrity of user's data according to the General Data Protection Regulation (EU) (GDPR)[5] regulation thus avoiding the risk of information disclosure by attackers. The physical perimeter layers are not taken into account because this factor does not concern the system that is presented in this project since it is intended to be distributed among virtual machines when the bastion of contact with the outside must be the firewall software.

To append more enhanced security to the network infrastructure, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) must be taken into account to discern incoming packets so that, by consulting a database of Cyber threats, make decisions about which packets are allowed and which should be quarantined or discarded.

[5] GDPR Source https://eur-lex.europa.eu/eli/reg/2016/679/oj

Due to the nature of the problem, it requires applying security techniques to the virtualization of the virtual machines (or containers) that are employed, according to *(N. Zhang, et. al., 2013),* different methods must be applied to guarantee security:

- **Access control:** This method concerns limiting access to virtual environments to authorized users, that is, limiting the number of people with elevated privileges for each environment, leaving only one administrator with full access and the rest with access only to their directories or shared directories between groups, implemented at the level of application.
- **Virtual Machine Monitor or Hypervisor:** Within virtualization, the most important layer to protect is that of the Hypervisor or VMM in which the different guest virtual machines that make up the system where the system is implemented are controlled. This has a high risk of confidentiality, integrity, and availability (CIA) so it is a critical aspect to control because attacks such as "hyper-jacking[6]" malware can be caused once it penetrates a VM that is on the Hypervisor, this can attack the hypervisor or other hypervisors giving complete control to the malware on all the VMs that it implements.
- **Virtual Firewall**: This layer of security is attached to each VM individually to enforce the security of the VMs together with the previously discussed firewall, providing those firewalls with the quality to function as a packet strainer. This layer can even be applied in a virtual network switch that is located at the entrance to the network subnet established for the system.

Furthermore, concerning access to applications, it is vitally critical to control access to services by auditing the different logs and to analyze this information to detect possible attack attempts at the application level in the TCP/IP stack *(Durairaj and Manimaran, 2015)*. Since the SaaS applications that can be integrated within this system may be from third parties, application procedures must be adopted to secure network traffic using SSL/TLS encryption since all the applications approached in this system are designed to be accessed through a web browser.

On the other hand, regarding the security of the data saved in the system, whenever it involves saving user passwords in the applications, a hash method must be applied, such as MD5, to store the passwords in the database. In addition, each of the different services must be isolated from the rest at the network and data level so that only the corresponding users have access and cannot access from other services,

[6] Hyper-jacking malware concept: https://en.wikipedia.org/wiki/Hyperjacking

this is achieved with virtualization which is the main solution proposed for implementation.

Finally, the services that are implemented in the system are based on interactive lessons, therefore must contemplate the collection of the results of the users, which can be saved in the same environment where the solution is implemented or may be required to be transferred to an external system controlled by another institution. In the last case, the transmission of data must be fully encrypted end to end using secure protocols such as SSH, SCP or SFTP or business solutions that use similar methods.

These factors are of vital concern to implement a production solution where thousands of students and teachers access the services provided, in order to maintain their privacy and avoid the risks of data breaches. However, since the objective of this project is not focused solely on security, not all the requirements are pursued with the further implementation of this project, although they are always considered during the development and as future work.

3.2. Proposed solution

3.2.1. Conceptual modelling

Applying MDA to the presented system resulted in defining three main models.

- **Teacher Model** (requirements): Model generated from objective data entered by a teacher concerning service characteristics.
- **Platform Independent Model** (generalization): Generic abstraction in which the technical requirements of the environment system are specified, which does not change from one platform to another.
- **Platform Specific Model** (specialization): architecture design solution and generation of the target application.

For a better understanding of the distinction between these methods, a diagram of activities corresponding to the dynamic modelling behaviour of the selected architecture is depicted in Figure 5. It illustrates the most important behaviours for system operation. These are divided into the three stages mentioned above, whose function depends mainly on the data generated by the previous model.

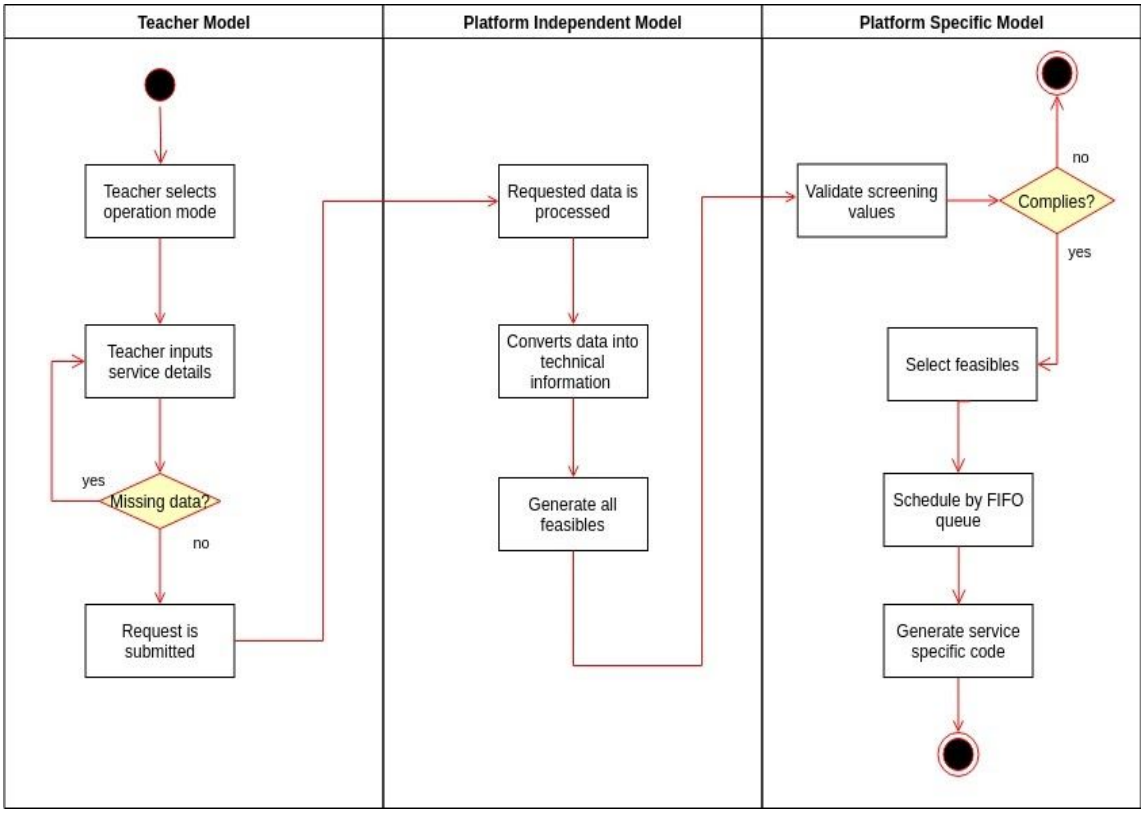


Figure 5. MDA System solution activity diagram

The process begins with the teacher model specified by a teacher actor, who must select what mode of operation it wants to perform, these can be to create, modify or destroy a service. To describe the behaviour of the process it is necessary to focus on the creation of the service, in the next chapter the complete design will be described in detail.

Next, the teacher actor must enter all the details and characteristics of the service that is being requested. Once the data set is complete and validated, the request will be processed by the system, generating it in a data serialization format for more trustworthy handling.

Subsequently, the data is into the form of an Independent Platform Model. The conversion process will be carried out by external templates to obtain better data comparability. This information about the conversion system is abstract to all kinds of platform implementations that can be carried out. Once converted, the process will select all kinds of specific platforms that provide the requested services.

Eventually, the data produced goes to the specific platform model whose method makes the most vital decision for the destination of the process. Through screening policies, all the data entered is evaluated to verify if it complies with those specified by the administrator actor. In the event that there is any data that does not comply with the

policies, the process will end and the teacher actor will be properly notified that it has been breached. If not, the validation will be successful and will be passed on to the next decision policy on which specific platform of all possible feasible is the most appropriate following the stipulated selection policies.

Once selected, this service request will be scheduled in a priority queue. The other independent processes that may exist will be added to this queue, being executed accordingly specified in the scheduling policy by the administrator actor. To end the process, when the queuing process is executed, it will generate the corresponding code of the specific platform, resulting in the creation of the service and the successful completion of the process.

This system aims to generate educational services in the shortest time possible from the moment the service is requested, employing Virtualization, and with the help of MDA philosophy, it ensures a high degree of credibility to the information generated through control policies, thus allowing the provision of applications with different nature under a common model.

3.2.2. Work Plan

The accomplishment of the project has been divided into three phases, and each one of these phases has been established for an approximate duration in days. Figure 6 shows the distinction between the phases and the tasks associated with each of them, as well as the relationship between phases.

	Task Name	Duration	Start	End	Graphic timeline
1	**Project execution**	**126 days**	**10.02.20**	**15.06.20**	
2	Solution approach	22 days	10.02.20	03.03.20	
3	Problem brainstorming	3 days	10.02.20	12.02.20	
4	Feasibilities analysis	8 days	13.02.20	20.02.20	
5	Set pursued goals	1 day	21.02.20	22.02.20	
6	Environment study and request	5 days	22.02.20	26.02.20	
7	Technologies study	4 days	27.02.20	01.03.20	
8	Task distribution	1 day	02.03.20	03.03.20	
9	Implementation	83 days	04.03.20	26.05.20	
10	System environment setup	19 days	04.03.20	22.03.20	
11	Code with selected technologies	37 days	23.03.20	28.04.20	
12	Configure system technologies	15 days	29.04.20	14.05.20	
13	Deal with errors	5 days	15.05.20	19.05.20	
14	Ensure system robustness	3 days	20.05.20	22.05.20	
15	Review	4 days	23.05.20	26.05.20	
16	Evaluation	19 days	27.05.20	15.06.20	
17	Test code execution	7 days	27.05.20	02.06.20	
18	Monitor worst cases	4 days	03.06.20	06.06.20	
19	Analyze feasible situations	5 days	07.06.20	11.06.20	
20	Performance test	2 days	12.06.20	13.06.20	
21	Gathering results	1 day	14.06.20	15.06.20	

Figure 6. Initial project's work plan schedule

For the first phase of analysis and study of the solution, three weeks of work were approached, since it is the basis of the solution's approach and one should think following gradual analysis steps. For the second phase of system implementation, approximately 11 weeks were planned, since it is the basis of the project and it is necessary to coordinate the evolution of the software along with the implementation in the selected environment. And finally, the evaluation phase was divided into three weeks of work together with load tests and system monitoring.

After defining the work plan, setbacks in the implementation of the system have arisen due to the fact that it has been carried out during the COVID-19 pandemic, the request for having off-premises servers on the MCHE has been delayed, so its implementation had to be slowed.

The system was projected to be available in late March, but this was delayed until mid-May when the system became available. Which led to alterations in the code and modification in the time of configuration of the environment which had to be moved from the day that was foreseen to the day that the access to the environment was provided, having to postpone the entire consequent project schedule.

All these setbacks were not so influential for the development because part of the implementation had been developed on a single server that the Faculty of Computer Science, Electronics and Telecommunications (AGH) had provided me before what occurred. In any case, the fact of having this system made the migration process from AGH to the MCHE servers much easier because the technologies and dependencies used were exactly the same.

Chapter 4. System concept design

The purpose of this chapter is to explain in detail and in-depth the design and architecture of the system concept adopted for this project accompanying a comparison of the technologies employed.

With the main objective of reducing the time in which a service is available, the system has been designed in such a way that it avoids human interaction to carry out logical operations and decisions in an orchestrated manner and following serialized data sheets with a high degree of reliability, due to the fact that only an administrator will have privileges to modify the repositories where the decision policies are stored. The architecture of the system as a whole is divided into two orchestrators with different actions but with a bond of union between them.

In the first place, the model orchestrator is in charge of processing the input data which will be transformed into different models through decisions based on policy templates stored in repositories. Once the corresponding transformations are completed, the last model will be in charge of generating the specific data for the platform that best adapts to the established conditions. In addition, this model will assign a priority in the services queue.

Secondly, once at least one service model has been generated, the specific platform orchestrator will process the requested services models through a priority queue one by one. For each service, individually, the application-specific source code by mapping the most relevant serialized data into the source code is generated. Eventually, after generating the source code, its implementation will be executed with the corresponding technology, resulting in a service accessible from the web browser.

In the following subchapters, the design and architecture of the two service orchestration models are presented, which focuses on detailing the functionality in depth.

4.1. Model Orchestration

The architecture of the developed model orchestrator system is presented in detail in Figure 7, which illustrates all the parts of the system and how they interact with each other.

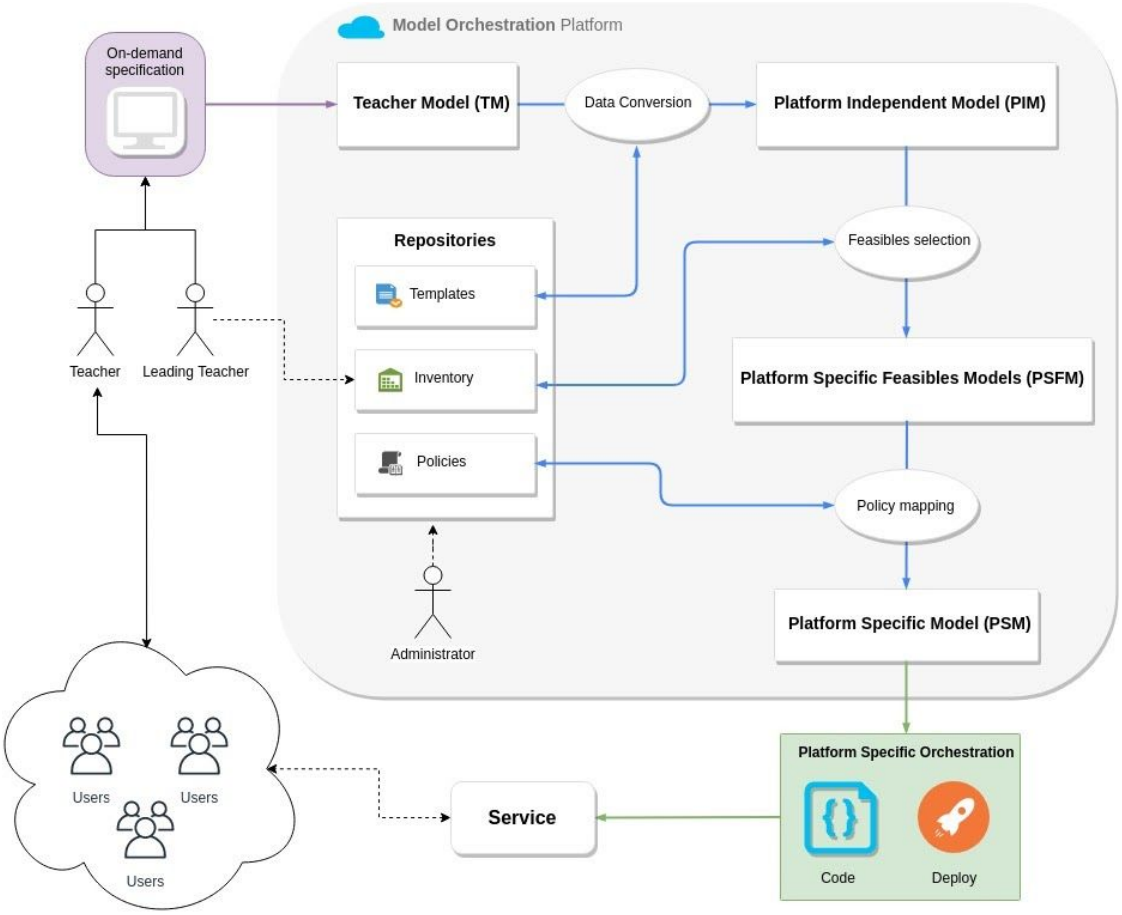


Figure 7. Model Orchestration Platform Architecture

At a glance, it can be discerned between the different parts that make it up and the relationship that exists between the actors presented above. First, a teacher performs the specification request regarding the service from a web form which generates the Teacher Model (TM). Then repository templates are applied to generate the Platform Independent Model (PIM) with the system specifications stipulated in the templates.

Next, the selection of possible specific implementations of applications are applied together with the destination nodes, which are stipulated by the leading teacher. Finally, the process goes on to check different policies specified in the repository that will decide which is the most appropriate implementation for this request and will process the request in the priority queue.

A diagram of UML classes which studies the classes involved in the process together with the dependencies that affect the processing is presented in Figure 8.

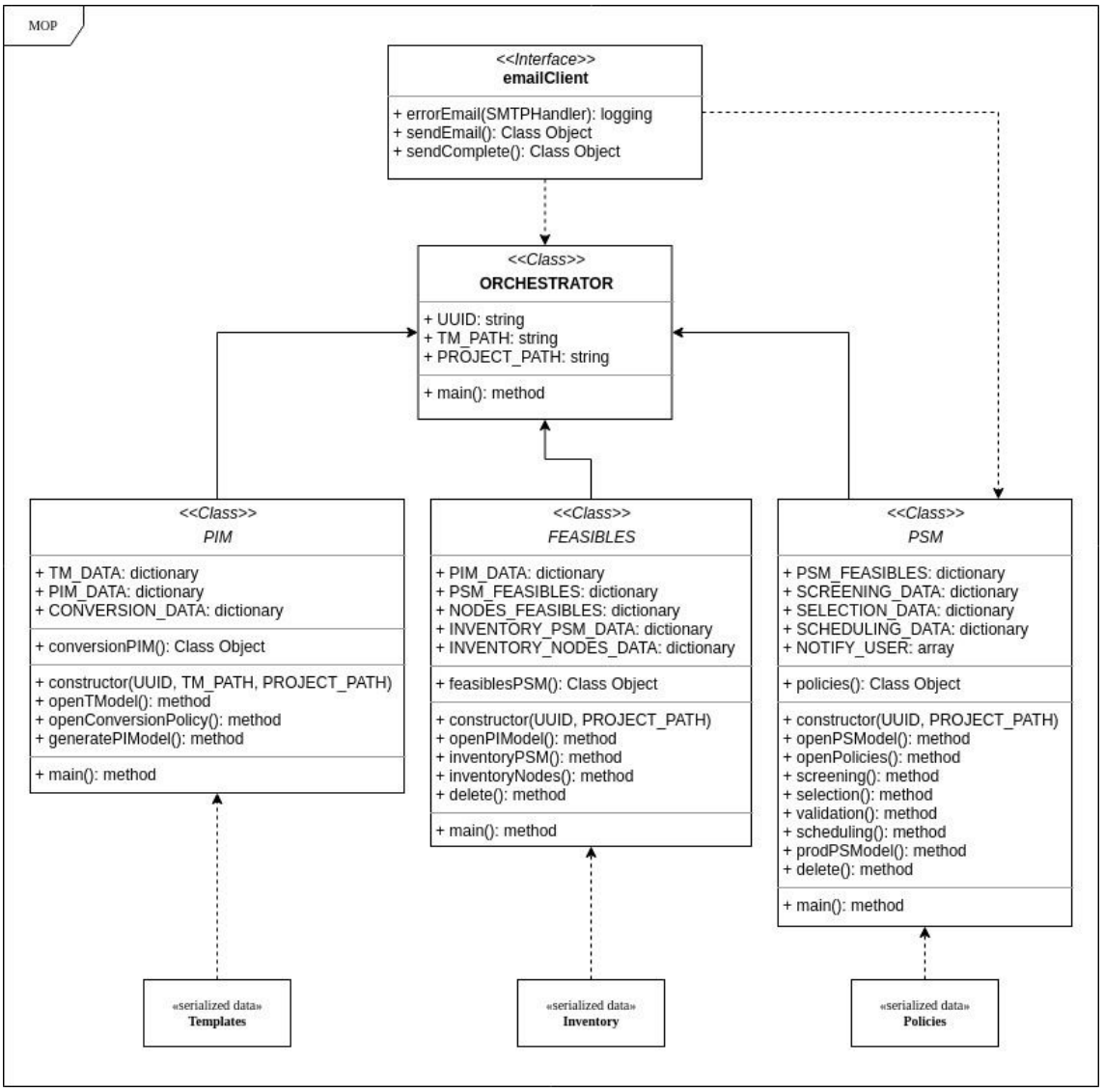


Figure 8. Model Orchestration Platform UML Class Diagram

From the main class ‘Orchestrator’ the processing of the models is orchestrated through the various inherited classes. This main class implements the ‘emailClient’ interface to provide the ease to control the execution of the entire process and to handle potential errors that may occur, thus notifying the system administrator. For each class inherited from the main class, the serialized data corresponding to the Templates, Inventories and Policies that control their processing is concocted from the system repositories. In addition, the ‘PSM’ class must implement the ‘emailClient’ interface in order to inform the teacher of the status of their request by email. The three main models of the system are described in detail hereinafter.

4.1.1. Teacher Model

To start by defining the parts of the system design, it is a must to start with the most independent part of the MDA approach, this is the Independent Computing Model (CIM) that belongs to business logic and is intended to specify the behaviour that should have the system using non-technical data. This model in my solution has been replaced by the Teacher Model (TM), which follows the same philosophy as the CIM but focuses on the main actor of the system, the teacher.

The Teacher Model is the main stage of the process, it specifies the generic data of the service introduced by a teacher. The data is focused only on the most essential aspects to form an lesson-based service which will not have any kind of abstract complexity and will be based on concepts that every teacher will be able to understand. In this way, by designing clear and concise fields without ambiguities, the process can be streamlined, resulting in a solid first impression from the teacher and avoiding the necessity to communicate with external actors to help understand any inadequately designed field.

In this approach, it has been chosen to design a web page to provide a form to easily interact with. Thus, the process is more compact and more manageable than other possible approaches because the data always remains in the corresponding system stored in the database and eases the handling of the information in addition to being able to ensure the integrity of the data as it is a requirement of this project.

