UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Using the Cloud in Lab Classes

Samuel Filipe Henriques Alves

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientadora: Profª. Doutora Paula Prata

Covilhã, Outubro de 2014

Agradecimentos

Gostaria de agradecer às pessoas que contribuíram e me apoiaram ao longo do desenvolvimento desta dissertação.

Agradeço à minha orientadora, a professora Doutora Paula Prata pelo apoio e conhecimento que me transmitiu ao longo do desenvolvimento desta dissertação.

Estou também agradecido aos meus pais e irmãos que me apoiaram ao longo de todo o este percurso académico e me deram ânimo para ultrapassar os obstáculos que foram surgindo.

Acknowledgements

I would like to thank all those who helped and supported me throughout the development of this dissertation.

I want to thank my advisor, Professor Dr. Paula Prata for support and knowledge conveyed during the development of this dissertation.

I also thank my parents and siblings who have supported me throughout this academic path and encouraged me to overcome obstacles have emerged.

Resumo

Esta dissertação apresenta o desenvolvimento de uma aplicação que permite criar e gerir laboratórios de computação virtuais usando plataformas de Nuvem (Cloud). O uso de laboratórios virtuais é usual em áreas como a engenharia e ciências da computação. No entanto, a maioria das soluções existentes são proprietárias e exigem algum esforço de aprendizagem.

A solução proposta, CSCLab – Computer Science Cloud Laboratory, permite aos profissionais de ensino criar os laboratórios necessários para dar uma aula, usando os recursos da Nuvem, sem a necessidade de conhecer a tecnologia subjacente. As máquinas virtuais do laboratório de computação podem ser acedidas pelos alunos utilizando um protocolo de comunicação remota (por exemplo, SSH, RDP). A plataforma CSCLab foi criada por forma a evitar o bloqueio a soluções proprietárias.

CSCLab segue uma arquitetura Modelo-Vista-Controlador, que nos permite trabalhar com cada um dos componentes da plataforma de forma independente. A plataforma foi desenhada para ser independente da infraestrutura de Nuvem. Finalmente, para validar a plataforma, desenvolvemos um protótipo da plataforma que utiliza uma infraestrutura de Nuvem baseada em OpenStack. Para alterar a plataforma por forma a utilizar outras infrastruturas de Nuvem apenas será necessário trabalhar com os modelos do protótipo. Para a implementação desta plataforma, foi também criada uma Nuvem privada baseada em OpenStack para ser usada como ambiente de desenvolvimento.

O código da plataforma é fornecido de forma livre em `http://www.csclab.s-alves.pt`.

Palavras-chave

Cloud, Computação na Cloud, Computação na Nuvem, CSCLab, Educação na Cloud, Laboratórios virtuais, OpenStack.

Resumo alargado

Os laboratórios de computação são uma necessidade crucial para as instituições de ensino, implicando normalmente grandes necessidades de manutenção e custos elevados. Estes laboratórios apresentam alguns desafios para os departamentos de informática das instituições. No início de cada ciclo de estudos é necessário configurar os recursos computacionais específicos para cada laboratório, assim como a instalação de Software necessário para as diversas atividades letivas.

A utilização de laboratórios computacionais virtuais é já um fato adquirido em áreas da educação como a engenharia e ciências computacionais. Isto é possível através da utilização de diversas tecnologias de virtualização. Mais recentemente, a computação na nuvem tem vindo a ganhar destaque por permitir a criação de ambientes de computação que permitem grande flexibilidade e personalização. A computação na nuvem pode também ser chamada de Nuvem ou Cloud.

A computação na nuvem é definida pelo Instituto Nacional de Standards e Tecnologia (NIST em inglês) como sendo um modelo que permite um acesso ubíquo, conveniente e a pedido (on-demand) a um conjunto de recursos computacionais que podem ser provisionados ou libertados com interação mínima com o fornecedor de serviço. De acordo com o NIST a nuvem é composta por três modelos de serviço, dando cada um deles, acesso a diferentes níveis de abstração dos recursos da Nuvem. O primeiro modelo, Software como um Serviço – SaaS, é o que permite aos seus utilizadores um menor controlo da infraestrutura. Depois temos o modelo de Plataforma como um Serviço – PaaS, que se apresenta como um modelo intermédio e que permite mais algum controlo sobre os recursos da Nuvem. Por fim, temos o modelo de Infraestrutura como um Serviço – IaaS, que se apresenta como o modelo que permite o maior controlo sobre os recursos da Nuvem. Utilizando este último modelo, é possível criar e gerir máquinas virtuais, usando recursos da Nuvem.

Esta dissertação apresenta uma plataforma que permite a criação e gestão de laboratórios a partir da Nuvem. Para o desenho da plataforma, começou por se estudar as tecnologias existentes na área da computação para a criação destes laboratórios. Deste estudo destaca-se o problema de as soluções existentes se basearem em Software proprietário e comercial, o que representa um problema para os utilizadores, já que estes ficam limitados a este tipo de tecnologia. Para além disso, as soluções existentes são destinadas a utilizadores com algum grau de conhecimento das mesmas, apresentando uma curva de aprendizagem que envolve tutoriais e horas de treino. Nós propomos uma solução que evita a dependência dos utilizadores a uma tecnologia específica, podendo ser facilmente adaptada a diversos tipos de plataforma de Nuvem. A nossa plataforma apresenta-se também como uma plataforma facilmente acessível por qualquer utilizador, não exigindo um conhecimento específico da tecnologia.

O desenvolvimento desta plataforma, CSCLab – *Computer Science Cloud Laboratory*, passou por um levantamento prévio de requisitos que possibilita a criação de um sistema que permita aos professores das instituições de ensino criar laboratórios personalizados para as aulas a lecionar. Este sistema permite que os estudantes das instituições acedam a estes laboratórios e possam utilizar as máquinas virtuais existentes. A plataforma permite ainda que os estudantes utilizem recursos de armazenamento de dados, também eles disponibilizados na Nuvem. O desenvolvimento da plataforma segue uma arquitetura Modelo-Controlador-Vista – MCV que permite que se trabalhe com cada componente da plataforma de forma independente. De seguida foi efetuado o desenho da plataforma de forma a ser independente da infraestrutura de Nuvem subjacente. Para validar a arquitetura e o desenho da plataforma foi desenvolvido um protótipo

que tem por base o OpenStack, mas que pode ser facilmente adaptado a outras infraestrutura de Nuvem. Previamente foi criada uma infraestrutura de Nuvem baseada em OpenStack, a qual foi usada para testar as funcionalidades implementadas na plataforma. O código da plataforma é fornecido de forma livre em `http://www.csclab.s-alves.pt`.

Abstract

Computation laboratories are an essential part of educational institutions. The management of these laboratories presents some challenges to these institutions and to their IT department. The laboratories need to be prepared in advance, before the start of the cycle of study and usually entail high costs to the institutions.

The use of virtual computing laboratories is not new to some areas like engineering and computer science. The creation of these laboratories relies on virtualization technologies. Currently, computation in the cloud has attracted some attention as it enables the creation of highly flexible and customizable computation environments.

The cloud is defined by the National Institute of Standards and Technology – NIST, as being a model that allows the access to a pool of computing resources, in a ubiquitous, convenient and on-demand way. These resources can easily be provisioned or released with a minimum of interaction with the service provider. NIST uses three service models to describe the cloud: the Infrastructure as a Service model – IaaS; the Platform as a Service model – PaaS; and the Software as a Service model – SaaS. These models allow access to different levels of abstraction of the cloud resources. SaaS model provides the most restrictive access, then PaaS provides a less restrictive access and finally, IaaS is the least restrictive one. Using IaaS, the users have access to the basic computing resources.

This dissertation proposes a solution, CSCLab – Computer Science Cloud Laboratory, to provision and manage virtual computing laboratories using cloud resources. The solution we propose enables any teaching professionals to create the laboratories necessary to teach a class. The machines of the virtual computing laboratory can then be accessed by the students using some remote desktop protocol (e.g., SSH, RDP). This platform was has been built aiming at the creation of a solution to avoid proprietary lock in's, a problem that is faced using existent commercial and proprietary solutions. Furthermore, these solutions present the users with a learning curve that implies the use of tutorials and training time. Our solution, on the other hand, is designed to be used by non-experts so that even a user with no knowledge of these technologies is able to use it without the need to follow tutorials or receiving training.

CSCLab follows a Model-View-Controller system architecture that allows us to work with each component of the platform independently. The platform was then designed to be agnostic to the cloud infrastructure. Finally, to validate the platform, we have developed a prototype of the platform that uses an OpenStack based cloud infrastructure. For the implementation of this platform, we have also created an OpenStack based private cloud to be used as a development environment.

The platform is available as an open source solution in `http://www.csclab.s-alves.pt`.

Keywords

Cloud, Cloud Computing, CSCLab, Education in the Cloud, OpenStack, Virtual laboratories.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheet
EGI	European Grid Infrastructure
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IT	Information Technology
JSON	JavaScript Notation Object
POX	Plain Old XML
OS	Operating System
REST	Representational state transfer
RDP	Remote Desktop Protocol
SDK	Software Development Kit
SSH	Secure Shell
UML	Unified Modelling Language
URI	Uniform Resource Identifier
VCL	Virtual Computing Laboratory
VM	Virtual Machine

Chapter 1

Introduction

Computing laboratories are an essential part of the educational institutions but their creation and maintenance is usually expensive [XHT12].
In the universities, the use of virtual computing laboratories is already usual in areas like engineering, computer sciences and information assurance [WHY10, XHT12, Ueh13, BLS13]. Currently, the use of the Cloud to create these virtual laboratories has gained great prominence. The use of the Cloud enables the creation of highly flexible and configurable environments [XHT12].

The Cloud is defined by **N**ational **I**nstitute of **S**tandards and **T**echnology – NIST – as a model for ubiquitous, convenient and on-demand network access to a pool of shared computing resources (e.g., network, storage, processing) that can be rapidly provisioned and released with a lower management effort and interaction with the service provider [MG12]. NIST describes the Cloud model as a set of service and implementation models, as well as a set of essential characteristics.

According to NIST, the Cloud is composed of three service models which are Software as a Service – SaaS, Platform as a Service – PaaS and Infrastructure as a Service – IaaS. Iaas is a Cloud service model that provides users with access to essential computing resources, being the less restrictive model for the Cloud clients. This model is composed by a Hypervisor (i.e., a Virtual Machine Monitor) which works on top of the hardware of the Cloud infrastructure. This Hypervisor can then be used to create and manage virtual machines – VM – and also it provides interfaces to network features which can be used to configure virtual networks.

The Cloud infrastructures can be deployed using four different deployment models: Public Clouds, Private Clouds, Hybrid Clouds and Community Cloud. The first two models are essentially the same, the main difference being the property and user access rights. In the public Cloud, the provider entity may be an enterprise, academic or government institution or even a mix of the referred entities and the infrastructure is located inside the providers' installations. The private Cloud is maintained for exclusive use by a single organization and the infrastructure can be managed by the organization, a third party or a mix of the entities. In this case, the infrastructure may be located inside the organization or be the property of an external entity. The combination of public and private Clouds creates a hybrid Cloud which can be used for load balancing between infrastructures. The community deployment model creates a Cloud infrastructure that is used for exclusive use by a community of users from organizations with equal concerns. This infrastructure can be managed by one or more of the community organizations, a third party or a mix of both. It can be the property of the involved organizations or of a third party and may or may not exist inside their installations.

1.1 Motivation

The computing laboratories of education institutions (i.e., universities, secondary schools) present many challenges for the Information Technology – IT – departments of these institu-

tions. These challenges include the preparation and management of the laboratories which need to be prepared with the necessary software for the class labs. Furthermore, building and management costs of these laboratories are usually high [XHT12].

The laboratories need to be prepared in advance, i.e., all the software necessary for the lab classes needs to be pre-installed. We need to take into consideration that, generally, the users have no administration rights, thus not being allowed to install new software.This way the lab classes are conditioned by the a priori available software. The IT department needs to prepare all the software and hardware, resulting in high costs for the institution [Ueh13].

Typical laboratories are conditioned by their reusability, flexibility and scalability [XHT12]. This happens because the software and hardware necessary for different classes is not the same, then a specific laboratory configuration may be required by each of the classes. The scalability problem is raised by the limitation of computers in classrooms.

1.2 Proposed solution

We propose CSCLab – Computer Science Cloud Laboratory, a platform to provision and manage virtual computing laboratories using cloud resources. This solution enables university professors and other university staff to create the laboratories necessary to teach a class. The machines of the virtual laboratory can then be accessed by the students using some remote desktop protocol (e.g., SSH, RDP). CSCLab is a platform that enables the control of IaaS cloud resources in order to create and manage flexible and configurable cloud virtual laboratories. The access to the platform can be made from any type of devices since it has a responsive interface.

The proposed solution was designed to be agnostic to the underlying cloud infrastructure. To validate the proposed solution, we have constructed a prototype using the IaaS OpenStack [Sof13]. The implemented platform can easily be ported to other cloud infrastructure by following the platform architecture and design.

CSCLab was also designed by taking into account the future users of the platform. This way, we have created a platform that could be easily used by non-experts. The use of the platform requires a minimal learning curve that does not require hours of training and tutorials.

1.3 Contributions

The accomplishments of this dissertation led to a set of contributions in the field of Cloud Computing for education:

- This dissertation makes a contribution to the study of the state of the art in the field of Cloud Computing for education. We describe the Cloud, its implementation and deployment models. Then, we describe the use of virtual computing laboratories in educational institutions and the technologies used.
- The study of the problem led us to gather a set of requirements to build a platform to create and manage virtual laboratories in the Cloud. We have made a requirements analysis for the platform.
- Since we needed unlimited access to a cloud infrastructure, we have designed the architecture of a private cloud, which was implemented using OpenStack. This architecture

was used to create a development environment for the implementation of a platform prototype.

- We have constructed a prototype for the platform, using an OpenStack based Cloud. The source code is available at: `http://www.csclab.s-alves.pt`.
- For the development of the platform, we have implemented additional features to OpenCloud SDK, a PHP SDK for OpenStack based clouds. The changes to the SDK will contribute to the PHP-OpenCloud project.
- Finally, in the course of the dissertation, a set of scientific papers have been published.

1.3.1 Scientific work

This dissertation led to the publication of two scientific papers:

Samuel Alves and Paula Prata,"*Proposal for an Agnostic Architecture of Cloud-based Computing Labs*". In INForum 2014, 6th Symposium on Computer Sciences, pages 244-259, 4th to 5th of September 2014.

Samuel Alves and Paula Prata,"*Using the cloud to build a computing lab*". In 9th Iberian Conference on Information Systems and Technologies (CISTI), pages 1-6, 18th to 21st of June 2014. Available from: `http://dx.doi.org/10.1109/CISTI.2014.6877038`.

1.4 Document organization

This dissertation has the following structure. In Chapter 2, the state of the art for Cloud Computing and the existing work in the field of virtual computing laboratories is presented. Chapter 3 describes the implementation of an OpenStack private cloud. It then, describes OpenStack API, where the communication with OpenStack services is presented. And, finally, it introduces PHP-OpenCloud SDK, how it is structured and the features we have added to it. In Chapter 4, we make the requirements analysis for the platform, then we present the system architecture and the design of the platform. Chapter 5 describes the implementation of the platform using OpenStack. Finally, in Chapter 6, we present our work conclusions and future work.

Chapter 2

State of the art

This chapter presents the Cloud Computing state of the art, the use of virtual computing laboratories – VCL in educational institutions and the paper of the Cloud in it. In the next section, we start by describing the Cloud, how it is defined and its implementation and deployment models, as well as the Cloud service providers. The following section is used to describe the use of virtual computing laboratories in education institutions. This section, describes the implementation of the VCL in those institutions and the technologies used. Finally, in section 2.3, we describe the disadvantages of those technologies, present our proposed solution and the advantages over existent solutions.

2.1 The Cloud

Computation in the Cloud, also know as Cloud, is an evolving paradigm [MG12] and is considered the next step in the evolution of computation [PP13]. This paradigm allows access to computing resources without previous knowledge from the users about the location and other information about the computation infrastructure [NNH12].

When talking about Computation in the Cloud, we are referring both the applications available in the form of services accessible through the Internet and the hardware and software present in data centres [ASZ+10], which in turn represent the Cloud infrastructure. Cloud computation is composed of a set of systems and technologies, as well as a set of service and implementation models [BGR12].

There are many definitions of the Cloud [MLB+11], according to [SPPH12] Cloud Computing is a *"massively scalable distributed computing paradigm driven by economics of scale that can be abstracted to deliver different level of services and can be dynamically configured and delivered"*. Thus, NIST presented its definition of Cloud Computing as being a model that allows the access, in an ubiquitous and convenient way, to a pool of configurable computing resources (networks, servers, storage) which can be rapidly provisioned and released with a minimum effort and interaction with the service provider [MG12].

The Cloud model presented by NIST consists of five essential features:

1. **On-demand self-service -** allows a user to use computational capabilities (e.g., network storage, server time) according to its needs without human interaction with the service providers.

2. **Broad network access -** the user can access the service through standard mechanisms that allow its use by the different platforms (e.g., smarthphone, tablet, personal computer).

3. **Pool of resources -** infrastructure resources are dynamically grouped, with different physical and virtual resources, according to the user's needs. There is a sense of location independence of the resources because, generally, the user has no knowledge or control over its location.

4. **Elasticity -** the automatic capability of using more or less resources depending on the service demand. Presents the user with a notion of infinite resources.

5. **Service management -** Cloud systems automatically control and optimize resource utilization by using service utilization's metering which are made at some level of abstraction, according to the resource type (e.g., storage, processing, bandwidth).

Some of these features were also highlighted in [ASZ+10]. The On-demand self-service mixed with the elasticity feature which presents the users with an apparent existence of infinite resources, without compromising the operation of the service. And, also, features that enable users to purchase the necessary hardware resources, without committing to unnecessary services.

In order to get this notion of elasticity and infinite resources upon request, a mechanism for automatic allocation and management of resources is required. There is an underlying number of technologies, services and infrastructure configurations that enables Cloud Computing. From these technologies, virtualization is one of the most important.

Virtualization, in its simplest form, allows us to abstract the hardware and system resources from a given Operating System (OS). Cloud uses a virtual machine monitor, the hypervisor, which links the infrastructure hardware and the operating system - OS - and that enables us to have one or more virtual machines running concurrently in the Cloud [YBvL11].

2.1.1 Cloud Service Models

According to the definition presented by NIST at [BGR12], Cloud Computing is composed by a set of three service models that describe the Cloud.

By the NIST's definition of Cloud Computing, the service models identified in the Cloud are the Software as a Service Model - **SaaS**, Platform as a Service Model - **PaaS** and Infrastructure as a Service Model - **IaaS**. These models (see Fig. 2.1) have their own strengths, allowing for each user to choose the service type that better adapts to its objectives.

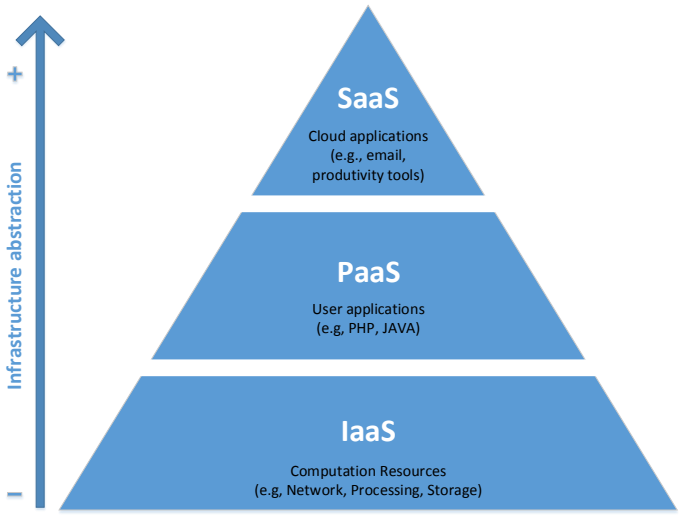


Figure 2.1: Cloud service architecture and abstraction level.

However, there are some authors that do not support this model. These authors argue that, since it is not always easy to make the division between the low level infrastructures and the high level platforms, this division makes no sense [ASZ+10].

2.1.1.1 Software as a Service

The SaaS term, also known as "Web Services", predates the Cloud Computing definition and has many implementation models, of which, the most advanced ones appear to correspond to the NIST definition of Cloud Computing [BGR12].

This type of service provides users with applications running in a Cloud infrastructure, which can be accessed from a user interface, either through a web browser or a program interface.

In this model (see Fig. 2.2a), the control of the users over the Cloud infrastructure is limited to user configurations regarding the application. The available applications through this model may be an email or other productivity tools. The service provider for SaaS is responsible by the support and test of the applications provided and due to the nature of much of the information data stored in the Cloud, a secure environment is mandatory [BGR12].

2.1.1.2 Platform as a Service

In the PaaS, the user has the capability to load applications to the Cloud infrastructure. These applications are created with specific languages, libraries, services and other tools supported by the infrastructure [BGR12, MG12]. PaaS providers offer development frameworks for both the server side and client side, which makes it easier for PaaS users to create well-written applications that are easily scalable.

In this service model (see Fig. 2.2b), the user has control over the applications and configurations specific for the application environment and/or host, having no control over the essential computing resources (e.g., processing, storage, network). The computing resources necessary for the applications are typically provided and maintained by the PaaS provider according to the needs of the application [BGR12]. The middleware layer, which is accessed by both users of the PaaS (Cloud provider and Cloud user), provides interfaces that are made available by the Cloud provider to the Cloud users who use it to access the resources of the Cloud [BGR12].

This type of service can be used as a traditional computing platform, for which a user can develop and run applications with the benefit for the users to create scalable applications. Once deployed, the application can be used as a normal SaaS [BGR12].

