INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Live-Migration in Cloud Computing Environment

MOHAMMAD ALSELEK
Outubro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Live-Migration in Private Cloud Computing Environment

Mohammad Ibrahim AlSelek

Dissertation for obtaining A Master's Degree in Computer Engineering, Specialization in Computer Systems

Supervision by:

ISEP: Dr. Jorge Pinto Leite

Novabase: Eng. Pedro André & Eng. Jonathan Carvalho

Porto, October 2016

Resumo

O tráfego global de IP aumentou cinco vezes nos últimos cinco anos, e prevê-se que crescerá três vezes nos próximos cinco. Já para o período de 2013 a 2018, anteviu-se que o total do tráfego de IP iria aumentar a sua taxa composta de crescimento anual (CAGR) em, aproximadamente, 3.9 vezes. Assim, os Prestadores de Serviços estão a sofrer com este acréscimo exponencial, que é proveniente do número abismal de dispositivos e utilizadores que estão ligados à Internet, bem como das suas exigências por vários recursos e serviços de rede (como por exemplo, distribuição de conteúdo multimédia, segurança, mobilidade, etc.). Mais especificamente, estes estão com dificuldades em: introduzir novos serviços geradores de receitas; e otimizar e adaptar as suas infraestruturas mais caras, centros de processamento de dados, e redes empresariais e de longa distância (COMpuTIN, 2015).

Estas redes continuam a ter sérios problemas (no que toca a agilidade, gestão, mobilidade e no tempo despendido para se adaptarem), que não foram corrigidos até ao momento. Portanto, foram propostos novos modelos de Virtualização de Funções da Rede (NFV) e tecnologias de Redes de Software Definidos (SDN) para solucionar gastos operacionais e de capital não otimizado, e limitações das redes (Lopez, 2014, Hakiri and Berthou, 2015).

Para se ultrapassar tais adversidades, o Instituto Europeu de Normas de Telecomunicações (ETSI) e outras organizações propuseram novas arquiteturas de rede. De acordo com o ETSI, a NFV é uma técnica emergente e poderosa, com grande aplicabilidade, e com o objetivo de transformar a maneira como os operadores desenham as redes. Isto é alcançado pela evolução da tecnologia padrão de virtualização TI, de forma a consolidar vários tipos de equipamentos de redes como: servidores de grande volume, *routers*, *switches* e armazenamento (Xilouris et al., 2014).

Nesta dissertação, foram usadas as soluções mais atuais de SDN e NFV, de forma a produzir um caso de uso que possa solucionar o crescimento do tráfego de rede e a excedência da sua capacidade máxima.

Para o desenvolvimento e avalização da solução, foi instalada a plataforma de computação na nuvem *OpenStack*, de modo a implementar, gerir e testar um caso de uso de *Live Migration*.

Palavras-chave: Computação na Nuvem, *OpenStack*, *Open Virtual Switch*, Redes de Software definidos, Rota Virtual Distribuída, Virtualização, Virtualização de Funções da Rede

Abstract

Global IP traffic has increased fivefold over the past five years, and will continue increasing threefold over the next five years. The overall IP traffic will grow at a compound annual growth rate (CAGR) nearly 3.9-fold from 2013 to 2018. Service Providers are experiencing the exponential growth of IP traffic that comes from the incredible increased number of devices and users who are connected to the internet along with their demands for various resources and network services like multimedia content distribution, security, mobility and else. Therefore, Service Providers are finding difficult to introduce new revenue generating services, optimize and adapt their expensive infrastructures, data centers, wide-area networks and enterprise networks (COMpuTIN, 2015). The networks continue to have serious known problems, such as, agility, manageability, mobility and time-to-application that have not been successfully addressed so far. Thus, novel Network Function Virtualization (NFV) models and Software-defined Networking (SDN) technologies have been proposed to solve the non-optimal capital and operational expenditures and network's limitations (Lopez, 2014, Hakiri and Berthou, 2015).

In order to solve these issues, the European Telecommunications Standards Institute (ETSI) and other standard organizations are proposing new network architecture approaches. According to ETSI, The Network Functions Virtualization is a powerful emerging technique with widespread applicability, aiming to transform the way that network operators design networks by evolving standard IT virtualization technology to consolidate many network equipment types: high volume servers, routers, switches and storage (Xilouris et al., 2014).

In this thesis, the current Software-Defined Networking (SDN) and Network Function Virtualization (NFV) solutions were used in order to make a use case that can address the increasing of network traffic and exceeding its maximum capacity.

To develop and evaluate the solution, OpenStack cloud computing platform was installed in order to deploy, manage and test a Live-Migration use-case.

Keywords: Network Function Virtualization, Software Defined Network, OpenStack, Virtualization, Cloud Computing, Open Virtual Switch, Distributed Virtual Route.

INDEX

List of Figures

List of Tables

Acronyms

CAPEX	CApital EXpenditures
CDNs	Content Delivery Networks
CPU	Central Processing Unit
DVR	Distributed Virtual Router
IaaS	Infrastructure as a Service
IoT	Internet of Things
IT	Information Technology
LTE	Long Term Evolution
NaaS	Network as a Service
NAT	Network Address Translation
NFV	Network Function Virtualization
NIC	Network Interface Controller
NIST	National Institute of Standards and Technology
OPEX	Operational EXpenditure
OVS	Open Virtual Switch
PaaS	Platform as a Service
PoC	Proof of Concept
QoS	Quality of Service
SaaS	Software as a Service
SDN	Software Defining Network
SLA	Service Level Agreements
SPOF	Single Point Of Failure
TCP/IP	Transmission Control Protocol/Internet Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	Virtual Machine Monitor
VNF	Virtual Network Function
VPN	Virtual Private Network

Chapter 1

1. Introduction

The chapter introduces some problems and challenges that are facing the server providers, telecommunication and IT organizations. After that, it mentions that the virtualization is the key solution of these problems. Moreover, the objectives that should be achieved, as well as, the main values of this thesis's project are included. Finally, the organization of the document for each chapter is made.

1.1 Motivation - Problem Description

The technology continues to impact our life positively in various aspects of our daily lives such as social, economic, health, and education. Meanwhile, the technology related to networking and telecommunication is growing exponentially with notable increasing number of devices and users who are using this technology with an exploding demand for various resource and services like security, mobility, scalability and automation. Around 40% of the total world population (3.5 billion mark) is using the Internet today, which is more than 3 times the number in the year 2005. The number of internet users is still increasing; it has increased tenfold from 1999 to 2013 (internetlivestats, Data, 2014).

Moreover, some researches are predicting that every person will probably have around 6.5 devices (50 billion devices in total) connected to the internet in 2020, while it was around 1.8 devices per person (12.5 billion devices in total) in 2010 (Jun et al., 2011).

Finally, the total global IP traffic could reach to 1000 Exabyte per year mark in 2016 comparing 600 Exabyte per year in 2013 (Swanson and Gilder, 2008). This huge increasing of devices in networking and telecommunication space along with its explosive traffic growth rates is coming with the emerging of new information technologies (IT) like Internet of Things (IoT) involving M2M communications, 4G/Long Term Evolution (LTE) for mobile communications, multimedia Content Delivery Networks (CDNs), and Big Data analytics for intelligent cloud-based services and applications(Turner and Taylor, 2005).

The current TCP/IP (Transmission Control Protocol/Internet Protocol) based Internet architecture, which is used until now, is not able to meet such explosive demands efficiently. As a result, obstacles could be faced because of limiting innovation in provisioning services and networks management. This limitation creates challenges for several service providers. Some of important challenges are reducing CAPEX and OPEX costs, time-to-market competition between service providers, and the service availability and scalability when it is required by the customer. These challenges come with complexity and inflexibility (Bertaux et al., 2015).

To address these challenges and get the benefits behind, service providers and network operators are starting to support the virtualization concept for their networks. The virtualization technique has the success key in addressing similar challenges and issues in computing and storage fields of Information Technology (IT). In addition, virtualization is the main factor for various cloud-based service models (e.g. IaaS, PaaS, and SaaS), which can overcome all the challenges and enable innovation in provisioning and management of services (Sen, 2013). In essence, virtualization is an architecture to share the physical resources in underlay infrastructure and enable innovation in provisioning and management of services running over it. Recently, new technologies working with or beside virtualization are proposed, like Software Defined Networking (SDN) and Network Function Virtualization (NFV) which are the revolution in innovation in the networking community.

1.2 Objectives

The main objective of this thesis is to design, implement and evaluate a Live-Migration use-case that address the data loss that occurs when the service provider is in maintenance phase of its infrastructure, by using SDN and NFV technologies as its key virtualization enabling network technologies. From this main objective the following step-by-step sub-objectives are derived:

1. Identifying the state-of-the-art related to SDN and NFV technologies;
2. Identifying the related work and existing solution including the use cases that are already deployed;
3. Developing a comprehensive cloud platform to provide rapid service innovation through software-based deployment and operationalization of network functions and end-to-end services;
4. Providing a solution (use-case) that is able to improve operational efficiencies by implementing automation and orchestration features;
5. Improving capital efficiencies, compared with dedicated hardware implementation, by reducing the power usage, maximizing time-to-market roll out services and managing and maintaining the entire network through a single and centralized cloud platform.

In order to achieve the main objectives of this work, it was necessary to prove that the Live-Migration Use-Case was deployed successfully which in turn leads to prove that all the behind objectives were obtained. This was done by demonstrating a step-by-step procedure of implementing the required DevStack building blocks, troubleshooting and testing them, then deploying the Live Migration mechanism.

1.3 Value Proposition

The cloud computing is the off-premises service that enables the business (organization) to run applications and store data in a pay-as-you-go model via networks such as the Internet. Cloud computing can help reduce the amount of on-premises computing infrastructure and IT overhead the organization requirements, because the only thing that should be paid is the amount of computing resources that are going to be used. As the business needs grow or shift, the cloud resources are scaled and managed flexibly to adapt the changing needs without needing to buy more computer hardware or lease/buy more space for a larger on-site datacenter (Katzan, 2011).

Cloud computing can also improve the agility of the organization. Without cloud computing technique the company has to buy, tend, update, secure, license, and manage them. While the cloud computing gives the possibility of procuring IT resources with minimal cost and essentially no time, which has significantly decreased the time it takes to bring a product to the market. Because instead of provisioning servers, the organization can focus on its business, deploy critical applications quickly, delight its customers, and stay one step ahead of its competitors. At the end, it is possible to get the same computational power without having to own the entire computing platform (Zhang et al., 2010).

A smaller company can become nimble and flexible quickly, because it is able to add intelligence and analytics from big data or customer relationship management without having to go through the process of purchasing, installing, and training. Just by signing up for a service that delivers big-picture analytics and intelligence in real time, allows you the company to take the opportunities more strategically and more quickly. With the availability of on-demand cloud resources, cloud project´s user of the company can try out new ideas or run new reports without big investments in new IT. With pay-as-you-go cloud resources, new configurations can be up and running within minutes (Foster et al., 2008).

1.4 Organization

The present document is organized as following:

Chapter 2 introduces the cloud computing and illustrates its architecture and service and deployment models. The technical values of cloud computing are showed, as well as, its business values. Moreover, the software-defined solutions are described in detail, such as virtualization, SDN and NFV. Furthermore, a private cloud solution so-called OpenStack is studied to be the main platform and the base of deploying the Live-Migration use-case.

Chapter 3 describes all the phases needed to implement, manage and evaluate the certain use-case. Therefore, three main steps are implemented which are: (i) pre-implementation to study and choose the best hardware and software tools that are going to be used. (ii) during-implementation which presents step-by-step procedure in order to deploy the Live-Migration use-case successfully. (iii) post-deployment to evaluate the results.

Chapter 4 concludes all the work done so far and reviews the results concerning assumptions and requirements. This chapter presents future work and possible improvements to be considered.

Chapter 2

2. State of The Art

In this chapter the cloud computing technology is illustrated in detail, as well as, the virtualization technology, SDN and NFV. After that, OpenStack which is the main software used to implement, deploy and evaluate cloud use-cases is described deeply.

2.1 Introduction

This chapter focuses on studying and analyzing the open source-based cloud computing technology so called OpenStack for private and public cloud, while it explains live migration concept and how much it is important to combine both mechanisms to achieve visibility, agility and total seamless migration.

First of all, it defines the concept of cloud computing and its related technologies. Moreover, it explains the service and deployment models, as well as, the cloud computing benefits, advantages and disadvantages. After that, an overview of the cloud computing business perspective is presented.

Moreover, this chapter illustrates the virtualization mechanisms, presents the software defined network architecture and introduces the network function virtualization framework.

In addition, OpenStack, the open source-based solution is described. In this part the OpenStack's architecture and its core services are explained which gives an idea about how OpenStack works. Then the market of OpenStack is mentioned to show the importance of OpenStack in reality.

Finally, the last part of this chapter presents three use-cases using OpenStack, which are the Live-Migration, Big-Data and Private Cloud. Deploying these use-cases will prove the proof-of-concept of cloud computing and OpenStack, as well as, the power of services provisioning. Therefore, one of the use-cases was chosen to be deployed, which was the live migration.

2.2 Cloud computing concept

Cloud computing is an elastic computational paradigm in which virtualized and dynamically scalable resources are provided as services and not as products, these services are delivered to

the end users/customers over a network, usually the internet through a variety of devices such as laptops, smartphones and PCs (Armbrust et al., 2010).

In other words, the existing hardware devices in the data center and their software are constituted the cloud. The applications/software that are created to be offered as services over the internet are often called software-as-a-service (SaaS). Both the hardware and software systems integrate together to comprise the cloud computing services. These services such as computation, software, data access, and storage services do not ask the end-user to be aware of infrastructure location or configuration of the system that provides the services (Mell and Grance, 2011).

Cloud computing gives the end-user/customer a feature called pay-as-you-use which is used to offer resources scalability, so the customer will just be charged for as much as he exactly used of resources existed in the data center. Unless the data center is located within the organization itself, then it is defined as private cloud. The concept is similar to the one found in the electric grid or water supply payment models, as the user/client pays for the amount of resources consumed (Lau et al., 2010).

The idea of cloud computing is coming from a continuous need of IT which leads to increase the capacity or add capabilities on demand without training new staff, investing in new infrastructure, or purchasing new software (Zhang et al., 2010).

With Cloud computing the scalable and frequently virtualized resources will be provisioned dynamically, and consequently the IT Services are delivered by using Internet Technologies. These features are the consequence of the ease-of-access to external resources by using web-based tools or applications that gives the ability to access through a web browser as if the resources were locally installed on the user's own computers. On the other hand, the fast, reliable and secure network infrastructures enable users to easily access to Cloud Computing services which are located in centralized datacenters around the globe (Foster et al., 2008).

The definition of Cloud Computing is a controversial topic. Academic researchers, analyst firms, IT organizations and Information Technology institutions define Cloud Computing in different ways. Gartner and Merrill Lynch, two analyst firms, put a definition as following:

- "A style of computing in which massively scalable IT related capabilities are provided "as a service" using Internet technologies to multiple external customers." - Gartner (Plummer et al., 2008);
- "The idea of delivering personal (*e.g.*, email, word processing, presentations) and business productivity applications (e.g., sales force automation, customer service, accounting) from centralized servers" - Merrill Lynch (Stanoevska-Slabeva and Wozniak, 2010).

The opinion of the scientific communities is not unanimous but still adds to the definition of the cloud computing. They are focusing on the end user perspective, and the architectural aspects of the infrastructure including data center hardware, virtualization and scalability, as can be read in the following definitions:

- "Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud." - Berkeley (Fox et al., 2009);
- "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers." - R. Buyya (Vaquero et al., 2008);
- Cloud Computing is "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet." – Foster (Foster et al., 2008).

More recently, the U.S. Government's National Institute of Standards and Technology (NIST) published the final version of Cloud Computing definition as (Mell and Grance, 2011):

- "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.*, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

2.3 Cloud Computing Technologies

The key technology of Cloud Computing is virtualization which refers to the creation of virtual resources, such as, computing, networking and storage. The main goal is to centralize management tasks while improving scalability, flexibility and overall hardware-resource utilization.

Recently, new technologies working with or beside virtualization have been proposed. SDN and NFV are the revolution in innovation within the networking community. SDN and NFV are complementary, but it is possible to develop one of them separately. The main objectives of SDN are decoupling the network control and data functions, centralization of control and programmability of network, while NFV is focusing on relocating the network functions from dedicated applications to generic servers. By applying NFV/SDN approaches, the new generation of service provider networks will have less of complexity, inflexibility and instead will support more scalability, reliability and interoperability (Costa-Requena et al., 2015).

The cloud computing architecture mainly based on three service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). After building the architecture, the cloud can be provisioned in four deployment models: Private, Community, Public or Hybrid. The following subchapters describe in detail these models.

2.4 Cloud computing Service Models

Cloud Computing enables the infrastructure resources and the software applications to be delivered as services. In this context, the term service means that they are provided on demand and paid on a utility computing usage basis. The cloud computing services have developed by different vendors. Usually these services are defined with specific taxonomy, for example, XaaS where X refers to the provided cloud service. According to NIST (Mell and Grance, 2011), these services can be classified into one of three delivery models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). These three service classes are also known as the Software, Platform, and Infrastructure (SPI) service model (Chen and Zhao, 2012).

The service orchestration consists of three layers, the service layer includes SaaS layer which is built on top of the PaaS layer and the PaaS layer in turn is created on the top of the IaaS service layer and all of these services are abstracted from the physical resource layer through a resource abstraction and control layer, as presented in the Figure [1].

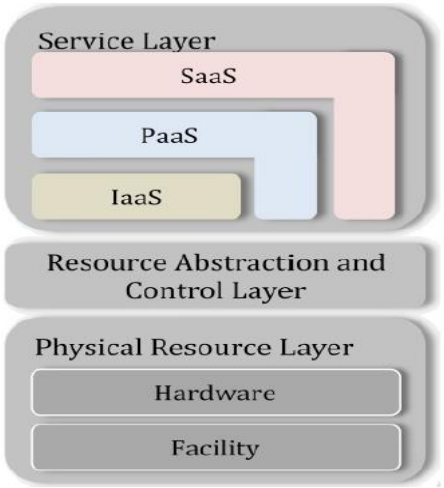

Figure 1 Service orchestration (Foster, 2013)

The service orchestration points out that the deployment of an upper service, such as SaaS, means that all the underlying services are provisioned by the cloud provider. This means that when the cloud consumer requests a SaaS the cloud provider assumes most of the responsibilities in managing and controlling the underlying applications and infrastructure. In the same way, when the cloud consumer requires IaaS, the physical hardware and cloud software that enables the provisioning of IaaS are supervised by the cloud provider, *such as*, the physical servers, network equipments, storage devices and hypervisors for virtualization. Depending on the contracted service, the cloud consumer can choose from a variety of solutions, *whether* from top level software applications or from lower level hardware resources. The SPI service model, as seen in Figure [2], is described in the following Infrastructure as a Service, Platform as a Service and Software as a Service subsections (Kourik, 2011).

Figure 2 The SPI service model, The Cloud Computing Stack (Talbot, 2014)

2.4.1 Infrastructure as a Service (IaaS)

This type of service provides on-demand virtualized resources, such as virtual computers, virtual storage, Virtual Network and other hardware assets as services to cloud consumer or system administrator. The service consumer gets access to the contracted virtual machines, network accessible storage and network infrastructure components, such as firewalls and configuration services.

The IaaS cloud provider manages all the physical infrastructure hardware and has administrative control over the resource abstraction and control layer that wraps the Virtual Machine Monitor (VMM), also known as Hypervisor, host OS and other necessary tools to implement the desired service. The consumer, on the other hand, can make requests to the hypervisor through the interface provided by the service vendor to create and manage new Virtual Machines (VMs), choose the guest OS to be run on the VM and control all the middle-ware and applications that run on top of the guest OS.

The service usage fees of IaaS are calculated typically with consideration of Central Processing Unit (CPU) per hour, Giga Byte (GB) stored per hour, network bandwidth consumed and other added features have used (*e.g.*, monitoring and automatic scaling). The Amazon Web Services is the largest example of IaaS providers (Bhardwaj et al., 2010, So, 2011).

2.4.2 Platform as a Service (PaaS)

In a platform as a service model, the PaaS provider is the one who responsible for the development, deployment and management process of the PaaS by providing tools to the PaaS consumers/developers such as application programming interfaces, Integrated Development Environments (IDE), development version of cloud software, Software Development Kits (SDK) as well as deployment and management tools.

In comparison to the IaaS model this service offers a higher level of abstraction to the cloud consumer. The PaaS provider manages the cloud infrastructure, the operating system and the enabling software whereas the PaaS consumers are responsible for installing and managing the applications that are deployed, to comply with the programming language requirements and distribution as well as with the payment terms (Mell and Grance, 2011).

In this service model cloud consumers have limits or no access to the underlying infrastructure platform, *i.e.*, network, servers, operating systems, or storage. The billing procedures are implemented according to, processing, database storage and network resources consumed by the developed application as well as the duration of the platform usage. The Google AppEngine is an example of a PaaS provider (Zhang et al., 2010).

2.4.3 Software as a Service (SaaS)

The software as a service model resides on the top of the SPI stack. The service provides applications are accessible to the cloud consumers. In comparison with the other two underlying models (PaaS and IaaS), the SaaS is the most abstract service solution from a consumer's perspective.

The SaaS provider is responsible for the underlying infrastructure, the platform where the applications are located and for the installation and maintenance of the provided applications (Bhardwaj et al., 2010). On the other hand, the SaaS consumer is only responsible for the access, uploading, maintaining and interaction with his data. The SaaS can ease the burden with software licensing and maintenance. The applied billing system is based on the number of end users, the time of use, the network bandwidth consumed, the amount of data stored or duration of stored data (Mell and Grance, 2011). Examples of SaaS providers are Google Apps for Business and Salesforce productivity applications (Patidar et al., 2012).

2.5 Cloud computing Deployment Models

Clouds can be classified regarding the underlying infrastructure deployment model. The deployment model focuses on the architecture and the purpose of the cloud and refers to the location and management of its infrastructure. As previously mentioned, four different deployment models for Cloud Computing were identified by NIST (Mell and Grance, 2011) which are: private clouds, community clouds, public clouds and hybrid clouds. In this subchapter those four deployment models are explained in detail.

2.5.1 Private Clouds

The infrastructures of private cloud are intended for the exclusive use of a single organization with multiple consumers and emulate cloud computing on private networks (Mell and Grance, 2011). In this operating model the infrastructure could be owned, managed and operated by the organization "on premises" or by the cloud provider "off premises" with some contractual SLA or even a combination of both scenarios. The on premises and off premises variants can be separated in: on-site private clouds and outsourced private clouds. Regarding the physical location, these two models affect differently the security perimeter of the computation resources. The on-site private cloud is presented in Figure [3], in this scenario the private cloud could be secured and controlled inside the organization by its IT administrators. Clients can access the cloud from within the security perimeter or outside the security perimeter through

a boundary controller composed by firewalls and Virtual Private Network (VPN) connections (Bohn et al., 2011).

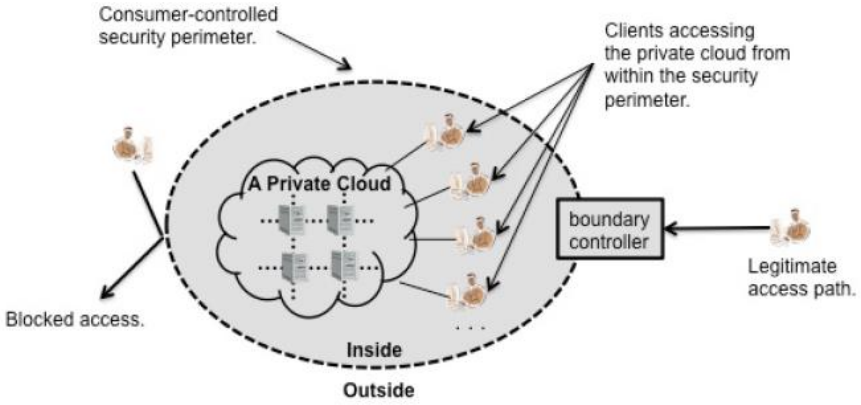

Figure 3 On-site (on premises) private cloud representation (Bohn et al., 2011)

The outsourced private cloud scenario represents an off premises type of private cloud. The cloud provider, sells its infrastructure services to the cloud consumer which could be the organization itself. In this case, the physical infrastructure belongs to the cloud provider, which is responsible for its management, but the provided service must follow the instructions issued by the contractual SLA between the two parties regarding security, management, privacy and other policies that could be concerned by the cloud consumer. Thus, as presented in Figure [4], the outsourced private cloud has two security perimeters that are connected by a protected communication link: one implemented by a cloud consumer "on the right" and the other implemented by the cloud provider "on the left".

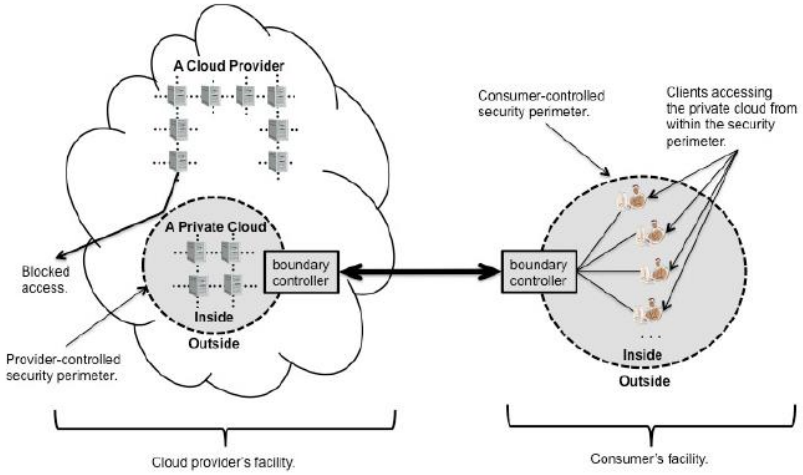

Figure 4 Outsourced (off premises) private cloud representation (Bohn et al., 2011)

The cloud provider is responsible to enforce his security perimeter and to separate the private cloud resources from the other cloud resources that are located outside the security perimeter. Depending on the consumer's security requirements, different mechanisms such as Virtual Local Area Network (VLAN), VPN, separate network segments or clusters can be applied. The

cloud consumer can manage the access to the private cloud resources within his security perimeter (Badger et al., 2011).

2.5.2 Community Clouds

The community cloud deployment model is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns, such as mission, security requirements, policy, and compliance considerations (Mell and Grance, 2011).
The use of a community cloud can be helpful by reducing the costs that could be paid by the organizations as compared to the implementation of individual private clouds, because the costs are supported and shared by a larger group. Similar to the private model of the cloud. In an on premises "on-site" scenario the infrastructure can be owned, managed and operated within the community by one or more of its organizations, while in an off premises outsourced scenario outside the community is owned, managed and operated by a third party such as cloud provider (Badger et al., 2011).
In terms of an on-site community cloud, Figure [5] shows that the represented community is composed by a set of participant organizations. Each one of them may provide cloud infrastructure services, consume cloud services or both (Badger et al., 2011).

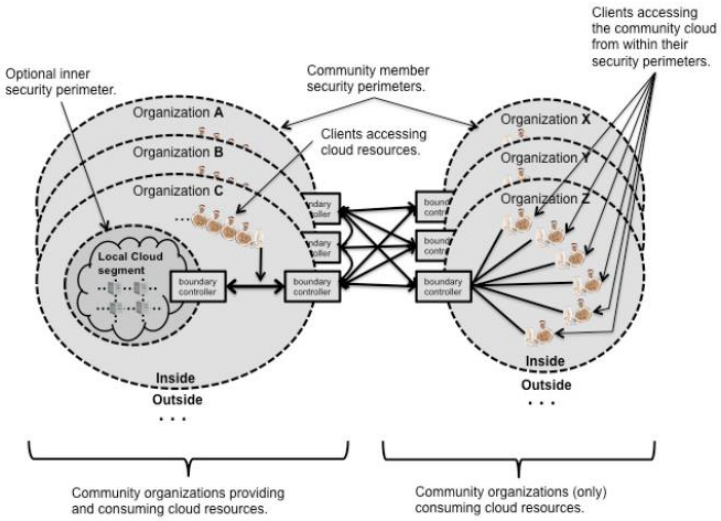

Figure 5 On-site (on premises) community cloud representation (Bohn et al., 2011)

In Figure [5], the organizations providing services are on the left and the ones that consume cloud services are on the right. The inner organizations networks are separated from the outside networks by a security perimeter like in the on-site private cloud scenario. The communication inside the community, i.e., between the organizations that provide services and the ones that consume them, are made through the boundary controller of each organization. Optionally, organizations can protect the cloud infrastructure from other computation resources with another layer of security perimeter.
On the outsourced community cloud, the organizations of a community only consume cloud resources. While in Figure [6], the infrastructure of the cloud is provided by a third party with

conditions identical to the ones described in the outsourced private cloud scenario, for example, the provided service is specified trough SLA agreements. The community organizations access the cloud resources inside the security perimeter through the boundary controller and the communication between the provider and consumers is done through a security link between the cloud provider and the community's organizations (Badger et al., 2011).

