

Open University of Cyprus

Faculty of Pure and Applied Sciences

Master Thesis

Systems and Network Security



**Automating external networks creation in Cyber Ranges using
scripting**

Georgios Benetatos

**Supervisor
Peratikou Adamantini**

05/2021

Open University of Cyprus

Faculty of Pure and Applied Sciences

**Automating external networks creation in Cyber Ranges using
scripting**

Benetatos Georgios

Supervisor

Peratikou Adamantini

This Master Thesis was submitted,
for the competence of the requirements to acquire

Master's degree in
System and Network Security

From Faculty of Pure and Applied Sciences
Open University of Cyprus

05/2021

Summary

Two virtual machines hosted in a given Virtualization environment are used to host an OpenStack deployment with all the basic components, like Neutron and Nova. Python scripts are written and deployed to create a virtual network and all the necessary components, like router and network interfaces. The communication with the stack is achieved utilizing the API endpoints created in OpenStack. Network performance tests, for latency and throughput, are made, measuring the connection between the already existed network 192.168.30.0/24 and a Virtual machine hosted inside Openstack in the newly created network 172.16.0.0/24 and the connection between two Virtual machines, both members of 172.16.0.0/24 network. Finally, the results from the tests are compared with the results from the communication of 2 Virtual machines hosted in the given Cyber Range, giving slightly better performance statistics.

Master Thesis	i
Systems and Network Security.....	i
Chapter 1.....	1
Introduction.....	1
1.1 Scope	1
1.2 Main Target.....	1
1.3 Layout.....	2
Chapter 2.....	3
Literature Review	3
2.1 Key concepts and definitions	3
2.1.1 Cyber Security.....	3
2.1.2 Cyber security training	4
2.1.3 Cloud computing.....	4
2.1.4 Cyber Ranges	5
2.1.5 Automations in Cyber Ranges.....	7
2.1.6 Openstack	8
2.1.7 OpenStack architecture and components.....	9
2.2 OpenStack Networking.....	10
2.3 Open Virtual Switch	13
2.4 Previous RESEARCH	14
2.4.1 Openstack - Virtualization.....	15
2.4.2 Automation and Python.....	16
2.4.3 Means of Compare and research.....	16
2.5 Conclusion	16
Chapter 3.....	17
Methodology	17
3.1 Type of Research.....	17
3.2 Development lifecycle	17
3.2.1 Proposed Lifecycle 1 - Waterfall Model	18
3.2.2 Proposed Lifecycle 2 - Spiral Model	20
3.2.3 Chosen model.....	21
3.3 Research main questions.....	22
Chapter 4.....	22
Deployment.....	22

4.1	Virtualization Deployment.....	22
4.2	Controller Node	24
4.3	Compute Node	27
4.4	Scripts and Code	28
4.4.1	Connection - Getting the authentication token.....	28
4.4.2	Network Creation.....	29
4.4.3	List networks.....	30
4.4.4	Subnet List	30
4.4.5	Subnet Create	31
4.4.6	Router list.....	32
4.4.7	Router Create	32
4.4.8	Port List.....	33
4.4.9	Port Create	33
4.4.10	Script structure	33
Chapter 5.....		34
Testing.....		34
5.1	Getting the Authentication Token.....	34
5.2	Creating Network	36
5.3	Create Subnet.....	37
5.4	Create new Router - External network.....	37
5.5	Creating Instance in OpenStack (from Horizon).....	38
5.6	Network diagram from OpenStack Horizon	40
5.7	Checking network with Iperf	41
5.7.1	Latency	43
5.7.2	Throughput	45
Chapter 6.....		48
Conclusion - Discussion.....		48
6.1	Discussion	48
6.1.1	Python for automations	48
6.1.2	Openstack API interaction	49
6.1.3	Comparison of Networks created in OpenStack with the preexisting in the virtualization.....	49
6.2	Reflections.....	50
6.2.1	Limitations.....	51
6.3	Future Works.....	51

Bibliography 51

CHAPTER 1

INTRODUCTION

Cloud Computing is starting to gain more and more attention in the IT market. Large companies invest huge amount of time and money to migrate from the on-premise deployment they use to have to a more flexible, cost-efficient and full modifiable solution, provided by cloud computing. Either it is Public or private cloud, this kind of migration needs perfect planning and very meticulous implementation. One of the most important parts in that process is the development of automations, well-designed and optimized to meet the needs.

1.1 SCOPE

In this research will be deployed an Openstack environment, utilizing all the basic OpenStack components as Neutron, keystone, Horizon, Nova. After the creation, python scripts will be deployed in order to automate the creation, management and change of external networks and virtual routers.

1.2 MAIN TARGET

The main goals for this research is to deploy a Multi Node Full OpenStack environment using all the common Openstack components (Neutron, keystone, Horizon, Nova). After the creation of the environment, python scripts will be deployed in order to investigate how automations can be implemented, using the API endpoints of the deployment. With the scripts the creation and management of external networks will be facilitated without the need to use either the Stack

controller neither the GUI environment provided from OpenStack (Horizon). The networks created will be tested as external networks in a virtualization environment.

1.3 LAYOUT

At the beginning a short description of basic definitions will be presented, followed by related works and research currently in progress or recently published. That will give a big picture of the subject as well as it will justify better the need of this research.

At the next section, a full detailed description of the OpenStack deployment will take place. This is a crucial subject as the settings applied in network as well as in compute layer will have great impact in the deployment of the Python automation scripts.

The scripts, with a detailed explanation will also be presented. Python code, with comments will be deployed in functional parts for better understanding the process. At the end a full program will be proposed, using python and linux bash scripts. The results of the scripts will also be a subject in this research.

At the end the created and managed networks and network components will be tested in a production virtualization environment .

CHAPTER 2

LITERATURE REVIEW

Internet security is one of the most important issues today, with a large number of researches trying to approach the issue from different angles. The needs of training, simulation and expansion of techniques (defensive and offensive) in the field of cyber security lead to the increasing use of Cyber Range, utilizing every opportunity (Networking, Software, Hardware, Cloud). The investigation of different types of networks, created within the Cyber Range, and the way they behave, as well as the effort to introduce automation mechanisms, is a field that this research will deal with. The results will be added to the overall effort made to optimize Cyber Range and make the most of it.

2.1 KEY CONCEPTS AND DEFINITIONS

Below we will present the basic concept and definitions for this research, trying to achieve a smooth and specific introduction to the topic

2.1.1 CYBER SECURITY

Cyber security, or else information technology security is the summarization of practices and mechanisms to defend electronic systems and networks from malicious attacks. Depending the target and the processes followed, cyber security can be divided into several categories such as Network Security, Application level Security, Information Security, Operational security, Disaster Recovery and so much more. **(Kaspersky LAB, 2019)**

Cyber security has a crucial impact in today's economy in all over the world. The most recent researches show that only in USA the financial damage from proved cyber crimes is more than 3,5 billion dollars every year, a number that seems to increase. More and more companies and organizations try to adopt sustainable methods to protect their companies intel and wealth from cyber criminals, that opens the horizons for a whole new market in information technology.

2.1.2 CYBER SECURITY TRAINING

Security awareness and training is becoming one of the most important parts in today's life. Even if someone has nothing to do with information technology, needs to be aware and of course trained in some way, to identify and avoid such attacks. All modern well practice standards that the enterprises adopt, suggest continuing training to all the employees about security awareness and dangers.

The need for advanced security training methods is not something that concerns only the end-users . New methods, more practical and less theoretical need, to be implemented in Cyber security training for the professionals, so they can meet the high standards. Old training methods that includes large theoretical parts seems now to be inadequate. More and more research are getting published, concerning new methods for security risk simulation, that provides to the cyber security professionals all the "on the job" skills needed. Safe and legal environments for cyber training, such us Cyber Ranges gaining more and more place in this highly demanding word.