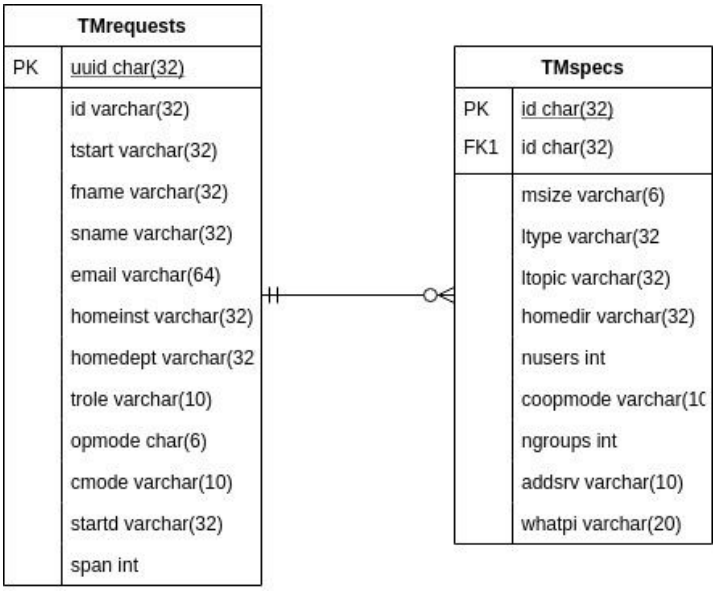


Figure 9. Teacher model database design

As it can be seen in Figure 9, a two-table database has been designed for this model, it is a one-to-one table relationship linking the two tables based on a Primary Key column in the child element 'TMspecs' which also it is a foreign key referencing the primary key of the row of the primary table 'TMrequests'. In this way, a relationship is obtained in each request through the union of the service's UUID between the teacher's data and the service's specification.

This database is used solely for storing and consulting the details of service requests. The data that is generated at the end of the process is not stored in the database since these are explicitly designed to be serialized and loaded from a file to decrease the access time to the database every time a model has been generated. The dynamic behavior of this model is represented in the activity diagram in Figure 10.

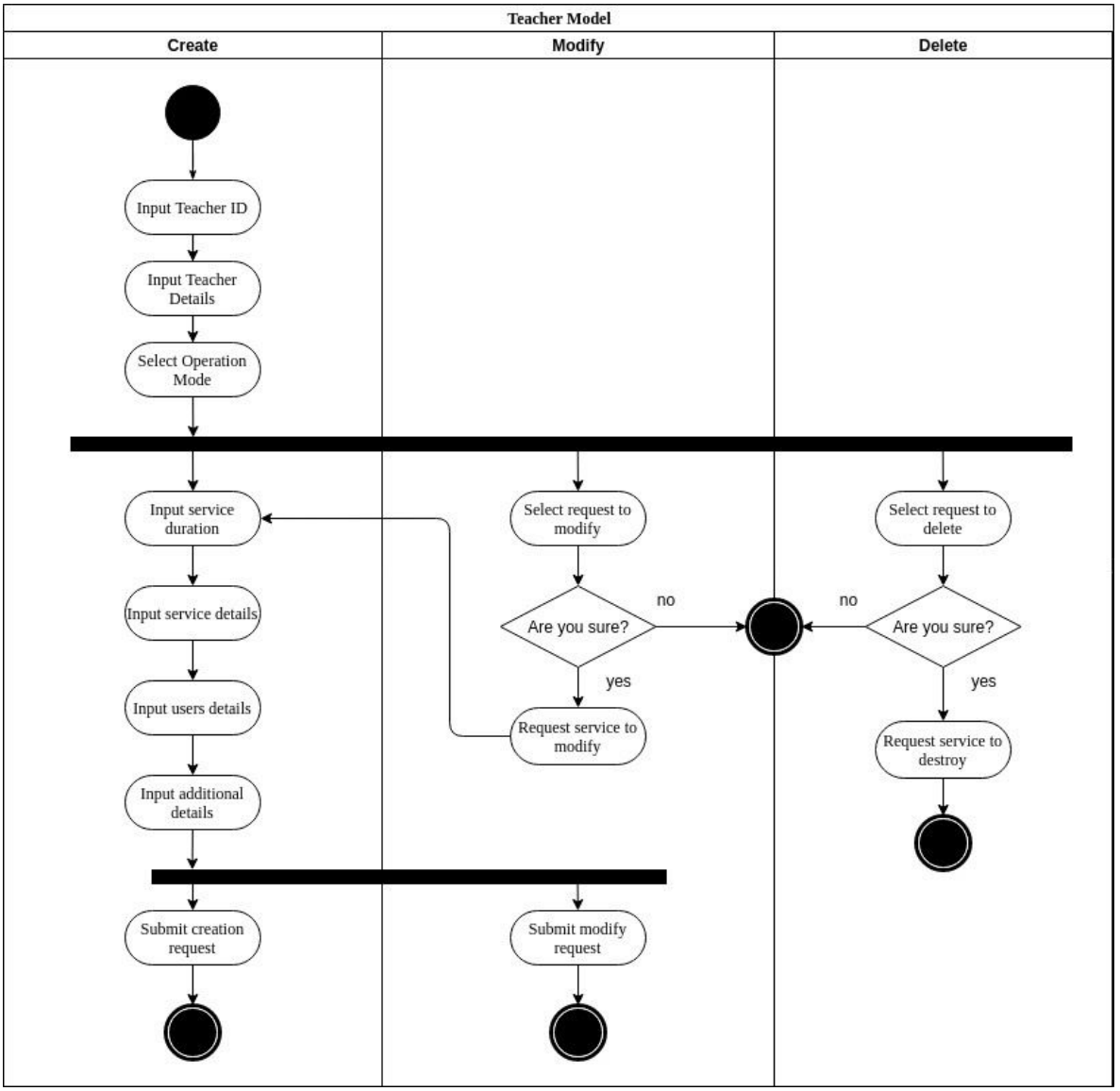


Figure 10. Teacher Model UML Activity Diagram

In the previous figure, it can be discerned the workflow that follows the creation of the Teacher Model, which follows a direct flow with different paths to select. First, once the teacher identifier and its details are entered, then the mode of operation to be performed on the service is chosen. The Modification and Destruction modes can be used only if the teacher has at least one active service.

In the creation process, various sections are classified, first, the duration defined for the service is selected, parameterized by the start date plus the duration in days. In the case of favouring a service with an indefinite time, then when it will be processed by the orchestrator, it will establish a limit between the maximum and minimum range set in the policies.

Alongside, a section for inserting the details of the service is presented:

- **Lesson type**: The different types of lesson types available for services are presented.
- **Lesson topic**: For the type of lesson, the topics within the field of knowledge that are offered are shown.
- **Service size:** Three service sizes are provided: Small, medium, Large. In order to facilitate understanding for teachers.
- **Home directory:** It specifies where the student data will be saved after deleting the service. These can be in the system where it is implemented or in the home institution.

All these variables will be of great help in making later decisions on how to choose and implement the service.

Following, the details of the users are provided, introducing the number of students who will access the service in order to personalize the accesses for each of them. Corresponding usernames will be specified and then the mode of cooperation between them will be selected; in isolated mode each user has their separate environment without having interaction with the other students, in group mode the different groups of users who will collaborate together are selected, in common mode there is a common point of collaboration between all the students of the service.

Finally, it is specified what data is going to be stored either User Data or User Statistics, in addition to selecting what additional services are needed such as SSH access or Web Interface. The service is submitted successfully if there is no missing information to fill in and then the teacher model is processed in the system.

4.1.2. Platform Independent Model

As a second model in the process, the Independent Model Platform appears to specify information about the system without referring to details of the specific platforms. Fundamentally, the model converts the general data that a teacher entered in the Teacher Model into technical information about the system infrastructure where the services are going to reside, for this the gathered information is abstracted from the possible implementations of the service specifying only the essential generic data that later will serve to conduct various feasible services.

The entry point of this model is the output data of the TM model after a request has taken place. The mode of operation selected in TM will be the conditional point in the system, in view of this, the Create and Modify modes of operation operate in an identical direction following the same flow, rather with the Delete action, the previous rules are leapt labelling it as a service to be destroyed. For the purpose of this project, the three types of actions have been designed and implemented, but it will be focused on the process of creation, which is the subject of study in this chapter.

As said before, the most important data for the conversion are processed such as the size of the service, the type of lesson chosen, the mode of the persistence of data or the type of teacher who is requesting the service. Next, the template repository is consulted where there are four different types of system templates, these are the system characteristics, user privileges, data persistence, network description and firewall rules. Each of them specifies how the system is independent of the platform in a technical approach, therefore for the requested service data to have relevant information for the system, a conversion of certain data must be carried out. This is where the generation of the PIM model comes into place, for this the conversions are applied one by one.

The following Table 1 shows the relationship for each template between the generic data in TM that is converted during the PIM generation process.

Table 1. Conversion rules from TM to PIM

# Template	TM	PIM
System	Small / Medium / Large	*vCPU, RAM, OS, Architecture, Storage, Virtualization
Privileges	isolated / groups / common	** Admin, Users, Groups, data directory

Persistence	Where: MCHE or home institution	SFTP network address
Network	-	Protocol: Dynamic / Static
Firewall	-	Action: Allow / Deny

* Values depend on the lesson type selected.

** Values depend on the user cooperation mode and the teacher type.

It can be seen how the system specified by the teacher is transformed into technical concepts such as virtual CPUs or system RAM, the values of which, in addition to being based on size, they will depend on the field of study to which it is applied. For example, Population Genetics-based services will require more vCPUs than a Basic Programming service in order to run different analyzes in parallel. In addition, the modes of collaboration between users are transformed into privileges for each type of actor based on the type of teacher who requests the service.

Regarding data persistence, the teacher only specifies where to store the user data (results, statistics), it is in the transformation process where the address of the server in which the data will be stored is specified for each potential destination. Following, the fields of network access and firewall rules are defined independently of the data entered in TM. Concerning the network, the possible protocols for acquiring network addresses are designated, by default DHCP is the preferred dynamic protocol, but the data can also be particularised manually.

Lastly, for generating the firewall rules, two important actions need to be taken, such as allowing or denying traffic. By default nothing is denied, for this, the firewall configuration or IPS system will always be taking into account that rules are not overwritten. An extended range of ports will be allowed so that it can host several services consecutively and thus having controlled a range of ports.

Once the data has been converted into relevant system information, the first transformation between PIM and PSM is carried out. In this transformation, every specific platform and the destination nodes specified in the Inventory that are most feasible are selected according to the PIM information consulting the inventories repository. I. In this way, it is possible to bring the previously generated model closer to specific platforms by relating to the most relevant fields when determining the most suitable service to provide to the teacher and their students.

On the one hand, it begins with the selection of feasible specific platforms, for this purpose three important variables come into play: machine size, type of lesson and topic of the lesson. First, the size of the machine describes the virtualization technology with which the platform is implemented on, in addition to the corresponding data in the system that the technology in question must satisfy. The other variables are linked in the sense that they jointly describe the field of study to which the requested service will be oriented since the services are based on interactive lessons, hence the variable lesson topic will specify the lessons to be loaded in the platform while implementing it.

As mentioned in 3.1.1, the person responsible for providing lessons to the inventory repository is the leading teacher. For each different PSM there is an individual serialized data file with the personalized data and the topics with which the platform can be associated, each of these are dynamically loaded into the system with the ease of being able to add new ones without affecting the behaviour.

On the other hand, feasible inventoried service destination nodes are analyzed. These nodes describe the potential systems where the implementation will take place in, as explained further in Chapter 5, one of the possible implementations for this system is based on the model of control Master/Worker. In the inventories of nodes, as in PSM, the node descriptors are collected, thus making environment scaling straightforward.

Each node follows the same data definition pattern, they are described in two blocks: network and specifications. The first one specifies connection related parameters like IP / mask, FQDN, DNS and Gateway, which are essential to identify the node and establish the connection for the implementation. Subsequently, the system specifications of the node are specified with the same fields as in the template used for the conversion to PIM. Each system data is analyzed and adapted to the data that has been specified about the system in PIM.

For selecting the nodes, the specifications of the system required in PIM and the system of the node in question enter into place, each attribute of both is compared between thus giving rise to a number of successes. To select the nodes that can host the service, it is required that the number of comparable events be equal to or greater than four, in order to ensure that the majority of the attributes are accomplished.

Eventually, every specific platform and feasible nodes are dumped on a new serialized data file synchronically with the foregoing PIM data.

4.1.3. Platform Specific Model

As the ultimate model, the Platform Specific Model is responsible for establishing logic in the process by validating and applying policies to previously generated information, which must be satisfied to proceed to generate the PSM model or, instead, the request will be completely rejected, and appropriate notification will be sent to the requesting person

As in the previous models, the most important variables must first be gathered from the previously generated model. Ensuing, policies data to be employed are dumped into three separate dictionary-type variables, in order to have reliable manageability of the data. For the design of this system, three distinctive policies have been applied each with a distinct purpose, in the following Table 2, the description of the three policies along its key purposes is shown.

Table 2. PIM Policy Repository

# Policy	Description	Keys
Screening	Verify values and notify	Maximums, Minimums and minimum time before the start
Selection	Select a PSM and a Node	Machine size and lesson type
Scheduling	Time to instantiate start and finish	Preempt

The screening policy is principally responsible for validating that the data of interest comply with the maximum and minimum values established in the policy. First, the policy verifies that the start time of the service selected by the teacher is greater than one day from the time of submitting the request. Subsequently, maximum and minimum policies are verified for the number of users, groups and duration of service. In the event that the duration of the service would be indefinite, a duration between the maximum and the minimum specified in the policies will be assigned.

In addition, in the event that any value is not compliant with the policies, the requesting teacher will be notified with the policy that has not been complied with and will be encouraged to modify the service request. Regarding system notifications, the types of behaviours that can be addressed are the following:

1. First, in the situation entirely verified data is correct, it is first required to notify the teacher that the request has been successfully processed in the system then it will proceed to check the subsequent policies.
2. Instead, if any errors occur during validation, the teacher will be notified in detail with the fields that comply with the stipulations of the custom policies.
3. Furthermore, in the case of a request for destruction of the service by the teacher, it will also proceed to notify that the initial request is ready to be destroyed, for this, there is no verification carried out.
4. Furthermore, external system errors are e-mailed to an administrator if some data is lacking or if some errors are handled.

Once the data is verified and complies with the policies, the Selection policy proceeds to decide which platform is the most suitable for this request. For this, the selection policy is consulted, which for each PSM specifies the different system sizes that are suitable for this platform. In the event that a platform does not require many resources to operate it will not be necessary to designate systems with abundant resources so as not to misuse these. In addition, the fields of study to which this platform can be applied are particularised, that is, the types of lessons that can be carried out with regard to the characteristics of the platform.

With these data, for each PSM the policy data values are compared with the request data, giving rise to a number of matches for each PSM. The determination is performed by obtaining the PSM with more properties favourable to its characteristics, that is, the platform is related to what the teacher is requesting for and satisfies their educational requirements.

Once the PSM is selected, the selection process proceeds to decide which node is the most feasible for performing the implementation of the service. For this, for the choice of the node, a further dynamic type of policy is convenient rather than in the case of the choice of the PSM where the policy is static, that is, it is checked in real-time for each node, what is the health status of the node with regard to the computational load and the current RAM available. Therefore, for each node checked, the values are stored in a dictionary and therefore it is determined in the dictionary which node has the most utmost RAM memory available at that time among others and solely one node is selected.

Eventually, once the selection process is completed, the request is waiting to be processed in the system. As it is a system oriented on providing educational services, it has been designated to implement a preempting priority queue scheduler, in which the requests that are processed in the system are handled, assigning a priority so as the requests that arrive first will be attended with more elevated priority than the rest.

Moreover, a preempt file is created, that is independent of the file with the request data, in which the parameters associated with the preempting of the service are defined. These parameters include the priority assigned as an integer (starting from zero), the destination node and the date on which it was explicitly requested for implementing the service.

Lastly, the information processed during this model, such as the data of the node and the selected platform, is dumped into a production file, which must be backed up by the technique that is deemed appropriate since this information is unique for this service and represents critical information.

4.2. Platform Specific Orchestration

Once the operation of the primary orchestrator has been defined, now it is analyzed how the implementation of the service is assembled on the specific platform orchestrator based on the information generated in the previous model orchestrator. To proceed to analyze it is important to attest how the architecture of this second section of the system is and how the internal components are associated with each other. For achieving it, in the following Figure 11, the system developed for the implementation of the Orchestrator Specific Platform is presented in detail.

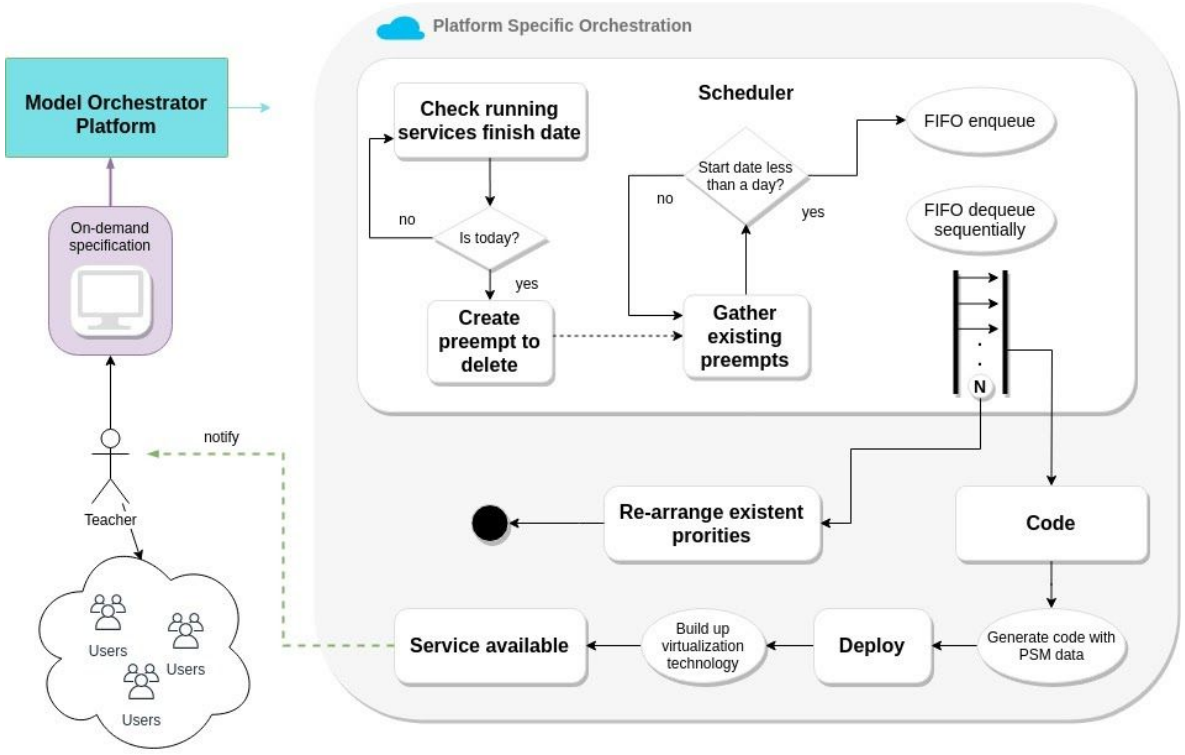


Figure 11. Platform Specific Orchestration Architecture

As can be observed, the principal entry point through this system is presented by the service scheduler. This system is developed to be executed at a certain point of the day, preferably outside the common working time slot, depending on the project where the platform is imposed, generally, after midnight it applies in most cases. Once it is executed, each of the current system requests will be processed together with the services that are already implemented and operating.

First, it is checked for each service that is currently operating whether its end date stipulated in the initial request is corresponding to the current date when the system is running. In the case where the end date is a prospect, the service is disregarded, however, if the date is smaller than a day, the service is explicitly acknowledged to destroy and a preempt file is produced with a priority given and pointing to the destination node where the service is running.

Once those services are examined and the preempts generated, then each of the preempts that are in the scheduler repository at that instant are gathered, verifying that the date to start the creation/destruction of the service corresponds to the current time. For each preempt that meets the stipulated condition, its service UUID along with the priority will be queued into a priority queue. Later, when the last preempt has been queued, each queued service execution will be performed in ascending order concerning priority. Each service is executed sequentially while there are services in the queue, that is, while the queue is not empty.

Individually, for each service, the system proceeds its execution with an identical logic of the program. First, four major variables are selected from the model generated in PSM for this process, those are the mode of operation, the name of the PSM, the virtualization technology of the PSM and the IP address of the destination node. For each separate PSM, there is a source code file, which can be derived from a generic template for further PSM, with identical classes (code, deploy, destroy) defined but with varying in terms of internal functionality. Likewise, the PSM code is imported as a module by its name as a variable and it is checked what mode of operation the service has allocated.

For the creation process, code and deploy classes from the PSM source code are executed in sequential order. First, the Code class is invoked, which is generally responsible for producing the source code linked to the PSM by including the generated information from the model orchestrator. Aforementioned source code formation develops four actions conditioned, first, the variables extracted from the PSM model corresponding to variable parameters of the service such as the identifier, the platform specifications and the firewall rules are settled.

Subsequently, it is examined which ports are open on the destination server in the range established in PIM using the firewall template values and performing a server socket validation to notice in which ports the server is currently listening to. The validation starts with the minimum established port until it finds a port that is open, then

it will be assigned to the service. Later, the source code is copied to a specific folder in order to replace the most significant variables that influence the operation and identification of the service.

To carry out this replacement, during the process, the data gathered is stored in a dictionary with the name of the source code variable and the value to be replaced with, in this way, it is more straightforward to iterate and manage the data for each source code file needed. After iterating over the required source code and replacing the values, the code generation process concludes.

Finally, the script that comprises the sequential steps required to start the service is executed on the destination node, halting at any time if an error appears during execution and so to immediately notify the system administrator about the issue. Once completed, the teacher who has requested the service is notified with the relevant details of the service along with the access instructions.

4.3. Choice of implementation technologies

Once the architecture and design of the solution have been explained in detail, it is important to establish the criteria for choosing the technologies used to implement the system. The present chapter presents the eligibility criteria in order to comply with the requirements analysis detailed in Chapter 3. As the web service to which the solution approaches has as its main objective to generate a file in serialized format to later be processed, in addition to being a service that will not have a large load of concurrent users because it is a form that is accessed mainly to query the service, the implemented design has been thought to take into account aspects such as security and robustness.

Following the above described, several distinct technologies have been used for the implementation of the system, which will be categorized into four different blocks for a better understanding of the scope.

1. Implementation of the request reception system and web form.
2. Request data serialization language
3. Language used for the development of the system source code
4. Application and service virtualization technologies

On the one hand, the solution implemented for the Teacher Model explained in Chapter 4.1.1, has been devised under the concept of the LAMP Open Source software package (Linux, Apache, MySQL, PHP) which makes up a web service stack to dynamically provide the webform and have absolute control of the request reception system. The stack architecture is based on the Presentation-Business-Data architecture in which it distinguishes three independent systems implemented in different technologies but connected together by a dedicated network.