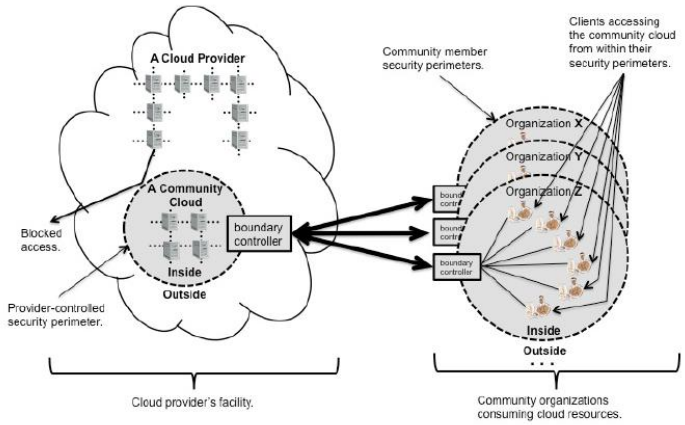

Figure 6 Outsourced (off premises) community cloud representation (Bohn et al., 2011)

2.5.3 Public Clouds

The public cloud model describes the cloud computing in a traditional mainstream sense whereby resources are dynamically provisioned on a self-service basis over the Internet. The cloud infrastructure that may be owned, managed and operated by a business, academic or governmental organization is intended for open use by the general public (Mell and Grance, 2011).

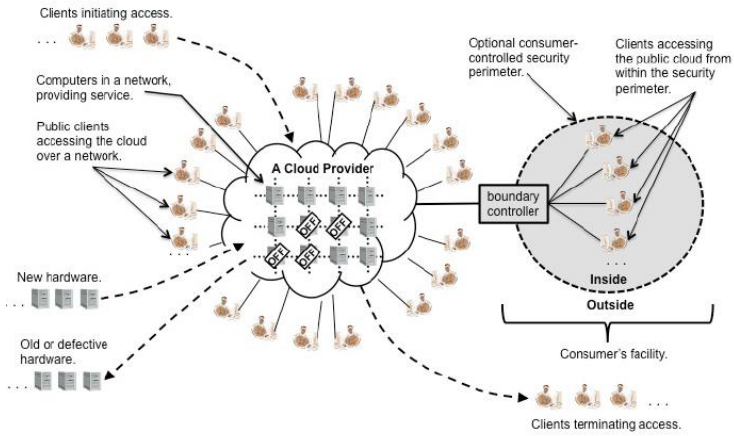

Figure 7 Public cloud representation (Bohn et al., 2011)

In the public cloud model, security system management, infrastructure maintenance and other operation issues are relegated to the cloud provider as described in Figure [7]. Compared to the

private cloud model, the service offered in the public cloud has a low degree of control and oversight of the physical and logical security aspects such as security perimeter used to separate computational resources (Badger et al., 2011).

2.5.4 Hybrid Clouds

Any composition of private, community or public clouds forms a hybrid cloud as represented in Figure [8]. The individual clouds remain unique entities and, so, a hybrid cloud can change over time with constituent clouds joining or leaving (Mell and Grance, 2011).

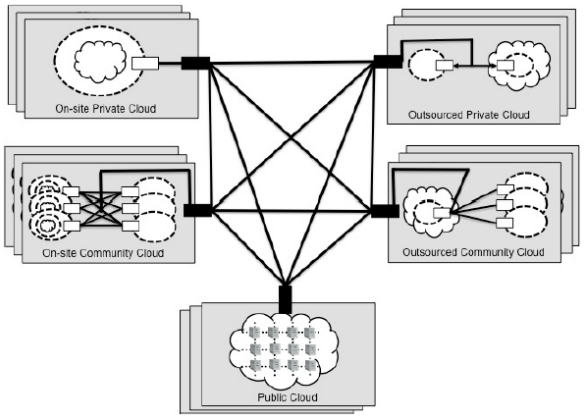

Figure 8 Hybrid cloud representation (Bohn et al., 2011)

The aggregation of clouds bound together by standardized or proprietary technologies that enable data and application to be exchanged among the clouds. Some of the benefits of this complex model are the access to external clouds during periods of high demand, to perform or to provide backup resources or to run non-core applications in a public cloud while maintaining core applications and sensitive data in an on premises private cloud (Badger et al., 2011).

2.6 Cloud Computing Challenges, Characteristics and Benefits

Every emerged technology such as cloud computing has strength points that provide the power to spread fast. Although these positive points, the new technology may have some vulnerabilities and challenges need to be solved.

2.6.1 Cloud computing Challenges

Although cloud computing has several benefits, there are also significant challenges. Generally, the advantages of cloud computing are more compelling to Small and Medium sized Enterprises (SME) than to large organizations. Larger organizations can easily create and maintain IT teams

and infrastructures as well as develop custom software solutions in the way to meet their own needs. On the other hand, the adoption of cloud computing implies application customization, control delegation and, most importantly, cloud provider dependency. Outsourced services evade the physical, logical and personal control of an organization IT staff and data outside a company's control security perimeter pose an intrinsic risk threat. These concerns can make corporate executives hesitate to migrate to cloud computing (Wei and Blake, 2010).
Three of the most significant barriers to the adoption of cloud computing are: (i) unpredictable performance; (ii) security risks and (iii) data privacy.

- **Unpredictable Performance**

Cloud consumers have certain expectations regarding the service level provided by the cloud. Some of these expectations are availability of service and overall performance. Due to the abstracting nature of the cloud provided services, the cloud consumer does not care how or where the contracted service runs on top of the physical infrastructure, the number of physical machines used or where they are. On the other hand, the economic benefits of cloud computing are based on the ability to increase the usage level of the infrastructure through multi-tenancy. Multi-tenancy introduces service under-performances, since it cannot ensure that the activities of different consumers are totally independent and disjoint. The cloud provider's challenge is, thus, to efficiently manage all the virtualized resources so that service performance remains as unaffected as possible by the number of consumer cloud resource demands (Kuo, 2011).

Physical resources such as CPU cores, disk space and network bandwidth must be divided and shared among virtual machines, running potentially heterogeneous workloads. The mapping between resources and corresponding virtual machines becomes increasingly complex to ensure the profitability of the physical interface. The VM mapping is dynamic and provides mechanisms to suspend, migrate and resume virtual machines by pre-empting low-priority allocations in favor of high-priority ones. These operations combined with load balancing, backup and recovery mechanisms cause performance variation.

Other important factors are external, non-controllable network related parameters such as latency, introduced by the communication link to the data center where the servers are hosted, as well as competing traffic in the communications link.

Cloud consumers expect a fixed performance/fee ratio, but this cannot be guaranteed as the performance may fluctuate. Furthermore, the cloud end user is unable to predict these variations, their magnitude and duration because the service measurements are made by the cloud provider services and not at the consumer premises. In these cases, the SLA is used as a warranty to compensate the cloud under-performance (Wei and Blake, 2010).

- **Security Risks**

Security is one of the main problems for enterprises to move their "in-house" data to public clouds or outsource private clouds. Most of the cloud providers do not guarantee the security of data while being transferred to the cloud (Hashizume et al., 2013). On the other hand,

because cloud computing represents a relatively new computing model, there is a great deal of uncertainty regarding how to provide security at the network, host, application and data levels. Compared to traditional technologies, the cloud has many specific features that make the traditional security mechanisms (*e.g.*, identity, authentication and authorization) insufficient (Li and Ping, 2009). Additionally, the cloud service models are interdependent since, on one hand, PaaS and SaaS are hosted on top of IaaS and, on the other hand, PaaS provides the platform to SaaS. Due to these dependencies, any attack to a cloud service layer can compromise the upper layers, for example a breach in the infrastructure service will impact the security of both PaaS and SaaS.

Relationships and dependencies between cloud models can also be a source of security risks. One SaaS provider may rent a development environment from a PaaS provider, which might also rent an infrastructure from other IaaS provider. This may result in an inconsistent combination of security models because each provider implements its own security model.

There are security issues in all three SPI model layers: (*i*) SaaS applications suffer from vulnerabilities and accessibility issues; (*ii*) PaaS, which depends on third-party relationships and development life cycle, inherits security issues; and (*iii*) IaaS experiences virtualization vulnerabilities and malicious VM images. Some of the countermeasures used to minimize security threats are the implementation of dynamic credentials, digital signatures, data encryption, frameworks to secure virtual networks and hypervisor data flow inspection (Hashizume et al., 2013).

- **Data Privacy**

Privacy issues are increasingly important in the World Wide Web (WWW). The delegation of data control to the cloud provider when utilizing a cloud service makes cloud consumers uncomfortable because of the risks to data integrity, confidentiality and privacy principles as data becomes accessible to an augmented number of parties.

Security breaches like the ones described in the security section can lead to private data access and exposure, but data confidentiality can also be breached unintentionally through, *e.g.*, data remanence. Data remanence is the residual representation of data that have been erased or removed. Due to the virtual separation of logical drivers and lack of hardware separation between multiple users on a single infrastructure, the remaining residual information stored in the physical storage units may lead to the unwilling disclosure of private data or to the intentional exploitation of that vulnerability.

Another fact that concerns enterprises regarding data privacy protection is the geographical distribution of the information inside the clouds. The fault tolerance and backup mechanisms that cloud providers utilize to prevent system failures as well as to provide data storage resources to fulfil cloud consumers space requirements lead to data fragmentation across geographic locations. The challenge with this distribution is that information can be stored in different countries with diverse data protection laws and, thus, cloud providers cannot ensure personal data requirements against the countries legal framework (JAMES, 2010).

The solution to this problem involves the creation of initiatives such as the Madrid International Conference (Conference, 2009) with the aim to create international agreements where a set of principles and rights guaranteeing the effective and internationally uniform protection of

privacy with regard to the processing of personal data between different countries to facilitate the international flow of personal information and ease data privacy concerns.
On the other hand, the adoption of proactive-measures requirements can be met through the implementation of Privacy Enhancing Technologies (PET)1. PET safeguards the individual data privacy and rights by protecting personal data and preventing its unnecessary or undesired processing.

2.6.2 Cloud computing Characteristics

Due to the lack of standardization and consensus regarding cloud computing, the characteristics that define the cloud vary between all interested parties (Jadeja and Modi, 2012). In order to promote simplification and standardization, the NIST identifies five essential and generally accepted main characteristics to describe the behavior of cloud computing (Mell and Grance, 2011): (i) on-demand self-service; (ii) broad network access; (iii) resource pooling; (iv) scalability; and (v) measured service. These cloud characteristics will be described in the following subsections.

- **On-demand Self-service**

The cloud consumer can use resources of the contracted cloud computing service such as system and network resources on demand without human interaction between consumer and cloud provider in an on-demand form. The resources can be controlled through a user-friendly self-service interface, for example, a Web page dashboard or Command Line Interface (CLI) console, providing effective means to manage the service offerings and promoting cost savings to both consumer and cloud provider. This cloud feature is important since it solves the resource demand fluctuation problem and reduces the costs by avoiding planning ahead the installation of additional resources to meet peak demands and underusing installed resources.

- **Broad Network Access**

The access to resources hosted in the cloud is available over private networks, *e.g.*, private cloud and community cloud implementations, or over the Internet, *e.g.*, public clouds, through standard mechanisms. Therefore, the broad network access provides independence platform. Thus, the services can be accessed from all types of operating systems and different types of computation platforms, *e.g.*, desktops, tablets or smart phones. This feature provides access to resources (mainly in the public cloud scenario) from a wide range of locations as long as they have Internet access.

- **Resource Pooling**

The resources that pooled together by the cloud provider are assigned and reassigned to serve multiple consumers in a multi-tenant model. These physical and virtual systems are dynamically allocated or reallocated as needed. The resources can be physically located at many geographic locations and assigned as virtual components of the service when requested. Thus, that could

create, as defined by NIST (Mell and Grance, 2011), “a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources”.

- **Rapid Elasticity**

Rapid elasticity refers to the ability of the cloud to expand or reduce provisioned resources quickly and efficiently, fulfilling the requirements of the on-demand, self-service, characteristic of cloud computing. Through the broad network access diverse computing infrastructures can cooperate to allocate these resources (*e.g.*, federation of clouds in a hybrid cloud or a community cloud scenario), manually or automatically, appearing to the consumer as a large, limitless pool of dynamic resources that can be purchased at any time and in any quantity. This feature is normally negotiated between cloud providers and cloud consumers through a SLA. In this type of agreement, the provider negotiates with the client the time response for the provision of the resources (as they are in fact finite) in order to respond to the customer’s instant demands as well as the penalties for not meeting the negotiated contract terms (in this case the accorded time response).

- **Measured Service**

Due to the service oriented characteristics of cloud computing, the amount of cloud resources used by the cloud consumer are monitored, measured and reported as they are allocated or consumed. This procedure is important for quality of service requirements, billing procedures and to provide transparency for both the cloud provider and the cloud consumer. As previously mentioned, cloud computing uses a utility computing model in which the consumer is charged based on the level of service provided. For that purpose, known indicators (metrics) such as amount of storage, network resources or level of processing power are used to measure the amount of consumed resources. Normally, this information is provided automatically via the consumer self-service interface.

2.6.3 Cloud computing Benefits

The following are some of the possible benefits of offering cloud computing-based services and applications (Dialogic, 2010):

- **Cost Savings** — Companies can reduce their capital expenditures and use operational expenditures for increasing their computing capabilities. This will give a lower barrier to entry and also requires fewer in-house IT resources to provide system support.
- **Scalability/Flexibility** — Companies can start with a small deployment and grow to a large deployment fairly rapidly, and then scale back if necessary. Also, the flexibility of cloud computing allows companies to use extra resources at peak times, enabling them to satisfy consumer demands.
- **Reliability** — Services using multiple redundant sites can support business continuity and disaster recovery.

- **Maintenance** — Cloud service providers do the system maintenance, and access is through APIs that do not require application installations onto PCs, thus further reducing maintenance requirements.
- **Mobile Accessible** — Mobile workers have increased productivity due to systems accessible in an infrastructure available from anywhere.

2.7 Cloud Computing Business Perspective

An important factor that differentiates cloud computing from the other computing technologies is its ability to converge existing technologies with a utility model making the cloud flexible to the adoption of different business perspectives. This approach enables IT efficiency, whereby the power of modern computers is utilized more efficiently through highly scalable hardware and software resources, as well as business agility, whereby IT can be used as a competitive tool through rapid deployment, parallel batch processing, use of computing-intense business analysis and mobile interactive applications that respond in real time to user requirements. Thus, cloud attractiveness increases as organizations discover that their substantial capital investments in information technology are often grossly underutilized and that the maintenance and service costs do not pay (Marston et al., 2011). On the other hand, the adoption of a utility model enables cloud computing services to reuse the traditional electricity, water, telecommunications price models, including the pay as you go model or metering model, the subscription model and the combination of both.

This section will describe the existing stakeholders and the contracted SLA between service vendors and service consumers in the cloud.

2.7.1 Stakeholders

In the previous computing paradigms, the main stakeholders were the system providers and their consumers. Whereas the providers were responsible for the sales, installation, licensing, consulting and maintenance of the involved technologies, the consumers used, maintained and owned the provided systems. With the technological shift to the cloud, some of the previously described functions of these stakeholders changed and due to the outsource characteristics of the cloud deployment services two new important parties must be considered in addition to the traditional stakeholders: the cloud enablers and the cloud regulators (Marston et al., 2011).

In cloud computing the consumers are effectively subscribers that, in comparison with the previous computing technologies, only purchase from the cloud providers the use of the system on an operational expense basis. The cloud providers own and operate the cloud computing systems in order to deliver third party services. They provide the infrastructure, systems and software, perform the corresponding maintenance and implement the pricing of the cloud services.

Cloud enablers are organizations that sell products or services that facilitate the delivery, adoption, use and management of the cloud infrastructures. This type of stakeholder is an

intermediary class that stands between the cloud provider and the service consumer, taking advantage of the currently lake of core competencies from the infrastructure providers in the interaction with the customers. One of cloud enabler examples is RightScale (Rightscale). The cloud regulators, on the other hand, regulate all aspects of the cloud computing value-chain and are, typically, sovereign government bodies and international entities. Their main functions are the provision of regulations regarding data, residency, privacy and other related cloud computing features, harmonization of a cross border regulations and promotion of inter-governance cooperation in the adoption and formulation of new cloud computing regulations.

2.7.2 Service Life-Cycle

A service life-cycle describes the different phases of a particular service provided by a cloud vendor and includes the service delivery systems necessary to meet the QoS (Quality of Service) objectives specified in SLA, i.e., all stages of the SLA life-cycle. The SLA life-cycle is composed of four stages:

1. SLA template design - The service provider defines the types of SLA he is willing to propose in order to ensure that the agreed QoS guarantees are realistic;
2. SLA negotiation - The service provider and customer attempt to define the terms of the SLA that will bind their business relationship, *i.e.*, that the agreed QoS guarantees are realizable and that the end-to-end QoS requirements are satisfied;
3. SLA runtime - The service provider and customer verify that the agreed QoS guarantees are satisfied;
4. SLA (template) archiving - The established SLA are stored for future reuse or learning.

On the other hand, the overall life-cycle, represented in Figure 9, consists of the following stages:

1. Design and development - consists on the development of artifacts needed for the service implementation;
2. Service offering - includes the SLA template design, where to offer the intended service and results in the specification of SLA templates;
3. Service negotiation - includes parts of the SLA negotiation, represents the actual negotiation between customers and provider and results in an agreed SLA;
4. Service provisioning - covers parts of the SLA negotiation, represents all activities required in system preparation and set-up for service operation, including booking, deployment (if needed) and configuration;
5. Service operations - includes SLA runtime, *i.e.*, it is a stage where the actual service instance is up and running and where adjustments can be made to enforce a SLA;
6. Service decommissioning - represents the end of a service instance (the customer can no longer access it) and corresponds to the stage where the SLA (template) archiving is done.

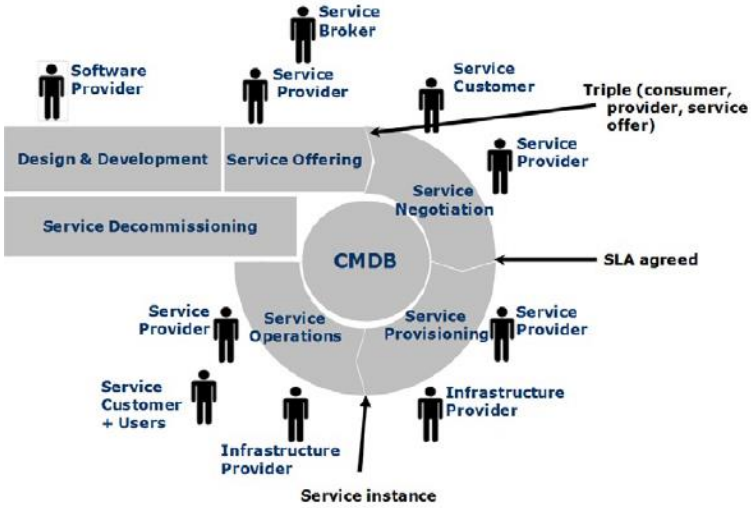

Figure 9 Service life-cycle (Wieder et al., 2011)

2.7.3 Service Level Agreements

When consumers adopt cloud computing solutions, the quality and reliability of the provided services are essential features. As previously described in the Concerns section, it is impossible to fulfil all consumer expectations from the service provider perspective and, hence, a balance needs to be achieved via a negotiation process. At the end of the negotiation, provider and consumer commit to an agreement. In a Service Oriented Architecture (SOA) this agreement is referred to as a Service Level Agreement (Wieder et al., 2011).

The establishment of SLA is frequently adopted in the software and telecommunications domain to specify a mutual understanding regarding the transactions between providers and consumers. Typically, a SLA is a bilateral binding statement signed between a service provider and a service consumer stating the agreed terms and conditions of the given service (Stanoevska-Slabeva and Wozniak, 2010). Additionally, it also defines the remedial actions and the penalties if performance falls below the agreed standard.

Ideally, SLA can be negotiated between any type of service providers and consumers but, in reality, in cloud-based business scenarios, SLA are usually not negotiated in Business to Consumer (B2C) and in Business to Business (B2B) transactions involving small clients (*e.g.*, SME) (Gangadharan and Parrilli, 2011).

A SLA acts as a legally enforceable document traditionally composed of:(*i*) subject terms; (*ii*) scope of rights; (*iii*) financial terms; (*iv*) representation; (*v*) service credits, credit requests and compensation procedures; (*vi*) evolution and support terms; (*vii*) warranty; (*viii*) indemnification; and (*ix*) limitation of liability. Although a standard cloud SLA model does not exist, the subject has been studied by the European Telecommunications Standards Institute (ETSI) (ETSI, 2012).

In the Web service domain there are two main specifications for describing a SLA: (*i*) Web Service Agreement (WS-Agreement) from the Open Grid Forum (OGF) (Andrieux et al., 2007); and (*ii*) Web Service Level Agreement (WSLA) language and framework from IBM (Keller and Ludwig, 2003).

The WSLA was proposed in 2001 and it allows the creation of machine-readable SLA for services implemented using Web services technology that define service interfaces in the Web Services Description Language (WSDL). This framework comprises several monitoring services that can be replaced with services developed by third parties. An SLA created using the WSLA language is normally formed the following sections:

1. Definition of the parties - contains the description of the service provider, the service consumer and, eventually, the third parties involved (when commission part of the SLA management responsibility to additional supporting parties). The information provided about each party should include contact information and a description of the organization as well as the name of a WSDL file that contains the interface description of the Web services the party implements;
2. Definition of the services and their operations - contains the description of the service provider interfaces. The services are represented by service objects and each service object is associated with one or more SLA parameters. This section specifies the relevant service level parameters and respective metrics as well as indicates which party is responsible for measuring each metric and how the measuring should be done;
3. Obligations of the agreement - specifies the conditions (*e.g.*, the average operation response time shall not exceed a predefined value) and the action guarantees, *i.e.*, the actions the parties commit to perform in a given situation.

On the other hand, the WS-Agreement specification appeared in 2007 and defines an XML-based language for agreements as well as a protocol for advertising the capabilities of service providers, creating agreements between service consumers and providers and monitoring agreement compliance (Andrieux et al., 2007). WSAgreement also provides an eXtensible Mark-up Language (XML) schema that defines the overall structure of an agreement document and, in addition to the WSLA, defines a protocol for negotiating and establishing agreements dynamically based on Web services. The main differences to the WSLA are the use of: (*i*) SLA templates that embody all the customization aspects of an agreement; (*ii*) an extensible XML structure containing several sections where the intended users are expected to define domain-specific elements and properties; and (*iii*) metrics defined by a domain-specific extension rather than associated with the parameters used in the agreement. The XML document of the WS-Agreement consists of the following parts:

- IDentification (ID) - holds a unique mandatory agreement ID followed by an optional name;
- Parties - contains the identification of the different parties, various metadata about the agreement (such as an expiration time and template ID in case the agreement was created following a template) and user-defined attributes;

- Interface declarations - defines all the information about the functional capabilities of the service. It encapsulates the information contained in traditional service description (*e.g.*, WSDL documents);
- Agreement terms - specifies the service terms and guarantee terms. Service terms identify and describe the services that are the subject of the agreement and also comprise the measurable service properties and exposed properties that can be used to express service level objectives. Guarantee terms specify the QoS that the parties are agreeing to, *e.g.*, minimum availability of a service, the number and type of CPU that the service provider should make available or the average response time for requests.

2.8 Moving towards Software-Defining Solutions

The demand for reducing capital expenditures (CAPEX) and operating expenditures (OPEX) has pushed information technology (IT) specialists toward contemplating designs to achieve more effective capital investments with higher return on capital. Toward this goal, the virtualization technology has emerged as a way to decouple software applications from the underlying hardware and enable software to run in a virtualized environment. In a virtual environment, hardware is emulated, and the operating system (OS) runs over the emulated hardware as if it is running on its own bare metal resources. Using this procedure, multiple virtual machines can share available resources and run simultaneously on a single physical machine (Sharkh et al., 2013).

The following section presents a comprehensive overview about the emerging technologies with their software-based solutions such as: Virtualization, Hypervisor technology, Software-defined networking (SDN) and network functions virtualization (NFV). They each offer a new way to design, deploy and manage the network and its services.

2.8.1 Virtualization

Virtualization is a key technology that enables part of the most important characteristics of cloud computing (*e.g.*, resource pooling and rapid elasticity). This technology has been proposed and developed over a relatively long period by International Business Machines (IBM) in 1960-1970 (Magazine, 2012, Oracle, 2011, Antonopoulos and Gillam, 2010). Its purpose in cloud computing is to divide and abstract the resources of the physical infrastructure into multiple segregated virtual systems, virtual machines with virtual networks and virtual storage, through which the cloud services are provided.

"A virtual machine is taken to be an efficient, isolated duplicate of the real machine." (Popek and Goldberg, 1974). It has its own address space memory, processor resource allocation and device Input/Output (I/O) trough virtual device drivers. These virtual systems are independent from each other and provide a complete platform for support and execution of an operating system. Normally, an OS running on a virtual machine is called a Guest OS and the OS that runs on top of the physical hardware is called Host OS. The VM lifetime has six phases: create, suspend, resume, save, migrate and destroy. Multiple virtual machines can run

simultaneously on the same physical infrastructure node (*i.e.*, physical server) and each VM, residing in the same computing node, can execute different Guest OS.

2.8.1.1 The Architecture

Three types of architectures are existed, full virtualization, para-virtualization and OS-level virtualization. They are described in detail in the following part:

- **Full Virtualization**

Full virtualization was first introduced by IBM with the Virtual Machine Facilityl370 (VM/370) operating system (Creasy, 1981). This virtualization method, illustrated on Figure 10, emulates a full hardware environment of a computing node, through binary code translation/rewriting, into independent virtual machines created on top of the hypervisor layer. The binary translations are used to adapt the non-virtualizable instructions into virtualized ones. More recently, the full virtualization method was enhanced with the appearance of the hardware virtualization.

Hardware virtualization provide mechanisms to run unmodified guest virtual machines without the overheads inherent in the standard full virtualization method. It started to be supported by Intel and Advanced Micro Devices (AMD) x86 architecture processor families through the Intel VT-X and AMD Virtualization (AMD-V) hardware assisting virtualization technologies (Intel, AMD).

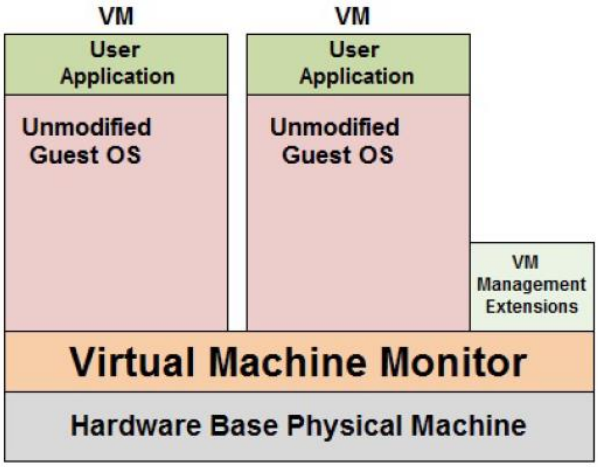

Figure 10 Full virtualization (Antonopoulos and Gillam, 2010)

Since the VM emulates the full system, the guest operating system does not need to be modified, meaning that both open-source and proprietary OS work with this type of virtualization methodology. The guest OS kernel is not aware of the virtualization and communicates directly to the hypervisor layer as if it is the hardware base physical infrastructure. In this approach, the hypervisor has to manage, control and map the requests from all guest OS to the limited amount of physical resources. As a consequence, the performance experienced is lower when compared to native OS. On the other hand, the actual quantity of hardware resources, provided by the computing node, define the limits to the creation of simultaneous VM. Two examples of known commercial and open-source software

tools for server virtualization that implement full virtualization are: VMware and Kernel-based Virtual Machine “KVM” (VMware, KVM).

- **Para-virtualization**

In para-virtualization the guest OS kernel is modified to become aware of the hypervisor as illustrated on Figure 11. In this virtualization methodology, OS-level information about the virtual machines can be passed explicitly from the guest OS to the hypervisor without the necessity to trap or translate every OS call.

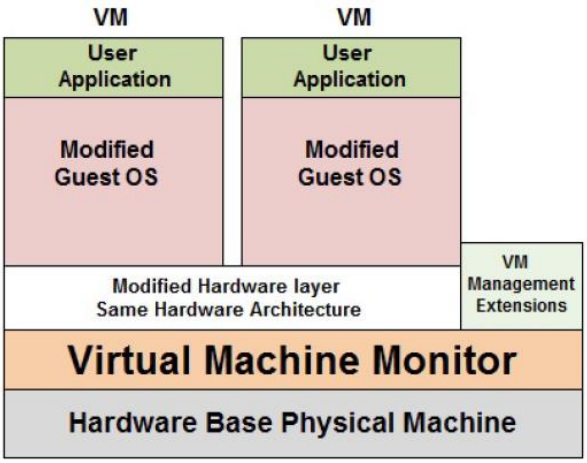

Figure 11 Para-virtualization (Antonopoulos and Gillam, 2010)

Due to the described behavior, the para-virtualization approach results in lower virtualization overhead than full virtualization, providing close to native guest OS performance. However, this methodology is less flexible than full virtualization because it requires OS-level modifications, which is a characteristic non implementable with legacy and close-sourced commercial OS. An open-source example of a software tool for server virtualization capable of para-virtualization implementation is Xen (Xen).