2.1.3 CLOUD COMPUTING

Cloud computing is the innovation of the last five years, bringing together all the disciples technologies and models that are used till now in the IT business. What makes cloud computing so important and succeed in the market so far is the "As a Service" options that gives to the user. IT resources (networks, hardware, applications, coding) can now be delivered serverless, dynamically and with full scalability using the internet.

Cloud providers supply the users with a virtual unlimited number of resources on demand that can be any time adjust to meet every need. For business continuity, cloud providers, such us Amazon, Google, Microsoft, also provide strong Service level Agreements (SLA) as a warranty for the availability of the resources. Billing options are also scalable and depends on the consumed resources.

The basic areas so far that cloud computing offers are the following:

- **Infrastructure as a Service (IaaS):** The whole equipment, like virtual machines, storage units and networks, is provided by the cloud provider, while the end user needs to maintain the application, runtimes, databases and server software.
- **Platform as a Service (PaaS):** The and user develop its own applications using the services provided. Cloud providers maintain the runtime Cloud, SOA integration, databases, server software, storage units and virtualization server hardware
- **Software as a Service (SaaS):** The entire application is available in the cloud and the users just connect to the application via Internet connection.

2.1.4 CYBER RANGES

To mitigate the dangers of Cybercrimes researchers are developing cyber defense tools and procedures that need to be tested before going live, so they can evaluate their effectiveness as well as to calculate any possible risk in productive environments. Some years before, testing in small networks or even in a single computer was enough. Now, is insufficient as it does not give the required level of realism. Real life defensive or offensive scenarios need to be recreated so the researchers can obtain valuable intel by analyzing them.

As a crudest definition , Cyber Ranges are typical hardware with the ability to connect to each other forming complex network topologies, emulating a great variety of traffic models. What makes Cyber Range a great environment for cyber security training is that they are in most cases completely separated from the Internet so any tests cannot impact productive networks. Maintaining a cyber range can be a very expensive and time-consuming effort, with a large number of information technology staff needed to operate them. That is the reason why the largest Cyber Ranges are owned by large organizations and governments.

To be more specific depending on their purpose of use, we can classify cyber ranges in some categories.

- **Military and Defense use**

They are large scale cyber ranges used by Military organizations and government agencies all over the world to properly train Cyber Warriors against Cyber Terrorism. The most well-known Cyber Ranges of this type is the National Cyber Range (NCR) of Defense Advanced Research Project Agency (DARPA) working for the Department of Defense of the United States of America. European Union, is also up to develop its own Cyber Range for research and training reasons.

- **Education**

The idea of using a Cyber Range in education is not earlier than 2015 and it is first introduced in USA as a program to retrain veterans. Soon the idea spread all over the world. Cyber range in education are not only meant to be digital playgrounds for students. Educators and students can benefit from cyber ranges developing different scenarios that can best simulate real life conditions and gain valuable skills.

- **Enterprise and Commercial**

They are cyber ranges developed by large commercial organizations to conduct games and simulations in order to strengthen security awareness and capabilities of their employees. They are expensive but research so far shows that they have incredible and fast results. The first and enterprise Cyber Range was developed by IBM with the name of IBM X Force Command Center.

- **Open Source**

Like the Arizona Cyber Warfare Range, this kind of cyber ranges provide a safe environment for novice as well as advanced security professionals to test their hacking knowledge and freely and legally test their security skills.

- **Law enforcement**

As mentioned before, many agencies understand the need to train their personnel about the importance of cyber security. Taking as an example the Michigan Cyber Range, law enforcement agencies can benefit from cyber ranges by hardening their security policies and by upgrading their forensics and tech knowledge.

2.1.5 AUTOMATIONS IN CYBER RANGES

A Cyber Range, as we previously mentioned, serves many different purposes. Each scenario that is built has its own variables and settings. While the hardware part of a cyber range almost every time remains stable, network topologies change to fulfill its goal. Cyber Range administrators have to make these changes and monitor their implementation. This process could be extremely complex and time consuming without the use of automation tools such as OpenStack controller which allows fast and efficient network design and reconfiguration using APIs. **(MIT Lincoln Laboratory Lexington United States, 2016)** In more details cloud controllers like OpenStack are used as network management configuration tools that can create and manage physical nodes and network experiments with multiple virtual machines and network services. Each network topology can well be managed among nodes. **(Zhihong, et al., 2018)**

It is very common in Cyber Ranges deployment, especially when a cloud based solution is chosen, to use again and again the same scenario in order to achieve all the wanted results. Building from scratch may be usable in small testing environments but while the complexity of the project is raising, this would be difficult and extremely time consuming. For that reason, the establishment of a basic framework in such deployments is under constant investigation by community.

Many researchers propose the use of Virtual Scenario Description Language (VSD). The syntax of the language is extremely rich. It contains the full description of the core elements of a deployment, including nodes, network topologies, rules and settings, configurations, all written and defined through a group of statements. **(Costa, Russo, & Armando, 2015)**

Even it is still in project and research stage, this could be the future of automations in OpenStack and in all Cyber Range deployments in general.

```
1 ; Scenario elements
2 (declare-fun Phone () Int)
3 (declare-fun ApacheS () Int)
4 (declare-fun Laboratory () Int)
5 ; ...
6 ; Hardware constraints: Phone
7 (assert (forall ((u Int)) (and (< (node.cpu u Phone) 16192) (< (node.disk u Phone) 32768))))
8 (assert (forall ((u Int)) (and (>= (node.cpu u Phone) 512) (>= (node.disk u Phone) 2048))))
9 (assert (forall ((u Int)) (< (node.disk u Phone) 8192)))
10 (assert (forall ((u Int)) (< (node.cpu u Phone) 2048)))
11 ; ...
12 ; Network constraints: Laboratory
13 (assert (forall ((u Int) (n Int))
14 (or
15 (and
16 (<= (network.node.address u n Laboratory) 134744065)
17 (>= (network.node.address u n Laboratory) 134744128)
18 )
19 (= (network.node.address u n Laboratory) 0)
20 )))
21 (assert (forall ((u Int)) (= (network.node.address u RSLaptop Laboratory) 134744067)))
22 (declare-fun t () Int)
23 (assert (< t 40))
24 (assert (forall ((u int))
25 (and
```

PICTURE 1 EXAMPLE OF VSDL SCRIPTING LANGUAGE

2.1.6 OPENSTACK

OpenStack is a well-established Infrastructure as a Service (IaaS) with numerous software solutions that meets all the needs for today's cloud computing, both public and private cloud. It is Open Source, highly configurable and flexible and that is the reason why it is one of the most used cloud stack all over the world.

Many researchers try to compose an accurate definition of OpenStack. All agrees that Open Stack complex architecture as well as the great number of solutions that have developed and still are being developed is the key to success of this cloud stack that enterprises as well as large Cloud Providers can use to set up and maintenance their cloud infrastructure. In general, we can tell that OpenStack is a cloud platform that facilitates the creation and management of different sets of computing resources, storages and networks, using web interface. (Lima, Rocha, & Licinio, 2019)

The first release of Open Stack that includes only OpenStack Swift and Openstack NOVA, was in 2010 by Rackspace and NASA, named as Austin. The goal of these two colossal enterprises was to create an open-source cloud platform that could provide managed services both in public and private cloud. One of the greatest success from this project is that in less than 5 years, a multi member community was created, including more than 180 companies and 6000 individuals. This community actively contributes to the development and the improvement of the project. Official documentation (OpenStack Docs) as well as Blogs and other communication channels use to expand OpenStack possibilities and capabilities.

2.1.7 OPENSTACK ARCHITECTURE AND COMPONENTS

The architecture is not something fixed. Every day new components are added, providing more services in the IaaS layer. Three key goals must be met by all OpenStack releases and architectures. It must be an opensource Cloud environment, very modular and flexible, using API for service unification and automation as well as supporting a great number of virtualization standards, and it must be full scalable.

