



Diogo Luís Barradas Alves

Licenciado em Engenharia Electrotécnica e de Computadores

Implementation of a Private Cloud

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Pedro Miguel Figueiredo Amaral, Professor
Assistente, Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2016

Implementation of a Private Cloud

Copyright © Diogo Luís Barradas Alves, Faculty of Sciences and Technology, NOVA University of Lisbon.

The Faculdade de Ciências e Tecnologia and the Universidade NOVA de Lisboa have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

In memory of my father.

ACKNOWLEDGEMENTS

In the first section I would like to take the opportunity to express my gratitude to everyone I had the pleasure of meeting during the elaboration of my thesis and helped me reach my goals.

First and foremost I would like to thank my advisor, Professor Pedro Amaral, for the possibility of developing and building a cloud infrastructure, as well for promoting my fascination for telecommunications, networking and distributed systems.

I would like to express my extreme gratitude to Kátia Ludmila for believing in me and giving me a chance to teach at 4sfera and also to everyone at the academy, it was without a doubt one of the best and most enriching experiences of my life and for that I am ever grateful, not only the vote of confidence but also for receiving me into their family and treating me as one of their own.

To everyone I've met over the years they'll all have a special place in me, wherever I've worked, studied or had fun I'll carry forever with me the experiences we lived. To all the people I've met in college from day one Vanda Valentim, Liliana Sequeira, Ricardo Peres, and all of those that came after Tiago Silva, Diogo Tristão, Ricardo Rodrigues, Carlos Simão, Fabiana Paradinha, Vitor Rodrigues and Bruno Semedo thank you for making these years phenomenal. As I could not forget to my closest friends Teresa Rodrigues, Filipe Vale, Ricardo Silva, Pedro de Brito, Diogo Sardinha, Filipe Rodrigues, Pedro Rebelo, Tiago Cardoso, Miguel Ferreira and Tiago Cristovão thank you for everything you've done for me. To everyone mentioned above a sincere thank you as I am fortunate for having such a rich and wonderful foundation.

And last but not least, I would like to thank my family for everything they've done but in special to my mother who is my example of strength and perseverance without your sacrifices over the years this wouldn't be possible, to my brother who is one of the most hardworking, smart and driven person I've ever met and to a very special girl that made my last years at college easier, that has accompanied me since I've met her and I hope will accompany me for years to come.

ABSTRACT

The exponential growth of hardware requirements coupled with online services development costs have brought the need to create dynamic and resilient systems with networks able to handle high-density traffic.

One of the emerging paradigms to achieve this is called Cloud Computing it proposes the creation of an elastic and modular computing architecture that allows dynamic allocation of hardware and network resources in order to meet the needs of applications.

The creation of a Private Cloud based on the OpenStack platform implements this idea. This solution decentralizes the institution resources making it possible to aggregate resources that are physically spread across several areas of the globe and allows an optimization of computing and network resources.

With this in mind, in this thesis a private cloud system was implemented that is capable of elastically leasing and releasing computing resources, allows the creation of public and private networks that connect computation instances and the launch of virtual machines that instantiate servers and services, and also isolate projects within the same system.

The system expansion should start with the addition of extra nodes and the modernization of the existing ones, this expansion will also lead to the emergence of network problems which can be surpassed with the integration of Software Defined Network controllers.

Keywords: cloud computing; private cloud; hardware; elastic resources; Software Defined Networks; OpenStack;

RESUMO

O crescimento exponencial dos requisitos de hardware aliado aos custos de desenvolvimento de serviços online trouxeram a necessidade de criar sistemas dinâmicos e resilientes dotados de redes capazes de lidar com alta densidade de tráfego.

Um dos paradigmas emergentes denomina-se *Cloud Computing*, permite a implementação de um sistema elástico e modular de computação permitindo a alocação de recursos de hardware e rede dinamicamente, de maneira a responder às necessidades das aplicações web.

A implementação de uma *Cloud* privada com base na plataforma *OpenStack* implementa esta ideia. Esta solução descentraliza os recursos da instituição na medida em que é possível agregar hardware que está distribuído geograficamente e permite também uma otimização dos recursos computacionais e de rede.

Com isto em mente, a presente tese implementa um sistema de *cloud* privada capaz de elasticamente reservar e libertar recursos de computação, criar redes públicas e privadas que permite a comunicação entre instancias de computação, lançar máquinas virtuais que instanciam servidores e serviços e também isolar projetos dentro do mesmo sistema.

A expansão do sistema deve começar com a adição de nós e com a modernização dos nós já implementados, com esta expansão é esperado o surgimento de problemas de rede que podem ser superados com a integração de controladores *Software Defined Network*.

Palavras-chave: cloud computing; cloud privada; hardware; recursos elásticos; Software Defined Networks; OpenStack;

CONTENTS

List of Figures	xv
List of Tables	xvii
Acronyms	xix
1 Introduction	1
1.1 Background & Motivation	1
1.2 Challenges	2
1.3 Thesis Outline	3
2 State-of-the-Art	5
2.1 Cloud Computing	5
2.1.1 Service Models	6
2.1.2 Deployment Models	6
2.2 Available Platforms	8
2.2.1 Eucalyptus	8
2.2.2 OpenNebula	9
2.2.3 OpenStack	9
2.2.4 Comparative Analysis	10
2.2.5 Final Verdict	11
2.3 OpenStack	11
2.3.1 OpenStack Compute Nova	11
2.3.2 OpenStack Networking Neutron	12
2.3.3 OpenStack Image Service Glance	13
2.3.4 OpenStack Identity Keystone	14
2.3.5 OpenStack Dashboard Horizon	14
2.3.6 OpenStack Telemetry Ceilometer	14
2.3.7 OpenStack Object Storage Swift	15
2.3.8 OpenStack Block Storage Cinder	16
2.4 Software Defined Networks	16
2.4.1 OpenDaylight	17
2.4.2 FloodLight	18

2.4.3	Open Network Operating System (ONOS)	19
2.5	SDN Integration with OpenStack	21
3	Architecture	23
3.1	OpenStack Logical Architecture Overview	23
3.1.1	OpenStack Compute Overview	25
3.1.2	OpenStack Network Overview	27
3.1.3	OpenStack Identity Overview	29
3.1.4	OpenStack Object Storage Overview	31
3.1.5	OpenStack Block Storage Overview	33
3.1.6	OpenStack Image Overview	34
3.2	Openstack Physical Architecture Overview	35
3.2.1	Common Structures / Services	38
3.2.2	Hardware Specifications	39
4	SDN Architecture	41
4.1	Logical SDN Architecture	41
4.2	Physical SDN Architecture	45
4.3	SDN Controller Comparative Analysis	45
5	Results and Validation	49
5.1	Resource Optimization	49
5.2	Instance Deployment and Access	50
5.3	Use Cases	53
5.4	Discussion of Results	54
6	Conclusion and Future Work	55
6.1	Conclusion	55
6.2	Future Work	55
	Bibliography	57

LIST OF FIGURES

2.1	OpenDaylight Architecture [32].	18
2.2	FloodLight Architecture [33].	19
2.3	ONOS Architecture [35].	21
3.1	OpenStack Logical Architecture Overview [16].	24
3.2	OpenStack Compute Architecture.	27
3.3	OpenStack Network Architecture.	29
3.4	OpenStack Identity Concept Relation.	30
3.5	OpenStack Object Storage Architecture.	32
3.6	OpenStack Volume Storage Architecture.	34
3.7	OpenStack Image Architecture.	34
3.8	Implemented basic node deployment.	35
3.9	OpenStack suggested basic node deployment. [16]	36
3.10	Cloud Network Layout.	37
3.11	Network Traffic [20].	38
4.1	OpenStack Logical SDN Architecture.	42
4.2	SDN Controller Integration into OpenStack [39].	42
4.3	OpenDaylight to OpenStack communication.	43
4.4	FloodLight to OpenStack communication.	44
4.5	ONOS to OpenStack communication [41].	44
4.6	OpenStack Physical SDN Architecture.	45
5.1	Resource usage with two instances running.	50
5.2	Image launch configuration.	51
5.3	VCN access to an Ubuntu Server instance deployment.	51
5.4	SSH access to an Ubuntu Server instance running.	52
5.5	10 cirrOS instances running.	53

LIST OF TABLES

3.1	OpenStack Services.	25
3.2	Compute Service Component Overview.	25
3.3	Network Service Component Overview.	28
3.4	Keystone Service Concepts.	30
3.5	Object Storage Service Component Overview.	31
3.6	Block Storage Service Component Overview.	33
3.7	OpenStack Components [16].	35
3.8	OpenStack Hardware Specifications. [16].	39
3.9	Used Hardware Specifications.	40
5.1	Resource usage with two instances running.	50
5.2	Available resources after overcommit.	50

ACRONYMS

ACL Access Control List.

API Application Programming Interface.

AWS Amazon Web Services.

CC Cluster Controller.

CLC Cloud Controller.

DHCP Dynamic Host Configuration Protocol.

ext-br External Bridge.

FL Floodlight.

GUI Graphic User Interface.

HTTP HyperText Transfer Protocol.

IaaS Infrastructure as a Service.

IP Internet Protocol.

IT Information Technology.

JSON JavaScript Object Notation.

L2 Layer Two.

L3 Layer Three.

LDAP Lightweight Directory Access Protocol.

MAC Media Access Control.

ML2 Modular Layer 2.

NAT Network Address Translator.

NC Node Controller.

ACRONYMS

NIC Network Interface Controller.

NIST National Institute of Standards and Technology.

ODL OpenDaylight.

ONF Open Network Foundation.

ONOS Open Network Operating System.

OS Operative System.

OSGi Open Service Gateway Initiative.

PaaS Platform as a Service.

PC Personal Computer.

PNI Physical Network Interface.

QoS Quality of Service.

SaaS Software as a Service.

SAL Service Abstraction Layer.

SC Storage Controller.

SDN Software-Defined Networking.

SPICE Simulated Program with Integrated Circuits Emphasis.

SSH Secure Shell.

TCP Transmission Control Protocol.

VLAN Virtual Local Area Network.

VM Virtual Machine.

VNC Virtual Network Computing.

VNI Virtual Network Interface.

VTN Virtual Tenant Network.



INTRODUCTION

1.1 Background & Motivation

Computing as we know has evolved into a new paradigm, "**Cloud Computing**". In cloud computing data and computation tasks are fulfilled remotely in a "cloud" providing the delivery of hardware, system software and applications as a service over the Internet.

Cloud Computing is a fast-growing technology, that supplies the Information Technology (IT) industry demands for stable software and dynamic hardware. It enables large scale data processing and also allows universal access, ease of management and a lower starting investment since this solution can be deployed on a pay-per-use base.

In a conventional processing environment there is a need to scale up the resources (e.g. adding more RAM, disk space or CPU power) to satisfy demand but in a cloud environment the resources are scaled out, due to its elastic nature. This allows virtual instances to share workload between them, it is possible to replicate instances on demand. With the increase of the cloud complexity new problems such as network complexity can rise. To solve this dilemmas new network technology and tools are used. One of the most popular approaches is Software-Defined Networking (SDN), a model which enables a better control over the network layout, resulting in a better overall performance of the system.

In the academic IT environment there is a need for large data processing, simulation and service providing. These tasks can take weeks to run or be very hardware intensive. But when given access to a larger pool of computing resources it can be done in a few hours or days. While it is not conceivable to build a super computer the cloud enables the possibility of grouping several computers and combine their resources.

The cloud also enables the development of software or services without the concern for underlying hardware requisites or compatibilities, since it runs on a virtual layer the

resources can be allocated and released on demand thus removing the need for provisioning in early stages of a project.

Cloud Computing can also allow teachers and students alike to use its resources in the lab or in lectures, for instance professors can create virtual environments with the needed software for a project and every time a student needs to work on such project it is possible to access that instance and work on it, retrieving the results later.

A cloud platform can really greatly impact an educational institution by supplying on-line learning applications and environments, it supports mobile learning, scalability also reduces the costs of hardware and proprietary software making it a all around solution as a educational platform. The network access control and management has a huge impact not only on the users experience but also on the cloud performance so, network services should also be taken into consideration.

My motivation for this thesis was to be able to create a a private cloud by integrating the existing hardware in order to attain all the benefits of a cloud computing infrastructure in the department.

1.2 Challenges

The implementation of a cloud in an educational environment provides some interesting challenges, such as:

- Limited resources, what are the minimum resources to implement a private cloud into a scholar environment?
- Flexibility to change the underlying network, is it possible to build a private cloud and run a SDN Controller in this scenario, what would be the advantages of doing so?
- Use of open-source software. What are the best open-source solution available in the market and how would this approach affect the standard architecture?

For the first challenge, we discovered that when it comes to cloud computing services the hardware requirements are minor, so it is possible to build a simple cloud that can manage simple mundane tasks. The hardware requirements only have to scale to accommodate the workload requirements, so in a scholar environment it is possible to build a prototype cloud that can host a limited amount of instances, and then increase the hardware further to accommodate higher workloads.

Concerning the second problem, it is possible to run a cloud on top of a software defined network and it is also possible to assign all the network management to a SDN Controller, the later option is usually the best approach since it optimizes the network management and control.

As for the last challenge, a comparative analysis was made to outline the available open-source solutions and how these solutions would affect the implemented architecture.

This thesis goal is to build a cloud system using open-source software and the hardware available in the department. The primary objective is to use the hardware resources available and prepare the system for expansion, so it can deal with higher workloads and be more suitable to be used as a research and development tool of the teaching and student core.

Using OpenStack it was possible to implement a cloud architecture that had as an underlying hardware three HP Proliant servers that power the cloud one of the server is running as a cloud controller and the other two run as cloud compute nodes. The controller node is responsible for the synchronization of the services and communication between them and the compute node is responsible for the life cycle of the virtual instances that they are running. The open source software also allows the control and management of users/projects, network and orchestration.

The implemented model should be able to manage users, user access to projects, virtual networks and instances. It is also capable of creating and deleting private virtual networks on top of the physical network, deploy instances onto the virtual network, create virtual routers that establish connections between private and public virtual networks and also private to private virtual networks. Finally it is capable of running several Operative System (OS) desktop and server instances that can communicate in between them and the Internet, the architecture should also be able to load balance instances amongst similar computing nodes.

With large expansion in mind it is expected that network complexity and congestion problems arise, so a set of SDN solutions were outlined and taken into consideration how these solutions would affect the implemented architecture, and what would be the changes to be made in hardware and software configuration.

After the architecture was implemented several workload and access tests were made to validate the implemented system although only in a closed environment as it was impossible to test in a true production environment due to the lack of capable hardware.

1.3 Thesis Outline

The first chapter of this thesis explains the core concepts of the technology used, the motivation behind the development of the system, the challenges faced, and goals reached.

Chapter 2 depicts all technology integrated in this work as well as a comparative analysis between several open source solutions still present in today's market.

Chapter 3 approaches the physical implementation of the project, how the different modules interact and how they work independently from each other.

In Chapter 4 are included the results of the study regarding the integration of the SDN technology with the implemented system.

Chapter 5 aims to validate the overall performance of the system, the optimization of the hardware resources and a discussion of the results.

In the last Chapter are included all the conclusions reached and the work to be performed in the future of the project.

C H A P T E R



STATE-OF-THE-ART

Cloud computing is the latest trend in IT and it has reached the point where it is no longer an auxiliary utility as they are a cornerstone to our routine. Users are able to access these services from anywhere without any perception of where the service is or how it is delivered making it possible for software to be used by thousands of people at the same time [1].

2.1 Cloud Computing

The industry struggled several years with a true definition of cloud computation. The term appears for the first time in computer network diagrams and uses the cloud symbol to represent the Internet, the final definition appeared by the hand of the National Institute of Standards and Technology (NIST) that defined cloud computing [2] *“a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.”*

The cloud should be composed by five characteristics: **On-demand self-service**, users can allocate server resources and network storage automatically without the provider interaction; **Broad network access**, the services are accessed over the network with all sorts of client platforms thus promoting heterogeneity; **Resource pooling**, the resources are pooled dynamically using a multi-tenant model with the resources location being shadowed by a layer of abstraction; **Rapid elasticity**, the physical system resources can be scaled inwards or outwards on demand giving the user a sense of unlimited provisioning; **Measured service**, the system itself optimizes the use of resources this way resource usage can be controlled and reported giving the provider and the consumer transparency.

Google, which is one of the pioneers in cloud computing, defines six keystone markers

to this reality: **it must be user-centric**, once you are connected to the cloud everything you create from documents to images and applications belongs to you; **it is powerful**, the agglomeration of thousands of CPUs creates an unimaginable amount of computing power unable to be reached with a single Personal Computer (PC) and since all the data is stored in the cloud users can retrieve it from several machines this way not creating bottlenecks; **it is intelligent**, due to a high amount of data the access to it has to be done in an effective way; **it's programmable**, data must be replicated to protect its integrity so that if one computer goes offline the data it contains has to exist as a copy on another computer in the cloud [3].