2.1.1.3 Infrastructure as a Service

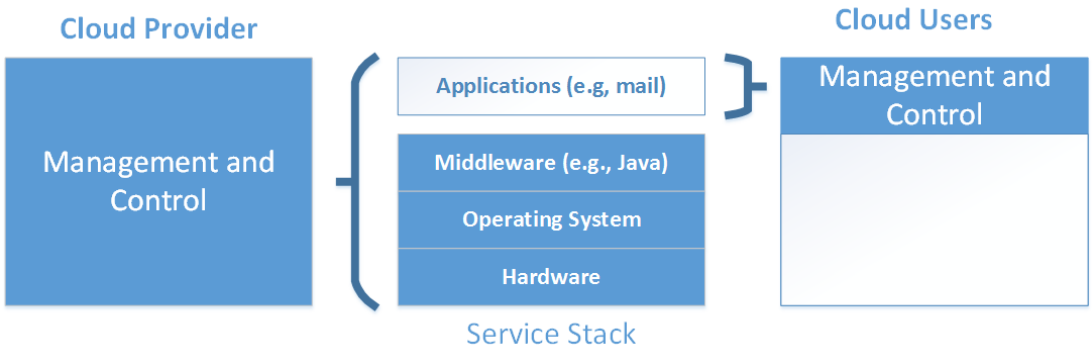
Using the IaaS service model (see Fig. 2.2c), a user has access to a set of essential computing resources (e.g., processing, network, storage) which can be used to implement and run software. This software can be an application and/or a full operating system [MG12].

This service model provides the users with the basic units, virtual machines, which have a specific hardware configuration and a determined set of applications. To create and manage the VMs, the IaaS uses the Virtual Machine Monitor, also known as Hypervisor [BGR12]. Using virtualization it is possible to have multiple VMs that may be on a single physical machine, isolated from each of the VMs and get a better use of the resources through workload consolidation [ICV12].

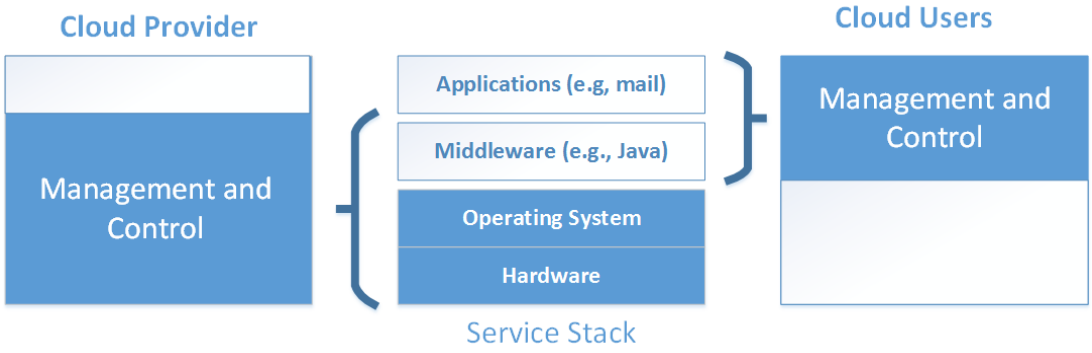
The VMs provided through IaaS appear like the actual computer to the service users which can be accessed through the network. These VMs can be configured and managed according to the user's needs (e.g, shutdown or restart machine, add or remove peripherals).

As seen in Fig. 2.2c, the access to the Hypervisor layer is enabled to both Cloud users, the Cloud provider and the Cloud users. The Cloud providers have an administrative control over

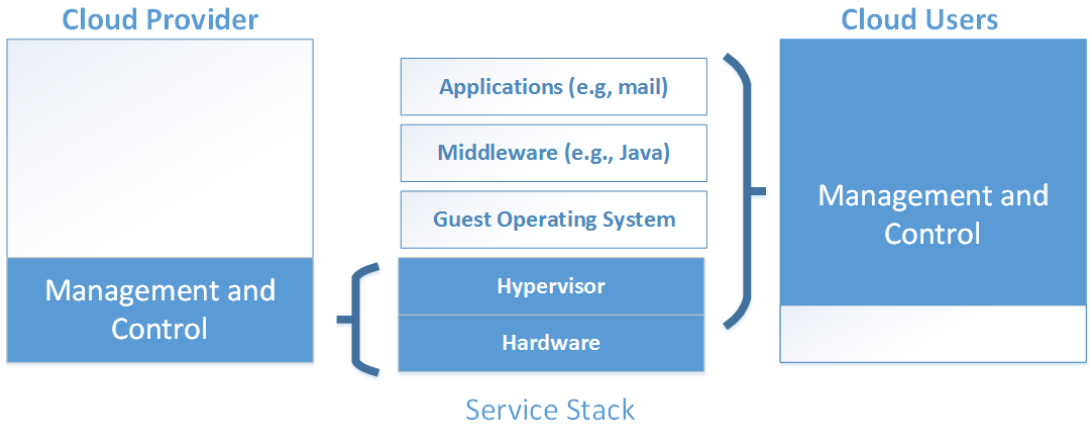
this layer. On the other hand, this layers can be used to make requests for the creation and management of VMs [BGR12].



(a) SaaS stack control.



(b) PaaS stack control.



(c) IaaS stack control.

Figure 2.2: Service model stacks.

Generally, it is possible to get better interoperability and portability for user applications by using IaaS because the blocks of this service model are better defined (e.g., network protocols, interfaces for legacy devices) [BGR12]. Thus, IaaS services are important tools for users wishing to have a big customization and management over their software stack. As the services using IaaS model put more service management responsibility in the hands of the clients than other service models, this requires more capabilities from them.

Then, there are users who wish a bigger abstraction over the Cloud infrastructure, which

use PaaS service [vDW12].

2.1.2 Cloud Implementation Models

The Cloud can be implemented following four distinct model types. As described in [MG12], these implementation models are: Public Cloud, Private Cloud, Community Cloud and Hybrid Cloud. These models have an implicit effect in the Cloud capabilities, as pointed out in [BGR12], *"Many statements commonly made about clouds (e.g., that clouds scale for very large workloads or that clouds replace capital expenses with operational expenses), however, are true only for certain types of clouds"*.

2.1.2.1 Public Cloud

The Public Cloud implementation model provides an infrastructure for use by the general public. This infrastructure is maintained by the service provider entity, which can manage and operate it. The service can be provided by an enterprise, academic or government institution or even a combination of the previously referred entities. This infrastructure is located in the installations of the provider's entity.

This model implements an infrastructure made of shared resources, which implements services that can be accessed over the Internet [Sup12].

2.1.2.2 Private Cloud

In a Private Cloud implementation model, the Cloud infrastructure implemented is maintained for exclusive use by one organization with multiple users. This infrastructure can be managed and operated by the organization, a third party, or a combination of both. The infrastructure can be located inside the organization or be the property of an external entity.

It is an infrastructure that simulates the characteristics of Cloud Computing in a private network [Sup12].

2.1.2.3 Community Cloud

The Community Cloud implementation model implements an infrastructure that is maintained for exclusive use of a users community with common concerns. It can be managed and maintained by one or more of the community organizations, a third party entity or a combination of both the community organizations and the third party entity. It can be property of the related organizations and/or a third entity and can or can not be located on site.

2.1.2.4 Hybrid Cloud

In a Hybrid Cloud implementation model, the Cloud infrastructure is a combination of two or more distinct infrastructures which remain unique entities. These infrastructures are connected by standard or proprietary technology which allows the portability of data and applications. This model is useful, for example, for load balancing between the infrastructures.

Public Cloud and Private Cloud implementation models are basically the same, the main difference between both models centers around the type of proprietary access rights of the

users. This leads to the Hybrid Cloud model, which allows the management of the service demand through time [CRB+11]. The Hybrid Cloud gains even more prominence when there are many clients with many computational and economic requirements which can not be met with only a Cloud infrastructure [SPPH12].

2.1.3 Cloud Service Providers

With the emerging market for Cloud management platforms, choosing a service over another becomes a complicated task because each platform has its set of characteristics. However, the articles that describe these features present, generally, only basic features, such as architecture, virtualization and service type. Then, there are also platforms in constant development and for this reason there are certain features that are not used in the articles that describe and compare the platforms [WGL+12].

Some of the most known Cloud public providers are: a) Amazon; b) Rackspace; c) Windows Azure; d) Google Compute Engine. These providers charge based on resource utilization.
From an open-source point of view, we have: a) Eucalyptos; b) OpenNebula; c) Nimbus; d) OpenStack. A comparison of these open-source Cloud Computing software platforms can be observed in Appendix A.

2.2 Related technologies

The implementation of virtual computing laboratories – VCL in education institutions (e.g., universities) is not new to engineering, computer sciences and information assurance [BLS13, WHY10, XHT12]. These virtual laboratories usually provide access to basic desktop virtualization services (e.g, operating system with specific software)[BLS13].

The implementation of the virtual computing laboratories is typically accomplished with technologies like Parallels [Par14], Microsoft Hyper-V [Hv14], VMWare [VMW14b], and Apache VCL [VCL14]. There is also an effort from infrastructure Cloud providers to present solutions for the education community, like the services provided by Amazon for Education [Ama14] for example.

There are some Cloud projects aimed at the research community, like the European Grid Infrastructure [Inf14] and FutureGrid [Gri14], which provide the community with a test-bed for experiments. EGI is composed of a set of providers (e.g., European Intergovernmental Research Organizations, Organizations through Memoranda of Understanding and a number of collaborative organizations with the EGI project). FutureGrid provides a computer grid and Cloud test-bed for researchers to tackle the complex challenges in computer sciences.

2.2.1 Parallels, Microsoft Hyper-v and VMWare

Parallels, Microsoft Hyper-v and VMWare are vendor specific solutions that enable desktop and server virtualization. Still, there are no solutions using these products that implement a complete VCL [BLS13].

There are a set of commercial solutions that enable the implementation of the VCLs:

1. Citrix XenApp and XenDesktop [Cit14a, Cit14b];
2. VMware ESX, Lab Manager and vCloud Director [VMw14d, VMw14c, VMw14a].

The solution 1 has its origins at the University of Cambridge with Xen. This product was spun off to a private company that has been later acquired by Citrix. Xen started as a hypervisor and since then it has evolved to support additional operating systems and hardware, being integrated into other Citrix products. In 2010 it was released under a GNU public license, still, most VCLs use additional products like XenApp and XenDesktop [BLS13].

The solution 2 is composed of a set of services from VMWare, which offers numerous desktop and server virtualization solutions. By using VMware ESX (i.e., bare-bones hypervisor solution) and Lab Manager (i.e., a portal component that is hosted in a Microsoft Windows Server and can be accessed over the network) it is possible to implement a VCL solution. The vCloud is a new product from VMWare directed at the Cloud. The users of Lab Manager are being migrated to this new product which provides more backend capabilities for the infrastructure. Thus, the services provided to the users are similar to the Lab Manager product.

2.2.2 Apache VCL

Apache VCL (i.e., Virtual Computing Laboratory) is an open source cloud computing solution for provisioning of virtual machines. Apache was developed by North Carolina State University in 2004, then it was donated to Apache Software Foundation in 2008, being a top level project since mid 2012 [VCL14, BLS13].

The solution provides a portal, a library for VM configuration, a deployment manager and a VM console interface. VCL resources can be provisioned in a range of computing resources from physical bare-metal machines, virtual machines hosted on different hypervisors to traditional laboratories found at university campus.

The user interface is a self-service web portal which can be used to make reservations for a custom environment. The users have access to a list of customized environments that can be used for the reservations. With VCL, users can create customized compute environments that can range from a simple virtual machine to a cluster of powerful physical servers. These virtual machines consist of a VM image with a specific operating system running specific productivity software.

In VCL a professor may customize a virtual machine with the necessary software for the lab classes and make the reservations using time-based reservations. The students can then request a VM from the reserved environment. After deployment of the VM with the custom image, the user can access it by using SSH or other remote desktop protocol like RDP [BLS13, IFY+13].

2.3 Proposed solution

CSCLab is a solution which intends to avoid proprietary lock in's from vendor solutions like Citrix and VMWare. These solutions also present some challenges for the users because they usually require some degree of knowledge for the users to be able to work with it. They have a learning curve that may imply tutorials and time for the users to know how to correctly use the platform services [WHY10].

Apache VCL, although being open source and vendor agnostic, is also not the solution to avoid proprietary lock in's since its implementation in the existent laboratories usually implement software and hardware from IBM [BLS13].

We propose a solution that solves some of the disadvantages of the presented technologies and, in addition, it presents some additional features. Our solution tries to solve proprietary

lock in's by presenting an agnostic system architecture. This allows us to implement a platform that may use different Cloud infrastructures. In our case, the platform was implemented to use OpenStack based clouds. Our solution, implements additional features that allow the users to use storage resources provided the platform. Furthermore, our solution allows the users to create customized environments. Finally, the solution we present was designed to be used by non-expert personal. This way, the platform does not require a big learning curve, which enables it to be used by anyone.

2.4 Summary

This chapter presents the state of the art for the cloud and for the use of virtual computing laboratories in education. From the study of the existent technologies, we can state that these raise a number of issues. The existent technologies present the problem of proprietary lock in's, i.e., the users get bonded to these technologies. Moreover, these technologies require the users to have some degree of knowledge about it, which can only be acquired through a set of tutorials and training time. By the other side, our solution relies in an agnostic architecture and design, avoiding the proprietary lock in's. Furthermore, our solution is designed to be used by non-experts, not requiring tutorials and training time.

In the next chapter, we present the implementation of a private cloud based in OpenStack and describes also the tools used to communicate and manage OpenStack services.

Chapter 3

OpenStack based private cloud implementation

This chapter covers the private Cloud that was created to validate the platform. The chapter introduces OpenStack and presents the architecture of the implemented private Cloud. Finally, we present the SDK used to communicate with OpenStack.

3.1 OpenStack

The OpenStack mission is *"to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable"* [Ope14b].

OpenStack is a cooperative effort of developers and other cloud technologists who produce a platform for open source cloud computing. OpenStack project was founded by Rackspace Hosting and NASA and is supported by a big number of enterprises and other organizations [Ope14a], having the objective of enabling any organization to have cloud computing services that can run on standard hardware.

OpenStack technology is composed of a set of interrelated projects which are components that enable the management of large pools of compute (Nova and Glance), storage (Swift and Cinder) and networking (Neutron) resources. OpenStack also has a set of shared services which intergrates the different components between each other and external systems. The Identity Service (Keystone) and Image Service (Glance) are part of the shared services.

3.2 Private Cloud architecture

3.2.1 Hardware

For the purpose of development and validation of our platform we have implemented a private cloud following the architecture of Fig. 3.1. This cloud infrastructure consists of two nodes which have the hardware specifications that we describe in Table 3.1.

The private cloud is composed of two nodes, NODE 1 and NODE 2, which are connected by two networks. In NODE 1, we have the controller and the network node of the OpenStack based cloud and NODE 2 is used as a computation and storage node. These nodes are connected to a management and data network, which is used to communicate between the nodes of the cloud (the cloud services, more precisely). Then, we have a second network, the API and external network, which is used by NODE 1 to provide access to the API of the OpenStack services. This network is also used to connect to the exterior and to the Internet.

3.2.2 OpenStack services configuration

The nodes (Fig. 3.1) of our Cloud were configured with a set of services to provide the minimal Cloud services necessary to use the CSCLab platform. These services were configured in

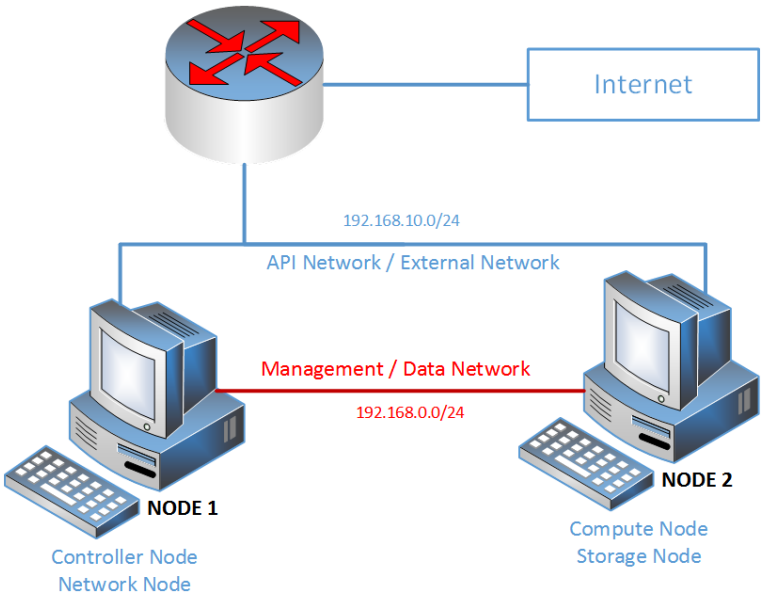


Figure 3.1: Private Cloud basic architecture.

Table 3.1: Nodes specifications

Hardware	NODE 1	NODE 2
CPU	i7	i7
RAM (Gb)	4	4
Storage(Gb)	500	500
Ethernet cards	2	2
Host OS	Ubuntu 13.10	Ubuntu 13.10

the corresponding nodes (i.e., NODE 1 and NODE 2) following the tutorial for the OpenStack Havana tutorial present in [Ope14e], with some adaptations and corrections for our system architecture. In table 3.2 we can see the services installed and configured in both nodes. The adaptations to the tutorial refer to the configuration of the network services, which should ideally be made in a dedicated node, but due to hardware restrictions they were made in the NODE 1. In table 3.3, we can see the network services configured in NODE1, which are used to configure and manage the network service and in NODE 2, the services used to connect this node with the network node.

The OpenStack Block Storage – Cinder – services should also be configured in a dedicated node but, in this case, they were also configured in the NODE 2. For the volume group necessary for the allocation of the Cinder volumes we used a partition of the node hard disk drive - **HDD** - with a size of 40% of the total size of the HDD. The cinder API and scheduler services were implemented in NODE 2, as described in the OpenStack guides.

The identification service of OpenStack, also known as Keystone, is fully implemented in the controller node as it is required for all other Cloud services. This service provides Identity, Token, Catalog and Policy services for OpenStack. This service is used to identify other services of the Cloud as well as the users, the tenants and the roles of the Cloud. Roles act as policy rules, defining the permissions of services and users. The services and users can be assigned to a specific tenant by means of a user-tenant-role association.

The Compute service is the main part of an IaaS system which is used for provisioning and managing of the virtual machines. This service relies on a set of virtualization technologies to manage the virtual machines and can de used with a wide range of hypervisors, of which KVM and XenServer are the popular and recommended choices for most of the use cases. This service uses the Image service to get the images necessary to instantiate the instances as needed, downloading it from Glance.
The controller services necessary for the Compute service were installed and configured in the controller node. In the compute node, we installed the compute service which uses, in this case, the KVM hypervisor to provision and manage the virtual machines.

Table 3.2: Services used in the nodes of the cloud infrastructure.

Services	**NODE 1**	**NODE 2**
Basic Services	mysql-server python-mysqldb rabbitmq-server ntp	mysql-client python-mysqldb ntp
Compute - Nova	nova-api nova-scheduler nova-conductor	nova-compute
Identity - Keystone	keystone (all services)	
Image - Glance	glance-api glance-registry	
Block Storage - Cinder	cinder-api cinder-scheduler	cinder-volume

The Network service – Neutron – manages software-defined networks and can be used to configure advanced network configurations. In our case, these configurations enable us to have private networks for each of the tenants. In order to get this network configuration, we have implemented Generic Routing Encapsulation, GRE, technology which is used in many VPNs. GRE wraps IP packets inside new packets with new routing information. Then in the destination the

packets are unwrapped and the previously encapsulated packet is routed. In order to use the GRE technology, Neutron create GRE tunnels, which are simply ports in bridges and that enable bridges in different systems to act as one.

Also, Neutron has an external network which is a representation of part of the physical network and provides access to the external network.

Note: The network configuration for the Neutron service is a little more tricky to configure as stated in the Networking chapter of the OpenStack install guide [Ope14e]. A more detailed guide for the chapter can be found in the Appendix B.

Table 3.3: Network services for the followed network architecture.

NODE 1	NODE 2
neutron-server neutron-plugin-openvswitch-agent openvswitch-agent neutron-dhcp-agent neutron-l3-agent openvswitch-datapath-dkms	neutron-plugin-openvswitch-agent openvswitch-switch openvswitch-datapath-dkms

3.3 OpenStack API

The communication with OpenStack services can be made through a number of ways. OpenStack provides a Representational State Transfer – REST – service API which we can use to manage and configure OpenStack services. By using the provided REST API, we can communicate and manage the OpenStack cloud services though the network, using standard protocols.

3.3.1 REST

REST is the architectural style that drives the Web [RR07]. Web Services use standardized protocols and formats to provide functionality and data on the Web. There are two big families of Web Services, the WS-* family and the REST services [LSS12]. REST being resource-oriented focus on application data transfer through HTTP, meaning it is data-centric [LKS06].

REST was introduced by Fielding who defined it with a set of designing principles [LKS06, PLH09]:

1. Explicit using HTTP:
   The resources are generally managed by HTTP standard methods, i.e., GET, POST, PUT and DELETE. Using these methods it should be possible to operate the resources (e.g., POST to create resources, GET to read a resource information, DELETE to delete a resource and PUT to make updates to a resource).

2. Stateless:
   This way, each interaction with the server is independent from the others. This improves the scalability of servers because different servers may handle different requests.