Open stack architecture is designed to be horizontally scalable as it needs to provide intelligent storage solutions to meet growing business needs. (Lima, Rocha, & Licinio, 2019)

As it is almost impossible to include all OpenStack projects in details, some basic components and services for OpenStack can be summarized in the bellow categorizes.

- **Compute (Nova):** This is the cloud computing infrastructure manager. It does not support any of virtualization capabilities by itself, though using the appropriate APIs can control and manage the instances as well as all the resources and the networks
- **Image Service (Glance):** This service is for making and retrieving virtual machines storages.
- **Object Storage (Swift):** OpenStack object store project, providing cloud storage software and automations using APIs. Swift is also responsible for clustering the data and distributing copies of each object stored by regions.

- **Dashboard (Horizon):** Is the graphical Web interface of Openstack, integrating all provided services.
- **Networking (Neutron):** Provides virtual networking to the VMs, attaching their vNIC to the virtual switch interface. The use of APIs can automate and facilitate raw networking in cloud.
- **Orchestration (Heat):** Project whose target is to manage the automatic way, a large amount of instances. The automation usually uses OpenStack RESTful API (HTTP) to make it happened

2.2 OPENSTACK NETWORKING

As mentioned, flexibility is a key concept in cloud computing. One of the most important and in parallel most challenging area of study is the implementation of networking in the cloud environments. OpenStack Networking allows to easily create and manage network objects, giving leveraged Layer 2 and of course Layer 3 connectivity, using subnets and ports as well taking advantage of numerous of different networking equipment, using the appropriate plug ins.

Neutron, that is the name of OpenStack's Networking service, provides an API that let you to define network connectivity and addressing in the cloud, making operators able to implement different networking technologies. There is also an API that allows the administrators to manage a great variety of services, from NAT, load balancing and redundancy mechanism to firewalls and VPNs. Summarizing, Networking services includes in general, an API server that supports Layer 2 networking and IP address management, many plug-ins and agent as well as a Messaging queue that accepts and routes RPC requests between agents to complete API operations.

To configure rich network topologies, you can create and configure networks and subnets and instruct other OpenStack services like Compute to attach virtual devices to ports on these networks. OpenStack Compute is a prominent consumer of OpenStack Networking to provide connectivity for its instances. In particular, OpenStack Networking supports each project having multiple private networks and enables projects to choose their own IP addressing scheme, even if those IP addresses overlap with those that other projects use.

There are two types of network, project and provider networks. It is possible to share any of these types of networks among projects as part of the network creation process. (OpenStack Docs, 2019)

- **Provider Networks / External Networks**

Provider networks offer layer-2 connectivity to instances with optional support for DHCP and metadata services. These networks connect, or map, to existing layer-2 networks in the data center, typically using VLAN (802.1q) tagging to identify and separate them. Provider networks generally offer simplicity, performance, and reliability at the cost of flexibility. By default only administrators can create or update provider networks because they require configuration of physical network infrastructure.

Also, provider networks only handle layer-2 connectivity for instances, thus lacking support for features such as routers and floating IP addresses.

In many cases, operators who are already familiar with virtual networking architectures that rely on physical network infrastructure for layer-2, layer-3, or other services can seamlessly deploy the OpenStack Networking service. In particular, provider networks appeal to operators looking to migrate from the Compute networking service (nova-network) to the OpenStack Networking service. Over time, operators can build on this minimal architecture to enable more cloud networking features.

In general, the OpenStack Networking software components that handle layer-3 operations impact performance and reliability the most. To improve performance and reliability, provider networks move layer-3 operations to the physical network infrastructure.

In one particular use case, the OpenStack deployment resides in a mixed environment with conventional virtualization and bare-metal hosts that use a sizable physical network infrastructure. Applications that run inside the OpenStack deployment might require direct layer-2 access, typically using VLANs, to applications outside of the deployment. (OpenStack Docs, 2019)

As a subcategory of provider networks, there are the **Routed provider networks**, that offers basic layer 3 connectivity to cloud instances. As provided networks maps on an existing physical layer 2

network infrastructure, routed provider networks communicate directly with layer 3 networks that exists in a datacenter. To be more specific this kind of networks use multiple Layer 2 segments, that we can describe as prover networks, adding a router gateway that giving the possibility to communicate with external networks .

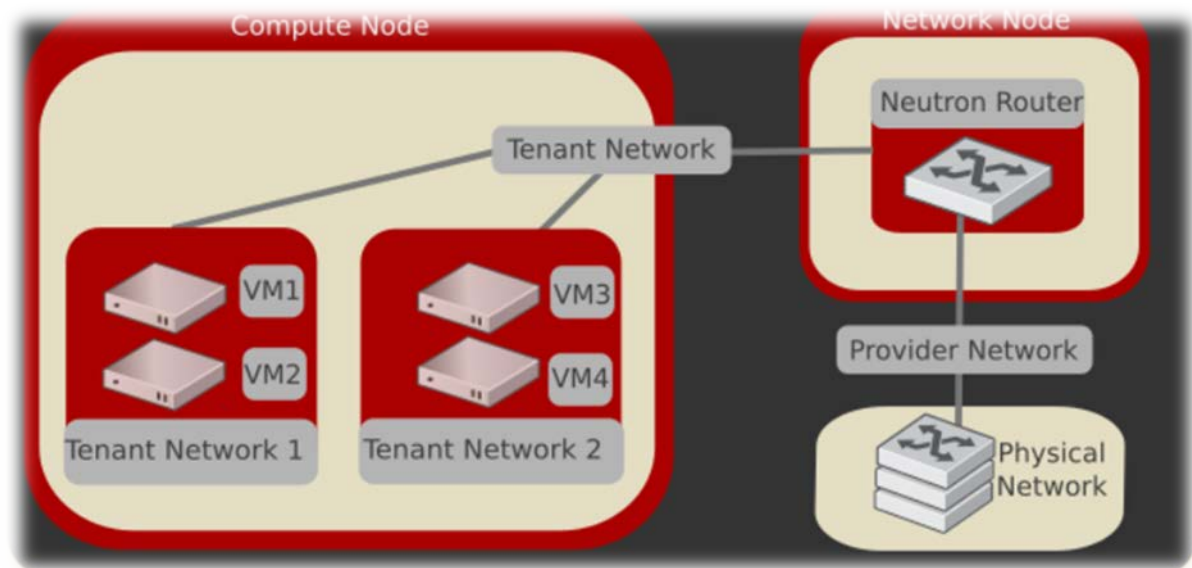
- **Self-service networks / Tenant Networks**

Self-service networks primarily enable general (non-privileged) projects to manage networks without involving administrators. These networks are entirely virtual and require virtual routers to interact with provider and external networks such as the Internet. Self-service networks also usually provide DHCP and metadata services to instances.

In most cases, self-service networks use overlay protocols such as VXLAN or GRE because they can support many more networks than layer-2 segmentation using VLAN tagging (802.1q). Furthermore, VLANs typically require additional configuration of physical network infrastructure.

IPv4 self-service networks typically use private IP address ranges (RFC1918) and interact with provider networks via source NAT on virtual routers. Floating IP addresses enable access to instances from provider networks via destination NAT on virtual routers. IPv6 self-service networks always use public IP address ranges and interact with provider networks via virtual routers with static routes.

The Networking service implements routers using a layer-3 agent that typically resides at least one network node. Contrary to provider networks that connect instances to the physical network infrastructure at layer-2, self-service networks must traverse a layer-3 agent. (OpenStack Docs, 2019)



PICTURE 2 VISUALIZATION OF NETWORKING IMPLEMENTATION WITH OPENSTACK. (OPENSTACK NETWORKING DOCUMENTATION [HTTPS://DOCS.OPENSTACK.ORG/](https://docs.openstack.org/))

2.3 OPEN VIRTUAL SWITCH

Open vSwitch is a multilayer software switch licensed under the open source Apache 2 license. The main goal is to implement a production quality switch platform that supports standard management interfaces and opens the forwarding functions to programmatic extension and control.

