Universidad Carlos III de Madrid

Bsc Computer Science and Engineering

BACHELOR'S THESIS

Development of an OpenStack Cloud deployment tool

Author Sergio Pérez Fernández

Tutor Alejandro Baldominos Gómez

September 2016

Table of Contents

Table of figures

Table of tables

Chapter 1: Introduction

1. Motivation

In my third year on the university we had a subject called “Distributed Systems”, which is the one of the courses which I have enjoyed the most on my degree.

At the end of the subject we had an assignment which was making a presentation to all the class about one of the topics which was at the list we were given, giving them an introduction in order to discover new things.

I started trying to find which topic was easy and interesting at the same, but while I was doing it, I found OpenStack. I knew that making the presentation about it was not one of the best options with this criteria, since I considered it to be more difficult that what I was looking for.

But at the same time, it was about a really interesting project so I decided to take the opportunity to learn more about OpenStack and also to show something as great at this project to my classmates.

The summer of that year I decided to take the “Professional Internships” subject, so I worked one month in a company where I deployed a cluster of Apache Spark among other things, something that increased a little bit more my interest about distributed systems.

After finishing on that company, I went back to the university, where I had to decide which was my bachelor's degree final project going to be. I checked the list in order to find the one I was more interesting at, and then I found a project related with all these things, called “Development of an OpenStack Cloud deployment tool”. After reading the description and speaking with Alejandro Baldominos, the person in charge of it, I decided to create this tool.

At the beginning I only knew some theoretical concepts about OpenStack, and doing a deployer for it as my bachelor's thesis was a good idea to learn a lot and discover new things about OpenStack that could be really useful in a future.

2. What is OpenStack

Illustration 1: OpenStack logo

OpenStack is a project made for controlling large pools of compute, storage and networking resources through a datacenter,all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

The OpenStack foundation was originally formed by RackSpace and NASA, but right now there are a lot of big companies which are AT&T, Canonical, HP, IBM, Intel, RackSpace, Red Hat, Suse, Aptira, CCAT, Cisco, Dell, DreamHost, EasyStack, EMC, Mirantis, Nec, NetApp, Symantec, UnitedStack and Virtuozzo. All of them collaborate in different ways with the OpenStack projects in order to make it better.

It also has 131 corporate sponsors who are providing funding to support the OpenStack foundation, being a lot of them big companies like Deutsche Telecom, Google, Oracle, Samsung, SAP, VMWare or Volkswagen.

There are also 460 companies who are also involved on making OpenStack successful in different ways, like contributing code, building an OpenStack product or helping build the community.

In order to check all the companies involved in the project there is a portal where all are listed.

https://www.openstack.org/foundation/companies/

In order to know who has contributed to OpenStack it is possible to check the OpenStack Community Dashboard.[1][2]

http://activity.openstack.org/

The goal of the OpenStack foundation is having a community where people with similar philosophies about cloud architecture and open-source software can collaborate together. It is a project made for anyone who wants to build a cloud system.

Developers can download the source code of OpenStack and submit patches following the “How To Contribute” official guide, since OpenStack believes in open source, open design, open development, anyone can participate. There are also a lot of tasks available in the project that do not require to be a developer, like documentation, bug reporting, asking and answering questions or translation. [3]

The project is formed by several projects that provide different functionalities, building all of them a platform which manages a Cloud infrastructure. Some of these projects are completely needed in order to make OpenStack work and, on the other hand, there are components which are completely optional, offering interesting services at a Cloud that can be needed by their users or not.

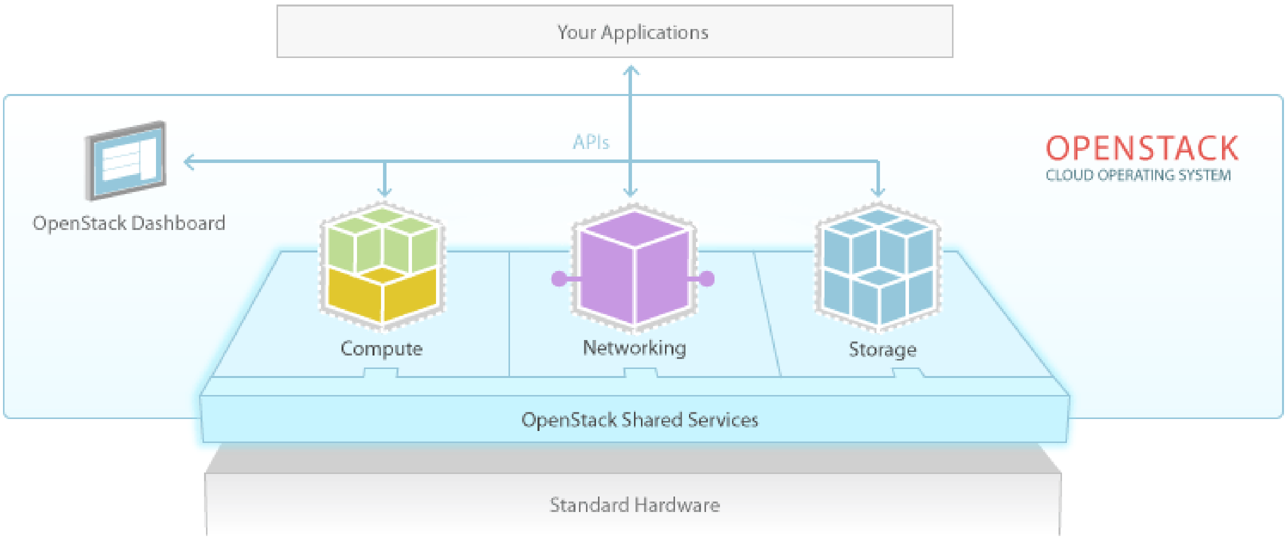


Illustration 2: OpenStack explained at an image

2.1. Keystone

Illustration 3: Unofficial Keystone logo

Keystone is an OpenStack service that provides clients authentication, service discovery and distributed multi-tenant authorization by implementing the OpenStack Identity API. [4]

2.2. Glance

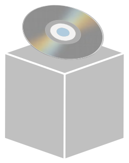


Illustration 4: Glance logo

Glance is a module which provides a service where users can upload and discover data assets that can be used with other services. This currently includes images as the used for virtual machines and metadata definitions.

Glance has a RESTful API and includes discovering, registering, and retrieving virtual machine images.

VM images made available through Glance can be stored in a variety of locations from simple file systems to object-storage systems like the OpenStack Swift project.

Glance is developed to have a component based architecture, to be highly available, fault tolerant, recoverable and to use open standards. [5]

2.3. Nova

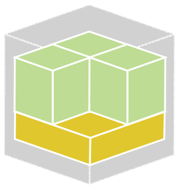


Illustration 5: Nova logo

Nova is an project designed to provide power massively scalable, on demand, self service access to compute resources. [6]

Nova has an abstraction layer for compute drivers, so it is compatible with a lot of different hypervisors that can be checked on the HypervisorSupportMatrix article at the OpenStack wiki:

https://wiki.openstack.org/wiki/HypervisorSupportMatrix

2.4. Neutron

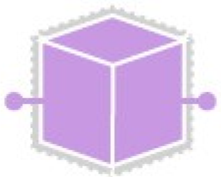


Illustration 6: Neutron logo

Neutron is an OpenStack project to provide “network connectivity as a service” between the interface devices managed by other OpenStack services like Nova. [7]

2.5. Other services

With the services listed until now it is possible to create an OpenStack cloud with the features of all the 4 of them, but, there are also several more services, like Cinder, which provides persistent block storage to running instances, Horizon, which provides a web based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses or configuring access controls.

In order to check which more services can be installed into an OpenStack cloud it is possible to check the project-navigator portal of the OpenStack foundation, where all them are listed and detailed.

The project navigator of OpenStack is hosted at the following URL:

http://www.openstack.org/software/project-navigator/

Chapter 2: State of the art

In the last years we have been hearing a lot about “Cloud Computing”, being it a name that will be heard even more and more.

1. What is Cloud Computing?

According to the National Institute of Standards and Technology, Cloud Computing is a model for enabling ubiquitous, Cloud computing “is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”. [8]

As the NIST states, there are five essential characteristics the Cloud Computing model has five essential characteristics, three service models and four deployment models.

1.1. Essential characteristics

There are several essential characteristics that a cloud platform needs to have in order to be as it is defined by the NIST.

- On-demand self-service: An user can take computing capabilities like server time or network storage without having to interact with an human.
- Broad network access: Capabilities are available through the network from devices of different types, like smartphones, thinclients or other devices.
- Resource polling: The provider computer resources are pooled to serve multiple consumers, reassigning the resources dynamically. The user has no control or knowledge about where the resources are coming from.
- Rapid elasticity: The provider capabilities can be upgraded, in some cases automatically, in order to make the system able to fulfill a possible demand. The user must see the system as if it had unlimited resources.
- Measured service: The cloud system must control, monitor and report the use of the resources of the system.

1.2. Service models

In cloud computing it is possible to differentiate three types of models, which depend on the quantity of control that the user has over the hardware of the software of a cloud service or platform.

- Software as a service (SaaS). The user can use applications which run in the provider infrastructure. These applications can be accessed by using a thin client like a web browser. An example of a SaaS based application is DuckDuckGo, a web search engine.

(https://www.duckduckgo.com)

- Platform as a service (PaaS). The user can deploy applications on the infrastructure, having control of its applications. An example of a PaaS platform is HerokuApp. (https://www.heroku.com)
- Infrastructure as a service (IaaS). The user is provided with processing, storage, networks and other fundamental resources where the user can run software. The main difference of a IaaS platform with a PaaS platform is that the user has the possibility of controlling more things, like operating systems, storage or networking components. An example of an IaaS service is Amazon EC2.

1.3. Deployment models

A cloud computing service can work in different ways which depend on the group of users a cloud platform is focused for.

- Private cloud: The cloud infrastructure is used by a single organization or company.
- Community cloud: The cloud infrastructure is used by a community of users with a goal.
- Public cloud: The cloud is used by the general public.
- Hybrid cloud: The cloud is composed by two or more different cloud infrastructures that are an unique entity.

2. Basics advantages in cloud computing

The basic in cloud computing is that the user does not have to make large investments buying hardware and having to run the hardware somewhere. Instead of having to do it, an user can access to the exact quantity of resources needed to power its idea or to manage its project, paying just for this quantity of used resources.

That feature gives the users a big advantage, since it will not be needed to think how much servers the business will need, that removes the possibility of loosing all money making bad predictions in the capability of the infrastructure of a project or company, since the customer will just pay for what its used while more resources can be acquired without having to invest in hardware.

At the same time, not having to worry about the infrastructure when an application or a system is being deployed or used is an advantage in terms of speed, since it is avoided to handle the hardware, also making it easier to do several things like deploying a virtual machine in seconds or taking snapshots of these virtual machines automatically.