2.1.1 Service Models

Every deployment of a cloud has three service models in them, these provide different levels of abstraction:

Software as a Service (SaaS): this is the highest layer of abstraction the user has no control over the network, operating system or hardware he is leasing, the application is accessible by the client device (ie. Smartphone, computer or tablet) or a program interface allowing the user to control only specific application settings [2, 4].

Platform as a Service (PaaS): the PaaS allows users to develop applications with tools supported by the provider without worrying about the underlying infrastructure and just like in the SaaS model there is no control or access to the lower layers of complexity henceforth the user has no control the network, physical layers or the architecture [5].

Infrastructure as a Service (IaaS): this is where consumer gain access to computer resources such as processing power, memory and storage so they can set up and run their applications, this model contrasts with the others since it is the lowest layer of abstraction giving this way access to the hidden infrastructure [2].

Cloud computing supplies a simplified software installation, maintenance and centralized control over the software that is running, so that this way the end-users can access the cloud services “anytime” and from “anywhere” just as long as they have a reasonably amount of bandwidth [6].

2.1.2 Deployment Models

The NIST also refers four possible deployment models with different characteristics to serve a different group of users, the models are the following [2]:

Public Cloud: This infrastructure is available to the general public, the resources are leased as services with a pay-per-usage model. The users can have their resource pool expand outwards or inwards on demand removing the hardware requirements

out of the equation. These clouds are managed by a third party organization that guarantees the Quality of Service (QoS) as long as the users have enough bandwidth to cope with the cloud traffic. Some of the biggest advantages of this model are **continuous uptime, on-demand scalability and the inexpensive setup** but, since it is a public cloud, it has drawbacks in **data security and privacy**. This is due to the fact that the location of where the data is stored or backed is unknown so there is a possibility of being accessed by unauthorized users. The biggest beneficiaries of this deployment model are smaller organizations that outgrew their infrastructures and use a public cloud to store information and services while upgrading their infrastructures or simply just migrate into a public cloud infrastructure [7].

Private Cloud: On a private cloud the access is restricted to an organization or a third party that has been given access. The main objective of this model is not to supply cloud services to the general public but to have the organization data spread around several points of interest. This cloud is hosted by the same company that uses it therefore, it provides more security than a public cloud. A private cloud enables the use of data center unused resources. These can be sold as services or computing resources. This way a private cloud allows companies to have a greater control and generate profit with their computational resources. The power of this power comes from the fact that it can maximize the hardware power using virtualization, reducing costs and complexity. Since the most important asset a company is its information, trusting this to an outside entity that can be vulnerable to cyber attacks is a major problem that led today's industry to shift to private clouds [2, 7].

Community Cloud: A community cloud aggregates the sustainability from Green Computing, the distributed resource provision from Grid Computing, the control from Digital Ecosystems and the self-management from Autonomic Computing. It competes with vendor clouds and makes use of its user resources to form a cloud with the nodes taking the roles of consumer, producer and coordinator. This concept removes the dependence of cloud suppliers. This cloud is a social structure due to the community taking ownership over the cloud. As the nodes act in self-interest having one control center would be unpractical, so it is nuclear to reward the nodes in order to redirect their computing power to the cloud and not themselves. In a cloud community each user would have an identity that supplies a service or website to the community. Since this cloud model isn't owned by a company it is not attached to that company lifespan, therefore it ends up being more resilient and robust to failures and immune to system cascading failures. The biggest challenge faced by this model is the QoS since it is a heterogeneous system the nodes will most likely need to reach critical mass to satisfy all the QoS requirement from different systems [8].

Hybrid Cloud: A hybrid cloud is the consolidation of a private and public cloud, it is an

intermediate stage of a company lifecycle. When a corporation tries to supply its services to the public, it can expand their data center and host a public cloud, bringing several benefits such as: **Optimal utilization**, since the data centers can be at times in an idle state and have only a small fraction of their resources being used, hybrid clouds increase the servers utility by making some of their resources public; **Data center consolidation**: since the cloud has expanded to a hybrid model the servers will consolidate their instances, reducing the operation costs; **Risk transfer**: with a hybrid cloud model the network workload management shifts from the operator to the vendor which is prepared to deal with higher workloads; **Availability**: a data center needs redundancy, backups and geographic dissemination this can be a very expensive to maintain so in a hybrid cloud environment the public counterpart can scale up and take the private cloud side if the company cloud goes down due to a failure or an attack [9].

2.2 Available Platforms

There are several open-source cloud platforms that enable the customization of the cloud and provide flexibility. These have features that enhance scalability, portability, flexibility and on-demand services. The three most used cloud platforms are Eucalyptus, OpenNebula and OpenStack so the following section will discuss the advantages and drawbacks of these three platforms [10, 11, 12].

2.2.1 Eucalyptus

Eucalyptus [10, 12] is an acronym for Elastic Utility Computing Architecture for Linking Your Program To an Useful System. It is used to deploy private and hybrid cloud solutions and it is developed by the company Eucalyptus Systems. This cloud platform implements an IaaS model and it became the first open source software compatible with Amazon Web Service Application Programming Interface (API). This platform has a sizable collection of virtualization technologies this enables the cloud to incorporate resources without modifying the base configuration. The Eucalyptus cloud is armed with five components: **Cloud Controller (CLC)**, the entry point for the administrators, project managers and users, this module also helps with the management of the virtual resources; **Cluster Controller (CC)**, this controller runs on a machine and is responsible for the management of the VM network so the controller must be in the same subnet has the nodes; **Storage Controller (SC)**, this controller provides the block-level network storage; and **Node Controller (NC)**, the NC is installed in all the computers and is used to monitor VM activities, like execution and termination of instances.

The main benefit of this platform is its efficiency, organization and agility but it also has some drawbacks for instance its scalability is lackluster when compared to other clouds and also some of its modules code are closed [11].

2.2.2 OpenNebula

OpenNebula [11, 13] was designed to help companies build a simple, cost-effective, reliable and open enterprise cloud on the existing IT infrastructure. It provides flexible tools that orchestrate storage, network, and virtualization technologies. It has a flexible design and it is modular to allow integration with different storage and network infrastructure and hypervisor technologies. This platform is mainly used as a virtualization tool, but it also has support to hybrid cloud besides private cloud, thus combining local infrastructure with public cloud-based structures enabling highly scalable hosting environments. The platform orchestrates storage, manages network, enables virtualization, resource monitoring and cloud security as it also deploys multi-tier services, namely computing clusters that work as VMs on distributed structures, combining resources of the data centers with the remote cloud.

OpenNebula [13] has three main layers: **the driver layer**, this is responsible for starting and shutting down VM, allocating storage for VMs and monitoring the operational status of the physical resources by communicating directly with the underlying operating system, thus it works as an abstract service to shadow the underlying infrastructure; **the core layer**, manages the VMs full life cycle, including setting up the virtual network dynamically so that allocated dynamic Internet Protocol (IP) address for the VMs do not overlap and managing VMs storage allocation; **the tool layer**, enables the communication with users and provides a control point to manage VM through. A scheduler manages the functionality provided by the core layer.

OpenNebula toolkit main features for integration are management, scalability, security and account. OpenNebula also claims to have interoperability and portability giving the providers a choice of several cloud interfaces [12].

2.2.3 OpenStack

OpenStack [12] is the quickest evolving open source platform for cloud computing, emerging in 2010 by the hand of NASA and Rackspace with the merging of Rackspace "Cloud Files" platform and NASA "Nebula" platform. This platform has been deployed by global enterprise customers whose processes and data stores are measured in petabytes the data therefore cannot be store the traditional way, it is stored in a distributed storage static data systems such as images of virtual machines, files, backups and archives in addition. OpenStack provides greater scalability, redundancy and durability, objects and files are stored on multiple disks scattered in the data center, providing replication and data integrity.

OpenStack [14] has an IaaS model where the services are modules and consists of many components working together that have open APIs and it is also possible to manage resources from the dashboard. The platform enables flexible network models for various applications or groups of users. It has the ability to manage network addressing, it allows the attribution of static and dynamic IP addresses and also allows the users to dynamically

redirect traffic to another compute nodes, which can be useful during maintenance or in case of failure. Users can create their own networks, control traffic and connect machines to one or more virtual networks.

2.2.4 Comparative Analysis

The comparative analysis of the aforementioned platforms will be based on several factors such as: released date, community support, the underlying architecture, cloud implementation, source code language, compatible hypervisors, Virtual Machine (VM) migration possibility, etc. [10, 15].

The Eucalyptus and OpenNebula were established in 2008 with the first being the result of a research project of the university of California computer science department and OpenNebula was funded by an European grant and the help of the companies Research In Motion, Telefonía and China Mobile. OpenStack only appeared in the summer of 2010 by the joint project of NASA and RackSpace. As of today the OpenStack platform is the one who gathers more support since it is backed by the giants of IT (around 850 companies) of which we can find names like HP, Dell, IBM, RackSpace, NASA, Cisco, AT&T which take part in the project. OpenStack community integrates around 7000 contributors over 87 countries [10, 11].

Eucalyptus architecture is hierarchically built with the Cloud Controller being on top and the Node Controller on the bottom of the hierarchy this way requiring at least two servers, the OpenNebula has a centralized architecture with three modules in its core while the OpenStack setup is based in four modules Nova, Neutron, Glance and Swift displayed in a classical node architecture having a front end and group of compute nodes running VMs in the back, all the nodes are required to have a physical network to connect them to the front end [10, 15].

All the three platforms are open source but while Eucalyptus and OpenNebula only deploy the private and hybrid cloud models and OpenStack also implements the public cloud model [15].

Based on the source code languages OpenStack is more restricted as it only supports Python while Eucalyptus also supports Java and C and OpenNebula supports Java, Ruby and C++ [15].

The three platforms support Linux and several of its distributions while Eucalyptus is the only one that is strictly Windows based [11, 15].

OpenNebula uses as a database management system MySQL, while Eucalyptus uses PostgreSQL to store metadata and user information and the OpenStack Nova module supports any database that is supported by SQLAlchemy [10].

Eucalyptus uses Euca2ools to bundle, unbundle, register, deregister, describe, download and upload images while Glance provides OpenStack the ability to discover, register and retrieving VM images. OpenNebula uses image repositories or datastores to allow users to setup images [11].

Eucalyptus does not allow virtual machine migration while OpenNebula and OpenStack have that capability [10].

2.2.5 Final Verdict

Based on the analysis done in the previous subsection and on the following literature [10, 11, 12, 15], there is no need to use proprietary cloud platforms when the open source platforms have so much support and capabilities, mainly due to the large communities to back them up. At first sight the platforms stand all at the same level and only differ in purpose or in community support, with OpenStack development focus being, the creation of an open source software that allows researchers and administrators to deploy IaaS and to manage virtual machines on top of the available computing resources, it seems this platform provides the best support for this thesis final objective which is the implementation of a private cloud, this coupled with the enormous community backing it up and an astonishing development rhythm makes it the perfect tool to implement the cloud.

2.3 OpenStack

The OpenStack [16] project enables the implementation of massively scalable and featured rich public, private and hybrid clouds using a set of interrelated services to provide an IaaS solution with each service offering an API to enable an easier integration.

The first release of OpenStack was on October 21, 2010 [17] and almost six years and twelve versions later the OpenStack Foundation is getting ready to release on October 6, 2016 the Newton version.

A standard deployment of an OpenStack [16] cloud integrate core modules which supply a wide range of services, these core modules are: OpenStack Compute **Nova**, OpenStack Networking **Neutron**, OpenStack Image Service **Glance**, OpenStack Identity **Keystone** and OpenStack Dashboard **Horizon**. On top of the previously mentioned modules, a cloud implementation may also include the OpenStack Object Storage **Swift** or OpenStack Block Storage **Cinder**, Openstack Telemetry **Ceilometer** and Openstack Orchestration **Heat**.

2.3.1 OpenStack Compute Nova

The OpenStack Nova [18] has a modular architecture composed of six components: nova-api, queue, nova-db, nova-conductor, nova-scheduler and nova-compute these components interact with each other to manage virtual machines.

The nova-api is the endpoint for user interaction, it supports OpenStack Compute API. Queue is the communication point for the nova components, it is a message passing mechanism among the components. Nova-db is the SQL database for state instances and also maintains information about the infrastructure. Nova-conductor works as a mediator

and facilitator for interactions between nova-compute and the database, it eliminates the direct access of nova-compute to the cloud database thus eliminating security problems. The nova-scheduler is responsible for VM handling, it accepts requests for virtual machine instances from the queue and using the scheduling algorithms assigns that VM to a compute server host to run it. Nova-compute is a daemon responsible for the life-cycle of virtual machine instances by using the hypervisors APIs supported by the virtualization technology [18].

Nova-compute receives user requests with instructions to performed from the queue, it executes system commands according to the received request and also updates the database if needed by communicating with it via the nova-conductor service. In a more in depth scenario we can say that the compute infrastructure in Nova works on top of the physical layer managing the resources available for virtualization, this infrastructure is elastic and can scale with the variation of resources. The compute infrastructure is divided in zones for the purpose of providing physical isolation granting therefore fault tolerance, within each zone exists several physical nodes that have the nova-compute service running, based on scheduling algorithms defined in nova-scheduler VMs are provisioned to these hosts. Within each host there is also a hypervisor and middleware that are used when a VM is assigned [18].

2.3.2 OpenStack Networking Neutron

OpenStack Networking or namely Neutron [19] is an API driven service that is used to define, manage networking and addressing it in the cloud itself, enabling the control of the network configuration. With Neutron it is possible to define, separate or merge networks on demand. Neutron is an abstraction layer as it does not implement any networking functions, this module only provides methods, agents and plugins, that can be used. The Neutron project provides a plugin mechanism that enables different networking implementation and consistent high-level API for abstraction, there are also API extensions that enable control over security, compliance, monitoring and troubleshooting functions.

Neutron mimics the physical implementation using virtualized subnets and ports for abstraction, it works similarly to the layer two segment, ports and subnets are assigned to network objects. The subnet abstraction layer consists a block of IP addresses, a gateway, a list of Domain Name Systems (DNS) and a list of host routes. This generates a large subnet address pool that Neutron works on top by assigning IPs to VMs. The port abstraction is a virtual point for each VM as the port receives an IP and Media Access Control (MAC) address from the pool of subnets, if needed users can request specific addresses from the subnet pool via an API call [19].

There are several types of agents in the Neutron API such as the plugin agent which is on the compute or network nodes and handles communication with the implementation, Dynamic Host Configuration Protocol (DHCP) agent which is used to assign IPs to VM instances via the API, Layer 3 agent (L3 agent) implements Layer Three (L3) functions

via Linux IP stack and ip tables allowing users to create routers that connect the Layer Two (L2) networks and Modular Layer 2 (ML2) agent that establishes a framework for Neutron plugins this agent will eventually replace the monolithic plugins associated with all the L2 agents [19].

OpenStack networking [19] uses three additional tools in its implementation which are network namespaces, Floating IP addresses and network security groups, the namespaces prevents the overlapping of subnets on the physical layer and tenants, this is implemented using Linux namespaces preventing duplicate IP addresses in the logical and physical networks used by the virtual machines. Floating IPs are public IPs that can be dynamically given to VM instances, these can also have a private IP addresses. In Neutron the L3 agent provides the Network Address Translator (NAT) functionality assigning floating IP addressed to the VM instances. Neutron implements the network security via Security Groups which allows users to specify the type of traffic and direction it can flow therefore in Neutron each port is associated to a security group and IP tables are used to implement the security groups.

2.3.3 OpenStack Image Service Glance

The Glance service[16] is central to IaaS. It receives requests for disk or server images and metadata definitions from users or Nova components. It also supports the storage of server and disk images in several repository including Cinder. OpenStack Image runs periodic processes supporting caching, also information consistency and availability are ensured through the replication services.

The OpenStack image service is composed by the following components: glance-api which is responsible for accepting image API calls for discovery and, retrieval and storage of images; glance-registry stores, processes and retrieves metadata about images, this metadata includes items information such as size and type of image, this registry is a private service as it is only meant to be used by the OpenStack Image service it should not be exposed to other users or services; the database is used to store image metadata, the database most used deployments are the MySQL and SQLite databases; Storage repository for image files there are several repository types available such has Object Storage, RADOS block devices, HyperText Transfer Protocol (HTTP) and Amazon S3 must be taken into consideration that some repositories are read-only usage; Metadata definition service it is a common API used by administrators, services and user to define custom made metadata, it can be used on different types of resources like images, artifacts, volumes, flavors and aggregates, the definitions include new property keys, description, constraints and types to be associated with [16].

2.3.4 OpenStack Identity Keystone

Keystone[16] is the central authentication, authorization and identity service mechanism for all OpenStack services components and users, it supports various forms of authentication this includes username and password credentials, token-based systems and public/private key pairs-style logins it also includes integration with existing directory services such as Lightweight Directory Access Protocol (LDAP).