- **OS-level Virtualization**

The operating system level virtualization differs from the other two virtualization methodologies by utilizing the kernel of a host OS embedded with a virtualization layer. This approach allows the creation of multiple user-space instances known as containers, Virtual Private Server (VPS) or Virtual Environment (VE) as illustrated on Figure 12. Since normal OS calls are used and no emulation is performed, the container performance is practically native with little imposed overhead. However, this methodology is not as flexible as the other two virtualization approaches since it cannot use guest OS that are different from the host OS, as the OS kernel is shared by the host OS and all guest OS containers. Examples of software tools capable of OS level virtualization implementation are OpenVZ and Parallels Virtuozzo Containers (OpenVZ, Parallels).

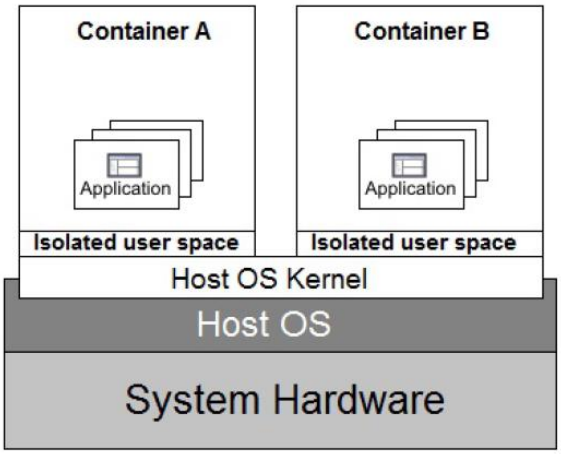

Figure 12 OS level virtualization

2.8.1.2 Advantages/Disadvantages

Beside of the architecture and the types of the virtualization, the IT market needs to know if this emerging technology has its own position in the future. For that reason some of advantages and disadvantages of virtualization are mentioned (NashNetworksInc):

Advantages:

- **Efficient use of hardware**. The typical conventional server runs only one major application (e.g. Exchange) and might be using only 10-15% of its processing capacity. Virtualized hardware works hard and is more cost-effective, particularly if it is running many virtual servers.
- **Better and cheaper backup and disaster recovery**. A virtual computer can be backed up very easily as an "image" - unlike conventional backups, which require complex and often expensive software. The images can be restored onto dissimilar hardware, unlike conventional backups, which often can't. The image can be brought up almost instantly. This provides a working service while the primary computer is restored, and dramatically reduces downtime. It's also very fast and easy to roll back a computer to a previous version, instead of having to fix it in, say, the case of a virus infection.
- **Better management**. Virtual computers can be centrally and efficiently managed. Central administration and "locked-down" desktop environments dramatically reduces support costs. Software can be installed and updated quickly and automatically across a virtual network. Virtual test machines can be created almost instantly so that testing can be done without any disruption of live machines.
- **Ability to set up and test new machines**. This can be done easily and with no disruption to live services.
- **Less network infrastructure**. Fewer physical machines mean less physical infrastructure – meaning lower purchase costs and less maintenance.

- **Lower power and cooling costs**. This is particularly relevant to data centers or organizations with many servers.
- **Lower co-location costs**. Fewer physical servers take up less space, slashing co-location costs. This has allowed the hosted application industry (e.g. hosted Exchange, hosted QuickBooks) to fly.
- **Better security**. Virtualization can provide centralized and secured computing environments. Desktop virtualization takes data off individual workstations and laptops, removing the risk of data loss through physical theft or loss of desktop machines.

Disadvantages:

- **Single point of failure**. If a virtualized server fails, the entire organization can be paralyzed. If a company uses virtualized desktops and the central hosting server goes down, users can literally do nothing on their desktops – compared with conventional desktops where users can keep functioning even if the network is down, and a single desktop failure doesn't impact anyone else. This is a significant downside and no virtualized system should be put in place without proper contingency planning.
- **More powerful hardware needed**. Virtualization uses fewer servers, and uses them better, but because they need to be more powerful, the reduction in total hardware costs may not be as dramatic as expected.
- **Greater demands on the network**. Network and bandwidth requirements will be greater. If the virtualized server is hosted remotely, adequate and redundant bandwidth are needed, and this can push up running costs.
- **Complexity**. Virtualization increases complexity on a computer, making it harder to manage and troubleshoot, particularly if it is not properly set up and documented. Without good automation tools, a virtual server can be almost impossible to manage.
- **Potential security problems**. Some sources feel that security can be more difficult to manage on a virtualized system. New forms of attack could target the virtualization software itself – though so far this has not happened.
- **Third-party support issues**. Some vendors may not be willing or able to support their software if it is running on a virtual server.
- **Lower tolerance for poor management**. Operators of virtualized systems have to manage and document virtual environments properly. If they don't, they could end up with a messy jumble of virtual machines that can be costly in terms of time and unnecessary licenses.

2.8.1.3 Hypervisor technology

To enable virtualization, a low-level program called Virtual Machine Monitor or Hypervisor is required to provide an environment for programs identical with the one of a physical machine. The hypervisor is in control of the system resources and provides them access to virtual machines and management functions regarding the existing VM on the computing node. The access to resources relies on load balancing techniques that are responsible for mapping logic

addresses to physical addresses and for the management of workloads. Depending on the installation of the virtualization layer, two types of hypervisors can be identified:

- Type 1 hypervisors: The virtualization layer containing the hypervisor is directly installed on top of the physical infrastructure node without the need for a host operating system. This architecture, also known as bare metal virtualization, enables direct access to the hardware resources;
- Type 2 hypervisor: The virtualization layer is installed on top of the host operating system rather than on top of the infrastructure hardware of the node. This architecture has no direct access to the hardware resources.

In Figure [13] a diagram of Type 1 and Type hypervisors is showed:

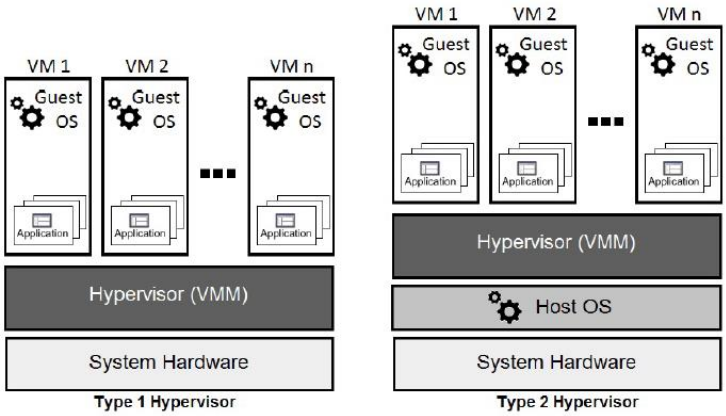

Figure 13 Type 1 and Type 2 hypervisors

Type 1 hypervisor architectures are more robust and, due to the close proximity to hardware resources, they can achieve higher virtualization efficiency than Type 2 hypervisors. Type 2 hypervisors are used mainly on client systems where efficiency is less critical as well as on systems where support for a broad range of I/O devices is important and can be provided by the host operating system.

There are also virtualization methods that are classified according to whether or not the Guest OS kernel needs to be modified, *e.g.*, Full virtualization and Para-virtualization, or whether the kernel of an operating system allows for multiple isolated user-space instances, instead of just one, *e.g.*, OS level virtualization.

2.8.1.4 OVS (Open Virtual Switch)

OpenvSwitch (OVS) is an open source switching stack for virtualization. It brings many features standard in hardware devices to virtualized environments. Basically OVS allows to create a new network access layer residing on hosts that connects the various VMs (Pfaff et al., 2009), some of OVS's features:

- A variety of tunneling protocols such as: VLAN, VxLAN and GRE in terms of security
- Traffic shaping to ensure the QoS (Quality of Service)
- OVS has an automated control feature through supporting OpenFlw protocol.

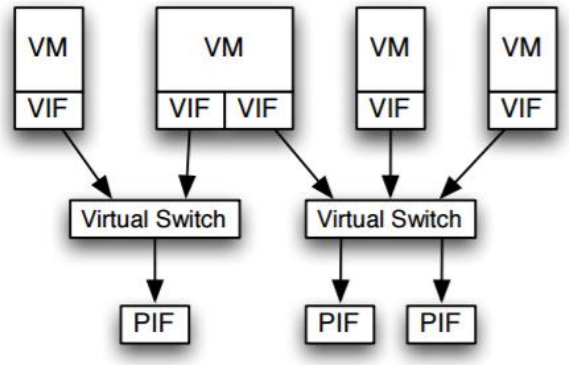

Figure 14 Openvswitch with Physical switch connection (Pfaff et al., 2009)

Virtual switches connect virtual interfaces (VIFs) to physical interfaces (PIFs). Each VM has at least one virtual network interface card (vNIC) and all vNICs are sharing physical network interface cards (pNICs) on the physical host through OpenvSwitch (Pfaff et al., 2009). OVS is widely deployed in IT enterprise, service provider, and Telecommunication environments.

2.8.1.5 DVR (Distributed Virtual Router)

Distributed Virtual Routers DVR is created to avoid single point of failure (SPOF) on network nodes. When using standard routers, all the traffic is passing out through Network servers. Inside network servers, router namespaces are created routing all traffic and NAT forwarding between instances and public networks. When a network node falls down, instance traffic will no longer be available until a new namespace is created and executed in another network node.

Distributed routers is a way to avoid the SPOF through network nodes. When using DVR, router namespaces, are directly created inside compute nodes where all instance and l3 traffic are routed.

The following two figures will give an idea about how the traffics go through the nodes before and after implementing DVR (Jack McCann et al., 2014).

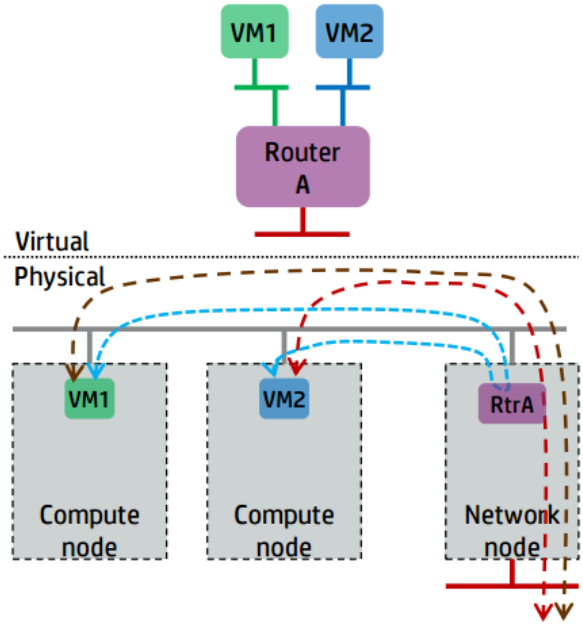

Figure 15 Legacy Routing (Jack McCann et al., 2014)

The previous diagram is representing the Legacy Routing in OpenStack environment, the network node is the core of networking here, all the traffic should pass through this node, eventually that will cause the SPOF and performance bottleneck.

In the other hand, by implementing DVR in each compute node, all the issue in the previous scenario will obviously decrease.

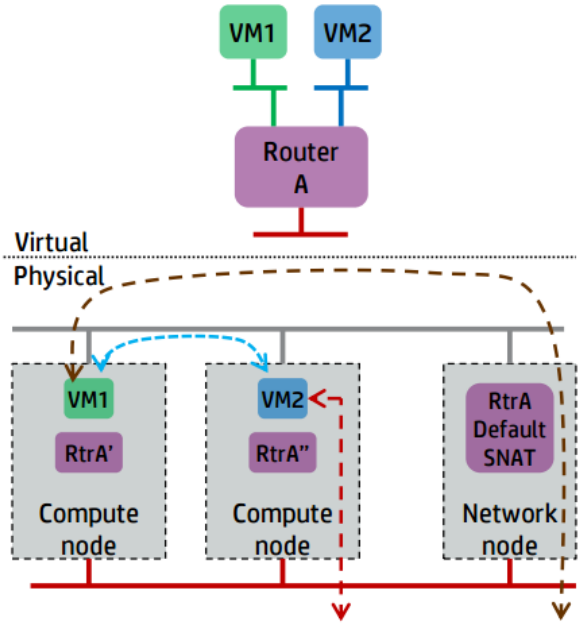

Figure 16 DVR Scenario (Jack McCann et al., 2014)

In DVR scenario, the compute node provides IP forwarding for local VMs which means that the connectivity between VMs from the same subnet is available without the need of network node. Moreover, every VM will get on flouting IP to connect external network, for example internet.

As a limitation of this mechanism the default SNAT function is still centralized in the network node which force some traffic to pass through.

2.8.2 SDN (Software Defined Network)

The aim of SDN is to provide open interfaces that enable the development of software that can control the connectivity provided by a set of network resources and the flow of network traffic though them, along with possible inspection and modification of traffic that may be performed in the network. These primitive functions may be abstracted into arbitrary network services, some of which may not be presently apparent (Foundation, 2014).

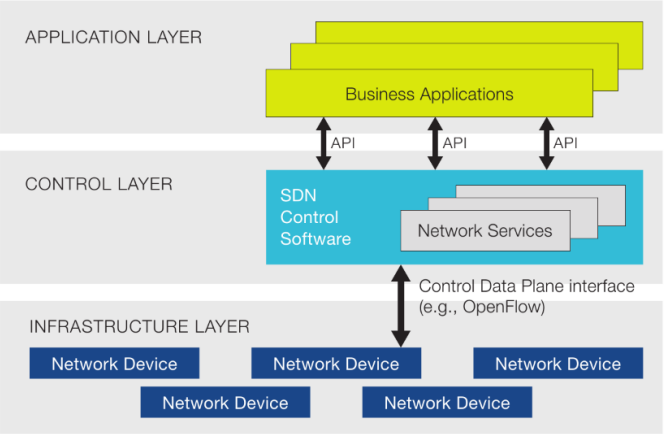

Figure 17 Basic SDN Architecture (ONF, 2012)

Figure [14] introduces the basic SDN components, with terminology similar to that from the original ONF white paper, “Software-Defined Networking: The New Norm for Networks” (ONF, 2012). The initial view comprised infrastructure, control and application layers (red text), which are designated in this architecture document as data, controller, and application planes (black text). The infrastructure layer (data plane, note) comprises network elements, which expose their capabilities toward the control layer (controller plane) via interfaces southbound from the controller. In (ONF, 2012), this is called a control-data plane interface. The SDN applications exist in the application layer (plane), and communicate their network requirements toward the controller plane via northbound interfaces, often called NBIs. In the middle, the SDN controller translates the applications’ requirements and exerts low-level control over the network elements, while providing relevant information up to the SDN applications. An SDN controller may orchestrate competing application demands for limited network resources according to policy.

Software-Defined Networking SDN main objectives are decoupling the network control and data functions, centralization of control and programmability of network (ONF, 2012).

In the non-SDN networks, there is no separation, each network device has data and control plane with local RIB (Routing Information Base) which is the gateway protocol system in the routing system management, and in addition the non-SDN networks use OSPF (Open Shortest

Path First) is a router protocol used to connect the networks, find the best path for packets, update the routing table which is running on each network device and exchange routes between devices to allow them to make out what the topology of the network looks like (Kamamura et al., 2015).

In this case, the network will miss the visibility and the only way to make some configurations is to use CLI (command line interface) to connect each device and individually configure that networking device manually.

As mentioned before, in the tradition network the data plane and the control plane are existed locally in each networking device, the data plane is the responsible for forwarding the traffic through the device. For example, in the figure [15] the traffic is going from port 1 and needs to be forwarded to port 2, this information policy is programmed in the local control plane in RIB.

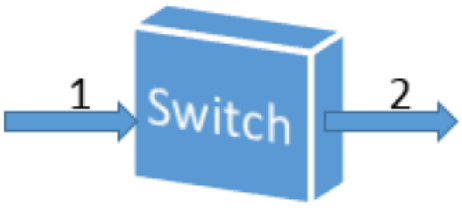

Figure 18 Forwarding traffic through switch

If there is a need to use a 100 switches/routers, then there will be 100 data plane, 100 control plane and 100 managed interfaces.

Moreover, the different vendors issue, there is no open interfaces on these devices, so making any change in the control plane is not allowed and the developer will be avoided to make any change because of the various property vendors.

In terms of applications development, the developer can create an app talks to abstraction layer like widows which hides the complexity of the hardware from the app developer, while that does not exist in networking nowadays because of the property reason.

All the changes will be made with open SDN and open flow. The idea is creating an open interface in networking devices and building an abstraction layer to allow the rapid app development.

2.8.2.1 The Architecture

The basic principles of the SDN Architecture are three-fold (ONF, 2014):

1. **Decoupling of controller and data planes.**
   This principle calls for separable controller and data planes. However, it is understood that control must necessarily be exercised within data plane systems. The controller plane interface (CPI) between SDN controller and network element is defined in such a way that the SDN controller can delegate significant functionality to network elements

(NEs), while remaining awareness of NE state. Clause 4.3.4 of [SDN ARCH] lists criteria for deciding what to delegate and what to retain in the SDN controller itself.

2. **Logically centralized control.**
   In comparison to local control, a centralized controller has a broader perspective of the resources under its control, and can potentially make better decisions about how to deploy them. Scalability is improved both by decoupling and centralizing control, allowing for increasingly global but less detailed views of network resources. SDN controllers may be recursively stacked for scaling or trust boundary reasons, a topic described in [SDN ARCH] clause 5.
3. **Exposure of abstract network resources and state to external applications**

Applications may exist at any level of abstraction or granularity.

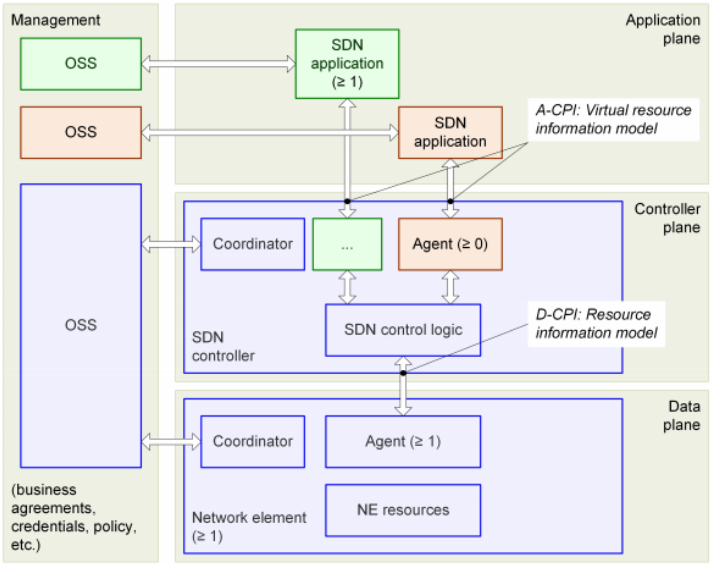

Figure 19 SDN Architecture Overview (ONF, 2014)

The SDN Architecture comprises three layers (ONF, 2014):

- The Data Plane comprises network elements, which expose their capabilities toward the control layer (Controller Plane) via the data-controller plane interface (D-CPI).
- In the Controller Plane, the SDN controller translates the applications' requirements and exerts more granular control over the network elements, while providing relevant information up to the SDN applications. Services are offered to applications via the application-controller plane interface (A-CPI, often called NBI) by way of an information model instance that is derived from the underlying resources, management-installed policy, and local or externally available support functions. An SDN controller may orchestrate competing application demands for limited network resources.
- SDN applications reside in the Application Plane, and communicate their network requirements toward the Controller Plane via the A-CPI.

2.8.2.2 Advantages/Disadvantages

Although SDN has advantages that make it one of most promised technologies in the future, it is still facing some obstacles. The following subsection presents some of these advantages and disadvantages of SDN (Serverwatch, 2015, Wiley, 2015).

Advantages:

Apart from the benefits highlighted below, the biggest advantage of adopting SDN is that it provides a company with the ability to model its physical networking environment into software, which, in turn, helps in reducing the overall CAPEX and OPEX. Here are some specific benefits SDN can offer:

Cost Reduction:
First, SDN does not require a huge investment. There are even a few SDN products that are free. And while you'll need to pay a license fee for some SDN solutions such as VMware's NSX, there are a few that ship with the operating system itself, including Microsoft's Hyper-V Network Virtualization.
And since SDN supports Layer 1 through Layer 3 networking models, there's no need to buy expensive networking devices. In other words, the use of SDN in a production environment can help reduce the costs involved in purchasing expensive hardware.

Overhead Reduction:
In a physical environment, the isolation for the customer workloads requires configuring VLANs on separate networking devices, including routers, switches, etc. Since most of the networking is done at the SDN, it is easy for service providers to isolate the customer virtual machines from other customers by using various isolation methods available in the SDN.

Physical vs. Virtual Networking Management:
Physical environments necessitate collaboration among different teams to get a task done. For example, if you require some modification at a physical networking device, it would often take a considerable amount of time and teamwork in most organizations before the task can be accomplished.
SDN provides you the ability to control the virtual and physical networking by using a central management tool. A virtual administrator can process the necessary changes without needing to collaborate with different teams.

Managing Virtual Packet Forwarding:
SDN can help you forward the virtual packets to a software or physical device running on the network. For example, if a virtual machine needs to access the internet, it becomes easy for virtual administrators to provide the necessary configuration to the virtual machine with minimal effort.

Reduced Downtime:
Since SDN helps in virtualizing most of the physical networking devices, it becomes easy to perform an upgrade for one piece rather than needing to do it for several devices. SDN also supports snapshotting the configuration, which helps you quickly recover from any failures caused by the upgrades.

Isolation and Traffic Control:
Cloud service providers can benefit from centralizing the networking control using a central management tool. At the same time, SDN provides several isolation mechanisms such as configuring ACLs and firewalls at the virtual machine NIC level. You can also define the traffic rules using the SDN management console, which helps in providing full control over the network traffic.

Extensibility:
Since SDN is software-based, it is easy to use SDN API references for vendors to extend the capabilities of an SDN solution by developing applications to control the behavior of networking traffic.

Central Networking Management Tool:
SDN can deliver all your networking needs in one product, enabling you to control every piece of an organization's network using a central management tool.

Network administrators often find it difficult to manage a physical router's configuration, and it quickly becomes time consuming and tedious when more than one physical router needs to be managed. SDN simplifies the management of physical routers by providing the management APIs in the SDN console.

SDN Benefits:
SDN provides several benefits to address the challenges facing legacy network architectures (Jammal et al., 2014).

1) **Programmability of the Network**:
   By implementing a new orchestration level, SDN can tackle the inflexibility and complexity of the traditional network. SDN provides enterprises with the ability to control their networks programmatically and to scale them without affecting performance, reliability, or the user experience. The data and control-plane abstractions constitute the immense worth of SDN. By eliminating the complexity of the infrastructure layer and adding visibility for applications and services, SDN simplifies network management and brings virtualization to the network. It abstracts flow control from individual devices to the network level. Network-wide data-flow control gives administrators the power to define network flows that meet connectivity requirements and address the specific needs of discrete user communities. With the SDN approach, network administrators no longer need to implement custom policies and protocols on each device in the network separately. In the general SDN architecture, control-plane functions are separated from physical devices and are performed by an external

controller (e.g., standard server running SDN software). SDN provides programmability on the control plane itself, through which changes can be implemented and disseminated either to a specific device or throughout the network hardware on a secure channel. This approach promises to facilitate the integration of new devices into the existing architecture. The SDN controller improves the traffic engineering capabilities of the network operators using video traffic. It enables network operators to control their congestion hot spots and reduces the complexity of traffic engineering.

2) **The Rise of Virtualization**:
SDN is a promising opportunity for managing hyper-scale data centers (DCs). Data centers raise significant scalability issues, especially with the growth of virtual machines (VMs) and their migration. Moving a VM and updating the media access control (MAC) address table using traditional network architecture may interrupt the user experience and applications. Therefore, network virtualization, which can be seen as an SDN application, offers a prominent opportunity for hyper scale data centers. It provides tunnels that can abstract the MAC address from the infrastructure layer, enabling Layer 2 traffic to run over Layer 3 overlays and simplifying VM deployment and migration in the network. Furthermore, SDN enables multi-tenant hosting providers to link their physical and virtual servers, local and remote facilities, and public and private clouds into a single logical network. As a result, each customer will have an isolated view of the network provider. SDN adds a virtualization layer to the fabric architecture of the cloud providers. This enables their tenants to obtain various views over the data-center network (DCN) according to their demands. SDN is a promising approach for offering Networks as a Service (NaaS) which will enable flexible service models and virtual network operators and endow enterprises with the ability to control DCs and their traffic. This paper introduces the benefits of NaaS and its consolidation with SDN using different cloud models.

3) Device Configuration and Troubleshooting:
With SDN, device configuration and troubleshooting can be done from a single point on the network which pushes us closer to realizing the ultimate goal of "a dynamic network" that can be configured and made adaptable according to needs. SDN also provides the capability to encourage innovation in the networking field by offering a programmable platform for experiments on novel protocols and policies using production traffic. Separating data flows from test flows facilitates the adoption of newer protocols and ideas into the networking domain. From a broader perspective, SDN offers a form of networking in which packet routing control can be separated from switching hardware. As a result, when the SDN and Ethernet fabrics are consolidated, real network intelligence is achieved.

Disadvantages:

The biggest obstacle the SDN technology could face is related to security. The architecture of SDN shows that there is one layer for the infrastructure layer to be under control. Although the centralized control layer, this layer cold be single point of failure and a potential vulnerability.

2.8.3 NFV (Network Function Virtualization)

Network Function Virtualization (NFV) is a new technology in networking that primarily involves implementing network functions in an open and standardized IT virtualization. In essence, NFV was proposed to decouple network functions (e.g. routers, firewalls, load balancers, NAT, DNS, and other dedicated network servers) from their dedicated hardware and implement them as software components on fully-virtualized network infrastructures.
It's a new way to define, create, and manage networks by replacing dedicated network appliances with software and automation. The mindset is to keep away from dealing with physical hardware that's inflexible and expensive. In an NFV environment, a virtual network function (VNF) takes on the responsibility of handling specific network functions that run on one or more virtual machines (VMs), on top of the physical networking infrastructure (Cui et al., 2012).

1. **Why is it important?**

Service Providers find so many challenges in terms of competition nowadays, the competitors have much more agility in bring services to market with lower cost to do that. All the competitors (SPs) should take the advantages of this new technology, otherwise they will lose the competition.
Technically, SP are struggling about bandwidth as the IP traffic is growing incredibly day by day. By using NFV the traffic IP bandwidth is managed in dynamic troubleshooting methods.

2. **NFV in the future**

Imagine yourself want a new service in your home Figure [20] for example a load balancer service, you may wait for days to have this service installed, configured and ready to use. While by using NFV approach you just need to make a call asking for your service (virtualized network function) or even press some buttons and you will get your service in few hours. So, NFV will facilitate our life and give the end-user the ability to manage the network services. A virtualized network function, or VNF, may consist of one or more virtual machines running different software and programs, on top of standard high-volume servers, switches and storage, or even cloud computing infrastructure, instead of having custom hardware appliances for each network function.

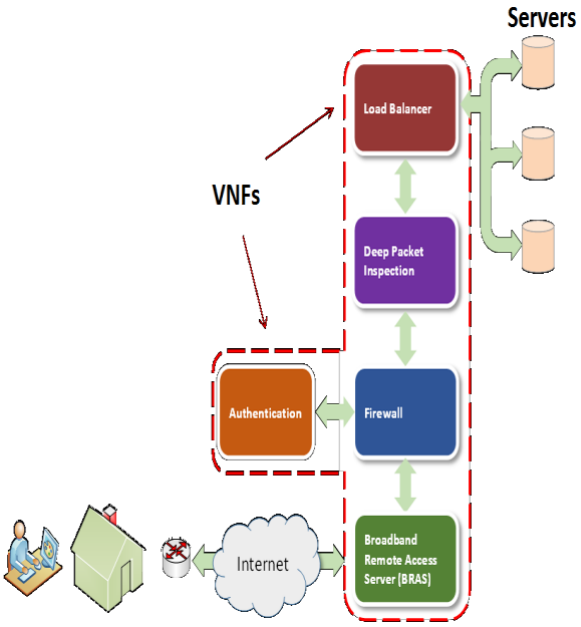

Figure 20 NFV overall view

With NFV it is possible to:

- Shift from equipment-based availability to a service-based (availability of VNF services)
- Orchestrated VNFs that used as required and then discarded when not needed

This flexibility enables SPs to optimize their NFV solutions to meet any VNF availability requirement.

2.8.3.1 The Architecture

The basic components of virtualized platforms where NFV is deployed are:

- Physical server: The physical server is the bare-metal machine that has all the physical resources such as CPU, storage, and RAM.
- Hypervisor: The hypervisor, or virtual machine monitor, is the software that runs and manages physical resources. It provides the virtual environment on which the guest virtual machines are executed.
- The guest virtual machine: A piece of software that emulates the architecture and functionalities of a physical platform on which the desired application is executed.

Virtual machines (VMs) are deployed on high-volume servers which can be located in datacenters, at network nodes, and in end-user facilities. Moreover, most VMs provide on demand computing resources using cloud. Cloud-computing services are offered in various formats: infrastructure as a service (IaaS) that is also referred to as hardware as a service (HaaS), platform as a service (PaaS), software as a service (SaaS), and network as a service (NaaS). There is no agreement on a standard definition of NaaS. However, it is often considered to be provided under IaaS. The NFV technology takes advantage of infrastructure and networking services (IaaS and NaaS) to form the network function virtualization infrastructure (NFVI). To achieve the

objectives promised by NFV, such as flexibility in assigning virtual network functions (VNFs) to hardware, rapid service innovation, enhanced operational efficiency, reduced power usage, and open standard interfaces between VNFs, each VNF should run on a framework that includes dynamic initiation and orchestration of VNF instances. In addition, it should also manage the NFVI hosting environment on IT virtualization technologies to meet all VNF requirements regarding data, resource allocation, dependencies, availability, and other attributes. The ETSI NFV group has defined the NFV architectural framework at the functional level using functional entities and reference points, without any indication of a specific implementation (Hawilo et al., 2014). The functional entities of the architectural framework and the reference points are listed and defined in Figure [21].