[9]

3. Available cloud services

Nowadays there are several companies which offer cloud platforms. Some of the biggest ones are Amazon, with the Amazon Web Services platform, Google, with the Google Cloud Platform, and Microsoft with Microsoft Azure.

3.1. Amazon Web Services

Amazon Web Services is an Amazon platform which offers compute power, database storage, content delivery and more functionalities. There are several big companies using these services, like Netflix, Airbnb, Expedia, Slack, Adobe, Samsung, Spotify and much more.

Illustration 7: Amazon Web Services logo

There is a really big number of services offered at this platform. Some of them are the compute and the storage service, being the most know ones shown at this point.

[10]

Amazon EC2

Amazon EC2 is a service which offers flexible computational capacity using instances. There are several types of instances, so the user just pays for what its needed. [11]

Several operating systems for the instances are offered, being them GNU/Linux and BSD distributions or different versions of Windows Server. [12]

Amazon S3

Amazon S3 offers safe, highly scalable and secure cloud storage. The user is billed depending on the space used and there are different plans which depend on the frequency the data is accessed. [13]

As mentioned before, the number of available services at the AWS platform is way bigger than the two services shown it here, being it possible to check the services on the AWS webpage.

https://aws.amazon.com/es/

3.2. Google Cloud Platform

As Amazon, Google has a platform where cloud services are offered. It is called Google Cloud Platform and it also offers a wide variety of services, being it possible to find equivalent services to the ones offered by Amazon.

Google Compute Engine

Illustration 8: Google Cloud Platform logo

Google Compute Engine offers virtual machine instances optimized for the user needs. It offers high flexibility and the users are billed taking into account the compute time used. [14]

There are several operating systems that can run on these instances, being them several GNU/Linux distributions or different versions of Windows Server. [15]

Google App Engine

Google App Engine is a service which allows an user to deploy its application on it. The user just uploads the code and Google App Engine manages the application availability, making the user to forget about servers to provision or maintain. [16]

Google Cloud Storage

Google Cloud Storage is a service which offers highly available and durable object storage, offering the user a simple and consistent method of access. The user is billed taking into account the quantity of space that is used, having three pricing plans available which depend on different use cases based on the access frequency. [17]

There are much more services offered by Google, which can be seen on their Google Cloud Platform portal. https://cloud.google.com/why-google/

3.3. Microsoft Azure

Microsoft Azure is the Microsoft alternative cloud platform of the other platforms shown until now. With a number of 58 different services, the similar ones to the shown on the sections of the other services are listed here. [18]

Microsoft Azure

Illustration 9: Microsoft Azure logo

Virtual Machines

The "Virtual Machines" service offers high scalable virtual machines, optimized for the user needs with several types of instances. The quantity of different operating system images is 1070 by the date this document is written, being them different Windows versions and GNU/Linux distributions, a lot of them bundled with different packages of software. [19]

Storage

The Storage service offers several storage services, being them object storage, file storage, table storage queue storage, and premium storage block storage. It is highly scalable, durable and high available. There are different pricing options, charging the user just for what is used.

There are much more public cloud platforms similar to the ones offered by Amazon, Google and Microsoft, being some of them IBM BlueMix, Oracle Cloud Platform or HPE ConvergedSystem. [20]

3.4. Your own cloud

There is a small section for a small parts of the cloud services that are available on Internet, being shown in this document the ones of Microsoft, Google and Amazon.

These clouds are private, but there are also private clouds that are deployable in hardware in order to use a infrastructure instead of paying for it to a company that offers it for a price.

There are different solutions which can be used to do that, which are listed on this section.

Private cloud platforms

Some of the private cloud platforms available nowadays are:

- OpenStack
    - As seen in the "What is OpenStack" section, a really complete, free and open-source platform to manage a cloud infrastructure.
- VMware vCloud
    - With a proprietary license, it is a product offered by VMware as a solution to manage a cloud infrastructure.

- Apache CloudStack
    - Apache CloudStack is a direct free and open-source alternative to OpenStack. It grows continuously but its much less supported than OpenStack.

4. Deploying OpenStack

The deployment of OpenStack is shown at the official documentation as a modular installation, where there are two different stage types.

The first stage is the deployment of the environment, which has to be done in all the nodes that are in the cloud that is going to get OpenStack. The second type of stage is the deployment of each one of the nodes required at the cloud.

4.1. Environment

There are several basic components that are required to be installed before installing OpenStack components, being the group of these called “Environment” at the official OpenStack deployment guide.

At the environment , it is required to:

- Configure the network on all the nodes, using two interfaces
    - Management interface:
        - Requires a gateway to provide Internet access to all nodes for administrative purposes such as package installation, security updates, DNS and NTP.
    - Provider interface:
        - Requires a gateway to provide Internet access to instances in the OpenStack environment.

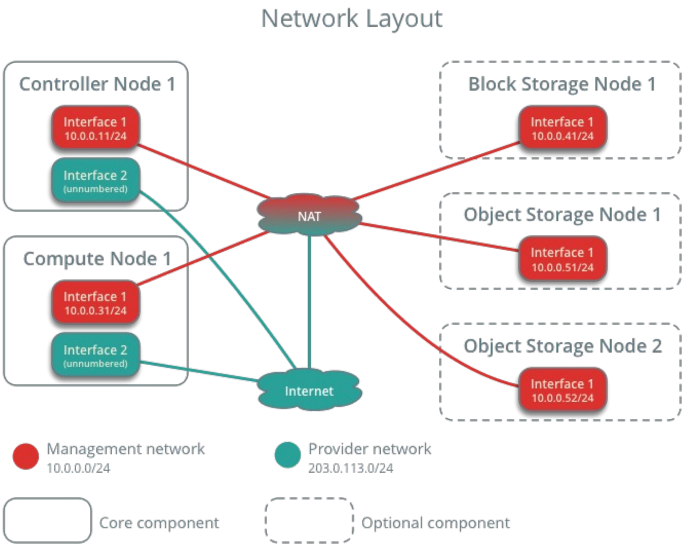


Illustration 10: Network layout example

The first part for deploying OpenStack is configuring the network on all the nodes. For doing so, there will be an static IP assigned for the management interface while the provider interface will not have any IP assigned, configured as a manual interface.

Once the network is configured and working, the name resolution of the nodes is configured in the hosts file of each one of them, relating each one of the IPs of the cloud with the corresponding node. This static name resolution could not be mandatory, but it makes the process of understanding the configuration files much more easier, since an user would be reading a name instead of an IP at them.

Network Time Protocol (NTP)

In order to properly synchronize services among nodes, it is required to set up a NTP service. At the official documentation it is recommended to use a software called Chrony for doing it.

Chrony needs to be configured for referencing a NTP server, being recommended to select one with the lowest possible stratum. Instead of having all the nodes querying to this server for the NTP data, it is recommended to configure Chrony in the nodes which are not the main controller in order to get this data from the controller node.

The stratum of a NTP server shows how direct is the synchronization with a reference clock. Here it is a nice illustration which shows what the stratum means.

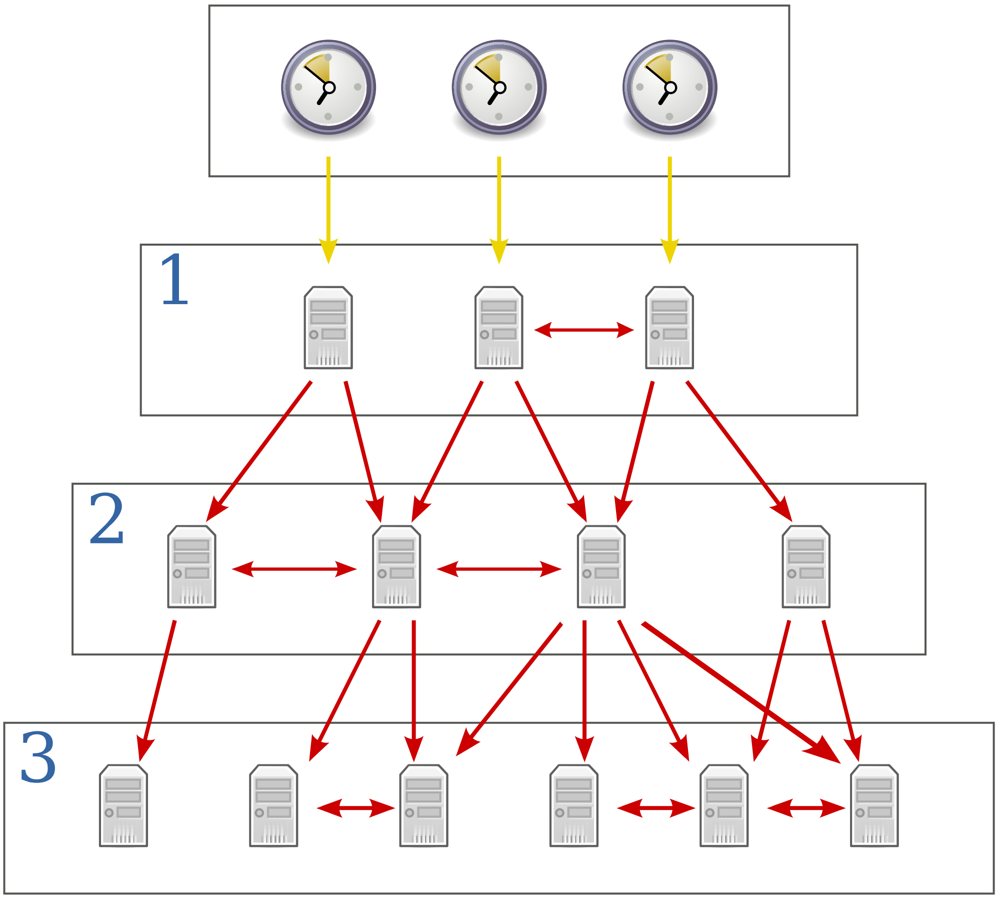


Illustration 11: NTP servers stratum

Image author: Benjamin D. Esham

OpenStack packages installation

It is required to install the OpenStack packages in all the nodes in order to use OpenStack software. In the case of Ubuntu 14.04, it is needed to add a repository in order to install them using the default package manager.

SQL database

Most OpenStack services use an SQL database for storing information. This database usually runs at the controller node and it can be run under different SQL systems like MySQL or PostgreSQL. [21]

NoSQL database

The Telemetry service needs a NoSQL database for storing its information, and this database also runs usually at the controller node. The official guide follows the installation with MongoDB. [22]

Message queue

OpenStack uses a message queue to coordinate operations and status information among the services. As the SQL and the NoSQL databases, the message queue service usually runs at the controller node. In the official guide RabbitMQ is used, but OpenStack also supports other systems like Qpid or ZeroMQ. [23]

Memcached

The identity service authentication mechanism uses Memcached to cache cache tokens. As the services mentioned before, the Memcached service usually runs at the controller node. [24]

4.2. Services deployment

Once the installation of all the components shown until now, it is time to proceed to another point of the deployment, which is deploying the OpenStack services that are required by the cloud administrator.