3. Each resource is identified by a URI.

4. Transfer JSON and POX/XML:
   Ddefines the representation of the resource.

REST services implement the architecture described in Fig. 3.2. A client sends a request message to a resource, which is represented by a URI and, in response, the server sends a message with the resource data (a typical HTTP request using a GET method). This information is typically in XML or JSON. Updating HTTP methods (e.g., as POST or PUT) on the other hand, send data to the resource in the server.

REST provides the advantages of the Web to the clients by implementing the same principles that drive the Web. These principles have been proven to simplify interoperability between resources, while providing scalability and flexibility [AZW09].

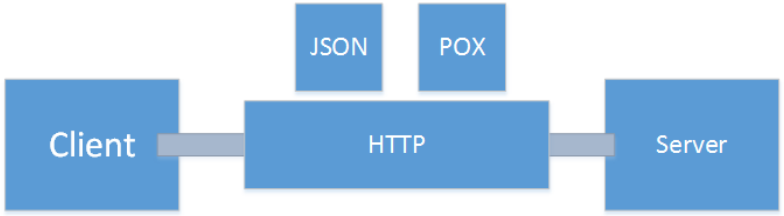


Figure 3.2: Architecture of a REST Web Service.

3.3.2 Using the API

We can make requests to the API using cURL (command-line tool to send HTTP requests and receive responses), the command-line clients from OpenStack, OpenStack Python SDK or REST clients (e.g., Firefox plugin: RESTClient; and Chrome plugin: rest-client;) [Ope14d].

By using OpenStack APIs, we can create containers and objects, launch instances and make many other operations in the Cloud [Ope14c].

The use of the API follows a well-defined work-flow (Figure 3.3).

1. We need to get an authentication token which we can use to authenticate access to OpenStack services.

2. Send API requests using the acquired token in a **X-Auth-Token**.

3. If request results in a Unauthorized error (401), then request new token.

To acquire the token we need to make a request (see Listing 3.1) to the OpenStack Identity service by using an endpoint registered in the Identity service and supplying a set of credentials in the request payload. This set of credentials is generally composed of a user name, a password and a tenant name or ID (optionally). We can also use a token in the request payload to get new tokens (not using a user name and password set). The token has an expiration date, meaning that it has a limit time, and may also become invalid if user permissions change (if the user roles change) or a Cloud administrator revokes it.

If the request is successful, then the response will include the token, an expiration date and possibly a service catalog and the user roles for the requested tenant (if provided in the request).

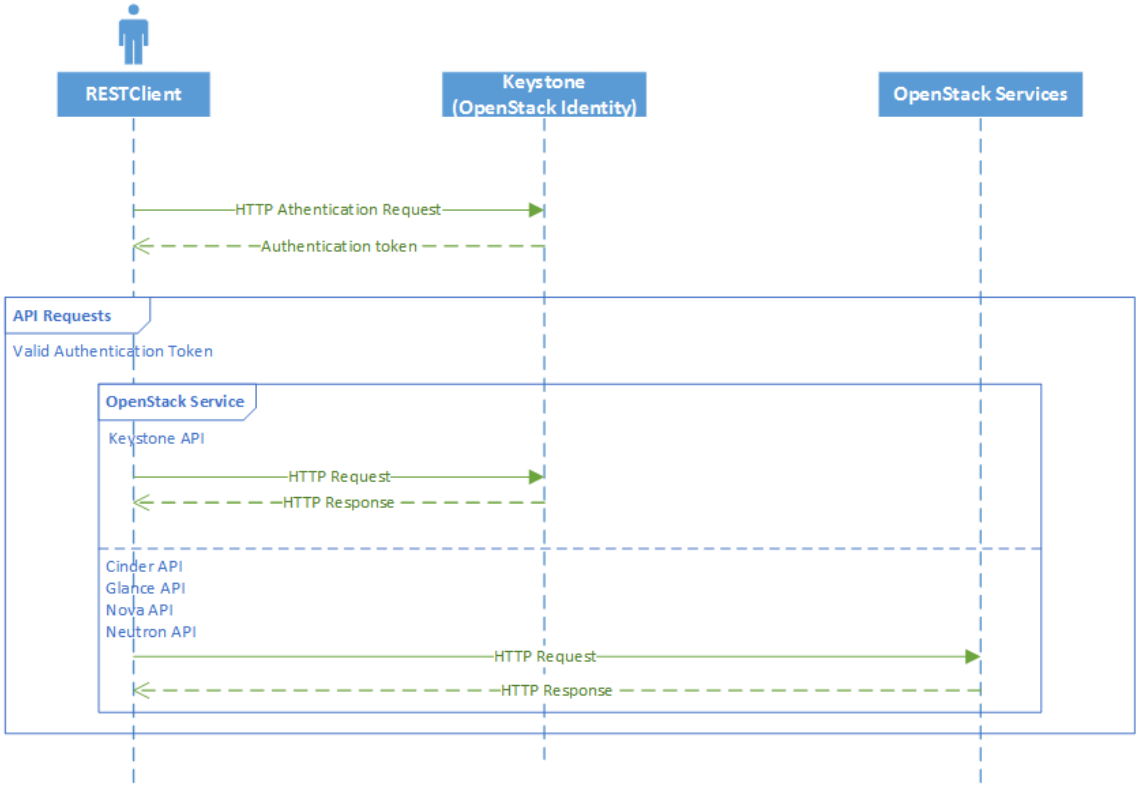


Figure 3.3: OpenStack services API consume sequence.

Listing 3.1: Request authentication token

```
Endpoint: Identity Endpoint
Method: POST
Headers: Content-type, Accept
Content-type: application/json or application/xml
Accept: application/json or application/xml
Payload: Credentials
```

We can then make calls to the OpenStack services APIs by using the acquired token in a specific header (i.e., **X-Auth-Token**), defining the method to be used in the HTTP request and using the service endpoint, and possibly, other request options (e.g., payload). The service endpoint to be used is present in the service catalog that might have been acquired in the request to get a valid token and if not, then we may make a request to the Identity API to acquire it.

3.3.3 OpenStack SDK: PHP OpenCloud

OpenStack provide developers with tools for the application development on private and public OpenStack Clouds [Ope14f]. At the moment there are SDKs for some of the most common programming languages (e.g., Java, Node.js, PHP and Python) which are a work in progress.

We opted for PHP OpenCloud [Rac14], a PHP SDK for OpenStack/Rackspace APIs, which works with most of the OpenStack-based cloud deployments. This SDK is composed of two main components as seen in Fig. 3.4, the OpenStack class which is used to get the connection with the OpenStack Identity service and a package comprising the OpenStack services.

The OpenStack class is a resource used to get access to the remaining resources of the cloud infrastructure. The resource is composed of a URI, which is the endpoint for the OpenStack Identity service, the data necessary to request the authentication token and stores the data retrieved in response to the request. The data necessary for the requests are the user credentials (i.e., user name, password, tenant id) or a valid authentication token. In response to the authentication request, we get the user information and the services catalog.

After a successful authentication request, we can then use OpenStack object to get access to the other OpenStack services. To use a specific service, e.g., the Compute service, we use an instantiated object of the OpenStack class to get access to a new object which will be a Compute service object. The SDK its made so that when this happens, the Compute service endpoint is retrieved from the service catalog present in the OpenStack object. Then, we can use this resource URI to make operations in Compute service.

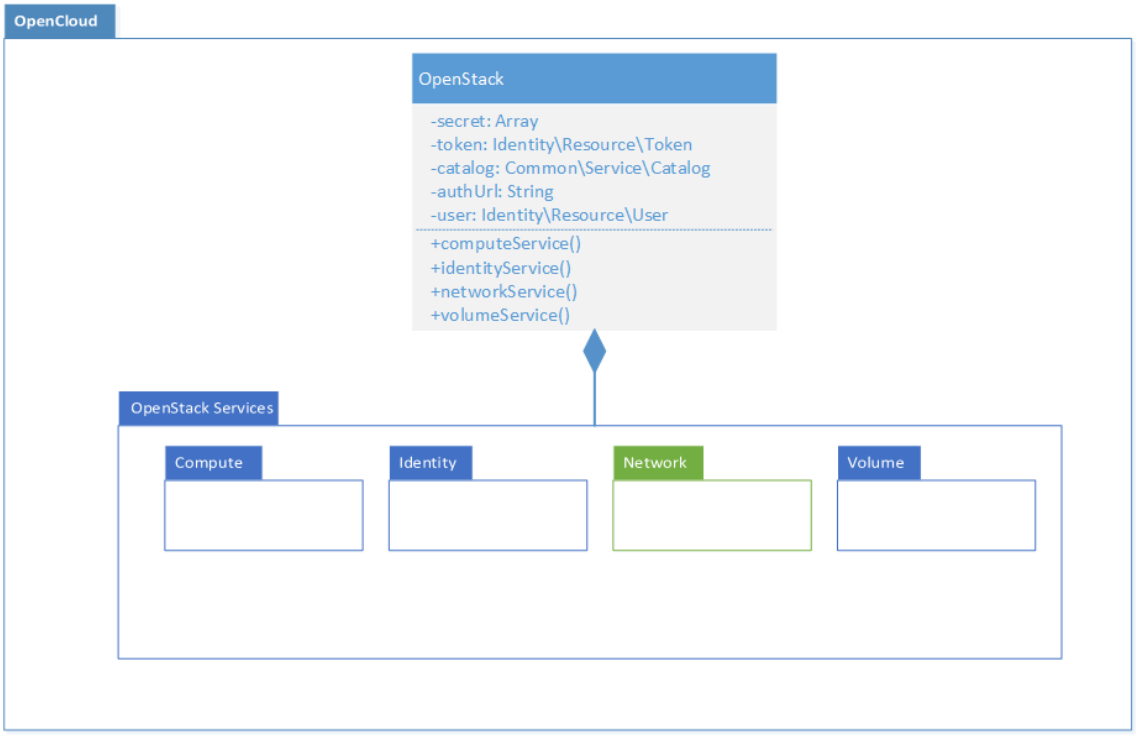


Figure 3.4: OpenCloud Class Diagram. Show the structure of the SDK.

The OpenStack services package (see Fig. 3.4) is composed of packages for each of the OpenStack services. Since this SDK is still a work in progress, some changes had to be made in it. The Network package was still part of the Compute service and it was necessary to create this part of the SDK, which was made by following the standards of the packages for the other services (see Fig. 3.5). The packages for each services follow the structural architecture, i.e., they have its own namespace (generally, the service name) and in the inside we find a Reources package and a service class. The creation of the Network package has also involved some changes in the resources from the Resource package of the Compute service, namely in the Server resource. These changes involved rebuilding the method to create the JSON object used to create new Server resources so that it would use this new package to get the network information. Finally, we also needed to have a method to get access to the network service from the OpenStack class.

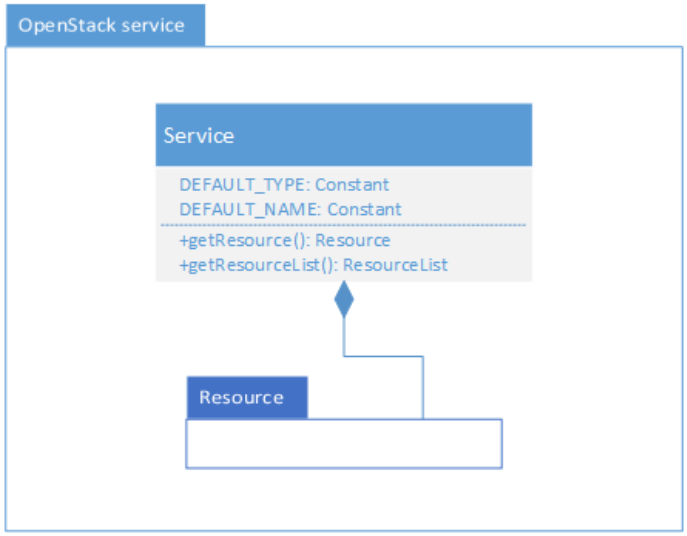


Figure 3.5: OpenStack service: generic representation of the services packages of the SDK.

3.3.3.1 Network Service

The addition of the network service to the PHP-OpenCloud SDK involved the creation of the Resource package which includes the classes for each of the resources of the network service and the creation of the service that gives access to these resources. In order to do that, a package that follows the structure of Fig. 3.6 was created. The Service class is used to get access to specific resources or to create new resources and it also implements methods to retrieve a collection of specific resources.

The classes which represent the state of the resources follows a common approach. They extend a class (i.e., PersistentObject) present in the Common package of the OpenCloud SDK, which implements the common part to the basic resources of the OpenStack service. Then, in the classes of each of the resources, we just need to create the objects to store the data that are exchanged in the requests that are made to the cloud resources. These classes also store static information about the resources, i.e., the top-level JSON object name and the URI of the resource. Finally, we override the methods that are responsible for the actions to create, delete and update resources by passing the correct parameters that are necessary to operate each of the resources.

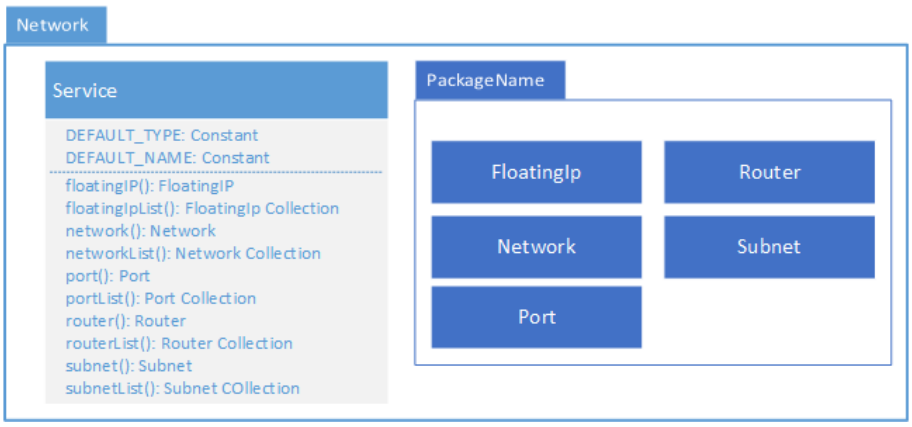


Figure 3.6: Network service package.

3.3.3.2 Identity Service

In this service we have added new features to the resources present in the Resource package (see Fig. 3.5), namely the User class, the Tenant class and the Role class. This new features allow us to create new Roles, Tenants and Users, get users list for a specific tenant and edit roles for a specific user (add and remove roles).

In the User class, we have added some aliases to the aliases array (this array is simply used to make names conversion between language variables and JSON field names), we have updated the JSON create method to fully comply with the API and added two methods which enable us to add and revoke user roles.

In the Tenant class we have added an array variable which has the keys required by the API to create new tenant resources and implements the method to create the JSON object. The method to create the JSON object consists of creating and returning a JSON object with a set of key-value peers. In this class, we have also created a new method to make requests to the API to get the list of users associated with the tenant.

In the Role class, we have created a method to create the JSON object and the array with the required key to create new Roles.

3.3.3.3 Volume Service

In this part of the SDK, we have added a class for a new resource provided by this service of OpenStack. This new resource is named Transfer and enables us to migrate the cinder volumes between tenants. The class consists of a set of attribute variables to get the state of the resource, i.e., the resource id, name and other information provided by the API. We have created the methods to create new resources, accept the transferences and to get a specific resource information (transfer information). Then, we have added a method in the Service class that gives us access to this new resource.

3.4 Summary

This chapter presents the implementation of a private cloud based in OpenStack and the tools used to manage OpenStack services. For the construction of the private cloud, we needed to have into account the available hardware, this way, some of the services that should use dedicated nodes were configured in the same nodes. Due to OpenStack being a project in development, the existent SDKs end up not implementing some of the last features available. This way, we needed to extend PHP-OpenCloud SDK to use these new features. For this purpose, we have added Neutron services to the SDK, as well as functionalities for the Volume and Identity service.

In the next chapter, we present the requirements analysis, then the platform architecture and finally, the design of the platform.

Chapter 4

CSCLab

In this chapter we present the CSCLab platform. CSCLab is a platform that allows the creation of laboratories by professors. The laboratories and all of its resources (e.g.,VMs) are created using cloud resources. These laboratories are composed of a set of virtual machines –VM, one for each student of the lab class that is using the laboratory. Each VM is composed of an operating system and a set of software specific for the laboratory being taught.

The chapter is outlined as follow: in the next section, the Requirements analysis section, we present the requirements for the CSCLab platform; the following section is the system architecture section, where we describe the architecture used to implement the platform; the third section presents the design of the system, starting by the design of a set of use cases for the platform, passing by the description of the static structure of the platform and finally, presenting the interactive part of the platform.

4.1 Requirements analysis

In this section it is made an analysis of the requirements of the CSCLab platform. These requirements represent the functions that we must implement in the platform.

In the first subsection we describe the base units of the platform, the Image, a Laboratory and a Persistent Laboratory.

4.1.1 Definition of concepts

- **Image:** is a resource that can be used to instantiate virtual machines. It includes an operating system, a set of software resources and a specific configuration.
- **Laboratory:** defines a set of resources related to the same domain. These resources include the virtual machines connected through a common network, a pre-defined image and a set of configurations.
- **Persistent laboratory:** it is a laboratory for which the duration time-line is undefined, i.e., the laboratory will exist till someone deletes it.

4.1.2 Requirements

1. The laboratory resources are cloud resources.
2. A laboratory is a set of virtual machines that use the same internal networks.
3. A laboratory must have an image assigned to it (the image is then used to create the laboratory VMs).
4. A laboratory must have a set of associated configurations.

   (a) Must be possible to define the type of resources used, i.e., we must be able to define multiple flavors. (A flavor is defined by the amount of RAM to use, disk space and virtual CPUs).

5. A laboratory must be a one time laboratory or a persistent one.
   When a laboratory is created, the professors may choose to create a laboratory that is deleted at the end of the class.
6. A virtual machine is composed of the base image for the laboratory.
   To create a new VM, a image specific for the laboratory in question is used to provision it. This image is chosen at laboratory creation time.
7. A virtual machine must be able to receive personalization files at booting time.
   These files may be configuration or data files that are injected to the VM.
8. The platform must have a catalog of pre-defined images.
   This catalog is composed of a set of images that can be used as a base for the creation of customized images or to simply provision new VMs.
9. The platform must enable the personalization of the provided images.
10. The platform must have a software repository.
    The software repository is composed of the software name, its license information and source.
11. The platform must have a priority system that is able to distinguish between three user types.
    (a) The platform must have a administrator user type.
    (b) The platform must have a lab manager user type. This is typicality the professor.
    (c) The platform must have a student user type.
12. The administrator is the top level user.
13. The administrator is responsible by managing the cloud resources, including other users.
14. The administrator must be able to delete persistent laboratories.
15. The administrator must be able to add and remove images from the image catalog.
16. The administrator must be able to manage the software repository.
17. The lab manager must be able to create and manage laboratories.
18. The lab manager must be able to accept the students that subscribe to classes associated with specific laboratories.
19. The lab manager must be able to create, update or delete virtual machines from the laboratories.
20. The lab manager must have access to all the machines of the laboratory.
21. The students must be able to subscribe to specific classes.
22. The students must be able to use virtual machines from the subscribed laboratories.
23. The student must be provided with cloud storage resources.
24. The student must be able to associate storage resources to virtual machines.

4.2 System architecture

Our system is client-server application and is implemented following a Model-View-Controller – MVC – pattern.

The MVC consists of three interconnected parts, separating the representation of information in the system from the way this information is presented to the user. Also, this model allows us to work with the different parts of the application individually.

The client-server model is a model used to structure distributed applications which enables the division of workloads and tasks between a provider (the server) and the clients.

The architecture of our system consists in the division of the different components of the MVC model into the two levels of the client-server model. In this case, the Model and Controller tasks run in the server while the View components of MVC runs in the client side. Our platform follows the architecture presented in Fig. 4.1.

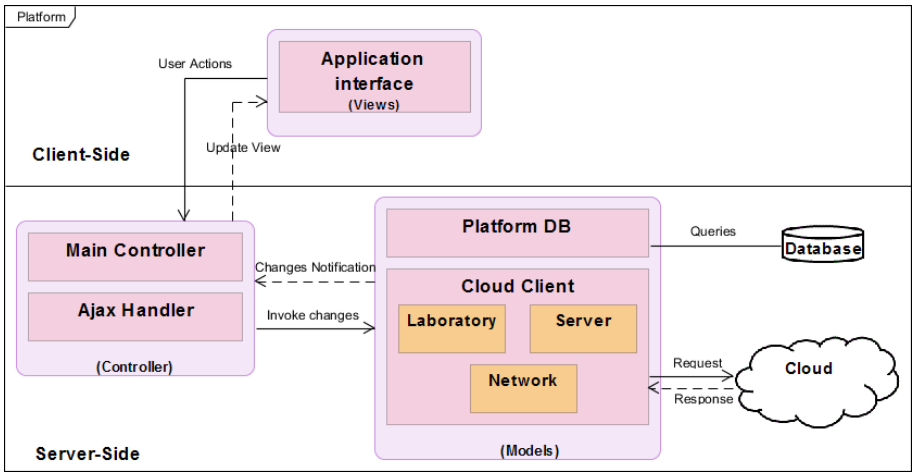


Figure 4.1: System architecture for the CSCLab platform.

The system is composed of a set of components. We have the views composing the application interface, which can be accessed in the client-side through a Web browser. Next, we have the controllers of the system running in the server-side. This part of the system is used to connect the application interface and the models of the system. We also have the models running in the server-side and use them to access the data, logic and rules of the platform database and of the cloud infrastructure services.