The NFV framework consists of three main components (ETSI, 2014):

1. Virtualized network functions (VNFs) are software implementations of network functions that can be deployed on a network function virtualization infrastructure (NFVI).
2. Network function virtualization infrastructure (NFVI) is the totality of all hardware and software components that build the environment where VNFs are deployed. The NFV infrastructure can span several locations. The network providing connectivity between these locations is regarded as part of the NFV infrastructure.
3. Network functions virtualization management and orchestration architectural framework is the collection of all functional blocks with interfaces through these functional blocks to exchange information for the purpose of managing and orchestrating NFVI and VNFs. Virtual network functions should be implemented as simple drag-and-drop operations in the orchestration management system. To make this a reality, both VNFs and computing infrastructure should be described using standard templates that enable automated management. The orchestration management system is responsible for providing and managing the NFV environment through north- and south-bound interactions. North bound interactions are used to manage and provide access to the VNFs. Moreover, VNFs could use them for information or request queries such as asking for more computing resources. South-bound interactions are used to interact with the NFVI and request information from other framework entities. In addition, they are used to request information about policies, VNF software images, descriptions, and network forwarding graphs.

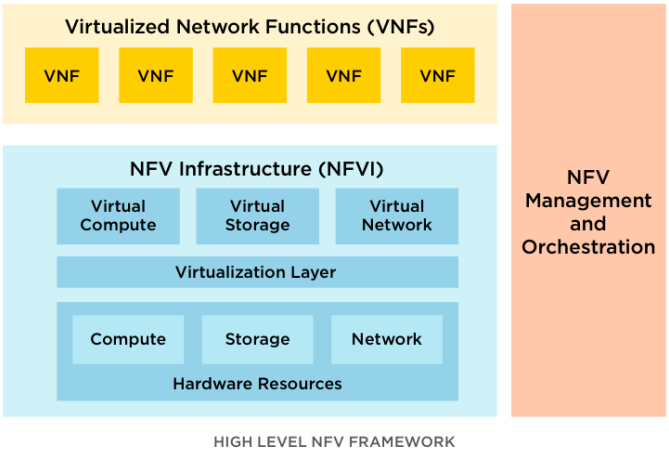

Figure 21 High-Level NFV Framework (Sdxcentral, 2015)

2.8.3.2 Advantages/Disadvantages

NFV provides many benefits to the telecommunications industry, such as Flexibility, Cost, Scalability, Security and others, the following points illustrate some of NFV's advantages:

1. Equipment costs and power consumption will be reduced (less equipment with increased production).
2. Time to Market will be increased by minimizing the typical network operator cycle of innovation. Economies of scale required to cover investments in hardware-based functionalities are no longer applicable for software-based development that the virtualizing depends on.
3. Availability of network appliance multi-version and multi-tenancy, which allows use of a single platform for different applications, users and tenants.
4. Services can be rapidly scaled up/down as required.
5. Pure software entrants that will enable a wide variety of eco-systems.
6. Telecom vendors and NOs can leverage ideas and solutions from the innovative, fast-moving data center ecosystem.
7. Re-use of existing hardware by NOs will help to control OPEX and CAPEX and promote vendors who can provide innovative software appliances.
8. Invite new software-centric vendors and in-house software development. This will lead to increased innovation and agility of the telco services.

Beside the advantages there are challenges to leverage these benefits. Since NFV's challenges still need to be addressed, they are considered as disadvantages. In this part, some of the NFV's challenges are discussed (Hawilo et al., 2014):

1. Achieving high performance virtualized network appliances for different hardware vendors and hypervisors.

2. Achieving co-existence between previous and new hardware based network platforms.

3. Define a unified interface which clearly decouples the software instances from the underlying hardware.

4. Security is an important aspect of the telecommunications industry. NFV should obtain a security level close to that of a proprietary hosting environment for network functions. The best way to achieve this security level is by dividing it according to functional domains. Security in general can be defined according to the following functional domains:
    a. Virtualization environment domain (hypervisor)
    b. Computing domain
    c. Infrastructure domain (networking)
    d. Application domain

5. Security attacks are expected to increase when implementing network functions in a virtualization environment. A protected hypervisor should be used to prevent any unauthorized access or data leakage. Moreover, other processes such as data communication and VM migration should run in a secure environment. NFV uses application programming interfaces (APIs) to provide programmable orchestration and interaction with its infrastructure. These APIs introduce a higher security threat to VNFs.

6. Computing Performance: the virtual environment underlying hardware server characteristics such as processor architecture, clock rate, cache memory size, memory bandwidth, and speed has a profound impact on VNF performance. VNF software design also plays a major role in VNF performance. VNF software can achieve high performance using the following techniques:
    a. A high-demand VNF should be implemented using multithreading techniques, and in a distributed and scalable fashion, in order to execute it on multiple cores or different hosts.
    b. Software instances should have independent memory structures to avoid operating system deadlocks.
    c. VNF should implement its own network stack and avoid networking stacks implementation in the operating system, which consume large amounts of computing resources.
    d. Direct access to input/output interfaces should be used whenever possible to reduce latency and increase data throughput.
    e. Processor affinity techniques should be used to take advantage of cache memories. Implementing these techniques in VNF software may require a different approach from the automated resource allocation within a given pool of servers currently used in IT environments.

7. Unlike the classical approach of interconnecting network functions by a direct connection or through layer 2 (L2) switches, a virtualized environment uses different approaches. In a virtualized environment, virtual machines can be connected in different scenarios:

a. If two VNFs are on the same physical server and the same local access network (LAN), they would be connected through the same Vswitch.
b. If two VNFs are on the same physical server but different LANs, the connection passes through the first Vswitch to the network interface controller (NIC), then to the external switch, and back again to the same NIC. This NIC forwards the connection to the Vswitch of the second LAN and then to the VNF.
c. If two VNFs are on different servers, the connection passes through the first Vswitch to the NIC and then to an external switch. This switch forwards the connection to the NIC of the desired server. Finally, this NIC forwards it to its internal Vswitch and then to the destination VNF.
d. Some NICs provide direct access from the VM. These NICs are single-root I/O virtualization (SR-IOV) compliant. They offer faster and higher throughput to VMs. Each connectivity technique has its own advantages in terms of performance, flexibility, and isolation. Virtual interfaces managed by the hypervisor have lower performance compared to virtual interfaces offered by SR-IOV-compliant NICs. However, virtual interfaces provided by the hypervisor are simpler to configure, and support VM live migration in a simpler way. The right choice depends on VNF workloads.

8. Portability: Virtualized network functions can be deployed in different ways, each with its own advantages and drawbacks. Virtualized network functions that are executed directly on bare metal ensure predictable performance because mappings of software instances to hardware are predictable. This kind of deployment sacrifices resource isolation and makes software-instance security difficult to achieve because multiple software appliances are executed as processes on the same operating system. In addition, the designed software would be OS-dependent.
   Deploying virtual network functions through a virtual environment improves portability and ensures that hardware resources are viewed uniformly by the VNF. This deployment also enables each VNF to be executed on its specific operating system while remaining unaware of the underlying operating system. In addition, VNF resource isolation is ensured because VNFs are executed on independent VMs managed by the hypervisor layer, which guarantees no unexpected interactions between them. Strict mapping of resources should be used to guarantee resource isolation.

9. Coexistence with Legacy Networks: Virtual network functions should be able to coexist with legacy network equipment. It means that:
   a. It should be able to interact with legacy management systems with minimal effects on existing networks.
   b. The network forwarding graph should not be affected by the existence of one or more VNFs.
   c. A secured transition should be ensured between VNF instances and physical functions, without any service interruption or performance impacts.

2.8.4 Conclusion

NFV and SDN aim to revolutionize the network and service provider industries by decoupling network functions from the underlying proprietary hardware and giving a central networking management. They provide all the benefits of IT virtualization platforms. Network engineers and researchers are exploiting virtual environments to simplify and enhance these technologies in order to find their way smoothly into the network and service provider industries. Besides all the advantages brought by NFV and SDN, they face technical challenges that might hinder their progress. For that reason, network engineers, network and telecommunication vendors, and researchers are contributing to address these challenges and explore new approaches to overcome them. OpenStack community is one of the most known cloud communities that keep trying to solve such issues. For that many of IT, network and telecommunication enterprises are using OpenStack as a main open source solution for their development.

2.9 OpenStack in Cloud Computing

OpenStack is an open-source software. Most multinational organizations define OpenStack as the future of Cloud Computing. The Internet and large volumes of data together have instigated the purpose of cloud computing, and OpenStack is one such platform to create and handle massive groups of virtual machines through a Graphical User Interface. It is a set of efficient software tools to manage private and public cloud computing platforms. More details about OpenStack, its architecture, its business value and its services are existed in the following chapter.

2.9.1 OpenStack Concept

OpenStack is one of the cloud operating system for managing public and private cloud management. It is large and everyday increasing project not a product (OpenStack). OpenStack has a long journey to travel. It facilitates the end user to manage virtual machines in virtual networks. It is made up of set of open source projects like OpenStack nova, OpenStack neutron, OpenStack Horizon etc. Every such open source project has their functions and roles to perform in the environment of OpenStack. OpenStack Nova project act as a control layer that sits on top of hypervisor layer and interact with hypervisor. Nova provides the way to access everything other services and it is not dependent on the hypervisor technology used (e.g.: KVM, Xen, VMware, etc.) underneath. OpenStack Neutron project manage networking between in virtual machines launched. OpenStack abstracts the services such as network, storage and exposes a REST API based interface accessible from OpenStack dashboard service which is OpenStack horizon project (Anshu Awasthi, 2015).

2.9.2 OpenStack Business Values

OpenStack can dramatically improve competitive advantage of the company by increasing innovation, lowering operational IT costs, and freeing up funding for projects that can expand the business (Barrett et al., 2016).

Simply put, OpenStack is an open source technology that provides a set of tools for building and managing private cloud computing platforms. It lets the IT folks get these environments operational far more quickly than they can today. Its primary benefits are:

1. Helping to generate significant operational cost savings for business by cutting IT and computer hardware expenses.
2. Helping drive top line benefits and competitive advantage by accelerating innovation within the company.

For organizations that want the peace-of-mind of enhanced security and privacy for their data and applications, a private cloud may be the right choice. A private cloud is dedicated to a single organization or a designated set of users. The pool of computing resources is restricted to these users only, enhancing data security and compliance. However, an enterprise can't just flick a switch and start deploying workloads to a private cloud.

And that's where OpenStack technologies come in. OpenStack provides a set of tools for building and managing private clouds that's backed by some of the biggest names in computing; in fact, Intel is one of the biggest contributors to OpenStack. Because it's an open source technology, you're not locked into a single vendor or to a proprietary technology. Thousands of developers from all over the world contribute to OpenStack, helping to create the strongest, most robust, and most secure cloud infrastructure product available.

OpenStack is much like the Linux operating system. It is based on open source technologies and "distributions" sold by multiple companies. Your firm can purchase OpenStack from firms including Red Hat, Cisco, and VMWare.

For example, by incorporating Intel processors and management technologies into an OpenStack deployment, the organization gains additional security features including data encryption, enhanced network connections to shield content as it's communicated over the network and throughout the cloud, and trusted hardware to assure system integrity and compliance. These technologies also enable energy cost savings by leveraging Intel power management technologies.

With OpenStack, the organization can gain these cloud computing advantages faster, more securely, and with less complexity (Allen, 2015).

2.9.3 OpenStack Architecture

The following figure shows the OpenStack architecture which consists of OpenStack shared services located over standard hardware and managed by the dashboard through APIs (OpenStack).

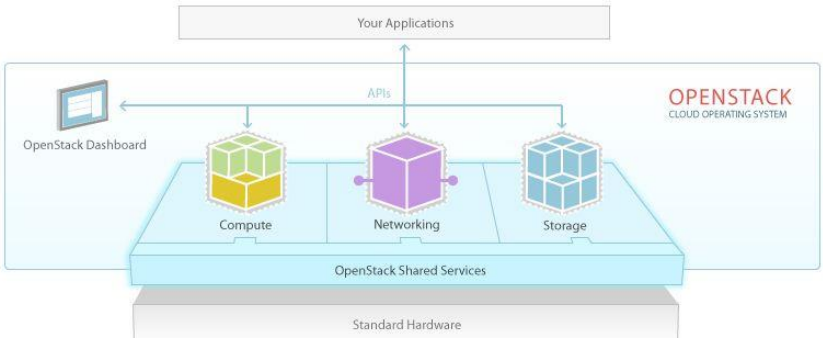

Figure 22 OpenStack architecture (OpenStack)

OpenStack can be built with only three nodes: controller, compute and network (Docs.OpenStack, 2016b).

- The Controller Node It is central point of the cloud services and controls all other nodes of cloud. Basic services provided by this node are the Dashboard, Image service and Identity service. Management portions of Compute and Network nodes are also maintained here.
- The Compute Node It provides the actual computational services. This node has the responsibility of running the hypervisor. Instances are spawned on this node. This mode also manages layer 2 agents which operate tenant networks, networking plug-in and implementation of security groups.
- The Network Node It provides and controls the networking services. It delivers different layer 2 and layer 3 services like deployment of virtual networks and tunnels, NAT, DHCP and routing. Internet connectivity of the instances is also the responsibility of this node.

There are three main networks in OpenStack environment. The management network is used to connect controller, compute and network nodes together. The data network is used for the traffic of the virtual machines (VMs) so the connection should be done between only the compute node and network node. The API network is external connectivity to the users of OpenStack services running on controller node. All the REST API exposed by OpenStack services can be accessed over API network. Network node also has the connectivity to external network and act as gateway to VM traffic towards public network.

OpenStack installation can be done on single node and multiple nodes including three nodes/four nodes. An example of installing OpenStack on single node could be done by using DevStack.

2.9.4 OpenStack Core Services

OpenStack provides a set of core services that can be represented by a modular architecture. These services facilitate scalability and elasticity as core design tenets, and will briefly be reviewed in this section (Docs.OpenStack, 2016b).

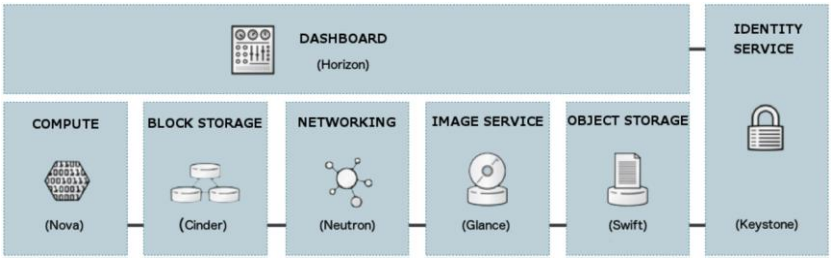

Figure 23 OpenStack Core Services (Docs.OpenStack, 2016b)

As shown in previous figure, OpenStack is a cloud operating system that contains compute, storage, and networking and other services, all of them are managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

2.9.4.1 Identity Service (Keystone)

The OpenStack Identity service (keystone) is a shared service that provides authentication and authorization services throughout the entire cloud infrastructure. The Identity service has pluggable support for multiple forms of authentication.

Security concerns here pertain to trust in authentication, management of authorization tokens, and secure communication.

2.9.4.2 Compute Service (Nova)

OpenStack Compute service (nova) provides services to support the management of virtual machine instances at scale, instances that host multi-tiered applications, dev/test environments, “Big Data” crunching Hadoop clusters, and/or high performance computing.

The Compute service facilitates this management through an abstraction layer that interfaces with supported hypervisors.

2.9.4.3 Storage Service (Swift - Cinder)

- Object Storage

The OpenStack Object Storage service (swift) provides support for storing and retrieving arbitrary data in the cloud. The Object Storage service provides both a native API and an Amazon Web Services S3 compatible API. The service provides a high degree of resiliency through data replication and can handle petabytes of data.

It is important to understand that object storage differs from traditional file system storage. It is best used for static data such as media files (MP3s, images, videos), virtual machine images, and backup files.

Object security should focus on access control and encryption of data in transit and at rest. Other concerns may relate to system abuse, illegal or malicious content storage, and cross authentication attack vectors.

- Block Storage

The OpenStack Block Storage service (cinder) provides persistent block storage for compute instances. The Block Storage service is responsible for managing the life-cycle of block devices, from the creation and attachment of volumes to instances, to their release.

Security considerations for block storage are similar to that of object storage.

2.9.4.4 Image Service (Glance)

The OpenStack Image service (glance) provides disk image management services. The Image service provides image discovery, registration, and delivery services to the Compute service, as needed.

Trusted processes for managing the life cycle of disk images are required, as are all the previously mentioned issues with respect to data security.

2.9.4.5 Networking Service (Neutron)

The OpenStack Networking service (neutron) provides various networking services to cloud users (tenants) such as IP address management, DNS, DHCP, load balancing, and security groups (network access rules, like firewall policies). It provides a framework for software defined networking (SDN) that allows for pluggable integration with various networking solutions.

OpenStack Networking allows cloud tenants to manage their guest network configurations. Security concerns with the networking service include network traffic isolation, availability, integrity and confidentiality.

By using the virtualization, all the physical network devices will move to virtualization layer. The routers and switches are the main network elements that are replaced by virtual ones. Open

Virtual Switch and Distributed Virtual Router are the most known algorithms that are used by OpenStack project.

2.9.5 Overcommitting in OpenStack

OpenStack allows you to overcommit CPU and RAM on compute nodes. This allows you to increase the number of instances you can have running on your cloud, at the cost of reducing the performance of the instances. OpenStack Compute uses the following ratios by default:

- CPU allocation ratio: 16:1
- RAM allocation ratio: 1.5:1

The default CPU allocation ratio of 16:1 means that the scheduler allocates up to 16 virtual cores per physical core. For example, if a physical node has 12 cores, the scheduler sees 192 available virtual cores. With typical flavor definitions of 4 virtual cores per instance, this ratio would provide 48 instances on a physical node.

The formula for the number of virtual instances on a compute node is ***(OR*PC)/VC***, where:

OR: CPU **O**vercommit **R**atio (virtual cores per physical core)

PC: Number of **P**hysical **C**ores

VC: Number of **V**irtual **C**ores per instance

Similarly, the default RAM allocation ratio of 1.5:1 means that the scheduler allocates instances to a physical node as long as the total amount of RAM associated with the instances is less than 1.5 times the amount of RAM available on the physical node.

For example, if a physical node has 48 GB of RAM, the scheduler allocates instances to that node until the sum of the RAM associated with the instances reaches 72 GB (such as nine instances, in the case where each instance has 8 GB of RAM).

This configuration is located in **nova.conf** file under [default] section:

- cpu_allocation_ratio = 16.0
- ram_allocation_ratio = 1.5

2.9.6 The Market of OpenStack

Basically, OpenStack is an open source project. Nevertheless, what makes OpenStack see the light are the vendors that have contributed to develop the free code of OpenStack and then turned it into a product business can be used.

Some companies have used OpenStack as the basis for their public clouds; Rackspace, for example, has proven that OpenStack can power a massive, geographically distributed cloud. Others are packaging the components that make up OpenStack into an easy-to-digest product sold to enterprises for building their own private cloud.

The following list of OpenStack companies (Butler, 2014) was chosen based on which companies have devoted the most resources to OpenStack, which have contributed the most code to the project, and which have the greatest ability to spread this open source platform into the market more broadly.

2.9.6.1 Rackspace

Rackspace is an OpenStack founding father. It started OpenStack officially, along with NASA in 2010. The original storage pieces were contributed by Rackspace, while NASA was the responsible for the computing side. Rackspace managed the project for the first two years before the OpenStack Foundation was established. Since then Rackspace is still seen largely as the public face of OpenStack and one of the most ardent and supportive backers of OpenStack. The company uses OpenStack as the basis for much of its public cloud and it offers customers a distribution of the software to create a private and hybrid cloud based on the same platform. The public cloud and managed hosting provider is one of the first to roll out new OpenStack features in production, and it provides one of the most robust public cloud deployments. Rackspace acts as an ongoing proven example that OpenStack can power a globally distributed massive scale public cloud. It seems that as long as there is OpenStack, Rackspace will be an important and relevant player in the community.

2.9.6.2 Red Hat

Red Hat made its first billion dollars productizing Linux for the enterprise. Now, it wants to do the same for OpenStack. The company has invested heavily in the project. According to stackalytics, it was the leading contributor to OpenStack code among vendors for the Icehouse OpenStack release.

Red Hat has its own distribution of OpenStack, which is integrated deeply with its flagship product, Red Hat Enterprise Linux and is called RHEL OpenStack. Red Hat has contributed many resources to OpenStack, so it is expected to be a player in this market for the long haul.

2.9.6.3 Dell

Dell has had fits and starts in the cloud, but one thing has remained clear: It's commitment to OpenStack. The company originally had plans to build a public cloud based on OpenStack. But, it scraped those plans and instead is now focusing on delivering consulting and implementation services for customers, undoubtedly with a heavy dose of Dell hardware and services on top of it all.

Dell says it wants to help customers deploy whatever is best for them. That could be a private cloud built on OpenStack, or connections to one of the company's public cloud partners, like Joyent or Rackspace. Dell holds some important leadership positions in the OpenStack governance bodies, so it is certainly a force in the OpenStack community.

2.9.6.4 HP

HP seems to be having trouble getting its cloud to catch on. The company has an impressive portfolio though. Its public cloud is based on OpenStack - with plenty of HP sauce layered on top. And that same OS is available to customers to run a private cloud, also leveraging OpenStack. Combined, HP says that creates a hybrid cloud for customers that rivals similar platform-centric plays from companies like VMware and Microsoft. In the Icehouse release HP trailed only Red Hat in terms of the number of contributions of code. So, HP is certainly among the big vendors in the OpenStack community.

2.9.6.5 IBM

Previously IBM announced that OpenStack would be a central part of the company's cloud plans moving forward. Since then, though it's unclear just how big of a part OpenStack plays in the company's cloud plans. IBM has certainly been committed to contributing toward the development of OpenStack. IBM is one of the leading contributors to the project, along with every other company on this list. IBM is using its experience in working with enterprise customers to improve areas such as quality assurance and aligning the OpenStack API to key standards. But, the company has not made OpenStack central to its own products it sells. IBM bought SoftLayer, an IaaS provider, and is in the process of expanding OpenStack support in SoftLayer's cloud. Since that OpenStack announcement IBM has also made commitments to Cloud Foundry, another open source project for application development. And it has announced BlueMix, a PaaS offering that's still in its early stages.

2.9.6.6 CISCO

It seems that the main goal of Cisco being involved in OpenStack is to ensure that the hardware the company makes - its networking, converged infrastructure and servers - are all compatible with OpenStack. Cisco also plays an important role on the leadership and marketing efforts, lending the head of its cloud technology team to be vice chairman of the OpenStack board of directors and sharing stories of how the company's WebEx team has deployed OpenStack within the company. It's yet to be seen how central of a role OpenStack will play in the company's recently announced InterCloud.

2.9.6.7 Mirantis

A number of companies have sprouted up as pure-play OpenStack vendors that focus solely on supporting and selling OpenStack-related products and services. Embracing this strategy has helped Mirantis, a Mountain View, Calif.-based company grow from a venture-backed startup to now one that has more than 400 employees. The company originally supported a variety of OpenStack deployment models, but as the open source project has matured, so too has

Mirantis's offerings. The company now has its own distribution of OpenStack which users can implement on their own, or with the support of Mirantis engineers. The company is also building up its partnerships, including those announced with Red Hat, VMware and it recently publicized a $30 million deal, which was accompanied by an investment, from mobile provider Ericsson to implement a cloud for that company.

2.9.6.8 Canonical OpenStack

The battle to control OpenStack is now being duked out by companies that made their name originally on Linux. Red Hat is the most prime example of that, but Canonical through its Ubuntu operating system has been integrating OpenStack features into its OS. Canonical is now one of the major OpenStack companies. In a survey of OpenStack users conducted by the OpenStack Foundation last year, it found that Canonicals' Ubuntu was the leading operating system for OpenStack deployments. Canonicals' latest release of Ubuntu 14.04 focuses heavily on integrating OpenStack.

2.9.6.9 VMware

VMware has what could be viewed as somewhat of a love/hate relationship with OpenStack. Fundamentally, OpenStack can be seen as a threat to VMware, as an open source alternative to VMware's products for building and managing clouds. OpenStack is basically a free version of software that accomplishes the same thing that VMware's software does. VMware says that it wants customers to be able to manage their OpenStack clouds using VMware's tools, including its ESX hypervisor. This is an evolving relationship that VMware is treading lightly on.

2.9.7 Use Cases using OpenStack

OpenStack powers a large number of organizations to serve different purposes. Real-life examples of the most widely deployed use-cases and their benefits will be mentioned in the following section.

2.9.7.1 Live Migration

Migration enables the system administrator to move a virtual-machine instance among the hosts, which is useful when a compute host needs maintenance. Migration can also be used to redistribute the load when many VM instances are running on a specific physical machine. There are two main migration types (Docs.OpenStack, 2016a):

- Non-live migration: The instance is shut down for a period of time to be moved to another hypervisor. In this case, the instance recognizes that it was rebooted.

- Live migration: Almost no instance downtime. Useful when the instances must be kept running during the migration.

OpenStack has three main nodes/servers in its infrastructure: the controller node, compute node and network node. The more resources are requested to enhance the processing level; the more compute nodes could be added. Therefore, the overall infrastructure of OpenStack could include one compute node or more as needed. Live migration is the movement of a live instance (VM) from one compute node to another. Memory, storage, and network connectivity of the virtual machine are transferred from the original guest machine to the destination. A hugely sought-after feature by cloud administrators, it's used primarily to achieve zero downtime during OpenStack cloud maintenance and can also be a useful feature to achieve performance as live instances can be moved from a heavily loaded compute node to a less loaded compute node.

2.9.7.2 Big Data

The continuous increase of computational power has produced an overwhelming flow of data. Big data is not only becoming more available but also more understandable to computers. For example, the famous social network Website, Facebook, serves 570 billion page views per month, stores 3 billion new photos every month, and manages 25 billion pieces of content. Google's search and ad business, Facebook, Flickr, YouTube, and LinkedIn use a bundle of artificial-intelligence tricks, require parsing vast quantities of data and making decisions instantaneously. Multimedia data mining platforms make it easy for everybody to achieve these goals with the minimum amount of effort in terms of software, CPU and network(Ji et al., 2012).

Then what is Big Data? the definition of big data as it is given by the Gartner: "Big Data are high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization" (Chen and Zhang, 2014).

Some new cloud-based technologies have to be adopted because dealing with big data for concurrent processing is difficult. OpenStack is one of the cloud-based platforms that has good community and ecological environment. The architecture and components of OpenStack are straightforward and stable, so it is a good choice to provide specific applications for enterprises(Ji et al., 2012).

Using OpenStack to manage and deal the Big Data gives some business benefits, such as:

- Rapid and dynamic provisioning of a cluster to provide researchers
- Ability to dynamically change the role of nodes providing elasticity
- Ability to scale resources as user's grow
- Open source benefits including ability to contribute to meet their needs

Chapter 3

3. Evaluation of OpenStack Performance in Data Center Environment

In order to evaluate the work, there is a need to apply a specific use-case. The Live Migration of the virtual machines (VMs) between the nodes was chosen for this purpose. Therefore, by implementing this use-case using OpenStack with its underneath technologies not only the cloud computing PoC (Proof-of-Concept) will be approved but also a useful and practical use-case will be managed.

3.1 Implementing the solution

The most challenging thing is how to install OpenStack. For that, there are three primary choices: paying someone to install and manage OpenStack “managed cloud”, using an OpenStack distribution “distro”, or downloading software from OpenStack.org and building a customized and own system “DIY” (Barrett et al.).

- **Managed cloud**

A managed cloud represents the most hands-off approach to cloud computing. It means that another business creates, manages, and supports the cloud. As a result, the organization that uses this approach will still need domain operators. This requires personnel to manage the

relationship between the end users of the cloud, employees or developers, and the cloud itself. They perform administrative tasks such as creating users, defining the quota of cloud computing services that users can consume, managing costs associated with the created cloud, and business partner troubleshooting.

- **OpenStack distributions (distros)**

It is possible to build and control a substantial portion of the cloud without having to do it alone. Cloud distribution providers develop tools to make the most of the OpenStack platform, such as testing, bug fixing, and packaging to prepare OpenStack for cloud architects. Effectively, software is open source and free. Organizations should pay for OpenStack implementation and back-end services to support cloud resources that they control and manage.

If the organization's choice is distro, it will need domain operators, similar to organizations with managed clouds. It also needs cloud architects, who select the cloud's hardware and software infrastructure and determine what services, such as storage, networking, or compute are going to be offered.

- **DIY Cloud**

A DIY cloud is great for organizations that want unlimited flexibility and customization. In addition to the resources, the organizations need an engineering team to handle tasks that distribution vendors would otherwise tackle, such as advanced support, implementation, packaging, and testing.

The following table shows a comparison of these three consumption models:

Table 1 Approximate business values by consumption models (Barrett et al.)