Although keystone supports several authentication and authorization forms, it relies mainly on tokens so when an user interacts with a service he is required to possess a token as an instance of his identity and roles, the user can have an unscoped token however it will not be usable in a typical daily operation like creating a virtual machine or uploading one machine image. When the user requests a scoped token keystone checks for his assignments to see if he is allowed to access that domain or project. Tokens are structured as JavaScript Object Notation (JSON) objects and include parameter like: user identification (id,name), authentication method, expiration date, service catalog and project or domain to which the token is scoped [20].

The identity service is also a major actor in the authentication and authorization of workflow due to its catalog lists of all the services deployed by the cloud, it handles the authentication through endpoints, an endpoint is a network address where a service is listening for requests so each service, for instance Nova or Glance, has at least one assigned endpoint. OpenStack Identity uses tenants to group or isolate resources it then issues tokens to authenticated users after that the endpoints can validate the token before allowing user access, additionally the Keystone service responds to service requests, places messages in queue, grants access tokens and update the state database [19].

2.3.5 OpenStack Dashboard Horizon

Managing the OpenStack environment through a command-line interface gives the administrator complete control over the cloud environment, but having an usable Graphic User Interface (GUI) to manage the environments and instances makes the process easier. OpenStack Dashboard known as Horizon [21] provides this GUI, a web service that runs from an Apache installation, using Python's Web Service Gateway Interface and Django for a quick development web framework.

2.3.6 OpenStack Telemetry Ceilometer

The Ceilometer[22] project is responsible for monitoring and meter the OpenStack cloud for billing, benchmarking, scalability and statistical purposes. In its first builds Ceilometer would only meter basic parameters such as CPU, ram and storage usage and some events like image upload but the later builds of Ceilometer introduced triggering mechanisms and network monitoring algorithm which are more complex and time sensitive. As

the list of meters continuously grows within the project it is possible to use the data collected by Telemetry to more than billing, for instance it is possible to trigger the auto-scale feature in the Orchestration service [16]. The Telemetry Data Collection [16] service provides several functions such as: efficiently pools metering data related to OpenStack services; collection of events and metering data by monitoring notifications sent from services; publication of collect data to various targets including data stores and message queues.

2.3.7 OpenStack Object Storage Swift

OpenStack Swift[23] is a highly available, distributed and consistent object storage that can operate in standalone mode or integrated with the rest of the OpenStack cloud computing platform. It is used to store a great amount of data efficiently, safely and cheaply using a scalable redundant storage system. Swift manages data as objects opposed to the conventional file systems that manage data using file hierarchy and block storage, each object includes the data itself, a variable amount of metadata and a globally unique identifier. With its well defined RESTful API, users can upload or download objects to and from the Swift storage.

Inside Swift, objects are organized into containers similarly to directories in a file system except that the Swift containers cannot be nested. An user is associated with a Swift account and can have multiple containers associated with that account, in order to manage user accounts, user containers and objects inside a container, Swift uses an Account Server, a Container Server and Object Servers correspondingly if an user account requests for an object inside a container (either for uploading or downloading), the Account Server looks for the account first in its database and finds associated containers with the account it then checks for the container database to find whether the requested object exists in the specified container and finally the Object Server looks into the object databases to find retrieval information about the object, in order to retrieve an object the Proxy Server needs to know which of the Object Servers are storing the object, and what is the path of the object in the local file system of that server [23].

Once an object is stored in Swift, the access control is determined by Swift Access Control List (ACL). Swift has different levels of ACL, Account level ACL allows users to grant account-level access to other users and Container level ACL which is associated with containers, allows read actions, write actions or listing actions, if an user is authorized to do read actions on a container through read ACL, he or she can read or download objects from the container similarly, write ACL enables uploading an object into a container and listing ACL enables the listing of operations on the container. Swift ACL is limited in the following ways: once an object is set accessible to someone, he or she gets the full content of the object but there can be some sensitive information that the publisher wants to hide out; Swift ACL allows sharing an object with others, but it does not allow to share objects selectively at the content level [23].

2.3.8 OpenStack Block Storage Cinder

The Block Storage [24] service enables the management of virtualized block storage devices on several storage back-end systems, the cinder-scheduler daemon is responsible for scheduling virtual volumes to physical storage hosts.

The default scheduling algorithm in Cinder is called Available Capacity and it is based on a filtering/weighting procedure the scheduler works in three steps at first when a new volume request comes in the scheduler first filters out the hosts which have insufficient resources like capacity or too much workload to meet the needs of the request then the scheduler calculates a weight for each of the remaining hosts based on the available capacity and sorts them based on this weight in an increasing order, the lightest will be on top, the host which has the least weight is chosen as the best candidate to serve the new request [24].

2.4 Software Defined Networks

Computer networks as they are known are divided in three layers: data, control and management layers. The first one (data layer) consists of routers, switches, bridges, etc. these are responsible for the forwarding of data, the second dimension (control layer) represents the protocols that service the forwarding tables of the data plane elements and for the management layer, it includes the software services that are used to audit and configure the control functionality. The aforementioned structure is highly decentralized which was considered a very important characteristic of the Internet since it guaranteed network resilience, a crucial design goal, however this led to a very complex and static architecture and is the main reason why traditional computer networks are stern and complicated to manage and control this led to a vertically-integrated industry where is hard to innovate [25].

SDN came to be at the hands of the Open Network Foundation (ONF) which works to develop, standardize and commercialize SDN. The ONF defines SDN as it follows: *"Software-Defined Networking is an emerging network architecture where network control is decoupled from forwarding and is directly programmable"*[26]. In this context SDN uniqueness resides in its ability to provide programmability by decoupling the control and data planes since it supplies simpler network devices in the case of active networking. As such, network intelligence can be removed from the switching devices and placed on the network controllers making the switching devices externally controlled by software thus having no on-board intelligence. This decoupling of the control and data layers offers a greater freedom for external software, it enables the control and help define the behavior of the network [27].

The SDN abstraction layer allows the user to create multiple Virtual Tenant Network (VTN) on top of the physical infrastructure which are independent of the network protocols, each VTN has its own control plane instance enabling a specific Quality of Service

agreement[28, 29]. Virtualization simplifies the deployment of systems since there is no cabling or setting up of the Network Interface Controller (NIC), neither the need to place the system in a rack, the only thing needed to spawn a virtual system is to launch an instance from an image and to allocate the hardware resources, traditional networking has a “closed architecture which is proprietary and vertical integrated”[30], this lack of flexibility disables the ability to define high level network policies to the entire network and addi this to the frequent changes in the network conditions, it makes the control of networking traffic in data centers one of the most challenging tasks due to the enormous amount of servers, applications and networking devices which is what SDN excels at.

Amongst the available open-source projects available that are integrated with cloud networks we can find OpenDaylight (ODL), Open Network Operating System (ONOS) and Floodlight (FL) which will be described and studied in depth in the following subsection [19, 30].

2.4.1 OpenDaylight

OpenDaylight (ODL) is an open source software SDN controller coded in Java, embed with a modular, pluggable and flexible controller architecture. The ODL project was established under Linux Foundation with the objective to help the dissemination and adoption of SDN thus creating an underlying foundation for NFV. The software supports a bidirectional northbound REST-based API that originated from the Open Service Gateway Initiative (OSGi) Model, which provides a framework for the modularization of applications into small bundles for the northbound API. The southbound interface supports multiple control and data plane protocols, also plugins, these modules are dynamically linked into a Service Abstraction Layer (SAL) so the SAL is a representation of all the modules that are written northbound. These layers are responsible to fulfill the requests from services independently of the underlying protocol used between the controller and the devices[31, 32].

The ODL software depicts five logical architectural layers (Figure 2.1): the first is the app or user interface layer, it consists of the user interactions and any applications that are consumed via the northbound API of the controller, and as such the user interface is also implemented as an application which is available to other applications; as for the second layer we can find a REST-based northbound API based in the OSGi framework (as mentioned before); SAL is the third layer and also the core of the modular controller design, it supports multiple protocols on the southbound interface and also provides consistent services for northbound applications such as, the device discovery service which is used in the topology manager and to discover devices characteristics; the forth layer consists of the plugins that deal with the southbound protocols and services, SAL constructs the networks services using the features that these plugins enable which in-turn are based on the capabilities of a network device so based on the services running SAL chooses the appropriate Southbound protocol to interact with each network device,

the supported southbound plugins and protocols are: OpenFlow, OVSDB, Netconf, LISP, BGP, PCEP and SNMP; as for the fifth and last layer it is the physical and virtual network devices layer, it interacts with the plugins and southbound protocols, it can be plug-in enabled (OpenFlow-enabled) or can be supported via a southbound protocol in addition to the logical architecture, ODL also supports multi-tenancy which allows the networks to be split for tenants and have dedicated controllers and module functions [19].

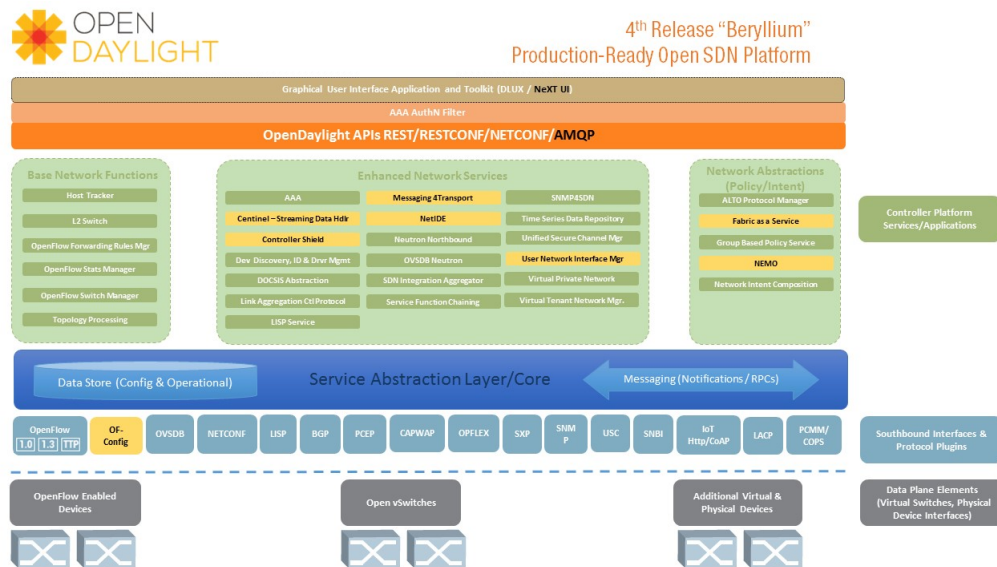


Figure 2.1: OpenDaylight Architecture [32].

2.4.2 FloodLight

Floodlight (FL) is more than an OpenFlow controller is also a collection of applications as we can see in Figure 2.2 that are built on top of the controller. This controller is able to manage and control an OpenFlow network, while the applications that run embed on top of the controller are able to solve the needs over the network. The Java module applications run from origin on the controller and each app is mapped in the REST API enabling application calls, information retrieving and service invocation despite the languages or protocols inherent to the calling application [33].

As the majority of the SDN controllers FL was designed as an extremely concurrent system, that can achieve high throughput since it has a multi-threaded design that explores the parallelism of multi-core computer architectures, it also has its own northbound API that abstracts the inner details and behavior of the control and data planes [25].

In a data center context a FL controller provides a comprehensive view of the VM activities, it is capable of listening to OpenFlow-based Transmission Control Protocol (TCP) connections from each of its switches and then it is able to manage the traffic so it can flow through it. When the devices communicate across the network the controller

software can estimate the best possible route and according to the a load factor and network traffic it can create a flow that leads the packets from the source to the destination, so in this way FL offers a high level of granularity in network management and also a central view of the network traffic. These features enable the network to be flexible thus enhance its performance [30].

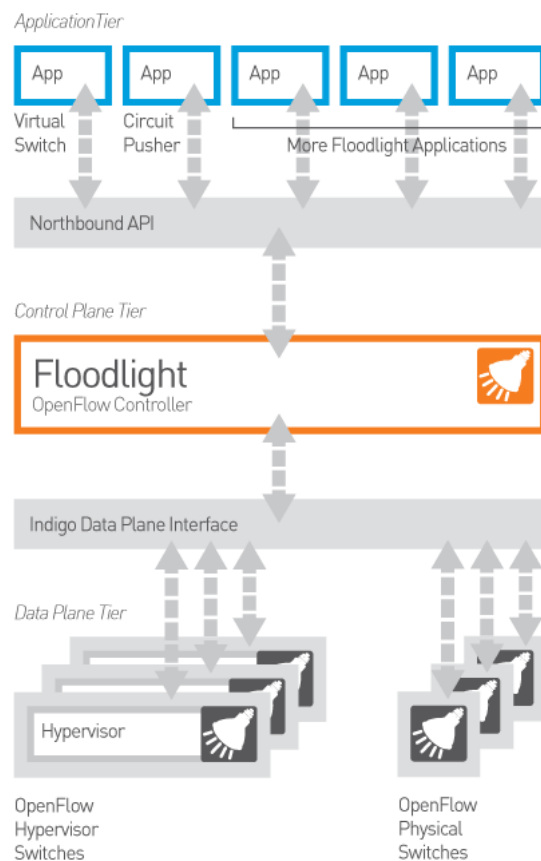


Figure 2.2: FloodLight Architecture [33].

2.4.3 Open Network Operating System (ONOS)

Open Network Operating System is an open source SDN network operating system (Figure 2.3) that features the high availability, scale-out and performance networks demand. ONOS uses northbound abstraction and APIs that enable the user to develop applications and a southbound abstraction and interfaces to allow the control of OpenFlow-ready and legacy devices, so in short ONOS enables: carrier grade features to the control plane; Web style agility; the migration of existing networks to white boxes; and the lowering of service provider CapEx and OpEx [34].

ONOS is more than a SDN Controller since SDN controllers fundamentally relay OpenFlow messages directly to apps and vice versa, this way it is safe to say that they act more like device drivers this way they are not designed to have critical scalability,

availability and performance features that complete the SDN control platform. ONOS differs from a SDN controller because it is responsible for the following features: it is able to manage a finite amount of resources on behalf of the consumer, and doing so, it ensures that none of the consumers aren't starved and are handled fairly; ONOS isolates and protects network operating system users from each other since each user only has access to a set of resources, it also multiplexes between multiple apps and devices, ONOS also virtualizes some or all the resources in order to allow consumers to have their own virtual instantiation of the OS; it provides useful abstraction to enable users to consume services and resources that are managed by the OS without having to understand all of the underlying complexity; the system provides security from the external world to the users OS; and finally it supplies useful services that remove the need for rebuilding the same services. In short controllers are too limited in scope as they may not provide the necessary services or protection when compared with ONOS [34].

The ONO System architecture[34] foundations are High Availability, Scale-out Capabilities and Performance and this is achieved with the following features:

- **Distributed Core** it provides scalability, high availability and performance thus, bringing carrier grade features to the SDN control plane. ONOS can run as a cluster which brings a huge amount of agility to the SDN control plane;
- **Northbound abstraction/APIs** that include network graph and application intents to ease the development of control, management and configuration services, adding once again agility to the control plane;
- **Southbound abstraction/APIs** that enable pluggable southbound protocols that enable the control of both OpenFlow and Legacy devices, it isolates ONOS core from the total of different devices and protocols. This feature is the main enabler of the migration from legacy devices to OpenFlow-based white boxes;
- **Software Modularity** makes it easy to develop, debug, maintain and upgrade ONOS as a software system, by the community and providers.

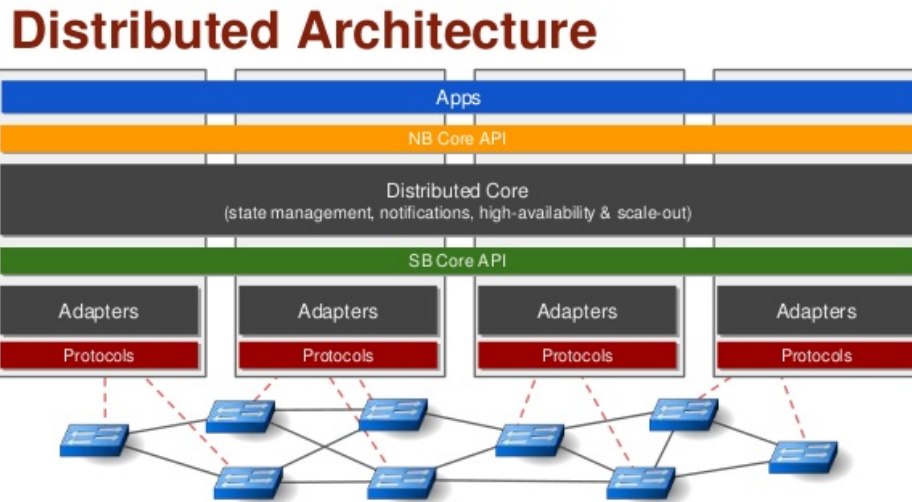


Figure 2.3: ONOS Architecture [35].