As mentioned in the environment paragraph, each one of the OpenStack services uses a database for storing information, so the first thing that has to be done for each service in order to deploy it is creating a database and granting privileges to the user on which the service will be running at.

By the time this document is written, there are 19 official OpenStack modules shown at their web portal. Each one of these OpenStack services will require at least installing several packages, managing the users that will be using it and the API endpoints of the service, editing one or more configuration files and restarting the system services in order to make them work without restarting the system. [25]

5. Why is an OpenStack deployer needed

Although the OpenStack project is really well documented, it is still not something easy to deploy, since we are speaking about several nodes and also several services which use a lot of configuration files and which depends on other services. In order to deploy a basic cloud it is needed to deploy at least the Identity, Image, Compute and Networking services.

The goal of creating this application is creating software which allows anyone to deploy an OpenStack cloud without making a bigger effort than configuring the network addresses of their computers, filling a small form in a webpage and clicking a button, making the deployment of OpenStack something extremely easy.

6. OpenStack deployers

This section shows several available software alternatives that are able to deploy an OpenStack cloud environment.

6.1. OpenStack-Ansible

OpenStack-Ansible uses the Ansible IT automation engine in order to deploy an OpenStack environment on Ubuntu Linux.

It is executed using commands in a terminal and uses SSH to perform the deployment, which has four main stages:

- Deployment host preparation
- Target hosts preparation
- Deployment configuration
- Deployment launching

[26]

6.2. Packstack

Packstack is a free and open-source deployment utility created by Red Hat to make it easier the deployment of a proof of concept cloud on one node, having the possibility of adding more nodes after that.

It works in Red Hat Enterprise Linux 7 and at equivalent versions of distributions like CentOS or Scientific Linux.

[27]

6.3. OpenStack Autopilot

Free, open-source and developed by Canonical, this deployer has a web interface which allows the user to deploy OpenStack in Ubuntu easily thanks to an user friendly interface, being it the direct alternative to this project.

[28]

Chapter 3: Different ideas for the deployer

There was several ideas in order to do it, having in mind three main installation schemas, three types of deployment schemas and three types of user interfaces.

1. Installation schemas

As mentioned before there are three main ides for the installation schema, changing at each of them the complexity of the interaction between the user and the system. These three installation schemas will be called “semi-guided”, “guided” and “automatic”.

1.1. Semi-guided

The user has a list of options that will help to install and configure each component of OpenStack step by step

1.2. Guided

The user has a guided installer that will provide a guided installation step by step, making the installation of OpenStack a fully guided and easy process.

1.3. Automatic

The system just request in one step for all the nodes and all the features that are on each node, deploying the cloud automatically.

2. Deployment schemas

There are several deployment schemas which would be acceptable to create this deployer, being the difference at all them the system where the deployer is executed. The options evaluated at this point are “PaaS”, “client-side” and “node-side”.

2.1. PaaS

There is a server which listens for requests from clients, performing all the tasks required for deploying OpenStack cloud on the given nodes.

2.2. Client-side

A single piece of software is in charge of deploying OpenStack. This software will run on a computer which will be in charge of deploying the OpenStack cloud on the given nodes.

2.3. Node-side

There are one or more pieces of software which are executed on each one of the nodes, performing the deployment on each one of them.

3. Interfaces

In order to choose one type of interface it is important to evaluate the different possible user interfaces that the application can use, being the ones considered for the deployer command line interfaces and graphical user interfaces.

3.1. Command Line Interfaces

That would be acceptable for all the schemas, since it would be possible to run the script showing a list of possible actions, to provide the user with a guide for installing OpenStack step by step or to just receive as arguments the data of the cloud nodes and the required services or configurations on each one of them.

It would be also possible to create user friendly command line interfaces by using libraries like ncurses, so the user would see a somehow beautiful interface instead of just the typical terminal interface.

3.2. Graphical User Interfaces

Web interface

In order to develop a web application for deploying OpenStack, it will be needed to follow a PaaS schema, since the frontend technologies are limited on several ways.

Desktop interface

As at a web application, it is possible to run the project with a PaaS schema, but at the same time, everything could be done on a single piece of software, since desktop software programming is much less limited than client-side web programming.

4. Decisions taken

The decision with the installation schema was more or less easy, since in order to make the installation as easy as possible as it was mentioned before, the automatic installation schema is preferred over the guided or the semi-guided.

Choosing a deployment schema was not as trivial as the installation one. The node-side deployment is directly discarded, since making the user to deploy OpenStack node by node is a tedious task compared to having to deploy everything from one single point. Having discarded it, there are two remaining options that would be perfectly valid, the PaaS schema and the client-side schema.

However, it is important to highlight that having a PaaS schema it would be possible to run a desktop application as it would be done with a client-side application, so the user would not perceive the difference of both them.

That means that designing the application with a PaaS schema would give the developers to have several configurations of it, developing one server and having the chance of easily developing several clients that would be calling this server, being this server or servers anywhere.

These advantages are considered beautiful enough in order to go with the a PaaS design.

As said, it would be possible to develop different clients in order to access this service, but in order to develop the first version of this software that will deploy OpenStack, it is important to decide one type of interface.

The gap of effort required to create a command line application, a desktop application and a web application is small, so the decision here is developing a web client, since it will be portable as long as the device where the system is used has a web browser.

Chapter 4: Application Design

1. Requirements

In order to create this application, a list of requirements was created first:

1) The application will be a web application.
2) The development will be based on the installation of OpenStack on a Ubuntu 14.04 cloud, but it must has features in order to make it prepared for developing modules for other GNU/Linux distributions.
3) The version deployed by the application will be OpenStack Mitaka.
4) The backend must be completely detached from the frontend.
5) The backend will act as an API which will deploy OpenStack based on the received requests.
6) The backend will deploy OpenStack sending to the nodes scripts written from scratch that will be driven by command line arguments.
7) It must be possible to test each one of the scripts that will be developed individually.
8) It must be possible to deploy OpenStack just by using the scripts on the nodes.
9) The backend code must not be responsible of a limit of nodes, so it must be designed to be flexible.
10) The frontend will send requests to an API which will be at the backend.
11) The frontend will add as much nodes as the user wants.
12) It must be possible to select which services are going to be deployed at each node.
13) A terminal will be shown at the frontend showing the progress of the installation in soft real time.
14) The structure of the system will follow a pattern, making it easier to develop the support of new services in the future.
15) The scripts which deploy OpenStack has to be as divided as possible, following the subsections of the official deployment guide as different scripts.
16) There will also be biggest scripts which function will be executing each one of the smallest scripts in order to do the tasks with less commands.
17) Each script will be classified using the goal, being the goal the name of the service that it will help to deploy.
18) The backend will have a main component that will call functions written at different modules.

19) Each module of the backend will have the task of deploying one service at a given node.

20) It must be possible to do deploy more than one cloud at the same time.

21) It must be possible to test each one of the components without the others.

22) At this first version, the Nova service will use KVM if it is available and QEMU if it is not.

23) All the passwords and tokens needed at an initial deployment must be generated at the environment deployment scripts.

With all the requirements and the design decisions taken until now, it is possible to build an UML sequence diagram which shows how the deployer will work.

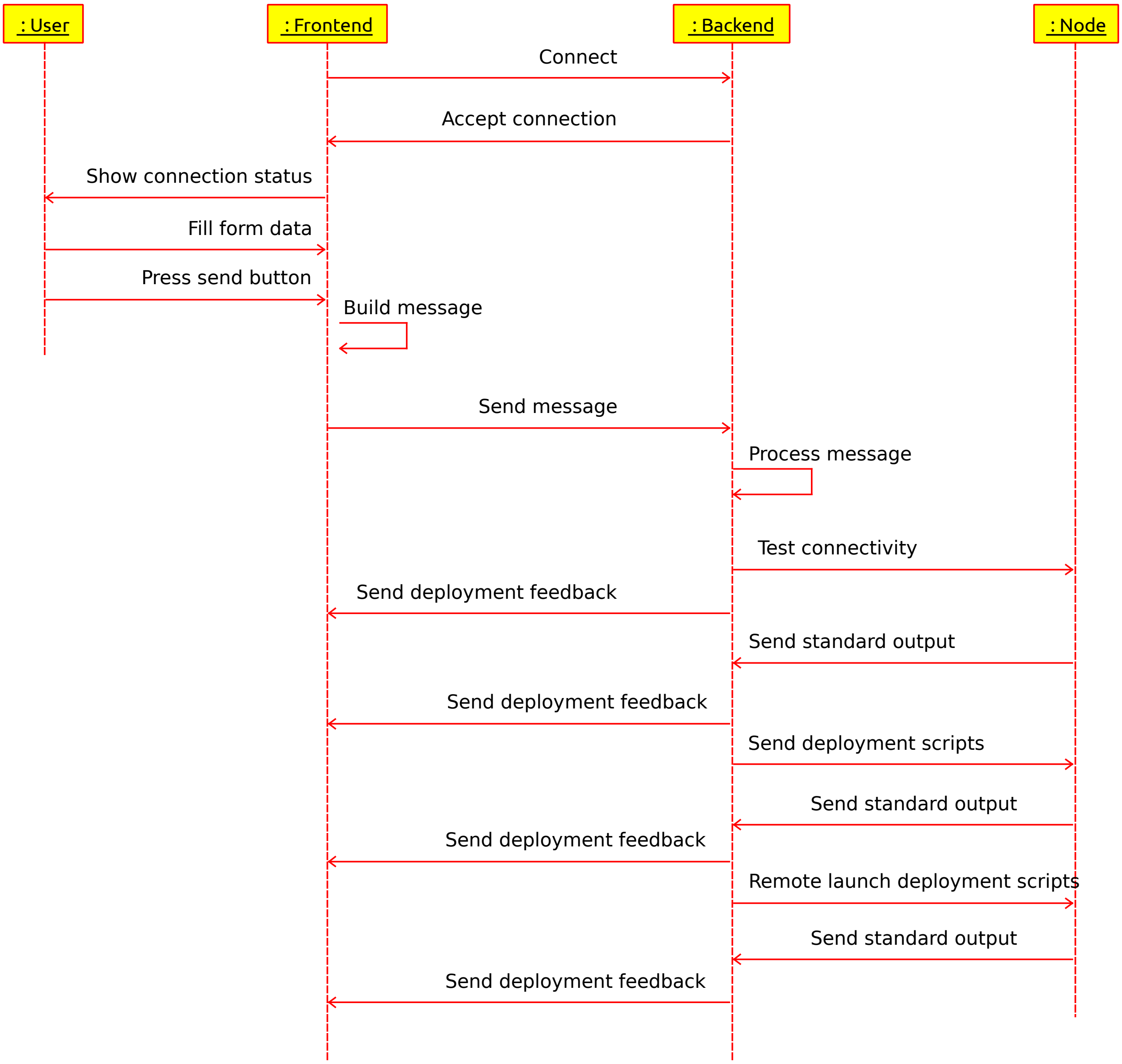


Illustration 12: Deployer UML sequence diagram

2. Technical decisions

It was considered that the most important part of the project is the backend, since as it is required at the seventh requirement, OpenStack would be deployable just with it, while just having the frontend we would have just something that send messages, so the backend was designed first.