The models have two distinct targets and for this reason they are divided into two sub-sets of models, The PlatformDB models and the Cloud models. The Cloud models comprise the model used to acquire communication with the cloud services and the models that are used for distinct services of the cloud infrastructure. We have "Cloud Client" model, which is used to acquire the initial connection with the cloud services. Then, we have "Laboratory", "Server" and "Network" which inherit the characteristics of the "Cloud Client" model and are used to access the different services of the cloud infrastructure. The "Laboratory" model is used to create new laboratories and to manage the laboratory users and their roles in it. The "Server" model is used to manage the computation services and the volume services. This model is used to manage the VMs of the laboratories, the volumes of the users and the attachments between VMs and volumes. The "Network" model is used to manage the network resources of the cloud infrastructure. To communicate and manage the platform database, we have the "PlatformDB" model.

Finally, the system has also a dedicated database for the platform and a cloud infrastructure. The communication with these two sub-systems is managed through the models of the platform.

4.3 System design

This section is used to describe the design of the platform. The design is an intermediate stage between the system architecture and the implementation of the system.

For the design of the system we use a standard modelling language, Unified Modelling language - UML - which is used in software engineering to visualize the design of the system.

We start by describing the design of the database, which is made using a Extended Entity-Relationship – EER – conceptual model. Then, we present the use cases for the system, which describe the interaction between the actors (users of the system) and the system. Next, we present a number of class diagrams that describe the components of the system. These diagrams present the attributes used in each class and the methods used to make operations in the cloud services and in the database of the platform. And finally, we use a number of sequence diagrams to describe the interaction between each part of the system in an ordered way.

4.3.1 Database conceptual model

The platform needs a way to manage and store all data that it needs. This way, we started by looking at the platform requirements in order to get a concise idea of all the information that the platform might generate. It was also needed to analyse the process through which the cloud infrastructure services manages its data.

We have analysed how the data was being managed in the cloud infrastructure databases. This process was needed because our platform uses these cloud resources provided by the cloud services, so we needed to know how the resources were identified and where was the information about resources stored. In the case of cloud resources we needed to know information about a specific set of entities. We needed to know information about the users of the platform, e.g., authentication data and relation to tenants and services of the cloud. How the cloud resources where related with each other.

By looking at the requirements we started enumerating the entities that were involved. These first entities to be referenced were laboratories, images, software, users and classes. Now, we had a point from where to structure the database for the platform.

Finally, we got to a conceptual model (see Fig. 4.2) for our database that meets the requirements of our platform. This model uses entities from the cloud infrastructure databases in order to avoid possible ambiguity problems that might arise. This way, the resources that we use in our database are also uniquely identified in the databases from the cloud infrastructure. These entities are prefixed with names associated with the cloud infrastructure services (e.g, neutron.subnets, keystone.project and cinder.volumes). Still, we must point out that these entities are not actually created in our database. They exist here for the simple fact that the relation between these entities actually exists, nonetheless we get information about it by using the cloud services API.

In this model the Laboratory entity is a representation of the laboratory being created in the cloud. It stores information about its creation (e.g., lab manger, image for the VMs and creation

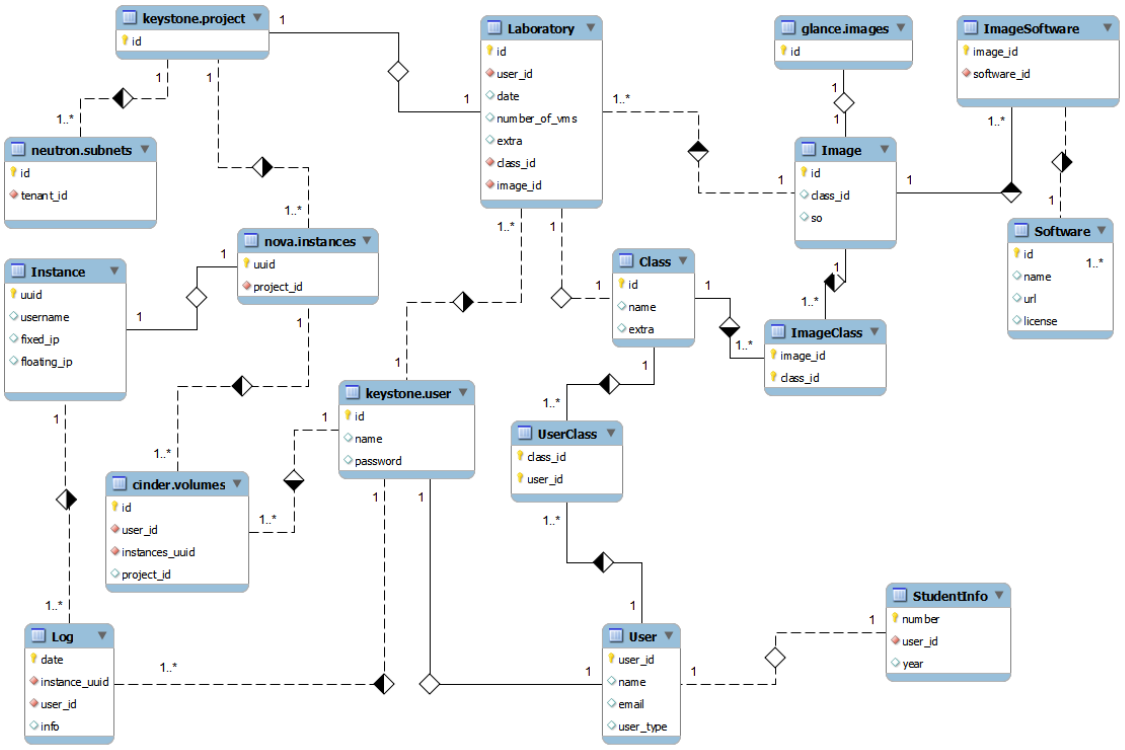


Figure 4.2: Extended entity-relationship database model for the platform database.

date) and has a relationship with entities from the cloud (keystone.project) for identification of laboratory resources in the cloud.

The Image entity make the connection between the images existent in the cloud services (glance.images) and the list of software (Software entity) for each instance of Image. The relationship between Image and Software is a many to many relationship represented by the ImageSoftware entity. The instances of Image may also be related to the classes they are designated through the relationship between these entities, i.e., Image and Class entity. Since this is a many to many relationship, it is represented through the ImageClass entity.

The Software entity is used to store the information about each software of the platform. In this case, the instance acts as software repository since it stores information about where it is as well as license information. The instances of this instance may be associated with instances of the Image entity through the ImageSoftware entity.

We use the Class entity to store information about each of the classes that are registered in the platform. This entity can be referenced by the Image and User entities since there are relationships between these entities. As the relationship between the Class entity and the User and Image entities is a many to many relationship, we created a set of new entities. This way, the relationship between the Class and Image entities is represented by the ImageClass entity and the relationship between Class and User entities is represented by the UserClass entity.

The Log entity is used to store information about actions taken in the virtual machines related to the platform. The instances of this entity consist of a message, references to the virtual machine (through the relationship with Instance entity) and references to the users of the platform (through the relationship with keystone.user entity).

In the instances of the Instance entity we store information which is used to map instances to users, through its relationship with keystone.user and to network addresses associated with the virtual machines, through the nova.instances entity.

4.3.2 CSCLab use cases

The use cases are a useful resource to represent the interaction between actors (e.g., a real person or a computer system) and the system (i.e., the platform). These diagrams provide us important information about the views and models to implement.

We have modeled a set of use cases (Tab. 4.1) to describe the system functionalities and for these purpose we started by identifying the actors of the system. In table 4.2 we describe these actors.

Table 4.1: Use cases used to describe the system.

Entity	Description
Log in System	This use case presents the log in process to access platform resources.
Laboratory management system	This use case presents the functionalities available to the Lab Managers.
Administration system	Use case used to describe the functionalities for the administration of the platform.
Students system	Use case used to represent the functionalities available to students.

Table 4.2: Actors used in the use cases that describe the platform.

Entity	Description
User	Represents a person which is not yet authenticated in the platform.
Administrator	Is an authenticated user with administrative rights.
Lab Manager	Is an authenticated user that can create and manage laboratories.
Student	Represents an authenticated user with limit access to the platform. May access storage resources and virtual machines for subscribed laboratories.
Database	Is a computer system which stores information about the system.
Cloud System	Is a computer system that represents the cloud infrastructure services.

The use cases identify the services that we need to provide to the users of the system. In our case, the services that are provided to platform users are made through a set of views which compose the interface of the application. As for the computer systems, the services are provided through a set of models.

We use models (MVC component) for the computer system because we need a way to preserve its state throughout the system. The models are used for this purpose, to manage application data, logic and rules. As we have two actors representing computer systems, we need at least two models. One for the Database actor and the other for the Cloud System actor.

The interactions between each entity of the system are managed using controllers (MVC component). Next, we describe the four main use cases of the platform, "Log in System", "Laboratory management system", "Administration system" and the "Students system". For each use case, we present its components and the relationship between them and the actors. The relationships between users and the components are described through a set of views for the application interface. The relationships between the computer systems and components are made through functionalities of the models of the platform.

4.3.2.1 Log in system

When regular users access the platform they are faced with the scenario presented in Fig. 4.3. This use case describes a scene where an unauthenticated user accesses the platform. It describes the functionalities available in the system, the interactions that occur and the actors that interact with the system. In table 4.3, we detail each entity involved in the use case.

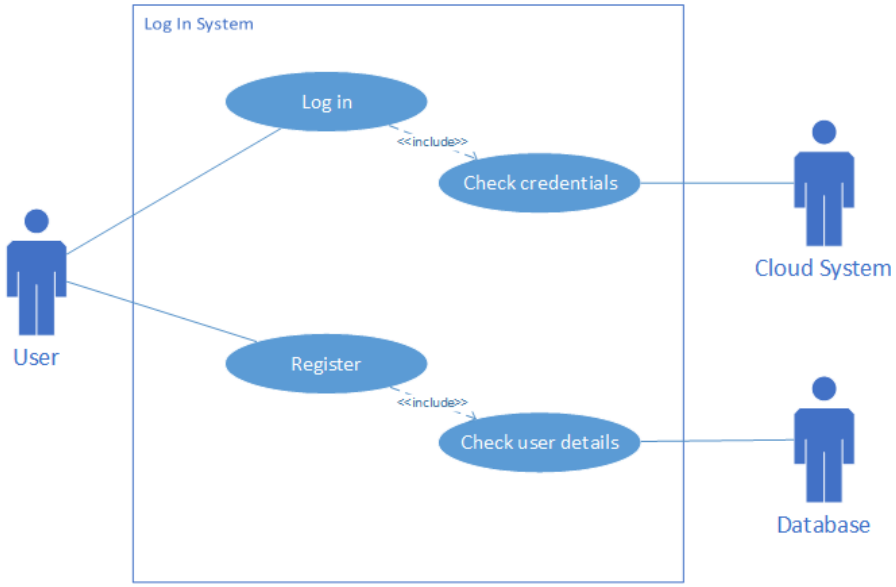


Figure 4.3: Log In system use case.

Table 4.3: Description of the entities involved in the Log In system.

Entity	Description
Log in	The user may use a set of credentials to log in into the platform.
Register	Allows the registration of new users into the platform.
Check credentials	Mechanism to check the validity of the user credentials.
Check user details	Verifies if the data entered by the user into the system is valid.

In this use case, a user interacts with two distinct entities. This means that we need to implement two views, one for the log in and the other for the registration process.

4.3.2.2 Laboratory management system

In the laboratory management system are represented the features involved in the management of the laboratories. The use case from Fig. 4.4 presents the design of the Laboratory management system. Then, in table 4.4, we describe the entities of the system.

The Lab Manager interacts directly with four entities of the platform. The interaction occurs in the views of the platform and in this case we need at list four views. This leads to the implementation of the views for the Lab Manager, which are the Lab Access View, the Create Lab view, the Notifications view (represented by the accept students feature) and the Statistics view.

4.3.2.3 Administration system

The use case represented in Fig. 4.5 describes the administration system. The use case presents the functionalities that this system needs to provide to administrators, as well as the inter-

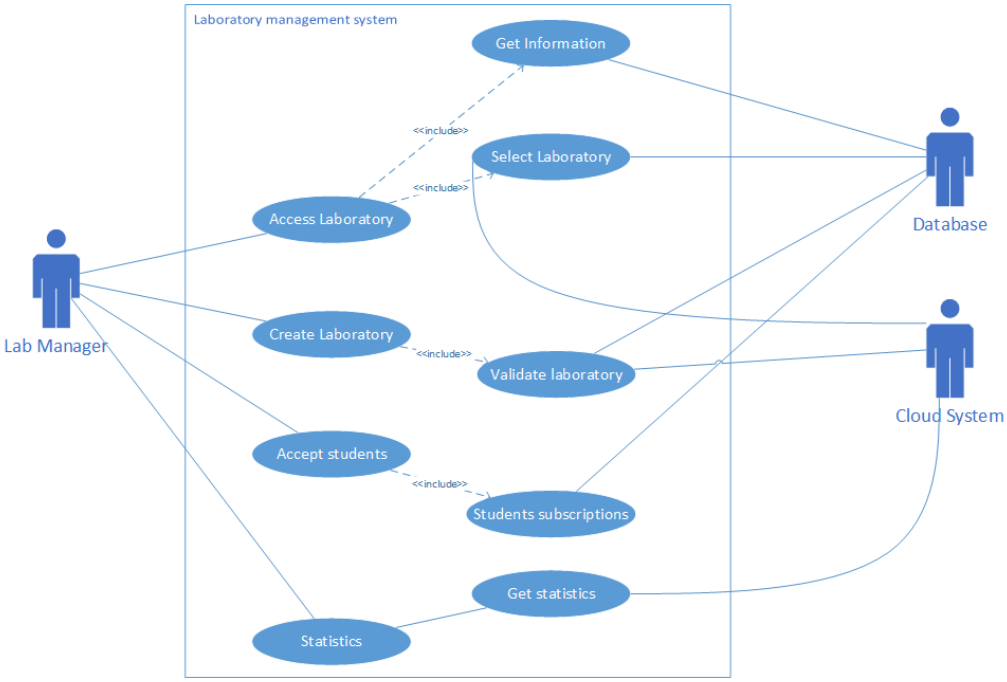


Figure 4.4: Use case for the laboratory management system.

Table 4.4: Description of the entities involved in the laboratory management system.

Entity	Description
Access Laboratory	The user has access to information about a specific laboratory.
Create Laboratory	The user can create a new laboratory.
Accept students	Presents notifications about students to be accepted to the laboratories.
Get information	Mechanism to get information about the selected laboratory.
Select laboratory	Mechanism used to select the laboratory to be used.
Validate laboratory	This mechanism is used to verify if the information introduced for the new laboratory is valid.
Students subscriptions	Gets the information about users in a wait list to be accepted to the subscribed laboratories.
Statistics	Presents usage of cloud resources to the users of the platform.
Get statistics	Mechanism to retrieve cloud resources statistics.

actions that happen in the system. Then, in table 4.5, we describe all the entities of the administration system.

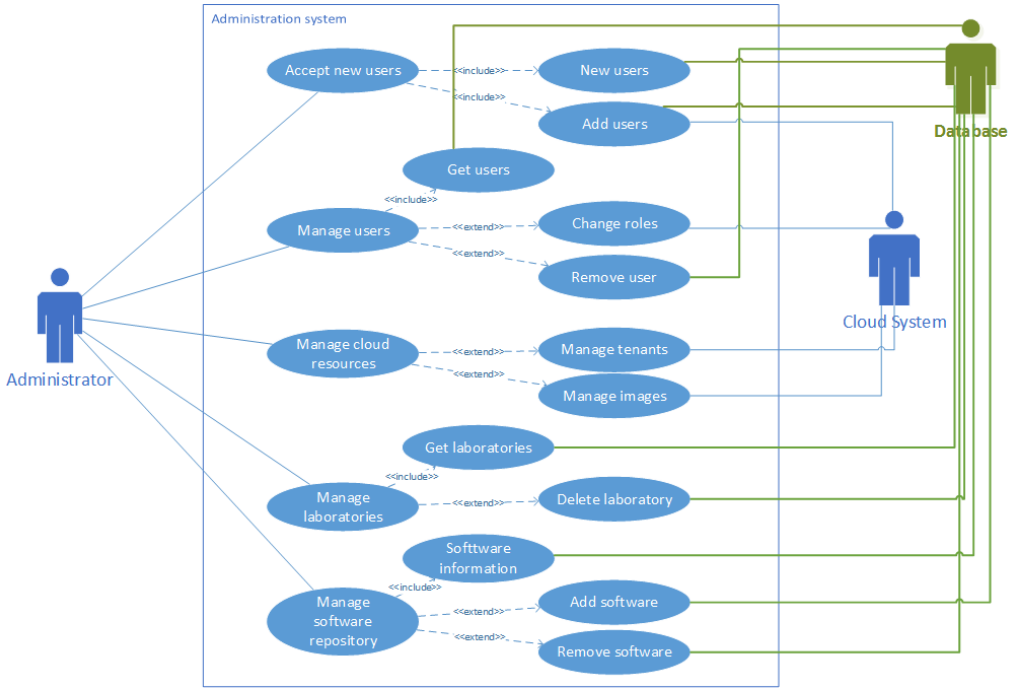


Figure 4.5: Use case for the administration system.

By the analysis of this use case we can understand the functionalities that we need to implement in the user interface. In this case, the views for the system need to implement five different functionalities, which imply implementing five views.

4.3.2.4 Students system

The students system use case, Fig. 4.6, represents the functionalities that the system must implement for the students. In the use case we present the entities involved in the system, as well as the actors that use the system. The description of the entities for this use case can be checked in table 4.6.

The interactions between the student and the platform are possible through a set of views that make the user interface. As for the functionalities that use the computer systems, these are implemented in the models of the platform.

4.3.3 System structure: class diagrams

The class diagrams are used in software engineering to describe the structure of the system. Next, we describe the structure of the system as a whole, then we describe the structure of the system models and finally, we present the structure of the system controllers.

The diagram from Fig. 4.7 presents the structure of the system as a whole, as well as the relationships between each component of the system. All models and controllers of the system are an instance of CI, the main class of the system.
CI is an instance of the platform core and is used to make the connection between each element of the system. The CI_Model is a class that implements the main functionalities of the models that make the connection with the remaining components of the system. The CI_Controller

Table 4.5: Description of the entities involved in the administration system use case.

Entity	Description
Accept new users	Presents notifications about the users to be accepted and registered in the platform.
Manage users	Allows the administrator to manage the permissions of each user in the platform.
Manage cloud resources	Presents information about cloud resources and allow its management.
Manage laboratories	Allows the user to manage the existent laboratories.
Manage software repository	Area where the administrator can manage the list of software existent in the platform, i.e, add, edit or remove a specific software.
Get users	This mechanism is used to retrieve the list of users registered in the platform and information about each of them.
Get laboratories	Mechanism used to retrieve information about each of the laboratories existent in the platform.
Software information	Mechanism to retrieve the information about the software existent in the repository.
New users	Is a mechanism to get a list of the users in a wait list to be accepted and registered in the platform.
Add users	Mechanism to effectively and new users in the platform.
Change roles	Mechanism to change roles of the user in the platform.
Remove user	Mechanism used to remove specific users from the platform.
Manage tenants	Allows the management of the tenants existent in the cloud infrastructure.
Manage images	Allows the maintenance of the image catalog provided by the platform
Delete laboratory	Mechanist to remove laboratories from the platform.
Add software	Mechanism that adds information about new software in the system.
Remove software	Mechanism used to remove software from the repository.

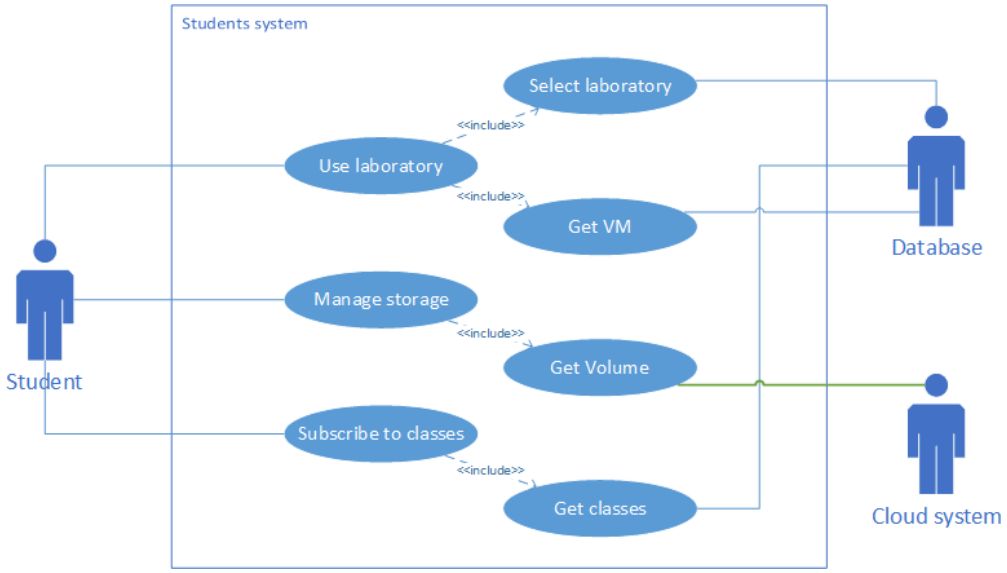


Figure 4.6: Students system.

Table 4.6: Description of the entities involved in the students system.

Entity	Description
Use laboratory	System area where a student has access to the resources of the laboratory.
Manage storage	Area where the student can manage its cloud storage resources.
Subscribe to classes	Allows a student to subscribe to classes existent in the platform.
Select laboratory	Mechanism to select the laboratory to be used.
Get VM	Mechanism to get information about the virtual machine from a specified laboratory that the student can use.
Get Volume	This mechanism gets information about the storage volume associated with the student.
Get classes	Mechanism to get classes that the student has not subscribed yet.

is the class that implements functions to make the connection between the controllers of the platform and the core of the platform.