As another possible option is the MEAN software stack (MongoDB, ExpressJS, AngularJS, and NodeJS) which has Javascript as its main programming language, which helps to simplify the web development process. The reasons why MEAN technologies have been dispensed with are various, which are going to be explained and compared.

First, the reason for choosing Linux as the main operating system is because it is a kernel capable of optimizing system resources to the maximum, stable, reliable and with extreme security backed by communities of developers that support it. The solution design is designed to work on Linux Kernel-based operating systems, however, in the case of MEAN it provides the facility to port applications to any operating system only with NodeJS installed. MEAN option would be ideal in the case of choosing to deploy an application with microservices distributed across several platforms.

Thus, choosing the LAMP stack allows the web form to be designed with the PHP server language, which is highly compatible with the MySQL database management system (DBMS) through the database connectors, supported by Linux systems, to store and gather each of the teachers' requests. Instead, developing the MEAN stack requires writing the entire web service code in JavaScript, both the client and server sides. Apache has been used as a web server due to the fact that it is a more mature and robust server than NodeJS, in addition to having great compatibility with the PHP files that make up the web form.

Lastly, LAMP has great support for coupling the stack structure to different languages, in this implementation PHP is used for web pages but Python could be used which would give great support to the entire system because the main language of the Source code of this project has been developed with it. However, to develop the application with MEAN, unique knowledge of Javascript is needed, which does not leave much flexibility to the developer.

LAMP is the most complete stack to implement the web form for the requesting system. A summary comparison of both technologies is given in Table 3.

Table 3. Implementation of the request reception system and web form

Technology	LAMP	MEAN
Portability		V
Robustness	V	
Compatibility	V	V
Widespread Support	V	
Languages	V	

On the other hand, the results of the requests are exported into a serialized data format for better data management and thus store data in an organized way to work later on them. In addition, the format is used to specify the different repositories of inventory, policies and templates, so it must comply with the requirements to apply the several repositories in the system.

The two technologies that are compared based on policy security criteria and data manageability are YAML and JSON. On the one hand, YAML is visually easier to understand the structure and therefore facilitates the readability of the data. Instead, JSON bases the data in the form of an array so it may be more unreadable after an administrator tries to change data in the system giving rise to possible errors or confusion.

Nevertheless, YAML has the benefit of being able to integrate code sections of JSON (or XML) code within the same YAML file. This implementation has taken advantage of this possibility in policies such as Screening, as can be seen in Table 4, the structure is defined in YAML but a JSON formatted set is introduced in the values

Table 4. YAML code belonging to the Screening policy.

```
---
maximums:
  teacher:
    small: {users: 15, groups: 5, availability: 7}
    medium: {users: 25, groups: 8, availability: 3}
    large: {users: 35, groups: 11, availability: 1}
```

However, JSON is a widely portable language between technologies and for processing data faster given its natural structure for programming languages (Eriksson, Malin and V. Hallberg, 2011). For Yaml, external modules or frameworks are typically needed to be able to operate with the data and have integration with the technology. In the case of this implementation, external modules have been required to implement with Python and PHP.

Regarding the expression of access control, in the YAML specification ACLs can be defined for the different actors of the system providing a different role expressed by policies through permissions and privileges. In YAML, it is easier to define these user roles than in JSON. It is generally preferred to use YAML to define the other policies based on the readability it provides.

Based on (Duflos, Diaz, Gay, Horlait, 2002) research, other types of Policy specification can be observed to be able to implement it as future work. As for example ISPS could be implemented for distributed communication in case security and readability are pursued as a primary objective. In this work, it is preferred to keep everything specified clearly with a single language between technologies to be able to handle the data in the same way. In the following Table 5 the comparison is summarized.

Table 5. Data serialization language comparison

Technology	**YAML**	**JSON**
Readability	V	
Quick processing		V
Natural processing		V
Access control	V	
Policy definitions	V	

Regarding the main structure of the system implemented in this project, the technology chosen to write the source code has been Python. It has been chosen mainly because it is the language that I have the most experience with, as a rival between the choice was Java, with which I started programming for the first time. In addition to personal experience factors, the two technologies will then be compared to discover which one brings the most benefits to the project.

On the one hand, due to Python being an interpreted language it executes relatively slower than a language compiled like Java. In general, the execution speed of a language can be compared only by testing the same implementation for both languages, for this, there is a project called 'The Benchmarks Game[7]' which compares the benchmarks for executing specific programs.

The syntax used by Python is dynamically typed so the interpreter will check the data types of the variables at runtime, as well as eliminating the enclosing braces to follow the indentation patterns established in the official PEP8 style guide. In the case of Java, it is the opposite, all types are statically defined before compiling the code making the task easier for the developer to understand the code, however, with dynamic values it makes it very difficult to understand.

An important point about Python that has been very useful for this project is the use of modules, which makes it easy to implement and work on YAML files, in addition to supporting modules to perform operations on the operating system or even import classes as other modules. This functionality has allowed us to have a very wide range of possibilities, in which all kinds of data handling in addition to communications distributed between servers through the use of sockets have been possible.

Another reason is the cross-platform portability that it offers with the ease of installing only the Python interpreter supported by all operating systems. On the other hand, with Java to have this portability, it is necessary to install Java together with the Java virtual machine (JVM) in order to compile the source code. Hence, with Python, the process is faster and more dynamic. In Table 6, the comparison summary is presented.

Table 6. Programming language comparison for the source code development

Technology	Python	Java
Self-experience	V	
Syntax		V
Modularity	V	
Portability	V	

[7] Source: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-java.html

Eventually, in the fourth block, two technologies that have been implemented together but differ in their essence as concept and architecture will be analyzed, in addition to having a different behaviour in the system. The implementation of the services for this project is intended to be integrated under the Virtualization concept in order to use the maximum capacity of the resources of the servers that are available.

Within virtualization there are two concepts that are important to differentiate, one of them are virtual machines which emulate a complete guest operating system on top of a hypervisor host, allowing different systems or applications to run on the same server. The problem is that these virtual machines tend to occupy a lot of RAM when they are run, that is why containers appear here, which are executed on a specific container engine, thus being able to deploy containers that run applications with their own libraries, eradicating the need to have an operating system dedicated for each.

In this project, the main objective is the online lessons, which are implemented in two examples of PSM as explained in Chapter 5, these PSM are each built with different technologies. First, Vagrant is used to create a workflow to provision a virtual machine application. Instead, for the other PSM, Docker is used to deploy the corresponding containers applications. Vagrant is compatible with any operating system because it can contain any complete operating system stack inside, however, Docker containers only run Linux-based applications if it is run from a Linux host.

Despite using the two technologies, there is an inclination for Docker for this implementation of the system, since the main objective is to minimize the deployment time of the services, Docker performs a particularly role in this time, since deploying any container in this technology results in an almost instantaneous action, however, with Vagrant, it first processes the workflow defined in the initial file and then deploys the service with the selected hypervisor giving a higher waiting interval, which slows down the overall process. In Table 7, the comparison summary is presented

Table 7. Application and service virtualization technologies comparison

Technology	**Vagrant**	**Docker**
Virtualization	Virtual Machines	Containers
Resource friendly		V
Deploy time		V
Widespread compatibility	V	
Performance		V

Chapter 5. Description of implementation

This chapter provides a formal representation of the implementation that has taken place in the Malopolska Educational Cloud (MCHE). Although it does not contain complete descriptions of the logic employed, the description illustrates what sorts of behaviours can be particularised and, using one of these as an example, it presents a specification and describes the process in broad terms.

The final result of the project has been implemented in the MCHE cloud, but as explained in point 3.2.2, due to the setbacks experienced for the provisioning of the servers, implementation has had to be delayed, although finally achieving the proposed objective on time. Figure 12 shows the architecture distribution of the virtual implementation of the servers in which the asymmetric Master/Worker communication model is adopted, where the master server controls the N worker servers that will host the technologies and thus the application services.

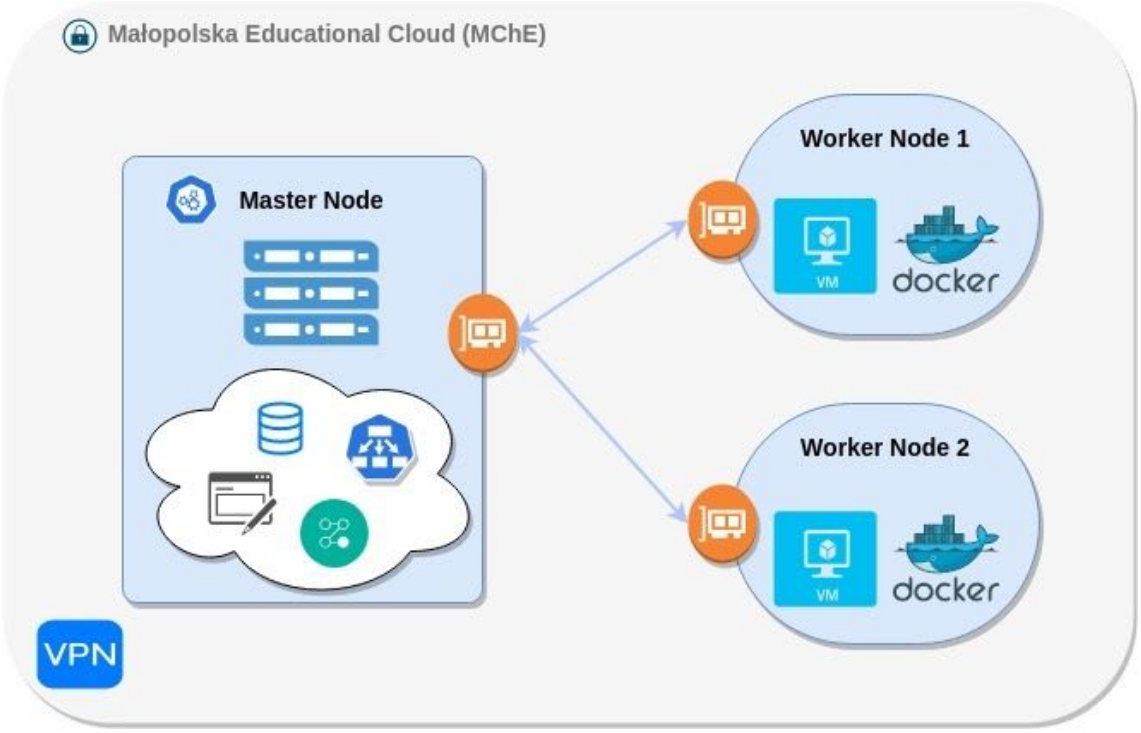


Figure 12. The virtual architecture of the implementation in MCHE

Each of the servers has a distinct role and, therefore, the resources that have been requested differ between servers. Table 8 shows the technical specifications of network connectivity and resources that each of the machines has. The connection is only from Master to each node, there is no direct connectivity between nodes because for this implementation the objective of balancing the load between node servers is not pursued nor is it intended to achieve the characteristic of rapid elasticity in cloud computing in which in an unusual user load, several instances of the service would be replicated between nodes.

Table 8. Technical specifications of the server environment in MCHE

Servers	**Master Node**	**Worker - Node 01**	**Worker - Node 02**
Static IP address	172.26.91.18/24	172.26.91.20/24	172.26.91.20/24
Gateway	172.26.91.254	172.26.91.254	172.26.91.254
DNS Servers	172.26.64.[3,4]	172.26.64.[3,4]	172.26.64.[3,4]
vCPU	2	8	8
RAM	4 GB	64 GB	64 GB
Storage	16 GB	16 GB	16 GB
OS	CentOS	CentOS	CentOS
Virtualization	IVT (on all cores)	IVT (on all cores)	IVT (on all cores)

Each server follows a related pattern of specifications, each has CentOS Linux distribution as an operating system which makes the implementation more manageable by having to use similar commands to configure and manage it. Next, Intel Virtualization (IVT) is enabled on each virtual CPUs so that it allows the operating system to perform dynamic resource management, improving system performance in order to be able to nest virtual machines and containers in oneself.

Furthermore, a user account is available in the LDAP directory distributed under the MCHE domain, however, during the implementation, the difficulty for using this account has risen due to the impracticality of employing the public key authentication via SSH which has permission restrictions set by LDAP policies. Notwithstanding, an account with administrator privileges has been created which likewise has the limitation of using only the disk capacity of the root partition, it has 16GB of which 8GB is occupied by the operating system and software libraries.

On the one hand, for the estimation of the vCPU and RAM, they have been approximated by means of an estimation formula based on the maximum number of concurrent users. Following the formula for estimating the vCPU needed:

$$CPU\ est. = (Est.\ number\ of\ users\ x\ Est.\ CPU\ usage\ per\ user) + Overhead$$

The number of virtual CPUs required per server to host a maximum number of 330 users, of which it could be expected that during working hours there will be 50-60% concurrent, and an estimated maximum CPU load per user of 0.03 adding an estimated overhead of 20%, we obtain that it would be advisable to use 8 vCPUs per server.

Next, following the formula for estimating the Memory needed:

$$Memory\ est. = (Est.\ number\ of\ users\ x\ Est.\ memory\ per\ user) + Overhead$$

On the other hand, as for the previous calculation, it is estimated that each server will host a number of 330 users, of which it could be expected that during working hours there will be 50-60% concurrent, and an estimated memory usage per user of 256 MB. Adding an estimated overhead of 128 MB for the different types of services oriented to different fields with different requirements, we obtain that it is recommended to use approximately 64 GB per server.

These previous calculations are applied for the possible worker nodes, in the case of the master node, it is different because there will not be numerous users (teachers) connected concurrently, in addition to the fact that the source code of the developed system does not run in parallel, therefore with few virtual CPUs it's more than enough for this implementation case study.

Furthermore, as reasoned throughout chapter 4.3, a web server has been implemented to host the web form where teachers perform requests. Detailed information on its usage can be found in Annex I. The Apache web server has been configured with a virtual server with the idea of being able to implement SSL verified by a Certificate authority (CA) in the future, so just by adding the new certificate on the virtual host with port 443 (HTTPS) it could enforce even greater security for requests to the server using SSL / TLS. In this implementation, self-signed have been created to fix what the configuration file will look like once verified. In addition, you can redirect traffic from port 80 (HTTP) to it so that it never accesses the insecure port by default.

For the configuration of this implementation, the basic security based on best practices for Apache web servers was taken into account. For example, disabling the server signature protects the server from imminent threats that could target attacks on the Apache version that is displayed when an address is wrongly entered.

The web pages that this server houses are implemented in PHP and are divided into three files. The first is in charge of greeting the teacher and displaying the main web form where his details are first presented. Once the user chooses the mode of operation, depending on the mode selected, in the second file the form corresponding to will be shown with the details of the service to be created. Finally, when the information is submitted, it is processed by inserting the data into the MySQL database, generating the YAML file with the parsed information and displaying a statistic in the form of a pie chart to the user of the most requested services in the system.

After the request has been submitted and a file in YAML format generated with the data entered in the system, then a Bash call is executed first changing the user context from apache to the user that houses the source code along with the service's UUID as an argument for the main source code Python call where the models are orchestrated.

With regard to the implementation created for the source code of the program with Python, it does not differ from the functionality presented in Chapter 3, so it won't be attempted to explain the functionality in this Chapter again. However, glancing at the Annexes can be comprehended how it has been implemented with Python in addition to being able to know the requirements to further add more functionality to the source code by a contributor.

For this implementation, as presented in Table 9, two PSM services have been employed to prove the functionality of the system, each implemented on different virtualization technology.

Table 9. PSM relationship with the virtualization technology implemented

PSM Application	Virtualization technology
Antidote SelfMedicate (NRE Labs)	Vagrant[8]
The Littlest JupyterHub	Docker[9]

[8] Vagrant Documentation: https://www.vagrantup.com/
[9] Docker Documentation: https://www.docker.com/

On the one hand, the open-source project Antidote has been adopted, which has been developed under NRE Labs, to facilitate the learning of the automation of network infrastructure. Which follows the same philosophy of this project as is the provisioning on-demand and serve the service interactively through the web.

The fascinating point concerning this project is that it provides a mechanism called Antidote Selfmedicate to implement an identical service by spinning up a virtual machine with Vagrant, whose lessons can be added to the platform through YAML configuration files. It is not required for these to be related to network automation, there could be applied different topics that involve the use of the terminal since in this service the lessons window is divided into two sides, on the left is the lesson text content and on the right is the Bash terminal.

This PSM is a truly complete service to implement IT-based lessons, due to the variety of modules that can be incorporated into the system, from operating systems with tools for Cybersecurity to different terminals that act as virtual routers with which virtual networks can be created with the ones to interact with and thus helping the secondary school students to understand the basics of Networking, or even more complex designs. The only drawback about this service is that it does not implement any authentication system for users, so the results of the students cannot be easily controlled.

Regarding the architecture that Antidote Selfmedicate presents, each environment is made up of four layers[10]:

- **Infrastructure:** It is presented in the form of a virtual machine loaded on a Virtualization provider. The code to configure and spin up the machine is defined sequentially in a file called Vagrantfile in which the provider to be used is specified, in this project Virtualbox and Libvirt are used for this purpose.
- **MiniKube**: It provides a single-node cluster to the service to give portability and the benefit of providing the lessons to the Antidote platform on demand, an objective pursued by this project.
- **Antidote platform**: It is the platform in charge of loading the lessons on demand and providing the interactive web to learn.
- **Curriculum**: This is where the lessons to be loaded are specified, they are in YAML format by default since it is readable to create your own lessons and the learning curve is not steep.

[10] Antidote Architecture: https://docs.nrelabs.io/antidote/antidote-architecture#lesson-resources

On the other hand, a specific implementation of the JupyterHub project has been chosen as the second PSM, this is The Littlest JupyterHub which is designed to be a lightweight solution to be implemented on a single server with a maximum of 100 users per service, which is perfectly adapted to the needs of the project because each service will have a maximum of 35 users.

The implementation of this service has been quick thanks to the excellent documentation available for this project, the source code has been implemented with the Docker virtualization technology with which in less than two seconds an environment is available to be succeeding configured.

This specific platform is ideal to cover different types of lessons, as an example, this wiki provides a collection of the most notable Jupyter notebooks[11] ranging from topics related to Computer Science to Biology which have been included in the selection policy for further configuration of the services. This collection gives a broad idea of the different applications that can be carried out, which are taken into account to apply the PSM selection policies in the implementation.

In contrast to Antidote, extra features such as user authentication can be added to the Jupyter service through various forms, such as PAM or LDAP, and it can also incorporate a specific results evaluator[12] which allows the implementation of a module so that at the end of the duration of the service the results of all the students are evaluated.

In conclusion, the environment provided by MCHE is ideal for the implementation of the approached system due to the characteristics of the resources provided. The specific platforms meet the expectations and are perfectly suited to achieve the objectives of this work, even though Antidote does not have any authentication method. However, it should be figured that this implementation is intended not to be settled directly into production without having previously configured aspects of security and high availability of the services by balancing them between different nodes.

[11] NotebooksWiki:https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks

[12] Students Evaluator: https://github.com/parmentelat/nbautoeval

Chapter 6. Evaluation procedure

In this chapter, the results obtained during the evaluation of the system in the environment configured in the implementation will be presented by simulating the actions a teacher would perform to interact with the system and produce the results. In addition, tests of the deployment and load time are presented for each of the implemented PSMs to verify their efficiency.

For the evaluation of the system, there have been the same setbacks explained in Chapter 3.2.2, for which the MCHE environment used in the implementation has not been completely adequate to the needs of the one of the two PSMs presented in the implementation. However, since the beginning of the project, a server was available at the Faculty of Computer Science, Electronics and Telecommunications at AGH.

This server was provided to be able to develop the project in its initial phase to have a base with which to further develop the implementation in MCHE. Therefore, this server was not intended to be included in the project, but given these circumstances, an exact replica of the system implemented in MCHE has been implemented in the server at AGH with the peculiarity that there is only one server that acts as master and worker. Table 10 presents the technical specifications of the resources and connectivity that the server has.

Table 10. Technical specifications of the server provided the WIET (AGH)

Server	**Master/Worker Node**
Static IP address	172.29.140.176/24
vCPU	8
RAM	16 GB
Storage	62 GB
OS	CentOS 8
Virtualization	IVT (on all cores)

However, this server contrasts with the implementation in MCHE, aside from the technical specifications, since it exclusively has a single node, which is not an obstacle to adapt due to the modularity of the developed system. It was only necessary to adjust

the number of nodes in the inventory and establish the specifications of this node, in addition to preparing the entire server from scratch to configure the corresponding technologies and configurations described in Appendix C.

Subsequently, three evaluations of the system developed were performed in order to verify its functionality and behaviour. Unfortunately, in the MCHE environment, only Jupyter PSM will be evaluated with Docker technology. Hence, Antidote will be evaluated on the AGH's single server with Vagrant technology and Virtualbox as the virtualization provider.

EVALUATION 1 - Model Orchestrator - Functionality

In the initial evaluation test of the system, the functionality of the model orchestrator is going to be evaluated in order to verify that given various conditions, the results are always the expected, that is, knowing the values set in the policies, it is expected that given an input data the system behaves in one way or another.

For this reason, four different requests will be made in the system, simulating the behaviour that a teacher would have by filling out requests in the web form. The requests performed are as shown in Table 11.