2.1. Backend

Having the requirements list, several decisions had to be taken. Some of them, like choosing a language for the scripts that would be being sent to the nodes where the application will be installing OpenStack were relatively easy, while choosing the backend programming language, framework or messaging system for it were a little bit more complex.

The goal of the sent scripts will be installing the required packages, editing configuration files and executing what is needed in order to deploy the module for which the scripts are done for.

2.2. Scripts language

There are several scripting languages that are feasible for the task, some of them would be Python, Perl, Ruby or Bash. Since the main goal of these scripts will be editing files and opening system processes like package managers, choosing any of them would be a good option as long as the system where they are going to be executed has the interpreter available. However, taking this point into account, it is needed to highlight the fact that almost all GNU/Linux distributions have Bash as the default shell and almost all them have the Perl interpreter installed.

Using a compiled language would also be feasible, but it would increase the complexity of the development.

Since the application requires a backend which sends scripts to GNU/Linux distributions, the scripting language chosen for these scripts will be the Bash scripting language, since Bash is the default shell on almost all GNU/Linux distributions. [1]

2.3. Sending and controlling the deployment scripts

Since the scripts have to be sent from the backend service to each one of the nodes, a decision has to be taken for choosing the system that will be used for doing so.

Remote control protocol

There are several protocols that can be used to control nodes to perform the OpenStack deployment at them. In order to execute scripts in the nodes it is possible to use standards like SSH or Telnet. At the same time, it is possible to develop software to do the task using other protocols like SMTP or directly exchanging this information through our own protocol using sockets. However, doing it would not be a good idea since the complexity of the project would increase significantly while the functionality would not be better than using SSH or Telnet. [29]

Telnet

Telnet is a protocol that would accomplish the functionalities that are required for doing the task of running software in a remote machine and listening to the standard output of it. However, there is another protocol called SSH, which stands for Secure SHell. This protocol has all the basic functionalities that Telnet has, having at the same time extra features like traffic encryption or traffic compression, being it considered better than SSH.

SSH

A lot of distributions are ready for installing an SSH server or even have it preinstalled. SSH provides remote connection to hosts using a secure encrypted channel, making it secure in the case of having someone sniffing the traffic since it would be extremely difficult for him to decrypt the information.

Most of the mainly used distributions have also a lot of documentation to run SSH on them. This documentation can be checked for free in the following links:

https://wiki.debian.org/es/SSH

https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-openssh-server-config.html

https://wiki.archlinux.org/index.php/Secure_Shell

https://es.opensuse.org/OpenSSH

http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

SSH will also give as output the standard output of the system where the connection is being done, so it would be also be perfect for retrieving the information that we want to show to the user who is deploying OpenStack with the system that is designed. [6]

Decision

Having SSH all these advantages over Telnet like the encryption or having it preinstalled in a lot of GNU/Linux distributions made it the perfect option, so it is the protocol chosen for the project.

With the decision of using Bash to write the scripts and SSH to execute them on the different nodes, the 8th requirement of the application that requires OpenStack to be deployable just with the scripts executed at the nodes is fulfilled.

2.4. Protocol to communicate the backend and the frontend

In order to make clients perform requests to the deployment server which is the backend, it is a need to decide how will the messages be exchanged between the frontend and the API that will be created for controlling the OpenStack deployment.

Communication protocols

In order to decide how will be these messages, the design is studied from the point of view of the frontend, where we have to fulfill all the requirements being shown before on this document. The critical point which makes it needed to decide which technologies are going to be used for doing it is the requirement of having a terminal which shows in soft real time the output of the scripts which are executed at each node for deploying OpenStack. This leaves us with two basic options, which are AJAX and WebSockets.

AJAX

AJAX stands for asynchronous JavaScript and XML and allows a client-side JavaScript application to make requests to the backend of the application. It polls the server for checking if there is new information.

WebSockets

WebSockets allows the client-side application to open a permanent connection to a server, where the exchange of messages is done. This allows the developers to create real time applications with much more data since they do not have the overhead that AJAX has having to create an HTTP connection each time the information is updated.

Decision

Both options would be perfectly feasible for this project, since the volume of the information sent from the backend to the frontend will not be extremely big, being it just some lines of text each seconds.

Since both options are considered correct for this project, the point that is taken into account for the decision is using the technologies that the development team is more trained at, and during the Computer Engineering degree several practices are done using sockets in the “Distributed Systems” course, designing protocols and then developing applications that use these protocols previously written in paper. On the other hand, AJAX has not been used at all in the course.

WebSockets are really similar to sockets, and we can say that they even easier to use. Having a developer who is experienced using sockets, and also having a lot of public examples that show how to use WebSockets at the Internet, makes them the way to go for this application.

Using WebSockets is an idea that will work perfectly to send the output of the scripts which are executed at the nodes to the terminal which will be at the front-end. At the same time, the backend has to receive a message or a query with all the data of the nodes in order to start a deployment on them. It means, the request of deploying OpenStack can be done through the WebSocket but it does not necessarily has be done through it.

Different options can be taken into account for performing this request, like setting up a REST API, but, since there will be a WebSocket connection established to show the terminal data, the design decision of this point will be the simplicity, which is using the WebSocket that is already up for performing the request to the server.

Since WebSockets work exchanging messages, there is the need of designing a messaging protocol in order to be able to decode at the server the messages sent by the client.

3. Messaging protocol

The backend will listen on the websocket for a JSON message composed by a hash which includes the name of each node and the data of it. The data of each node will be a hash with the of both network cards of the system, an username, a password, the type of the node, which for simplicity will be the same one as the name of the node, and an array of services, with the names of the services that are going to be deployed at the node.

```
{
 "nodename1": {
       "ip": "provider_ip1",
       "management_ip": "management_ip1",
       "username": "root",
       "password": "root",
       "type": "nodename1”,
       "services": ["service1", “service2”]
 },
 "nodename2": {
       "ip": "provider_ip2",
       "management_ip": "management_ip2",
       "username": "root",
       "password": "root",
       "type": "nodename2”,
       "services": ["service2", “service3”]
 }
}
```

The different possible services will be:

“identity”, “image”, “compute”, “compute-controller”, “networking”, “networking-controller”, “dashboard”, “blockstorage”, “sharedfilesystem”, “objectstorage”, “orchestration”, “telemetry” and “database”.

Once the message is received by the backend, the deployment of OpenStack will start on all the nodes, sending the output of the scripts that are being used to the frontend of the application in plain text, so the application will just have to print to the user what it receives on the webpage.

It is also perfectly possible to design this part using technologies different than JSON, like XML or YAML, which are the two markups languages that are considered as the main alternatives to JSON. On the other hand it would have also been possible to send the messages in a binary format, like Base64 or MessagePack, compressing more the information and getting a faster performance than if the project were using a markup language.

Although encoding the messages in a binary format would perform better, and at the same time it would be possible to find advantages and disadvantages using XML or YAML over JSON, the time that the application is supposed to spent processing the message in almost any format is negligible compared with the time that it will spend deploying OpenStack.

JSON is a subset of JavaScript, making it easier to be used with this language. It is also lightweight, language independent and easy to understand, so it is a good choice for this part.

[30], [31]

4. Frontend technologies

Since the application designed is going to have a web based user interface, the visual development has to be done in HTML and CSS or any two languages which compile to them.

At the same time, it is required to use a script language for handling the WebSocket connections, and the only scripting language which is supported by all the web browsers is JavaScript. As said in the last paragraph it is also possible to use a language which compiles to JavaScript.

4.1. HTML, CSS and JavaScript or other languages

Since the developer of the application has taken the course “User Interfaces” where he has been trained in these three technologies and he does not have knowledge on any other like HAML, SCSS or CoffeScript, languages which compile to HTML, CSS and JavaScript, the project is going to be developed directly in HTML, CSS and JavaScript.

In order to access to the web components a JavaScript library called jQuery is going to be used in order to make this coding part easier.

5. Backend technologies

At this point of the design it is required to decide which framework or which system is going to be used for handling the message communication of the WebSockets at the backend.

5.1. Libraries

After performing a wide search, several technologies were found, like faye-websocket-ruby, java-websocket or simple-websocket-server, being them libraries that provide an interface for using websockets in ruby on the case of faye-websocket-ruby, java on the case of java-websocket or python, in the case of simple-websocket-server. [32][33][34][35]

5.2. websocketd

On the other hand, while doing this search, the designer of the project found a really interesting daemon called websocketd.

This daemon takes care of handling WebSocket connections, launching programs to handle the WebSockets and passing the messages between the programs and the web browser.

It is language agnostic, so it provides the developers the chance of use any language they want if it can be ran using the command line.

It also requires no libraries, since it just reads the incoming text from the standard input and writes the outgoing text to the standard output.

At the same time, each inbound WebSocket connection makes the program run in a dedicated process, being the connections isolated by process. This provides a really good easiness of use since it avoids the developer to worry about making the application multi-threaded, fulfilling automatically the 20th requirement.

It also provides several features that can be interesting for this project, like a static web server, program routing for having different programs on different URLs, out of the box SSL and origin checking for restricting which pages can make WebSocket connections. It is also available for several platforms, like GNU/Linux, OS X, Windows, FreeBSD, OpenBSD and Solaris. [32]

At the same time, developing an application with the design of this one using websocketd is something that makes the testing process extremely easier, since both the frontend and the backend can be tested in a completely isolated way. While the backend can be tested running the server software in a terminal and writing the messages that it would receive in the standard output, the frontend can be tested using a simple echo server that always sends as a reply the messages received.

5.3. Decision

Having websocketd all these features makes it a perfect tool in order to develop a project like this, so it will be used in order to handle the WebSocket communications at the project.

6. Backend language

At this point, the only decision that has to be taken is on which language will the backend be programmed.

6.1. Alternatives

The main languages the student who is going to develop the application is comfortable with are Java, C, C++ and Ruby, being him much more comfortable with Ruby at doing an application with this design.

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write. [36]

Ruby is available in GNU/Linux, OS X, Windows, OpenBSD, FreeBSD and Solaris, just as websocketd, so it will not have disadvantages in this project with portability compared to a platform independent language like Java.