2.5 SDN Integration with OpenStack

Openstack Neutron is the focal point of this integration, due to its pluggable architecture and to the controller nodes receiving the REST API requests via the networking plugins. The Core Neutron plugins more specifically the ML2 plugin has a set of mechanism drivers that are compatible with most the SDN controllers enabling this way the integration into this components, also most SDN controllers and systems have Northbound APIs that map directly into the Neutron API thus allowing the ML2 plugins to communicate with SDN Controller-Neutron services this way the plugin acts like a proxy and sends all the requests to the SDN controller. This way it is possible to orchestrate virtual and physical networks and allow multiple implementations of Neutron services communicate with the controller, it is then obviously simplifies the network and reduces the load on the networking service since it moves the orchestration intelligence and complexity to the controller functions [19, 36].

CHAPTER



ARCHITECTURE

As previously discussed in Chapter 2, the cloud computing concept has been evolving over the years, mainly fueled by the IT industry necessities for a better network access to a shared pool of computing resources, with the capacity to be rapidly allocated and leased with minimal management.

With this in mind, the cloud computing system (OpenStack) described in this chapter aims to provide the core features of a cloud computing architecture, On-demand self-service that allows the user to lease computing resources automatically, broad network access which enables the access of services over the Internet independently of the client platform, rapid elasticity that enables the physical resources to scale inwards or outwards on demand, these are just some of the prime characteristics of this system.

This architecture was implemented using the OpenStack software which is one of the quickest evolving open source platform that implements the cloud computing paradigm, being deployed globally it has been tried and tested amongst all the giants of the telecommunication world. This platform provides the necessary scalability, redundancy and durability required in our environment.

OpenStack, as the name suggests, is an open source (hence Open) cloud system (Stack) that controls computing resources such as compute, storage and network assets with the help of a dashboard which enables the user to provision resources through a web interface, this software aims to implement a simple modular system that provides mass scalability.

3.1 OpenStack Logical Architecture Overview

Figure 3.1 depicts an overview of the services and relations they construct in the logical architecture.

In the project foundations we can find six core modules plus the GUI module so, in a

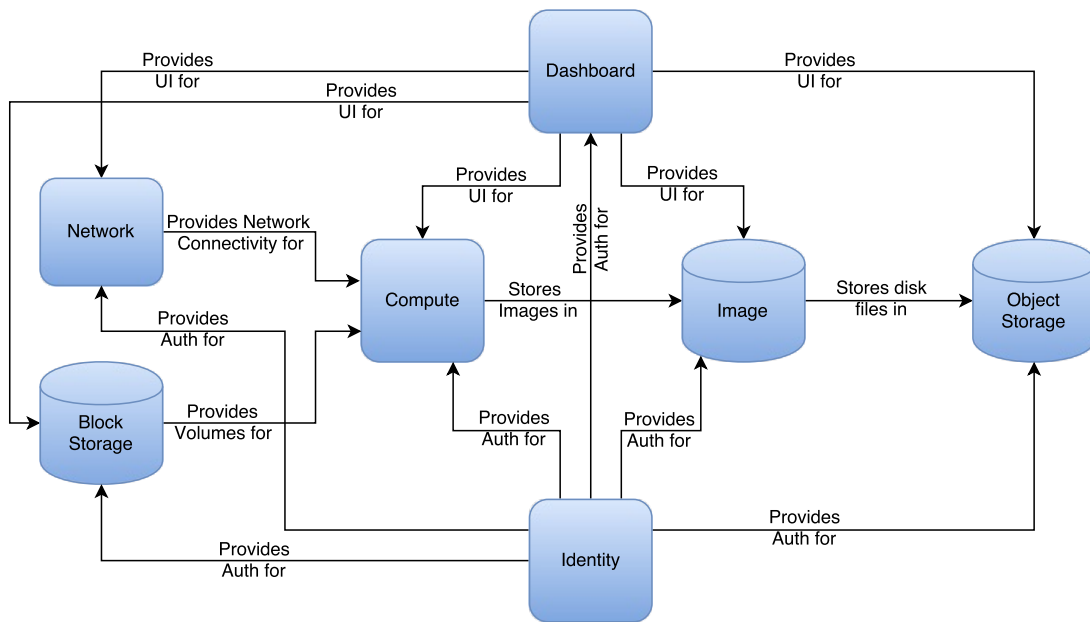


Figure 3.1: OpenStack Logical Architecture Overview [16].

way, each service plays its role in the coordination of the whole system, in order to provide the capabilities specified above (Figure 3.1). As for the services themselves - Object Storage (Swift), Identity (Keystone), Compute (Nova), Networking (Neutron), Block Storage (Cinder) and the Image Service (Glance) -, they are the core services implemented with the addition of the Dashboard (Horizon) service. There is also a total of twelve optional services that complement an array of tools to the architecture, such as the Telemetry service (Ceilometer) or the Orchestration service (Heat), where the first enables the metering of the system for benchmarking and scalability and the second makes it possible to orchestrate multiple cloud applications using the native HOT templates or Amazon Web Services (AWS) CloudFormation templates.

The current architecture runs only the six services as they were mandatory, plus the dashboard service as it is a quality of life service, since it enables an easier understanding and operability of the system.

As mentioned and illustrated in 3.1, the seven services and descriptions are the following:

Table 3.1: OpenStack Services.

Service	Project Name	Description
General Purpose		
Dashboard	Horizon	Provides a web-based interface that enables the interaction in between the user and the OpenStack services.
Compute	Nova	It is responsible for spawning and decommission of virtual machines, henceforth it controls the life-cycle of VMs.
Networking	Neutron	This service manages the network and enable connectivity for other OpenStack services. It also supports a symbiosis with other network technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured, it is highly fault tolerant due to data replication across several drivers.
Block Storage	Cinder	Enables persistent block storage for instances, has a pluggable driver architecture for creation and management of storage devices.
Shared Services		
Identity Service	Keystone	Responsible for the authentication and authorization of the other OpenStack Services.
Image Service	Glance	It manages the VM disk images, this service is used by compute during instance provisioning.

To further the understanding of the logical architecture, the following sections will explain the core functionalities and workflow of the service modules described above.

3.1.1 OpenStack Compute Overview

As OpenStack itself the Compute service is also modular. Therefore, it is divided in several services and daemons that cause OpenStack Compute to play a major role in an IaaS system. This module consists of the components in Table 3.2.

Table 3.2: Compute Service Component Overview.

Component Name	Description
Services	
nova-api	The service consists of the OpenStack Compute API, the Amazon EC2 API and the admin API. These enable the management of the cloud. Its main role is to accept requests from instances, to enforce policies and deal with orchestration activities.

nova-compute	It's a worker daemon that is responsible for the creation and termination of instances using the hypervisor APIs. (ie. libvirt for KVM and QEMU).
nova-scheduler	Responsible for taking instance requests from the queue and assigning them to compute server hosts.
Modules	
nova-conductor	Responsible for the mediation of the interactions between the nova-compute service and the database.
nova-cert	It is a server daemon that is used to generate certificates for the EC2 API.
Daemons	
nova-console	Worker daemon responsible for the creation and termination of VM.
nova-novncproxy	Enables proxy access to running instances via Virtual Network Computing (VNC) connection. Supports browser-based novnc clients.
nova-spicehtml5proxy	Enables proxy access to running instance via Simulated Program with Integrated Circuits Emphasis (SPICE) connection. Supports browser-based HTML5 client.
nova-xvncproxy	Enables proxy access to running instance via a VNC connection. Supports the OpenStack-specific java client.
Backend	
Message Queue	The central exchange point of messages between daemons and services.
Database	Responsible for the storage of build-time and run-time states of the cloud such as, Available instance types, Instances in use, Available networks and Projects.

The aforementioned components fall into four categories that fully describe the service architecture.

The **API Server** is the heart of the cloud framework this API enables storage, network control, management of the hypervisors. The API endpoints of the server are HTTP web services responsible for authentication, authorization and control functions using the API interfaces models. It is compatible with multiple existing tool sets which prevent vendor lock in.

Message Queue is responsible for managing the interactions between the control nodes, network nodes, API endpoints, schedulers and other similar components, these communication is done by HTTP requests in between the API endpoints. The messaging events begin with the API server receiving a request from an user, and after it has been authenticated and certified, if the object is available the message is sent to the queuing engine so it can be picked up by the worker services (Daemons). This types of services constantly listen to the queue for messages for their type. The worker then takes the message from the queue and executes it. When the task is finished it sends a reply to the queue, being then received by the API server and finally relayed to the user. During this

process information is added, removed and searched for into the database as necessary.

Compute Workers are responsible for the lifecycle of the computing instances on the host machines - they can run, delete and reboot instances; they can also attach or detach storage volumes to an instance and show the console output of that instance.

Finally, there is the **Network Controller** which is responsible for the management of network resources. It receives commands through the message queue from the API server and it is able to allocate IP addresses, configure Virtual Local Area Network (VLAN) for projects and configure networks for compute nodes.

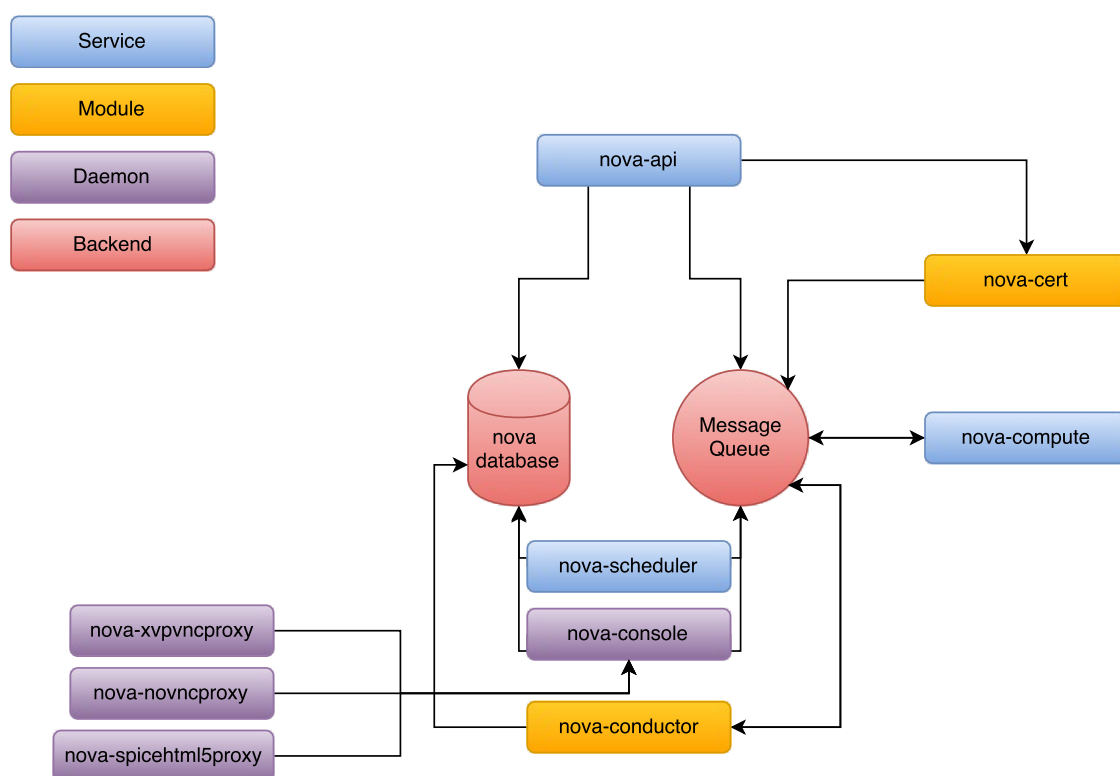


Figure 3.2: OpenStack Compute Architecture.

As we can see in Figure 3.2 and from the explanation made above, the Compute service makes use of the nova-scheduler service to choose the best fitting host to deploy an instance, this service (nova-scheduler) interacts with other components through the message queue or the database repository to schedule the tasks. A total of two services can process tasks from the queue, nova-compute and nova-conductor, and after they have terminated the task they proceed to send a message into the queue that is picked up by the scheduler and sent to the nova-api.

3.1.2 OpenStack Network Overview

This networking service is another modular component of the architecture, as it is composed by several components depicted in the following Table 3.3.

Table 3.3: Network Service Component Overview.

Component Name	Description
Services	
neutron-api	Responsible for accepting and routing API requests to the appropriate OpenStack Networking plug-ins for action.
SDN server	Provides additional networking services to tenant networks.
Agents	
plug-in agent	It runs on the hypervisor and is responsible for the configurations of the vSwitches.
dhcp agent	Enables DHCP services for the tenant networks.
l3 agent	Provides L3/NAT forwarding to external network access of VMs on tenant networks.
metering agent	Observes the L3 traffic of the tenant networks.
Backend	
Message Queue	Used to route information in between the neutron-server and the various agents also acting as a database to store networking states.

OpenStack networking is responsible for the Virtual Network Interface (VNI) and the access layer of the Physical Network Interface (PNI). This service makes it possible for tenants to create virtual networks topologies with embedded services like firewalls, load balancers and virtual private networks. It works as an abstraction layer for networks this way enabling sub networking and router creation thus mimicking all the functionalities of the physical counterpart.

The network service always creates an external network which is not just a virtual network but a slice of a physical external network; this way the services can be accessed by external users. In addition to external networks, neutron also has one or more internal networks; these are software defined networks that connect directly to the VMs.

In order for VMs to access outside networks and vice versa, routers have to be placed in between the networks and each router needs to have one gateway connected to the “Internet” and at least one interface connected to the internal network.

It is also possible to allocate IP addresses of external networks to a port on the internal network. These are called public IPs, and associate the external IP to a VMs port. This allows external entities to access the VMs.

Figure 3.3 depicts how the Neutron components communicate with each other. This communication is mainly done using remote process calls in between the native agents of the system, it uses REST communication when using the SDN service and SQL for all the accesses to the database.

As it is possible to observe, the neutron-server receives requests and passes them to the message queue which is being constantly pooled by the L3, Plugin and DHCP agents for tasks to be completed; when these tasks are finished, the agents send a result message

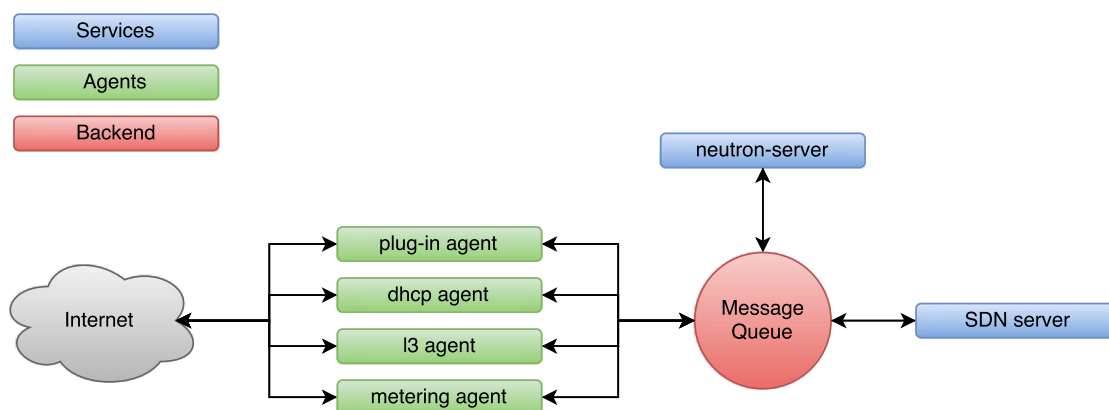


Figure 3.3: OpenStack Network Architecture.

into the message queue that is then picked up by the neutron-server and relayed to the requester, this usually being the compute service or the GUI.

Both the SDN service and the plugin agent work as a proxy for other SDN controllers that the user might want to implement in order to enhance the control and management of the network. This integration will be approached in Chapter 4.

3.1.3 OpenStack Identity Overview

OpenStack Keystone is the single point of integration for authentication, authorization and service cataloging, it is used by other services as a common unified API, so due to this, Keystone is the only service that interacts with all the other services.

When a OpenStack service receives a request from an user the service checks with Keystone if this user has a valid token in case of a positive result the service then proceeds to fulfill the request.

In the core of this service we find three types of components:

- Server, a centralized server that as the ability to authenticate and provide authorization using a RESTful interface;
- Drivers or service backend - which are integrated with the centralized server - enable the access to information that is stored in repositories external to OpenStack such as SQL databses or LDAP servers;
- Modules, whose main function is to intercept service requests, extract user credentials and transmit them to the centralized server for authorization.

In order to help understanding the functionality of Keystone, the following Table 3.4 describes the meaning of the most used concepts in this service.

Table 3.4: Keystone Service Concepts.

Concept	Description
Authentication	The process of confirming the identity of an user.
Credentials	Data that confirms the identity of an user. Can be the name and password or an authentication token.
Domain	It is a collection of projects and users that define administrative boundaries for managing entities.
Groups	Collection of users that are owned by a domain.
Project	A container that groups or isolates resources and identity objects.
Role	Personality with a defined set of user rights and privileges enabling them to perform a specific set of operations.
Token	An alpha-numeric string that enables access to the OpenStack API and resources.
User	Digital representation of a person, system or service that needs validation by the identity service.