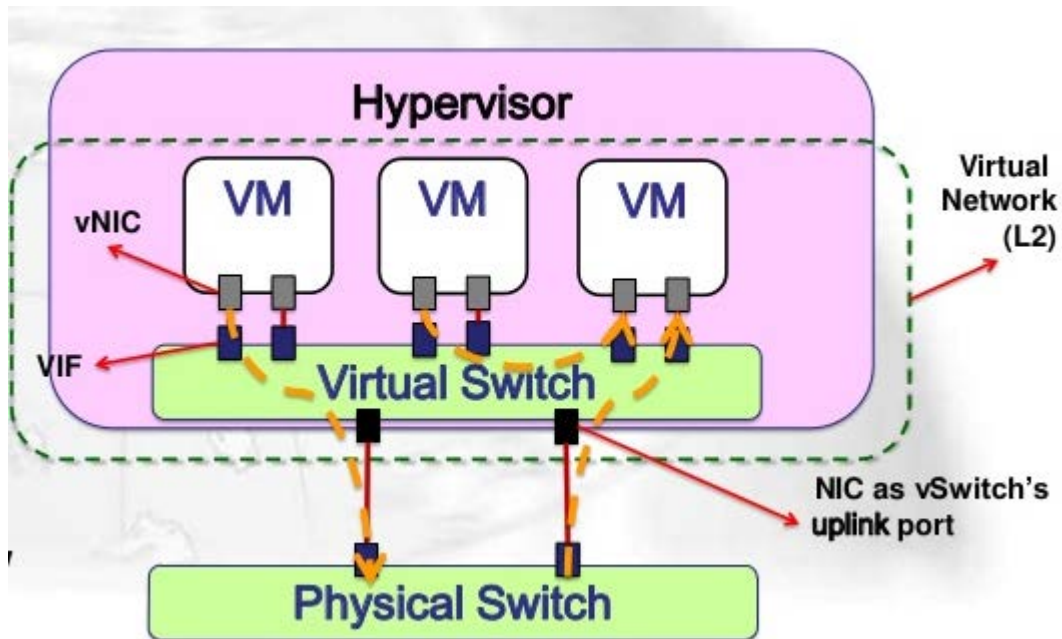
Open vSwitch is well suited to function as a virtual switch in VM environments. In addition to exposing standard control and visibility interfaces to the virtual networking layer, it was designed to support distribution across multiple physical servers. Open vSwitch supports multiple Linux-based virtualization technologies including Xen/XenServer, KVM, and VirtualBox (**OVS Docs, 2020**)

The bulk of the code is written in platform-independent C and is easily ported to other environments. The current release of Open vSwitch supports the following features:

- Standard 802.1Q VLAN model with trunk and access ports
- NIC bonding with or without LACP on upstream switch
- NetFlow, sFlow(R), and mirroring for increased visibility
- QoS (Quality of Service) configuration, plus policing
- Geneve, GRE, VXLAN, STT, and LISP tunneling
- 802.1ag connectivity fault management
- OpenFlow 1.0 plus numerous extensions
- Transactional configuration database with C and Python bindings
- High-performance forwarding using a Linux kernel module

The included Linux kernel module supports Linux 3.10 and up.

Open vSwitch can also operate entirely in userspace without assistance from a kernel module. This user space implementation should be easier to port than the kernel-based switch. OVS in user space can access Linux or DPDK devices. **(OVS Docs, 2020)**



PICTURE 3 TYPICAL OVS DEPLOYMENT - TE-YEN LIU INDUSTRIAL TECHNOLOGY RESEARCH INSTITUTE

2.4 PREVIOUS RESEARCH

The issue of Cyber Range has been actively and vividly addressed over the last decade, with a wealth of research trying to approach and expand it. According to the European Internet Safety Agency (ECS), in an attempt to approach the issue, two possible definitions are mentioned. From what is defined as an environment of systems, which create apart from the Internet, a simulation environment where legally and safely can be tested and trained professionals on security. This observation is clearly seen in the article of the European Internet Safety Agency **(ECS - p. 10)**. NIST seems to agree with this definition in the relevant article and material published at the end of 2019, substantiating the Cyber Range. **(NIST - pp. 4-5)**. At the same time, studying the literature, we

observe that many researchers accept the above and incorporate them in their work. **(Gabriele Costa, 2019 - p. 7).**

In contrast, some researchers prefer to approach the concept of Cyber Range more broadly, saying that it is a platform that can easily adapt to different conditions to meet different needs each time. Key elements in this definition are the features of the Cloud and the "As a Service" feature it provides. Both in this research and in others, networks are created based on the needs at a time, which are flexible and easy to install, and accessible to a large number of people. In **(Peter Mell, 2013, NIST - p. 2)**, we can see this relationship very purposefully and concisely. The fact that it has been approved and published by NIST gives great validity to the source

2.4.1 OPENSTACK – VIRTUALIZATION

As the use of Cyber Range increases, so does the need to develop complex architectures that combine not only the need for an isolated, protected environment but also the need for a broad user interface to run more responsive simulations with the greatest accuracy possible. The research of **(Rosenstein & Corvese, 2018 pp. 2,4,5,6)** carried out in the framework of a program of John Hopkins University, Laboratory of Applied Physics, presents a detailed architectural model of a Cyber Range, applied to the National Cyber Range (NCR). From this source, which is guaranteed from the numerous reports it has, since its publication, we draw important information about the main advantages and the need to continue research in this area **(as shown by the conclusions p. 8-9)**

A key, and common element of all the researches that have been carried out is the effort of automation in the creation and management of networks (whether they are implemented on a physical level or through virtualization). Openstack for the development of their experiment, the management of resources for this issue is one of the key elements that researchers use and apply.

The OpenStack kernel will be used as a Controller for this research. Based on previous studies, it is probably the most ideal choice as it has shown the best performance, although in its installation it is very demanding. In their publication **(Jaison Paul Mulerikkal, 2020 p. 1)**, where a comparative presentation of two Open Source systems is made, it seems that with Openstack more efficient and accurate resource management is done.

2.4.2 AUTOMATION AND PYTHON

From **(Wang & Zhang, 2017 - p. 3)**, we read that the services offered by Openstack enable the use of APIs for the management of network resources, participating in the 5-level architecture proposed (and supported) by the organization, by a plethora of guides and publications. **(OpenStack, 2018)**

The Python APIs offered by Openstack will be used for the implementation of automation and are suggested by many as a better solution compared to the classic way of REST APIs or command-line tools. **(Hochstein 2013), (SEFRAOUI, AISSAOUI and ELEULDJ 2012 - p. 39)**, in many parts of their presentations confirm this fact.

2.4.3 MEANS OF COMPARE AND RESEARCH

The final purpose of this research is to evaluate the characteristics of the networks that will be implemented as well as their quantitative comparison with the existing networks in the given Cyber Range. As mentioned in **(Tantardini, et al. 2015)**, there are a variety of methods for determining and comparing the quantitative characteristics of networks, which can provide reliable and important data for their operation. The use of mathematical and statistical models in this publication, as well as the detailed presentation (graphically and verbally) of the results certainly demonstrates validity in the source and makes it perhaps one of the main tools for processing the data that will be collected in the study.

2.5 CONCLUSION

We notice that in the literature there is a huge volume of publications, especially recent ones, that document the development of networks and their automation using Openstack. From the conclusions of the research it seems that only in recent years there has been a systematic attempt to use Python APIs, a fact that leaves room for further study. An important element that makes this research scientifically interesting is the attempt to apply the methods in Cyber Range environments. It seems at this point that there is ample room for improvement in scientific knowledge and many opportunities for research.