In a lot of cases it is possible to install Ruby directly from the official operating system repositories, while in others like in FreeBSD, OpenBSD or Windows it is needed to get it from different sources shown in the following links.

https://www.ruby-lang.org/en/documentation/installation/

http://cvsweb.openbsd.org/cgi-bin/cvsweb/ports/lang/ruby/

https://www.digitalocean.com/community/tutorials/how-to-install-ruby-on-rails-on-freebsd-10-1-using-rvm

6.2. Decision

Ruby is a language which the project developer is comfortable with, it has several features shown at the alternatives section and it is the 12th most popular language in means of the TIOBE index by the time this document is written, meaning that it is a widely used language. These reasons are the ones which make it the language chosen in order to develop the project. [37]

Chapter 5: Development planning

The development will take place in two parallel projects, the backend and the frontend. Since there will be only one developer both them will be developed sequentially.

1. Backend

The backend will be composed by different components, being them task scripts, component scripts, deployer modules and the deployer itself.

1.1. Task scripts

These scripts will be executed in the nodes where the OpenStack cloud is being deployed, and will be in charge of executing the basic tasks that are shown in each one of the parts of the official OpenStack deployment guide.

1.2. Component scripts

As the task scripts, the component scripts will be executed inside the machines which will be part of the deployed OpenStack cloud. They will deploy the OpenStack components, so the tasks which will manage the component scripts will be as an example, deploying the Nova service or the environment. In order to do it, they will use the task scripts which are in charge of doing each one of the small parts needed to deploy a service.

1.3. Deployer modules

The installation of each one of the OpenStack components and of the environment will be developed in single modules.

1.4. Environment module

The environment set-up is a part which must be done in all the nodes where the OpenStack cloud is deployed at.

This module must distinguish if the node where it is being installed is a controller node or not.

It will also be in charge of sending the task and component scripts to the nodes, being it the first step to deploy the environment.

1.5. Service modules

Each one of the service modules will be in charge of executing the required task scripts done to deploy the corresponding service. There will be one service module for each one of the OpenStack services that the application will support and another one to deploy the environment.

1.6. OpenStack deployer

The OpenStack deployer is the main component of the backend. It will be executed with websocketd and it will process a JSON with all the information of the nodes, using then the required modules in order to deploy what is requested using SSH.

It will also be in charge of testing the connectivity to each one of the nodes, being it possible to develop this test inside this module or inside a new one.

In order to make the design simpler, there will not be Object-Oriented Programming in the project, so each one of the modules will just contain functions and/or variables.

1.7. Run script

The run script is a script which will start a websocketd server with the OpenStack deployer, being it the one responsible of the server initialization.

2. Frontend

The frontend part, which will be the client-side part of the application will be simple, since it will be just one HTML web page with a CSS file and a JS script.

2.1. HTML web page

The web page will contain two buttons, one to add nodes and another one to start the installation.

Each time the “Add Node” button is pressed, a new element will be added to a form where it will be possible to add the data of a new node, being this data the hostname, ip, username, password and the list of services that must be installed on the node.

Inside each one of the fields added to the form there will be a button which allows the user to delete it from the form.

When the user has finished adding the required nodes, it will be needed to press the “Start installation” button in order to start the deployment.

At the header of the web page there will be a part where it will be possible to see whether the WebSocket connection to the deployment server is up or not.

At the bottom of the webpage there will be a terminal simulation which will be used to show the deployment progress to the user.

2.2. CSS file

The CSS file will be in charge of giving a plain interface to the users.

2.3. JS script

The script file will handle the dynamic tasks that the web page needs, as making the buttons work and handling the WebSocket connection and message exchanging, showing the received messages in the terminal simulation.

Chapter 6: Time scheduling

In this section the time estimations done before starting the project are shown. As the backend and the frontend development are two tasks that can be developed in parallel, they will be shown in two different tables.

1. Backend prediction

Component name	***Time predicted***
Task scripts	12 days
Component scripts	5 days
Deployer modules	6 days
OpenStack deployer	3 days
Total time	26 days

Table 1: Backend time prediction

As the backend is designed, there are some tasks which are dependent of others, being the

2. Frontend prediction

The design of the backend is completely parallel thanks to the technologies which are used at it. In the other side, the frontend development will be designed sequentially since creating parallel development tasks for a small webpage as the one that is going to be created would not give big advantages to one developer.

Since all the HTML, CSS and JavaScript code will be developed in the same stage, there will be only one prediction for all them, which will be the webpage prediction

Component name	***Time predicted***
Frontend	4 days

Table 2: Frontend time prediction

Chapter 7: Work scheduling

Having all the time predictions done, it is possible to do a time scheduling in order to develop the application.

The development phase will start on July 1, 2016 and the work will be based on deadlines:

Component name	Deadline
Task scripts	July 18, 2016
Component scripts	July 25, 2016
Deployer modules	August 2, 2016
OpenStack deployer	August 5, 2016
Frontend	August 11, 2016
System	August 16, 2016

Table 3: Project deadlines

As it can be seen, there is no “Start date” at each one of the tasks. The reason for that is that it is decided to start with the next task once the previous task is finished, so if one of the phases is completed before the deadline, the developer must start with the next one.

Chapter 8: Testing

This section contains the procedures that are carried out in order to check the correctness of the developed code.

Since each one of the component is different, several different methods are designed for testing them.

In order to perform all the tests, a VirtualBox environment with Ubuntu Server 14.04 which simulates two nodes of the future cloud must be created. The virtual machines running under VirtualBox will use a hard disk with the immutable flag, so it is possible to test the deployment from scratch just restarting the machines

1. Task scripts

Each one of the developed scripts must be tested on one of the nodes of the VirtualBox environment at the moment the script is written, checking that the output of it is the one expected at the OpenStack deployment guide.

2. Component scripts

Once all the task scripts are developed, the component scripts will be written. These scripts must be tested as the task scripts, executing them in a VirutalBox environment with Ubuntu Server 14.04 which simulates one of the nodes of the cloud that would be being deployed.

When the component scripts of an OpenStack component are executed inside a node, the tests shown at the “Verify Operation” section of the OpenStack guide for deploying the service must be successful with the output expected at the guide.

3. OpenStack deployer and deployer modules

Initially a pack of dummy deployer modules will be written, such that the only function of these modules will be executing the following command into the nodes where the system is being deployed:

```
touch ~/servicename
```

If the OpenStack deployer shows that the wanted services are shown at the nodes where they are supposed to be, the dummy deployer modules will be replaced with the real deployer modules.

Once the real deployer modules are included in the deployer, the output of all the ssh calls must be printed in the standard output in order to let the tester know what is happening at each stage.

Having done that, the deployer will be executed at a terminal without websocketd, pasting correct JSON messages that will be supposed to deploy each one of the services, checking the correctness of the installation as the component scripts were tested.

4. Frontend

In order to check the message sending function, the frontend will be modified in order to print the JSON message in the console instead of sending it through the WebSocket.

The other elements of the frontend will be tested by checking if they follow the designed functionalities:

- The button “Add node” must add a node to the form
- The button “Start installation” must get all the data of the form, building and sending the JSON, which in this case is not being sent, since it is just being printed at the terminal
- The button to remove a node must remove a node from the form
- The icon or text that shows if the deployment server is up or not must be checked in both ways, with the deployment server up and with the deployment server down.

5. System

Once everything has been tested, it will be possible to connect all the components of the system, building the final application. When it is done, the same tests that were done with the OpenStack deployer at the command line will be done using the web interface.

Chapter 9: Budget

This section shows an estimation of the costs of that are generate developing this projects.

The costs are shown divided in two distinct categories.

These categories are:

- Personal costs
- Hardware and software costs

1. Initial budget

The first estimation of the costs of the project is shown at this section. If there are decimals numbers these are rounded upwards to the tenth.

1.1. Staff costs

Name	*Category*	*Salary / Hour*	*Estimated hours*	*Total cost*
Alejandro Baldominos Gómez	Senior engineer	30 €	25	750 €
Sergio Pérez Fernández	Junior engineer	15 €	190	2850 €
Total				3450 €

Table 4: Staff costs

1.2. Hardware and software costs

Product	*Price*	*Amortization Period*	*Usage*	*Total cost*
Lenovo ThinkPad E460	600 €	48 months	2 months	25 €
Lenovo Dock Station	160 €	48 months	2 months	6,70 €
Cherry G80-3000 keyboard	80 €	60 months	2 months	2,70 €
Logitech B100 mouse	9 €	36 months	2 months	0,50 €
Dell UltraSharp U2414H monitor	240 €	96 months	2 months	5 €
I5 2500k desktop with 8GB RAM	500 €	96 months	2 months	40,50 €
Renting two dual-Xeon dedicated servers with 128GB RAM	600€	--	3 months	1800 €
Linksys WRT54G router	100€	144 months	2 months	1,40 €
Office material (pencil and white sheets)	5€	-	-	5€
Other office costs (electricity)	40€ / month	--	2 months	80 €
Software licenses	0 €			0 €
Total				1966,80 €

Table 5: Hardware and software costs

1.3. Project total cost

Description	*Total cost*
Staff costs	3450,00 €
Hardware and software costs	1966,80 €
Subtotal	5416,80 €
VAT (21%)	1137,53 €
Total	6554,33 €

Table 6: Project total costs

2. Why is the cost of the software zero?

The project follows the same philosophy as OpenStack, which is the Free Software philosophy, so the software that is used for developing the software is free.

2.1. Kubuntu 15.10

Kubuntu is an official derivative of Ubuntu shipped with a KDE environment installed. It is complete, free and open-source.

It is being used at the desktop and laptop computers of the project.

Illustration 13: Kubuntu logo

http://www.kubuntu.org/

2.2. Ubuntu Server 14.04

Ubuntu Server is a free GNU/Linux distribution made for servers. It is supported by Canonical and based in Ubuntu.

http://www.ubuntu.com/download/server

Illustration 14: Ubuntu logo

2.3. Vim

Vim is a high efficient text editor which works under a command line interface. It is free and open source.

http://www.vim.org/about.php

Illustration 15: Vim logo

2.4. Ruby

Ruby is a dynamic programming language focused in simplicity and productivity. It is free and open source.

The library used for accessing to the servers using SSH is net-ssh, which is also free and open source.

https://www.ruby-lang.org/

Illustration 16: Ruby logo

2.5. jQuery

Presented with the sentence “write less, do more”, jQuery is a fast and small JavaScript library which makes it easier to do several tasks like manipulating HTML, handling events and a multitude of features.

Illustration 17: jQuery logo

2.6. websocketd

websocketd is a free and open-source daemon which takes care of handling WebSocket connections, making it easier to develop servers which use them.

http://websocketd.com/

2.7. OpenSSH client

In order to do the tests connecting to the nodes the OpenSSH client is used at the project, which is a free and open-source SSH client. It is the SSH client which is installed by default in Ubuntu.