The highest separation entity is the Domain. In each domain there are Roles, Groups and Projects and inside each group we can find one or several users which can have a different role in the same group and also have access to the same or different projects. Associated with each user there are credentials that can be a Token or an username and password of that user. These relations are described in the following figure (Figure 3.4).

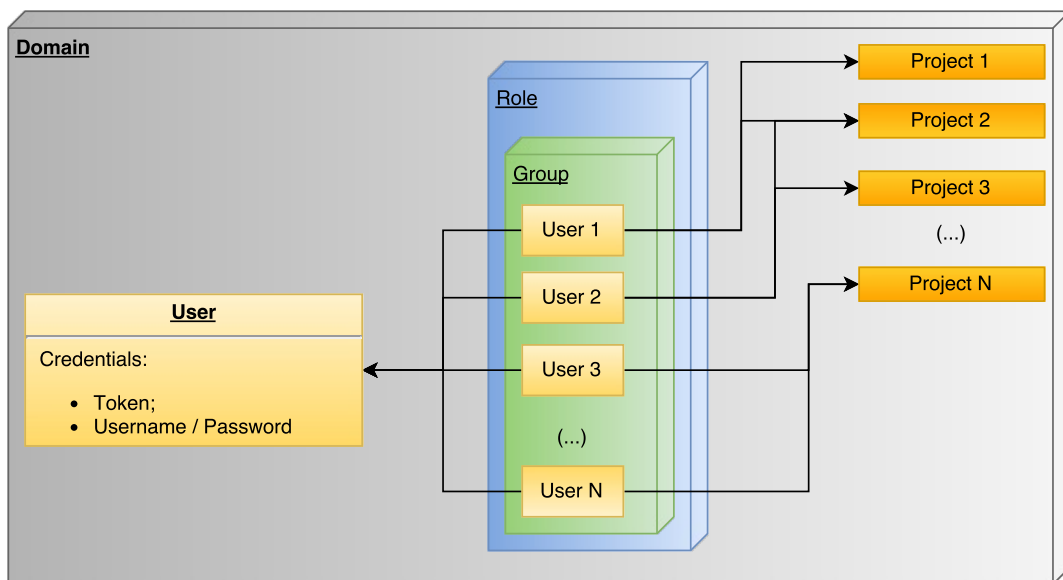


Figure 3.4: OpenStack Identity Concept Relation.

As it was seen, keystone cornerstone concepts for authentication and authorization are based from the common credential system, they present the credentials to the server which enables them to access the information of a specific group/groups.

In order for an user to authenticate itself the service needs to validate its credentials,

the first time the user makes a request its credentials consist of their username and password, the credentials are validated and henceforth it is possible to issue a token. From this point on the user can present the token whenever authentication is necessary. It is important to underline that this token has a finite duration and can be revoked at any time.

To contextualize how Keystone controls access we can take the cloud administrator role as an example, this user is able to list all the instances running in the cloud, also change any instance characteristics and resources allocated to them whereas a normal user can only check the instances in their current group, the resource quotas, cores being used, disk space and all other resources associated with the project it belongs to.

3.1.4 OpenStack Object Storage Overview

Object Storage makes it possible for users and applications to store and retrieve objects through a REST API. These objects, which are agglomerates of data, are stored hierarchically and offer three types of access: anonymous read-only , ACL defined and temporary.

Swift is composed of components in the Table 3.5 and its organization is as displayed in Figure 3.5.

Table 3.5: Object Storage Service Component Overview.

Component Name	Description
Servers	
swift-proxy-server	It is responsible for dealing with HTTP requests such as file upload, metadata modification and container creation.
swift-account-server	Controls the accounts used in Object Storage.
swift-container-server	Maps the containers or folder that the storage nodes hold.
swift-object-server	Manages files in the storage nodes.
Services	
swift client	Enables the use of the REST API via command-line client as long as the user is authenticated.
swift-recon	A client tool that retrieves metric and telemetry informations about the cluster being used.
Backend	
Account Database	Stores accounts information.
Container Database	Stores containers information.
Object Database	Stores object information.

The robustness of Swift comes from the simple principle that all hardware fails at some point or level, so in order to avoid data loss it is important to understand that the major factor that influences data availability and durability is its placement and not the hardware's reliability. This service takes three as the baseline value of replicas. This means that an object is only marked as written when at least two other copies exist. As increasing this number would lead to an augment in the robustness of the file system the

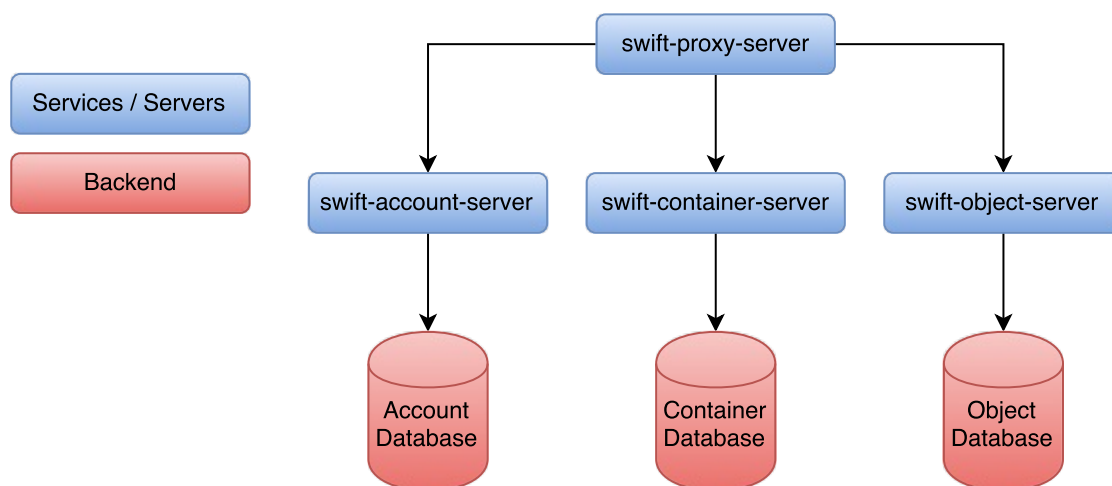


Figure 3.5: OpenStack Object Storage Architecture.

available space would diminish and verbosity in the network would scale exponentially as trade back.

The main characteristics of the Swift Service sums up to:

- All objects stored have a respective URL;
- All the files are replicated three times in separated areas these can be drives, nodes, racks, etc;
- Each object has its own metadata;
- The interactions with the object storage are via the RESTful HTTP API (Get / Put / Delete);
- Objects are scattered anywhere in the cluster;
- The cluster grows by adding more nodes without sacrificing performance thus allowing a cost-effective storage expansion;
- Data and nodes can be added to a cluster without downtime;
- Failed nodes can be replaced without downtime.

One drawback that is also a virtue of this service is its verbosity. Since the amount of replicas must always be three, the service easily floods the network with sync messages. Due to this, the main points of network congestion are: amongst object, container and account servers; between the aforementioned servers and the proxy server; and between the proxy server and the users.

Taking the total failure of a large data server (around 50Tb of data) as an example, all the information needs to be transferred to another server in order to keep the three replica

rule alive. This will be a huge strain on the network alone not taking into consideration that there is also traffic from other services flowing through the network.

Another scenario of high congestion is when a file is being uploaded into the Object Storage. It must be taken into consideration that the proxy server is obliged to allocate bandwidth accordingly to the amount of replicas, so a 10 Gbps stream actually translates into a 30 Gbps stream of data in the network.

3.1.5 OpenStack Block Storage Overview

Block Storage (or Volume Storage) allows the user to attach block-storage to virtual machines; these volumes are persistent as they can be removed from one instance and attached to another while keeping the data in them intact - just like a pen drive or an external disk.

Cinder enables several solutions to hardware fault such as: if an instance needs to be relaunched or crashes it can be without the loss of information since the user can simply detach the volume and re-attach it once the instance comes online; also, a cinder volume can be used as a boot disk for an instance this way removing the need for an ephemeral storage disk.

As all other services, Cinder has a set of components which are depicted in Table 3.6. The overall architecture can be seen in Figure 3.6.

Table 3.6: Block Storage Service Component Overview.

Component Name	Description
Services	
cinder-api	Manages the API requests and routes them to cinder-volume to be dealt with.
cinder-volume	Main liaison in between the cinder daemons, it is responsible for read and write requests.
Daemons	
cinder-scheduler daemon	Chooses the best storage node on which to create the volume.
cinder-backup daemon	It creates backups of volumes on the backup storage nodes.
Backend	
Message Queue	Routes information between Block Storage processes.
Database	Used to store block information.

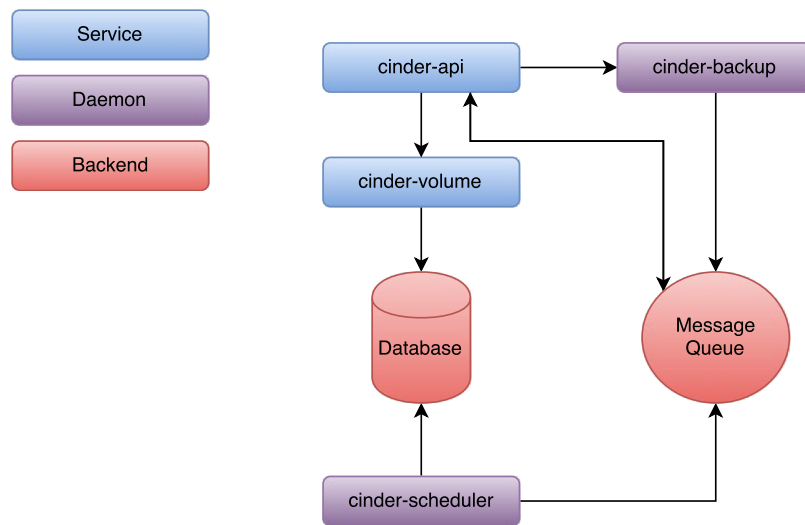


Figure 3.6: OpenStack Volume Storage Architecture.

3.1.6 OpenStack Image Overview

The Image service is responsible for managing the requests for server images and meta-data definitions by users or the Compute components. This service has two core parts: the glance-api and glance-registry, where the first is responsible for providing images and the latter maintains the metadata information associated with the VMs images, this component requires a database.

The overall architecture layout of this service can be seen in the following figure (Figure 3.7).

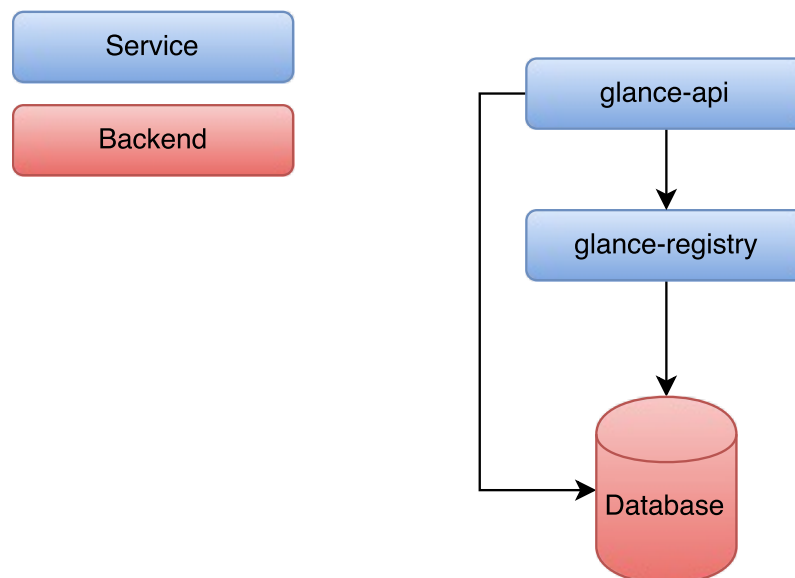


Figure 3.7: OpenStack Image Architecture.

3.2 Openstack Physical Architecture Overview

The OpenStack Dev guide [16] suggests the layout displayed in Figure 3.9 as the basic node deployment. It's a simple architecture built upon for compute and it has a single cloud controller along with several compute nodes. Since this is the deployment used in standard data centers it is clear that implementing such an architecture is inconceivable due to network and hardware constraints. The architecture implemented features its layout in Figure 3.8. This implementation tries to follow the guidelines while supporting our hardware and network layout. The following table (3.7) shows the components that are part of the architecture.

Table 3.7: OpenStack Components [16].

Component	Details
Openstack release	Neutron
Host operating system	Ubuntu 16.06
OpenStack package repository	Ubuntu Cloud Archive
Hypervisor	KVM or QEMU
Database	MySQL
Message queue	RabbitMQ
Networking service	Nova
Network manager	FlatDHCP
Image Service backend	file
Identity Service driver	SQL
Block Storage Service backend	LVM or iCSI
Object Storage	OpenStack Object Storage

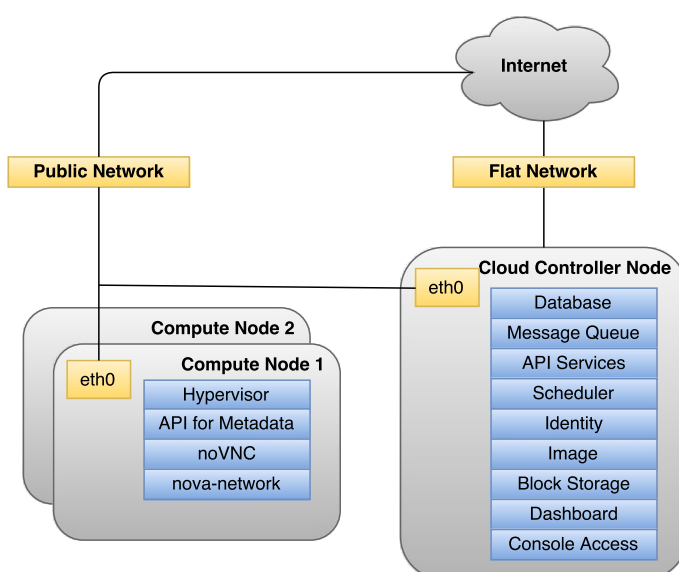


Figure 3.8: Implemented basic node deployment.

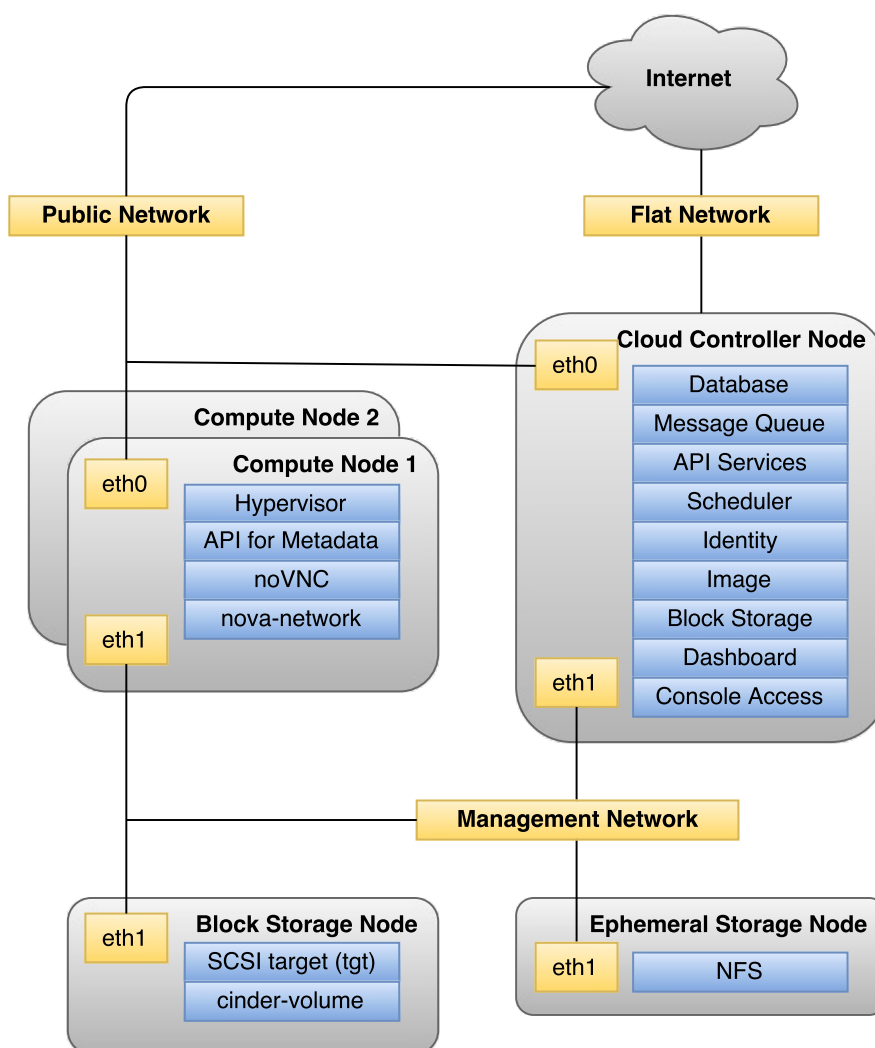


Figure 3.9: OpenStack suggested basic node deployment. [16]

As it is possible to see, the cloud controller runs the following services: dashboard, API services for Neutron and Nova, the database (MySQL), the message queue (RabbitMQ), the scheduler that selects the free compute resources (nova-scheduler), identity services (Keystone and nova-consoleauth), Image services (Glance API and glance-registry) and the Block Storage services (Cinder API and cinder-scheduler). As for the compute node this holds the computing resources so it runs the hypervisor, libvirt which is the driver for the hypervisor that enables live migration from node to node, nova-compute and Nova API that are used for multi-host mode as it retrieves instance-specific metadata, nova-network and nova-vncproxy.