	Managed	**Distro**	**DIY**
Time to production	Fastest	Moderate	Slowest
Configuration flexibility	Low	High	Highest
Software Customization	None	Depends on vendor policy	Unrestricted
Support	Vendor	Vendor	Community
External Costs	High	Med	Low
Internal Resource Requirements	Low	Med	Highest
Level of Resource Skill	low	Med	High

Implementation is the culmination of the work in order to solve the faced problem and requires big attention to detail. There are three basic stages involved:

- Pre - Implementation

- During - Implementation
- Post – Deployment

3.1.1 Pre – Implementation

This stage aims to plan and prepare all what could be needed to implement the solution. For example, it may include a comprehensive study of the hardware and software tools that are going to be used, as well as the parameters in order to make some measurements, comparison and analysis of the reached results. All of that should be known before starting the implementation phase.

3.1.1.1 Hardware and Software tools

In order to implement, manage and evaluate the certain use-case, some of the hardware and software tools are selected. The description of these tools with the reason of choice each one are detailed as following:

- **VirtualBox**

VirtualBox is a software virtualization package that installs on an operating system as an application. VirtualBox allows additional operating systems to be installed on it, as a Guest OS, and run in a virtual environment.

More and more people are interested in cloud computing and OpenStack but many of them give it up because they can't test or interact with this kind of infrastructure. This is mostly a result of either high costs of hardware or the difficulty of the deployment in a particular environment.

In order to help the community to interact more with cloud computing and learn about it, VirtualBox allows to set up a cloud environment on a personal laptop/desktop, no matter which operating system installed (Windows, Linux, OS X). It also gets the job done with a free and familiar virtualization environment. On the other side, VMware also supports multiple host and guest operating systems but it is not completely free, it has restrictive license and the size of the installation file is heavier that the VirtualBox's installation file (Li, 2010). For that, VirtualBox was chosen to develop and deploy the use-case of this thesis.

- **DevStack**

DevStack is an open source shell script which can be used to deploy OpenStack. However, DevStack supports the installation of OpenStack only on Ubuntu versions, Fedora versions and RHEL releases. It can be used by developers to get an easy setup of OpenStack. It is easy to get a hands-on on OpenStack via DevStack. DevStack installs the OpenStack environment within minutes. The default services constituted by DevStack are Identity "keystone", Object Storage

“swift”, Image Service “glance”, Block Storage “cinder”, Compute “nova”, Networking “nova”, Dashboard “horizon”, and Orchestration “heat” (Anshu Awasthi, 2015).

- **SuperPuTTY**

PuTTY is an open source application which has the capability to act as a client for the SSH, Telnet, rlogin, and raw TCP computing protocols, as well as, a serial console client. Although it was originally created for Microsoft Windows, it is also used with Unix based platforms. SuperPutty is a GUI “Graphical User Interface” based application for PuTTY SSH Client which can not only perform regular PuTTY commands, but also allows it to be opened in multiple tabs.

This application -through SSH protocol- has the ability to connect and manage the hosts remotely. In Live-Migration use-case, SuperPutty is the graphical user interface of OpenStack, because it has handled the control, network and compute nodes, as well as, the virtual machines that are located in the compute node including the VMs that are going to be migrated.

- **TCPdump**

TCPdump is a common packet analyzer that runs under the command line. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached (tcpdump).

The best feature of this tool is the multiple output form. TCPdump’s output could be displayed on the command line interface immediately, or collected in “.pcap” file. After that, it can be retrieved for external analysis. Basically in the Live-Migration use-case, TCPdump collects the traffic of the virtual machine that is migration from host to another. After the Live-Migration is done the whole traffic is saved in “.pcap” file for analyzing later.

- **Wireshark**

Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education.

The output file that is saved by TCPdump as a “pcap” file is analyzed by Wireshark which has the ability to open, analysis and manage this file. Moreover, Wireshark generates graphs and reports depending on the result of the output file.

- **System Requirements**

A new VM has Ubuntu Server 14.04. as its main operating system, as well as, the hardware resources minimum requirements as following:

- Processor - at least 2 cores
- Memory - at least 8GB
- Hard Drive - at least 60GB

When allocating CPUs and RAM to the DevStack, assess how big clusters you want to run. A single Hadoop VM needs at least 1 cpu and 1G of RAM to run. While it is possible for several VMs to share a single cpu core, and they can't share the RAM.

3.1.1.2 Network Performance Parameters

A list of parameters is determined to measure the network performance during the Live-Migration process. The objective of this section is to analyze and compare the following results and take conclusions about the performance of OpenStack when a VM is migrating from one physical server to another one.

The purpose is to evaluate bandwidth, jitter, delay and packet loss. They are described as follows:

- **Bandwidth**

Bandwidth is the bit-rate of available or consumed information capacity expressed typically in metric multiples of bits per second. Variously, bandwidth may be characterized as network bandwidth, data bandwidth, or digital bandwidth.

The general formula for bandwidth is measured by bytes/over time. For example, transfer a 100 MB file from one computer to the other takes 10 seconds to transfer, the bandwidth is 10MB/s.

- **Jitter**

Jitter is defined as a variation in the delay of received packets. The sending side transmits packets in a continuous stream and spaces them evenly apart. Because of network congestion, improper queuing, or configuration errors, the delay between packets can vary instead of remaining constant.

- **Delay**

Network delay is an important design and performance characteristic of a computer network or telecommunications network. The delay of a network specifies how long it takes for packets/data to travel across the network from one node to another. It is typically measured in multiples or fractions of seconds.

- **Packet Loss**

Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is typically caused by network congestion. Packet loss is measured as a percentage of packets lost with respect to packets sent.

3.1.2 During – Implementation

Implementing the solution by applying practical work procedures for each phase of the solution. This phase describes how to deploy the Live Migration use-case from scratch. Therefore, all the hardware and software requirements from the infrastructure layer to the application layer will be addressed. But before all of that it is mandatory to know what are the implementation phases of OpenStack deployment.

3.1.2.1 Implementation Phases

Implementing a cloud could be a challenging process for enterprises, it depends on different factors including budget, people, and consumption model (Barrett et al.). For most enterprises, OpenStack is implemented in three phases:

1. **Proof-of-concept (**PoC**):** Focusing on planning the overall deployment and the initial goals.
2. **Pilot:** Works on specific use-cases to production phase.
3. **Production:** Focusing on the program deployment and long-term goals.

During this subsection, considerations about each phase will be highlighted.

- **OpenStack services to support the selected use-case**

OpenStack offers many services such as compute, storage, application catalog, and database. Depending on the use cases, workload requirements, and implementation phase, a group of OpenStack services could be chosen. Most PoC deployments start with a simple set of core services and slowly include additional services after gaining experience with OpenStack. For example, some might start with only the compute capability (Nova) to provide a self-service VM provisioning feature without any persistent storage capability (Cinder or Swift). Otherwise, the opinion of OpenStack expert should be taken for other deployments.

- **Cloud capacity and performance**

Planning for enough capacity to support the use cases is important to the success of the cloud deployment. In OpenStack, every virtual machine is associated with a virtual hardware template called a flavor, which defines the vCPUs, memory, and ephemeral storage for each virtual machine. The following table illustrates the default flavors in most OpenStack installations, these flavors are able to be customized. Defining new ones that suit the environment is a possible option as well.

Table 2 Default OpenStack flavors

Flavor Name	vCPU	Memory	Ephemeral Disk
m1.tiny	1	512 MB	1 GB
m1.small	1	2048 MB	20 GB
m1.medium	2	4096 MB	40 GB
m1.larg	4	8192 MB	80 GB
m1.xlarg	8	16384 MB	160 GB

The total amount of vCPU, memory, or ephemeral storage consumed by all the VMs of the use-case will determine the hardware architecture. Most deployments tend to overcommit the CPU (e.g.1:5 ratio of physical CPUs to virtual CPUs) but not overcommit the memory (1:1 ratio of physical memory to virtual memory). Moreover, an extra CPU and memory usage required by the OpenStack services and local system processes should be considered, and the amount of disk space required to provision the VM's ephemeral disk should be determined previously. SSD drives can be used on the compute hosts for faster performance.

In OpenStack, storage can be ephemeral or persistent. Ephemeral storage will persist with the lifespan of the associated VM. Anything written to that storage will be lost if the associated VM is deleted. In most default installations, the virtual disks associated with the VMs are ephemeral. If the use cases require saving the data regardless of the VM lifespan, it is necessary to use persistent storage. OpenStack offers a few types of persistent storage such as block storage and object storage. Data written to these persistent storage types will remain available regardless of the state of the associated VM. This information will be useful to estimate the storage requirements. If the use case requires high disk I/O performance, high-speed SSD drives are considered as the best choice here as well.

Networking is also an important aspect when it comes to OpenStack deployments. Use case requirements such as latency, throughput, and security will influence the network architecture design. For better performance, high-speed 10G network equipment or network bonding could be an option. Different vendor distributions might use different network topologies in their architecture. Then it is necessary to work closely to the enterprise network architect and information security architect to determine how to integrate OpenStack network with the corporate network environment such as vlan, firewall, and proxy.

- **Automation and tools**

It is highly recommended to incorporate automation capabilities, even in the early phase of the cloud implementation. Running an OpenStack cloud requires a new operating model, as compared to the conventional IT model. Adopting a DevOps mindset allows to get the best benefits of OpenStack cloud and accelerate the time-to-value. On the other hand, deploying OpenStack manually (either DIY or using distro), becomes burdensome beyond just a few hosts. It is not recommended to deploy OpenStack services manually given its services dependency complexity. Automation and configuration management are the keys to successfully deploying and maintaining a healthy OpenStack cloud.

Most OpenStack distributions have their own deployment tools. In case of using a distribution and moving to build a DIY cloud has decided, there are tools available and will be useful such as: OpenStack Ansible, Chef OpenStack, Puppet OpenStack, Kolla, TripleO, and Fuel. If these options don't work or are missing features needed for the DIY cloud, the cloud responsible can build something by himself. However, there is a need to sort out the package dependency issues for a successful installation. It is key to do some sort of automation to get the cloud up and running quickly in a stable and manageable manner.

Other tasks should be considered for day-to-day cloud operation such as host and VM monitoring and alerts, logging, accounting, disaster recovery, and user management. Most enterprise IT organizations have existing tools to support data center operation. Other open-source toolsets and plugins are available that provide similar capabilities to operate the cloud.

- **Evolving from PoC to pilot to production**

The first phase of deploying an OpenStack cloud is the proof-of-concept (PoC). If the PoC is for an initial feasibility study or prototyping, the best choice is running an all-in-one deployment on a single physical host. Many of the OpenStack distros provide a quick and easy “all-in-one” installation. This helps familiarize the team with OpenStack concepts before the company decides to invest more resources into the next phase.

To help estimate the bill of materials, it is necessary to revisit the cloud capacity and plan for future expansion. Many moving parts (e.g. CPU, memory, storage, networking) are needed to run an OpenStack cloud. Understanding the use-case scenario will help avoid over-provisioning one element and under-provisioning another.

OpenStack can be deployed in high availability mode (HA), which requires a more complex configuration and more physical nodes. Depending on the use-case requirements, budget, and objectives, a simple approach is to deploy a functional OpenStack cloud in non-HA mode using the fewest, minimally configured physical nodes during the PoC phase before you move into HA mode in pilot or production phase.

Moving the OpenStack cloud into the production phase requires thorough architecture planning. One of the most frequently asked questions is how to scale the cloud in a production environment. The OpenStack cloud is designed to be a modular and scalable system that allows services to be distributed across multiple nodes or clusters. Most production deployments choose an HA architecture to provide reliable, high quality service to mission-critical applications.

- **OpenStack Services Types**

Various OpenStack services are stateless, where every request is treated as an independent transaction, unrelated to any previous or subsequent request. This service does not need to maintain the session or status information for each communication. Examples include nova-api, nova-conductor, glance-api, keystone-api, neutron-api. To provide HA, you can simply provide redundant instances of the service and load balance them.

In OpenStack, there are also stateful services that need to be handled with care in an HA configuration. In a stateful service, a request depends on the results of another request to complete a transaction or action. This requires that the session or status information for each communication is maintained. Examples include backend databases and message queues. Clustering technologies such as MySQL Galera, RabbitMQ Clustering, Pacemaker, and Corosync are generally used in OpenStack HA deployment, which requires more complex configurations.

- **For Better Performance**

OpenStack provides mechanisms that segregate the cloud and allow to deploy applications in a distributed or redundant site, eliminating single points of failure. In OpenStack, cells and regions are used to distribute a single cloud across different geographical locations. Availability zones or host aggregates are used to partition a deployment within a single site. They can be used to arrange a set of hosts into logical groups with physical isolation. For example, grouping the compute hosts so they use different power sources, where a failure of one will not bring down both of them. Host aggregates are used to group a set of hosts that share common features, for example, two types of hypervisor (e.g. KVM vs ESXi) or different classes of hardware (e.g. SSD vs. HDD storage). These segregation methods allow to design the cloud infrastructure as

common logical groups based on physical characteristics or capabilities, and to ensure that specific workload/use-case can be distributed and placed on relevant resources.

3.1.2.2 Live Migration Architecture

This process is aiming to develop and test a redundant mechanism and provide a non-disruptive migration of VMs, ensuring seamless service during the live migration. The following figure shows the conceptual architecture of this process.

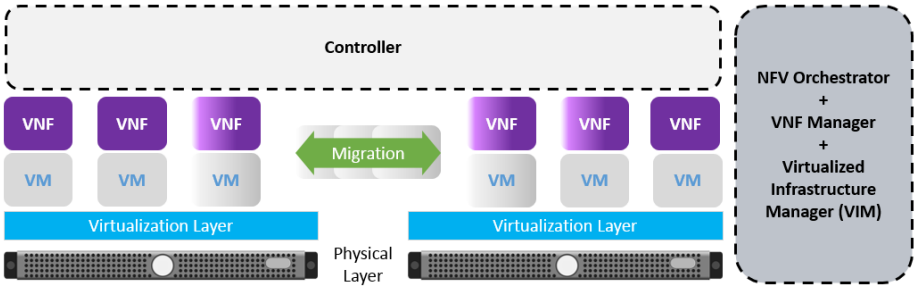

Figure 24 Conceptual Architecture of Live Migration Process

As the figure presented, live migration use-case deployment needs some components to be included. The Physical layer has at least two compute nodes. The compute nodes are the hosts for running the instances or VMs. These nodes run the bare minimum services as much as possible to facilitate the instances. In nova (compute service), they only run the nova-compute service. It also runs few neutron services (network services) for establish networking for the instances such as the OpenvSwitch and Linux Bridge agent services. Each compute node runs a hypervisor that creates the virtualization layer which gives the ability to create various virtual machines (VMs) with different operating systems, each VM has a specific task/applications. The management services will be the responsibilities of the controller.

Therefore, in case there is a need to make Live Migration of some instances located in one compute node to another. After creating a suitable environment which requires a shared zone among the compute nodes so both hypervisors have access to shared storage, it will be possible to execute live migration command by the controller and only after this procedure the live migration will be done successfully.

3.1.2.3 Live Migration Process

First approaches to implement migration of a virtual machine relied on suspending the virtual machine before the transmission. In order to reduce the resulting downtime of the virtualized system, researchers and later on vendors turned to so-called live migration which reduces virtual machine downtime significantly. Today, the majority of virtualization products support live migration for moving a running virtual machine (VM) to a new physical host with minimal

service interruption. This renders live migration an attractive tool also for dependable computing. However, each migration procedure still consumes time and still involves some short service unavailability (Salfner et al., 2012).

The process of Live Migration consists of five steps, assuming there are two compute nodes “Host A and Host B”:

1. **Pre-Migration**: Active VM on physical host A, host B selected by scheduler or preselected

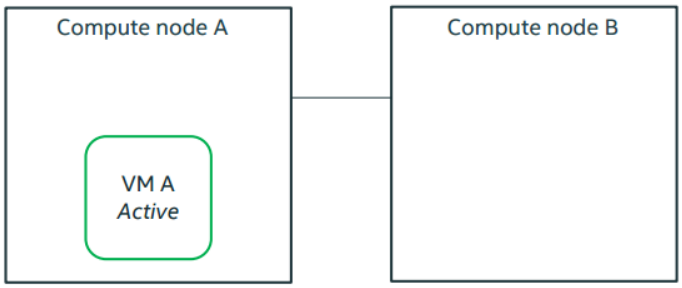

Figure 25 Live Migration Process - Pre-Migration

2. **Reservation**: Confirm availability of resources on host B; reserve a new VM.

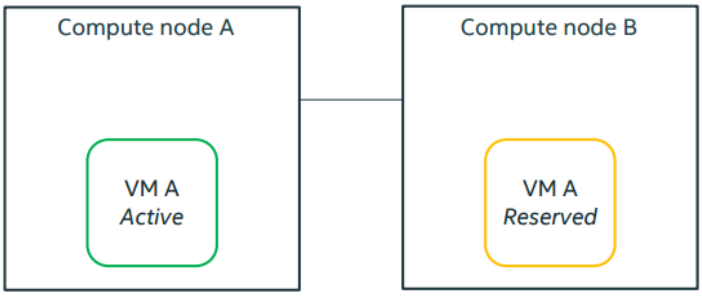

Figure 26 Live Migration Process - Reservation

3. **Iterative pre-copy**: Memory is transferred from A to B and next dirtied pages are iteratively copied.

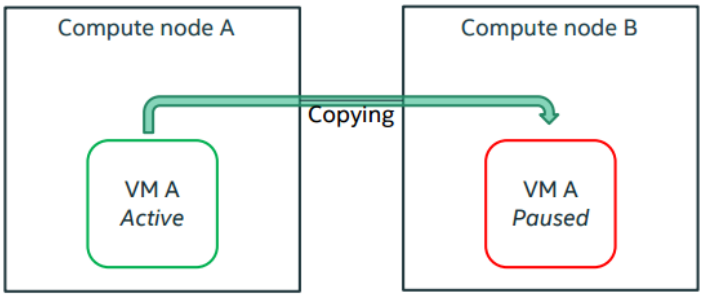

Figure 27 Live Migration Process - Iterative pre-copy

4. **Stop and copy**: Suspend VM and copy remaining pages and CPU state.

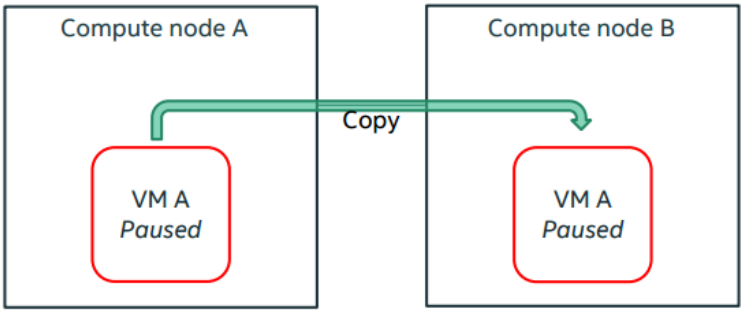

Figure 28 Live Migration Process - Stop and copy

5. **Commitment**: Host B becomes primary host for VM A.

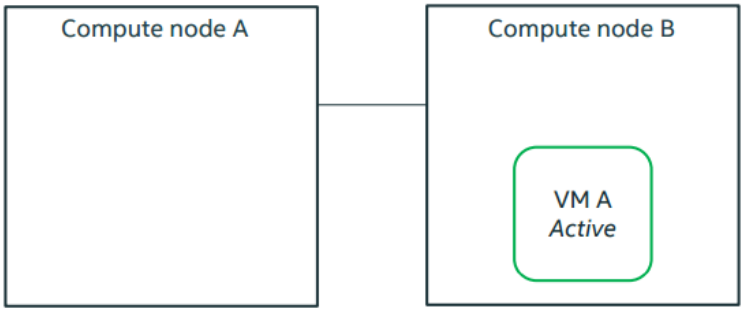

Figure 29 Live Migration Process - Commitment

3.1.2.4 Practical Work Procedures

This section is going to show how to configure every component needed from scratch. There are four steps of configuration should be done, so the live migration use case will be deployed eventually. These main four steps are:

1. **Environment Setup:**
   After this phase of configuration is done, the infrastructure of the work will have three servers (nodes) configured with specific hardware requirements and will be running Ubuntu 14.04 LTS as their operating system. The first node will be the responsible of the operations among the other nodes and is called the controller. The second node is the core of the processing and its name is the compute node. The network node is the final node and its mission is managing the network connections among the nodes.

2. **Network environment:**
   There is a need to create 3 networks before installing OpenStack, which are:

- Management Network: to give a full control over the nodes.
- Tunnel Network: the VMs that are going to be created inside the compute nodes will have a specific virtual network, so they can connect with each other.
- External Network: it will provide the ability to access the outside of OpenStack environment, for example internet, it is for the hosts themselves (controller, compute and network) and the VMs in the compute nodes afterword.

3. **DevStack Installation:**
   The installation process will end up with OpenStack with Open Virtual Switch (OVS) and DVR installed and configured. After this level it is possible to access the dashboard and check the ability of creating new networks, routers and instances.

4. **Live Migration Configuration:**
   The first essential requirement is configuring a shared file by creating this shared file on controller and mounting it from all the compute nodes. The idea is that all the instances from different compute nodes will be located in this shared file logically, after that the ability to migrate the instance from side to another will be possible.

These steps are described in detail in Appendix A and Appendix B, they are written in order to be a guide for specialists who are looking for building the same project or similar one.

3.1.3 Post- Deployment

After the live migration use-case deployment phase, it is mandatory to review and analyze the results. The hardware and software tools and the key network parameters that are mentioned previously will be used to serve the objectives of this section.

3.1.3.1 The Practical Results

Some commands should be executed in order to make sure that all the required components are existed and everything is running as expected.

1. It is possible to know the name, id and the version of the image that is used by the virtual machines as their operating systems, by typing “glance image-list” in the command line:

```
stack@controller:~/liberty/devstack$ glance image-list
+--------------------------------------+---------------------------------+
| ID                                   | Name                            |
+--------------------------------------+---------------------------------+
| 2596e3bb-ebdc-4659-91df-d74b7e617b82 | cirros-0.3.4-x86_64-uec         |
| 49c1c73d-1e63-4ab3-998b-0ccb4d55330a | cirros-0.3.4-x86_64-uec-kernel  |
| f8f34f75-41fc-4fc9-8b9d-317fea614592 | cirros-0.3.4-x86_64-uec-ramdisk |
| a3f3fae1-ff40-4632-a9d7-f4361897f4b0 | Core_64                         |
+--------------------------------------+---------------------------------+
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$
```

Figure 30 DevStack - glance image-list

2. The list of the available networks in OpenStack project will be printed after entering this command “neutron net-list”:

```
stack@controller:~/liberty/devstack$ neutron net-list
+--------------------------------------+---------+----------------------------------------------------+
| id                                   | name    | subnets                                            |
+--------------------------------------+---------+----------------------------------------------------+
| 3d48b47d-d3b1-4b8c-bbf6-e8a314ef691a | public  | 1114bcc9-00e2-4c83-a418-76014e3898fe 10.100.1.0/24 |
| 1ff923f9-61cb-4e53-a06d-12fde1512a9b | private | 7f250b58-7341-4aaf-93e1-436da913b97a 10.99.0.0/24  |
+--------------------------------------+---------+----------------------------------------------------+
stack@controller:~/liberty/devstack$
```

Figure 31 DevSatck neutron net-list

3. The current flavor list which refers to the virtual hardware template that defines the vCPUs, memory, and ephemeral storage for each virtual machine, by using: “nova flavor-list”:

```
stack@controller:~/liberty/devstack$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True      |
| 42 | m1.nano   | 64        | 0    | 0         |      | 1     | 1.0         | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True      |
| 84 | m1.micro  | 128       | 0    | 0         |      | 1     | 1.0         | True      |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$
```

Figure 32 DevStack nova flavor-list

4. The list of the available hypervisors with their hostname, status and state by using: “nova hypervisor-list”:

```
stack@controller:~/liberty/devstack$ nova hypervisor-list
+----+---------------------+-------+---------+
| ID | Hypervisor hostname | State | Status  |
+----+---------------------+-------+---------+
| 1  | controller          | up    | enabled |
| 2  | compute1            | up    | enabled |
+----+---------------------+-------+---------+
stack@controller:~/liberty/devstack$
```

Figure 33 DevStack nova hypervisor-list

Practical formulas:

Transmission time (in seconds) = Size of file (in bits) / Bandwidth (in bits/second)

Bandwidth (in bits/second) = Size of file (in bits) / Transmission time (in seconds)

- **MTR tool**

MTR is a network software which combines the functionalities of the **traceroute** and **ping** software in one network diagnostic tool.

MTR probes routers on the route path by limiting the number of hops that individual packets may have, and listening to responses of their expiry. Usually it is repeated once per second, and keeps tracking of the response times of the hops along the path. For live migration use-case, the most important parameter can be measured by this tool is the packets loos which will be increased in case the virtual machine goes down while migrating.

This tool will be used not only for proofing that live migration is executing correctly, but also to make a comparison between various virtual machines with different flavors. This compassion depends on the down time of each VM.

Nano flavor:

It is possible to create new instance and define its name, imageID, flavorID and nicID, by typing:

nova boot --image *imageID* **--flavor** *flavorID* **--nic net-id=***nicID* ***VM_Name***

These parameters are available by executing the commands that are mentioned previously in this section, so the virtual machine's name could be "Nano_instance" and the rest parameters as below:

"nova boot --image Core_64 --flavor m1.nano --nic net-id=1ff923f9-61cb-4e53-a06d-12fde1512a9b Nano_instance"

OpenStack dashboard gives the ability to check if the instance is created and running without any problems, as well as it shows its IP and the host's name which is the location of the instance:

Host	Name	Image Name	IP Address	Size	Status	Task	Power State	Time since created
controller	Nano_instance	Core_64	10.99.0.22	m1.nano	Active	None	Running	4 minutes

Figure 34 Instance characteristics taken from dashboard

Running MTR tool on the server (sending and receiving packets to/from the VM (Nano_instance) that is going to be migrated), by running "mtr –report-cycles 50 10.99.0.22":

Figure 35 mtr on the server

In the meanwhile, the live migration process is executing. The live migration command is "nova live-migration INSTANCE_NAME (HOST NAME which the instance is moving to)", so as noticed from the figure below, the host of the instance was "controller" and after the live migration became "compute1":

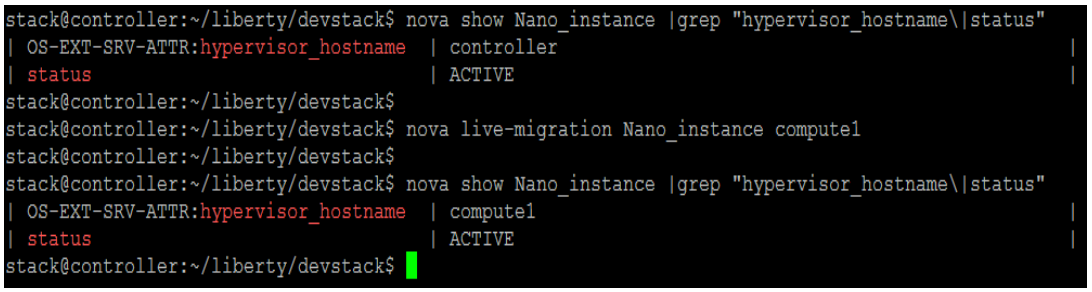

Figure 36 Before and After Live-Migration of Nano instance

The result of mtr in order to calculate the down time that has been taken during the live migration time:

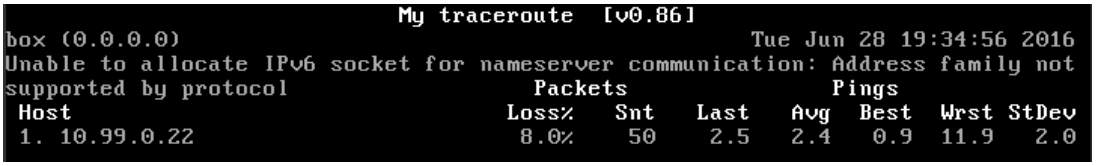

Figure 37 mtr results

The analysis:

8.0% packet loss, 50 packets have been sent within 2.4 milliseconds as an average round-trip time (Sending & Receiving) for every single packet.