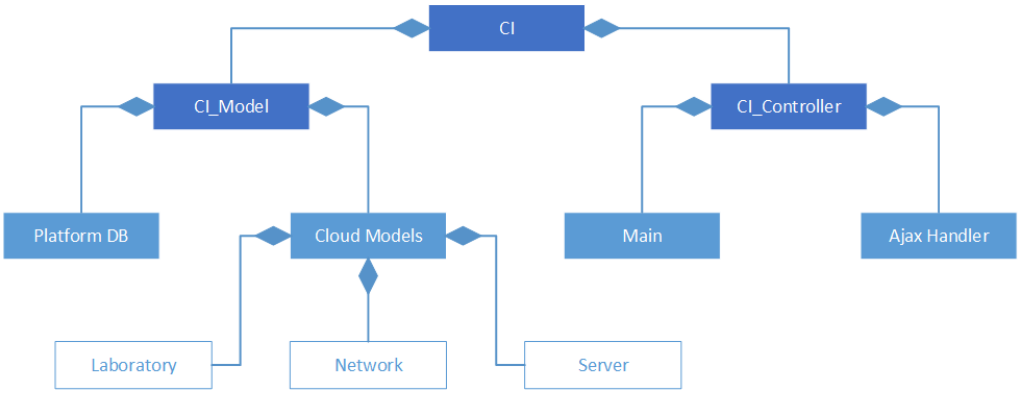


Figure 4.7: Class diagram for the CSCLab platform. Represents the static structure of the system.

4.3.3.1 Models

Our platform is composed of two sets of models, the Platform DB models and the Cloud Models. The first set is used for the application data, rules and logic of the platform database. And the Cloud Models are used to preserve the application data, logic and rules for the services of the cloud infrastructure.

The Cloud Models package, Fig. 4.8, contains a class Cloud_Client which has the necessary information to use the cloud infrastructure services and a set of functions. The class uses a set of constants to define services, provides a OpenStack object to communicate with the cloud infrastructure and a set of objects to communicate with the cloud services (compute service, network service and volume service). To use the cloud services, we need to have an authenticated used and for this an authentication method is implemented in the package. This method verifies user credentials and then, if the user is valid, the credentials are saved to cache. We have also a method to change between tenants (i.e., laboratory instances), to avoid the need for new authentication when changing between tenants (this only works if the user has access to the required tenant). The class implements a function that imports cached credentials that can be used when requesting a new OpenStack client and a method to require an administrative OpenStack client object. A method to change the authentication endpoint for the cloud service is also provided. Then, the class implements methods to set the clients

for the cloud services. To get these clients, the OpenStack object is used to get the required service (volume, network or compute).

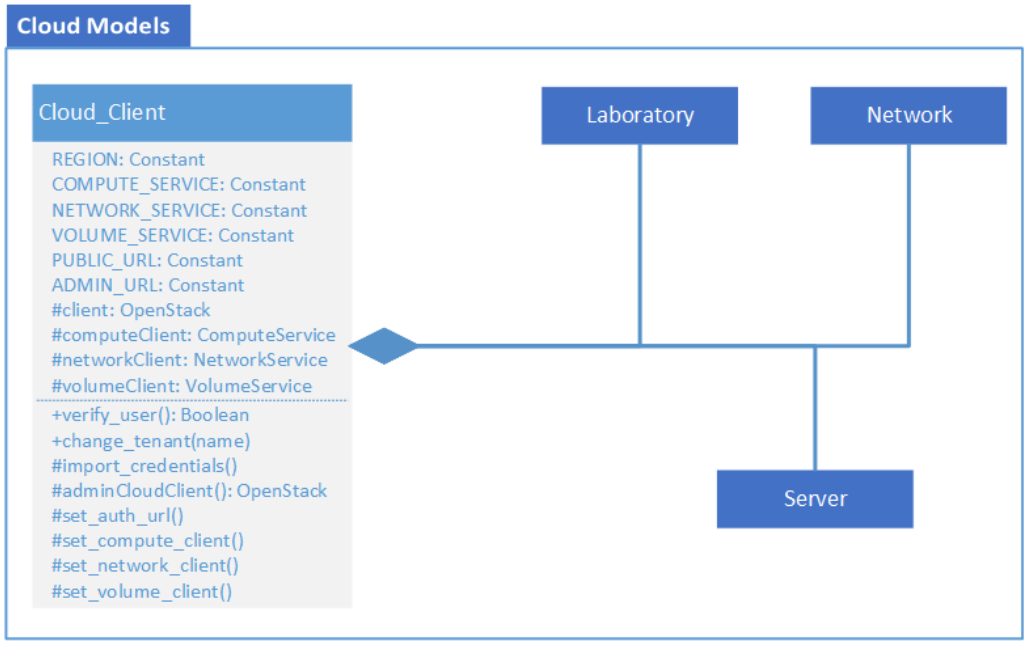


Figure 4.8: Class diagram for the Cloud Models package. Represents the static structure of the cloud models.

The package includes also the Laboratory class, the Network class and the Server class. These classes are part of a whole, they only exist as long as the Cloud_Client class exists. They are used to describe the static structure of part of the system that represents each service of the cloud infrastructure.

The Laboratory class, Fig. 4.9, has an object to represent the Identity service of the cloud infrastructure and implements a set of methods to operate identity resources of the cloud. The class has methods to get specific users, roles and tenants or a list of the referred resources. The class also implements methods to create, edit or remove cloud resources. It also implements methods to assign or remove roles of users in specific tenants, by assigning a role we are basically assigning the user to the tenant. If we were to remove all roles for a user in a specific tenant, it would be like removing the user from the tenant. These are two important methods to assign or remove users from laboratories. The class implements also methods to retrieve resources associated with a specific tenant (images and flavors). The class allows us to use the methods using a different identification (authenticated tenant) without the need for manually making new authentication against the cloud Identity service by simply passing the tenant name for authentication (an optional parameter of most of the methods).

The network class, Fig. 4.10, is used to represent the static structure of the system for the implementation of the network features. It has a set of methods to work with the many different resources of the cloud infrastructure network service (e.g., floating IPs, networks, subnets, routers and interfaces). The class implements methods to create, update and delete each of the network resources. It implements methods to associate routers with networks through the methods to add or remove router interfaces. Finally, the class has also a method to create a fully functional network for a laboratory and a method to completely remove a laboratory network.

The Server class, Fig. 4.11, implements methods to work with two completely different

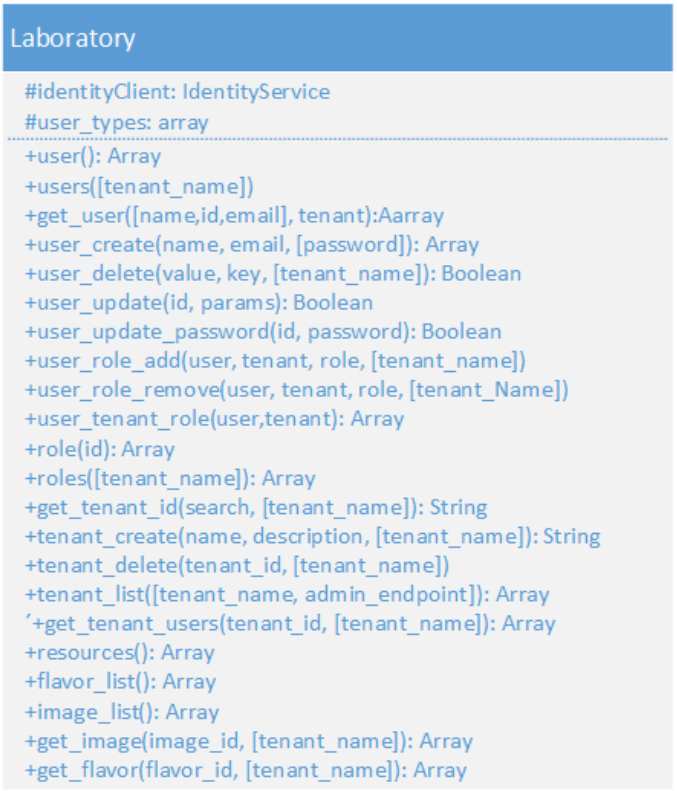


Figure 4.9: Class diagram for the Laboratory class.

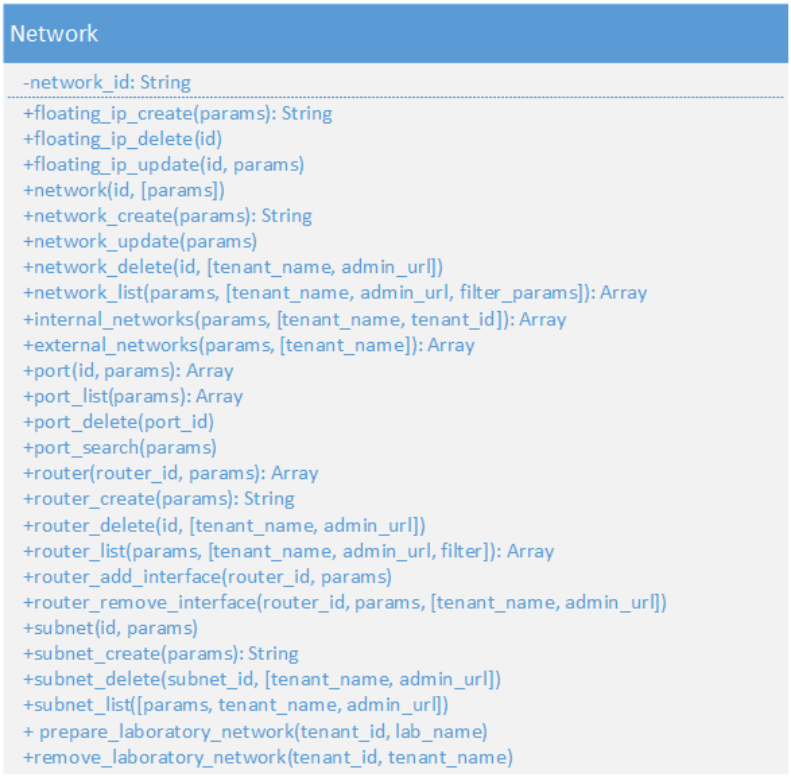


Figure 4.10: Class diagram for the Network class.

types of resources, i.e., server instances and volumes (from the storage service). As these two resources depend on each other, they are both operated using this class. The class implements methods to create, edit, reboot or terminate server instances. It also implements methods to get specific information about the server instances or a list of server instances. To work with the volumes, the class implements methods to create, update or delete this type of resource. Then, it implements methods to attach or detach server instances and volumes. The class, also implements methods to transfer and accept transferences of the resource (volumes) between tenants, as well as a method to transfer the resource to the main tenant of a user (i.e., to the tenant the user has always access, e.g., for the students is the students tenant).

Figure 4.11: Class diagram for the Server class.

The Platform DB package is composed of only one class as seen in Fig. 4.12, the Platform_db class. This class is used to communicate with the platform database. This class implements methods to retrieve the information necessary for the correct execution of the platform, get list of classes, get list of subscriptions and pre-subscriptions (subscription wait-list), get information about images and software, get information about the laboratories and about the users (the new user method is used to get new pre-registrations to the platform and the accept users method is used to make the pre-registration permanent). The class also implements methods to insert or update data in specific tables, as well as a method to delete specific information from specific tables.

4.3.3.2 Controllers

Our platform is composed by two controllers, the Main controller and the Ajax Handler controller. These controllers are used to map interactions between the users and the models and to update the view having for base the user interactions with the platform.

The Main controller, Fig. 4.13, is the main controller used in the platform. This controller

Figure 4.12: Class diagram for the Platform DB package.

has two attributes, arrays of data, to distinguish the tenants of the cloud Identity services that identify the user types (e.g, student, lab-manager or administrator) and the cloud tenants that identify cloud services (e.g., neutron, nova and cinder). The Main controller implements an index method, which by default presents the main view to the users (e.g., registration view, lab management view or administration view). It has also methods that map users to different views of the user interface (e.g., administration, laboratory_create, laboratory_information methods). Implements the methods to receive the actions of the forms, i.e, the forms submission are mapped to these methods (e.g., laboratory_create_form and add_class_form methods. Then, the class has some private methods, which in a posterior revision of the system should be implemented in a specific model. These methods are used to update the login session (set_user_type) with the type of user that is logged in, get a representation of the information to be displayed about a specific instance in the required way (get_instance_params) and the method to assign a new instance to a student. The controller has also a method to prepare the laboratory, which is constantly used to verify if the laboratory resources are correctly configured (e.g., network configuration).

The Ajax Handler controller, Fig. 4.14, has the methods that are called using Ajax from the user interface (the views). This controller is used to make partial refreshes to the views of the interface on user specific actions. It has methods to work with server instances, the virtual machines, to create, edit, reboot or terminate specific instances. Implements methods to update the view components (e.g., flexi_vmachine and flexi_notifications) and to work with cloud storage volumes (e.g., volume_information and volume_create). The class has also methods to retrieve laboratory information (e.g., verify_laboratory_name to verify the name used to create a new laboratory, laboratory_information and laboratory_users_list).

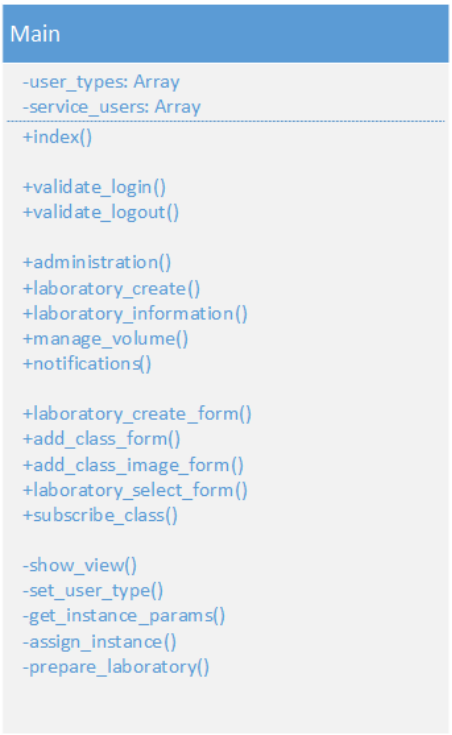


Figure 4.13: Class diagram for the Main controller class.

Figure 4.14: Class diagram for the Ajax Handler controller class.

4.3.4 Sequence diagrams

The sequence diagrams are used for modelling the interaction between processes in a time sequence. We have modelled the interaction between processes for the four use cases presented above (the Log in use case, the Students use case, the Lab Management use case and the Administration use case).

For the Log in use case, we have created the sequence diagram from Fig. 4.15. This use case has two main features, the login feature and the registration feature, which are separated using an alternative fragment. The login process starts by the submission of the credentials to the platform, which are then checked in the cloud identity service. If the credentials are valid, the user is redirected to a user view. By the other side, if the credentials are not valid, an error is shown to the user. The registration process starts by the submission of the registration data to the platform. If the submitted data is valid, the user is redirected to the login view, else, the user is shown the error information.

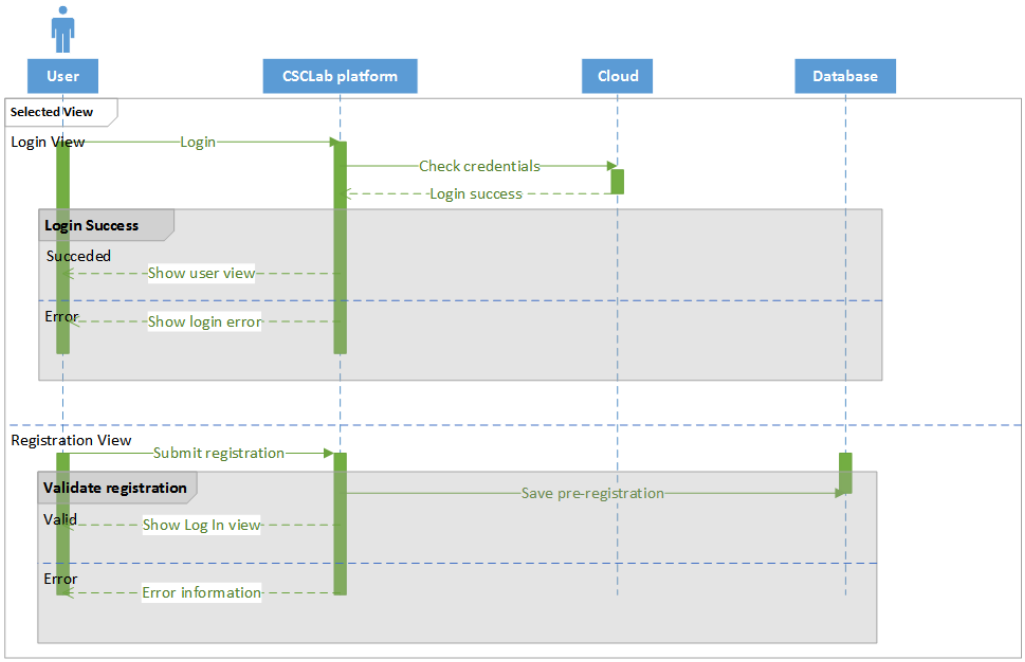


Figure 4.15: Sequence diagram for the Log In use case.

The Students sequence diagram, Fig. 4.16, shows the order by which the interactions between each process of the platform occur. When the student accesses the platform, a list of the associated laboratories is retrieved from the cloud services. Then, a set of alternative processes can happen, depending on the selected view. The use of a laboratory starts by the selection of the desired laboratory. When the laboratory is selected, the platform requests a new authentication token to the cloud services and updates the credentials stored in the platform (the authentication token id). Then, the platform verifies if the student has a virtual machine assigned and if so, then it retrieves the information about the VM from the cloud services. If the user has no VM assigned, then, a new VM is created and it is assigned to the student. The Management of the storage process is started by the verification of the storage volume. If the volume exists, then, information about the volume is retrieved from the platform and the student can then attach or detach the volume from a VM. If the students have no storage volume, then, they can create it. When a new volume is created, the view is updated.

For the Lab Management use case, we have created a set of sequence diagrams to include

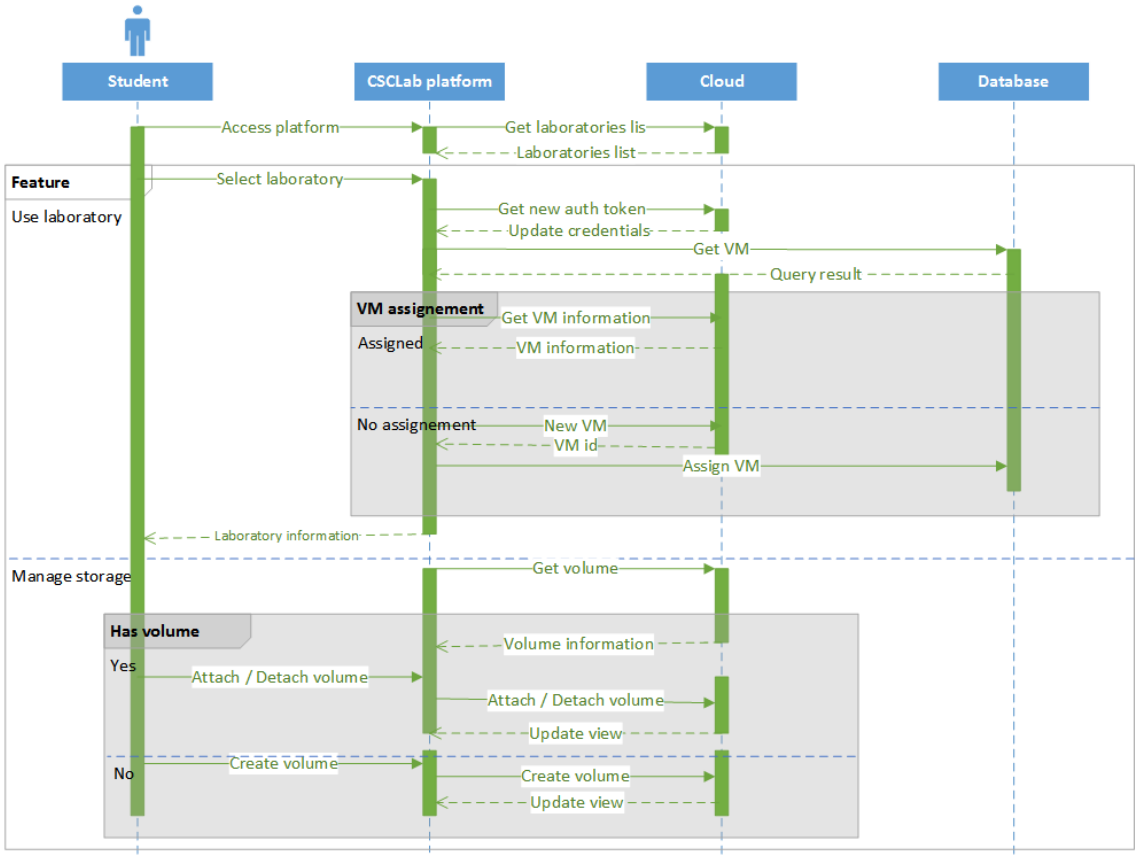


Figure 4.16: Sequence diagram for the students use case.

each feature presented to the Lab Manager. We have created the diagrams from Fig. 4.17 and 4.18. In the first figure, we present the interactions that happen when accessing a laboratory and when creating a laboratory. In the second diagram, we include the interactions that happen when accepting students and when retrieving cloud statistics. All interactions start by the access to the platform, which also retrieve a list of laboratories associated with the student. To access a laboratory, the Lab Manager must first select a laboratory. In the sequence of the selection, a new authentication token is generated and the information about the resources of the laboratory is retrieved from both, the cloud services and the database. The Lab Manager can then manage the VMs of the laboratory (e.g., terminate or create a VM). The creation of a laboratory starts by filling of a set of fields and new information is requested as needed (e.g., images list and image information). The creation of the laboratory is finished when the laboratory details are saved to the database and the laboratory resources are created in the cloud.