The network consists of two switches: one for the management network that deals with block storage due to high amount of traffic (being a private one) and the public one that covers the public access, so in the suggested architecture the nodes have two network interface controllers, but since in the implemented architecture the dedicated storage

nodes were removed there is no need for this network to exist.

The implemented system follows the structure displayed in Figure 3.10, the cloud has three HP servers of which one is the controller and the other two are running as compute nodes. The cloud is integrated in the laboratory network 172.16.4.0/23, the available IPs are from 172.16.4.1 to 172.16.7.255. The physical machines are allocated to 172.16.4.50, 172.16.4.51 and 172.16.4.203 and our virtual instances are being deployed on the 172.16.7.0/24 subnet with the first IP 172.16.7.1 being the IP and 172.16.7.255 the broadcast.

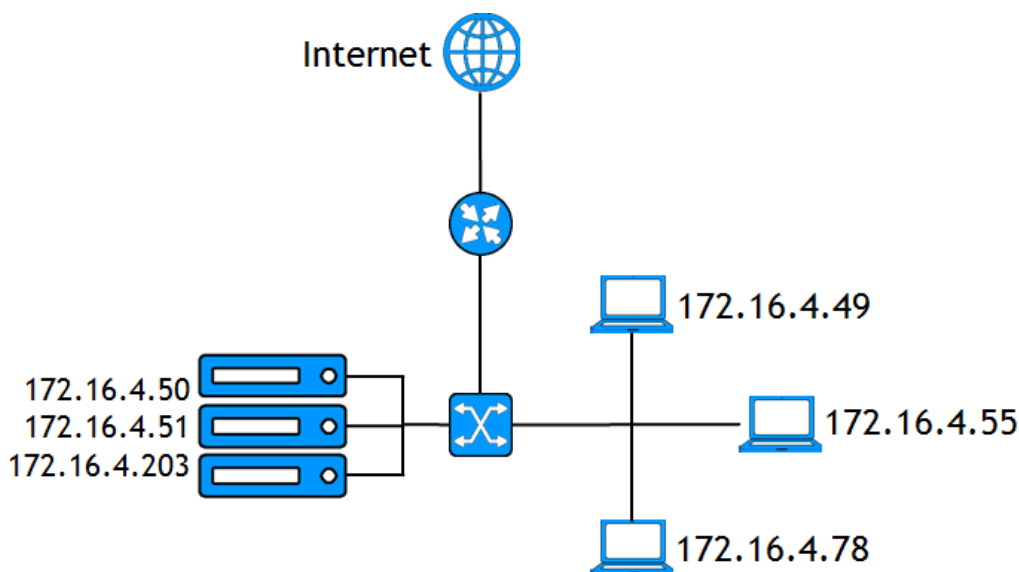


Figure 3.10: Cloud Network Layout.

In Figure 3.11 we can see how the instances communicate with the outside and with each other, there is a virtual bridge (in this case br100) that connects each instance to the compute node network interface all the traffic made by the instance will be forwarded there and then to the outside world. Although the packets are leaving via the eth0 if they are coming from an instance they will have the instance MAC address. The same happens when instances try to communicate with other instances. In the case of the instances being inside the same controller the traffic doesn't reach the eth0 network card as the communication is done via the br100 interface. When the instances try to communicate with other instances outside that compute node the process is the same as communicating with the outside world.

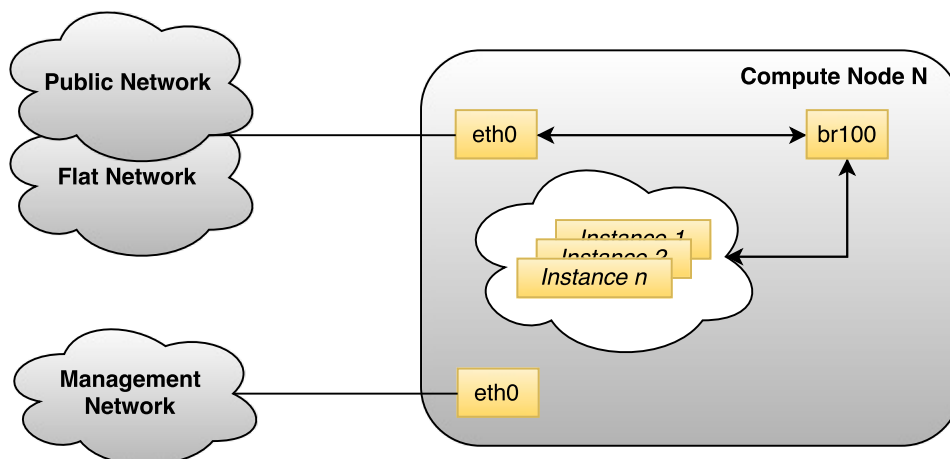


Figure 3.11: Network Traffic [20].

3.2.1 Common Structures / Services

3.2.1.1 Database

As most of the OpenStack Services need a database to store and retrieve information, states and configurations, this architecture uses several SQL databases. Some are shared amongst all the services; as for the others, those are specific to that service alone. The database technology used is MySQL as it is the most tested database for use with OpenStack and its heavily documented.

3.2.1.2 Message Queue

Most services need to communicate amongst them and also have components inside them that communicate with each other and so all the communication is done via the message queue. The message queue used was RabbitMQ as it is extremely easy to use and test, also it can be natively clustered in order to avoid the message queue to grind to a halt which would make it enter a read-only state and this would leave the information stuck.

3.2.1.3 Networking Manager

Although there are several options for managing the network FlatDHCP was chosen in Multi-Host networking mode in order to provide high availability, there is a nova-network daemon running per Compute node this provides a robust mechanism that ensures that the network interruptions are isolated to each compute host.

3.2.1.4 Identity Service backend

A SQL backend was chosen for the Identity Service since this backend is simple to install and robust also, there is the need for many installations to bind with existing directory

services and this backend provides options to do so.

3.2.1.5 Image Service backend

Since this deployment only uses the Object Storage to store virtual machines images as it can be easily added as a backend due to being native to OpenStack.

3.2.2 Hardware Specifications

When it comes to the hardware necessary, as it was mentioned before, the specifications scale with the amount of the load the system has to deal with. As for the suggested hardware the OpenStack Dev guide has the following specifications (Table 3.8).

Table 3.8: OpenStack Hardware Specifications. [16]

Node Type	Hardware Example
Controller	Model: Dell R620 CPU: 2 x Intel Xeon CPU E5-2660 0 @ 2.00GHz Memory: 32 GB Disk: 2 x 300 GB 10000 RPM SAS Disks Network: 2 x 10G network ports
Compute	Model: Dell R620 CPU: 2 x Intel Xeon CPU E5-2650 0 @ 2.00GHz Memory: 128 GB Disk: 4 x 300 GB 10000 RPM SAS Disks Network: 2 x 10G network ports
Storage	Model: Dell R720xd CPU: 2 x Intel Xeon CPU E5-2620 0 @ 2.00GHz Memory: 64 GB Disk: 2 x 500 GB 7200 RPM SAS Disks + 24 x 600 GB 10000 RPM SAS Disks Raid Controller: PERC H710P integrated RAID controller, 1 GB NV Cache Network: 2 x 10G network ports
Network	Model: Dell R620 CPU: 1 x Intel Xeon CPU E5-2620 0 @ 2.00GHz Memory: 32 GB Disk: 2 x 300 GB 10000 RPM SAS Disks Network: 5 x 10G network ports

The hardware specifications of the implemented hardware is described in Table 3.9.

Table 3.9: Used Hardware Specifications.

Node Type	Hardware
Controller	Model: HP Proliant DL160 Gen 9 CPU: Intel Xeon E5-2609 v3 Memory: 32 GB Disk: 2x 300Gb HP H240 Network: 2x 1Gb network cards
Compute 1	Model: HP ProLiant DL260p Gen 8 CPU: 2 x Intel Xeon CPU E5-2620 v2 @ 2.10GHz Memory: 16 GB Disk: 500 GB 5000 RPM SATA Disk Network: 4x 1Gb network cards
Compute 2	Model: HP Proliant DL160 Gen 9 CPU: Intel Xeon E5-2609 v3 Memory: 32 GB Disk: 2x 300Gb HP H240 Network: 2x 1Gb network cards

SDN ARCHITECTURE

With continuous expansion in mind, the cloud development will reach a breakpoint where network problems such as network congestion and resource fragmentation as well higher response times will surface, the integration of SDN into the established architecture is a solution that can bridge the gaps formed by these problems [37, 38]. In chapter 2 three platforms were approached FloodLight, ONOS and OpenDaylight all three are state of the art platforms that can integrate OpenStack networking services in order to enhance them, so with this in mind in this chapter it will be shown how these platforms integrate the architecture described in the aforementioned chapter 3 as well as a comparative analysis of the three, in order to choose the one performing better in our environment.

4.1 Logical SDN Architecture

The integration of a SDN controller in the Logical Architecture has little to no impact in its core design, in addition to all the services displayed in Figure 3.1 the integration of a network controller leads to a cooperation in between this element, the SDN controller, and the Neutron service so instead of only one service providing the networking capabilities to compute nodes we have now two elements that do so, as we can see in Figure 4.1.

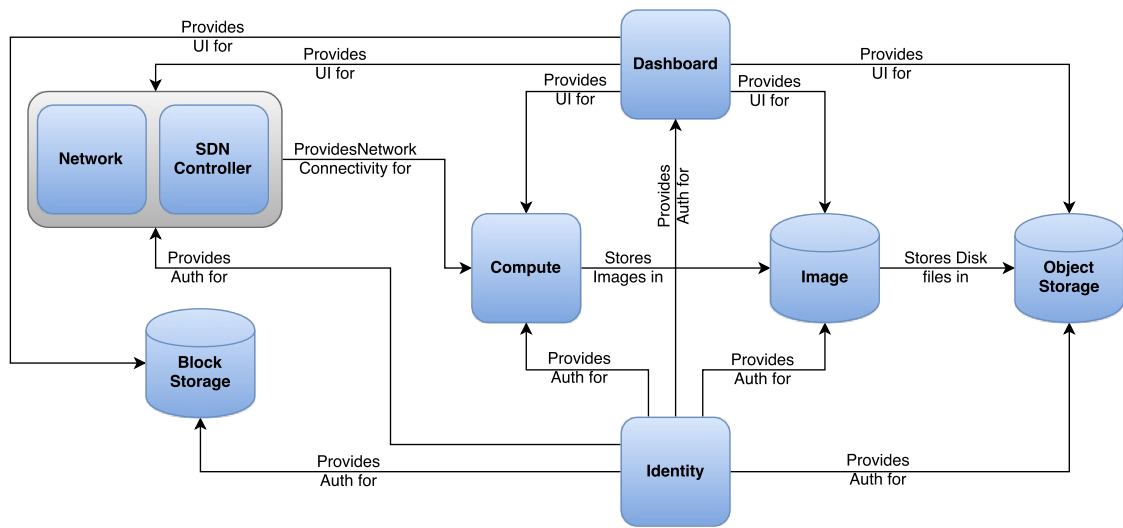


Figure 4.1: OpenStack Logical SDN Architecture.

In this scenario Neutron is responsible for passing the network management operations onto the controller using the controller Northbound API 4.2, which processes the information and enables a more accurate management of the data plane layer, the controller also manages the hardware via the Southbound API using OpenFlow. The controller integration is done by using plug-ins that enable a centralized network management, some of these plug-ins are native to OpenStack Neutron such as the Linux Bridge and Modular Layer 2 agent other are SDN plug-ins like OpenVSwitch. The use of the integration bridge (br-int) that connect the VMs in between them using L2 tunneling while the External Bridge (ext-br) is responsible for the external network, this bridge will connects to the Southbound API to manage the physical network.[30, 39]

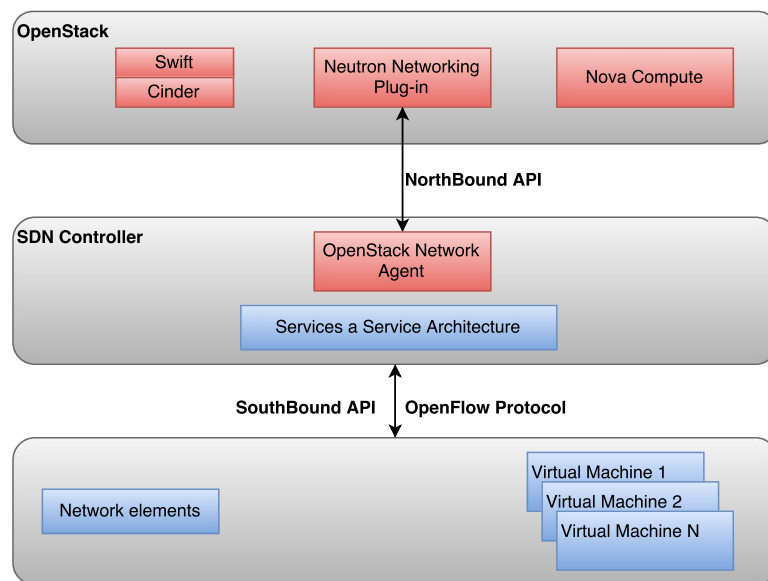


Figure 4.2: SDN Controller Integration into OpenStack [39].

Although all the three platforms have the same objective and stand at the same logical place in the architecture they communicate using different tools and protocols. [19, 37]

OpenDaylight controller (Figure 4.3) communicates with OpenStack Neutron via the modular layer-2 plug-in, the ODL project has a Neutron API service, which makes its Northbound API map directly into Neutron API this, allows the ODL plug-in to communicate with ODL Neutron services making the modular layer-2 plug-in act like a proxy that forwards the requests to ODL. This way ODL can manage virtual and physical networks and also moves several implementations of the network services in ODL. This integration simplifies the network service as it removes the complexity of the service onto the SDN controller [19, 32, 38].

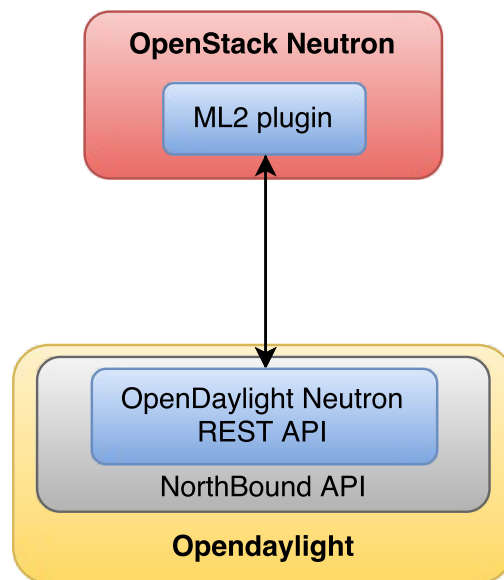


Figure 4.3: OpenDaylight to OpenStack communication.

The **FloodLight controller** (Figure 4.4) has two main components the VirtualNetworkFilter which maps the Neutron API, since Neutron exposes a network-as-a-Service model which implements a REST API, and the Neutron RestProxy plug-in. The first module (VirtualNetworkFilter) implements a MAC-based layer 2 network isolation in OpenFlow that is exposed via the REST API, as for the RestProxy plug-in it was designed to run as part of the Neutron Service so this plug-in replaces the ml2 plug-in. [33, 39]

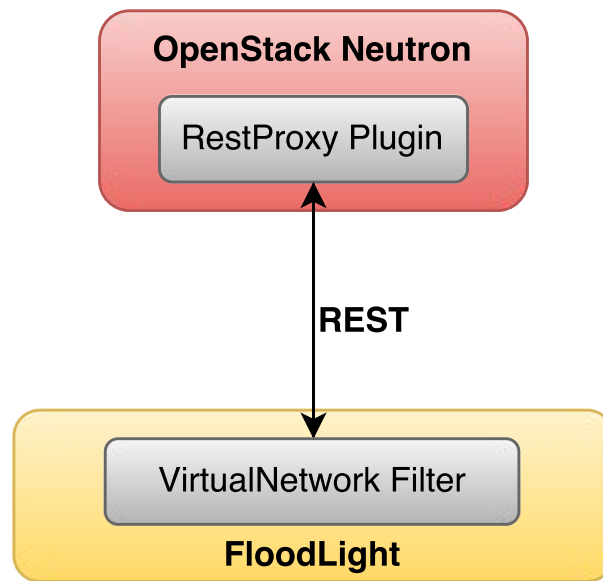


Figure 4.4: FloodLight to OpenStack communication.