>> Packet loss × The total number of packets = The number of lost packets during the migration process

>> 8% × 50 = 4 packets lost

>> The number of lost packets × The Average round-trip time of all the packets = The approximate Downtime of the VM while migration

>>The down time >> 4 packets lost × 2.4 ms = **9.6** milliseconds

Micro flavor:

Create an instance with micro flavor using OpenStack dashboard:

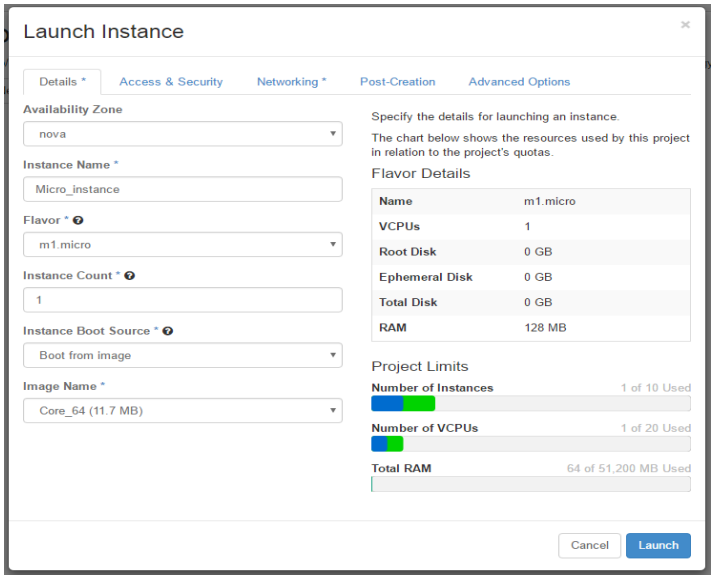

Figure 38 Creat an in stance by dashboard

Live migration of Micro virtual machine:

```
stack@controller:~/liberty/devstack$ nova show Micro_instance |grep "hypervisor_hostname\|status"
| OS-EXT-SRV-ATTR:hypervisor_hostname  | controller                                                    |
| status                               | ACTIVE                                                        |
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$ nova live-migration Micro_instance compute1
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$ nova show Micro_instance |grep "hypervisor_hostname\|status"
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute1                                                      |
| status                               | ACTIVE                                                        |
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$
```

Figure 39 Before and After Live-Migration of Micro Instance

The result of mtr tool:

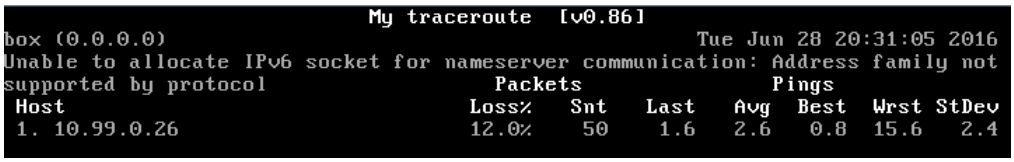

Figure 40 mtr result

The Analysis:

Packet lost >> 12% × 50 = 6 packets

Downtime >> 6 × 2.6 = **15.6** milliseconds

Tiny flavor:

After creating a Tiny virtual machine and making sure that the live migration is done successfully as showed by the following figure:

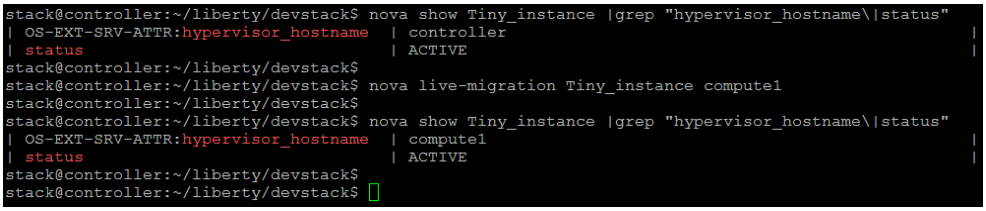

Figure 41 Before and After Live-Migration of Tiny instance

The results of mtr:

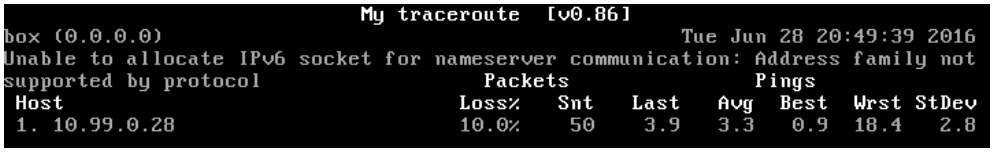

Figure 42 mtr results

The Analysis:

Packet lost >> 10% × 50 = 5 packets

Downtime >> 5 × 3.3 = **16.5** milliseconds

Small flavor:

After creating a Small virtual machine and making sure that the live migration is done successfully as showed by the following figure:

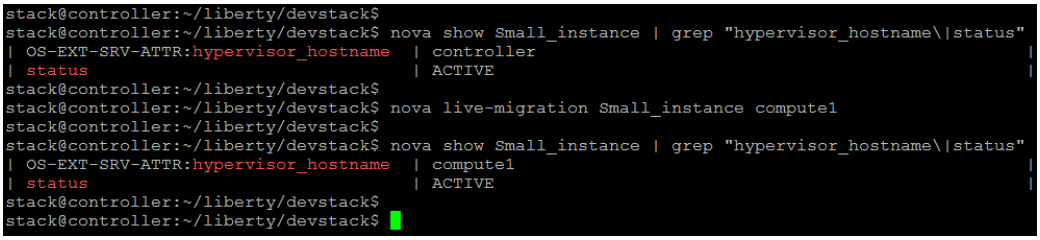

Figure 43 Before and After Live-Migration of Small Instance

The results of mtr:

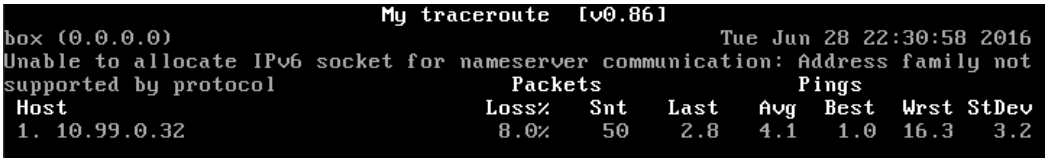

Figure 44 mtr results

The Analysis:

Packet lost >> 8% × 50 = 4 packets

Downtime >> 4 × 4.1 = **16.4** milliseconds

Medium flavor:

After creating a Medium virtual machine and making sure that the live migration is done successfully as showed by the following figure:

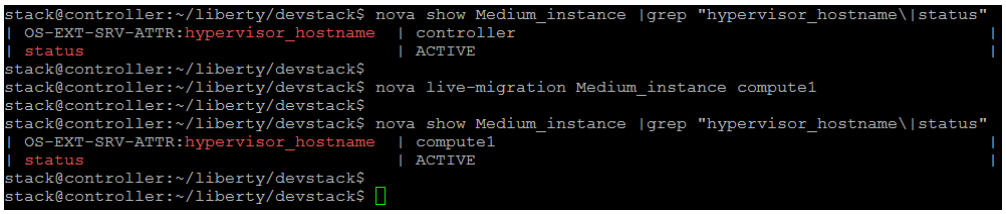

Figure 45 Before and After Live-Migration of Medium Instance

The results of mtr:

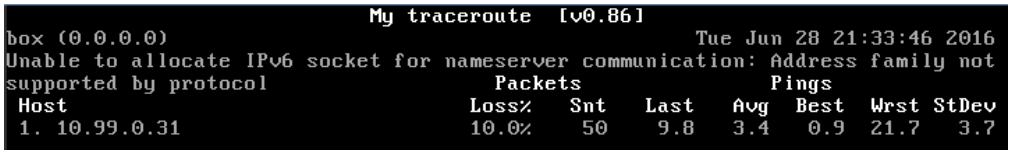

Figure 46 mtr results

The Analysis:

Packet lost >> 10% × 50 = 5 packets

Downtime >> 5 × 3.4 = **17** milliseconds

Large flavor:

After creating a Small virtual machine and making sure that the live migration is done successfully as showed by the following figure:

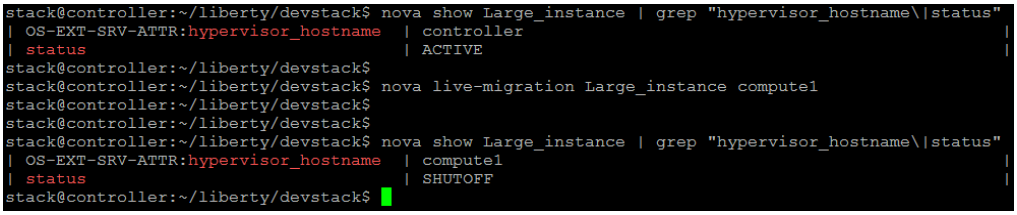

Figure 47 Before and After the Live-Migration of Largr instance

The VM (Large_instance) has failed during the live migration process: >> status of VM was SHUTOFF >> Live migration was not done successfully!

MTR output shows how unbelievable packets loss increasing was. The results of mtr:

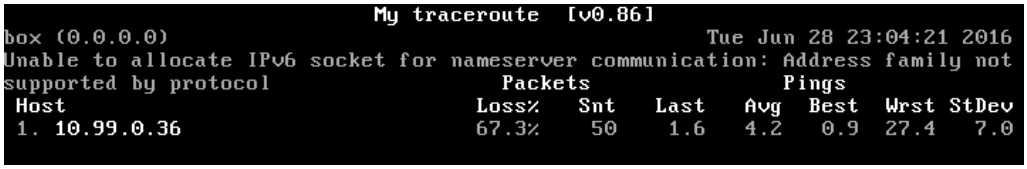

Figure 48 mtr results

It is obvious that the RAM and the Root Disk was more than can be suitable to use for Live-Migration, it has 1.5 GB of RAM and 5 GB of storage, as presented in the next figure:

Flavor Name	VCPUs	RAM	Root Disk
m1.nano	1	64MB	0GB
m1.micro	1	128MB	0GB
m1.tiny	1	512MB	1GB
Small	1	700MB	2GB
Medium	1	1GB	3GB
Large	1	1.5GB	5GB

Figure 49 Flavors Parameters before changes

So, the flavor could be managed by adjusting its resources, it will have 1 GB of RAM and 4 GB of storage:

Flavor Name	VCPUs	RAM	Root Disk
m1.nano	1	64MB	0GB
m1.micro	1	128MB	0GB
m1.tiny	1	512MB	1GB
Small	1	700MB	2GB
Medium	1	1GB	3GB
Large	1	1GB	4GB

Figure 50 Flavors Parameters After changes

Now it is possible to check whether the live migration is able to execute or not:

```
stack@controller:~/liberty/devstack$ nova show Large_instance | grep "hypervisor_hostname\|status"
| OS-EXT-SRV-ATTR:hypervisor_hostname  | controller                                                    |
| status                               | ACTIVE                                                        |
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$ nova live-migration Large_instance compute1
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$ nova show Large_instance | grep "hypervisor_hostname\|status"
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute1                                                      |
| status                               | ACTIVE                                                        |
stack@controller:~/liberty/devstack$
```

Figure 51 Live Migration of Large Instance

The results of mtr after making sure that the live migration is done correctly this time:

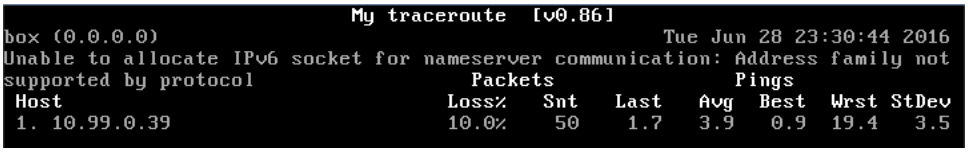

Figure 52 mtr results

The Analysis:

Packet lost >> 10% × 50 = 5 packets

Downtime >> 5 × 3.9 = **19.5** milliseconds

- **WIRESHARK**

All what could be needed to take all the measurements and metrics can be done via Wireshark. So now all the analysis can be performed accurately. Throughput, downtime, bandwidth, delay and jitter are the main metrics going to be measured.

Basically, the live migration can be concerned as a transferred data that's going to be moved from host to another through the internal network between the hosts, so the internal interfaces are already known. After that it is possible to apply this command:

"**sudo tcpdump -i eth1 -G 50 -W 1 -w pcapture.log"**

sudo >> there is a need of admin permission

tcpdump >> the magic tool that's going to be used

-i eth1 >> listen to specific interface

-G 50 –W 1 >> -G sec (rotate dump files every x seconds) and -W count (limit # of dump files)

-w pcapture.log >> writing the output into pcapture.log file

Flavor >> nano flavor

Downtime: when the tcp packets cannot be received which is going to lead to retransmission again.

No.	Time	Source	Destination	Protocol	Length	Info
22…	19.646452	192.168.100.10	192.168.100.21	SSHv2	65…	Client: Encrypted pa
22…	19.646490	192.168.100.21	192.168.100.10	TCP	66	22 → 42658 [ACK] Seq
22…	19.647409	192.168.100.10	192.168.100.21	SSHv2	29…	Client: Encrypted pa
22…	19.647478	192.168.100.10	192.168.100.21	SSHv2	8754	Client: Encrypted pa
22…	19.649481	192.168.100.21	192.168.100.10	TCP	78	22 → 42658 [ACK] Seq
22…	19.649551	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649577	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649598	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649616	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649635	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649653	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649672	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649690	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649710	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649729	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649747	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649766	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649785	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649804	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649822	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649841	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649860	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649880	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]
22…	19.649901	192.168.100.10	192.168.100.21	TCP	1514	[TCP Retransmission]

Figure 53 Downtime instance first method

Downtime = 19.650628 - 19.649551 = 0.001077 seconds = 1.077 milliseconds

Another way to measure the downtime:

In the following graph, there is a sharp rising of transferring (in our case sending) the packets through eth1 of controller node because the VM that´s wanted to be migrated was created in the controller node and its destination through the live migration process is compute1 node.

The process speeds up and the packets/seconds exceeds 2400, during this period of time the vm will be migrating. We can approximately read that time from the graph (from 17 to 22 seconds) which gives us the approximate migration time that's about 4 seconds.

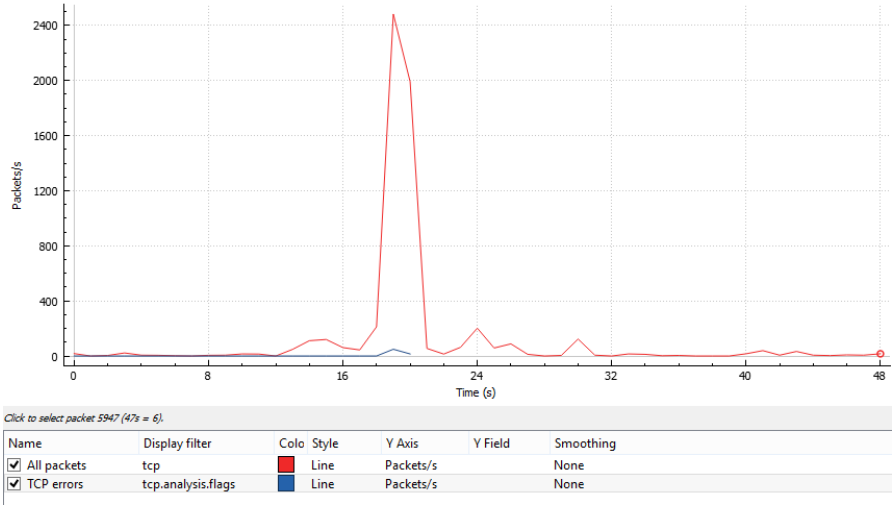

Figure 54 Downtime instance second method

By going deeply in the results, copying them from Wireshark and put them in excel sheet. Now it is possible to deal with the results with more details.

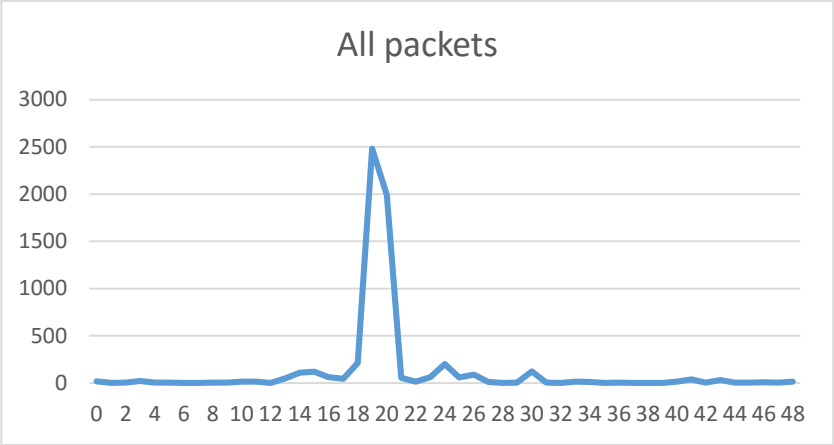

Figure 55 Live Migration time

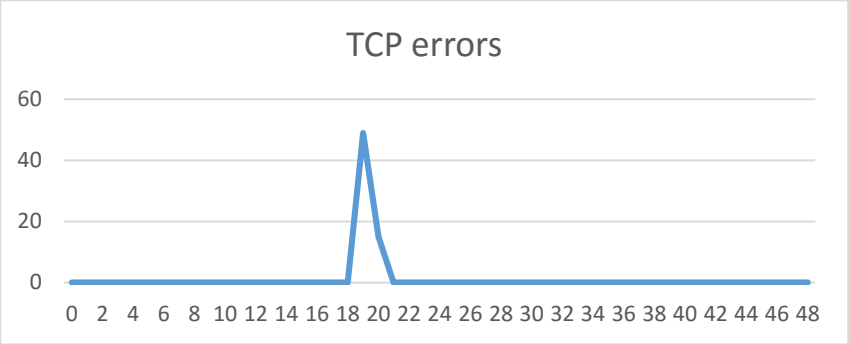

Figure 56 The downtime of Virtual machine during live migration

The graph above is showing the representation of lost TCP packets, in other hand these lost packets are occurred when the VM is down.

To measure the downtime, the time scale should be in milliseconds instead of seconds, as it is given in the next graph:

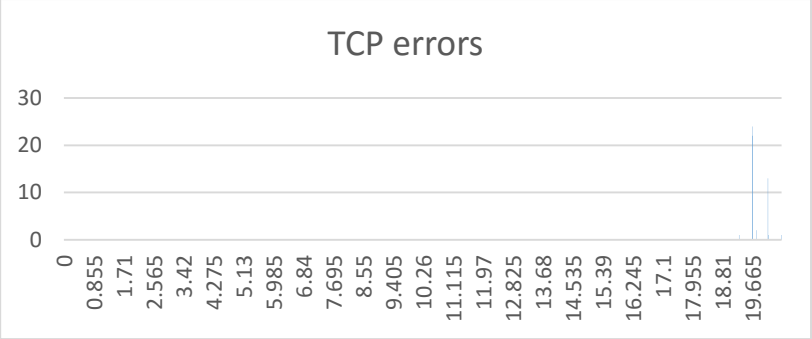

Figure 57 TCP error packets represent the downtime of migrated instance

3.1.3.2 **Comparisons**

The comparisons of the experimental results are made in two methods. The first one "the same infrastructure comparison" which is depending on the flavor of the virtual machine. In this method the results are taken by MTR tool which has measured the packet loos and the average time of each sending/receiving packet as shown in Figures [37, 40, 42, 44, 46, 52], then as a result, the down time of the migrated VM can be calculated. Therefore, deferent VMs with deferent flavor are migrated in the same infrastructure. The following table and graph show the comparison among these live-migration procedures:

Table 3 the same infrastructure comparison

Flavor Name	Packet loos	Avg. time	Down Time (ms)
Nano	8%	2.4	9.6
Micro	12%	2.6	15.6
Small	8%	4.1	16.4
Medium	10%	3.4	17
Large	10%	3.9	19.5

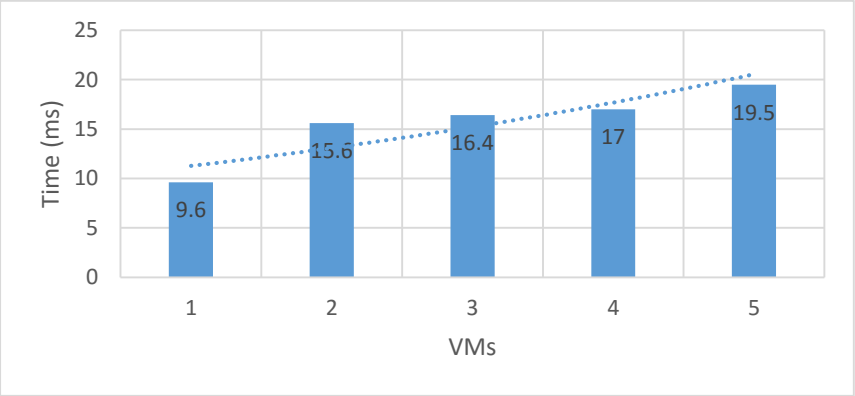

Figure 58 Down time of VMs with various flavors in the same infrastructure

It is obvious from Figure [59] that the down time is increasing exponentially. These results could be deferent because they depend on the resource consumption of the infrastructure.

The second method so-called “The different infrastructure comparison” which aims to compare the experimental results of this work with other experimental results reached after implementing the Live-Migration in specific infrastructure (Salfner et al., 2012).

The external experimental results show that a virtual machine with XenServer as its hypervisor and CentOS as its operating system, takes 89.73 ms to be migrated from host to another. The down time was 7.69 ms. As a conclusion, the downtime is only 8.6% of the overall migration time. Therefore, it is possible to compare the last value with the same value that can be calculated from Wireshark practical results.

Figure [55] shows the total live-migration time. It can be measured approximately as the time between the point that the line started to be unstable to the point that the line returns to its stability, so it is between 14 and 24.

In this practical test, the live-migration of the virtual machine with KVM as its hypervisor and Ubuntu as its operating system is almost 10 milliseconds, and the down time is 1.077 milliseconds. The percentage of the down time to the overall migration time is about 10.77 %. Table [4] shows these results as following:

Table 4 Data Comparison

Hypervisor type	Guest	Live-Migration (ms)	Down Time (ms)	Down Time %
KVM	Ubuntu	10	1.077	10.77%
XenServer	CentOS	89.73	7.69	8.6%

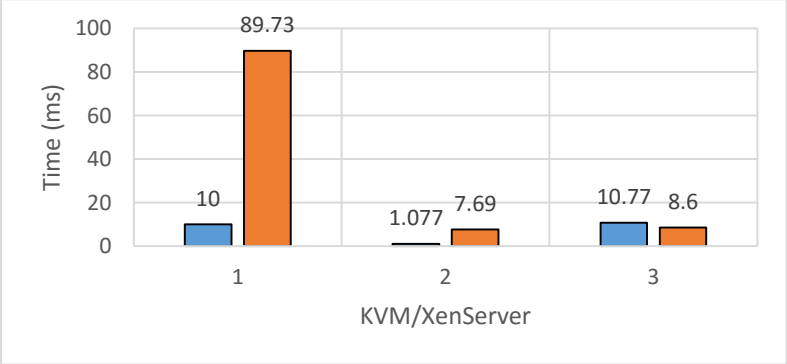

Figure 59 The comparison between two deferent VMs in deferent infrastructures

In the Figure [59], the first column presents the deferent values of the live-migration, while the second one shows the down time values, as well as the last column represents the percentage values of the down time.

As a conclusion, the percentage values of the down time are the best parameters to make the judgment of the live migration’s performance. Since the down time is not noticeable in the both cases, the performance of the migrated virtual machine whether if it has KVM or XenServer hypervisor is considered as a good performance.

3.1.3.3 Overall Discussion, Analysis and Conclusions

First of all, the experiments that were conducted demonstrate the performance of the VM during live migration. The first parameter considered to evaluate the performance is the total migration time. The total migration time is the time taken to migrate the VM from one host to another. The second parameter is the VM Down time, which is the time period during which VM doesn't reply to ICMP echo request from remote host, as shown in Figures [40, 42, 46, 48, 52]. Total downtime is counted by summing of all lost packets multiplied by request's interval. Therefore, as the results given, the down time differs from VM to another, it depends on the VM´s flavor, so it depends on the resources of the virtual machines, such as vCPU, RAM and the storage disk. The more workload the virtual machine has, the more down time takes.

The other network parameters that are described in previous section such as delay, jitter, bandwidth and packet loos give a comprehensive picture of the performance of the live migration. The first parameter, the delay that could be measured from end of the previous packet to the end of the current packet, the figure [53] shows that the delay between every two packet is proximately the same delay of the next two packets. As a result, the jitter is almost zero and the migration is done smoothly.

Chapter 4

4. Conclusion

The importance of compute nodes was obvious in this thesis as they are the workhorse of the cloud and the place where the applications will execute. Ensuring that all the customers' applications will be saved with non-stop running is the most important feature of live migration. Many of organizations are competing to have this mechanism implemented in their infrastructures and ready to use in order to meet the increasing demands and to reserve an advanced position in the market. On the other hands, OpenStack is useful in developing any software-as-a-service (SAAS) applications, for new developments or to improve existing solutions.

The future of Internet architectures will likely depend on cloud computing infrastructures, giving the world many advantages behind the virtualization, especially considering the emerging Network Function Virtualization (NFV) and Software Defined Networking (SDN) technologies. The cloud infrastructure will be then the responsible of the performance of the virtualized networks and their services.

Cloud computing has opened new horizons for organizations to meet increasing demand of computing and storage resources without huge upfront investment. Public and private Cloud infrastructures are two of the most common deployment models. Whilst public clouds led the trend of Cloud computing adoption, there is an increasing trend to build and manage private cloud infrastructures for several reasons with security, privacy, and data location management being the predominant concerns. Therefore, this thesis aimed to implement, operate, troubleshoot, and manage a secure and scalable private cloud infrastructure. Depending on open-source software so-called OpenStack technology, as well as, it presented the procedure of designing and implementing cloud system by using DevStack which is a quick hands-on OpenStack. In addition, the reader is provided with important knowledge and understanding about deploying and managing private cloud infrastructure with DevStack practically. Moreover in the thesis, there is a description about how OpenStack deals with multi-tenant network virtualization and evaluates the Live Migration of the instances among the compute nodes.

Many enterprise organizations and their software developers are turning to OpenStack for its open APIs, flexible architecture and large commercial ecosystem to compete in a completely new paradigm of software development and deployment. Many of these enterprises also operate large virtualized infrastructures to support a legacy of mission-critical, scale-up applications and enterprise databases. They value the compute, network, storage and management technologies of these virtualized infrastructures. But they are also encountering new use cases that demand agility.

OpenStack software offers an alternative solution of public clouds that allows IT organizations to maintain internal control over their systems, applications, and data. Businesses and independent service providers are using this comprehensive cloud platform to set up private cloud networks that mirror the services of the large public clouds. By establishing a cloud-like control plane for managing their compute, network, and storage infrastructure, they are able to address the demands of their business units through self-service provisioning and on-demand scaling.

Developed use case results show that the data center manager can make a live migration of virtual machines without any noticeable downtime. Basically this means the concept of cloud computing is proved and OpenStack was implemented successfully.

Appendix A

Environment Setup

After this phase of configuration is done, the infrastructure of the work will have three servers (nodes) configured with specific hardware requirements and will be running Ubuntu 14.04 LTS as their operating system. The first node will be the responsible of the operations among the other nodes and is called the controller. The second node is the core of the processing and its name is the compute node. The network node is the final node and its mission is managing the network connections among the nodes. The following procedure describes how to setup the infrastructure step by step:

1. Download your suitable version of VirtualBox from here.
2. Install VirtualBox on your computer.
3. To create new virtual machine open VirtualBox and go to Machine >> New...
4. Write the name of the virtual machine (in our case we need to create three VMs, their names could be: controller, network and compute1), choose the type which is Linux and the version is Ubuntu (64-bit)
5. Memory size (RAM): controller with 4GB and both of network and compute1 with 2GB.
   NOTE: Make sure that the memory size of the host of all these VMs is enough, my recommendation: never leave your host with too little RAM. If you have 4 GB on host, don't assign more than 2 GB for VMs (in our case the host has 16GB RAM, 8 GB for VMs and 8 GB for the host´s tasks)
6. Choose create a virtual hard disk now (following step 3). Fill all the fields as it was mentioned in steps 4 & 5 and press on create.

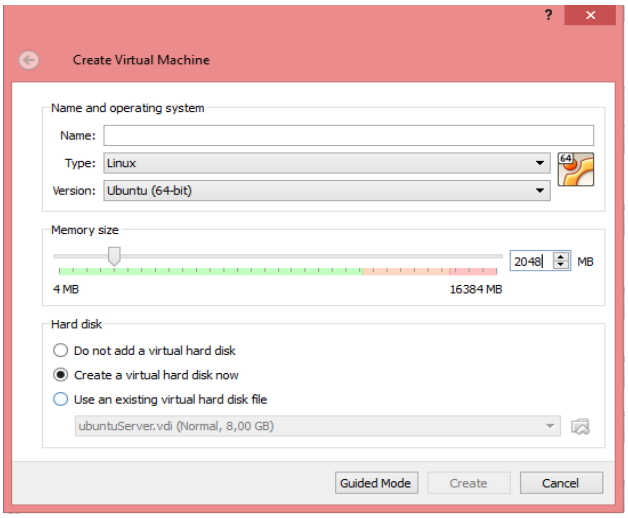

Figure 60 Create Virtual Machine

7. Select the file location that you have enough spaces in to store your VM
8. Choose a suitable size for the VM and HD type is VDI (VirtualBox Drive Image), the minimum hard disk requirements is 5GB (controller & network nodes) and 10GB for compute node.