Table 11. Evaluation 1 - Description of the requests performed

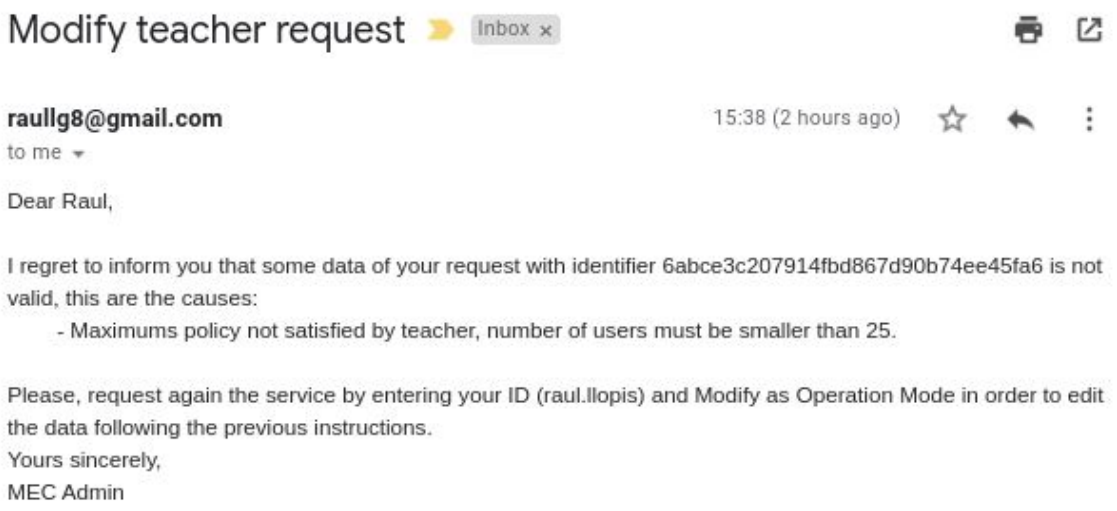
Teacher Model	Request #1	Request #2	Request #3	Request #4
UUID	70520772df6b4b5aa63235039aac64ad	c723fd632040443fa7b36adb9ea03abc	6abce3c207914fbd867d90b74ee45fa6	31ea8b1bafb34ef78ffe07c008b2a59f
Teacher Role	Leading	Teacher	Teacher	Leading
Machine Size	small	medium	medium	large
Lesson Type	it	biology	maths	it
Lesson Topic	Operating Systems	Applied Bioinformatics	Calculus	Networking
Users	8	14	26	14
Groups	4	isolated	common	isolated
Availability	Defined	Undefined	Defined	Undefined
Start date + duration	In 2 days + 14	-	In 4 days + 15	-

In the table above, the most important values for the evaluation are presented, these are the ones that have influence when verifying the defined policies. Each request has been planned in such a way that they differ from each other and, therefore, the developed system can be evaluated by predicting what behaviour should be in each of the requests and verifying what the result is.

Table 12. Evaluation 1 - Predictions and results of shown behaviour

PSM	Request #1	Request #2	Request #3	Request #4
Prediction	Success	Success	Fail	Success
Result	Request completed	Request completed	Request needs to be modified (users)	Request completed

Furthermore, Table 12 presents the prediction of each of the requests that have been made, of which it had been predicted that, particularly, request number three would fail due to the fact that the maximum number of users established in the policies is not met by the requested data. After sending the request, the email shown in Figure 13 has been received, which informs the teacher that the service has been requested and there is the need to modify the service requested data. by describing the infractions. This email shows in details which policies have not been satisfied to clarify what data should be modified.


Modify teacher request Inbox ×

raullg8@gmail.com 15:38 (2 hours ago)

to me

Dear Raul,

I regret to inform you that some data of your request with identifier 6abce3c207914fbd867d90b74ee45fa6 is not valid, this are the causes:

- Maximums policy not satisfied by teacher, number of users must be smaller than 25.

Please, request again the service by entering your ID (raul.llopis) and Modify as Operation Mode in order to edit the data following the previous instructions.

Yours sincerely,

MEC Admin

Figure 13. Evaluation 1 - Mail result of request 3 that does not comply with a policy

In the case mentioned above, no PSM YAML file has been generated because it is pending modification by the teacher. In the case of the other requests, these have been processed in the system by first notifying the teacher in a sort of completion email. In addition, the system has generated the results of the specific platform for each completed request as shown in Table 13.

Table 13. Evaluation 1 - PSM results generated from initial requests

	Request #1	**Request #2**	**Request #4**
	PSM		
UUID	70520772df6b4b5aa63235039aac64ad	c723fd632040443fa7b36adb9ea03abc	31ea8b1bafb34ef78ffe07c008b2a59f
Dest Node	node02	node02	node01
PSM Service	Antidote	Jupyter	Antidote
Virtualization Technology	Vagrant	Docker	Vagrant
Service resources	2 vCPU, 8GB RAM, virtualbox	4 vCPU, 16GB RAM	8 vCPU, 32GB RAM, libvirt
	Preempt		
Priority	0	1	2
Start date	In two days	In one day	In one day

For each request, the process has generated a pair of final results. First, the result corresponding to the PSM production data has been generated. For this model, the most important attributes that have resulted from the policies and the transformations carried out are shown in Table 13. It can be discerned how each request has an increasing amount of resources due to the sizes specified in the teacher model. Moreover, based on the type and the topic of the lessons a PSM service has been chosen. In addition, a node has been determined for each one depending on the memory available in the system.

On the other hand, a preempt file has been created corresponding to the data for the priority queue. These have, unsurprisingly, an increasing priority starting from zero. In addition, the service start date has been assigned based on the data entered, in the case of undefined duration, a start date of a day has been settled.

EVALUATION 2 - Platform Specific Orchestrator - Antidote

In this evaluation test, the time it takes for the specific platform orchestrator to run and build the specific Antidote service using Vagrant on a virtual machine with different amounts of resources is evaluated. This evaluation is carried out once the previous requests have been generated, which have a PSM file with the final data and the preempt with the associated priority, the time it takes to execute the whole process is not evaluated because in a real-life case the service orchestrator would be executed during hours outside of working hours, thus both orchestrators are not intrinsically connected by request actions but rather by a specific scheduler (E.g 'crontab').

Since the server that has been prepared for this unforeseen situation has 8 vCPUs and 16GB of RAM, some reserved for the system, the tests will be carried out adapting the policies to the available resources and will only be measured when there is no Antidote service concurrently running.

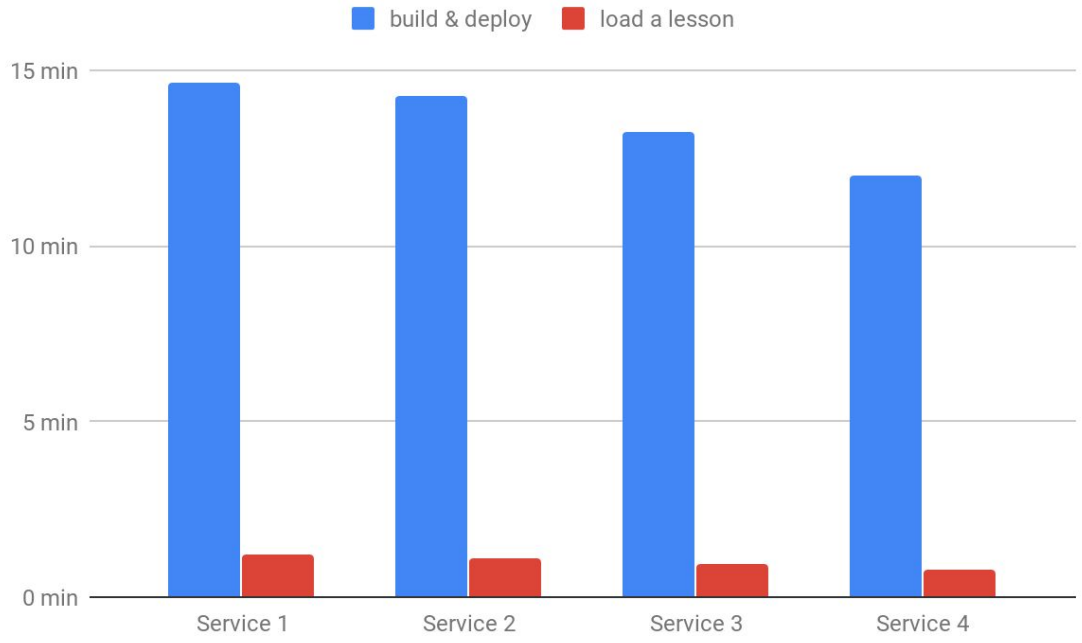
For this evaluation, the templates of the specific system have been adapted to test different combinations based on the maximum resources. Four requests have been created for which it has been predicted that the selected service would be Antidote, and this has been the one selected with the new resources generated. The service specifications implemented for each evaluation are as shown in Table 14.

Table 14. Evaluation 2 - Specifications of the services created to be evaluated

	Service 1	**Service 2**	**Service 3**	**Service 4**
Machine size	Small	Medium	Medium	Large
vCPU	2	2	4	6
RAM	4 GB	8 GB	8 GB	12 GB
Provider	Virtualbox	Virtualbox	Virtualbox	Virtualbox

Subsequently, since the system does not have many resources, I have proceeded to generate the corresponding service for each of the requests individually. The following Table 15 presents the results obtained for the execution of each of the above services using the command 'time python3 service.py' measuring the real time of the duration of the called program, that is, the time it takes to generate to the code and deploy the service, along with the time it takes to execute a 'Linux Basics' lesson which has a virtual machine for interaction with the terminal.

Table 15. Evaluation 2 - Graphical comparison of the creation times of Antidote



Based on the results obtained and presented in Table 15, regarding the build and deploy time, it can be clearly discerned that the specific Antidote platform demands a lengthy time to perform with the Service 1 which has 4 GB of RAM and 2 vCPU, approximately the entire process takes 15 minutes of real-time to run. However, with the Service 4 which has 6 vCPUs and 12 GB RAM, this time is reduced to 12 minutes. This results gives an average time of 13 and a half minutes for the Antidote service to perform in the given server, this time is understandable due to the technologies used by the service within the virtual machine like Kubernetes which delays the arrangement time.

On the other hand, for Service 1, it takes over a minute to fully load a lesson about Linux Basics. In contrast, for Service 2, this loading time for the same lesson is reduced to less than one minute. This happens because the platform internally provides virtual machines and namespaces for each individual lesson, in this case, the loaded lesson has only one terminal (a VM), so for more complex lessons, the waiting time will be even higher.

In this evaluation, it was not possible to verify if there is a malfunction in the system because the execution of the four services individually has been successful at the first attempt, both the request and the deployment of the service have given the expected result.

EVALUATION 3 - Platform Specific Orchestrator - Jupyter

Ensuing, the time required to implement the services with the specific Jupyter platform integrated with Docker technology along with the time needed to load several lessons is evaluated. Unlike Evaluation 2, for this evaluation has been performed in the MCHE environment with the servers disposition depicted in Chapter 5.

For this evaluation, the storage setback could not be resolved in time, this problem occurs because when a third concurrent service is implemented on a server, the storage error occurs, which is notified to the system administrator by an automatic email that informs there is no space left in the system. However, this has not been a problem as four similar services have been distributed between the two node servers. The services have been distributed as shown in Table 16.

Table 16. Evaluation 3 - Distribution of services on node servers

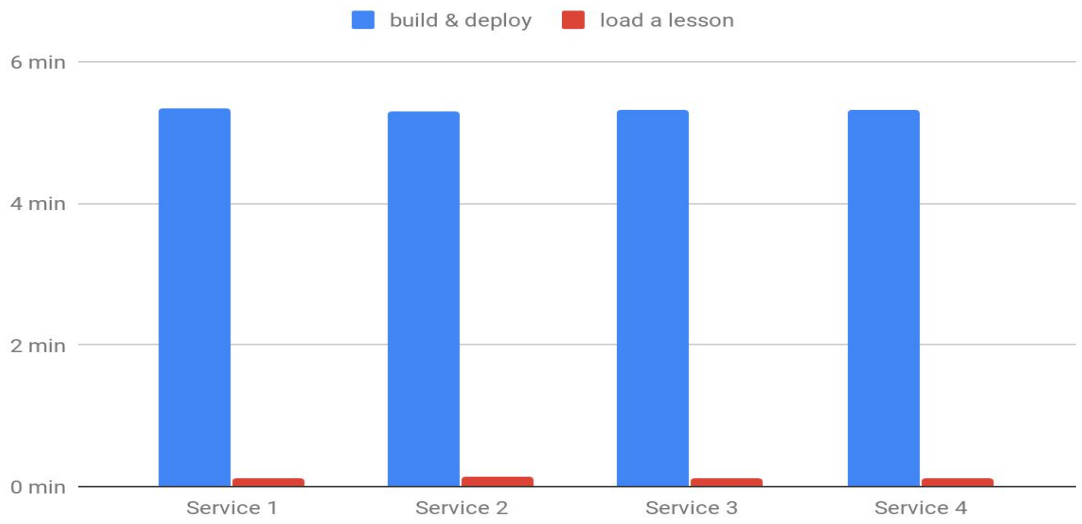
	Node 01		**Node 02**	
	Service 1	Service 2	Service 3	Service 4
Machine size	small	large	medium	medium
vCPU	2	8	4	4
RAM	8 GB	32 GB	16 GB	16 GB

In the case of Docker, both RAM and vCPU limit has been specified for this evaluation. Unlike in Antidote VMs, for Jupyter containers, the limits do not affect the performance when deploying the services. This is because the limit is applied once the service creation is executed, thus being set for the subsequent use of the service users, therefore this is not limited to the execution time.

In the following Table 17, the results of the evaluation for the Jupyter PSM service are presented, which determine the building times of the source code and the deployment of the services accompanied with the time it needs each one to load a lesson about Quantum Mechanics[13]. In this case, the model orchestrator results have been accommodated so that the services ought to be distributed on different nodes and having equivalent lessons.

[13] Quantum Mechanics Notebook Source: https://github.com/jrjohansson/qutip-lectures

Table 17. Evaluation 3 - Graphical comparison of the creation times of Jupyter



As can be seen in Table 17, the build and deploy time of the service does not vary much from one server to the other. In general, this service offers a deployment time of approximately five minutes. Regarding lesson loading times, a similar time of approximately seven seconds has been obtained in each of the services implemented for the identical lesson.

To conclude, in general terms, the evaluation carried out has been positive for both the model orchestrator and the platform orchestrator, for which there was no behaviour that was not expected. With regard to Evaluation 1, it has been verified that the desired results have been achieved for the tests carried out, in which the predictions have been accurate. In addition, in the final results of the PSM, it has been observed that the orchestrator has balanced the load to different nodes and the chosen services correspond to the characteristics of these based on the selected lesson type and topic in TM, thus ensuring the functionality of the orchestrator.

Finally, comparing the last evaluations with the platform orchestrator, the variation in the service deployment time and lesson load depending on the virtualization technology used is appreciated. The Antidote service with VMs takes approximately 10 minutes longer to deploy the service than Jupyter with Containers, moreover the loading time of a lesson is reduced to seven seconds with containers. This evaluation concludes that Docker is the ideal technology to implement specific services for this project because it manages to reduce the time to make a service available.

Chapter 7. Summary

7.1. Conclusion

In this thesis, the cloud-based orchestrator of educational services approach was introduced. The conceptual models that led to its singular design along with the directions in which it can be applied to reach a vast number of distinct educational services have been defined. In the first stages of the project, the main objectives were defined which have been taken into account throughout the development of the project, focusing the design and implementation of the system on methodologies and technologies that facilitate the achievement of these.

First, the primary objective of the project, which gives its name to it, is the design of an orchestration system that would provide educational services on demand. This objective, defined as a characteristic of cloud computing, has been successfully accomplished as planned due to the ability of the system to process subjective data about the service into objective information, resulting in the service deployed in the environment thus allowing access for the teacher.

Furthermore, the choice of LAMP as the implementation stack for the web form has made it easier to demonstrate the on-demand self-service capability by directly providing the form in the browser, which is connected to the main project code that in the moment a request is submitted, the TM input data is linked with the model orchestrator, giving autonomy to the process.

In addition, the objective of providing educational services has been achieved thanks to the implementation of the two PSM based on interactive educational lessons directly in the web browser. This facilitates the achievement of other initial purposes, such as the ubiquitous network access feature that through the existing VPN at MCHE, teachers and students can have direct access to the service in a secure way through any web browser accessible from anywhere, ensuring the data being encrypted end to end.

Another feature that has been achieved is resource pooling by the fact of providing the different sizes of the service and the possibility of modifying the service with which teachers have the false feeling that there are unlimited resources in the environment, giving them confidence about the service provided.

As has already been mentioned during the course of the project, the technology that has had the greatest impact on this project, which is linked to Cloud Computing, is Virtualization. This technology has greatly facilitated the costly task of providing entire environments for each service easily and efficiently using specific virtualization technologies that have facilitated the configuration and deployment of the services. In this way, since the configuration of the service and its deployment is carried out automatically by the system, it is not necessary for a third person to arbitrate in the process.

Furthermore, due to the modularity of the system, the platform orchestrator can be applied to various platforms regardless of the technologies used to develop, always having previously configured the platform according to the template described in Appendix B. This feature also applies to the system nodes, which can be modified in the inventory following the same templates, in this way it can be adapted to larger environments or even to a sinlge node as seen in Evaluation 2 of Chapter 6 for development purposes.

Subsequently, it has focused on implementing two specific platform solutions based on different virtualization technologies, VM and containers. Through this incision in both technologies, the aim has been to evaluate two PSM services based on interactive lessons to determine which technology would minimize the deployment time of the services, thus giving the project a comprehensive approach.

In the case of containers, for the Jupyter service, it has had an initial advantage because the concept under which it is integrated is intended to spin-up applications faster. After the system has been evaluated, the previous prediction has been confirmed, giving a deployment time for containers almost four times lower than with VMs.

To conclude, I am generally satisfied with the results obtained in the project despite the obstacles that have arisen during the course of the project. Personally, I wish I could add more technologies and features like rapid elasticity for dynamic resource provisioning. However, these pending tasks are included in the next chapter to specify in detail which aspects should be addressed in the future work or contributions in order to improve the system presented in this project.

7.2. Future Work

In general, there are still some aspects that I would have relished to cover in this project, however, due to the short-term duration of the project and the unforeseen events discussed in chapter 3.2.2, these topics have had to be left to be implemented in the future work in order to achieve them. Future work will include a series of topics to be further developed, these are:

- Modify and delete operation modes
- Add extra security to the system's implementation
- Manage users and link users IP's via LDAP
- Adapt services to be containerized

First, the project has incorporated the global functionality to modify and delete the services. The Modify mode works when the service has not been created yet, thus modifying the serialized data. In the case of Delete mode, the functionality up to the PSM data has been implemented. However, both modes do not have any functionality beyond the first orchestrator, so as a primary aspect for improving the functionality of the system is to implement for each PSM the code corresponding to these modes of operation with which to modify the resources of the service that is running and to be able to destroy it when the duration ends, previously saving the results of the students in the specified home directory.

By implementing the Modify mode, it would be possible to achieve the characteristic of rapid elasticity with which teachers in the case of incorporating new students to the service could change the size of the service on demand and take effect without interruptions.

Secondly, as specified in the requirements analysis, the security aspect of the system is important even if it is going to be in a private cloud, that is why the work carried out in this project should consider security as an aspect essential when adapting the system. A practice that was desired to be implemented in this solution was the use of a firewall and an IPS as a point of entry to the system. The pfSense[14] software could be implemented on a dedicated server as an entry point to the system and thus have control of the incoming packages to it. In addition, an IPS software like Snort[15] can be incorporated into the master server, which acts as an intrusion detector

[14] pfSense: https://www.pfsense.org/
[15] Snort: https://www.snort.org/

in the system to avoid suspicious access, thus avoiding data leakage from the master server where the most sensitive data resides.

On the other hand, since the implementation time in MCHE was slight, it was not possible to incorporate the functionality of mapping the IPs of the users to the actual users of the active directory (LDAP) used in this environment. In general, it is necessary to develop a more effective technique to provide personalized data directories for each user along with the generation of passwords and the previously mentioned mapping.

These passwords must be handled with great caution, being encrypted for them to be stored in the system and in addition to encrypting the messages sent to the teachers. Regarding encryption, the web form certificate must be authenticated to be trusted by browsers and to not confuse the teacher.

Finally, the most important point to improve in the system is the adoption of educational applications for containers. During the course of the implementation and evaluation of the results, it has been conceivable to observe the difference of working with virtual machines to work with containers, the containers provide many benefits to this project in order to achieve the objective of minimizing the waiting time since the service has been requested until it is available.

In a case of having to process an unfavourable number of services in the system, with the container applications the processing of them would not be affected and it would be an inexpensive task of time. In contrast, creating virtual machines with their shared hardware and their operating systems makes it a very expensive task, which adds an overhead while this process lasts.

The proposed implementation has been adapted to the resources and time available, as a further example, if this project is integrated only into containers it could be orchestrated with Kubernetes, in order to distribute the service load between servers and to have a higher degree of reliability and availability in the environment.

In this section, the main aspects I desired to have contemplated in the present work have been mentioned, consequently, they have had to be postponed to be developed in the future. However, it should be noted that there are undoubtedly many aspects related to the functionality of the source code that can be developed, that is why I open the door to further contributions to the project, it is available on Github under the GPLv3 license as specified in Appendix B.

References

Acemoglu, Daron, and Pascual Restrepo. “The Race between Man and Machine: Implications of Technology for Growth, Factor Shares, and Employment.” *American Economic Review*, vol. 108, no. 6, 2018, pp. 1488–1542., doi:10.1257/aer.20160696.

Alabbadi, Mohssen M. “Cloud Computing for Education and Learning: Education and Learning as a Service (ELaaS).” *2011 14th International Conference on Interactive Collaborative Learning*, 2011, doi:10.1109/icl.2011.6059655.

Czekierda, Lukasz, et al. “Benefits of Extending Collaborative Educational Cloud with IoT.” *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2017, doi:10.1109/wetice.2017.57.

Deeba, Farah, et al. “Data Transformation of UML Diagram by Using Model Driven Architecture.” *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2018, doi:10.1109/icccbda.2018.8386531.

Dijkstra, Edsger W. A Discipline of Programming. Prentice-Hall, 1976.

Duflos, Sandrine, et al. “A Comparative Study of Policy Specification Languages for Secure Distributed Applications.” *Management Technologies for E-Commerce and E-Business Applications Lecture Notes in Computer Science*, 2002, pp. 157–168., doi:10.1007/3-540-36110-3_16.

Durairaj, M., and A. Manimaran. “A Study on Security Issues in Cloud Based E-Learning.” *Indian Journal of Science and Technology*, vol. 8, no. 8, 2015, p. 757., doi:10.17485/ijst/2015/v8i8/69307.

Eldein, Amar Ibrahim E.sharaf, and Hany H. Ammar. “Model-Driven Architecture for Cloud Applications Development, A Survey.” *International Journal of Computer Applications Technology and Research*, vol. 4, no. 9, 2015, pp. 698–705., doi:10.7753/ijcatr0409.1010.

Ercan, Tuncay. “Effective Use of Cloud Computing in Educational Institutions.” *Procedia - Social and Behavioral Sciences*, vol. 2, no. 2, 2010, pp. 938–942., doi:10.1016/j.sbspro.2010.03.130.