In the second sequence diagram of the Lab Manager, Fig. 4.18, we present the interactions between the processes and the Lab Manager to accept new student subscriptions and to present statistics to the Lab Manager. To accept new students, the Lab Managers first need to get a list of the subscriptions from the database, then they can accept or decline the subscriptions. If the subscriptions are accepted, then we add the students to the laboratories associated with a class, else, we remove the subscription from the database and the view gets then updated. The statistics option starts by getting statistics form the cloud services and then presenting the statistics in the view.

The Administrator sequence diagram, Fig. 4.19, describes the use case for the administration system in a order sequence of interactions that occur. The platform starts by gathering

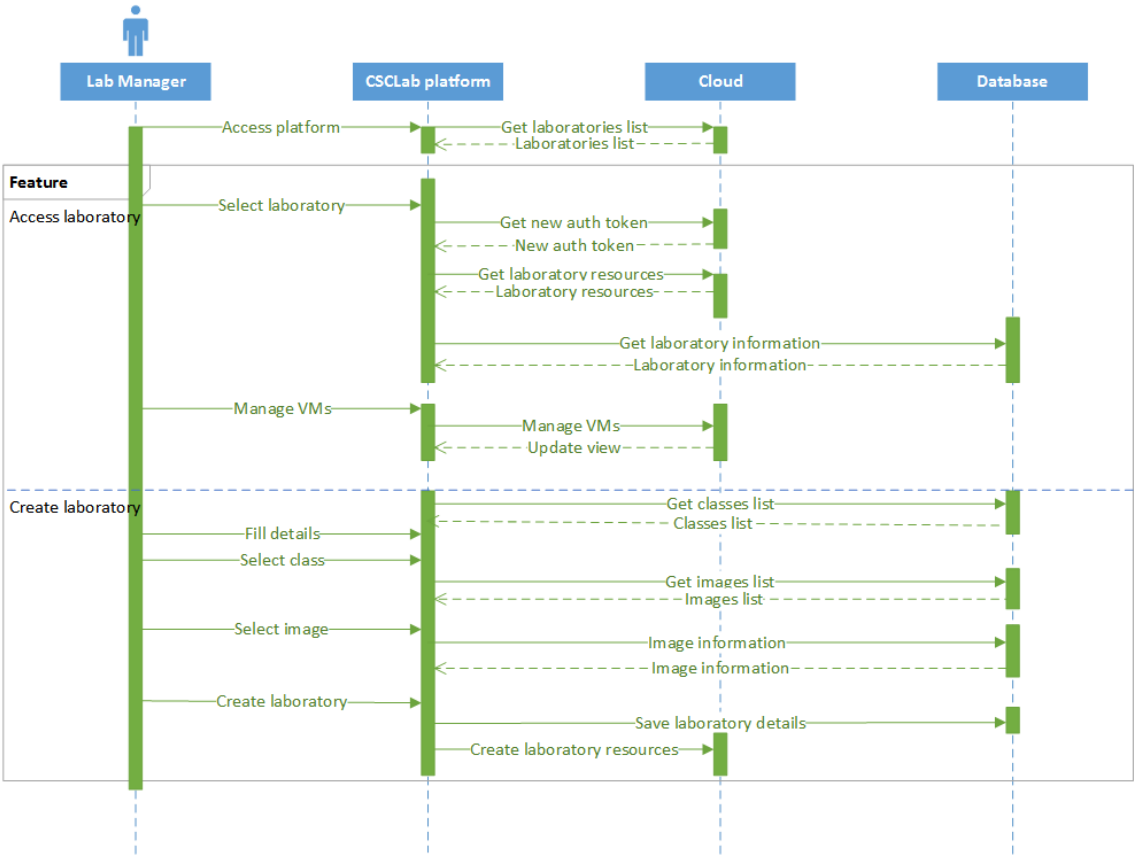


Figure 4.17: Sequence diagram for the laboratory management system. Sequence diagram part 1.

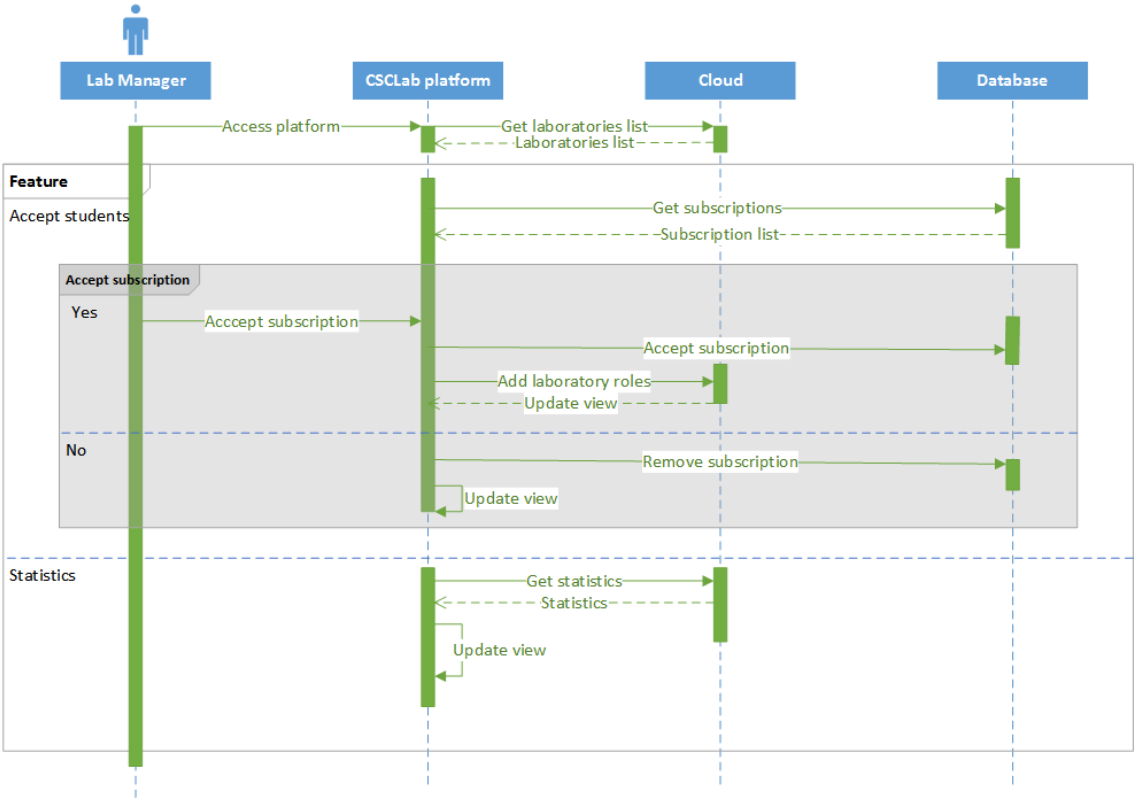


Figure 4.18: Sequence diagram for the laboratory management system. Sequence diagram part 2.

information from the platform database and from the cloud services. When all informations is retrieved, the user can then interact with the platform to accept new users, remove users from the platform, manage laboratory resources or manage the software repository.

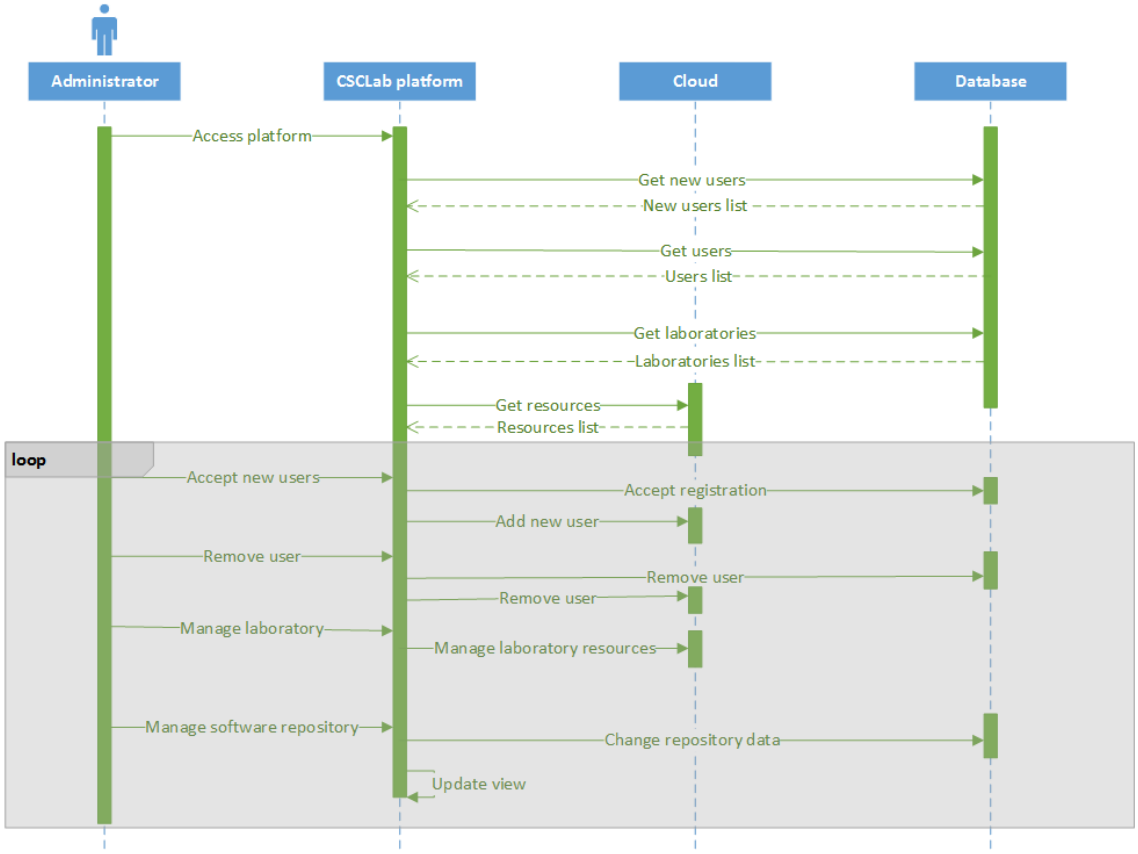


Figure 4.19: Website wireframe for Student and Lab Manger users.

4.4 Summary

In this chapter we present the CSCLab platform. We present a requirements analysis which presents a list of the features that must be implemented. These requirements describe the features that a platform for the creation of virtual computing laboratories must implement. Then, we describe the architecture of the platform and its components. This is a MVC architecture and is composed of a set of views that represent the application interface, the platform controllers and models. And finally, we have the platform design. By using this design we can implement a platform that may use different cloud infrastructures, i.e.,the design is valid for other implementations. To make the platform use other cloud infrastructures, we just need to work with the components that communicate with the cloud services. In this case, we just need to work with the platform models.

In chapter 5, we describe the development of a platform prototype. This prototype follows the platform architecture and the design presented in this chapter using an OpenStack implementation.

Chapter 5

CSCLab Implementation

In this chapter we describe the implementation of the CSCLab platform using OpenStack. This implementation is a prototype of the platform that uses an OpenStack based cloud and is used to validate the architecture and design of the platform.

The chapter starts by the description of the PHP framework used and then describes the implementation of the platform components. We describe the implementation of the views, then we describe the platform controllers and models. We start by the implementation of the controllers and at the end, we describe the implementation of the platform models.

5.1 MVC Framework

For the implementation of the platform we have used a MVC framework for PHP. This framework is CodeIgniter [Ell14] and is a framework with a big support from the PHP community. The use of this framework allows us to develop the software solution by following the MVC architecture, using a robust structure. The configuration files for the framework can be found in the config folder located inside the application folder.

The first thing we did using the framework was to auto-load some libraries (i.e., database, parser and form_validation) which provides us access to tools (i.e., objects and methods) to control the database and forms. The parser is used to parse data that can be passed from controllers to views. This is done in the *autoload* file provided by the framework.

Next, we have configured the access to the platform database in the provided *database configuration* file. The configurations for the database include database location, credentials, the database driver to use and cache configurations.

The following step was to configure some security options in the provided *config* file. In this file we have configured cross-site scripting - XSS - filtering and cross-site request forgery - CSRF - in order to provide some additional security to our platform.

5.2 Views (MVC)

The views are the front-end of the application, they create the user interface to which the users have access. These views are isolated components of the MVC model, i.e., they can be implemented independently of the remaining components of the platform. They do not depend on the back-end of the system, i.e., they are simply the means by which the user interacts with the system and are composed of information prepared for the user.

By following the design of the system, we can now implement the views of the system. We need to implement four sets of views, one for each of the systems designed in the use cases presented in sub-section 4.3.2:

- **Log in views:** these are the views presented to users with no session started in the platform.

- **Laboratory management views:** the views that are presented to the Lab Managers.
- **Administration views:** these views create the user interface for the administration system
- **Students views:** the views that are visible to the students.

For the implementation of the views we use HTML5 and CSS since they are supported by all major web browsers for both desktop and mobile platforms. In order to create a responsive interface, i.e., an interface adaptive to screen size, we use CSS3 Media Queries. This CSS feature allow us to control the style of the HTML web pages depending in the screen size details of the screen used to access the platform. The views use also some additional tools:

- **JavaScript:** is used to add some dynamic features to the views. In this case, we use it essentially to make asynchronous communications with the models of the platform.
- **jQuery:** this is a library that adds features to JavaScript.
- **FlexiGrid:** is a data grid used to create dynamic grids. It is composed of a set of CSS and JavaScript files. We can use this tool to make asynchronous connections, using Ajax, to load content.
- **Alertify:** is a JavaScript framework that allow us to create personalized dialog and notification windows.

We use JavaScript to automatically submit some information and trigger automatic actions, depending in other actions taken by the users. These actions are the automatic submission of some forms and updates to the views depending on changes in form elements. Then, by using Ajax, we make POST requests to submit and retrieve new information for the actions triggered. The POST requests all include a key-peer field in the data passed through the request that uses a CRSR token element for security reasons. This allows us to also make these Ajax requests using HTTPS.

FlexiGrid works by following a specific set of steps. First, we use a table tag with a specific ID in the web page. For this web page we need to include the FlexiGrid style sheet file (CSS file), the FlexiGrid JavaScript file (FlexiGrid engine) and a JavaScript file with the configurations specific for the grid we are constructing. The configuration of the grid is made by retrieving the correct table element and then using a method defined in the FlexiGrid engine to create the grid. To do that, we need to define the url from where to acquire the data to be shown, the data format to be used (in this case JSON) and an array of columns to include in the grid. For each column, we need to set a display name, a column name and a few optional parameters (e.g., sortable, data alignment and width). Then, we also define an array of buttons to add to the grid. These buttons are defined by some parameters (name, button class and the onpress identifier which is used to call a specific function on button press). Finally, we can also define some parameters for the grid (e.g., sorting parameters, rows by page). For each button of the grid, we create a function with the procedures to be taken when the button is pressed.

The implementation of the views started by the creation of a template to be followed by each view. These templates were then used to create the web layout for the views. For the Log in views, we have created the layout visible in Fig. 5.1, for the Laboratory management views and the students views we have followed the template 5.2 and for the administration views we have created the template presented in Fig. 5.3. These templates are intended to be auto-explicative to user, allowing the use of the platform without the need for following tutorials. We have also created a version of the templates for the mobile platforms, which can be seen in the appendix C.

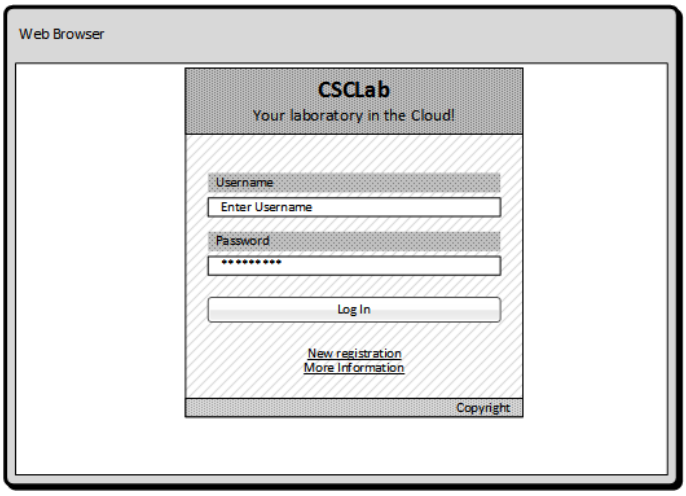


Figure 5.1: Website wireframe for login page.

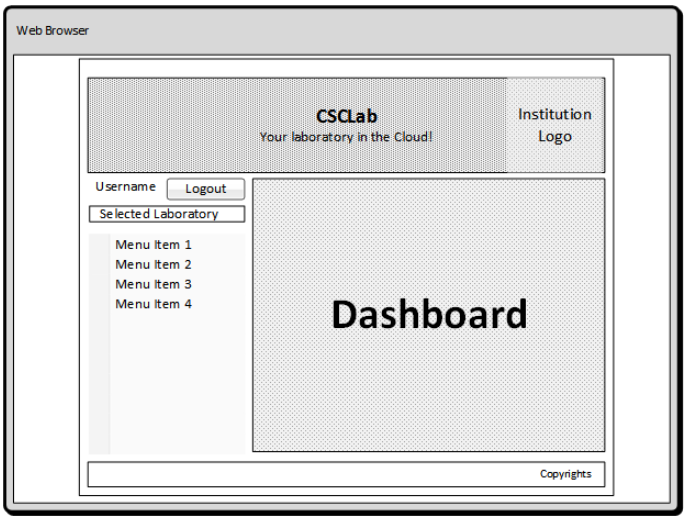


Figure 5.2: Website wireframe for Student and Lab Manger users.

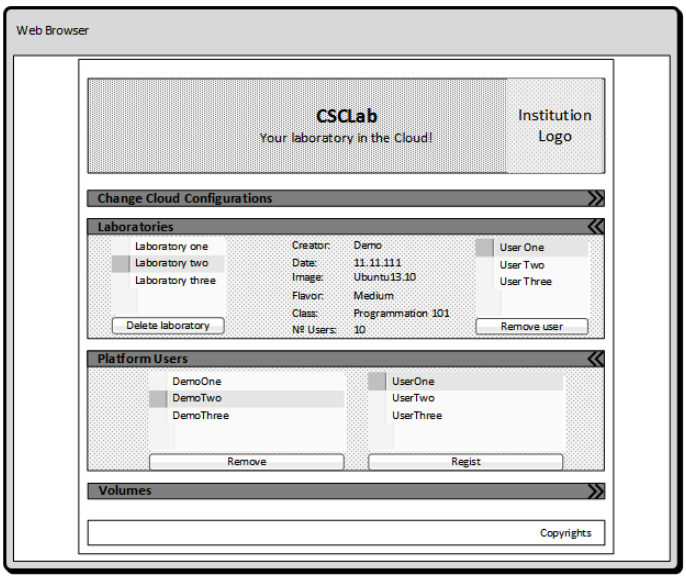


Figure 5.3: Website wireframe for Administrator users.

5.2.1 Login View

The login view is presented to a user when there is no session started in the web browser for the CSCLab platform and follows the structure of the wireframe Fig. 5.1.

In this view, the users are able to insert the credentials in a form which, when submitted, will check the validity of it. This form is implemented with tools from CodeIgniter, the Form Helper, which allows us some automation in validation, security and treatment of possible errors that may arise during validation.

This view can also be used by a user to get access to a registration view and a view with information about the platform. This is made with typical hyperlinks using HTML.

The final product is the view visible in Fig. 5.4.

Figure 5.4: Login view

The implementation of the registration view follows the same lines of the Log in view. The view is composed by a form with some fields that are required to be filled. When submitted, the data is saved in the platform database to be presented to a user with higher privileges, which can the accept or deny the registration. The final layout of the registration view can be seen in Fig. 5.5.

5.2.2 Laboratory Manager Views

The Lab Manager views can be accessed by the Lab Manager users. When the credentials submitted in the login view are validated, the users gain automatically access to these view.

In this case, were created a set of three views, i.e., the Lab Access view, the Lab Create view and the Notifications view. All these views use the same layout implemented by following the wireframe presented in Fig. 5.2. The difference in these views is what is shown in the Dashboard area presented in this wireframe. The Statistics features that should be accessible to the Lab Manager were postponed since they are not essential to operate the platform.

As it is possible to see in the wireframe, the view has a side-bar which is composed of a set of items. This side-bar has a logout form, used to destroy the user session and that redirects the user to the login page once it is done. In this side-bar, the user has also a form that can be used by the Lab Manager to select the desired laboratory. This "select form" presents the users with a list of the laboratories they can access and once a new laboratory is selected, the user can start managing the laboratory. The form submission is made with a JavaScript submit

Figure 5.5: Registration view

function that executes when a laboratory is selected. The side-bar has also a menu, which can be used to navigate between lab management views and each menu option is an hyperlink used to redirect the user to the desired view.

The Lab Access view, Fig. 5.6, presents the Lab Manager with information about the configuration of the laboratory and a dynamic grid (FlexiGrid). The dynamic grid presents a list of the virtual machines existent in the laboratory, as well as some information about each of the VMs (i.e., who is using it, its state and a remote console link). In this view, the Lab Manager can select a set of machines from the dynamic grid and terminate them or create new virtual machines, which will then be created in the cloud infrastructure and associated with the laboratory. The FlexiGrid has a set of buttons for these operations (e.g., create new VM or terminate VM) that once pressed triggers an asynchronous POST request. The request sends information about the submitted action (selected VM IDs or number of VMs to create depending on the action). Once complete, the FlexiGrid gets updated and the Alertify library is used to show a completion message to the user.