Illustration 18: OpenSSH logo

http://www.openssh.com/

2.8. OpenSSH server

As at the client, OpenSSH is used at the nodes, but this time with the server version. It is the SSH server which is installed by default at Ubuntu Server.

http://www.openssh.com/

2.9. Konsole

Konsole is the default terminal emulator shipped with Kubuntu 15.10. It is used for all the tasks in the project different that browsing the Internet, writing this document or to do some of the tests of the system.

Illustration 19: Konsole logo

https://konsole.kde.org/

2.10. Firefox 41

Firefox is a free and open-source web browser designed by Mozilla, a global community which works to keep the Web open, public and accessible to all. All the JavaScript code will be tested with its engine.

Illustration 20: Firefox logo

https://www.mozilla.org/es-ES/firefox/new/

2.11. Writer

Writer is the text processing tool of the LibreOffice suite, a free and open-source office suite.

Illustration 21: Writer logo

https://libreoffice.org/

2.12. Umbrello

Umbrello is an UML diagram modeler program based on KDE technologies. It allows creating diagrams of software and other systems in a standard format. It is a free and open source project.

2.13. OpenStack

The goal of this project is creating a tool to deploy OpenStack so it is needed to use it, at least for testing the result. OpenStack is also free and open-source.

https://www.openstack.org

2.14. VirtualBox

VirtualBox is a free and open-source virtualization product for both home and enterprise use. It is actually being actively developed by Oracle.

https://www.virtualbox.org/

Illustration 22: VirtualBox logo

3. Hardware usage

There are several hardware components shown at the hardware costs, and each of them have an usage on this project.

3.1. Working system

The system where all the work is done is a Lenovo Thinkpad E460 laptop with a dock station, a keyboard, a mouse and a monitor. This is the system used in order to design, develop the OpenStack deployment tool and also to write this document.

3.2. Desktop with i5 2500k and 8GB RAM

This system is going to be used as a support to test the development of the tool continuously, using VirtualBox to run several Virtual Machines to simulate nodes.

3.3. Linksys WRT54G router

This is the router used in order to create a network to connect the laptop, the desktop and to also provide networking to the virtual machines hosted at the desktop.

3.4. Dual-Xeon dedicated servers

In order to test OpenStack in a real environment, it is needed to have two real servers to run it at them.

3.5. Office material

The initial design is not done using a computer, so there are office materials that are going to be needed like white sheets and pencils.

3.6. Other office costs

It is needed to pay the electricity that will be used at the office used for developing this software, so an estimation of the costs is included here.

Chapter 10: Conclusions

Having done this project gave me the chance of learning a lot about Cloud Computing, a term which I knew but which also was not clear at all for me before I started with this project.

At the same time, I had the chance of facing the problem of creating an application that deploys a complex distributed system like an OpenStack cloud, discovering technologies which I had never used before like WebSockets.

As it is going to be mentioned in the next section, it has been a nice chance to learn about OpenStack and cloud services, giving me skill that I would consider important if I work with these systems in the future, using them or developing them.

Chapter 11: Future work

A name will be decided for the project and it will be published with a license which has not been decided until now, but which will probably be the MIT license. The project will be hosted at GitHub, so anyone who wants to collaborate on it will be able to do so.

The first goal is supporting all the actual OpenStack services and fixing any bug found on the system.

Once all the OpenStack services are supported, the module called “deployer module” in the “Development planning” section will be modified in order to handle the message reception and the message decoding in different modules.

Taking into account the modular design of the application, the support of different OpenStack version and Operating Systems will be added.

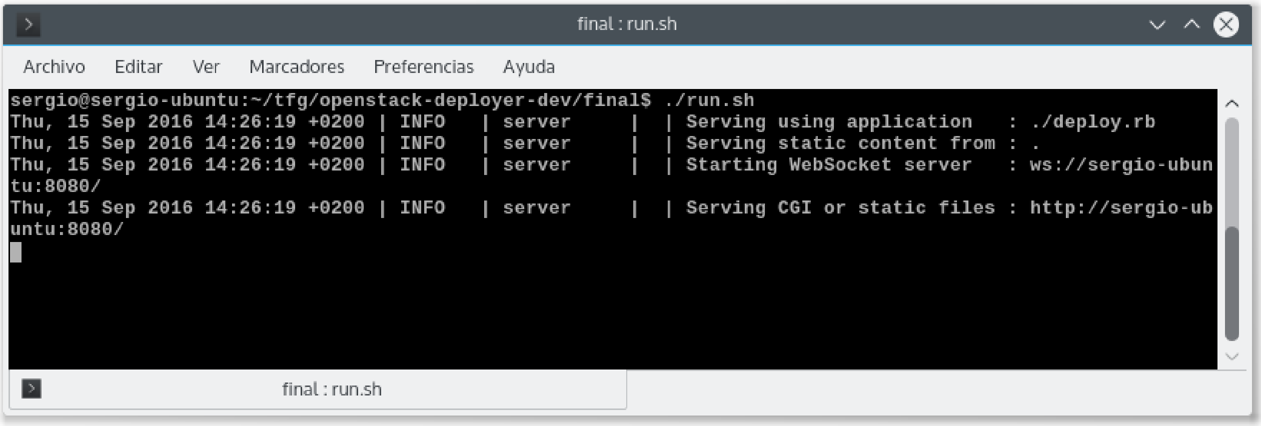
On the other hand, the designer of this project did not have enough knowledge about OpenStack when this project started, so, having done this project made him gain this missing knowledge that he had at the beginning, so it would also probably be a good idea to design a new and better OpenStack deployer based on everyone of the learned things at the project.

Chapter 12: Deploying OpenStack

At this point everything is known about the tool that is developed at this project, but there is a missing part which is also really important. This part is going to be shown at this section, which is how to setup a deployment server and also how to deploy OpenStack.

1. Running a deployment server

In order to run a deployment server, it is as easy as opening the script called run.sh



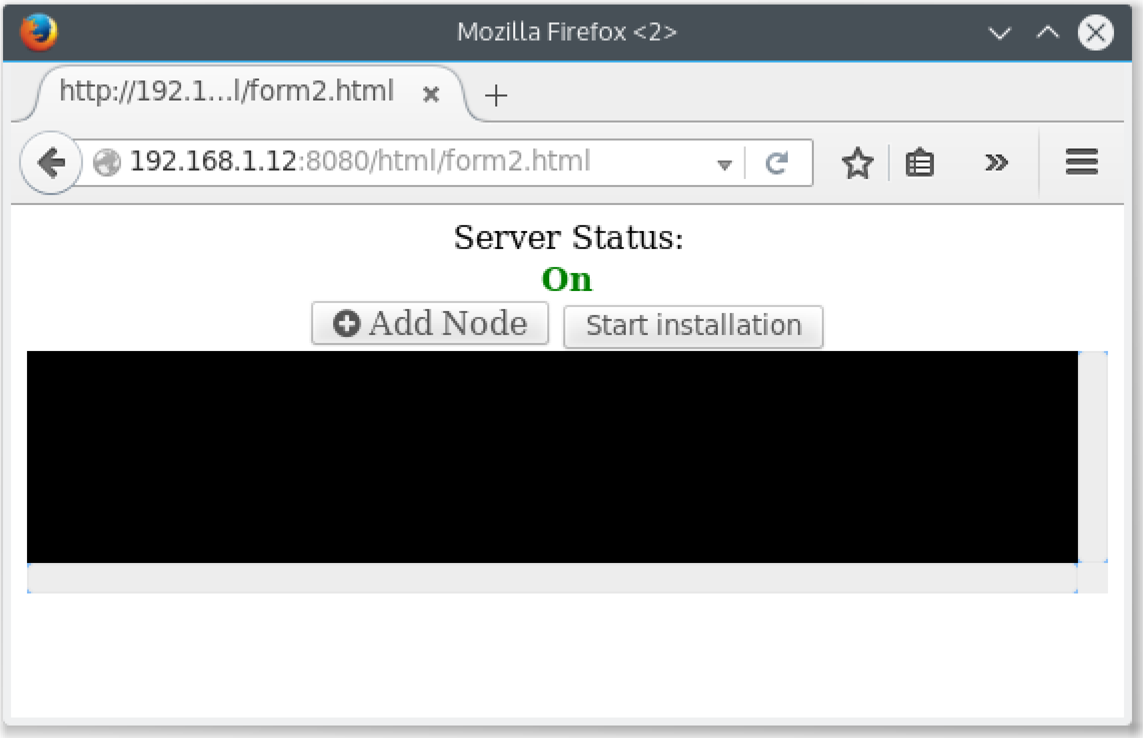
```
sergio@sergio-ubuntu:~/tfg/openstack-deployer-dev/final$ ./run.sh
Thu, 15 Sep 2016 14:26:19 +0200 | INFO   | server     |  | Serving using application   : ./deploy.rb
Thu, 15 Sep 2016 14:26:19 +0200 | INFO   | server     |  | Serving static content from : .
Thu, 15 Sep 2016 14:26:19 +0200 | INFO   | server     |  | Starting WebSocket server   : ws://sergio-ubun
tu:8080/
Thu, 15 Sep 2016 14:26:19 +0200 | INFO   | server     |  | Serving CGI or static files : http://sergio-ub
untu:8080/
```

Illustration 23: Running the deployment server

2. Deployment an OpenStack cloud from the web client

Deploying an OpenStack cloud using this tool is not much harder than running the deployment server.