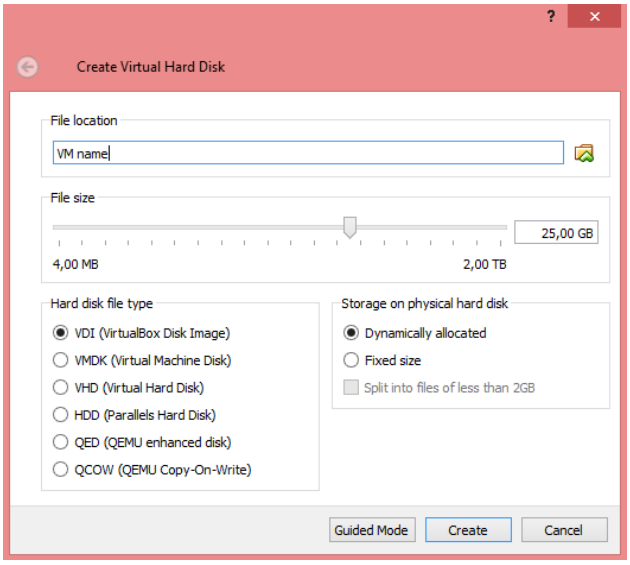

Figure 61 Create Virtual Hard Disk

9. Assign 2 vCPU for each VM when it is powered off, select the VM (starting with controller, compute and network VMs afterword), Sittings >> System >> Processor tab:

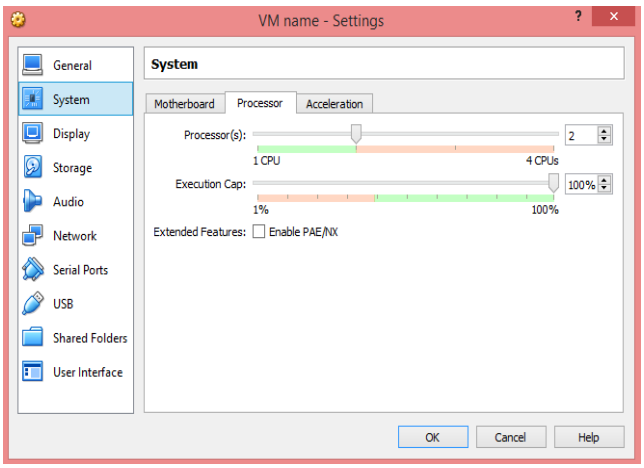

Figure 62 Assigning vCPU

10. After repeating all the previous steps on Compute and Network node, you should have something like:
    - Controller Node: 2 vCPU, 4 GB RAM, and 5 GB storage (minimum)
    - Network Node: 2 vCPU, 2 GB RAM, and 5 GB storage (minimum)
    - Compute Node: 2 vCPU, 2 GB RAM, and 10 GB storage (minimum)
11. All the VMs now are ready, power them on and install Ubuntu server OS (one by one).

NOTE: you can install any distribution you would like but the lighter one you choose, the more performance you get (in our case Ubuntu server 14.04 minimal (Minimal CD)), following the link >> under 64-bit PC (amd64, x86_64) choose Ubuntu 14.04 LTS "Trusty Tahr" and download this distribution.
Select the VM (starting with Controller), Start >> choose the image file (the image that you already downloaded it in step 11) >> press start.

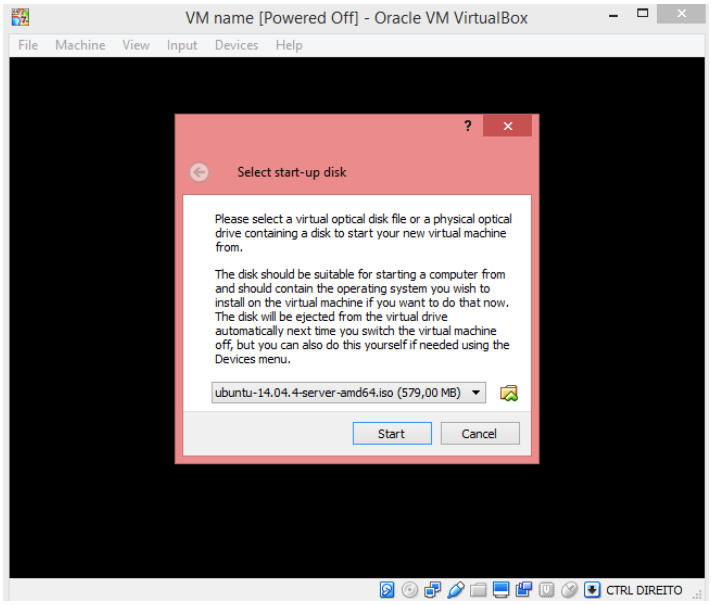

Figure 63 Select start-up disk

12. By this level, you should have controller, compute and network VMs with Ubuntu server 14.04 installed and running.
13. Now it is the time to connect all the VMs together through creating 3 different networks.

Network environment

There is a need to create 3 networks before installing OpenStack/DevStack, these networks are necessary to be done, then they can be used by DevStack:

- Management Network (192.168.100.0/24) among the nodes
- Tunnel Network (192.168.110.0/24) to give the VMs that are going to be created inside the compute nodes a specific network layer (VxLAN), so they can connect with each other. Note that in our case Controller node plays as controller and compute node at the same time.
- External Network (10.100.1.0/24) which will provide the ability to access the outside of OpenStack environment, for example internet, it is for the hosts themselves (controller, compute and network) and the VMs in the compute nodes afterword.

The following diagram shows how the controller, compute and network nodes are connecting using three different networks.

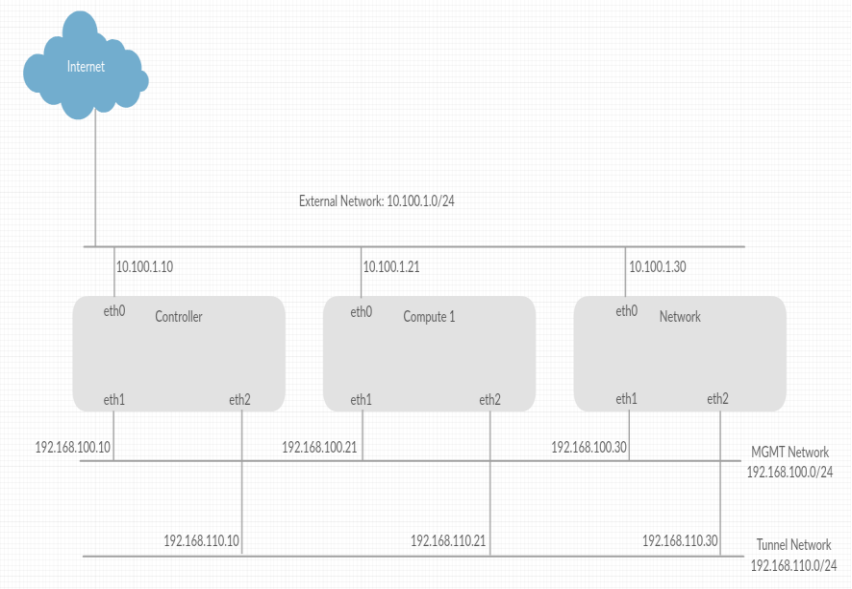

Figure 64 Network Digram

1. Create NAT network without DHCP with CIDR 10.100.1.0/24 (external network/internet) File >> preferences >> network >> add new NAT network.

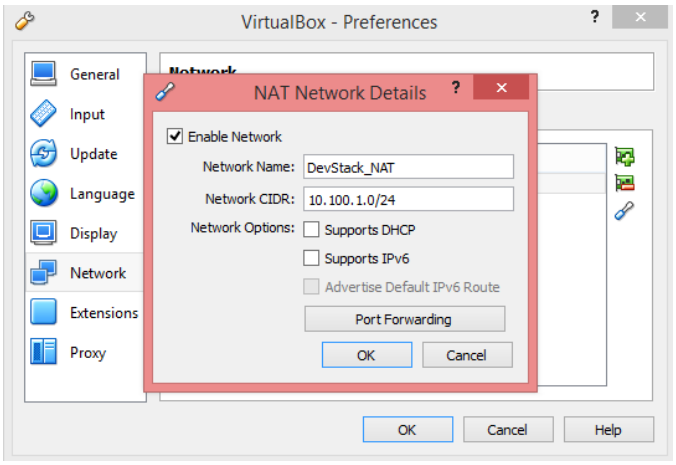

Figure 65 NAT Network Details

2. From the same path switch to the other tab to create two host only networks vboxnet1 and vboxnet2 with CIDR (192.168.100.0/24 – MGMT e 192.168.110.0/24 - Tunnel)

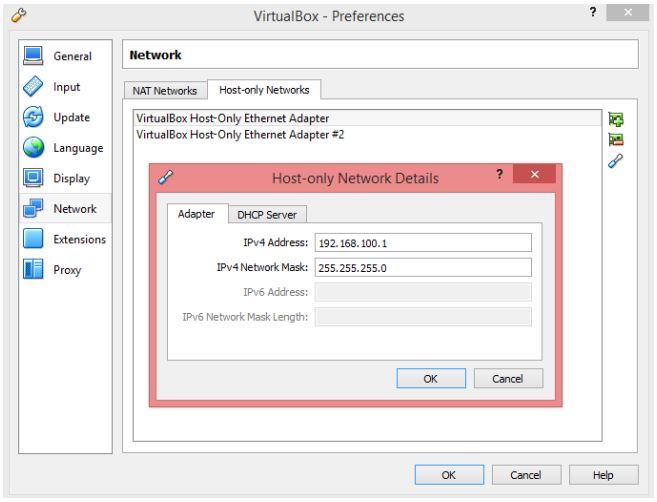

Figure 66 Host only Network Details

3. Each VM has 3 different IP:
   Select the (Controller node first) Sittings >> Network >> Adapter1 (Attached to NAT network, Name is DevStack_NAT that we already did in 1).

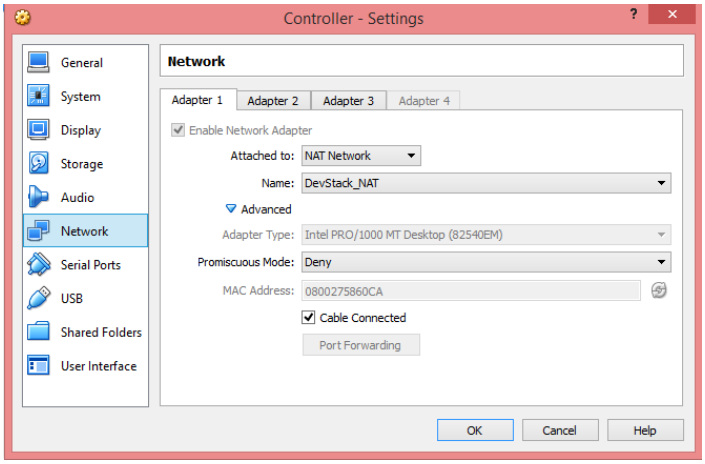

Figure 67 Cntroller´s network settings - Adapter 1

4. Adapter2 and Adapter3 are attached to Host Only network, choose the Name that is created in step 2 (vboxnet1 and vboxnet2).
   Note: Adapter1 and Adapter3 should be promiscuous.

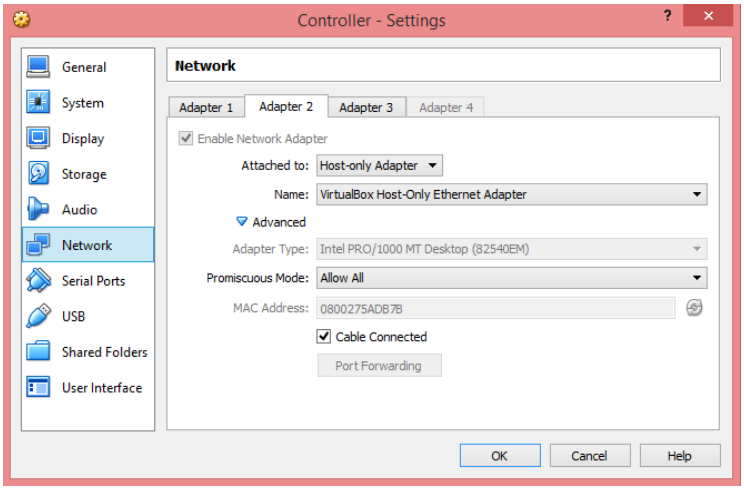

Figure 68 Controller's network settings - Adapter 2

5. Repeat step 4 to configure Adapter 3 as well (choose the other Host-Only Network).
6. Once the OS is installed, each VM should include the configuration below. By editing interfaces file: (Applied in all the nodes)
7. sudo vi /etc/network/interfaces
   - Controller:

```
# The loopback network interface
auto lo
iface lo inet loopback

# eth1: MGMT interface
auto eth1
iface eth1 inet static
address 192.168.100.10
netmask 255.255.255.0

# eth2: Tunnel (Data) interface
auto eth2
iface eth2 inet static
address 192.168.110.10
netmask 255.255.255.0
```

Figure 69 controller /etc/network/interfaces

- Compute1 has the same configuration with different IPs:
  Eth1= 192.168.100.21
  Eth2= 192.168.110.21
- Network also needs the same steps with:
  Eth1=192.168.100.30
  Eth2=192.168.110.30

DevStack Installation

The installation process will end up with OpenStack with Open Virtual Switch (OVS) and DVR installed and configured. After this level you can enter the dashboard and check the ability of creating new networks, routers and instances. The installation Process as following:

1. VM credentials: each VM needs new user with password for DevStack installation, the following commands are applied on all the nodes (Controller, compute1 and network).
    - Username: stack // $ sudo useradd stack // adduser
    - Password: stack // $ sudo passwd stack
2. After switching to “stack” user, Update apt-get all packages, run:
    - sudo apt-get update
    - sudo apt-get upgrade
3. Install git on Ubuntu using the following command:
    - sudo apt-get install git
4. Change the privilege:
    - sudo su -c 'echo "stack ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers'
5. Download DevStack project from github in specific directory:
    - mkdir ~/liberty
    - cd ~/liberty
    - git clone https://git.openstack.org/openstack-dev/devstack -b stable/liberty
6. Create a file of configuration for each virtual machine local.conf
    - cd ~/liberty/devstack~
    - vi local.conf

    Then, Put the configuration that the devstack script uses in local.conf DevStack configuration is modified via the file local.conf. It is a modified INI format file that introduces a meta-section header to carry additional information regarding the configuration files to be changed.

    A sample is provided in devstack/samples

    The new header is similar to a normal INI section header but with double brackets ([[...]]) and two internal fields separated by a pipe (|). Note that there are no spaces between the double brackets and the internal fields. Likewise, there are no spaces between the pipe and the internal fields, for example:

    ```
    [[post-config|$NOVA_CONF]]
    [DEFAULT]
    use_syslog = True
    [osapi_v3]
    enabled = False
    ```

    For more information, you can see the original document from Devstack here.

Now, follow the Appendix B and then come back to continue the installation procedure.

7. Run the stack.sh script inside devstack directory (starting with Controller). This script takes a while to run. It installs Mysql, RabbitMQ, compiles the OpenStack binaries and set up a whole bunch of stuff. So type:
    - cd ~/liberty/devstack
    - ./stack.sh

   As improvement that you have an Openstack working on your environment, an IP of dashboard will be printed at the end of executing last command on your console, like:

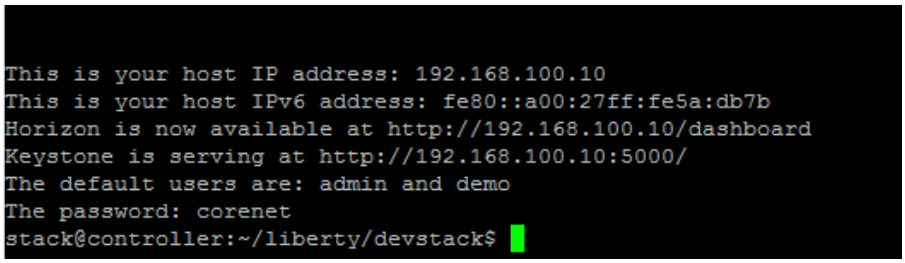

Figure 70 Controller - stack.sh

8. OpenStack credentials (you can use them now to lunch the dashboard, but wait a little bit for compute1 and network finish executing stack.sh)
    - Username: admin or demo
    - Password: corenet
9. After you repeat all the previous steps in Compute1 and Network VMs, you should see an IP of each one of them, like:

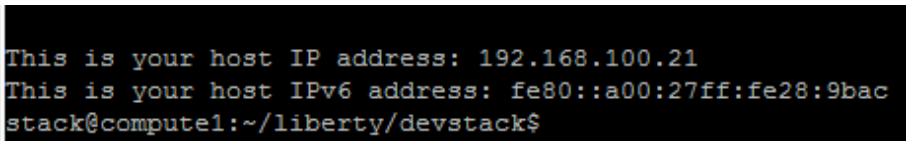

Figure 71 Compute1 - stack.sh

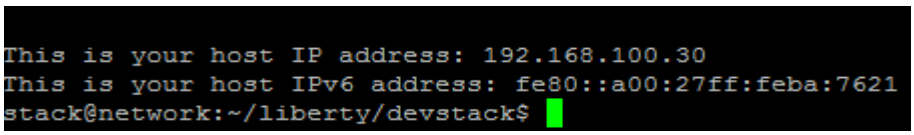

Figure 72 Network - stack.sh

10. By this phase, you should have the Openstack with openvswitch and DVR configured “Depending on the local.conf of each VM that should have different configuration form each other ” (description in other sections)

11. When everything is installed and working, it is possible to put the machine into hibernation or turn off the infrastructure with:
    - ./unstack.sh
12. After executing the last command, all the services will be unattached, so if you want to retrieve OpenStack services, you need to type the following commands on controller, after they finish and only when they finish executing, repeat these commands on Copmute1 and then on Network nodes.
    - sudo reboot
    - cd ~/liberty/devstack
    - ./stack.sh

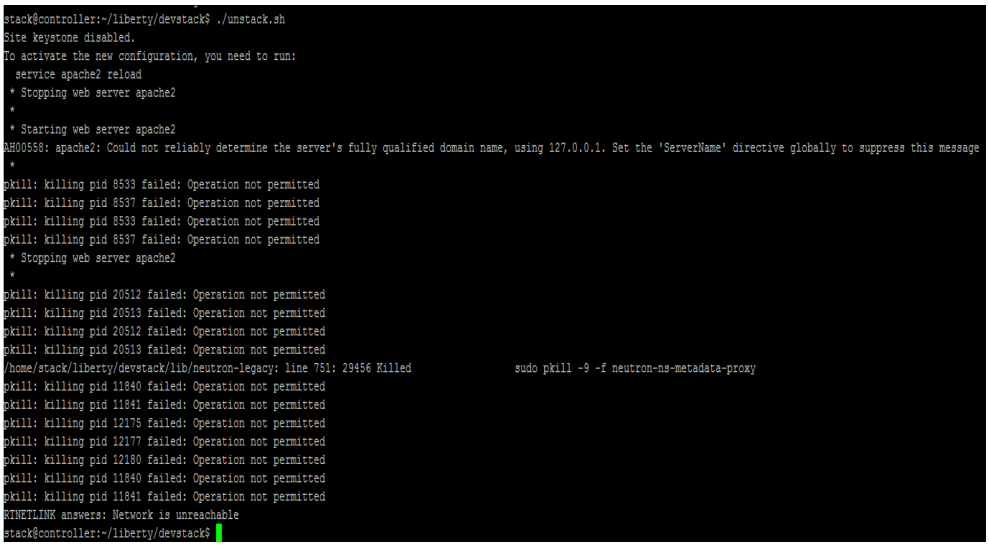

Figure 73 Controller - unstack.sh

13. You can clean up all the files that Devstack installed (Use this command when you want to reinstall everything again), run:
    - ./clean.sh

Live Migration Configuration

Adding certain configurations, simply we need to configure a shared file (create this shared file on controller and mount it from the compute nodes). Imagine that all the instances from different compute nodes will be located in this shared file logically, so now we will have the ability to migrate the instance from side to another by testing that with command line.

1. All commands are run from the controller. First of all, create hostnames - needed for instances migration:
    - cat /etc/hosts
    - echo "192.168.100.10 Controller Devstack-liberty-Controller" | sudo tee -a /etc/hosts
    - echo "192.168.100.21 Compute1 Devstack-liberty-Compute1" | sudo tee -a /etc/hosts

- echo "192.168.100.30 Network Devstack-liberty-network" | sudo tee -a /etc/hosts

Secondly, generate the RSA key - Running on the home of stack user:

- ssh-keygen -t rsa

Thirdly, copy the keys to the root user:

- cat ~/.ssh/id_rsa.pub | tee –a ~/.ssh/authorized_keys
- sudo mkdir -p /root/.ssh
- sudo cp ~/.ssh/id* /root/.ssh
- sudo cp ~/.ssh/authorized* /root/.ssh

2. Replicate the configuration to the other VMs. The idea is to give each node the public keys of the other nodes (from the controller node).
    - ssh stack@compute1 "sudo mkdir -p ~/.ssh && sudo mkdir -p /root/.ssh"
    - scp ~/.ssh/* compute1:~/.ssh/
    - ssh stack@compute1 "sudo cp ~/.ssh/id* /root/.ssh"
    - ssh stack@compute1 "sudo cp ~/.ssh/authorized* /root/.ssh"
    - ssh stack@network "sudo mkdir -p ~/.ssh && sudo mkdir -p /root/.ssh"
    - scp ~/.ssh/* network:~/.ssh/
    - ssh stack@network "sudo cp ~/.ssh/id* /root/.ssh"
    - ssh stack@network "sudo cp ~/.ssh/authorized* /root/.ssh"

3. Test the accessibility
    A. Controller >> compute1
        - ssh compute1
        - echo "192.168.100.10 Controller Devstack-liberty-Controller" | sudo tee -a /etc/hosts
        - echo "192.168.100.21 Compute1 Devstack-liberty-Compute1" | sudo tee -a /etc/hosts
        - echo "192.168.100.30 Network Devstack-liberty-network" | sudo tee -a /etc/hosts
        - ssh Devstack-liberty-Compute1
        - exit
        - ssh root@compute1
        - exit

    B. Controller >> Network
        - ssh network
        - echo "192.168.100.10 Controller Devstack-liberty-Controller" | sudo tee -a /etc/hosts

- echo "192.168.100.21 Compute1 Devstack-liberty-Compute1" | sudo tee -a /etc/hosts
- echo "192.168.100.30 Network Devstack-liberty-network" | sudo tee -a /etc/hosts
- exit
- ssh root@network
- exit

C. Compute1 >> controller

- ssh controller
- exit
- ssh root@controller
- exit

D. Network >> controller

- ssh controller
- exit
- ssh root@controller
- exit

NFS configuration

NFS (Network File System) configuration used to create a shared file among the compute nodes which will be created in the controller and mount this file from the compute nodes (in our case the controller plays as a compute node at the same time).

In the controller node:

- sudo apt-get install nfs-kernel-server

1. Edit the file / etc / exports and add the folder to share:
   - echo "/opt/openstack/data/nova/instances 192.168.100.0/24(rw,fsid=0,insecure,no_subtree_check,async,no_root_squash)" | sudo tee -a /etc/exports
   - more /etc/exports (to make sure that the modification is done)
2. Add permission to execute the folder:
   - chmod o+x /opt/openstack/data/nova/instances
3. Activate the service and confirm that the folder is being shared:
   - sudo /etc/init.d/nfs-kernel-server restart
   - sudo showmount -e controller
4. Confirm that the folder is created and create a test file:
   - ls -la /opt/openstack/data/nova/instances
   - sudo touch /opt/openstack/data/nova/instances/created_in_controller

In the compute1 node:

1. Install the client of NFS
   - sudo apt-get install nfs-common
2. Confirm that we have visibility of the shared folder:
   - sudo showmount -e controller

3. Confirm the contents of the folder before and after connecting:
   - ls -la /opt/openstack/data/nova/instances
   - sudo mount controller:/opt/openstack/data/nova/instances /opt/openstack/data/nova/instances
   - ls -la /opt/openstack/data/nova/instances
   - sudo umount /opt/openstack/data/nova/instances
4. Mount the NFS folder from during installing devstack the compute node (via script):
   - touch ~/liberty/devstack/local.sh
   - echo -e '#!/usr/bin/env bash \nsudo mount controller:/opt/openstack/data/nova/instances /opt/openstack/data/nova/instances' | tee ~/liberty/devstack/local.sh
   - chmod 744 ~/liberty/devstack/local.sh
   - more local.sh
5. Reboot the controller and compute1:
   - sudo reboot

For Testing:

1. Run Openstack variables
   - . openrc admin
2. Sometimes and for uncertain reason the default router that is created could have down state, so to avoid this problem and put yourself in the safe side, type:
   - neutron router-show router1
   - neutron router-update --admin-state-up false router1
   - neutron router-update --admin-state-up true router1
3. Create an instance:
   - nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.micro testec1

```
stack@controller:~/liberty/devstack$ nova boot --image cirros-0.3.4-x86_64-uec --flavor m1.micro testec1
+--------------------------------------+--------------------------------------------------------------------+
| Property                             | Value                                                              |
+--------------------------------------+--------------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                             |
| OS-EXT-AZ:availability_zone          |                                                                    |
| OS-EXT-SRV-ATTR:host                 | -                                                                  |
| OS-EXT-SRV-ATTR:hostname             | testec1                                                            |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                                                                  |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000008                                                  |
| OS-EXT-SRV-ATTR:kernel_id            | 9ffeabd5-510e-4295-86bd-9966bdfa9b94                               |
| OS-EXT-SRV-ATTR:launch_index         | 0                                                                  |
| OS-EXT-SRV-ATTR:ramdisk_id           | bb622355-38c7-4a95-82e8-c4285c5996d6                               |
| OS-EXT-SRV-ATTR:reservation_id       | r-b29gox3x                                                         |
| OS-EXT-SRV-ATTR:root_device_name     | -                                                                  |
| OS-EXT-SRV-ATTR:user_data            | -                                                                  |
| OS-EXT-STS:power_state               | 0                                                                  |
| OS-EXT-STS:task_state                | scheduling                                                         |
| OS-EXT-STS:vm_state                  | building                                                           |
| OS-SRV-USG:launched_at               | -                                                                  |
| OS-SRV-USG:terminated_at             | -                                                                  |
| accessIPv4                           |                                                                    |
| accessIPv6                           |                                                                    |
| adminPass                            | RbEtQpG8Q7bk                                                       |
| config_drive                         |                                                                    |
| created                              | 2016-05-11T16:07:53Z                                               |
| flavor                               | m1.micro (84)                                                      |
| hostId                               |                                                                    |
| id                                   | f28695ac-d5e3-42fa-9b10-e86598995b73                               |
| image                                | cirros-0.3.4-x86_64-uec (0b6d31f7-007c-4c12-8449-4a287e00f3a9)     |
| key_name                             | -                                                                  |
| metadata                             | {}                                                                 |
| name                                 | testec1                                                            |
| os-extended-volumes:volumes_attached | []                                                                 |
| progress                             | 0                                                                  |
| security_groups                      | default                                                            |
| status                               | BUILD                                                              |
| tenant_id                            | 7cf03a03f12448d89a7a32b4e0ff8f3d                                   |
| updated                              | 2016-05-11T16:07:53Z                                               |
| user_id                              | 34934c8750aa42839a763e3f7684c553                                   |
+--------------------------------------+--------------------------------------------------------------------+
stack@controller:~/liberty/devstack$
```

Figure 74 create instance

4. Display information about a specific instance, (here we are caring about the hostname of this instance)
   - nova show testec1

```
stack@controller:~/liberty/devstack$ nova show testec1
+--------------------------------------+----------------------------------------------------------------+
| Property                             | Value                                                          |
+--------------------------------------+----------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                         |
| OS-EXT-AZ:availability_zone          | nova                                                           |
| OS-EXT-SRV-ATTR:host                 | controller                                                     |
| OS-EXT-SRV-ATTR:hostname             | testec1                                                        |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | controller                                                     |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000008                                              |
| OS-EXT-SRV-ATTR:kernel_id            | 9ffeabd5-510e-4295-86bd-9966bdfa9b94                           |
| OS-EXT-SRV-ATTR:launch_index         | 0                                                              |
| OS-EXT-SRV-ATTR:ramdisk_id           | bb622355-38c7-4a95-82e8-c4285c5996d6                           |
| OS-EXT-SRV-ATTR:reservation_id       | r-b29gox3x                                                     |
| OS-EXT-SRV-ATTR:root_device_name     | /dev/vda                                                       |
| OS-EXT-SRV-ATTR:user_data            | -                                                              |
| OS-EXT-STS:power_state               | 1                                                              |
| OS-EXT-STS:task_state                | -                                                              |
| OS-EXT-STS:vm_state                  | active                                                         |
| OS-SRV-USG:launched_at               | 2016-05-11T16:08:05.000000                                     |
| OS-SRV-USG:terminated_at             | -                                                              |
| accessIPv4                           |                                                                |
| accessIPv6                           |                                                                |
| config_drive                         |                                                                |
| created                              | 2016-05-11T16:07:53Z                                           |
| flavor                               | m1.micro (84)                                                  |
| hostId                               | 1be7654bcbda8432bb52631940755a51ecc556b48667ef9942651cd4       |
| id                                   | f28695ac-d5e3-42fa-9b10-e86598995b73                           |
| image                                | cirros-0.3.4-x86_64-uec (0b6d31f7-007c-4c12-8449-4a287e00f3a9) |
| key_name                             | -                                                              |
| metadata                             | {}                                                             |
| name                                 | testec1                                                        |
| os-extended-volumes:volumes_attached | []                                                             |
| private network                      | 10.99.0.11                                                     |
| progress                             | 0                                                              |
| security_groups                      | default                                                        |
| status                               | ACTIVE                                                         |
| tenant_id                            | 7cf03a03f12448d89a7a32b4e0ff8f3d                               |
| updated                              | 2016-05-11T16:08:06Z                                           |
| user_id                              | 34934c8750aa42839a763e3f7684c553                               |
+--------------------------------------+----------------------------------------------------------------+
stack@controller:~/liberty/devstack$
```

Figure 75 instance location before Live Migration

So, the VM was on controller

5. Perform the Live Migration:
   - nova live-migration testec1
6. Check the changes on “testec1” instance
   - nova show testec1

```
stack@controller:~/liberty/devstack$ nova live-migration testec1
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$
stack@controller:~/liberty/devstack$ nova show testec1
+--------------------------------------+----------------------------------------------------------------+
| Property                             | Value                                                          |
+--------------------------------------+----------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                         |
| OS-EXT-AZ:availability_zone          | nova                                                           |
| OS-EXT-SRV-ATTR:host                 | compute1                                                       |
| OS-EXT-SRV-ATTR:hostname             | testec1                                                        |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute1                                                       |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000008                                              |
| OS-EXT-SRV-ATTR:kernel_id            | 9ffeabd5-510e-4295-86bd-9966bdfa9b94                           |
| OS-EXT-SRV-ATTR:launch_index         | 0                                                              |
| OS-EXT-SRV-ATTR:ramdisk_id           | bb622355-38c7-4a95-82e8-c4285c5996d6                           |
| OS-EXT-SRV-ATTR:reservation_id       | r-b29gox3x                                                     |
| OS-EXT-SRV-ATTR:root_device_name     | /dev/vda                                                       |
| OS-EXT-SRV-ATTR:user_data            | -                                                              |
| OS-EXT-STS:power_state               | 1                                                              |
| OS-EXT-STS:task_state                | -                                                              |
| OS-EXT-STS:vm_state                  | active                                                         |
| OS-SRV-USG:launched_at               | 2016-05-11T16:08:05.000000                                     |
| OS-SRV-USG:terminated_at             | -                                                              |
| accessIPv4                           |                                                                |
| accessIPv6                           |                                                                |
| config_drive                         |                                                                |
| created                              | 2016-05-11T16:07:53Z                                           |
| flavor                               | m1.micro (84)                                                  |
| hostId                               | ca911e5861c8ebe5b7cad35ed97f29fd3df2dc31aa3cf0e62e79fbc7       |
| id                                   | f28695ac-d5e3-42fa-9b10-e86598995b73                           |
| image                                | cirros-0.3.4-x86_64-uec (0b6d31f7-007c-4c12-8449-4a287e00f3a9) |
| key_name                             | -                                                              |
| metadata                             | {}                                                             |
| name                                 | testec1                                                        |
| os-extended-volumes:volumes_attached | []                                                             |
| private network                      | 10.99.0.11                                                     |
| progress                             | 19                                                             |
| security_groups                      | default                                                        |
| status                               | ACTIVE                                                         |
| tenant_id                            | 7cf03a03f12448d89a7a32b4e0ff8f3d                               |
| updated                              | 2016-05-11T16:13:22Z                                           |
| user_id                              | 34934c8750aa42839a763e3f7684c553                               |
```

Figure 76 instance location After Live Migration

As we can see the Live Migration is done and the instance “testec1” has migrated from controller to compute1 successfully.