Eriksson, Malin, and V. Hallberg. “[PDF] Comparison between JSON and YAML for Data Serialization.: Semantic Scholar.” *Undefined*, 1 Jan. 1970, www.semanticscholar.org/paper/Comparison-between-JSON-and-YAML-for-Data-Eriksson-Hallberg/636a2b04d98c0af8e9d6f59148352dd63af4f0c1.

Gaur, Ayush, and Manoj Manuja. “Implementation Framework for Cloud Based Education-as-a-Service.” *2014 IEEE International Conference on MOOC, Innovation and Technology in Education (MITE)*, 2014, doi:10.1109/mite.2014.7020241.

Katiyar, Nishant, and Rakesh Bhujade. “A Survey : Adoption of Cloud Computing in Education Sector.” *International Journal of Computer Trends and Technology*, vol. 60, no. 1, 2018, pp. 15–25., doi:10.14445/22312803/ijctt-v60p102.

Masud, Md Anwar Hossain, and Xiaodi Huang. “A Novel Approach for Adopting Cloud-Based E-Learning System.” *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 2012, doi:10.1109/icis.2012.10.

Mell, P M, and T. Grance. “The NIST Definition of Cloud Computing.” 2011, doi:10.6028/nist.sp.800-145.

Miller, Joaquin and Mukerji, Jishnu. *“Model Driven Architecture (MDA) Guide Version 1.0.1”, 2003,* http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

NRE Labs Documentation, docs.nrelabs.io/. Source code: https://github.com/nre-learning/antidote-selfmedicate

N. Zhang, D. Liu and Y. Zhang, "A Research on Cloud Computing Security," 2013 International Conference on Information Technology and Applications, Chengdu, 2013, pp. 370-373, doi: 10.1109/ITA.2013.91.

Pena, Joaquin, et al. “A Model-Driven Architecture Approach for Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems.” *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006, doi:10.1109/dasc.2006.8.

Sabiri, Khadija, et al. “Towards a Cloud Migration Framework.” *2015 Third World Conference on Complex Systems (WCCS)*, 2015, doi:10.1109/icocs.2015.7483315.

Sharma, Ritu, and Manu Sood. “Cloud SaaS: Models and Transformation.” *Advances in Digital Image Processing and Information Technology Communications in Computer and Information Science*, 2011, pp. 305–314., doi:10.1007/978-3-642-24055-3_31.

Sharma, Ritu, and Manu Sood. “Enhancing Cloud SAAS Development with Model Driven Architecture.” *International Journal on Cloud Computing: Services and Architecture*, vol. 1, no. 3, 2011, pp. 89–102., doi:10.5121/ijccsa.2011.1307.

The Littlest JupyterHub *Documentation*, tljh.jupyter.org/. Source Code: https://github.com/jupyterhub/the-littlest-jupyterhub

Vacca, John R. “Computer and Information Security Handbook”. Morgan Kaufmann Publishers, 2017. ISBN 978-0-12-803843-7

Zhang, Qi, et al. “Cloud Computing: State-of-the-Art and Research Challenges.” *Journal of Internet Services and Applications*, vol. 1, no. 1, 2010, pp. 7–18., doi:10.1007/s13174-010-0007-6.

Zhang, Xiuwei, et al. “On-Demand Service-Oriented MDA Approach for SaaS and Enterprise Mashup Application Development.” *2012 International Conference on Cloud and Service Computing*, 2012, doi:10.1109/csc.2012.22.

Zhu, Zemin, and Xiao Fei. “Research and Design of Information System for Basic Education Based on SaaS.” *2009 Second International Conference on Future Information Technology and Management Engineering*, 2009, doi:10.1109/fitme.2009.123.

Zieliński, K., et al. “Recognizing Value of Educational Collaboration between High Schools and Universities Facilitated by Modern ICT.” *Journal of Computer Assisted Learning*, vol. 33, no. 6, 2017, pp. 633–648., doi:10.1111/jcal.12207.

List of Figures

List of Tables

Appendices

Appendix A: User guide

INTRODUCTION

PURPOSE AND SCOPE

The main objective of this manual is to guide teachers through requesting a service in the project implementation in MCHE in order to understand how the web form works and to be clear about the states through which the service passes through until it is ready. This guide clarifies concepts of usability of the web form so as to not cause confusion to the teacher and facilitate the process of requesting the service.

Finally, it is shown how the service is accessed through the web interface with the data sent by mail and also a short manual on how to use a specific service is provided.

OVERVIEW

Without further delay, the structure of the subsequent guide is presented:

1. Accessing the MCHE environment
2. Requesting the service
3. Waiting for the service completion
4. Accessing and using Antidote and Jupyter service
    a. Antidote
    b. Jupyter

WORKFLOWS

To begin with, the main workflows of the user guide will be explained to guide the user in a clear and concise way.

1. Accessing the MCHE environment

To have access to the requested services, you must have a VPN account provided by an MCHE administrator. To do this, if you do not have direct contact with

the administrators, you must contact a teacher leader of the MCHE project who can request the creation of accounts. in the environment. Once access is available, the VPN client is configured and connected to MCHE's internal private network. You should use your conventional web browser and enter the following IP address where the web form is located.

http://172.26.91.18/ or https://172.26.91.18/

At the time of writing this document, if you try to access via HTTPS you will get a screen like the following. Click on 'Advanced' and then click on 'Proceed to ...'.

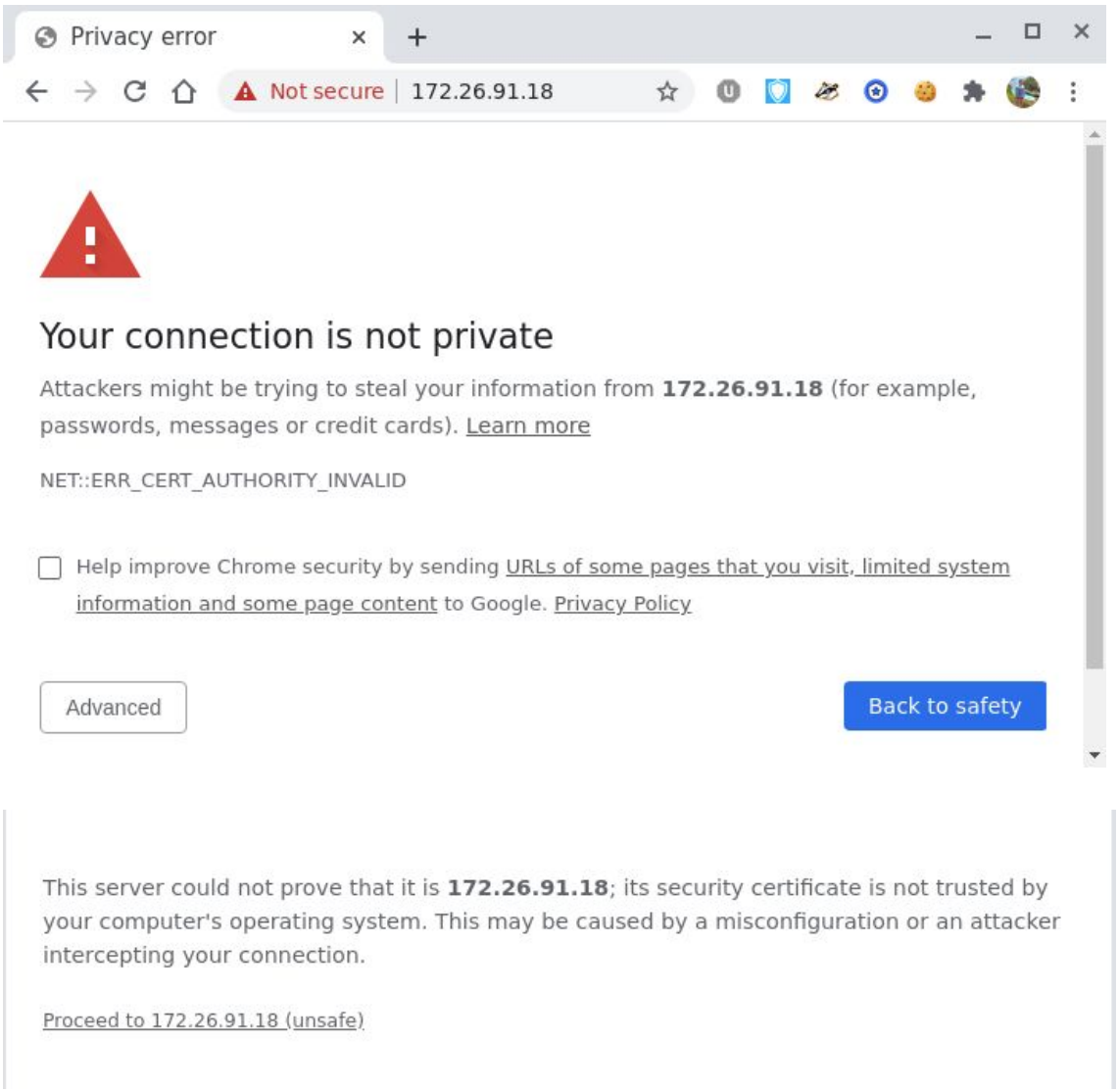


Figure A1. Accessing the web from with HTTPS

This screen is not because the web form is a danger, but rather that the website does not have a certificate validated by a Certification Authority (CA) for the

implementation, so that web browsers, unable to verify its authenticity, show that the website is insecure.

2. *Requesting the service*

Next, the main web page is presented with the welcome to the teacher and the main fields on personal details

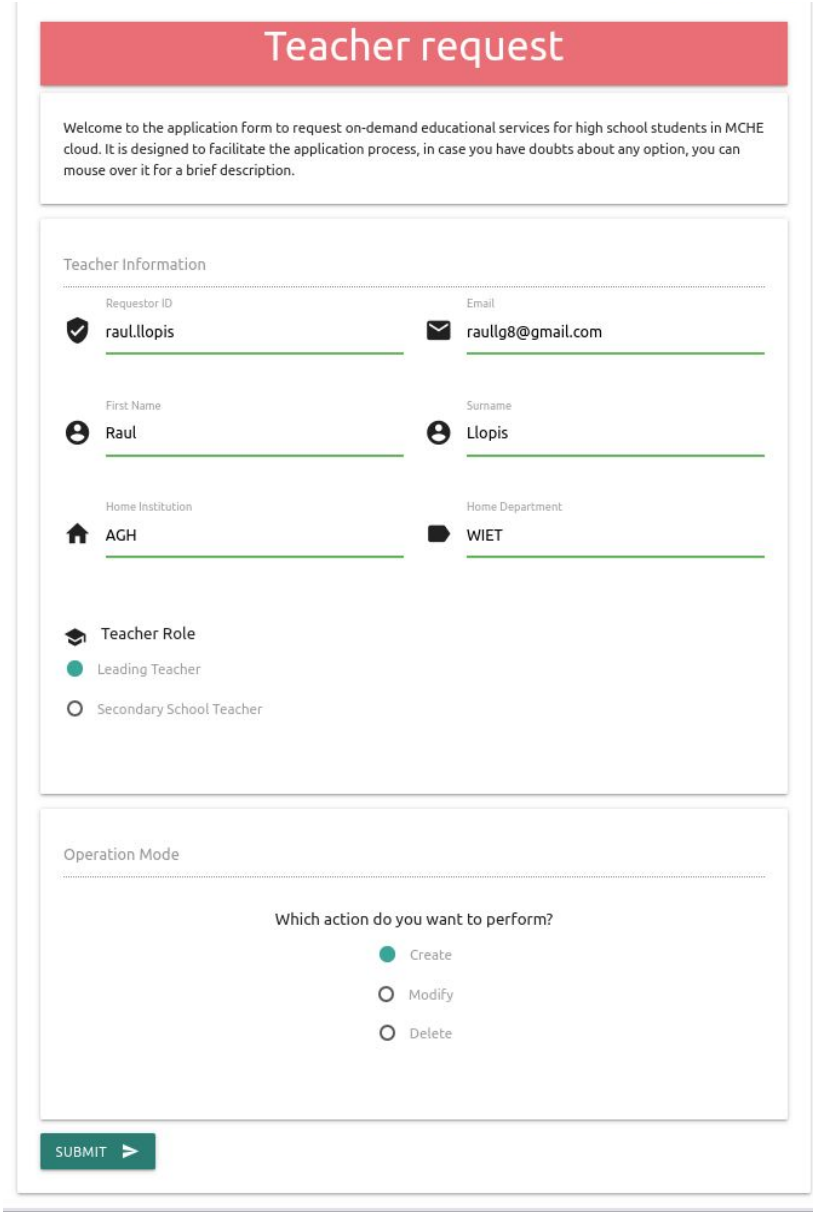


Figure A2. Teacher request web form main page

As you can see, the website has a clean interface and with the fields to be filled in clearly without ambiguity. First, you must enter your user ID in MCHE which must be unique and will serve to identify yourself in the form and consult your services. You

must also provide an email to where the service statuses will be sent during the process. Then enter your name and surname, in addition to the initials of your home institution (E.g. AGH) and your department or faculty (E.g. WIET). Then, you must select what your role is to apply for the service, whether you are a university leader teacher or a secondary school teacher.

At the end, you must enter the mode of operation you want to perform. By default the teacher when you request a service is implicitly creating it, then we select Create in this case. This guide does not go into detail about the other modes, but it should be noted that they do not differ from the functionality presented in the creation mode.

In the case of Modify and Delete, the same interface with the services that the teacher with the identifier has previously requested is presented first.

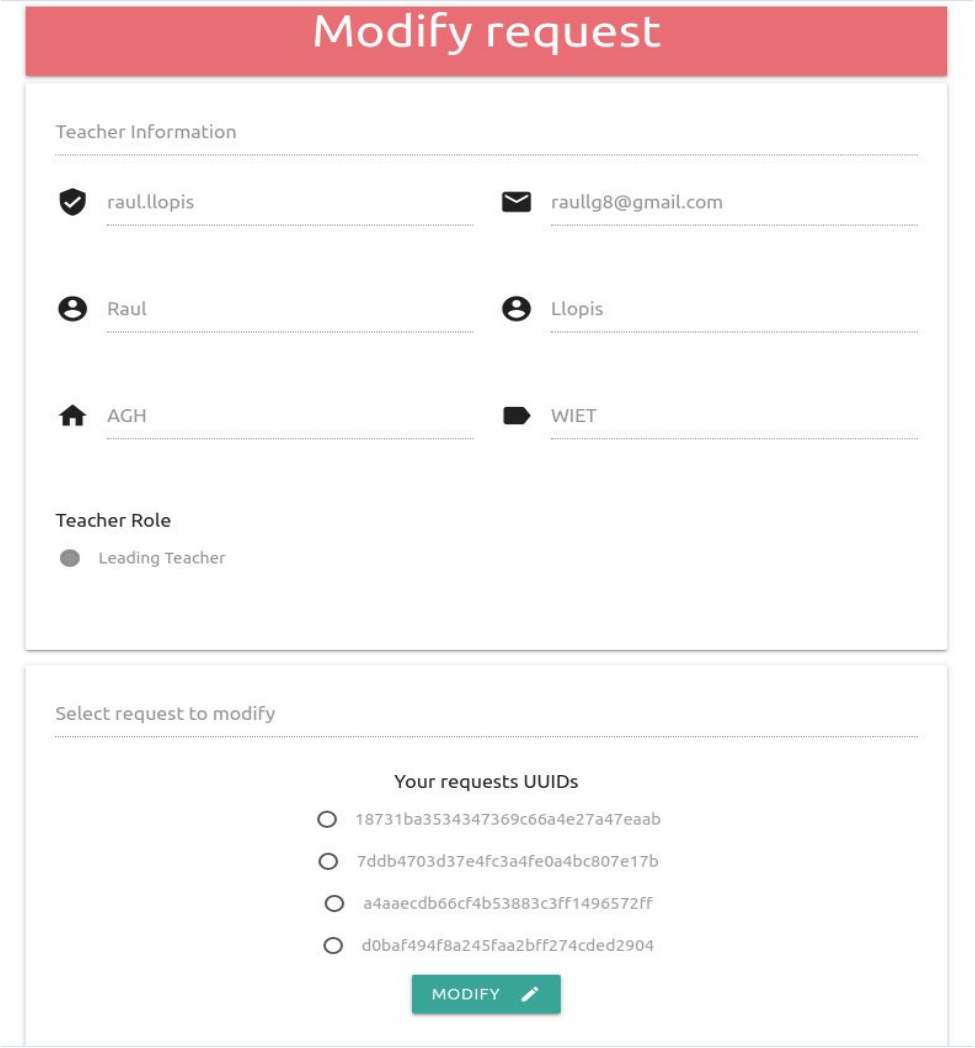


Figure A3. Modify request web form

Once you continue with the Creation mode, the interface that will appear will be like the one in the following image, this is the same as in the Modification mode. The section that the teacher should focus on now is the Service duration where you will specify if you want a duration limited by a date range or indefinite.

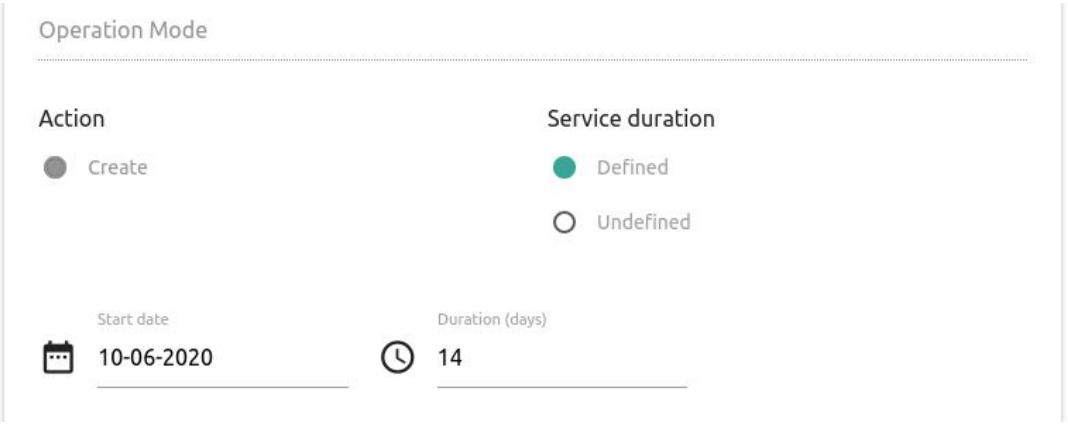


Figure A4. Operation mode web form

Next, the Service detail section is presented, where you begin by selecting the field of study to which the interactive lessons will be related, then the topic field related to the previously selected field will appear on the right. Now the size of the service to be requested is presented, this consideration should be made based on the concurrent students who can use the service and the difficulty of the selected topic.

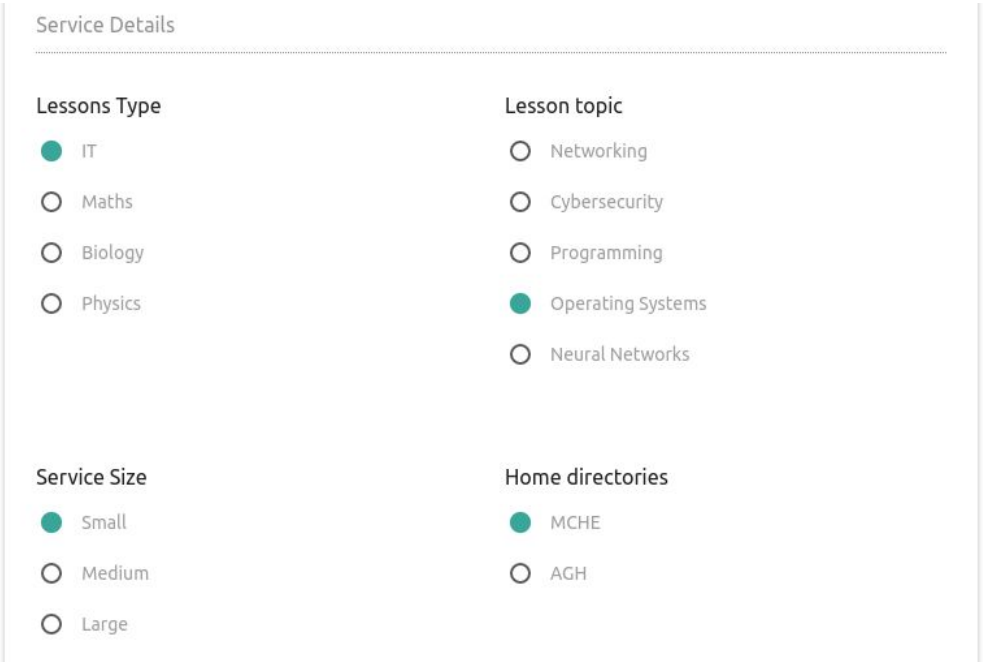


Figure A5. Service details web form

For example, for a basic service on Operating Systems and 20 users with a small size it is more than enough. On the other hand, if the lesson is about Population Genetics it will require more processing so a medium or large size would be ideal for this service. After choosing which is the directory where user data will be stored, it can be on the MCHE platform or at the home institution. By default, it is required to save the data in MCHE for security reasons, these can be consulted by asking an administrator for permission.

Next, the number of users who will use the service is chosen by entering their usernames so that the system generates their default passwords. Then, the mode of cooperation between users is selected, this can be Isolated in case it is preferred that there is no interaction between users as in the case of an examination test. In Groups mode you must select the number of groups and relate which will go in the same group and which will not. In Common mode all users will have a common point of collaboration with what will be a many-to-many relationship.