CHAPTER 3

METHODOLOGY

3.1 TYPE OF RESEARCH

This will be a quantity research. After the initial deployment of OpenStack environment in the Cyber Range, we will try to measure how easier it is to use automation scripts written in Python to manage, create and control networks and network components through the OpenStack environment.

The networks created, will be then tested in a given Cyber Range in comparison with other networks preexisting in the deployment.

3.2 DEVELOPMENT LIFECYCLE

- As first step, the initial installation of OpenStack will take place. As Controller is picked a Virtual Machine deployed in the Cyber Range. The OS is Ubuntu Server LTS 18.04 fully patched with the latest updates.
- For Compute Node, a second Ubuntu Server VM will be used, running the same OS Version and patches. The Compute Node will act as an Availability Zone (AZ) for the deployment.
- After the installation is finished, the API endpoints created, will be tested . The scripts are written in Python, importing the library “requests” for the API HTTP calls and the library “json” so we can send and receive through the code json containers.

- After the network and network components are created , they will be imported in the virtualization environment as external provider network, trying to test their functionality and their efficiency.

The scripts have the following capabilities:

- Authenticating through OpenStack Keystone component. The fernet token received is saved in a variable to use in all the other scripts.
- Listing Networks: The network name, ID and description will be included in the result
- Creating network
- Listing Subnets
- Creating New Subnets and attaching them in an already created network.
- List virtual Routers
- Create Vistual Routers
- List ports
- Create New Ports

3.2.1 PROPOSED LIFECYCLE 1 - WATERFALL MODEL

As proposed from waterfall model every development step will be completed separately with its own mini-plan and each phase “waterfalls” into the next.

Step 1: Identify the current Problem.

Openstack environment can be extremely complex and time consuming, especially when dynamic performance analysis is the target. There is a need to add automations, written in universal code, able to run in the majority of systems, utilizing the API end points of the deployment.

Step 2: Plan

Analyzing the components that the scripts must create and manage. For the needs of our testing environment the code must have the ability to authenticate to the deployment using the Keystone identity of Openstack acquiring the needed keys. When the authentication is achieved the scripts must have the ability to list, create and manage network components (router, ports, networks, subnets), using the network API endpoint of the OpenStack deployment.

Step 3: Design

Following the plan and validating the requirements, Python is chosen as main scripting language as it is universal, easy to deploy and debug with numerous ready – to – use libraries powered by the community. For the authentication process a Linux bash script is used to get the authentication token and inject it to the main code. A menu helping the end user / tester to navigate through the different functions of the script must also be implemented.

Step 4: Build

First, we build the bash script that can be run either on Controller Node or in the Compute Node and will ask for the Access Key of the Deployment. This key will be injected in the main script code and used during the authentication API call to get the fernet token. The last part of the bash script will be the execution of the Python code. The main code is separated in many different functions, based on the job each one do. For example one function for listing networks, different one for creating . The user will have the ability to use and manage this capability using a “menu” listed in numbers. All the code will be within a loop.

Step 5: Code Test

The testing of the code will take place at the end of the building process. Each one of the functions will be tested and the results will be evaluated. If a defect or deficiency is found, then we fix the issues until the product meets the original specification.

Step 6: Software Deployment

The source code will be deployed immediately after finish testing in the “Production” environment by copying both bash script and Python source code to the VM.

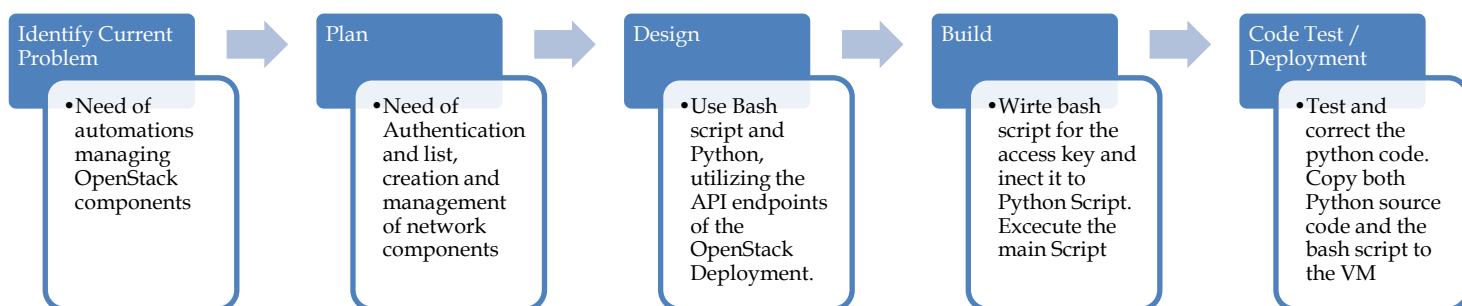


FIGURE 1 WATERFALL MODEL DEVELOPMENT LIFECYCLE

3.2.2 PROPOSED LIFECYCLE 2 - SPIRAL MODEL

This more complex model give emphasis on the repetition and continues improvements. The spiral model goes through the planning, design, build and test phases over and over, with gradual improvements at each pass.

For this model, while the identification of the problem remains the same, will be separated in smaller parts each one with its own plan, design, code building and testing.

- At the beginning we need the authentication (problem phase). An access code needs to be acquired in order to have successful API calls (Plan). This will be implemented using Linux Bash script in the Controller or Compute Node (Design). Create a bash script with the appropriate command asking from the stack the access code of the project. This code will be injected in the main Python script (Build). The access code will be shown as a return in the screen to verify the code. (Test).

- As the spiral continues, a fernet token is needed (problem phase 2). We will get this token making a call to the Keystone end point of our deployment (Plan phase 2). Python code with the suitable JSON libraries will be used to make the API call and decode the result, saving the fernet token in a variable (Design / Build). For testing purposes, the resulted token will be displayed on screen as the return of our code. (Test phase 2)
- The cycles of the deployment will continue for each of the needed components separately (build network, build subnet, build router, build port). Each one will be tested and be debugged before moving to the next one.
- Once all components are ready the finally source file will be copied in the VM, having the deployment completed.

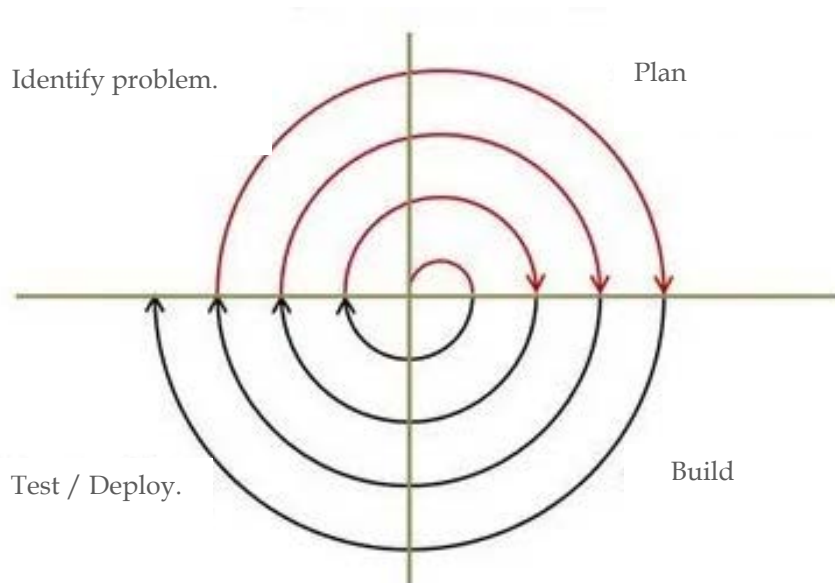


FIGURE 2 SPIRAL MODEL LIFECYCLE (BIG PICTURE)

3.2.3 CHOSEN MODEL

The model that we will follow in this environment is the Waterfall model. It is the oldest and most straightforward. The biggest drawback of this model is that it does not give the opportunity of continues improvement and that even if small details are left incomplete in each phase can hold up the entire process.