Appendix B

- The content of local.conf file which has all the configurations needed to get DevStack working correctly with Openvswitch, DVR and Live Migration.
- Controller >> cd “to DevStack directory ” in our case >> cd liberty/devstack >> sudo vi local.conf (and then press insert button in your keyboard, so you are now able to insert your special configuration).
- Copy the specific text >> Past it there.
- To save the file and exit, type: esc >> :x
- Replicate the same steps for Compute1 and Network nodes with their different configurations.
- **Local.conf of Controller:**

```
# Controller Config - Com br-ex, br-tun e NFS
# Versão 5
# 20160506

[[local|localrc]]
DEST=/opt/openstack
DATA_DIR=$DEST/data
LOGFILE=$DATA_DIR/logs/stack.log
SCREEN_LOGDIR=$DATA_DIR/logs
MYSQL_PASSWORD=corenet
RABBIT_PASSWORD=corenet
SERVICE_TOKEN=corenet
SERVICE_PASSWORD=corenet
ADMIN_PASSWORD=corenet
SERVICE_DIR=$DEST/status
SUBUNIT_OUTPUT=$DEST/devstack.subunit
DATABASE_TYPE=mysql
PIP_UPGRADE=True
#ENABLE_IDENTITY_V2=False

#RECLONE=yes

# Como está a correr em 14.04 não é preciso forçar a instalação
#FORCE=yes

# Descomentar para se poder trabalhar sem internet
#OFFLINE=True

# Permitir nested VM (aceleração de VM dentro de VM)
LIBVIRT_TYPE=kvm
libvirt_cpu_mode=none
compute_driver=libvirt.LibvirtDriver

# Enable Logging
VERBOSE=True
LOG_COLOR=True

disable_service n-net
```

```
disable_service cinder c-api c-vol c-sch c-bak
#enable_service cinder
#enable_service c-api
#enable_service c-vol
#enable_service c-sch
#enable_service c-bak
enable_service neutron
enable_service q-svc
enable_service q-meta
enable_service q-agt
#enable_service q-dhcp
enable_service q-l3
#enable_service swift3

#Cinder
VOLUME_GROUP="stack-volumes"
VOLUME_NAME_PREFIX="volumes-"
VOLUME_BACKING_FILE_SIZE=7250M


# HOST_IP=[actual address of your adapter (eth0, etc.)
# FLOATING_RANGE=[range of floating addresses.. externally accessible from OpenStack]
*Does not need to exist before stacking
# PUBLIC_NETWORK_GATEWAY=[gateway address of public network that will be used to
connect internal OpenStack networks with external networks.]
# FIXED_RANGE=[internal network range for VMs to use]

# # controller ip
HOST_IP=192.168.100.10
HOST_IP_IFACE=eth1
MULTI_HOST=True
IP_VERSION=4
Q_META_DATA_IP=192.168.100.10
TUNNEL_ENDPOINT_IP=192.168.110.10
Q_DVR_MODE=dvr

# Configuração de rede
## Neutron options
Q_USE_SECGROUP=True
Q_L3_ENABLED=True
FLOATING_RANGE=10.100.1.0/24
FIXED_RANGE=10.99.0.0/24
Q_FLOATING_ALLOCATION_POOL=start=10.100.1.128,end=10.100.1.200
PUBLIC_NETWORK_GATEWAY=10.100.1.1
NETWORK_GATEWAY=10.99.0.1

# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_INTERFACE=eth0
PUBLIC_PHYSICAL_NETWORK=public
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex

# Configuracao para Live Migration
#FORCE_CONFIG_DRIVE=False
```

```
#config_drive_format = vfat

[[post-config|$NOVA_CONF]]
[DEFAULT]
default_ephemeral_format = ext2
force_config_drive = False
config_drive_format=vfat
[libvirt]
live_migration_uri = qemu+ssh://stack@%s/system
#live_migration_flag += VIR_MIGRATE_TUNNELLED
#virt_type = qemu
# cpu_mode = custom
# virt_type = kvm



[[post-config|$NEUTRON_CONF]]
[DEFAULT]
router_distributed=True
[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ml2]
type_drivers=flat,vlan,vxlan
tenant_network_types=vxlan
mechanism_drivers=openvswitch,l2population

[ml2_type_vxlan]
vni_ranges=1000:1999

# IP do interface Tunnel
[ovs]
local_ip=192.168.110.10

[agent]
tunnel_types=vxlan
l2_population=True
enable_distributed_routing=True

[[post-config|$Q_L3_CONF_FILE]]
[DEFAULT]
agent_mode=dvr
router_delete_namespaces=True

[[post-config|$Q_DHCP_CONF_FILE]]
[DEFAULT]
dhcp_delete_namespaces=True
```

- **Local.conf of compute1:**

```
# Compute1 Config  - Com br-ex, br-tun e NFS (via local.sh)
# Versão 5
# 20160506

[[local|localrc]]
DEST=/opt/openstack
DATA_DIR=$DEST/data
LOGFILE=$DATA_DIR/logs/stack.log
```

```
SCREEN_LOGDIR=$DATA_DIR/logs
MYSQL_PASSWORD=corenet
RABBIT_PASSWORD=corenet
SERVICE_TOKEN=corenet
SERVICE_PASSWORD=corenet
ADMIN_PASSWORD=corenet
SERVICE_DIR=$DEST/status
SUBUNIT_OUTPUT=$DEST/devstack.subunit
DATABASE_TYPE=mysql
PIP_UPGRADE=True
#ENABLE_IDENTITY_V2=False

#RECLONE=yes

# Como está a correr em 14.04 não é preciso forçar a instalação
#FORCE=yes

# Descomentar para se poder trabalhar sem internet
#OFFLINE=True

# Permitir nested VM (aceleração de VM dentro de VM)
LIBVIRT_TYPE=kvm
libvirt_cpu_mode=none
compute_driver=libvirt.LibvirtDriver

# Enable Logging
VERBOSE=True
LOG_COLOR=True

ENABLED_SERVICES=n-cpu,neutron,n-novnc,q-agt,q-l3,q-meta,n-api-meta

# HOST_IP=[actual address of your adapter (eth0, etc.)
# FLOATING_RANGE=[range of floating addresses.. externally accessible from OpenStack]
*Does not need to exist before stacking
# PUBLIC_NETWORK_GATEWAY=[gateway address of public network that will be used to
connect internal OpenStack networks with external networks.]
# FIXED_RANGE=[internal network range for VMs to use]

# # controller ip
HOST_IP=192.168.100.21
HOST_IP_IFACE=eth1
MULTI_HOST=True
IP_VERSION=4
Q_META_DATA_IP=192.168.100.10
TUNNEL_ENDPOINT_IP=192.168.110.21
Q_DVR_MODE=dvr

# Configuração de rede
## Neutron options
Q_USE_SECGROUP=True
Q_L3_ENABLED=True
FLOATING_RANGE=10.100.1.0/24
FIXED_RANGE=10.99.0.0/24
Q_FLOATING_ALLOCATION_POOL=start=10.100.1.128,end=10.100.1.200
PUBLIC_NETWORK_GATEWAY=10.100.1.1
NETWORK_GATEWAY=10.99.0.1
```

```
# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_INTERFACE=eth0
PUBLIC_PHYSICAL_NETWORK=public
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex
#Q_OVS_USE_VETH=true;
TUNNEL_ENDPOINT_IP=192.168.110.21

# TODO: Set the controller's IP
SERVICE_HOST=192.168.100.10
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
Q_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292

VNCSERVER_PROXYCLIENT_ADDRESS=$HOST_IP
VNCSERVER_LISTEN=0.0.0.0

[[post-config|$NOVA_CONF]]
[DEFAULT]
default_ephemeral_format = ext2
force_config_drive = False
config_drive_format=vfat
[libvirt]
live_migration_uri = qemu+ssh://stack@%s/system
#live_migration_flag += VIR_MIGRATE_TUNNELLED
#virt_type = qemu
# cpu_mode = custom
# virt_type = kvm

[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ovs]
local_ip=192.168.110.21

[agent]
tunnel_types=vxlan
l2_population=True
enable_distributed_routing=True

[[post-config|$Q_L3_CONF_FILE]]
[DEFAULT]
agent_mode=dvr
router_delete_namespaces=True
```

- **Local.conf of Network:**

```
# Network Config - Com br-ex e br-tun
# Versão 5
# 20160508

[[local|localrc]]
DEST=/opt/openstack
```

```
DATA_DIR=$DEST/data
LOGFILE=$DATA_DIR/logs/stack.log
SCREEN_LOGDIR=$DATA_DIR/logs
MYSQL_PASSWORD=corenet
RABBIT_PASSWORD=corenet
SERVICE_TOKEN=corenet
SERVICE_PASSWORD=corenet
ADMIN_PASSWORD=corenet
SERVICE_DIR=$DEST/status
SUBUNIT_OUTPUT=$DEST/devstack.subunit
DATABASE_TYPE=mysql
PIP_UPGRADE=True
#ENABLE_IDENTITY_V2=False

#RECLONE=yes

# Como está a correr em 14.04 não é preciso forçar a instalação
#FORCE=yes

# Descomentar para se poder trabalhar sem internet
#OFFLINE=True

# Permitir nested VM (aceleração de VM dentro de VM)
LIBVIRT_TYPE=kvm
libvirt_cpu_mode=none
compute_driver=libvirt.LibvirtDriver

# Enable Logging
VERBOSE=True
LOG_COLOR=True

ENABLED_SERVICES=neutron,q-agt,q-l3,q-meta,q-dhcp
#ENABLED_SERVICES=neutron,q-agt,q-l3,q-meta,n-api-meta,q-dhcp
#ENABLED_SERVICES=key,mysql,q-svc,q-agt,q-dhcp,q-l3,q-meta,neutron

# HOST_IP=[actual address of your adapter (eth0, etc.)
# FLOATING_RANGE=[range of floating addresses.. externally accessible from OpenStack]
*Does not need to exist before stacking
# PUBLIC_NETWORK_GATEWAY=[gateway address of public network that will be used to
connect internal OpenStack networks with external networks.]
# FIXED_RANGE=[internal network range for VMs to use]

# Network host IP
HOST_IP=192.168.100.30
HOST_IP_IFACE=eth1
MULTI_HOST=True
IP_VERSION=4
Q_META_DATA_IP=192.168.100.10

# Configuração de rede
## Neutron options
Q_USE_SECGROUP=True
```

```
Q_L3_ENABLED=True
FLOATING_RANGE=10.100.1.0/24
FIXED_RANGE=10.99.0.0/24
Q_FLOATING_ALLOCATION_POOL=start=10.100.1.128,end=10.100.1.200
PUBLIC_NETWORK_GATEWAY=10.100.1.1
NETWORK_GATEWAY=10.99.0.1

# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_INTERFACE=eth0
PUBLIC_PHYSICAL_NETWORK=public
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex
#Q_OVS_USE_VETH=true;
TUNNEL_ENDPOINT_IP=192.168.110.30
Q_DVR_MODE=dvr_snat

# TODO: Set the controller's IP
SERVICE_HOST=192.168.100.10
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292

[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ovs]
local_ip=192.168.110.30

[agent]
tunnel_types=vxlan
l2_population=True
enable_distributed_routing=True

[[post-config|$Q_L3_CONF_FILE]]
[DEFAULT]
agent_mode=dvr_snat
router_delete_namespaces=True
```

Bibliography

[1] ALLEN, S. R. 2015. *The Business Value of OpenStack Private Cloud* [Online]. Available: https://www.linkedin.com/pulse/business-value-openstack-private-cloud-scott-r-allen.

[2] AMD. *AMD Virtualization* [Online]. Available: http://www.amd.com/us/solutions/servers/virtualization/Pages/virtualization.aspx.

[3] ANDRIEUX, A., CZAJKOWSKI, K., DAN, A., KEAHEY, K., LUDWIG, H., NAKATA, T., PRUYNE, J., ROFRANO, J., TUECKE, S. & XU, M. Web services agreement specification (WS-Agreement). Open Grid Forum, 2007. 216.

[4] ANSHU AWASTHI, R. G. 2015. *Comparison of OpenStack Installers* [Online]. Available: http://ijiset.com/vol2/v2s9/IJISET_V2_I9_92.pdf.

[5] ANTONOPOULOS, N. & GILLAM, L. 2010. *Cloud computing: Principles, systems and applications*, Springer Science & Business Media.

[6] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A. & STOICA, I. 2010. A view of cloud computing. *Communications of the ACM,* 53**,** 50-58.

[7] BADGER, L., GRANCE, T., PATT-CORNER, R. & VOAS, J. 2011. Draft cloud computing synopsis and recommendations. *NIST special publication,* 800**,** 146.

[8] BARRETT, C., BRITTEN, T., CACCIATORE, K., CHADWICK, P., PHIPPS, P., PRÜßMANN, G., ROSSETTI, M., SUN, Y. L., TAHIR, S., TRETHEWAY, H. J. & WU, S. *OpenStack: The Path to Cloud* [Online]. Available: https://www.openstack.org/assets/path-to-cloud/OpenStack-6x9Booklet-online.pdf.

[9] BARRETT, C., CACCIATORE, K., KAPADIA SR, A., PITZELY, D., PRÜßMANN, G., ROSSETTI, M., SUBRAMANIAN, S., SHAMAIL TAHIR, S., TAN, A. & WALLI, S. 2016. OpenStack: A Business Perspective.

[10] BERTAUX, L., MEDJIAH, S., BERTHOU, P., ABDELLATIF, S., HAKIRI, A., GELARD, P., PLANCHOU, F. & BRUYERE, M. 2015. Software defined networking and virtualization for broadband satellite networks. *IEEE Communications Magazine,* 53**,** 54-60.

[11] BHARDWAJ, S., JAIN, L. & JAIN, S. 2010. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology,* 2**,** 60-63.

[12] BOHN, R. B., MESSINA, J., LIU, F., TONG, J. & MAO, J. NIST cloud computing reference architecture. 2011 IEEE World Congress on Services, 2011. IEEE, 594-596.

[13] BUTLER, B. 2014. *15 most powerful OpenStack companies* [Online]. Available: http://www.networkworld.com/article/2176960/cloud-computing/15-most-powerful-openstack-companies.html.

[14] CHEN, C. P. & ZHANG, C.-Y. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences,* 275**,** 314-347.

[15] CHEN, D. & ZHAO, H. Data security and privacy protection issues in cloud computing. Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on, 2012. IEEE, 647-651.

[16]COMPUTIN, C. 2015. Elastic optical networks–a new approach for effective provisioning of cloud computing and content-oriented services. *architecture (SOA),* 7**,** 9.

[17]CONFERENCE, M. I. 2009. *WELCOME TO THE 31st INTERNATIONAL CONFERENCE OF DATA PROTECTION AND PRIVACY* [Online]. Available: www.privacyconference2009.org/home/index-iden-idweb.html .

[18]COSTA-REQUENA, J., SANTOS, J. L., GUASCH, V. F., AHOKAS, K., PREMSANKAR, G., LUUKKAINEN, S., PÉREZ, O. L., ITZAZELAIA, M. U., AHMAD, I. & LIYANAGE, M. SDN and NFV integration in generalized mobile network architecture. Networks and Communications (EuCNC), 2015 European Conference on, 2015. IEEE, 154-158.

[19]CREASY, R. J. 1981. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development,* 25**,** 483-490.

[20]CUI, C., DENG, H., TELEKOM, D., MICHEL, U. & DAMKER, H. 2012. Network Functions Virtualisation.

[21]DATA, I. 2014. Statistics Division: The world in 2014: ICT facts and figures. *International Organization for Standardization*.

[22]DIALOGIC. 2010. *Introduction to Cloud Computing* [Online]. Available: http://www.dialogic.com/~/media/products/docs/whitepapers/12023-cloud-computing-wp.pdf .

[23]DOCS.OPENSTACK. 2016a. *compute configuring migrations* [Online]. Available: http://docs.openstack.org/admin-guide/compute-configuring-migrations.html .

[24]DOCS.OPENSTACK. 2016b. *Introduction to OpenStack* [Online]. Available: http://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html

[25]ETSI. 2012. *CLOUD; SLAs for Cloud services* [Online]. Available: http://www.etsi.org/deliver/etsi_tr/103100_103199/103125/01.01.01_60/tr_103125v010101p.pdf .

[26]ETSI, G. 2014. 003,"Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[27]FOSTER, I., ZHAO, Y., RAICU, I. & LU, S. Cloud computing and grid computing 360-degree compared. 2008 Grid Computing Environments Workshop, 2008. Ieee, 1-10.

[28]FOSTER, I. A. 2013. *NIST's Security Reference Architecture for the Cloud-First Initiative* [Online]. Available: https://www.hpcwire.com/2013/06/28/nist_s_security_reference_architecture_for_the_cloud-first_initiative/ .

[29]FOUNDATION, O. N. 2014. *SDN Architecture, Issue 1, ONF TR-502* [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf [Accessed Mar, 2016].

[30]FOX, A., GRIFFITH, R., JOSEPH, A., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A. & STOICA, I. 2009. Above the clouds: A Berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS,* 28**,** 2009.

[31]GANGADHARAN, G. & PARRILLI, D. M. 2011. Service Level Agreements in Cloud Computing: Perspectives of Private Consumers and Small-to-Medium Enterprises. *Cloud Computing for Enterprise Architectures.* Springer.

[32]HAKIRI, A. & BERTHOU, P. 2015. Leveraging SDN for the 5g networks: Trends, prospects and challenges. *arXiv preprint arXiv:1506.02876*.

[33]HASHIZUME, K., ROSADO, D. G., FERNÁNDEZ-MEDINA, E. & FERNANDEZ, E. B. 2013. *An analysis of security issues for cloud computing* [Online]. Available: https://pdfs.semanticscholar.org/f274/36b9542ca0f6216040b1fc3606c3b2b7dfec.pdf.

[34]HAWILO, H., SHAMI, A., MIRAHMADI, M. & ASAL, R. 2014. NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Network,* 28**,** 18-26.

[35]INTEL. *Intel Virtualization Technology List: About Intel Virtualization Technology* [Online]. Available: http://ark.intel.com/products/virtualizationtechnology .

[36]INTERNETLIVESTATS. Available: http://www.internetlivestats.com/internet-users/.

[37]JACK MCCANN, RAJEEV GROVER, SWAMINATHAN VASUDEVAN & NARASIMHAN, V. 2014. *Architectural Overview of Distributed Virtual Routers in OpenStack Neutron* [Online]. Available: https://www.openstack.org/assets/presentation-media/Openstack-kilo-summit-DVR-Architecture-20141030-Master-submitted-to-openstack.pdf .

[38]JADEJA, Y. & MODI, K. Cloud computing-concepts, architecture and challenges. Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on, 2012. IEEE, 877-880.

[39]JAMES, B. 2010. Security and privacy challenges in cloud computing environments.

[40]JAMMAL, M., SINGH, T., SHAMI, A., ASAL, R. & LI, Y. 2014. *Software defined networking: State of the art and research challenges* [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1406/1406.0124.pdf [Accessed 72].

[41]JI, C., LI, Y., QIU, W., AWADA, U. & LI, K. Big data processing in cloud computing environments. 2012 12th International Symposium on Pervasive Systems, Algorithms and Networks, 2012. IEEE, 17-23.

[42]JUN, Z., SIMPLOT-RYL, D., BISDIKIAN, C. & MOUFTAH, H. 2011. The internet of things. *IEEE Commun. Mag,* 49**,** 30-31.

[43]KAMAMURA, S., MORI, H., SHIMAZAKI, D., GENDA, K. & UEMATSU, Y. OSPF and BGP State Migration for Resource-Portable IP Router. 2015 IEEE Global Communications Conference (GLOBECOM), 2015. IEEE, 1-6.

[44]KATZAN, H. 2011. Cloud computing, I-Service, and IT service provisioning. *Journal of Service Science (JSS),* 1**,** 57-64.

[45]KELLER, A. & LUDWIG, H. 2003. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management,* 11**,** 57-81.

[46]KOURIK, J. L. For small and medium size enterprises (SME) deliberating cloud computing: a proposed approach. Proceedings of the European Computing Conference, 2011. 216-221.

[47]KUO, M.-H. 2011. Opportunities and challenges of cloud computing to improve health care services. *Journal of medical Internet research,* 13**,** e67.

[48]KVM. *virtualization solution* [Online]. Available: http://www.linux-kvm.org .

[49]LAU, P. Y., PARK, S., YOON, J. & LEE, J. Pay-as-you-use on-demand cloud service: An IPTV case. Electronics and Information Engineering (ICEIE), 2010 International Conference On, 2010. IEEE, V1-272-V1-276.

[50]LI, P. 2010. Selecting and using virtualization solutions: our experiences with VMware and VirtualBox. *Journal of Computing Sciences in Colleges,* 25**,** 11-17.

[51]LI, W. & PING, L. Trust model to enhance security and interoperability of cloud environment. IEEE International Conference on Cloud Computing, 2009. Springer, 69-79.

[52]LOPEZ, D. R. Network functions virtualization: Beyond carrier-grade clouds. Optical Fiber Communications Conference and Exhibition (OFC), 2014, 2014. IEEE, 1-18.

[53]MAGAZINE, I. S. 2012. 40 Years of VM Community—and Going Strong!

[54]MARSTON, S., LI, Z., BANDYOPADHYAY, S., ZHANG, J. & GHALSASI, A. 2011. *Cloud computing—The business perspective* [Online]. Available: http://www.cs.joensuu.fi/~parkkine/LuK2015/CloudComputing-DecisionSupportSystems2011.pdf.

[55]MELL, P. & GRANCE, T. 2011. The NIST definition of cloud computing.

[56]NASHNETWORKSINC. *Virtualization: A Small Business Perspective* [Online]. Available: http://www.nashnetworks.ca/UserFiles/File/Virtualization%20whitepaper.pdf .

[57]ONF, O. N. F. 2012. *Software-defined networking: The new norm for networks* [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf .

[58]ONF, O. N. F. 2014. *SDN architecture* [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf .

[59]OPENSTACK. *Open source software for creating private and public clouds* [Online]. Available: https://www.openstack.org/ .

[60]OPENVZ. *Container-based virtualization for Linux* [Online]. Available: http://openvz.org/ .

[61]ORACLE. 2011. *VM User's Guide: Brief History of Virtualization* [Online]. Available: http://docs.oracle.com/cd/E20065_01/doc.30/e18549/intro.htm .

[62]PARALLELS. *Parallels Virtuozzo Containers* [Online]. Available: http://www.parallels.com/products/pvc/ .

[63]PATIDAR, S., RANE, D. & JAIN, P. A survey paper on cloud computing. 2012 Second International Conference on Advanced Computing & Communication Technologies, 2012. IEEE, 394-398.

[64]PFAFF, B., PETTIT, J., AMIDON, K., CASADO, M., KOPONEN, T. & SHENKER, S. Extending Networking into the Virtualization Layer. Hotnets, 2009.

[65]PLUMMER, D. C., BITTMAN, T. J., AUSTIN, T., CEARLEY, D. W. & SMITH, D. M. 2008. Cloud computing: Defining and describing an emerging phenomenon. *Gartner, June,* 17.

[66]POPEK, G. J. & GOLDBERG, R. P. 1974. Formal requirements for virtualizable third generation architectures. *Communications of the ACM,* 17**,** 412-421.

[67]RIGHTSCALE. *Universal Cloud Management Platform* [Online]. Available: http://www.rightscale.com/ .

[68]SALFNER, F., TRÖGER, P. & RICHLY, M. 2012. Dependable estimation of downtime for virtual machine live migration. *International Journal On Advances in Systems and Measurements,* 5.

[69]SDXCENTRAL. 2015. *SDxCentral NFV Landscape Report 2015 – Online Edition* [Online]. Available: https://www.sdxcentral.com/reports/nfv-2015/the-components-of-an-nfv-architecture/ .

[70]SEN, J. 2013. Security and privacy issues in cloud computing. *Architectures and Protocols for Secure Information Technology Infrastructures*, 1-45.

[71]SERVERWATCH. 2015. *Eight Big Benefits of Software-Defined Networking* [Online]. Available: http://www.serverwatch.com/server-tutorials/eight-big-benefits-of-software-defined-networking.html .

[72]SHARKH, M. A., JAMMAL, M., SHAMI, A. & OUDA, A. 2013. Resource allocation in a network-based cloud computing environment: design challenges. *IEEE Communications Magazine,* 51, 46-52.

[73]SO, K. 2011. Cloud computing security issues and challenges. *International Journal of Computer Networks,* 3.

[74]STANOEVSKA-SLABEVA, K. & WOZNIAK, T. 2010. Cloud basics–an introduction to cloud computing. *Grid and cloud computing.* Springer.

[75]SWANSON, B. & GILDER, G. 2008. Estimating the exaflood. *Discovery Institute white paper*.

[76]TALBOT, S. 2014. *Cloud for the Hybrid Data Center Private Cloud & Service Provider Panel Session* [Online]. Available: http://www.slideshare.net/NetAppUK/cloud-for-the-hybrid-data-center-private-cloud-service-provider-panel-session .

[77]TCPDUMP. *Powerful command-line packet analyzer* [Online]. Available: www.tcpdump.org/ .

[78]TURNER, J. S. & TAYLOR, D. E. Diversifying the internet. GLOBECOM'05. IEEE Global Telecommunications Conference, 2005., 2005. IEEE, 6 pp.-760.

[79]VAQUERO, L. M., RODERO-MERINO, L., CACERES, J. & LINDNER, M. 2008. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review,* 39, 50-55.

[80]VMWARE. *virtualization software company* [Online]. Available: http://www.vmware.com .

[81]WEI, Y. & BLAKE, M. B. 2010. Service-oriented computing and cloud computing: challenges and opportunities. *IEEE Internet Computing,* 14, 72.

[82]WIEDER, P., BUTLER, J. M., THEILMANN, W. & YAHYAPOUR, R. 2011. *Service level agreements for cloud computing*, Springer Science & Business Media.

[83]WILEY, J. 2015. *Software Defined Networking For Dummies* [Online]. Available: http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/sdnfordummies.pdf .

[84]XEN. *Open Source Virtualization* [Online]. Available: http://www.xenserver.org/ .

[85]XILOURIS, G., TROUVA, E., LOBILLO, F., SOARES, J., CARAPINHA, J., MCGRATH, M., GARDIKIS, G., PAGLIERANI, P., PALLIS, E. & ZUCCARO, L. T-NOVA: A marketplace for virtualized network functions. Networks and Communications (EuCNC), 2014 European Conference on, 2014. IEEE, 1-5.

[86]ZHANG, Q., CHENG, L. & BOUTABA, R. 2010. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications,* 1, 7-18.