SONA which stands for **Simplified Overlay Networking Architecture** (Figure 4.5) is the project that integrates ONOS into the OpenStack cloud system, it implements OpenStack neutron ML2 drivers and L3 Services for the ONOS controller, so it is needed to replace the aforementioned drivers by ONOS OpenStackSwitching plug-in, that is responsible for layer 2 communication in between VMs thus creating VxLan networks and OpenStackRouting responsible for L3 communication creating virtual logical networks in between nodes, all the communication is done via REST for both plug-ins.[40]

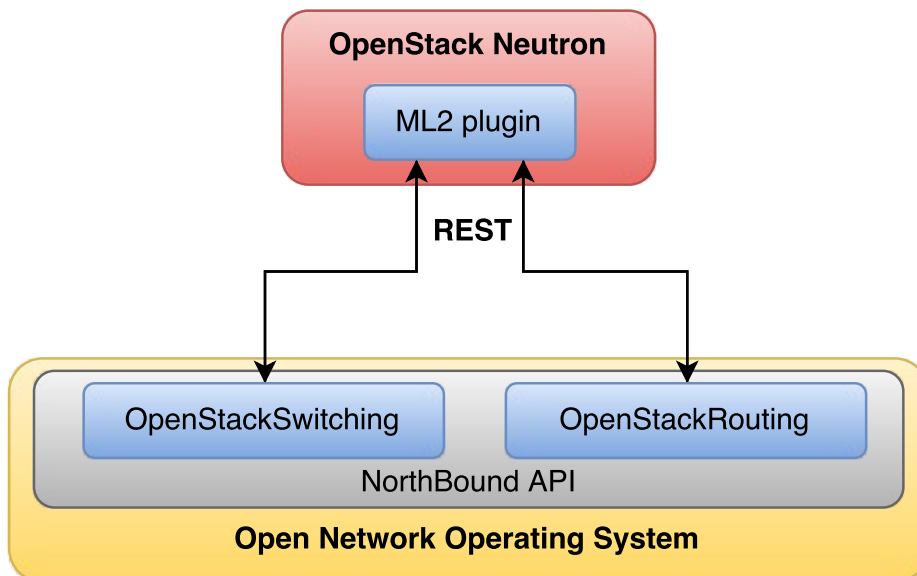


Figure 4.5: ONOS to OpenStack communication [41].

4.2 Physical SDN Architecture

For the physical architecture layout small changes had to be changed, only the hardware setup needs to change and the service placement in these machines. Regarding the machine setup it would be optimal to have a dedicated node for the SDN controller in order to not clutter the node with unnecessary services (Figure 4.6), it is suggested also that all the nodes have two NICs this way it is possible to create another physical network and thus isolating the SDN traffic. In other to host the necessary controller specific plug-ins it would be necessary to adjust the controller node Neutron services and make the according changes to the drivers and plug-ins this node is using.

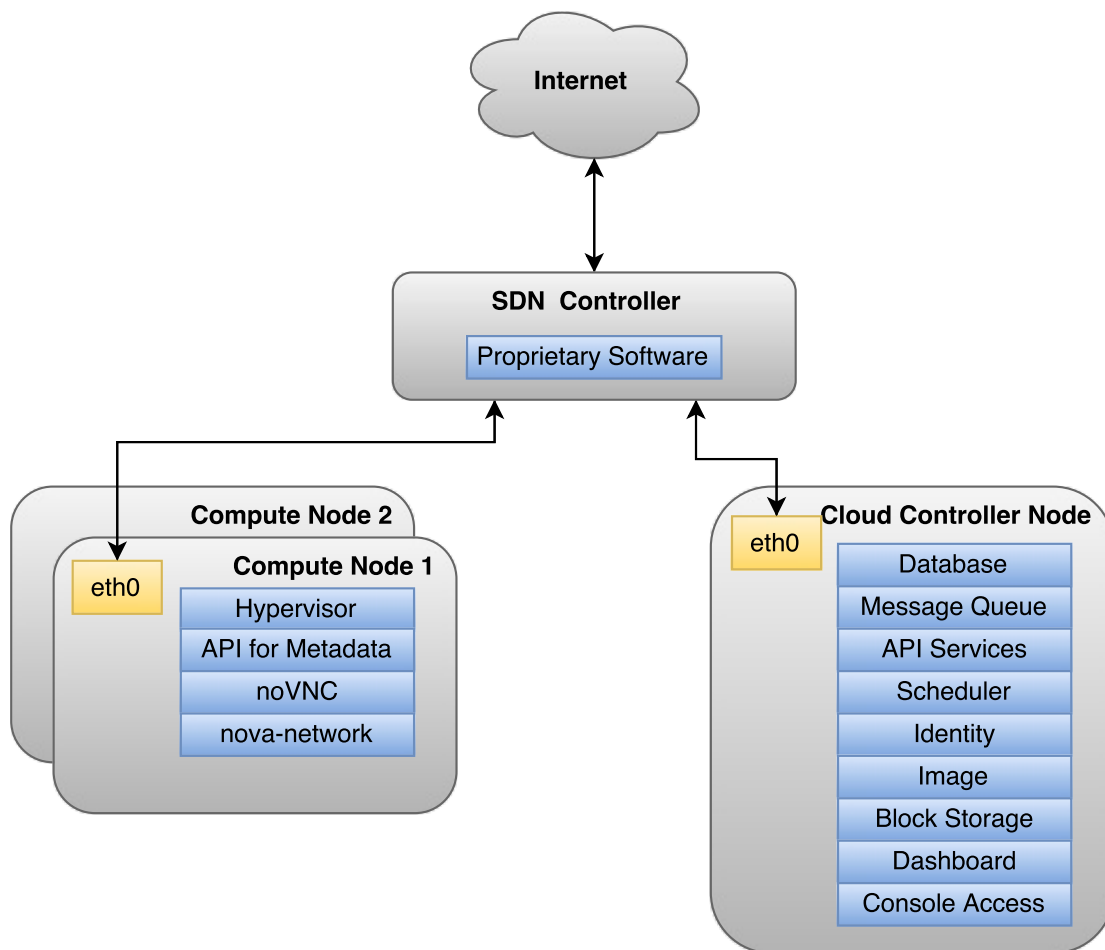


Figure 4.6: OpenStack Physical SDN Architecture.

4.3 SDN Controller Comparative Analysis

SDN Controllers work as the brain of the system when it comes to networking, they are responsible for traffic shaping and routing so, in large data centers its normal that these

controllers create choke points with this in mind there are some solutions that can help bypass if not completely avoid these problems.

The study made in [42], suggests that an away to alleviate the the controllers bottlenecks is to reduce the activity control functions, simply putted, have the controller execute fewer operations, in a linear way it can actually increase the controller performance but it can also make it work ineffectively if too many functions are pruned so, it is fundamental to find a balance which leads to a trade-off between reducing computational overhead and maintaining flow efficiency of the control execution. In order to achieve this balance a fine tuning of the operating parameters such as the pooling of network statistics rate of the Monitoring modules or an optimization of the routing algorithms in the Routing module.

In order to avoid the inevitable congestion problems inherent to large data center networks, it is possible to develop a Congestion detection module, like the one presented in [43], it aims to query, consolidate and store statistics from OpenFlow switches that would help to evaluate the load on each path and try to reroute packets onto lighter paths. The module would need to pool the switches on a fixed interval and when these switches reports heavy flow of packets the congestion conditions would be met leading to a rerouting of the packets by the controller.

When the time comes to choose a SDN controller from the three aforementioned options it all comes down to purpose, as several studies reveal [31, 44, 45, 46, 47], a comparative analysis in between the FloodLight controller and OpenDaylight [31] reveal that in small network FloodLight achieves better results in latency tests and throughput tests as a grand part of the problem with ODL being the malfunctioning of the OpenFlow plug-in.

When comparing ODL and ONOS in a small networking environment, as in [46], it is possible to conclude that ONOS leads with a lower converge time and in TCP bandwidth between hosts closely followed by ODL, as ONOS main characteristics are being a high availability controller with emphasis on scale out and performance it is obvious that on quality of service aspects it would perform better.

It is shown in [47] that ODL really shines when in a clustered environment as it has various programmable network services such as the topology service and forwarding service, it as a model driven-service abstraction layer (MD-SAL) as we saw before that when clustered it enables multiple controllers to run an identical set of network services also ODL makes it possible for a device to connect to multiple ODL controllers thus providing load balancing using a master slave connection management. Finally [44] concludes, that OpenDaylight is the a great all around controller that supports a large array of applications and excels in the integration of the emergent paradigm the Internet of Things.

As closing argument there is no clear pick on which is the best controller as, each has its perks, as for FloodLight is great for closely monitor small systems and to retrieve traffic information of the network, while ONOS deploys the best QoS as it is a carrier

4.3. SDN CONTROLLER COMPARATIVE ANALYSIS

grade controller that shines when it has to deal with traffic control over large networks, as for ODL its main characteristic is its distributed architecture that allows connectivity and control over several devices independently of the hardware.

RESULTS AND VALIDATION

The following chapter describes the tests made to the infrastructure presented in Chapter 3 in order to validate it as working industry project. The main focus of this chapter will be the optimization capabilities of the architecture, the flexibility of the hardware allocation and finally to showcase an use case in which this platform can be used and impact the actual institution. The chapter is divided in three main sections that validates the aforementioned topics.

The first section will depict how the virtualization of the hardware allied with the customization of the instance images can lead to an optimized use of the allocated resources. This information can be collected from the OpenStack Dashboard.

The second topic will show us the versatility of the platform, it will be shown what are Operative Systems it can run, how many instance it is possible to run at the same time and how they are available in the network and can be made reachable to any user inside the network.

The last subsection will showcase an use case of how and where the cloud can impact the laboratory classes of the telecommunication section or even with enough resources the most of the department classes.

5.1 Resource Optimization

The cloud environment's major strong point is its virtualization capabilities, these enable the virtualization of physical resources of the system supplying a larger pool of resources, leading to a superior optimization of the system. From the Horizon OpenStack project is possible to retrieve statistics about the global system, such as compute nodes and network resources available, so with this in mind we can easily verify our available resources how their virtualized counter-parts.



Figure 5.1: Resource usage with two instances running.

Table 5.1: Resource usage with two instances running.

Hostname	vCPUs (used/total)	RAM (used/total)	Storage (used/total)
Compute1	1 / 6	640Mb / 31.2Gb	2Gb / 517Gb
Compute2	1 / 24	512Mb / 31.4Gb	2Gb / 518Gb

So from Figure 5.1 and Table 5.1, it is possible to see that overall we have 30 CPUs, around 62 GBytes of RAM and more than 1 Terabyte of Storage spread across our two compute nodes.

When the system reaches a point where these resources aren't enough to support the workload, it is possible to overcommit the resource pool and this way enlarge it. By using the overcommitting ratios provided by Openstack we will achieve the total amount depicted in Table 5.2.

Table 5.2: Available resources after overcommit.

Hostname	vCPUs (used/total)	RAM (used/total)	Storage (used/total)
Compute1	1 / 96	640Mb / 46.8Gb	2Gb / 517Gb
Compute2	1 / 384	512Mb / 47.1Gb	2Gb / 518Gb

The virtualized resources are allocated to instances at the launch of the same. The amount of resources to be allocated can be customized based on the OS image characteristics and its objective, the same image can be deployed with different resources due the OpenStack Glance flavor selection (Figure 5.2), so by creating specific flavors for each image it is possible to spare hardware this way optimizing the remaining resources.

Even though this virtualization technology is very flexible it has its limits, as it is impossible to overcommit one instance above the available physical resources of the highest resourced computing node. This serves as failsafe feature to avoid hardware burnout.

5.2 Instance Deployment and Access

The instances are deployed on the public sub network created by OpenStack. At the launch of an instance, and unless it has specific characteristics, it will be launched on

Launch Instance

Details * Access & Security Networking * Post-Creation

Advanced Options

Availability Zone
nova

Instance Name *

Flavor *
m1.small

Instance Count *
1

Instance Boot Source *
Boot from image

Image Name *
Fedora (195.1 MB)

Specify the details for launching an instance.
The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.small
VCPUs	1
Root Disk	20 GB
Ephemeral Disk	0 GB
Total Disk	20 GB
RAM	2,048 MB

Project Limits

Number of Instances 3 of 10 Used

Number of VCPUs 3 of 20 Used

Total RAM 4,608 of 51,200 MB Used

Cancel Launch

Figure 5.2: Image launch configuration.

the highest scoring compute node with the lowest load. It will be assigned an IP via the DHCP agent and after the deployment is done it can be accessed via Secure Shell (SSH) or VNC. In addition, post-deployment access VNC also allows the user to access the instances deployment in real time as it can be seen in Figure 5.3.

```

Connected (unencrypted) to: QEMU (instance-00000040)
[ 3.348754] regulator-dummy: disabling
[ 3.349418] Magic number: 0:38:454
[ 3.350104] rtc_cmos 00:00: setting system clock to 2016-09-16 23:26:12 UTC (
1474068372)
[ 3.351770] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[ 3.351968] EDD information not available.
[ 3.381191] Freeing unused kernel memory: 1336K (ffff81d20000 - ffffffff8
1e6e000)
[ 3.381526] Write protecting the kernel read-only data: 12288k
[ 3.383557] Freeing unused kernel memory: 760K (ffff880001742000 - ffff880001
800000)
[ 3.394815] Freeing unused kernel memory: 680K (ffff880001b56000 - ffff880001
c00000)
[ 3.395221] usb 1-1: new full-speed USB device number 2 using uhci_hcd
[ 3.573631] usb 1-1: New USB device found, idVendor=0627, idProduct=0001
[ 3.575304] usb 1-1: New USB device strings: Mfr=1, Product=3, SerialNumber=5
[ 3.575538] usb 1-1: Product: QEMU USB Tablet
[ 3.575704] usb 1-1: Manufacturer: QEMU
[ 3.575833] usb 1-1: SerialNumber: 42
[ 3.756604] Switched to clocksource tsc
[ 3.849832] systemd-udev[99]: starting version 204
[ 4.293153] FDC 0 is a S82078B
[ 5.476364] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042
/serio0/input/input3

```

Figure 5.3: VCN access to an Ubuntu Server instance deployment.

Since OpenStack assigns an IP to an instance, the access via SSH can be done with keypair authentication and via PuTTY as we can see in Figure 5.4 is possible to work on

this instance that is running in our cloud.

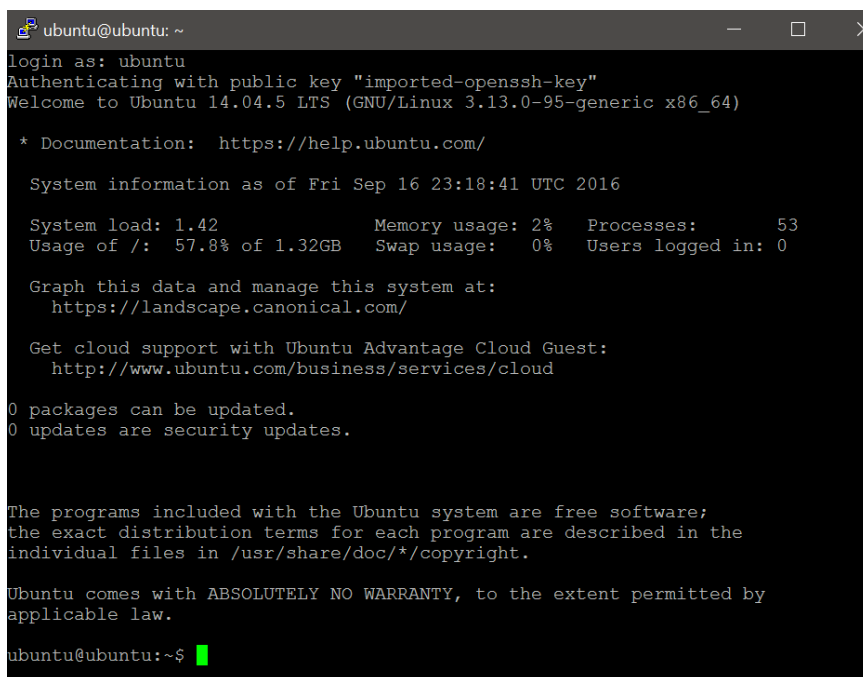
A screenshot of a terminal window showing an SSH session. The terminal title is 'ubuntu@ubuntu: ~'. The output shows the login process: 'login as: ubuntu', 'Authenticating with public key "imported-openssh-key"', and 'Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-95-generic x86_64)'. It then displays system information as of 'Fri Sep 16 23:18:41 UTC 2016', including system load (1.42), memory usage (2%), swap usage (0%), processes (53), and users logged in (0). It also provides links for documentation and system management. At the bottom, it shows '0 packages can be updated.' and '0 updates are security updates.' followed by a copyright notice and the prompt 'ubuntu@ubuntu:~\$' with a green cursor.