Although, evaluating the risk, and taking under consideration the simplicity of the project, this method gives the best possible results, in the fastest possible way. Other way, we will spend too much time in preparation without any actual need.

3.3 RESEARCH MAIN QUESTIONS

- How we can better and more efficient utilize Python automations in a OpenStack environment, deployed in a Cyber Range
- Is Python a good choice to script for OpenStack automations?
- Should we consider networks created with OpenStack better or worst in compare with the networks already installed in the Virtualization environment?
- Is the restful API calls a good choice to interact with OpenStack environments?

CHAPTER 4 DEPLOYMENT

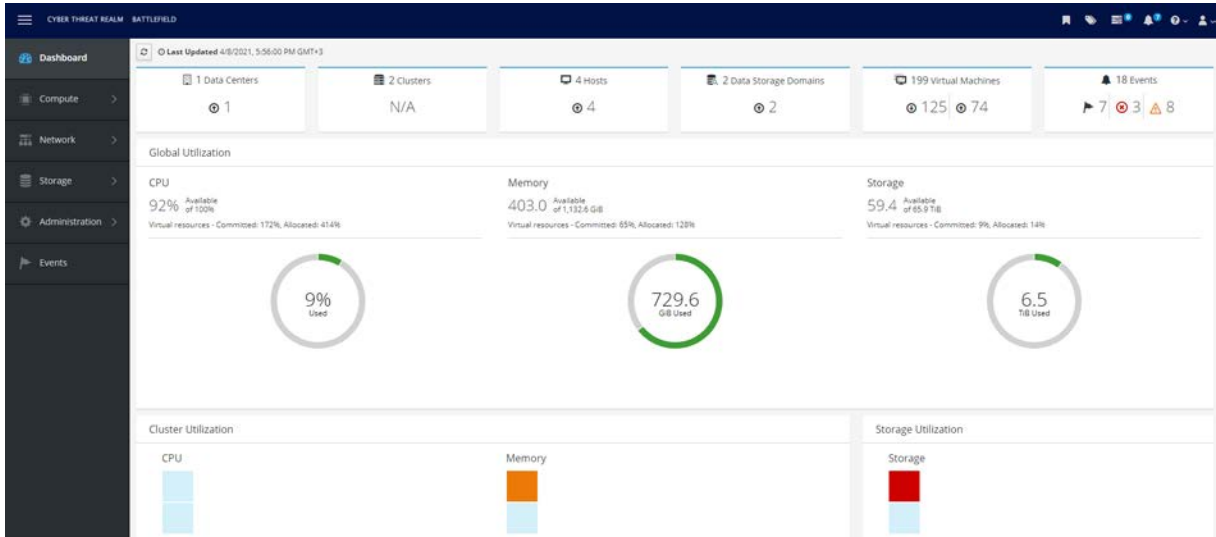
4.1 VIRTUALIZATION DEPLOYMENT

The servers that are hosting Openstack deployment are hosted in Virtual Machines of an already existent Cyber Range, also based on Openstack.

In more details, the virtualization environment is hosted in five (5) nodes running CentOS. From Network perspective there are two Switches, one for the internal network and the other for the external connectivity.

For storage management, there is deployed a storage domain that is hosting the “Virtualization Manager” using iSCSI. All five hosts have iSCSI as shared storage in the domain and there is where all

Virtual machines are stored. The virtualization environment also contains an ISO NFS for the images and an export storage NFS

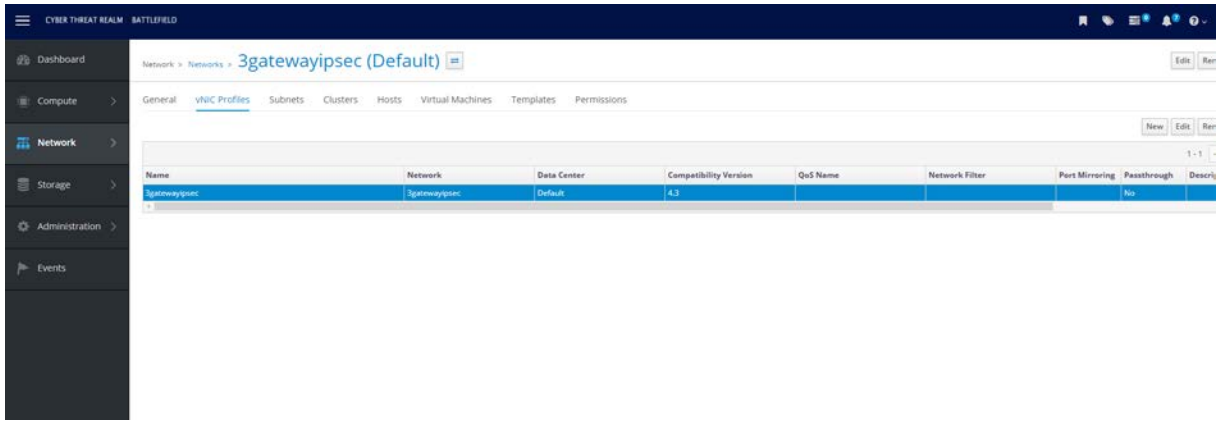


PICTURE 4 SCREENSHOT FROM THE VIRTUALIZATION ENVIRONMENT DASHBOARD

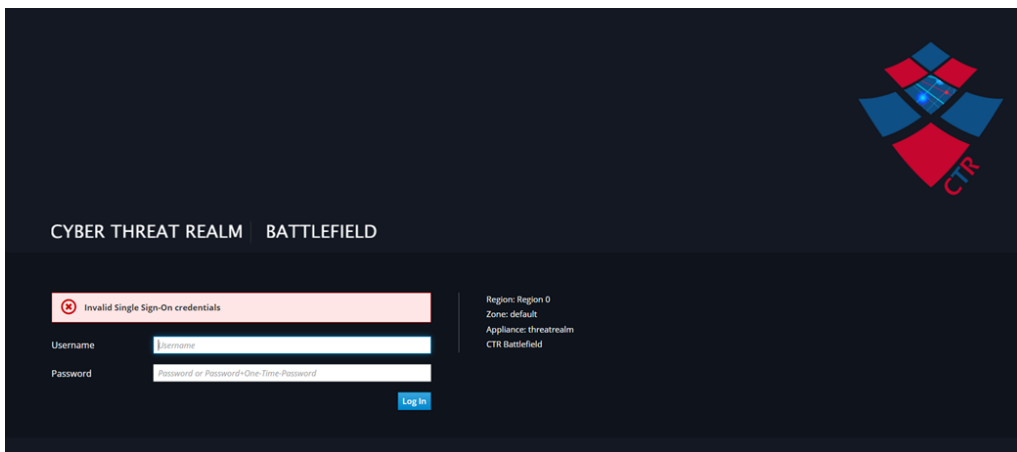
The screenshot shows the 'provider-ovn' network configuration page. It displays a table of networks with columns for Name, External ID, and Data Center. The table lists 20 networks, including CR10_1 through CR9_2, and two special networks: CyberRange1 and EXT_NET_TEST.