The Create Lab view, Fig. 5.7, can be used by the user to create new laboratories. The dashboard area of the view is composed of two essential areas, i.e., the submission form area and the information area. The first area is composed of a form created using the CodeIgniter helper form and is used to collect the configuration of the laboratory to be created. The second area displays the configuration data for the new laboratory and is updated each time the user interacts with the form. The form requires a valid name for the laboratory, which is verified using Ajax when is filled in, then it requires the user to specify the area it is designated to (e.g., Computer sciences or Mathematics). When the area is selected, we retrieve a list of adequate images from the platform catalog of images using Ajax. The user must then select an image

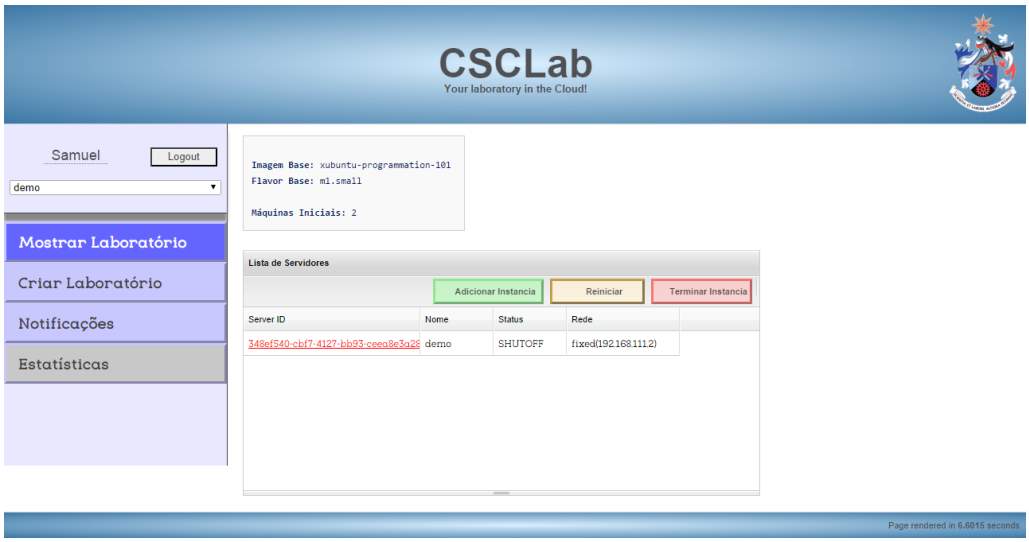


Figure 5.6: Lab Manager interface: Access laboratory view

from this catalog or add a new one (which passes by the personalization of a specific image). Finally, the user may check a set of checkboxs to select some laboratory features (persistent laboratory, storage management). When the user submits the form, the information submitted will be validated and the laboratory is created.

Figure 5.7: Lab Manager interface: Create laboratory interface

The Notification view, Fig. 5.8, displays information about new subscriptions to the laboratories. Here, the Lab Managers can accept the users that submit new subscriptions to specific classes they are responsible for. This view presents, in the dashboard, a dynamic grid with a list of the subscriptions. The Lab Manager can select a set of rows from the grid and accept or decline the subscriptions. The implementation of the grid for this view follows the same lines followed for the grid implemented in Lab Access view, i.e, it has a set of buttons to access or decline the subscriptions and the actions are then performed using Ajax.

In resume, the Lab Manager views implemented provide the features necessary for this user. We have implemented the Lab Access view to provide this user with access to the resources of the laboratories. Then, we have implemented the Create view to allow the Lab Manager to cre-

Figure 5.8: Lab Manager interface: Notifications interface

ate new Laboratories. We have also implemented the Notifications view which can be used by the platform to inform these users about class subscriptions in a wait list. The implementation of the Statistics view was postponed because it is not a essential feature of the platform.

5.2.3 Student Views

The Student views are part of the user interface that is provided for the students. These views are provided to the students once their credentials are checked. These views are automatically shown to users that are marked in the platform as being students.

The students views are composed of two sets of views: the Lab Access view and the Volume Management view. To implement these views, we first started by implementing the layout following the template of Fig. 5.2 which is also applied to the Lab management views. The main difference between these two views is the dashboard.

The Lab Access view, Fig. 5.9, can be used by the students to retrieve information about the resources assigned to each student. The view is divided into two specific areas: the laboratory settings and the class subscription settings. In the laboratory settings the users have access to information about the resources from a selected laboratory that are assigned to them. This information is the IP address for the machine and the login credentials (username and password). At the time the credentials are specific to each image but the interface is prepared to present a set of credentials specific to each user. The class subscription settings is composed of a form that presents a list of unsubscribed classes to the students in a multiple select form. The student may subscribe to one or more classes and when submitted, the form information is saved into the platform. Then, the list of subscribed classes is used to notify the Lab Managers of the subscriptions in the Lab management Notifications view.

The Volume management view, Fig. 5.10, can be used by the students to get access to the storage volumes from the platform. If a student has no volume assigned yet, then the view will present the user with an option to create a volume. If the student already has a volume assigned, then this view presents the user with a form to mount the volume into a specific machine. This form is composed of a peer of selection boxes. One box is used to select the volume (although at the time the platform will only allow one volume for each student) and other box with the list of virtual machine instances to which the user can assign the volume.

Figure 5.9: Students interface: Lab access view

This list is composed of machines from laboratories to which the student has access (and for which has a VM assigned) and an option to create a new VM with a specific image for the purpose of managing the volume. When the students select the VM, the information to connect to the VM is shown in a section of the view. This information is the IP address and login credentials. The user has also buttons to directly reboot or terminate the VM, without the need to get access to a console for the machine (no need for Remote Desktop connection).

Figure 5.10: Students interface: Volume management view

5.2.4 Administration Views

The administration views are part of the user interface for the users with administrative rights. The access to these views is gained through the validation of the Log in credentials for an Administrator user.

The implementation of the views for the administration area started by the construction of the layout for each view following the template of Fig. 5.3.

For the administrative area we have implemented four views. These views are sub-views

that are put together to form the administrative view. The administrative view has a collapsible block incorporating each of the views. The views we have implemented are the Cloud settings view, Laboratories view, Platform users view and Volumes view.

The Cloud settings view allows the administrator to change the endpoint to be used for communication with the Identity service of the cloud infrastructure. So far, this view does not implements any features and is marked as a work in progress that must be implemented in the future.

The Laboratories view is divided into three sections: the laboratories list section, the laboratory information section and the laboratory users section. The first section is a form that implements a select box that the administrator can use to select the laboratory to manage and is auto-submitted on select. When the form is submitted, a POST request is sent to the server, which retrieves information about the selected laboratory and the users list for the selected laboratory. On complete, the section two and three of the view are updated with the corresponding information. On section two of the view it is presented information about the creation of the laboratory (creation date, lab manager, base image and flavor for the VMs, class it is associated with and the number of VMs). On section three it is presented a form with a list of laboratory users in a multi-select box. The administrator may select one or more users from the list and remove them from the laboratory.

The Platform users view implements features that enable the user to remove users from the platform and accept new registrations. This view is divided into two section blocks which are the Platform users block and the Wait-list block. Both blocks are created with a form containing a multi-select box and a set of buttons. The first section presents a list of platform users and a button that allows the deletion of the selected users. The section two presents a list of users in a waiting list to be accepted to the platform. This view has a button to accept the registrations and a second one to reject registration. If the user registrations are accepted, then they are added to the Identification services of the platform (platform registration and creation of user in cloud infrastructure Identity service). If the users are rejected, then they are simply deleted from the waiting list.

The Volumes view implements features for the administrator to manage cloud resources, in this case the volumes from the storage service. This view is composed of two sections: the volumes list section and the volume information section. The first section was created using a simple form composed of a multi-select element that presents a list of existent volumes. When a volume is selected, then a POST request is made in order to retrieve information about the selected volume. In the second section, we present information about the selected volume, this information is the assigned user, volume state, volume name (typically a default name), volume size and to which laboratory the volume is associated with in the cloud infrastructure services.

In order to put together the administrative user interface, we include each of the views in a block that uses collapsible characteristics to show or hide the corresponding view. The collapsibility of the blocks is accomplished by using a combination of HTML and CSS elements. In the end, we have the administrative view Fig. 5.11.

5.2.5 Models and controllers (MVC)

The models and controllers of the platform were implemented using the structure presented in the design section. For this purpose, we have created a set of PHP classes that implements the attributes and methods described in the class diagrams. The Controllers inherit the CI_Contoller

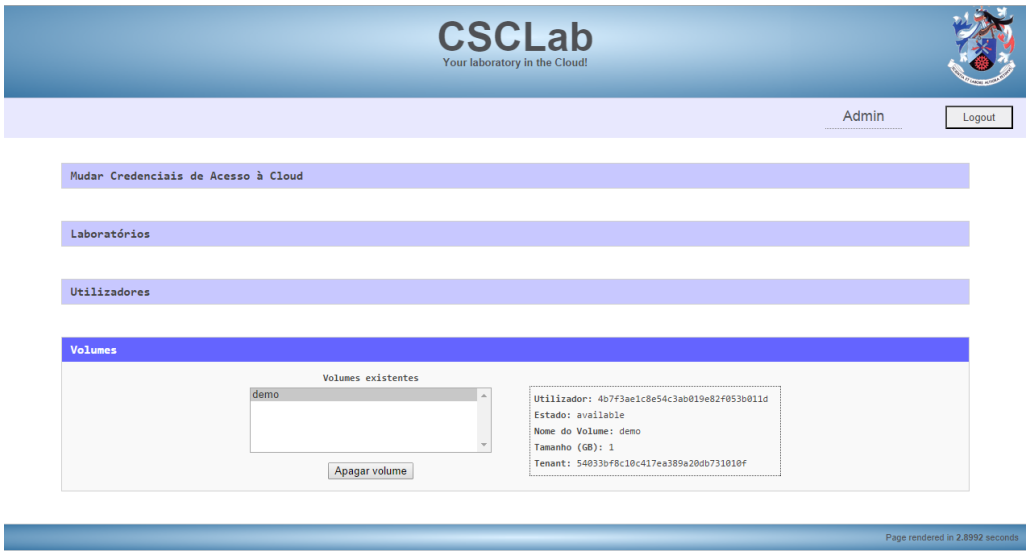


Figure 5.11: Administrator interface: Administrative view

class provided by the Framework and the Models inherit the CI_Model class of CodeIgniter.

5.2.6 Controllers

We have implemented two controllers, the Main controller and the Ajax Handler controller. The Main model is used to route the URLs to the corresponding views, to map user actions to the corresponding models and to update the views. The Ajax Handler model is used to route Ajax requests to the corresponding models and to return the responses to the requests, having for base the information returned by the platform models.

The Main model implements the Main class, which is an extension of the CI_Controller and then implements the attributes and methods of the class. The class constructor starts by verifying if the access is being made over a TLS/SSL connection, using HTTPS, and redirecting the request if necessary. This way we guarantee that all connections are made over a secure connection. Then, we set some security configurations for cookies and sessions, i.e., the cookies are set over HTTP connections only, we set cookies only over secure connections and the sessions use only inicialized IDs. This is made using the *ini_set* routines of PHP. After the cookies and sessions configuration, we start a new session. Then, the constructor loads a cache driver to be used in the system models. Finally, we load the models of the platform to the controller.

The methods of the Main controller used to show the views to users are implemented by following a defined set of steps:

1. **Verify session:** if the users are not allowed to access the view, then they get redirected to the main view.

2. **Acquire data:** the information that is needed for the views is gathered using the models, more precisely the methods that acquire and prepare the information to be shown to the users.

3. **Show view:** after gathering the necessary information for the view, we call the method to compose the final view (show_view method implemented in the controller), to which we pass the sub-views tab composes the final view and the parameters to pass to the view (an array with a set of key-peer values).

The Main controller has also a set of methods that are used on form submissions. These methods were also implemented following a specific set of steps:

1. **Required fields:** verify is the required fields of the form are correctly filled in.
2. **Validate data:** the information submitted is validated using the models if required.
3. **Take action:** use the models to make a specific action. Depending in the form it may, for example, create a new laboratory or add a new class.
4. **Update view:** update the user view using prepared data returned by the models or redirect the user to a new view.

The Ajax Handler controller implements a class that inherits the CI_Controller class. The class controller verifies if the requests are made over HTTPS, then loads the cache driver that is used in the models to save and load information from cache and finally, loads the platform models.

We implemented a set of methods to format the Ajax response, the ajax_success and the ajax_error methods. These methods set the headers of the response: the header response code with an OK or an ERROR code and a header with the content-type of the data that is used, JSON in this case. Then, we encode the message, by using a PHP json encoder.

The methods implemented in this controller also follow a set of pre-defined rules:

1. **Verify POST data:** we need to verify if the request has the required data to make the operation.
2. **Execute action:** the models are used to execute the action, and the returned data is saved if available.
3. **Response:** the success or error method is used to response to the request. The response depends on the execution of the action in the model.

5.2.7 Models

We have implemented two sets of models, the Cloud models and the Platform DB models. The first set is used to control the application data, logic and rules of the cloud services and the second is used for the platform database.

The cloud models are a set of four models: the Cloud_client model, the Server model, the Laboratory model and the Network model. Each of the models was implemented using a class. For each model we start by declaring the namespaces we use through the class. The cloud_client inherits the CI_Model class of the CodeIgniter framework and all other cloud models inherit the "cloud_client" class.

In the cloud_client class, we start by defining a set of constants that identify cloud resources. These constants define the name of the cloud services and the public and administrative endpoints of the platform. Then, in the class constructor, we instantiate the OpenStack client and set the cloud services clients to null values. The method to verify the user sets the secret (username and password) to the OpenStack client, then we call the authentication method of the OpenStack object and if no exception is thrown (authentication succeds) the OpenStack credentials are saved to cache. If an exception is thrown, it gets logged and then we return an error code to the controller. The import credentials method gets user credentials form cache and then tries to authenticate the credentials, if these credentials are changed then we update

the cached credentials. This method is used to import saved credentials to a new OpenStack client when necessary. The methods to set the service clients use OpenStack client to call the method to retrieve the required service and assign it to the corresponding object in the class.

The Server class, the Laboratory class and the Network class all inherit the Cloud_client class and implement a set of methods to work with different services of the cloud infrastructure. All the methods were implemented following a set of rules:

1. **Import credentials:** instantiate the OpenStack client with the cached credentials, if necessary for the specific action.
2. **Instantiate service:** instantiate the service client (e.g., network or compute).
3. **Prepare parameters:** prepare the parameters to pass to the client method (e.g., prepare an array of key-values).
4. **Take action:** use service client to take a specific action.
5. **Prepare response:** in case the action requires the return of information this is prepared following a set of rules.
6. **Return data:** return data to the controller, which may then update the views.

The Platform DB models set is more precisely a single model. This model is a class that inherits the CI_Model class from CodeIgnter. The class is composed by a set of methods to retrieve information from the platform database and methods to update, insert or delete information from the database. The actions to be made to the database (e.g., query, update, insert or delete) are made using the Database Class from CodeIgniter. We implement the methods of the models as follow:

1. **Prepare statement:** we prepare the SQL statement to retrieve the required information.
2. **Prepare parameters:** the query parameters are prepared (an array of parameters is created).
3. **Make query:** make the query to the database.
4. **Return result:** return the query result.

5.3 Summary

This chapter describes the development of the CSCLab prototype using OpenStack. To make a new implementation using other cloud infrastructures, we could maintain the implemented views and controller. We would work the implemented models, making the necessary adaptations to work with the required infrastructure. Furthermore, if we which to change some of the views or the controller, we can do it without the need for changing other components. This can be accomplished because of the MVC architecture.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have presented a solution to create virtual computing laboratories using the Cloud. Virtual computing laboratories have already been used in learning institutions for quite some time but these solutions are typically proprietary and have a learning curve that may imply tutorials and time for the users to learn how to use them correctly. Our solution involves the development of an agnostic architecture that can be used to work with different cloud infrastructure technologies. The proposed solution was also developed with the aim of being easily used by non-experts.

For development purposes, we have implemented a private cloud using OpenStack. This private cloud is composed by a set of services that allow us to have our private IaaS. Due to hardware limitations, there were services that should run in dedicated nodes stacked in common nodes which can affect the performance of the cloud services. This was the case of the Network service and the Block storage service which were configured in the controller node and in the compute node respectively. OpenStack provides a REST API which can be used to manage and configure OpenStack services. This API can be accessed through a set of tools and in our case, we opted for using an OpenStack SDK for PHP, the PHP OpenCloud SDK. Since this SDK is still a work in progress, there were some functionalities missing. To solve this problem, we have implemented the features that were not present but that we would need. So, we have implemented a complete package to work with the network API, by implementing the network service and each of the network resources provided by the service. We have also implemented additional features for the Volume service provided by the SDK. In this case, we have added the Transfer resources of the OpenStack volume service which are useful to transfer volumes between tenants.

To create CSCLab, we started by gathering a set of requirements to describe how a typical VCL must work and the features that the platform needs to implement. Then, we have presented a set of use cases to make a high level description of the features of the platform and, we have also designed the structure and behaviour of CSCLab. For the development of the platform, we followed its design, which was implemented using a MVC architecture model that enabled us to work with the different parts of the platform independently. By following the architecture and the design of the platform, we have constructed a prototype using OpenStack. This prototype provides a responsive application interface that enables an easy access to the platform either through a mobile or a desktop device and can be easily adapted to work with different cloud infrastructure technologies.

Finally, the prototype was made available as an open source solution. This way, the solution can benefit from new features and improvements that can be made by others. The prototype is publicly available at `http://www.csclab.s-alves.pt`.

6.2 Future Work

There is still work to be done in CSCLab. This work has to do with the implementation of a multi-cloud environment using the platform, the production of a study of additional features that may be required for a virtual computing laboratory and a small scale test of the platform.

The implementation of a multi-cloud environment would allow the use of different cloud infrastructures simultaneously (e.g., by using a hybrid cloud) and possibly the use of cloud infrastructures based in different cloud computing software platforms. This, implies changes in the implementation of the models of the platform, with no need for changes in other platform components.

The addition of new features would require a survey to the many departments of the educational institutions including teaching professionals, IT staff and students.

It is also planned as a future step, the implementation of the platform in a small scale cloud infrastructure. This will allow us to test the platform in a production environment.

Bibliography

[Ama14] Amazon. Aws for education [online]. 2014. Available from: http://aws.amazon.com/education/ [cited 1 Junho 2014]. 10

[ASZ+10] Michael Armbrust, Ion Stoica, Matei Zaharia, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin. A view of cloud computing. *Communications of the ACM*, 53, 2010. Available from: http://dx.doi.org/10.1145/1721654.1721672. 5, 6

[AZW09] K. Al-Zoubi and G. Wainer. Using rest web-services architecture for distributed simulation. In *Principles of Advanced and Distributed Simulation, 2009. PADS '09. ACM/IEEE/SCS 23rd Workshop on*, pages 114 – 121. IEEE, 2009. 17

[BGR12] L. Badger, T. Grance, and Patt-Corner, R. *NIST Special Publication 800-146 Computing Synopsis and Recommendations*. NIST, 2012. 5, 6, 7, 8, 9

[BLS13] S. D. Burd, X. Luo, and A. Seazzu. Cloud-based virtual computing laboratories. In *46th Hawaii International Conference on System Science*, pages 5079–5088, 2013. 1, 10, 11

[Cit14a] Citrix. Xenapp [online]. 2014. Available from: http://www.citrix.com/products/xenapp/overview.html [cited 1 Agosto 2014]. 10

[Cit14b] Citrix. Xendesktop [online]. 2014. Available from: http://www.citrix.com/products/xendesktop/overview.html [cited 1 Agosto 2014]. 10

[CRB+11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41, 2011. Available from: http://dx.doi.org/10.1002/spe.995. 10

[Ell14] EllisLab. A fully baked php framework [online]. 2014. Available from: https://ellislab.com/codeigniter [cited 25 Agosto 2014]. 43

[Euc13] Eucalyptus. Eucalyptus [online]. 2013. Available from: www.eucalyptus.com/. 62, 63

[Gri14] Future Grid. Future grid [online]. 2014. Available from: https://portal.futuregrid.org/ [cited 1 Junho 2014]. 10

[Hv14] Microsoft Hyper-v. Microsoft hyper-v [online]. 2014. Available from: http://technet.microsoft.com/en-US/evalcenter/dn528863.aspx [cited 1 Junho 2014]. 10

[ICV12] Shigeru Imai, Thomas Chestna, and Carlos A. Varela. Elastic scalable cloud computing using application-level migration. In *UCC '12 Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 91–98. IEEE, 2012. 7

[IFY+13] Eisuke Ito, Brendan Flanagan, Chengjiu Yin, Tetsuya Nakatoh, and Sachio Hirokawa. A private cloud environment for teaching search engine construction. In *Proceedings of the 21st International Conference on Computers in Education. Indonesia: Asia-Pacific Society for Computers in Education*, 2013. 11

[Inf14] European Grid Infrastructure. European grid infrastructure [online]. 2014. Available from: `www.egi.eu/infrastructure/index.html` [cited 1 Junho 2014]. 10

[KZ12] Nadir K.Salih and Tianyi Zang. Survey and comparison for open and closed sources in cloud computing. *International Journal of Computer Science Issues*, 9:118, 2012. 62

[LKS06] M. Laitkorpi, J. Koskinen, and T. Systa. A uml-based approach for abstracting application interfaces to rest-like services. In *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*. IEEE, 2006. 16

[LSS12] Olga Liskin, Leif Singer, and Kurt Schneider. Welcome to the real world: A notation for modeling rest services. *Internet Computing, IEEE*, 16(4):36 – 44, 2012. 16