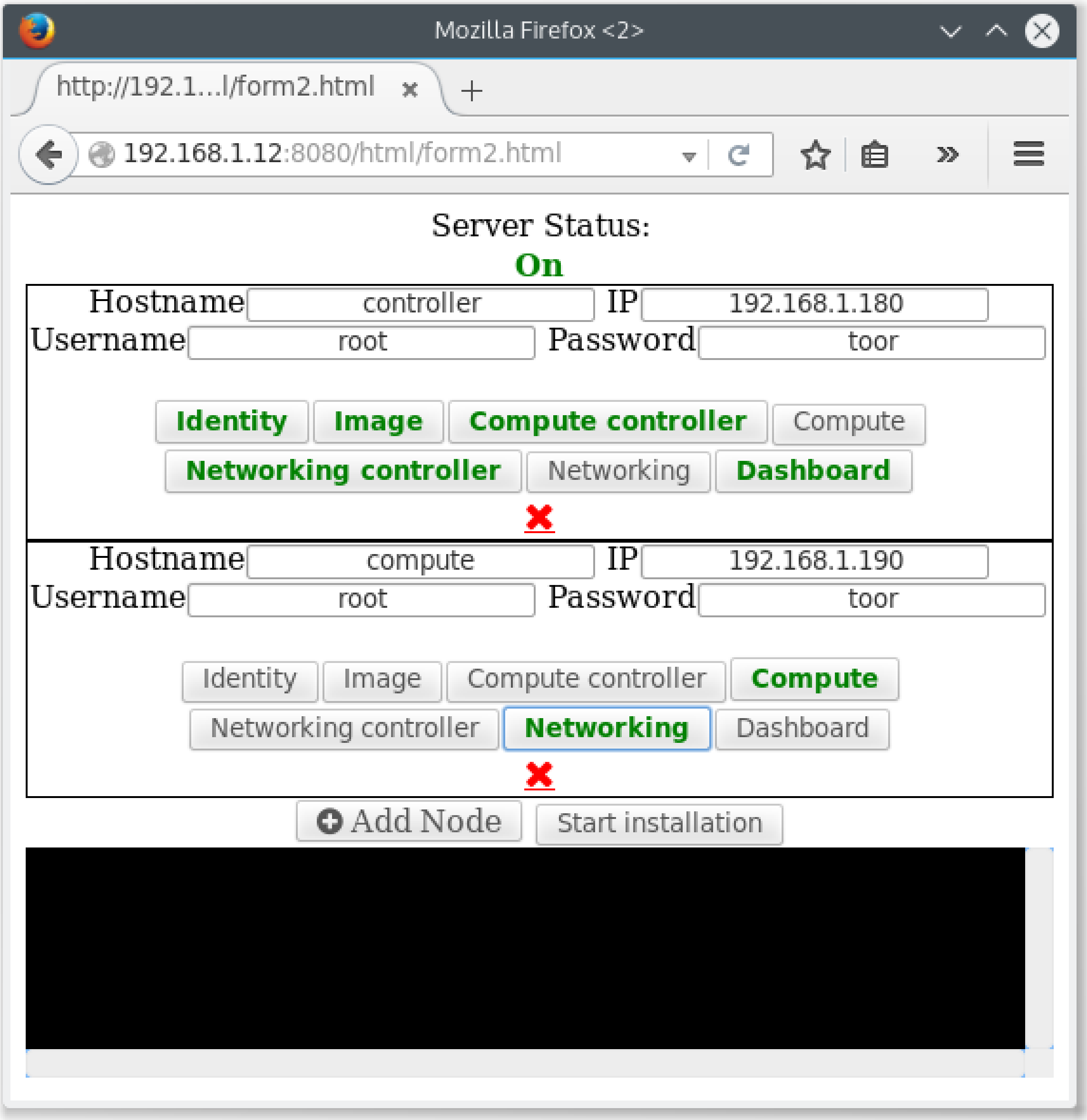
Firstly, it is needed to open the client, which in this case is an HTML + JavaScript webpage


Mozilla Firefox <2>
http://192.1...l/form2.html
192.168.1.12:8080/html/form2.html
Add Node
Start installation

Server Status:
On

Illustration 24: Empty frontend

Once the client is opened, it is possible to see if the deployment server is up or not. If it is up, the procedure is clicking the “Add Node” button in order to add new nodes to the cloud that is going to be deployed.



Server Status:
On

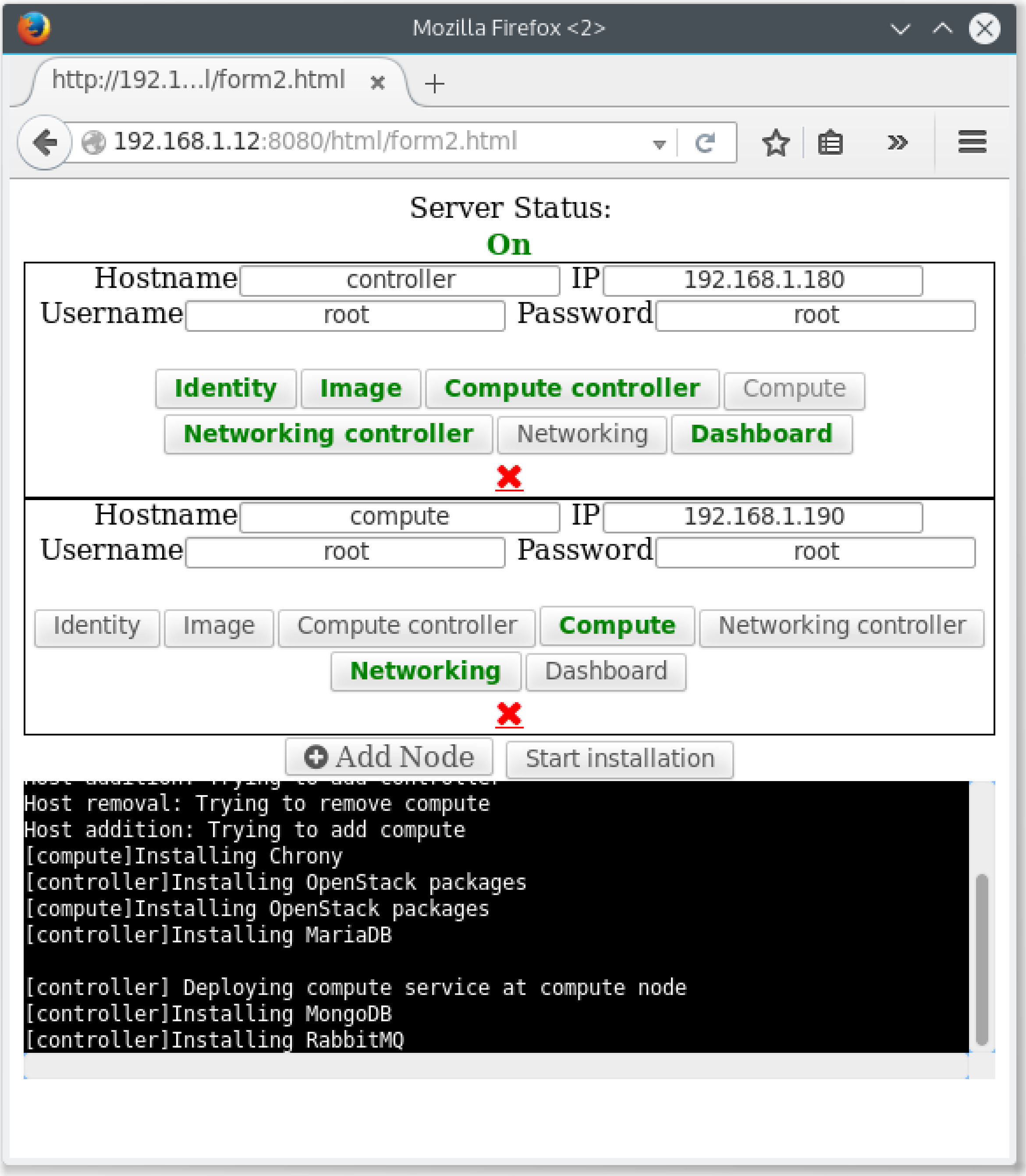
Hostname controller IP 192.168.1.180
Username root Password toor

Hostname compute IP 192.168.1.190
Username root Password toor

Illustration 25: Frontend with some nodes

Once a node is added, it is required to fulfill the required data and also to click on the name of the services that are going to be installed on each one of the nodes.

When the user is decided to install OpenStack in the nodes, it is just needed to press the “Start Installation” button, so the installation will start and it will be possible to see the output of the deployment process on the nodes at the terminal.



Hostname: controller
IP: 192.168.1.180
Username: root
Password: root

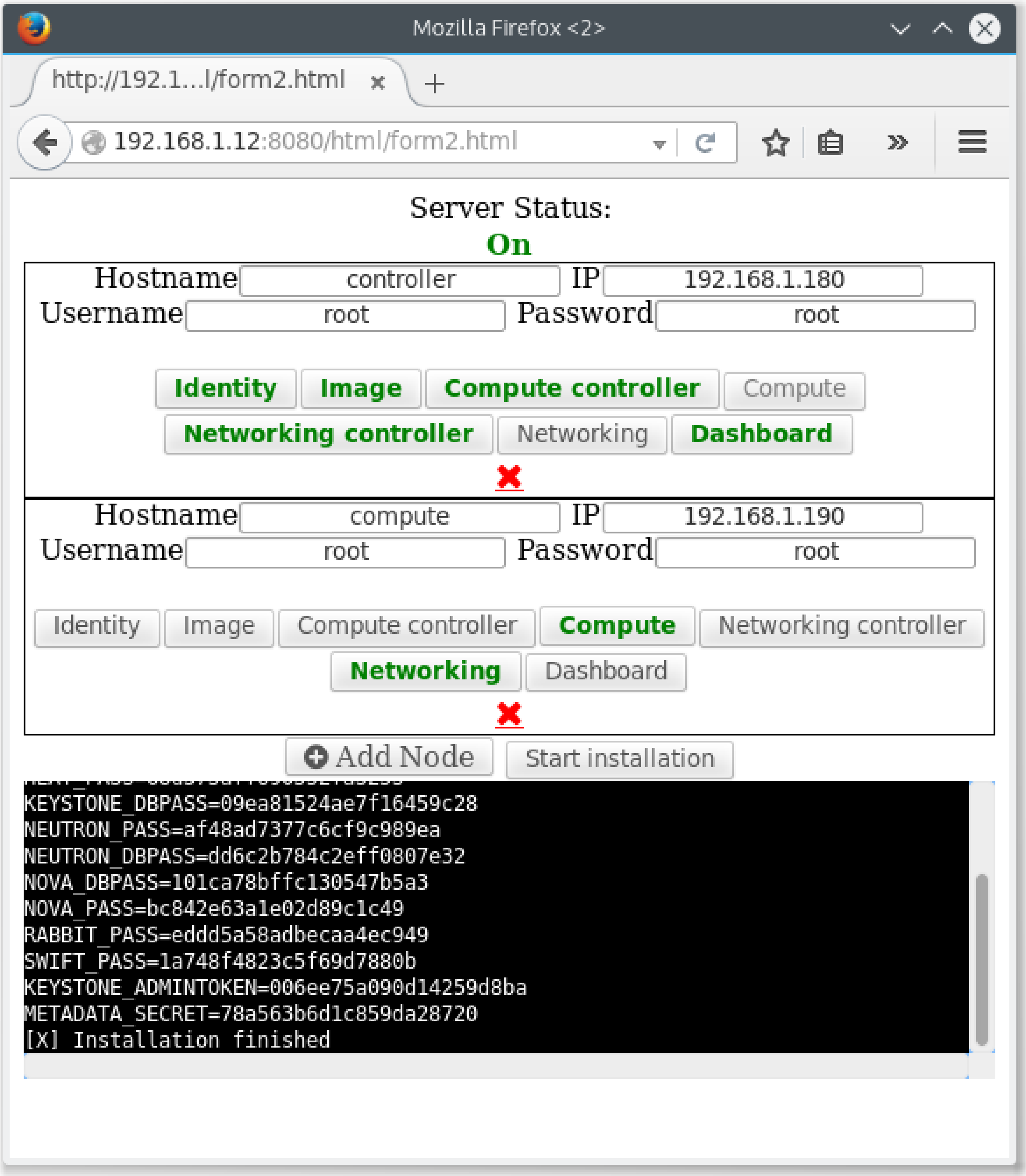
Hostname: compute
IP: 192.168.1.190
Username: root
Password: root

```
Host removal: Trying to remove compute
Host addition: Trying to add compute
[compute]Installing Chrony
[controller]Installing OpenStack packages
[compute]Installing OpenStack packages
[controller]Installing MariaDB

[controller] Deploying compute service at compute node
[controller]Installing MongoDB
[controller]Installing RabbitMQ
```