Name	External ID	Data Center
CR10_1	1c34917b-e0c3-415d-8159-eb8422ed37eb	Default
CR10_2	53fa72ee-c71d-40e8-a512-36ccc604f39e	Default
CR11_1	a891a058-7043-44ac-92df-cfd120391c54	Default
CR11_2	58d72705-9199-40c1-b248-a86f9795227c	Default
CR2_1	61a30e72-88e4-416d-b7c2-51f507867285	Default
CR2_2	267c6638-501b-405c-b44d-2cab206d3175	Default
CR3_1	743c7b0d-f22a-4369-81b1-74e5466d4481	Default
CR3_2	84378307-a537-4fc3-be9d-c36d12ec247e	Default
CR4_1	71923803-f126-4307-aa87-e95e160653fa	Default
CR4_2	00f3d710-d9f7-499e-a7dc-3c8687114dca	Default
CR5_1	4cd80fc1-2670-4555-9cfd-57562fa0971	Default
CR5_2	77f1e3b6-34d5-406b-a7d2-e3a41a5f0933	Default
CR6_1	e5371821-450e-4496-be54-43b09a39506	Default
CR6_2	c14fa807-3490-4252-a901-0b23e3b9ecfb	Default
CR7_1	2328bd65-2413-429f-b52a-8c0976a095e4	Default
CR7_2	5614976f-4bee-4079-4a73-45680fa61020	Default
CR8_1	5e412e85-b3da-481a-9e3a-d88951d3934	Default
CR8_2	b0390577-e00e-4abe-a40b-d73ad56127ec	Default
CR9_1	31dc15fc-439a-4c7b-9176-11410e777621	Default
CR9_2	eaa2b25b-9d76-43fb-84ff-0e669879f5d6	Default
CyberRange1	e2f5ba9e-245e-412b-8511-91466099156e	Default
EXT_NET_TEST	186b5586-3d39-439e-a002-337bcb9cc607	Default

PICTURE 5 NETWORKS CREATED AND STORED IN O VIRT



PICTURE 6 VIRTUAL GATEWAY IN OVRT



PICTURE 7 OVRT LOGIN PAGE

4.2 CONTROLLER NODE

For the controller Node installation, microstack snap for Ubuntu is used. This snap gives an easy to install and full manageable OpenStack environment provided by Canonical for development testing in systems running Ubuntu. Currently is only suitable to Ubuntu version 18.04 LTS.

After downloading the snap, an initial configuration is needed. This will install all the basic OpenStack components including Nova for computing, Neutron for virtual networking, keystone for authentication, cinder for imaging and horizon for the web GUI of the OpenStack. For networking and authentication, we set up an configuration file (local.conf).

The Virtual Machine currently working as Controller has 3 interfaces. Two of them are in the management network of the virtualization (network name: ovirtmgmt) and the third one is a different network also provided from virtualization (network name : studentlan). The whole deployment and the API endpoints that are created are based on interface enp1s0 with IP address 192.168.30.152 /24 .

After the initialization of the OpenStack completed, a virtual bridge is created so the control Node can communicate with the whole stack. For the bridge Open Virtual Switch is used. In the bridge interface enp9s0 – part of the ovirtmgmt vlan and the tap interface created from the stack were added. For the bridge to be accessible as interface, we set up an IP at the default external network created in the OpenStack Deployment process (10.20.20.1 /24)

```
bene@micro:~$ ifconfig
br-ex: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.20.20.1 netmask 255.255.255.0 broadcast 0.0.0.0
    inet6 fe80::ac80:d3ff:fe11:ee45 prefixlen 64 scopeid 0x20<link>
    ether ae:80:d3:11:ee:45 txqueuelen 1000 (Ethernet)
    RX packets 2037 bytes 138018 (138.0 KB)
    RX errors 0 dropped 70 overruns 0 frame 0
    TX packets 13 bytes 1006 (1.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.30.152 netmask 255.255.255.0 broadcast 192.168.30.255
    inet6 fe80::546f:3cff:fee5:9c prefixlen 64 scopeid 0x20<link>
    ether 56:6f:3c:e5:00:9c txqueuelen 1000 (Ethernet)
    RX packets 9418 bytes 5139219 (5.1 MB)
    RX errors 0 dropped 545 overruns 0 frame 0
    TX packets 3890 bytes 6313829 (6.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp9s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::546f:3cff:fee5:3 prefixlen 64 scopeid 0x20<link>
    ether 56:6f:3c:e5:00:03 txqueuelen 1000 (Ethernet)
    RX packets 2701 bytes 212689 (212.6 KB)
    RX errors 0 dropped 35 overruns 0 frame 0
    TX packets 28 bytes 1976 (1.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

PICTURE 8 IFCONFIG DOCUMENTATION FROM CONTROLLER NODE

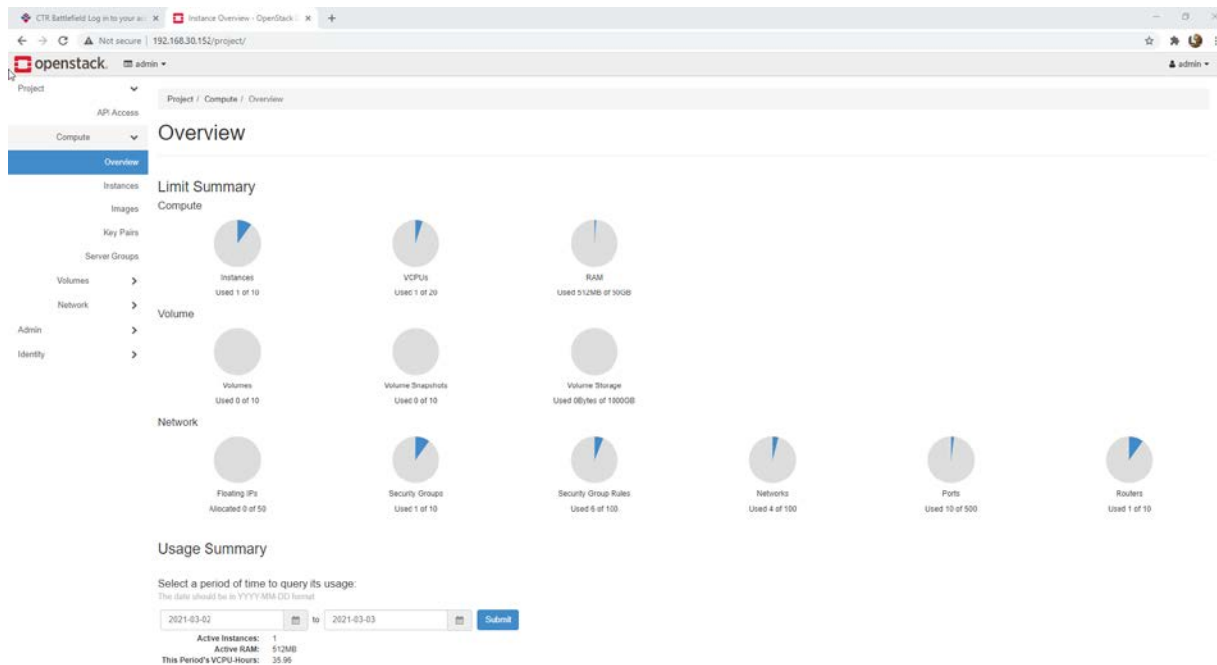
```