[Mal12] Arpit Malani. Taxonomy, classification & implementation of open source cloud computing platforms. Technical report, Indian Institute of technology Bombay, May 2012. 62

[MG12] P. Mell and T. Grance. *NIST Special publication 800-145 NIST definition of cloud computing*. NIST, 2012. 1, 5, 7, 9

[MLB+11] Sean Marston, Z Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing: The business perspective. *Decision Support Systems*, 51(1):176–189, 2011. 5

[MMHJ11] M. Mahjoub, A. Mdhaffar, R.B. Halima, and M. Jmaiel. A comparative study of the current cloud computing technologies and offers. In *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pages 131 – 134. IEEE, 2011. 62

[Nim13] Nimbus. Nimbus [online]. 2013. Available from: `www.nimbusproject.org/`. 62, 63

[NNH12] Tien-Dung Nguyen, Mui Van Nguyen, and Eui-Nam Huh. Service image placement for thin client in mobile cloud computing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 416 – 422. IEEE, 2012. 5

[Ope13] OpenNebula. Opennebula [online]. 2013. Available from: `www.http://opennebula.org/`. 62, 63

[Ope14a] OpenStack. Companies supporting the openstack foundation [online]. 2014. Available from: `http://www.openstack.org/foundation/companies/` [cited 1 Agosto 2014]. 13

[Ope14b] OpenStack. Openstack [online]. 2014. Available from: `https://wiki.openstack.org/wiki/Main_Page` [cited 1 Agosto 2014]. 13

[Ope14c] OpenStack. Openstack api complete reference [online]. 2014. Available from: `http://developer.openstack.org/api-ref.html` [cited 25 Agosto 2014]. 17

[Ope14d] OpenStack. Openstack api quick start [online]. 2014. Available from: http://docs.openstack.org/api/quick-start/content/ [cited 25 Agosto 2014]. 17

[Ope14e] OpenStack. Openstack installation guide for ubuntu 12.04 (lts) [online]. 2014. Available from: http://docs.openstack.org/havana/install-guide/install/apt/content/index.html [cited 25 Agosto 2014]. 15, 16

[Ope14f] OpenStack. Resources for application development on private and public openstack clouds [online]. 2014. Available from: http://developer.openstack.org/ [cited 25 Agosto 2014]. 18

[Par14] Parallels. Parallels [online]. 2014. Available from: http://www.parallels.com/ [cited 1 Junho 2014]. 10

[PLH09] Dunlu Peng, Chen Li, and Huan Huo. An extended usernametoken-based approach for rest-style web service security authentication. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 582 – 586. IEEE, 2009. 16

[PP13] R. Patel and S. Patel. Survey on resource allocation strategies in cloud computing. *International Journal of Engineering Research & Technology*, 2, 2013. 5

[PS12] Anita S. Pillai and L.S. Swasthimathi. A study on open source cloud computing platforms. In *EXCEL International Journal of Multidisciplinary Management Studies*, pages 31 – 40. Zenith Research, 2012. 62

[Rac14] Rackspace. Php-opencloud [online]. 2014. Available from: https://github.com/rackspace/php-opencloud [cited 25 Agosto 2014]. 18

[RR07] Lionard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, 2007. 16

[Sof13] OpenStack Cloud Software. Openstack [online]. 2013. Available from: www.openstack.org. 2, 62, 63

[SPPH12] I. Santana-Perez and M.S. Perez-Hernandez. A semantic scheduler architecture for federated hybrid clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 384 – 391. IEEE, 2012. 5, 10

[Sup12] Rackspace Support. Understanding the Cloud Computing Stack. 2012. 9

[Ueh13] Minoru Uehara. Proposal for byod based virtual pc classroom. In *16th International Conference on Network-Based Information Systems*, 2013. 1, 2

[VCL14] Apache VCL. Apache vcl [online]. 2014. Available from: https://vcl.apache.org/ [cited 1 Junho 2014]. 10, 11

[vDW12] G. von, J. Diaz, and Wang. Comparison of multiple cloud frameworks. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 734 – 741. IEEE, 2012. 9

[vLDWF12] G. von Laszewski, J. Diaz, Fugang Wang, and G.C. Fox. Comparison of multiple cloud frameworks. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 734 – 741. IEEE, 2012. 62

[VMw14a] VMware. vcloud director [online]. 2014. Available from: `http://www.vmware.com/products/vcloud-director` [cited 1 Agosto 2014]. 10

[VMW14b] VMWare. Vmware [online]. 2014. Available from: `http://www.vmware.com/` [cited 1 Junho 2014]. 10

[VMw14c] VMware. Vmware vcenter lab manager [online]. 2014. Available from: `https://www.vmware.com/support/pubs/labmanager_pubs.html` [cited 1 Agosto 2014]. 10

[VMw14d] VMware. vsphere esx and esxi info center [online]. 2014. Available from: `http://www.vmware.com/products/esxi-and-esx/overview` [cited 1 Agosto 2014]. 10

[WGL+12] Xiaolong Wen, Genqiang Gu, Qingchun Li, Yun Gao, and Xuejie Zhang. Comparison of open-source cloud management platforms: Openstack and opennebula. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2457 – 2461. IEEE, 2012. 10, 62

[WHY10] X. Wuang, G. Hembroff, and R Yedica. Using vmware vcenter lab manager in under-graduate education for system administration and network security. In *SIGITE10, ACM conference on Information technology education*, pages 43–52, 2010. 1, 10, 11

[XHT12] L. Xu, D. Huang, and W.-T. Tsai. V-lab: A cloud-based virtual laboratory platform for hands-on networking courses. In *ITiCSE12, 17th ACM annual conference on Innovation and technology in computer science education*, pages 256–261, 2012. 1, 2, 10

[YBvL11] R. Younge, A.J.and Henschel, J.T. Brown, and G. von Laszewski. Analysis of virtualization technologies for high performance computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 9 – 16. IEEE, 2011. 6

Appendix A

Cloud Computing software platforms comparison

In table A.1, we make a comparison of features for open-source cloud computing platforms. For this comparison, we use Eucalyptus, OpenNebula, Nimbus and OpenStack.

Table A.1: Comparison of Cloud Computing software platforms [WGL+12][MMHJ11][PS12][vLDWF12][KZ12][Mal12].

	Eucalyptus [Euc13]	**OpenNebula** [Ope13]	**Nimbus** [Nim13]	**OpenStack** [Sof13]
Producer	Santa Barbara University and Eucalyptus System Company	European Union	Chicago University	Rackspace, NASA, Dell, Citrix, Cisco, Canonical, other companies and organizations
Objectives / Concept	Mimic Amazon EC2	Private Cloud, High personalization level	Scientific solution for Cloud Computing	Virtualization portability
Consumers	Companies	Cloud computing and virtualization researchers	Scientific companies	Companies, service providers and researchers
Supported OS	Linux and Windows.	Linux and Windows.	Most of Linux distributions	Linux and Windows. Requires x86 server.
Architecture	Hierarchical and modular design with a minimum of 2 servers.	Centralized, three components and a minimum of 2 servers.	Centralized, three components and a minimum of 2 servers.	Modular
Hypervisor	KVM, Xen and VMWare	KVM, Xen and VMWare	KVM an Xen	KVM, Xen, VMWare vSphere, LXC, UML and MS HyperV
Storage	Walrus. NFS and LVM.	NFS, SCP, Shared FS	GridFTP, Comulus and SCP	OpenStack Storage (Swift). NAS, NFS, LVM and SAN.
Network	DHCP server on cluster controller	Manual configuration, VLAN	DHCP server installed on nodes	Flat, Flat DHCP, VLAN DHCP, IPv6
Load balancing	Cloud controller (CLC)	Nginx	Context broker	Cloud controller (CLC)
Fault tolerance	Separation of the cluster controller	Back-end Database	Periodic verification of the nodes	Replication
VM Location	Node controller	Cluster node	Physic node	OpenStack Compute
Migration of VMs in use	No	Yes	No	Yes

Continued on next page

Table A.1 – continued from previous page

	Eucalyptus [Euc13]	**OpenNebula** [Ope13]	**Nimbus** [Nim13]	**OpenStack** [Sof13]
Authentication	Credentials X509.	Credentials X509, SSH RSA Key-Pair, Password, LDAP.	Credentials X509, Grids.	Credentials X509, LDAP.
Interface	Amazon EC2 e S3 API, OCCI API and proprietary API (REST API).	Amazon EC2 e S3 API, OCCI API, Libvirt API and proprietary API.	Amazon EC2 e S3 API, Libvirt API and proprietary API.	Amazon EC2 e S3 API, OCCI API and proprietary API (REST API).
Language	Java, C, python	Java, Ruby, C++	Python, Java	Python
Personalization	Some for administration, except for the user.	Basically everything.	Many parts, with exception for the storage of images and Globus credentials.	Basically everything.
Compatibility	Amazon EC2	Amazon EC2	Amazon EC2 and S3, support to many platforms	Amazon EC2 and S3, Multi-platform
License	Open Source - Apache	Open Source - Apache	Open Source - Apache	Open Source - Apache

Appendix B

OpenStack Network installation and configuration

In this appendix are described the install and configuration steps of the OpenStack Network service – Neutron – in an architecture that implements Neutron services in the controller node. The appendix starts by the configuration of the controller node. Then, the configuration for the compute node is described. Finally, we describe the creation of an external and an internal network.

B.1 Controller node

Before we start the configuration of the neutron service, we need to configure the environment:

1. Create neutron database.
2. Create neutron user, service and endpoint.

Next, we can install and configure neutron services in the controller node.

1. **Install services**

Listing B.1: Install packages.

```
$ prompt > apt-get install \
neutron-server neutron-plugin-openvswitch-agent \
openvswitch-switch neutron-dhcp-agent neutron-l3-agent \
openvswitch-datapath-dkms
```

2. **Change package filtering options**

Listing B.2: Edit file "/etc/sysctl.conf".

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

Listing B.3: Get new configurations.

```
$ prompt > sysctl -p
$ prompt > service networking restart
```

3. **Set Neutron authentication configuration**

Listing B.4: Edit file "/etc/neutron/neutron.conf:".

```
# Authentication service
auth_strategy = keystone
```

```
# Keystone credentials
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS

# Rabbit options
rabbit_host = controller
rabbit_userid = guest
rabbit_password = RABBIT_PASS

# Database section
[database]
connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron
```

Listing B.5: Edit file "/etc/neutron/api-paste.ini".

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:
    filter_factory
auth_host = controller
auth_uri = http://controller:5000
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

4. **Configure Open vSwitch (OVS) plugin -** Used to perform software-defined networking.

   (a) Configure bridges

Listing B.6: Configure bridges.

```
$ prompt > # Restart service
$ prompt > service openvswitch-switch restart
$ prompt > # Add network bridges
$ prompt > ovs-vsctl add-br br-int
$ prompt > ovs-vsctl add-br br-ex
$ prompt > # Add a port from the external interface  to br-ex
$ prompt > ovs-vsctl add-port br-ex EXTERNAL_INTERFACE
```

   (b) Configure external interface

Listing B.7: Edit file "/etc/rc.local".

```
# Set NIC into promiscuous mode
ip link set p132p1 promisc on
ifconfig p132p1 up
```

(c) Set br-ex with the network address of external network

(d) Configure L3 and DHCP agent

Listing B.8: Edit file "/etc/neutron/l3_agent.ini" and "/etc/neutron/l3_agent.ini".

```
interface_driver = neutron.agent.linux.interface.
    OVSInterfaceDriver
use_namespaces = True
```

(e) Configure Neutron to use OVS

Listing B.9: Edit file "/etc/neutron/neutron.conf".

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
    OVSNeutronPluginV2
```

(f) Use OVS for GRE tunneling

Listing B.10: Edit file "/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini".

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP/MANAGEMENT_INTERFACE_IP
```

(g) Configure firewall plugin

Listing B.11: Edit file "/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini".

```
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

(h) Restart OVS agent

Listing B.12: Restart OVS agent service.

```
$ prompt > service neutron-plugin-openvswitch-agent restart
```

5. **Configure DHCP**

(a) Configure DNS IP – add DNS IP to file "/etc/resolv.conf".

(b) Configure DHCP agent.

Listing B.13: Edit file "/etc/neutron/dhcp_agent.ini".

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
dnsmasq_dns_server= "external␣DNS␣IP"
```

6. **Configure metadata agent.**

Listing B.14: Edit file "/etc/nova/nova.conf".

```
neutron_metadata_proxy_shared_secret = METADATA_PASS
service_neutron_metadata_proxy = true
```

Listing B.15: Edit file "/etc/neutron/metadata_agent.ini".

```
[DEFAULT]
auth_url = http://controller:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_PASS
```

7. **Additional configurations**

Listing B.16: Edit file "/etc/neutron/neutron.conf".

```
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_port = 5672
```

Listing B.17: Edit file "/etc/nova/nova.conf".

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.
    LinuxOVSInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
#security_group_api=neutron
```

8. **Restart services**

Listing B.18: Restart services

```
$ prompt > service nova-api restart
$ prompt > service neutron-server restart
$ prompt > service neutron-dhcp-agent restart
$ prompt > service neutron-l3-agent restart
$ prompt > service neutron-metadata-agent restart
```

B.2 Compute node

Next, we present the configuration of the network service for the computation node. To add new nodes to the cloud infrastructure, we just need to follow the the steps presented bellow.

1. **Disable package fieltering.**

Listing B.19: Edit file "/etc/sysctl.conf".

```
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

Listing B.20: Apply new configuration.

```
$ prompt > sysctl -p
```

2. **Install and configure networking plugins.**

   (a) Install OVS plugin

Listing B.21: Install networking plugins packages.

```
$ prompt > apt-get install neutron-plugin-openvswitch-agent \
openvswitch-switch openvswitch-datapath-dkms
```

Listing B.22: Restart OVS plugin.

```
$ prompt > service openvswitch-switch restart
```

Listing B.23: Add bridge for internal network.

```
$ prompt > ovs-vsctl add-br br-int
```

   (b) Identify plugin to use.

Listing B.24: Edit file "/etc/neutron/neutron.conf".

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
   OVSNeutronPluginV2
```

   (c) Plugin configuration

Listing B.25: Edit file "/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini".

```
# GRE tunneling configuration
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP

# firewall configuration
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

3. **Configure neutron core services**

Listing B.26: Edit file "/etc/neutron/neutron.conf".

```
# service credentials
auth_host = controller
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
auth_url = http://controller:35357/v2.0
auth_strategy = keystone
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_port = 5672
#rabbit_userid = guest
rabbit_password = RABBIT_PASS

# database connection
[database]
connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron
```

Listing B.27: Edit file "/etc/neutron/api-paste.ini".

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:
    filter_factory
auth_host = controller
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

4. **Configure computation service to use Neutron**

Listing B.28: Edit file "/etc/nova/nova.conf".

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.
    LinuxOVSInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron
```

5. **Restart services**

Listing B.29: Restart services

```
$ prompt > service nova-compute restart
$ prompt > service neutron-plugin-openvswitch-agent restart
```

B.3 Create a network

To create an external network, we must follow a set of steps:

1. Create a network.
2. Create a sub-network.
3. Create a router.
4. Set router gateway.

Listing B.30: Create an external network

```
$ prompt > SPECIAL_OPTIONS="--provider:network_type␣gre␣\
--provider:segmentation_id␣SEG_ID"
$ prompt >
$ prompt > SEG_ID="2"
$ prompt > DEMO_TENANT_ID="demo"
$ prompt >
$ prompt > neutron net-create ext-net --router:external=True [
    SPECIAL_OPTIONS]
$ prompt >
$ prompt > neutron subnet-create ext-net --allocation-pool \
start=FLOATING_IP_START,end=FLOATING_IP_END \
--gateway=EXTERNAL_INTERFACE_GATEWAY \
--enable_dhcp=False EXTERNAL_INTERFACE_CIDR --name subnet1
$ prompt >
$ prompt > neutron router-create ext-to-int --tenant-id DEMO_TENANT_ID
$ prompt > EXT_TO_INT_ID="router_id"
$ prompt > EXT_NET_ID="created_net_id"
$ prompt > neutron router-gateway-set EXT_TO_INT_ID EXT_NET_ID
```

To create an internal network, we have to follow a similar approach to the creation of the external network

Listing B.31: Create an internal network

```
$ prompt > neutron router-create router-demo
$ prompt > neutron net-create net-demo
$ prompt > neutron subnet-create net1 172.17.0.0/24 --name subnet-demo
$ prompt > neutron router-interface-add router-demo subnet-demo
```

Appendix C

Wireframes for mobile platforms

This appendix presents the templates that were used to create the layout of the platform view for the mobile platforms.

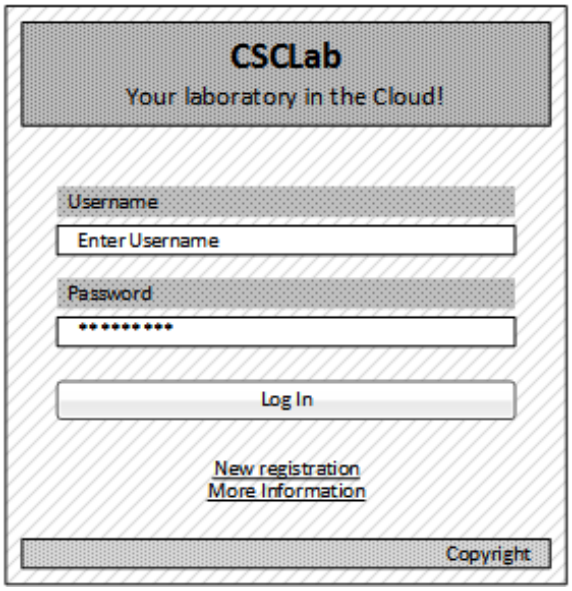


Figure C.1: Website wireframe for login page.

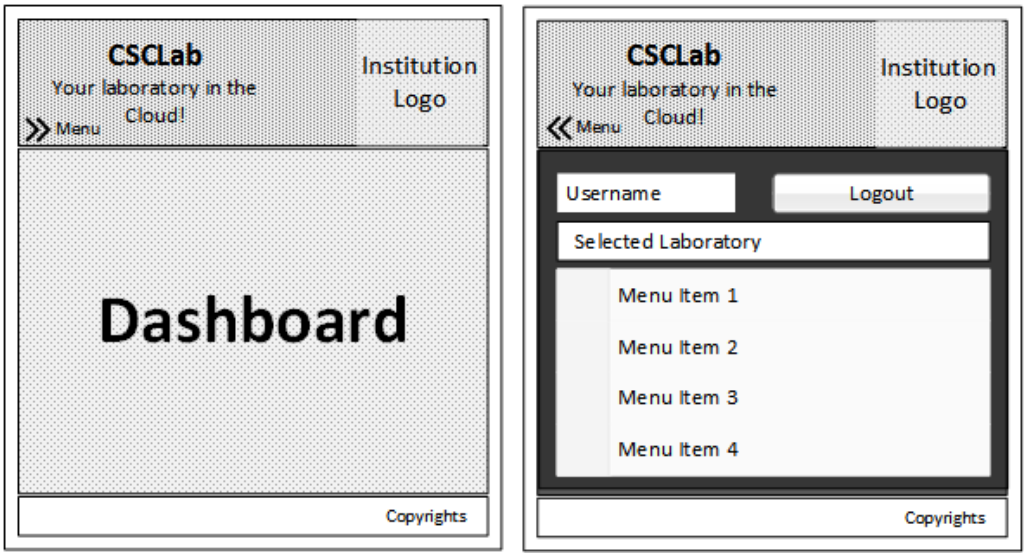


Figure C.2: Website wireframe for Student and Lab Manger users.

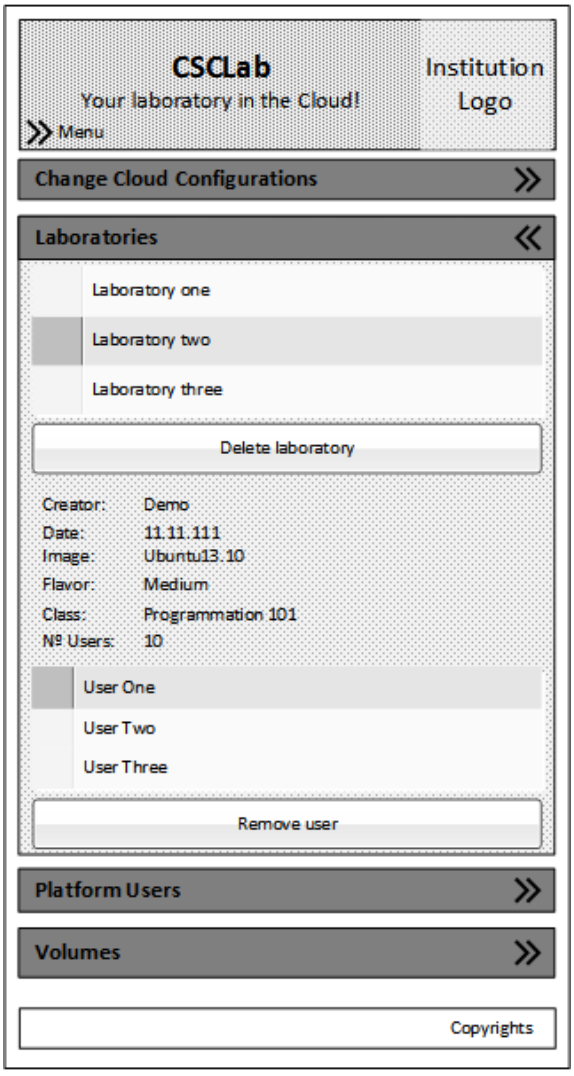


Figure C.3: Website wireframe for Administrator users.