Illustration 26: Deployment in progress

The terminal will show all the keys and tokens used at the deployment with a message that show that the deployment has been finished



Server Status:
On

Hostname controller IP 192.168.1.180
Username root Password root

Hostname compute IP 192.168.1.190
Username root Password root

```
KEYSTONE_DBPASS=09ea81524ae7f16459c28
NEUTRON_PASS=af48ad7377c6cf9c989ea
NEUTRON_DBPASS=dd6c2b784c2eff0807e32
NOVA_DBPASS=101ca78bffc130547b5a3
NOVA_PASS=bc842e63a1e02d89c1c49
RABBIT_PASS=eddd5a58adbecaa4ec949
SWIFT_PASS=1a748f4823c5f69d7880b
KEYSTONE_ADMINTOKEN=006ee75a090d14259d8ba
METADATA_SECRET=78a563b6d1c859da28720
[X] Installation finished
```

Illustration 27: Deployment finished

As an example to test the OpenStack cloud that has been deployed it is possible to enter to the URL of dashboard at the node selected to have the dashboard service, which is: http://node_ip/horizon

By default there are two users created, which are admin and demo, and their passwords can be got from the randomly generated passwords which are shown at the terminal at the end of the deployment

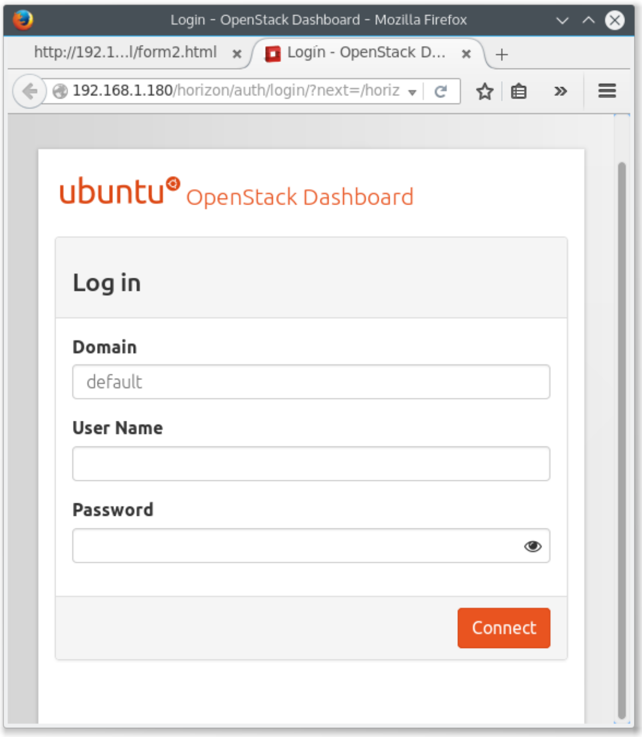
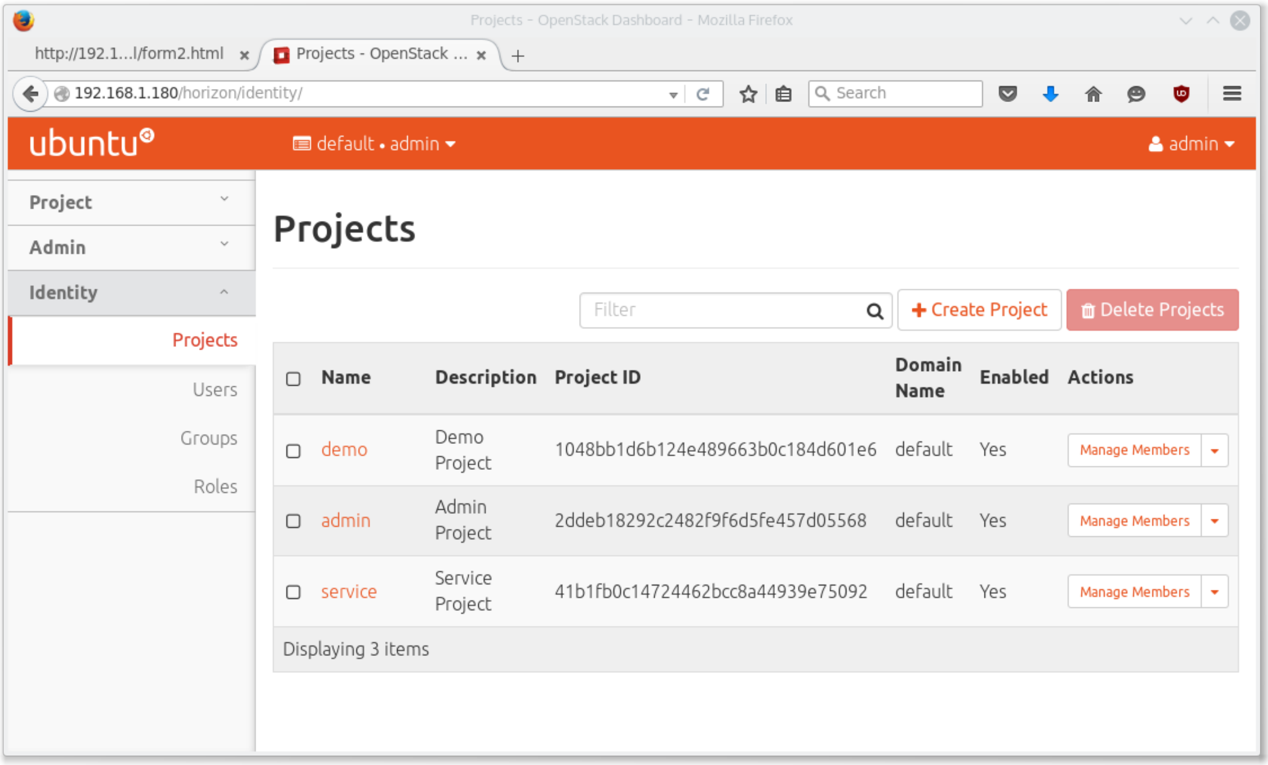

Login - OpenStack Dashboard - Mozilla Firefox
http://192.1...l/form2.html
Login - OpenStack D...
192.168.1.180/horizon/auth/login/?next=/horiz
ubuntu
OpenStack Dashboard
Log in
Domain
default
User Name
Password
Connect

Illustration 28: OpenStack dashboard login page



☐	Name	Description	Project ID	Domain Name	Enabled	Actions
☐	demo	Demo Project	1048bb1d6b124e489663b0c184d601e6	default	Yes	Manage Members
☐	admin	Admin Project	2ddeb18292c2482f9f6d5fe457d05568	default	Yes	Manage Members
☐	service	Service Project	41b1fb0c14724462bcc8a44939e75092	default	Yes	Manage Members

Displaying 3 items

Illustration 29: OpenStack dashboard

Bibliography

1: , RackSpace OpenStack portal, https://www.rackspace.com/es/openstack,
2: , List of companies supporting OpenStack, https://www.openstack.org/foundation/companies/,
3: , Official OpenStack faq, https://www.openstack.org/projects/openstack-faq/,
4: , KeyStone documentation, http://docs.openstack.org/developer/keystone/,
5: , Glance documentation, http://docs.openstack.org/developer/glance/,
6: , Nova documentation, http://docs.openstack.org/developer/nova/,
7: , Neutron documentation, http://docs.openstack.org/developer/neutron/,
8: , OpenStack database configuration guide, http://docs.openstack.org/mitaka/config-reference/database-service/database.html,
9: , OpenStack Mitaka deployment guide : NoSQL database, http://docs.openstack.org/mitaka/install-guide-ubuntu/environment-nosql-database.html,
10: , OpenStack Mitaka deployment guide : Message Queue, http://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html,
11: , OpenStack Mitaka deployment guide : Memcached, http://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html,
12: , OpenStack Project Navigator, https://www.openstack.org/software/project-navigator,
13: Peter Mell, Timothy Grance, The NIST definition of Cloud Computing,
14: Amazon, What is Cloud Computing?,
15: Amazon, What is AWS, https://aws.amazon.com/what-is-aws/,
16: Amazon, Amazon EC2 portal, https://aws.amazon.com/en/ec2/,
17: Amazon, AWS Marketplace Operating Systems, https://aws.amazon.com/marketplace/b/2649367011?searchTerms=&category=2649367011&page=1,
18: , Amazon S3 portal, https://aws.amazon.com/en/s3/,
19: Google, Google Compute Engine portal, https://cloud.google.com/compute/,
20: Google, Google Compute Engine documentation, https://cloud.google.com/compute/docs/images,
21: Google, Google App Engine portal, https://cloud.google.com/appengine/2011,
22: , Google Cloud Storage portal, https://cloud.google.com/storage/,
23: Microsoft, Microsoft Azure documentation portal, https://azure.microsoft.com/en-us/documentation/,
24: Microsoft, Microsoft Azure Virtual Machines portal, https://azure.microsoft.com/en-us/services/virtual-machines/,
25: Microsoft, Microsoft Azure Storage portal, https://azure.microsoft.com/en-us/services/storage/,
26: , OpenStack Ansible portal, http://docs.openstack.org/developer/openstack-ansible/install-guide/overview-osa.html,
27: Red Hat Inc., RDO QuickStart, https://www.rdoproject.org/install/quickstart/,
28: Canonical Ltd., OpenStack Autopilot portal, http://www.ubuntu.com/cloud/openstack/autopilot,
29: Dr. Kyle B. Wheeler, Telnet, http://www.memoryhole.net/~kyle/telnet.html,
30: w3schools, JSON Tutorial, http://www.w3schools.com/json/default.asp,
31: , JSON in JavaScript, http://www.json.org/js.html,
32: , WebSocketD portal, http://websocketd.com/,
33: , Simple-WebSocket-Server documentation and code portal, https://github.com/dpallot/simple-websocket-server,
34: , Java-WebSocket documentation portal, http://java-websocket.org/,
35: , faye-websocket code and documentation portal, https://github.com/faye/faye-websocket-ruby,
36: , RubyLang Portal, https://www.ruby-lang.org/en/,
37: TIOBE, TIOBE Index, http://www.tiobe.com/tiobe-index/,