Figure 5.4: SSH access to an Ubuntu Server instance running.

Once inside the instance, the user can work as if it was on its own machine. It can have access to sudo commands, deploy apps and run programs; these instances also have access to the Internet but at this time it is impossible to access them from the Internet since there are no public IPs available. Once these become available, to make the instances accessible from the outside network, they need to have a floating IP (Public IP) assigned to them.

Based on the resources the user wishes to allocate to each instance it is possible to run several at the same time, for instance in Figure 5.5 we can see up to 10 instances running. Since cirrOS is a minimal OS, it only requires only 1 vCPU, 1 Gb root disk and 512 vRam, we would be able to run up to 20 instances since we are only limited by the amount of virtual cores we can emulate.

For a more realistic example lets use an Ubuntu Server image: it has higher resources requirements but even allocating instance with 1 vCPU, 20Gb root disk and 2Gb of Ram we would reach the same constraint of 20 vCPUs before the instances would be limited by other resources.

So it is possible to run several instances in our cloud depending on the performance we want it to have. By allocating more resources, the trade-off is a lower amount of instances running. As of now the quotas established for each project are at ten in order to be able to prevent one user of starving the others.

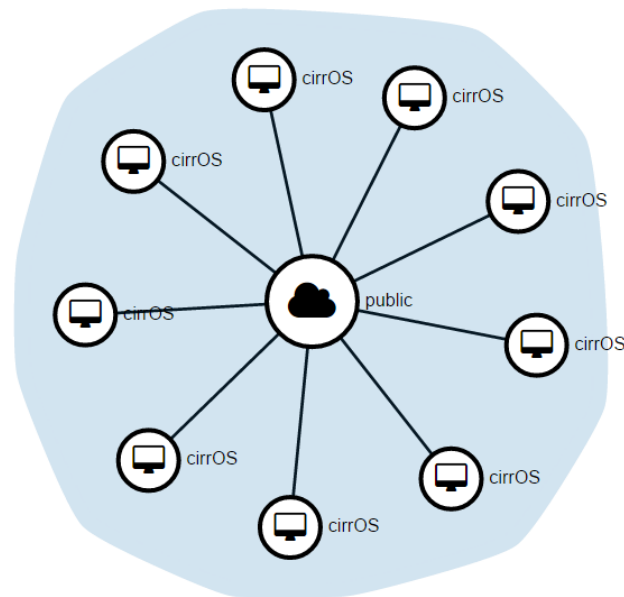


Figure 5.5: 10 cirrOS instances running.

5.3 Use Cases

In order to offer some purpose to the implemented cloud, an use case was designed. It revolves around laboratory classes as the majority of the work done in experimental classes is done via virtual machines. These machines are often large and demand some setup which usually means one or more classes lost with the setting up.

So in order to avoid this situation, the virtual machines that are usually provided to students can be uploaded into the cloud and have the students work remotely from their machines.

Even though as of now students wouldn't be able to work from home - which isn't necessarily a bad thing since it would enforce students to work at class and not skip it - the teacher could monitor the students' progress more closely. Another advantage of this scenario is that there are no hardware constraints for virtual machines, so students would be able to work on heavier VMs from their computers or even lab computers that usually have lower specs. One downside of this use case is that the network where the cloud is deployed needs to be expanded and made available to the lab rooms so that students can access instances.

It is important to underline that students don't need to have access to the cloud, only the instances. Ergo, no account and privilege setup is needed in order for them to use the platform.

5.4 Discussion of Results

It is safe to assume that the cloud is fully functional and ready to be deployed in the academic environment. Although the hardware available is lackluster and cannot muster the required resources for load heavy tasks required in an investigation and development environment, it can be used as a support for lab classes and student tasks and tests. Some problems related with the hardware are possible to surface, since the cloud is not deployed on actual servers but desktops instead. It would be a good practice to migrate the platform into newer and more potent machines.

The cloud has Fedora, Ubuntu and CirrOS images ready to be launched. These instances are accessible via SSH, and a lot more images such as Windows server or Ubuntu desktop can be uploaded if required. The system as of now only has these three, since it wasn't necessary to add more instances to test the cloud infrastructure.

Some problems that can arise like it was mentioned before are network related. Since there isn't a management network dedicated for instance communication, it is possible that when several instances are launched the network quickly reaches a disruption point and crash. In order to avoid this the study made in Chapter 4 will help further development of the cloud when these problems surface.

CONCLUSION AND FUTURE WORK

6.1 Conclusion

It is possible to conclude from the work developed and described in this document that it is possible to implement a cloud infrastructure with the available hardware, the overall characteristics of the cloud were preserved, modularity and flexibility are cornerstone to the architecture.

Furthermore, the instances deployment tests were successful, as it was possible to launch up to twenty OSs instances as long as only one vCPU is allocated to each instance. The cloud instances can be accessed via SSH or VNC protocols, the architecture is also able to support image upload, subnet management and access control.

The aforementioned characteristics validate the cloud structure and applicability as a real industrial cloud that in a small scale research and development scenario can be used as a testbed also the cloud is ready to support lab classes within the department.

The results presented on Chapter 5 suggest that although the hardware and network resources available aren't tailored to support a cloud system it is safe to admit that a robust cloud was created that can successfully deploy the virtualization environment.

6.2 Future Work

Future efforts should focus on augmenting the cloud infrastructure, it should be migrated to hardware designed for its purpose, add more computing and storage nodes in order to enable the processing of the more complex tasks and allow volume storage allocation to instances. The creation of a dedicated network would also enhance the platform, adding a management network in order to avoid node congestion due to instance verbosity and move the network management features from Neutron to a SDN Controller.

BIBLIOGRAPHY

- [1] R. Buyya, R. Buyya, C. S. Yeo, C. S. Yeo, S. Venugopal, S. Venugopal, J. Broberg, J. Broberg, I. Brandic, and I. Brandic. “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. In: *Futur. Gener. Comput. Syst.* 25.JUNE (2009), p. 17. ISSN: 0167-739. DOI: 10.1016/j.future.2008.12.001.
- [2] P. Mell and T. Grance. “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology”. In: *Natl. Inst. Stand. Technol. Inf. Technol. Lab.* 145 (2011), p. 7. ISSN: 1472-0213. DOI: 10.1136/emj.2010.096966. arXiv: 2305-0543.
- [3] S. P. Mirashe and N. V. Kalyankar. “Cloud Computing”. In: 51.7 (2010), p. 9. DOI: 10.1145/358438.349303.
- [4] Q. Zhang, L. Cheng, and R. Boutaba. “Cloud computing: State-of-the-art and research challenges”. In: *J. Internet Serv. Appl.* 1.1 (2010), pp. 7–18. ISSN: 18674828. DOI: 10.1007/s13174-010-0007-6. arXiv: S0167739X10002554.
- [5] I. Sriram and A. Khajeh-Hosseini. “Research Agenda in Cloud Technologies”. In: *1st ACM Symp. Cloud Comput. SOCC cs.DC* (2010), pp. 1–11. arXiv: 1001.3259.
- [6] N. Antonopoulos and L. Gillam, eds. *Cloud Computing: Principles, Systems and Applications*. Vol. 1. Springer, 2015. ISBN: 9788578110796. DOI: 10.1017/CB09781107415324.004. arXiv: arXiv:1011.1669v3.
- [7] S. Goyal. “Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review”. In: *Int. J. Comput. Netw. Inf. Secur.* 6.3 (2014), pp. 20–29. ISSN: 20749090. DOI: 10.5815/ijcnis.2014.03.03.
- [8] A. Marinos and G. Briscoe. “Community cloud computing”. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 5931 LNCS (2009), pp. 472–484. ISSN: 03029743. DOI: 10.1007/978-3-642-10665-1_43. arXiv: 0907.2485.
- [9] R. Palwe, G. Kulkarni, and A. Dongare. “A New Approach to Hybrid Cloud”. In: *Int. J. Comput. Sci. Eng. Res. Dev.* (2011), pp. 35–42.

BIBLIOGRAPHY

- [10] S. Yadav. “Comparative Study on Open Source Software for Cloud Computing Platform : Eucalyptus , Openstack and Opennebula”. In: *Int. J. Eng. Sci.* 3.10 (2013), pp. 51–54.
- [11] C. Paper, R. Kumar, and J. Foundation. “Open Source Solution for Cloud Computing”. In: *Int. J. Comput. Sci. Mob. Comput.* MAY (2014). DOI: 10.13140/2.1.1695.9043.
- [12] J. S. Manjaly and S Jisha. “A Comparative Study on Open Source Cloud Computing Frameworks”. In: *An Int. J. Eng. Technol.* 2.6 (2013), pp. 2026–2029.
- [13] S. Ismaeel, A. Miri, D. Chourishi, and S. M. R. Dibaj. “Open Source Cloud Management Platforms: A Review”. In: *2015 IEEE 2nd Int. Conf. Cyber Secur. Cloud Comput.* (2015), pp. 470–475. DOI: 10.1109/CSCloud.2015.84.
- [14] D. Grzonka. “Short Analysis of Implementation and Resource Utilization for the Openstack Cloud Computing Platform SHORT ANALYSIS OF IMPLEMENTATION AND RESOURCE”. In: MAY (2015). DOI: 10.13140/RG.2.1.3328.9444.
- [15] T. G. Lynn, P. Healy, and J. P. Morrison. “A Comparative Study of Current Open-Source Infrastructure as a Service Frameworks A Comparative Study of Current Open-Source Infrastructure as a”. In: MAY (2015).
- [16] *OpenStack Docs: OpenStack Architecture Design Guide*. URL: <http://docs.openstack.org/arch-design/> (visited on 02/20/2016).
- [17] *OpenStack Docs: OpenStack Releases*. URL: <http://releases.openstack.org/> (visited on 02/20/2016).
- [18] A. Datt, A. Goel, and S. C. Gupta. “Analysis of Infrastructure Monitoring Requirements for OpenStack Nova”. In: *Procedia Comput. Sci.* 54 (2015), pp. 127–136. ISSN: 18770509. DOI: 10.1016/j.procs.2015.06.015. URL: <http://dx.doi.org/10.1016/j.procs.2015.06.015>.
- [19] K. Bakshi. “Network considerations for open source based clouds”. In: *IEEE Aerosp. Conf. Proc.* 2015-June (2015), pp. 1–9. ISSN: 1095323X. DOI: 10.1109/AERO.2015.7118997.
- [20] K. Pepple. *Deploying OpenStack*. Ed. by M. Loukides and M. Blanchette. First Edit. Vol. 1. Sebastopol: O’Reilly, 2015. ISBN: 9788578110796. DOI: 10.1017/CB09781107415324.004. arXiv: arXiv:1011.1669v3.
- [21] F. Brey, S Guggenbichler, and J. Wollmann. *OpenStack Cloud Computing Cookbook*. 2008. ISBN: 9781782167587. URL: <http://medcontent.metapress.com/index/A65RM03P4874243N.pdf>.
- [22] B. Dongmyoung and L. Bumchul. “Analysis of Telemetry Service in OpenStack”. In: (2015), pp. 272–274.

- [23] P. Biswas, F. Patwa, and R. S. Sandhu. “Content Level Access Control for OpenStack Swift Storage”. In: *Proc. 5th {ACM} Conf. Data Appl. Secur. Privacy, {CODASPY} 2015, San Antonio, TX, USA, March 2-4, 2015* (2015), pp. 123–126. DOI: 10.1145/2699026.2699124.
- [24] Z. Yao, I. Papapanagiotou, and R. Griffith. “Serifos: Workload Consolidation and Load Balancing for SSD Based Cloud Storage Systems”. In: (2015). arXiv: 1512.06432.
- [25] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Member, and S. Uhlig. “Software-Defined Networking : A Comprehensive Survey”. In: *Proc. IEEE* 103.1 (2015), pp. 14–76. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999. arXiv: arXiv:1406.0440v3.
- [26] Open Networking Foundation. “Software-Defined Networking: The New Norm for Networks [white paper]”. In: *ONF White Pap.* (2012), pp. 1–12.
- [27] J. Chen, X. Zheng, and C. Rong. “Survey on software-defined networking”. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 9106.1 (2015), pp. 115–124. ISSN: 16113349. DOI: 10.1007/978-3-319-28430-9_9.
- [28] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez. “Integrated SDN/NFV Management and Orchestration Architecture for Dynamic Deployment of Virtual SDN Control Instances for Virtual Tenant Networks [Invited]”. In: *J. Opt. Commun. Netw.* 7.11 (2015), B62. ISSN: 1943-0620. DOI: 10.1364/JOCN.7.000B62.
- [29] V Shamugam and I Murray. “Software Defined Networking challenges and future direction: A case study of implementing SDN features on OpenStack private cloud”. In: 012003 (2016), pp. 0–8. ISSN: 1757899X. DOI: 10.1088/1757-899X/121/1/012003.
- [30] A. Mann, W. Wei, R. Badana, T. Zhang, and T. A. Yang. “Use Cases and Development of Software Defined Networks in OpenStack”. In: *2015 IEEE Int. Conf. Comput. Inf. Technol. Ubiquitous Comput. Commun. Dependable, Auton. Secur. Comput. Pervasive Intell. Comput.* (2015), pp. 947–953. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.142.
- [31] Z. K. Khattak, M. Awais, and A. Iqbal. “Performance evaluation of OpenDaylight SDN controller”. In: *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS 2015-April* (2015), pp. 671–676. ISSN: 15219097. DOI: 10.1109/PADSW.2014.7097868.
- [32] *ODL Beryllium (Be) - The Fourth Release of OpenDaylight | OpenDaylight*. URL: <https://www.opendaylight.org/odlbe>.
- [33] *Floodlight OpenFlow Controller -Project Floodlight*. URL: <http://www.projectfloodlight.org/floodlight/>.

- [34] T. O.N. L. (ON.Lab). "Introducing ONOS - a SDN network operating system for Service Providers !" In: (2014).
- [35] *ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out*. URL: <http://onosproject.org/>.
- [36] P. R. Srivastava and S. Saurav. "Networking agent for overlay L2 routing and overlay to underlay external networks L3 routing using OpenFlow and Open vSwitch". In: *17th Asia-Pacific Netw. Oper. Manag. Symp. Manag. a Very Connect. World, APNOMS 2015* (2015), pp. 291–296. DOI: 10.1109/APNOMS.2015.7275442.
- [37] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea. "Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case". In: *2014 IEEE 3rd Int. Conf. Cloud Networking, CloudNet 2014* (2014), pp. 132–137. DOI: 10.1109/CloudNet.2014.6968981.
- [38] A Mayoral, R Vilalta, R Munoz, R Casellas, R Martinez, and J Vilchez. "Integrated IT and Network Orchestration Using OpenStack , OpenDaylight and Active Stateful PCE for Intra and Inter Data Center Connectivity". In: 1 (), pp. 1–3.
- [39] O. Tkachova, Mohammed Jamal Salim, and Abdulghafoor Raed Yahya. "An Analysis of SDN-Openstack Integration". In: 0 (2011), pp. 42–43.
- [40] *SONA: DC Network Virtualization - ONOS - Wiki*. URL: <https://wiki.onosproject.org/display/ONOS/SONA/3A+DC+Network+Virtualization>.
- [41] *SONA Architecture - ONOS 1.5 - Wiki*. URL: <https://wiki.onosproject.org/display/ONOS15/SONA+Architecture>.
- [42] C. Caba and J. Soler. "Mitigating SDN controller performance bottlenecks". In: *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN 2015-Octob* (2015). ISSN: 10952055. DOI: 10.1109/ICCCN.2015.7288429.
- [43] M. Gholami and B. Akbari. "Congestion Control in Software Defined Data Center Networks Through Flor Rerouting". In: (2015), pp. 654–657.
- [44] O. Salman, I. H. Elhadj, A. Kayssi, and A. Chehab. "SDN Controllers : A Comparative Study". In: 978 (2016), pp. 18–20.
- [45] S. Scott-hayward. "Design and deployment of secure , robust , and resilient SDN Controllers". In: *NetSoft* (2015), pp. 1–5. DOI: 10.1109/NETSOFT.2015.7258233.
- [46] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici. "A comparison between several Software Defined Networking controllers". In: *2015 12th Int. Conf. Telecommun. Mod. Satell. Cable Broadcast. Serv.* (2015), pp. 223–226. DOI: 10.1109/TELSKS.2015.7357774. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7357774>.
- [47] D. Suh, S. Jang, S. Han, and S. Pack. "On Performance of OpenDaylight Clustering". In: (), pp. 407–410.