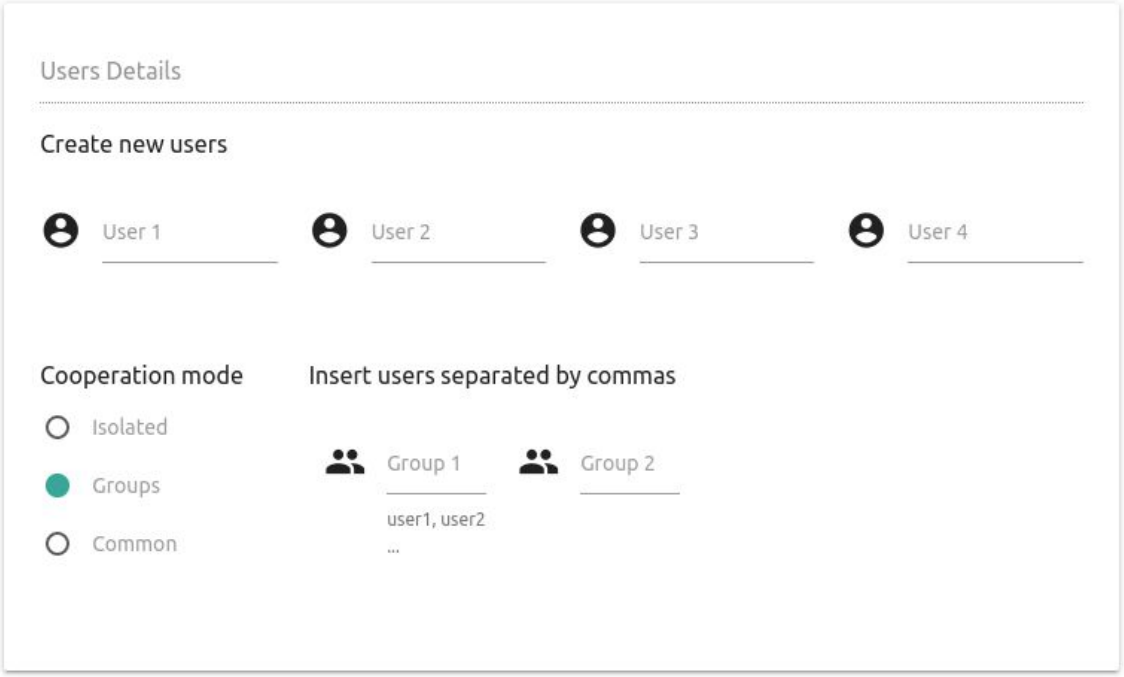


Figure A6. Users Details web form

Finally, the last section on additional details about the service is presented. First, you choose which user data you want to save once the service duration has ended. These can be either the User data or the statistics of the users. Lastly, it is selected what services of access to the service the teacher wants to have, by default the web interface is included. Then, before pressing the Submit button, it is recommended to review all the data entered

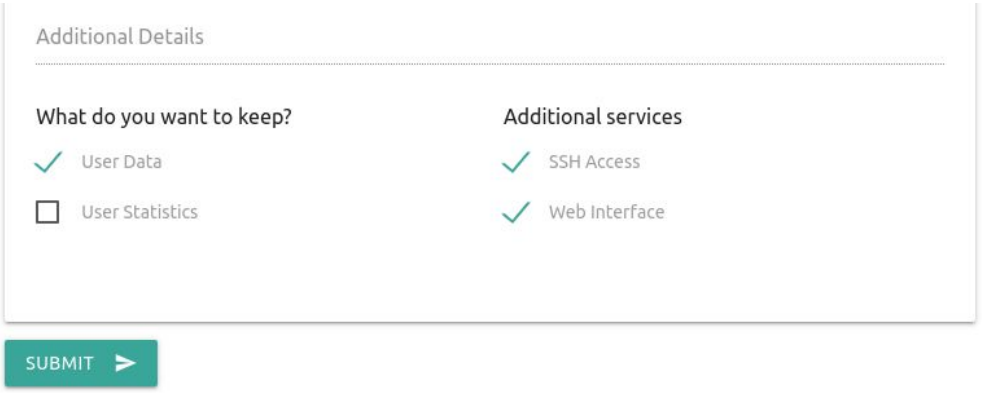


Figure A7. Additional details web from.

Finally, an interface is presented advising the teacher that the service is requested.

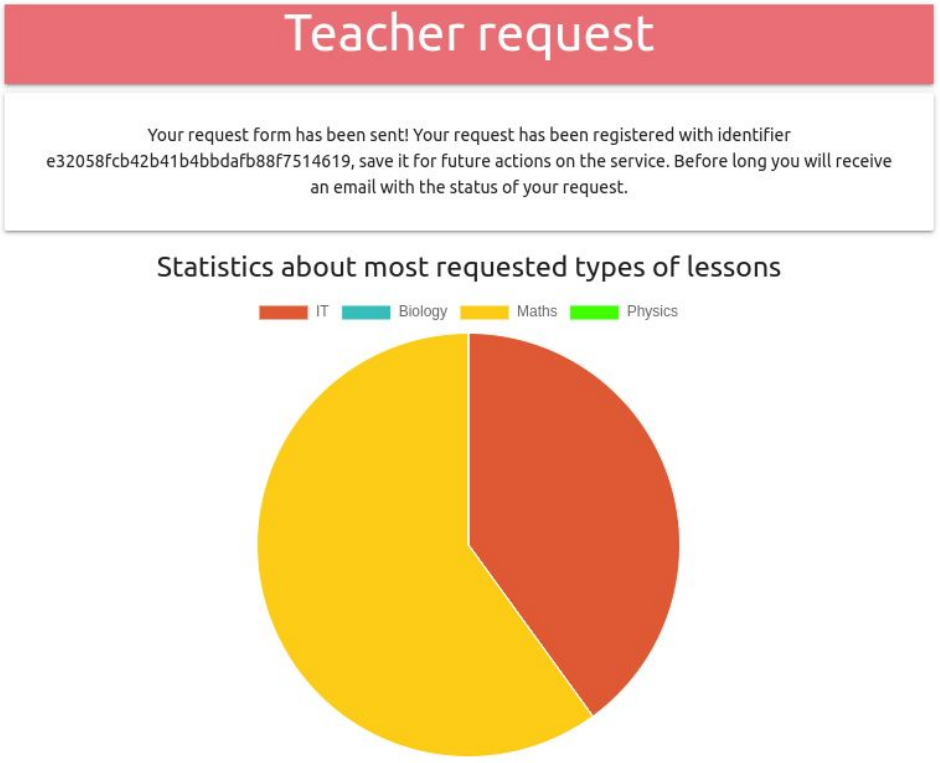


Figure A8. Service successfully requested web from

3. Waiting for the service completion

Once the service is requested, it will not take long to have a message in the inbox of your personal email. This message will indicate that the solution has been processed in the system correctly, that is, policies have been verified and you have already specified which service will be used with the lessons it will have.

The message you will see when the processing of the solution is complete is similar to the following:

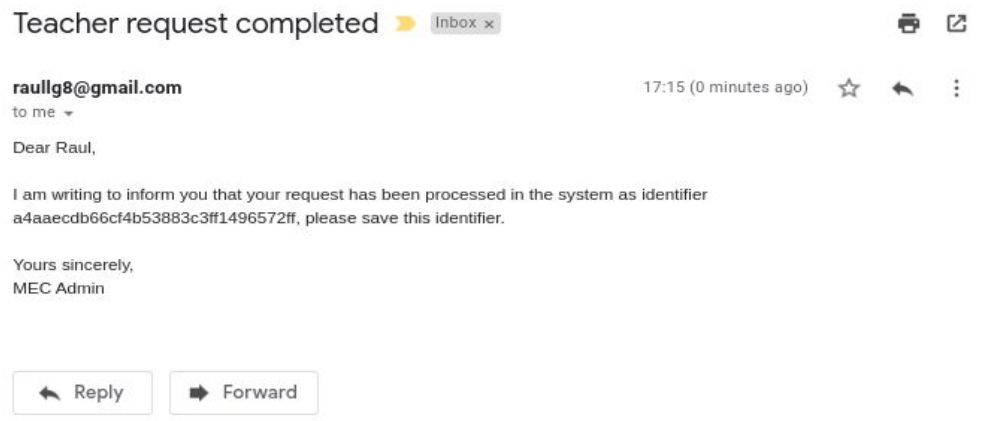


Figure A9. Example mail when the service is processed in the system

In case of having any data that does not comply with the policies of the system, the message will be as follows:

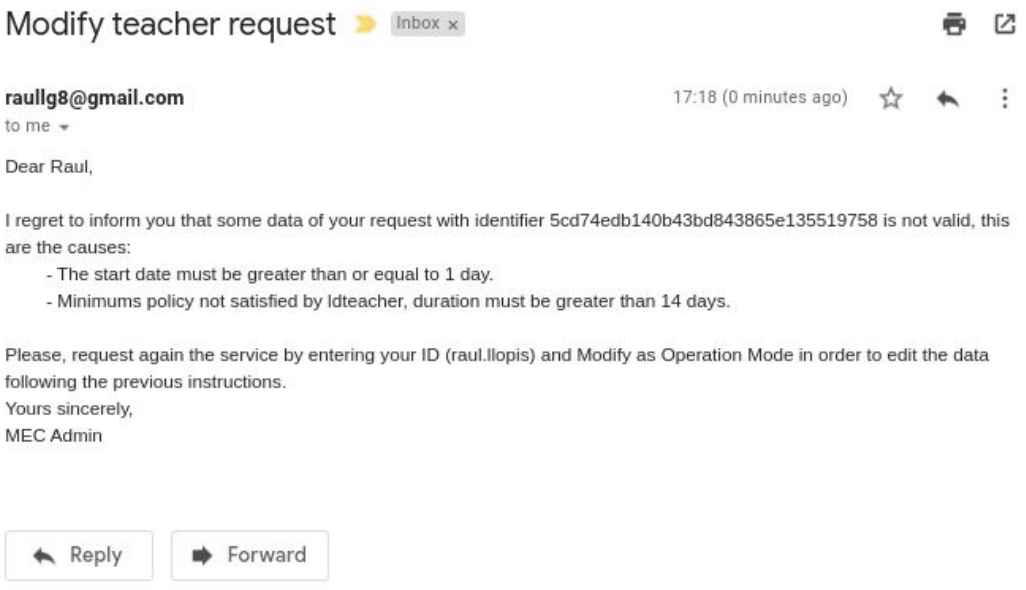


Figure A10. Example mail when the service needs to be modified

So you will have to access the platform again and modify the corresponding data specified by the email received. When the message is successfully completed, the teacher should wait until the start date specified when requesting the service, if you entered this date indefinitely it will be the next day. When the service has been built in the system, a message will be sent informing it with the access data and instructions. An example of a service completed message would be the following:

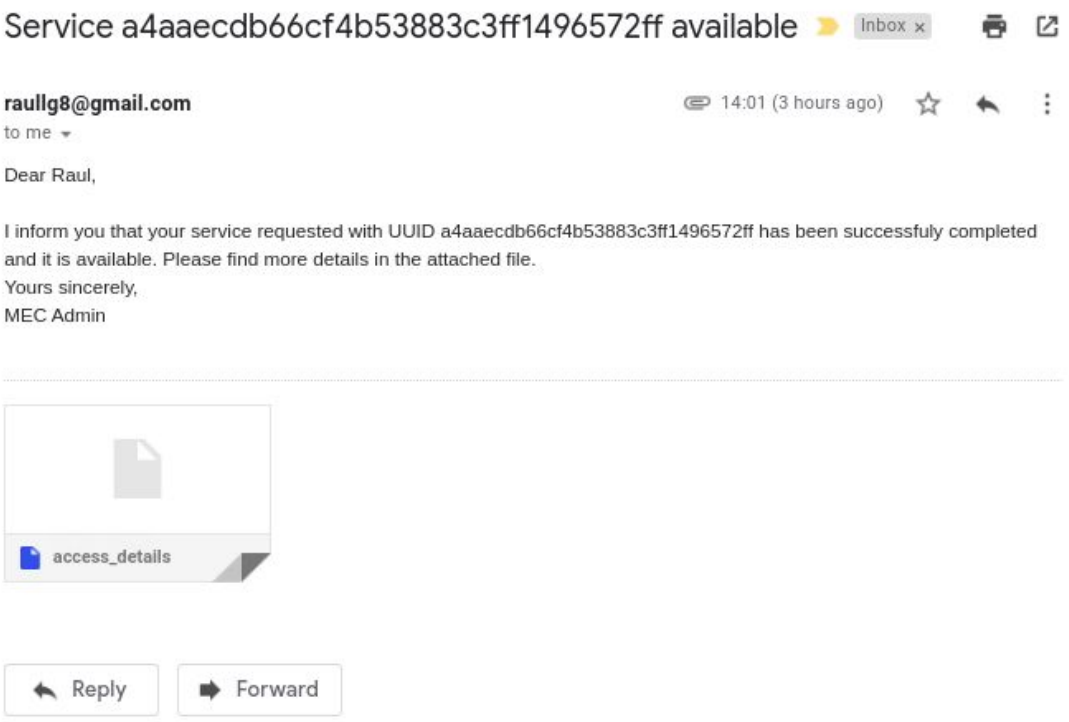


Figure A11. Example mail when the service creation is completed

4. Accessing and using Antidote and Jupyter service

The part corresponding to access and use of a service will depend on the data specified in the email and the instructions. In this section, two particular use cases will be explained, so the data will not be the same in a real situation, but the steps will be similar for both PSM.

In general, first we connect through the VPN configured in step 1 and go to the web browser and enter the address indicated in the previous email.

A. ANTIDOTE

Once accessed, students must choose a lesson either by searching for it by name or in the collection of lessons where they can see all the lessons available in the service.

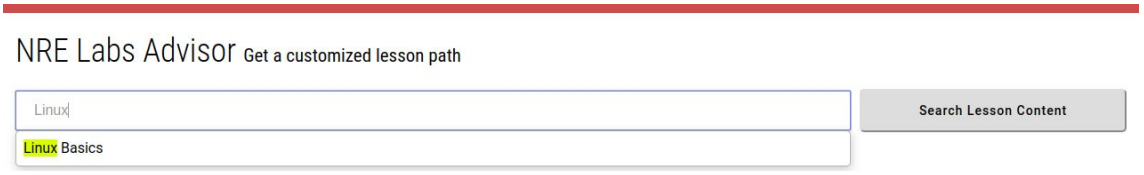


Figure A13. Get a customized lesson path based on the lesson to achieve

Once a lesson has been selected, the main interface of Antidote is as follows, with on the one hand on the left the content of the lessons is located and on the right is the terminal where you can interact as the lessons progress.

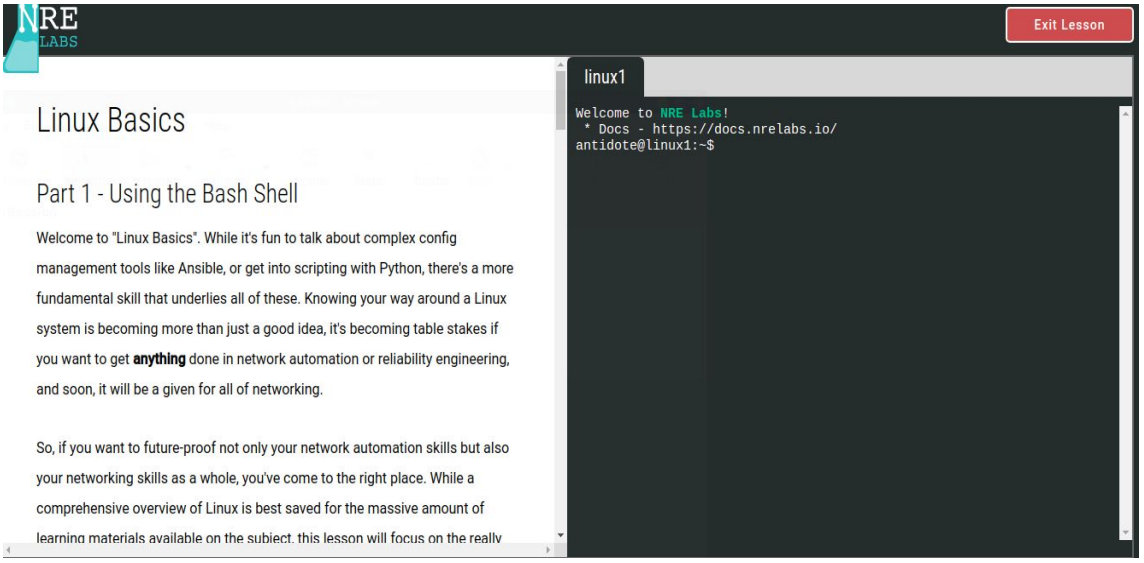


Figure A14. Antidote PSM lesson web interface

B. JUPYTER

In the case of Jupyter, once connected to the MCHE VPN and accessed the URL provided in the email, it will be presented with an initial screen like the following figura, where you must enter the credentials provided to authenticate into the Jupyterhub server.

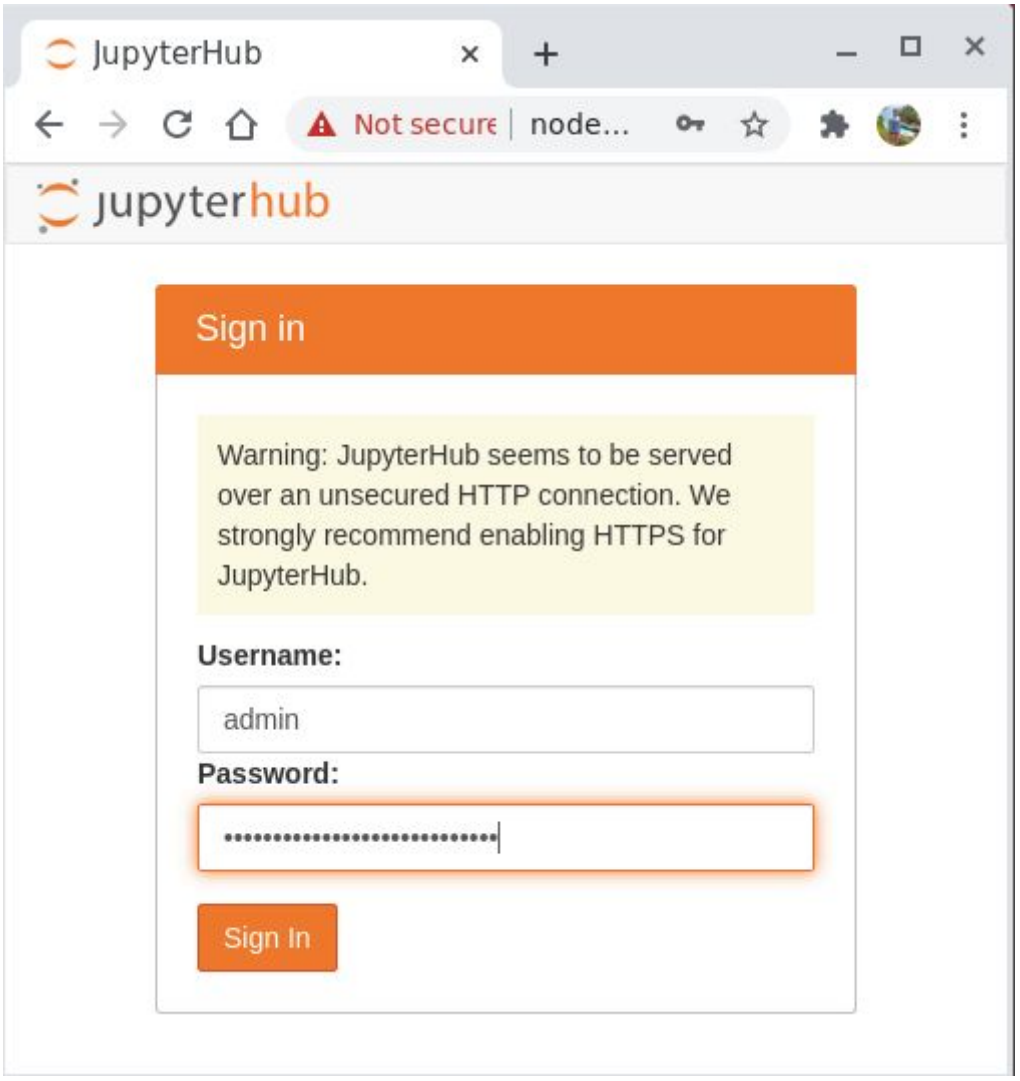


Figure A15. Jupyterhub login

By default, when you log in the first time, the server automatically starts up, although it will always be under the teacher's control in the Control Panel button at the top right, where you can either stop or start the server when needed.

Then, by default, the main directory will be empty. To copy the requested lessons, follow these steps:

- First, click on the ‘New’ button and select ‘Terminal’ at the bottom.

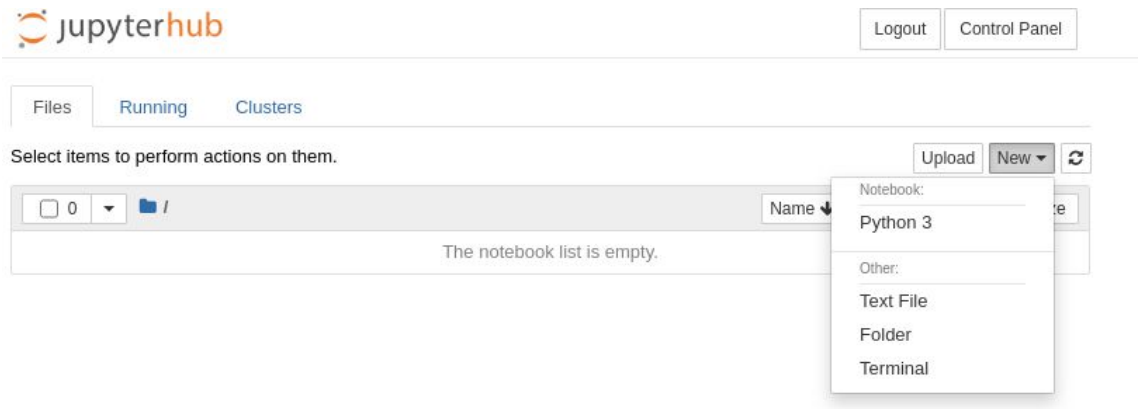


Figure A16. Jupyterhub accessing terminal

Then a black screen will appear, this is the terminal. With it you can perform the same or more actions on the server than with the graphic interface. What we are going to do is copy the lessons from a shared folder to our personal folder.

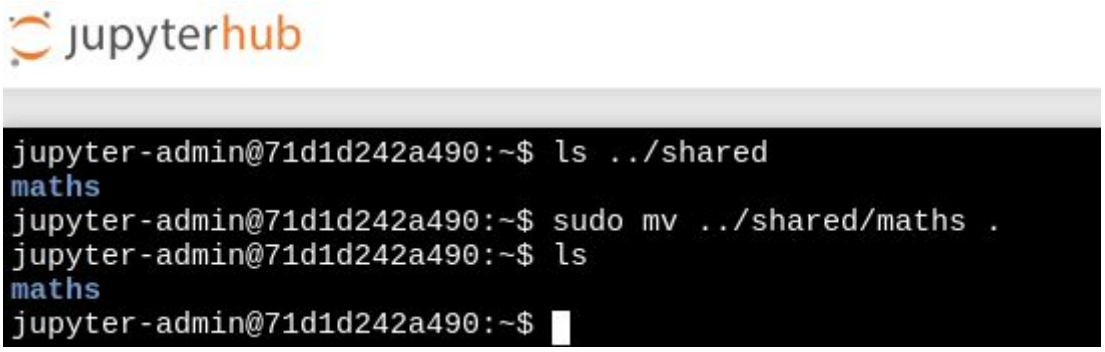


Figure A17. Jupyterhub moving lessons in terminal

The first command returns as output 'maths', in each case it will be different depending on the type of lesson requested. In the second command, you should replace 'maths' with what was in the first command. For example: in the first command it says 'it', then I have to execute this as a second command:

```
sudo mv ../shared/it .
```

 # It is very important to include the last dot in the command.

Finally click on 'Control Panel' and 'My Server', now you will see the lessons.

Figure A18. Jupyterhub example lessons in workspace

Then go into the folder and you will see the available lessons, now click in one lesson and you should get a view like the one shown in the following figure. From now on, the materials can be distributed to the students.

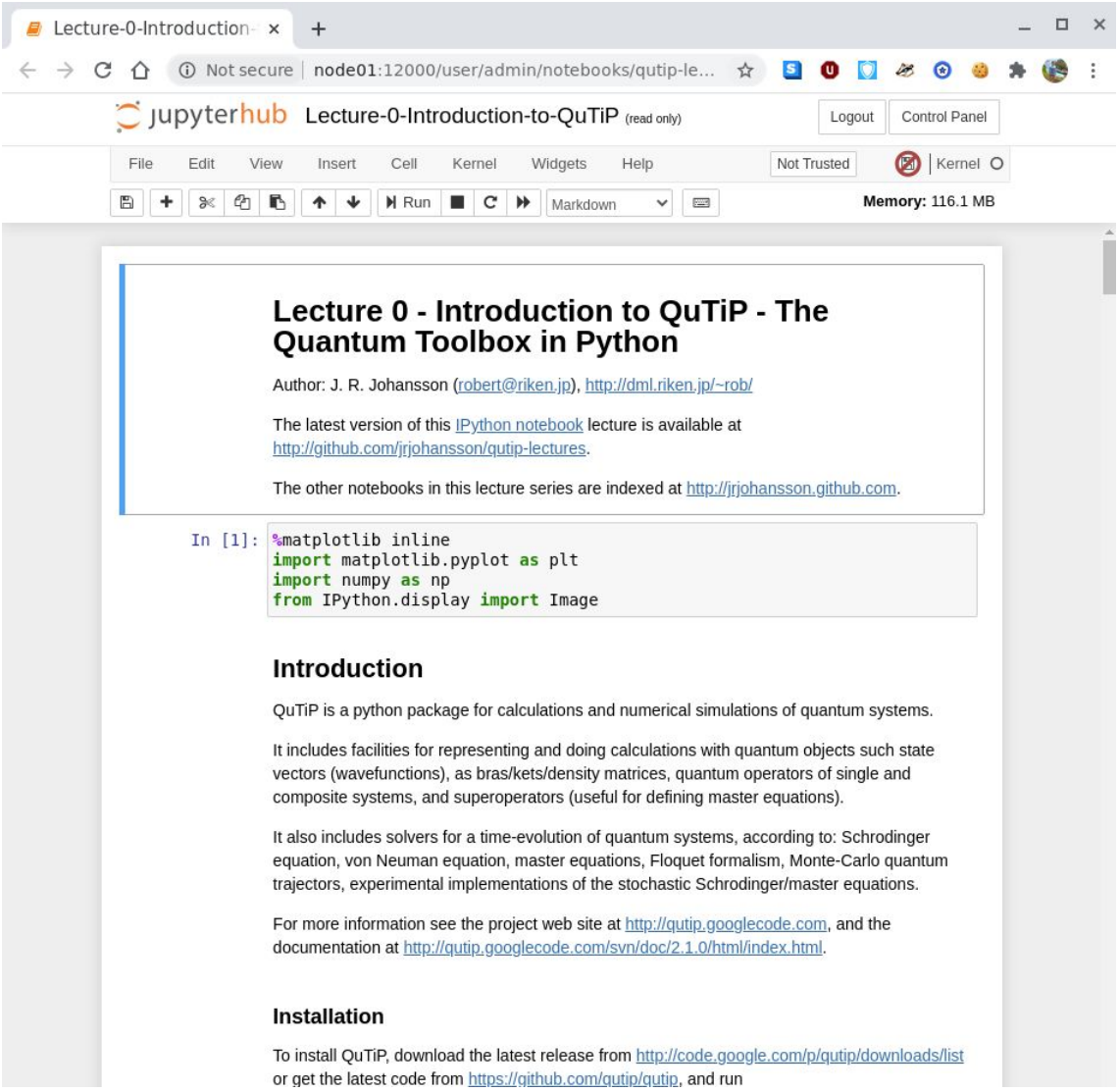


Figure A19. Jupyterhub example real lesson

Appendix B: Technical documentation

INTRODUCTION

Welcome to the technical documentation guide of the cloud orchestration project source code. This document will guide you through how to maintain and develop the system source code along with technical aspects of important pieces of the system developed.

To start with, it is not necessary to have an environment like the one used in the implementation. In the case of arranging it by yourself you can see Appendix C: SETUP GUIDE for the complete steps to configure the master and worker nodes with its dependencies, so the steps in section 'SIMULATE THE ENVIRONMENT' will not be necessary.

In general, the first step to take is to clone the source code repository in the $HOME directory of the environment where the project will be developed, in the designated master node.

```
cd ~
git clone https://github.com/llopisga/cloud-orchestration.git
```

It is highly recommended to be cloned in this location, in case you have to clone it in another directory for personal reasons, you must edit the main path in the configuration file that will be seen later.

OVERVIEW

The technical documentation guide has the following content structure:

1. Simulate the environment
2. Source Code
    a. Model Orchestration
    b. Platform Orchestration
3. Provision Deploy
    a. Antidote Selfmedicate
    b. The Littlest Jupyterhub

1. SIMULATE THE ENVIRONMENT

In order to simulate the environment, to simulate the web form requests you must modify a base request that is included in the application source code.

```
vi
/home/$USER/cloud-orchestration/TM/thesisraul/public_html/requests/T
Mrequest-a4aaecdb66cf4b53883c3ff1496572ff.yaml
```

The request has the YAML format presented in the next piece of code, which must be manually modified to perform tests on different requests that may be generated in the real web form as previously mentioned.

```
---
requestor_id: raul.llopis
timestamp_start: 01-05-2020 22:49:01
firstname: Raul
surname: Llopis
request_UUID: a4aaecdb66cf4b53883c3ff1496572ff
requestor_email: raullg8@gmail.com
home_institute: EPSA
home_department: DCOM
teacher_role: ldteacher
operation_mode:
  create:
    Undefined: true
machine_size: small
lesson_type: it
lesson_topic: OperatingSystems
home_directories: mche
users:
  user1: Lucia
  user2: Anna
  user3: Sebastian
  user4: John
user_cooperation_mode:
  isolated:
additional_services:
  - web: "true"
persisted_items:
  what:
  - user_data
```

Next, we must go to the configuration file that is loaded in the Python files each time it is orchestrated, in this file we observe the following fields.

```
vi /home/$USER/cloud-orchestration/code/conf/config.yaml
  PROJECT_PATH: "/home/USERNAME/cloud-orchestration"
  TM_PATH: "/var/www/thesisraul/public_html/requests"
  SMTP_SERVER: "smtp.gmail.com"
  SMTP_PORT: 587
  NOTIFY_ADMIN: "raullg8@gmail.com"
```

There is only a variable that is not modified is the PROJECT_PATH, unless you clone the repository in another location The USERNAME variable is replaced by the user who executes the code automatically at runtime, so it must not be replaced manually. Regarding the TM_PATH variable, this is where the requests folder is located, in this case we must replace the default string with the following, changing the home user with yours:

```
TM_PATH:
"/home/raulloga/cloud-orchestration/TM/thesisraul/public_html/reques
ts"
```

For the other variables, the SMTP server is specified with its port, in this case the one provided by gmail has been used. In addition, the mail of an administrator who will be notified in case of errors in the execution of the code or on the system must be placed in the 'NOTIFY_ADMIN' value field.

Then, you must adapt the number of nodes you have in your system, in the case of working on the same machine you must go to repositories/inventory/nodes and leave only one node with the data of your system. The most important thing here is the IP address which should be changed by the IP loopback, typically 127.0.0.1, of your machine or the static IP.

```
node01:
  network:
    fqdn: node01.mche.edu.pl
    domain: mche.edu.pl
    ip: 127.0.0.1
```

Also, you must add this line in /etc/hosts:in order to set a local DNS for the nodes.

```
127.0.0.1    node01
```

Regarding the execution of the code, once these steps have been performed, you must change the current directory to the folder where the source code resides, from which you will execute the following command.

```
cd ~/cloud-orchestrator/code
python3 orchestrator.py <UUID>
```

The code ‘orchestrator.py’ is called with Python 3, passing as the argument the UUID of the service that we are going to process, this <UUID> must be changed by the one you want to process every time.

After generating the number of requests you desire, the following command should be entered from the directory code/, and the service orchestrator will start and complete if there is any service generated from the other model.

```
python3 service.py
```

2. SOURCE CODE

A. MODEL ORCHESTRATION

I. orchestrator.py

Next, I proceed to explain what are the most important points to understand about the source code. The main starting point is the orchestrator.py file, which contains the three models that are processed in each request. First, the variables that will be passed to the PROJECT_PATH and TM_PATH models are introduced, they are loaded from the configuration YAML file and the UUID is gathered from the first argument passed to the program.

Next, the variables are defined regarding the SMTP service authentication to send the errors that may happen during the execution, for this a try-catch is implemented in the main method which in throwing an exception will be received by the 'logging' module and this will send it by calling the errorEmail class in the file emailClient.py.

For each file, a main class has been defined in which the code functions are called in the desired order and the steps that the code takes can be logged. These logs are stored in *~/cloud-orchestration/logs/* with the service's UUID, the format of the logs is as follows.

```
2020-05-26 22:52:50,064 - pim - Generating PIM Model
```

Next, each of the main classes belonging to the modules pim.py, feasibles.py and psm.py are executed in the same order.

II. pim.py

In this model, dictionary type variables are defined to save the loaded data from YAML files. A constructor is defined in the conversionPIM class which includes the variables passed from the main code.

```
TM_DATA = dict()            # Handles TM request data in a dictionary
PIM_DATA = dict()           # Handles all PIM data
CONVERSION_DATA = dict()    # Handles conversion policy data

class conversionPIM:
    def __init__(self, UUID, TM_PATH, PROJECT_PATH):
        self.UUID = UUID
        self.TM_PATH = TM_PATH
```

```
        self.PROJECT_PATH = PROJECT_PATH
```

Next, the 'openTM' function is responsible for storing the most important data for program execution in variables. Then, the function 'openConversion' checks in the templates repository the templates that start with the pattern 'PIM_' and for each of them it is opened in reading mode and the elements that are later saved in a variable are parsed.

```
listFiles = [f for f in os.listdir(templatesPath) if
f.startswith(pattern)]
for pim in listFiles:
    with open(templatesPath + pim, 'r') as conversionfile:
        [CONVERSION_DATA.update({key: value}) for key, value in
yaml.full_load(conversionfile).items()]
```

Then, the 'generatePIM' function is in charge of checking if the mode of operation is 'delete' or different, in the case of being deleted, simply change the mode of operation to delete. In the creation or modification process, the size of the TM machine is first converted to values understandable by the system. Then, set the privileges based on the type of teacher who has requested the service. And finally, the network and firewall data is loaded. All of this data is dumped along with the TM data to a PIM file.

```
for key in CONVERSION_DATA:
    for x in CONVERSION_DATA[key]:
        if (x == lessontype):
            systemDict = dict(CONVERSION_DATA[key][x][machinesize])
        elif (x == cooperationmode):
            privilegesDict = dict(CONVERSION_DATA[key][x][teacher])
```

III. feasibles.py

In this model the PSMs and nodes feasible for the system are generated, this program follows the same role as 'pim.py'. Global variables are defined first and then the class with its constructor. In the main main method, the process logger is defined and the condition to carry out different actions in case the service is eliminated.

In general, the previously generated PIM model is first loaded with the most important data stored in the corresponding variables. Then, in the case of service creation mode, the PSM inventory is processed first.

It is consulted in the repository for each inventory that eimpice by the pattern 'PSM_' is loaded individually. PSMs are selected that have lessons based on the type and topic of the lesson that has been taken. All the PSMs that meet this condition are stored in an array that is then processed to gather the URL of the lesson and the requirements of the virtual machine of the specific platform.

```
for psm in listFiles:
    with open(inventoryPath + psm, 'r') as conversionfile:
        convert = yaml.full_load(conversionfile)
        for key, value in convert.items():
            INVENTORY_PSM_DATA.update({key: value})
            for k in value:
                for x in value[k]:
                    if (x == lessontype):
                        for y in value[k][x]:
                            if (y == lessontopic):
              # Selects feasibles PSM based on Lesson Type and Topic
                                feasibles.append(key)
```

Next, the repository of Feasible nodes is consulted in which, following the file pattern 'node_', the specifications of each node are checked. For each specification that meets the requirements, one is added to the counter, at the end the values are saved in a node and result dictionary. Nodes that have more than four conditions fulfilled are added as Feasible nodes to the final dictionary.

```
for node in listFiles:
    with open(inventoryPath + node, 'r') as conversionfile:
        convert = yaml.full_load(conversionfile)
        for key, value in convert.items():
            INVENTORY_NODES_DATA.update({key: value})
            if (value['specs']['vcpu'] > vcpu):
                counter += 1
            if (value['specs']['ram'] > ram):
                counter += 1
            if (value['specs']['os'] == opsys):
                counter += 1
            if (value['specs']['vt'] == vt):
                counter += 1
```

```
            if (value['specs']['storage'] > storage):
                counter += 1
            matching.update({key: counter})
feasiblesNodes = {}
for i in matching:
    if (matching[i] >= 4):
        feasiblesNodes.update({i: INVENTORY_NODES_DATA[i]})
```

Finally, the PIM data is dumped along with the PSM and Feasible Node data.

IV. psm.py

This model is where the application of policies on the data processed so far happens. In this model, it is necessary to notify the teacher of the status of the requested service, so the SMTP server data is loaded in variables. As in the previous models, the class and its constructor are defined, then the previously generated feasible PSM model is opened and the most important data is saved.

Next, the three main policies of the system are loaded by processing their content and saving it in dictionary variables for more efficient handling.

First, the screening policy is applied by checking whether the defined policies are followed. It is verified that the defined start date is greater than or equal to the current one by means of a function 'time_difference' which calculates the difference in days and compares it to the value specified in the policy. In the case of being indefinite, it is established that the start date will be one day from the moment it is executed.

```
for key in SCREENING_DATA:
    if (key == 'min-time-before-start'):
        try:
            if (duration != 'undefined'):
                if (self.time_difference(start_date) <
SCREENING_DATA[key]):
                    NOTIFY_USER.append(
                        'The start date must be greater than or
equal to ' + str(SCREENING_DATA[key]) + ' day.')
            else:
                tmp = datetime.now() + \
                    timedelta(days=SCREENING_DATA[key])
                start_date = tmp.strftime('%d-%m-%Y')
        except:
            pass
```

Then the minimums and maximums of the policy are checked. These fields are the number of users, the number of groups and the duration established in the request. IF this is indefinite, the minimum established in the policy is defined and in the same way we proceed with the maximums. At the end, the duration of the service is calculated using a random number of days between the minimum and maximum value.

If any policy has not been complied with, the string will be saved with the corresponding error in an array, the channel will be checked later if it is empty or contains any non-compliance.

Next, the selection policy is applied in which the feasible PSM is checked first. The policy specifies for each PSM an array of sizes and types of lessons they accept, these data are related to the properties of the particular PSM. For example, in this implementation Antidote is oriented to IT lessons and Jupyter can instead be oriented to any of the topics that have been specified.

```
---
psm:
  antidote:
    size: [small, medium]
    type: [it]
  jupyter:
    size: [medium, large]
    type: [it, biology, maths, physics]
```

For this reason, the code includes for each PSM the size and type of lesson specified with those specified in the policy. These are stored in a dictionary with PSM key and value the number of matches. For all the PSM, it is chosen which is the PSM that obtains more coincidences with the policy and finally one of them is chosen.

```
for psm in feasiblesPSM:
    for key in SELECTION_DATA:
        if (key == 'psm'):
            for rule in SELECTION_DATA[key]:
                if (rule == psm):
                    for x in SELECTION_DATA[key][rule]:
                        for i in SELECTION_DATA[key][rule][x]:
                            if i in PIMarray:
                                counter += 1
    matchPSM.update({psm: counter})
```

```
    counter = 0
bestPSM = max(matchPSM, key=matchPSM.get)
#print('Selected PSM is ' + bestPSM)
```

In the case of the nodes, a more dynamic policy pattern is followed, for each node in the generated feasibles, two dynamic parameters are checked using a 'nodesHealth' function: CPU Usage and RAM Available. For each node these two parameters are saved and returned as a result. Next, the node is saved to a dictionary as a key and the RAM value in MB available. At the end the node with the most available RAM memory is selected.

```
for node in feasiblesNodes:
    nodeDict = self.nodesHealth(node)
    if node in nodeDict:
        print("Node " + node + " is down.")
    else:
        matchNode.update({node: nodeDict['MemAvailable']})

bestNode = max(matchNode, key=matchNode.get)
#print('Selected Node is ' + bestNode)
```

Next, the validation of the data of the Screening policy is applied. As previously mentioned, if there is a warning in the 'NOTIFY_USER' array, they are added to the body of the email to inform the user and are sent to the teacher to warn them that they must modify the request and the program will end its execution. In the event that it complies with the policies and the operating method is not ‘delete’, a message will be sent notifying that the service has been successfully registered in the system and is waiting to be built. In the case of the service with operation mode delete, the corresponding mail will be sent, proceeding to generate the following preempt.

The sending of the mail is done through the ‘emailClient’ module, calling the senEmail class which acts as an interface where the SMTP server values and the data to send are passed.

Next, the Schedule policy is applied, which is based on the preempt priority theory. First, all the preempted files that currently exist in the system corresponding to the active services or marked to be deleted are loaded. The priority of each one is gathered and it is checked which one has the highest priority of the set in order to

assign an even higher priority than this one. When I refer to high, it will be processed later than all the previous ones due to the priority queue.

```
pattern = 'preempt-'
listFiles = [f for f in os.listdir(
    schedulePath) if f.startswith(pattern)]
priorities = []
try:
    for p in listFiles:
        with open(schedulePath + p, 'r') as conversionfile:
            convert = yaml.full_load(conversionfile)
            for key, value in convert.items():
                for x in value:
                    if (x == 'priority'):
                        priorities.append(value[x])
    priority = max(priorities) + 1
except:
    priority = 0
```

Then the preempt corresponding to the current service is created, with the following data template.

```
preempt:
  arrival_time: DD-MM-YYYY HH:MM:SS
  destination: nodeXX
  priority: Y
  start_date: DD-MM-YYYY
  state: ready
  uuid: UUID
```

To end the process, the feasibles are updated to those selected in the main dictionary and the new YAML file is dumped with the final PSM data.

B. PLATFORM ORCHESTRATION

On the other hand, the code corresponding to the platform orchestrator is presented, which has a main class in charge of providing the logic for N Python programs corresponding to the N PSMs that are in the system. These are located in / code / PSM / and follow the template that we will see later.

I. service.py

To run this program in a production environment, a Crontab could be configured to run every day at the specified time. This program is designed to run once and the creation of all platforms are orchestrated in the same execution.

First, all variables are defined with the values loaded from the configuration file on the data of the SMTP server to which errors will be sent, if any, exactly as in the main orchestrator. Then the main directories where the preempts, the final PSM requests and the Python programs corresponding to the PSMs are defined. When the main function of the code starts to execute, it makes a call to the function 'checkDuration()' which is mainly in charge of creating preempt files for services that are about to end.

For each PSM in the production requests directory, the duration and the start date specified at the time are extracted. The calculations are made to check if the current time is the same as the start date plus the duration, in which case the same PSM file modifies the operation mode field and is set to delete. In addition, the preempt corresponding to this service is generated with the same logic as in a creation process to determine the corresponding priority.

Then, when all the active services have been verified, each preempt that exists at that moment is added to the priority queue. In the priority queue two values are added, the UUID of the service along with its priority.

```
pattern = 'preempt-'
listFiles = [f for f in os.listdir(
    PREEMPT_PATH) if f.startswith(pattern)]
q = PriorityQueue()
for p in listFiles:
    with open(PREEMPT_PATH + p, 'r') as conversionfile:
        convert = yaml.full_load(conversionfile)
        for key, value in convert.items():
            for x in value:
                if (x == 'start_date'):
                    start = datetime.strptime(value[x], '%d-%m-%Y')
                    now = datetime.now()
                    diff = start - now
                elif (x == 'priority'):
                    priority = value[x]
                elif (x == 'uuid'):
                    UUID = value[x]
```

```
    try:
        if (diff.days < 1):
            q.put((priority, UUID))
    except:
        pass
```

When all the preempts have been placed in the priority queue, the main program loop is executed, which while the queue is not empty, the priority queue is emptied based on the priority. For each one, open your PSM file and gather the values that are necessary to create the service, these are: Operation Mode, PSM Name, Virtualization Technology and Destination Node IP.

Next, a very important function is used for the logic of this program, it is presented below.

```
# Importing PSM Python file as a module
psm = importlib.import_module(PSM_NAME, package=None)
```

For the specific PSM its code is imported as a module, which facilitates the execution of the program because each one simply loads the module it needs. Otherwise, all modules should be loaded once and mapped, so this option is very handy for this solution.

Once the code is imported, depending on the mode of operation, its creation or destruction will be executed. For the creation two classes of the code are executed, these are code and deploy. When processed, in the end the preempt is destroyed and more PSM continues to be processed until it ends.

```
try:
    if (op_mode != 'delete'):
        logger.warning('Executing PSM code module of ' + PSM_NAME)
        process1 = psm.code(UUID, PROJECT_PATH,
                            PSM_NAME, VIRT_TECH, PSM_DATA)
        process1.main()
        logger.warning('Code modified and ready to be deployed')
        logger.warning(
            'Deploying with specific virtualization technology')
        process2 = psm.deploy(UUID, PROJECT_PATH, DEST_NODE,
PSM_DATA)
        process2.main()
```

```
        logger.warning('Finished deploying')
    else:
        deleteps = psm.destroy(UUID, PROJECT_PATH, DEST_NODE,
PSM_DATA)
        deleteps.main()
except Exception:
    logger.exception('Unhandled Exception')
```

Once the processing of all the PSMs is finished, it is ready to reorder the priority of all the preempts that remain to be executed in the system. First, the values of the current preempts are loaded into a dictionary variable which will contain the priority and the service's UUID.

```
try:
    # Re-arrange current preempt files if there are gaps between
priorities
    gaps = [ele for ele in range(
        max(priorities)+1) if ele not in priorities]
    # If there are gaps between the priorities numbers
    if (gaps):
        priorities.sort()
        arranged = {}
        tmp = list(range(len(priorities)))
        for x in priorities: # Rearrange the priorities
            new = tmp[0]
            del tmp[0]
            arranged.update({new: data[x]})
    [...] # Change priority to its file
except:
    pass
```

II. PSM template code

The same defined structure is followed for each PSM template. There must be three classes: code, deploy and destroy. Each one will be called in service.py depending on the mode of operation. There will be a variable to save the PSM data and another to save the data that is needed to configure the code and the service.

2.1. Code

The code generation class follows an almost identical structure for all PSMs. First, the function 'setVariables ()' is executed, which obtains the important data for the

service and saves it in the dictionary with the string that will be replaced in the file and the value to be replaced.

```
ANTIDOTE_DATA.update(
    {'ANTIDOTE_VMNAME': self.PSM_DATA['request_UUID']})
ANTIDOTE_DATA.update(
    {'ANTIDOTE_MEMORY':
self.PSM_DATA['service'][self.PSM_NAME][self.VIRT_TECH]['memory']})
[ . . . ]
```

Then, the function 'checkAvailablePorts ()' is executed which is based on the range of ports specified in PIM with which it will be verified by the socket module that ports are free and not used to be assigned.

```
a_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
port_range = 1000
for x in range(startp, stopp):
    location = (dstServer, x)
    result_of_check = a_socket.connect_ex(location)
    if not result_of_check == 0:
        ANTIDOTE_DATA.update({'ANTIDOTE_PORT': x})
        break
```

Next, the 'copyPSMcode()' function is executed, which copies the source code found in ~/cloud-orchestration/PSM/repositories/<PSM>, if there is any lesson to incorporate into the PSM it will be downloaded using 'git clone' and all the source code will be copied to the following location ~/cloud-orchestration/production/deploy/<PSM>.

Finally, the function 'replaceVariables()' is executed which replaces the objective data of the PSM in the source code, for this the files to be modified are specified in an array and these are opened in writing mode to replace the necessary data.

```
ANTIDOTE_DIR = str(DEPLOY_PATH) + 'antidote-selfmedicate/'
arrayOfFiles = ['antidote-config.yml', 'selfmedicate.sh']
        for file in arrayOfFiles:
            fin = open(ANTIDOTE_DIR + file, "rt")
            data = fin.read()
            for i, j in ANTIDOTE_DATA.items():
                data = data.replace(i, str(j))
                fin.close()
                fin = open(ANTIDOTE_DIR + file, "wt")
                fin.write(data)
            fin.close()
```

2.2. Deploy

The service deployment class aims to execute the code previously generated on the destination server. To do this, the copyFile () function is called first, which does exactly what the function says. Once copied to the destination server under the $ HOME directory, the file that all PSMs must have to interact with the command line using the Bash language is executed.

```
command = "/bin/bash " + DEPLOY_PATH + "provision.sh " + DEPLOY_PATH
+ " >>" + DEPLOY_PATH + "logs/output.txt 2>&1"
stdout, stderr = Popen(['ssh', SSH_NODE, command]).communicate()
print(stdout)
```

This ‘provision.sh’ file is called from a function which will wait for it to finish. Once finished, it means that the service has been completed, then the teacher is notified that the service is ready by calling the sendEmail program with the sendComplete class which is specific for this case because it will attach the file with the generated data as an attachment by the PSM so that the teacher can access and have the instructions.

2.3. Delete

This class is designed to be implemented in future work, although it can be seen how the destruction of the service is planned. First, depending on the virtualization, the service is located to obtain the results of the students and save them on an SFTP server. Then, the service is stopped and destroyed in the virtualization platform, in addition to removing the directory with the data of the service on the destination server. Finally, the teacher is notified that the service has been removed and where they can consult the saved data.

3. PROVISION DEPLOY

This section explains how the generic provisioning script works for all existing and future PSMs which is called by the Python code of the specific PSM. This provision file is written in Bash to have maximum efficiency when executing the commands in the Linux terminal for bringing up the service. All provision files must have this code structure:

```
#!/bin/bash
set -e
trap 'last_command=$current_command; current_command=$BASH_COMMAND' DEBUG
trap 'echo "\"${last_command}\" command filed with exit code $?."' EXIT

[... SPECIFIC DEPLOY CODE ...]

cd $DEPLOYPATH
echo "The service with UUID $UUID, is ready at URL antidote-local:$PORT."
>> completed.txt

[... Specific instructions ...]
```

It begins by explicitly defining that the Bash language is used from its binary, then the program is marked to exit immediately if a command exits with a non-zero status, that is, when an error occurs in the execution of a command. Next, the specific code for the service deploy is specified along with the variables that gather the arguments passed to the program. Finally, change the directory to the parent where the deploy is made and a file 'completed.txt' is generated with the necessary information and instructions.

Regarding the specific PSM code in the provision file, the code used for both PSM implemented in the final project will be commented.

A. ANTIDOTE SELFMEDICATE

Next, the Antidote provisioning code is going to be discussed.

```
#!/bin/bash
set -e
trap 'last_command=$current_command; current_command=$BASH_COMMAND' DEBUG
trap 'echo "\"${last_command}\" command filed with exit code $?."' EXIT
```

```
# The main variables are assigned using the arguments of the code call
VMPROVIDER=$1
DEPLOYPATH=$2
UUID=$3
PORT=$4

# Change directory to the antidote-selfmedicate path
cd $DEPLOYPATH

# In this implementation, libvirt has not been introduced, although work
has been done so the code is left.

if [ "$VMPROVIDER" == "virtualbox" ]; then
    # If the provider is Virtualbox

    # Check if Virtualbox and vagrant are installed. Abort if not.
    command -v VBoxManage >/dev/null 2>&1 || { echo >&2 "I require
VBoxManage but it's not installed.  Aborting."; exit 1; }
    command -v vagrant >/dev/null 2>&1 || { echo >&2 "I require vagrant but
it's not installed.  Aborting."; exit 1; }

    # Change directory to antidote selfmedicate and install needed plugins.
    cd ./antidote-selfmedicate

    vagrant plugin install vagrant-vbguest
    vagrant plugin install vagrant-hostsupdater

elif [ "$VMPROVIDER" == "libvirt" ]; then
    # If the provider is Libvirt

    # Check if Libvirt and vagrant are installed. Abort if not.
    command -v libvirtd >/dev/null 2>&1 || { echo >&2 "I require libvirtd
but it's not installed.  Aborting."; exit 1; }
    command -v vagrant >/dev/null 2>&1 || { echo >&2 "I require vagrant but
it's not installed.  Aborting."; exit 1; }

    # Change directory to antidote selfmedicate and install needed plugins.
    cd ./antidote-selfmedicate

    vagrant plugin install vagrant-libvirt
    vagrant plugin install vagrant-hostsupdater

fi

# Be sure to be in Antidote dir and perform the deployment of the Antidote
code
cd ./antidote-selfmedicate
vagrant up

# Add designed port to firewall exception
```

```
sudo firewall-cmd --permanent --add-port=$PORT/tcp
sudo firewall-cmd --reload

# Change working directory to the parent deploy directory
cd $DEPLOYPATH

# Create the instructions for the teacher if the service deployment
succeeds
echo "The service with UUID $UUID, is ready at URL antidote-local:$PORT."
>> completed.txt
echo "Before accessing it, add this line:" >> completed.txt
echo "Linux: gedit /etc/hosts" >> completed.txt
echo "Windows: Open with Administrator privileges
c:\Windows\System32\Drivers\etc\hosts" >> completed.txt
echo "" >> completed.txt
echo "In any case add this line:" >> completed.txt
echo "$(eval "hostname -i")    antidote-local" >> completed.txt
```

Although the previous script is the most important, variables are also edited in the following. First, in *antidote-selfmedicate/antidote-config.yml*, the variables are replaced in the execution of the code class which will be loaded in the execution of 'vagrant up' in Vagrantfile for the configuration.

```
---
version: "0.6.0"
# Customizable configuration options are provided here for the vagrant up
command. This is used in conjunction with the vagrantfile.  Any changes in
this file require vagrant reload --provision
vm_config:
  vmname: ANTIDOTE_VMNAME
  memory: ANTIDOTE_MEMORY
  cores: ANTIDOTE_CPUS
  provider: ANTIDOTE_PROVIDER
  port: ANTIDOTE_PORT
  porttmp: ANTIDOTE_TMP
  ssh: ANTIDOTE_SSH
```

Finally, in *antidote-selfmedicate/selfmedicate.sh*, three variables responsible for configuring the platform within Virtualbox are also replaced in code class.

```
[...]

SELFMEDICATE_PORT=ANTIDOTE_PORT
CPUS=${CPUS:=ANTIDOTE_CPUS}
MEMORY=${MEMORY:=ANTIDOTE_MEMORY}
[...]
```

B. THE LITTLEST JUPYTERHUB

Finally, the Jupyter provisioning code is going to be discussed.

```
#!/bin/bash
# Steps:
#
http://tljh.jupyter.org/en/latest/contributing/dev-setup.html?highlight=dev

# exit when any command fails
set -e

trap 'last_command=$current_command; current_command=$BASH_COMMAND' DEBUG
trap 'echo "\"${last_command}\" command filed with exit code $?."' EXIT


# The main variables are assigned using the arguments of the code call
DEPLOYPATH=$1
LESSON=$2
TYPE=$3

# Change directory to the antidote-selfmedicate path
cd $DEPLOYPATH

# If the main Jupyter code is not present, then clone it
if [ ! -d "the-littlest-jupyterhub" ]
then
    git clone https://github.com/jupyterhub/the-littlest-jupyterhub.git
fi
# Jump to the Jupyter code directory
cd the-littlest-jupyterhub/

# Build the docker image and run it with variables
docker build -t tljh-systemd . -f integration-tests/Dockerfile
docker run --privileged --detach --name=JUPYTER_VMNAME --publish
JUPYTER_PORT:80 --mount type=bind,source=$(pwd),target=/srv/src
tljh-systemd
# After being created, first execute the installation and then set
resources limits
docker exec -t JUPYTER_VMNAME python3 /srv/src/bootstrap/bootstrap.py
--admin admin:JUPYTER_PASSWORD
docker exec -t JUPYTER_VMNAME sudo tljh-config set limits.memory
JUPYTER_MEMORYM
docker exec -t JUPYTER_VMNAME sudo tljh-config set limits.cpu JUPYTER_CPUS

# For N users requested, create them in the system
for i in {1..JUPYTER_USERS}
do
    docker exec -t JUPYTER_VMNAME sudo tljh-config add-item users.allowed
user$i
```

```
done

# Future work: All info in tljh/configurer.py
# Add collaboration modes
#if groups:
#   docker exec -t JUPYTER_VMNAME sudo tljh-config set
users.extra_user_groups.group1 user1

# Reload all applied config in TLJH
docker exec -t JUPYTER_VMNAME sudo tljh-config reload

# Clone the selected lesson in the shared folder under /home
docker exec -t JUPYTER_VMNAME git clone $LESSON /home/shared/$TYPE

# Add designed port to firewall exception
sudo firewall-cmd --permanent --add-port=JUPYTER_PORT/tcp
sudo firewall-cmd --reload

# Change working directory to the parent deploy directory
cd $DEPLOYPATH

# Create the instructions for the teacher if the service deployment
succeeds
echo "The service with UUID JUPYTER_VMNAME, is ready at URL $(eval
"hostname -i"):JUPYTER_PORT." >> completed.txt
echo "Administrator credentials:" >> completed.txt
echo "Username: admin" >> completed.txt
echo "Password: JUPYTER_PASSWORD" >> completed.txt
```

Appendix C: Setup guide

INTRODUCTION

This guide is designed to encourage future collaborators to prepare the environment in which the solution of this project has been implemented and to build a very similar environment thus being able to contribute innovative functionalities to the system or enhance the implementation that has been proposed.

The commands are based on Centos 7, so in case the server where this solution is implemented is under another Linux distribution, you should look for the synonymous commands for Advanced Packaging Tool (APT).

The structure of the chapters of which this guide is composed is as follows:

- Master node
- Worker nodes
- Authenticate worker nodes with public key
- Master node: Web Server

MASTER NODE

First we proceed to update the system packets, change the server hostname and add local DNS for the nodes.

```
sudo yum update -y && sudo yum upgrade -y
sudo hostnamectl set-hostname master
sudo cat << EOF >> /etc/hosts
# Final Project hosts
172.26.91.18      master      www.thesisraul.mche.edu.pl
172.26.91.20      node01
172.26.91.21      node02
EOF
```

Then, install all the necessary dependencies to be able to continue.

```
sudo yum install -y epel-release yum-utils git curl php73
mysql-server python3 perl yaml php73-php-pecl-yaml.x86_64 php-yaml
php-mysqli pip3 pyyaml mod_ssl
```

Then ports 80 (HTTP) and 443 (HTTPS) are opened in the system to be able to locate the web server on these.

```
sudo firewall-cmd --list-all
sudo firewall-cmd --permanent --add-service=httpd
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --permanent --add-port=80/tcp
sudo firewall-cmd --permanent --add-port=443/tcp
sudo firewall-cmd --reload
```

Then, we generate a pair of asymmetric keys to be able to authenticate ourselves further on the node servers without entering the password manually. We also enter the permissions in the directory where they are stored octally to only allow access to these keys to the owner of them.

```
ssh-keygen -t rsa -b 4096
sudo chmod -R 700 .ssh/
sudo chmod -R 600 ~/.ssh/id_*
sudo chmod -R 600 ~/.ssh/authorized_keys
```

Using PIP3 install the YAML extension for Python that will be required to execute the code.

```
pip3 install --user pyyaml
```

Now we set the two global variables to .bash_profile that will be used to authenticate the Gmail SMTP server.

```
sudo cat << EOF >> ~/.bash_profile
export MAIL_USERNAME=admin@mail.com
export MAIL_TOKEN=xxxxxxxxxxxxxxxx
EOF
source ~/.bash_profile
```

Next, we add the current user to the apache group and start the httpd and mysqld services.

```
sudo usermod -aG apache $(whoami)
sudo systemctl enable httpd
```

```
sudo systemctl start httpd
sudo systemctl enable mysqld
sudo systemctl start mysqld
```

WORKER NODES

In each of the worker servers, you must first verify that virtualization is enabled on all CPUs, this is a requirement because the services are mounted on top of virtualization platforms.

```
lscpu | grep Virtualization
- Virtualization: VT-x
```

Now update the system, change the hostname and put the local DNS for the master node and the current node.

```
sudo yum update -y && sudo yum upgrade -y
sudo hostnamectl set-hostname node01
sudo cat << EOF >> /etc/hosts
# Final Project hosts
172.26.91.18      master      www.thesisraul.mche.edu.pl
172.26.91.20      node01 # This is the same worker host where we are
configuring
EOF
```

Next, all the packages that are required to continue must be installed.

```
sudo yum install -y epel-release yum-utils git curl python3 gcc dkms
make qt libgomp patch kernel-devel-$(uname -r) kernel-headers
kernel-headers-$(uname -r kernel-devel binutils glibc-headers
glibc-devel font-forge dkms qemu-kvm libvirt libvirt-python
libguestfs-tools virt-install
```

This step must be performed if Virtualbox is used because it requires having some GUI to be able to execute the virtual machines.

```
sudo yum groupinstall "Cinnamon Desktop"
systemctl set-default graphical
```

```
reboot
```

Then install Docker, which is necessary to start the service The Littlest Jupytherhub.

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
Centos 7
    sudo yum install docker-ce docker-ce-cli containerd.io
Centos 8
    sudo dnf install docker-ce-3:18.09.1-3.el7
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $(whoami)
```

On the other hand, Vagrant is installed, which is used with the Antidote Selfmedicate service.

```
sudo wget
https://releases.hashicorp.com/vagrant/2.2.2/vagrant_2.2.2_x86_64.rp
m
sudo rpm -i vagrant_2.2.2_x86_64.rpm
```

And together, Virtualbox is installed.

```
sudo wget
https://download.virtualbox.org/virtualbox/rpm/el/virtualbox.repo -P
/etc/yum.repos.d
sudo yum install VirtualBox-6.1
```

Then the current user is added to the group docker and vboxusers to have absolute control over these platforms. Also create the ssh directory, if it does not exist, and the permissions are changed to only have access to the same user and no one else.

```
sudo usermod -aG docker $(whoami)
sudo usermod -aG vboxusers $(whoami)
mkdir -p .ssh/
sudo chmod -R 700 .ssh/
sudo chmod -R 600 ~/.ssh/authorized_keys
```

AUTHENTICATE WORKERS WITH PUBLIC KEY

After finishing the individual configuration, it logs back into the master server and the following command is executed for each node that is available. This command is in charge of copying the public key of the master node to the authorized_keys files which will allow accessing the node via SSH without putting a password.

```
master:~ $ ssh-copy-id -i ~/.ssh/id_rsa.pub $(whoami)@nodeXX
```

MASTER NODE - WEB SERVER

The idea of this web server is to serve as a basis for teachers to request the service that best suits their needs through a simple guided web form. The web application is built on LAMP (Linux, Apache, MySQL, PHP).

MySQL

Proceed with a guided installation to guarantee the minimum security of MySQL.

```
mysql_secure_installation
   -  Change the root password? [Y/n] n
   -  Remove anonymous users? [Y/n] y
   -  Disallow root login remotely? [Y/n] y
   -  Remove test database and access to it? [Y/n] y
   -  Reload privilege tables now? [Y/n] y
```

Create database

```
mysql -u root -p
create database cloud;
use cloud;
```

Create table TMrequests and TMspecs

```
create table tmrequests(id varchar(32), tstart varchar(32), uuid
char(32), fname varchar(32), sname varchar(32), email varchar(64),
homeinst varchar(32), homedept varchar(32), trole varchar(10),
opmode char(6), cmode varchar(10), startd varchar(32), span int,
PRIMARY KEY(uuid));
```

```
create table tmspecs(id char(32), msize varchar(6), ltype
varchar(32), ltopic varchar(32), homedir varchar(32), nusers int,
coopmode varchar(10), ngroups int, addsrv varchar(10), whatpi
varchar(20), PRIMARY KEY (id), FOREIGN KEY (id) REFERENCES
tmrequests(uuid) ON UPDATE RESTRICT ON DELETE CASCADE);
```

Exit the database and we restart the MySQL service

```
exit
sudo systemctl restart mysqld
```

INFORMATION - Commands to check the records of the tables.

```
mysql -u root -p
use cloud;
select * from tmrequests;
select * from tmspecs;
```

APACHE - CONFIGURATION

Assuming that the project source code is downloaded, we copy the Apache directory, change the permissions and the owner and restart the service.

```
sudo cp -r ~/cloud-orchestration/TM/thesisraul/ /var/www/
sudo chown -R apache:apache /var/www/
sudo chmod -R 755 /var/www/
sudo systemctl restart httpd
```

Next, we generate a self signed certificate for the implementation of the project. In a real environment this must be verified by a CA. Next we move the generated keys and change the permissions together with the owner.

```
openssl req -new -newkey rsa:4096 -days 3650 -nodes -x509 -subj
"/C=PL/ST=Malopolska/L=Krakow/O=MCHE/CN=www.thesisraul.mche.edu.pl"
-keyout thesisraul.key -out thesisraul.crt
sudo mkdir -p /etc/ssl/crt
sudo mv thesisraul.* /etc/ssl/crt/
sudo chown -R apache:apache /etc/ssl/crt
sudo chmod -R 755 /etc/ssl/crt
```

Then, configure httpd to link the web site to the virtual hosts.

```
sudo mv /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.bak
sudo cp ~/cloud-orchestration/TM/httpd/conf/httpd.conf
/etc/httpd/conf/httpd.conf
sudo mkdir -p /etc/httpd/sites-available
sudo mkdir -p /etc/httpd/sites-available
sudo cp ~/cloud-orchestration/TM/httpd/sites-available/*
/etc/httpd/sites-available
sudo ln -s ../sites-available/thesisraul.conf thesisraul.conf
sudo ln -s ../sites-available/thesisraul-ssl.conf
thesisraul-ssl.conf
sudo systemctl restart httpd
```

COMMON ISSUE

Then we modify the apache user so that he has access without a password, this is the solution to an error that occurred when calling a bash file from PHP.

```
sudo visudo
  apache  ALL=(ALL) NOPASSWD:
/var/www/thesisraul/public_html/bash_call.sh
```

PHP - Configuration

Just keep in mind that the PECL version installed must be greater than 0.5.0

```
PECL yaml >= 0.5.0
```

Then disable SElinux so that there are no problems with the process in general, since for this implementation it is not required to have its operation.

```
sudo vi etc/selinux/config
# This file controls the state of SELinux on the system.
SELINUX=disabled
```

Finally we change the context in the bash_call.sh file which will call the main codiog of the program as user the one specified. This is necessary since by default the apache user is used because the call is made from it.

```
sudo vi /var/www/thesisraul/public_html/bash_call.sh
```

```
USR_CONTEXT='raulloga' # Must be changed to the user used in the
implementation
```

SOURCE CODE

Regarding the source code, you must first clone in the $HOME directory and edit the configuration variables as appropriate.

```
cd ~
git clone https://github.com/llopisga/cloud-orchestration.git
```

```
vi cloud-orchestration/code/conf/config.yaml
PROJECT_PATH: "/home/USERNAME/cloud-orchestration"
TM_PATH: "/var/www/thesisraul/public_html/requests"
SMTP_SERVER: "smtp.gmail.com"
SMTP_PORT: 587
NOTIFY_ADMIN: "raullg8@gmail.com"
```