root@micro:/home/bene# microstack.ovs-vsctl show
57dd69d16-c82d-4941-89ab-b39b778bec18
  Bridge br-ex
    datapath_type: system
    Port patch-provnet-d8d36e7b-f70c-414d-b9fe-ec22f579bdcb-to-br-int
      Interface patch-provnet-d8d36e7b-f70c-414d-b9fe-ec22f579bdcb-to-br-int
        type: patch
        options: {peer=patch-br-int-to-provnet-d8d36e7b-f70c-414d-b9fe-ec22f579bdcb}
    Port br-ex
      Interface br-ex
        type: internal
    Port enp9s0
      Interface enp9s0
  Bridge br-int
    fail_mode: secure
    datapath_type: system
    Port tap467501f5-00
      Interface tap467501f5-00
        error: "could not open network device tap467501f5-00 (No such device)"

```

PICTURE 9 OVS BRIDGE CONFIGURATION IN CONTROLLER NODE

At this point the OpenStack Horizon , is accessible through web browser. Any changes made using the GUI will have immediate impact in the controller node, as tap interfaces will be automatically created.



PICTURE 10 HORIZON DASHBOARD

To access networks and interfaces created with OpenStack from the controller Node, static routes are added, setting as egress interface the br-ex and as next hop IP address the 10.20.20.20, that is the external's network interface in the virtual router in OpenStack.

As shown in the scripts bellow, packets from controller node can reach every virtual interface created with OpenStack GUI:

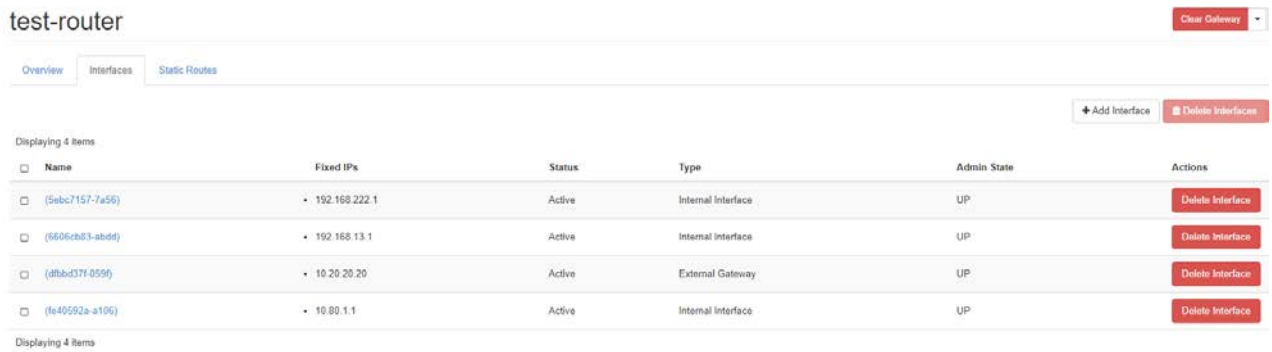


FIGURE 11 TEST INTERFACES IN VIRTUAL ROUTER CREATED IN OPENSTACK

```

root@micro:/home/bene# ping 192.168.13.1
PING 192.168.13.1 (192.168.13.1) 56(84) bytes of data.
64 bytes from 192.168.13.1: icmp_seq=1 ttl=254 time=0.872 ms
64 bytes from 192.168.13.1: icmp_seq=2 ttl=254 time=0.525 ms
64 bytes from 192.168.13.1: icmp_seq=3 ttl=254 time=0.474 ms
64 bytes from 192.168.13.1: icmp_seq=4 ttl=254 time=0.470 ms
64 bytes from 192.168.13.1: icmp_seq=5 ttl=254 time=0.498 ms
^X^C
--- 192.168.13.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4102ms
rtt min/avg/max/mdev = 0.470/0.567/0.872/0.156 ms
root@micro:/home/bene#

```

FIGURE 12 SUCCESSFUL PING FROM CONTROLLER TO CREATED SUBNET 192.168.13.0/24

4.3 COMPUTE NODE

One of the most significant advantage of cloud computing is the possibility to deploy in multiple availability zones in order to achieve greater availability and fault tolerance in the production environments. For this research, a second node is deployed acting as compute node in the stack. This enables a second Availability Zone (AZ)

A second Ubuntu 18.04 LTS server is deployed, fully patched at the latest updates. As in the controller node, snap image of microstack is used for this deployment too. The Compute VM has two network interfaces in the ovirtmngm vlan. The Compute Node is clustered, configuring the local.conf file to match the needs of this research.

```

root@micro:/home/bene# microstack.openstack hypervisor list
+-----+-----+-----+-----+
| ID | Hypervisor Hostname | Hypervisor Type | Host IP | State |
+-----+-----+-----+-----+
| 1 | micro | QEMU | 192.168.30.152 | up |
| 2 | compute.sec.ouc.ac.cy | QEMU | 192.168.30.153 | up |
+-----+-----+-----+-----+
root@micro:/home/bene# █

```

PICTURE 13 HYPERVISORS LIST FROM CONTROLLER

4.4 SCRIPTS AND CODE

For the scripts, Python 3 is used, having imported JSON library and Requests library for the API calls.

4.4.1 CONNECTION – GETTING THE AUTHENTICATION TOKEN

```

payload = {
  "auth": {
    "identity": {
      "methods": [
        "password"
      ],
      "password": {
        "user": {
          "name": "user",
          "domain": {
            "name": "Default"
          },
          "password": "a"
        }
      },
      "scope": {
        "project": {
          "domain": {
            "id": "default"
          },
          "name": "admin"
        }
      }
    }
  }
}

ipl=('http://'+ip+':5000/v3/auth/tokens')
#The actual HTTP POST request - part of "requests" Python library
res = requests.post(ipl,
                    headers = {'content-type':'application/json'},
                    data=json.dumps(payload))

#Getting from returned Header the value of Token - stored in variable called token
token=res.headers['X-Subject-Token']

print ("You got authenticated -- Your token is: ",token)

```

PICTURE 14 AUTHENTICATION

In this part, we set in the container payload some basic parameters for authentication such as user name, password, the Openstack project name along with OpenStack account

name and ID. The input is structured in JSON format. It is important to have already imported JSON library for that.

In the variable “ip1” we set the IP of the API endpoint giving the opportunity to the end user to enter the actual IP address of the controller node, in case the server is migrated. The rest of the URL indicates the path to keystone authentication system. This gives to the script an elasticity and fault tolerance.

In the variable “res” we have the actual REST request. Both Requests and JSON libraries are used. The result of this request will be JSON formatted, that is the reason why we filter the output, saving only “X-Subject-Token” in the “token” variable. With the token acquired from this step we will work in all scripts.

4.4.2 NETWORK CREATION

```
def network_create(dose):

    ip2=('http://'+dose+'9696/v2.0/networks')
    netname = input("Please assign a name to network")
    descrip = input("Please describe your network")
    dnsdomain = input("Enter DNS domain")

    netcreate= {
        "network": {
            "name": netname,
            "description": descrip,
            "dns_domain": 'my-domain.org.',
            "mtu": 1400
        }
    }

    res = requests.post(ip2,
                        headers={'content-type': 'application/json',
                                'X-Auth-Token': token
                                },
                        data=json.dumps(netcreate)
    )
    print(res.text)
```

FIGURE 15 NETWORK CREATION

This is the function created for new network creation. In the “netcreate” variable, we set the basic information about the request, regarding the new network name, a domain – name, a verbal description of the network along with the MTU size, loaded in JSON format.

In the “res” variable the result of the request is saved. Using POST we insert the request to OpenStack. For the authentication the fernet token acquired during authentication process is used.

To have visual verification of the success, the reply, again in JSON format is printed on screen.

4.4.3 LIST NETWORKS

```
def network_list(dose):
    ip3=('http://'+dose+':9696/v2.0/networks.json?fields=id&fields=name')
    res1 = requests.get(ip3,
                        headers={'content-type': 'application/json',
                                'X-Auth-Token':token
                                },
                        )
    print (res1.text)
```

PICTURE 16 LIST NETWORKS

Function to generate a list with all the available networks. In the URL of the request, filters are used so to minimize results only network IDs and network names. For the authentication again the fernet token is used

4.4.4 SUBNET LIST

```
def subnet_list(dose):
    ip9=('http://'+dose+':9696/v2.0/subnets.json?fields=id&fields=name')
    res9 = requests.get(ip9,
                        headers={'content-type': 'application/json',
                                'X-Auth-Token': token,
                                },
                        )
    print (res9.text)
```

PICTURE 17 LIST SUBNETS

Lists all available subnets filtering only to display the name and the ID. It is important that the ID is displayed as this is the information that will be used to attach this subnet to a router. Authentication is achieved with the fernet token.

4.4.5 SUBNET CREATE

```
def subnet_create(dose):
    answer1=input("If you know you network's ID please press ENTER, To display the available networks press 1: ")
    if (answer1=="1") :
        network_list(dose)

    netid=input("Please insert your Network ID: ")
    cidr=input("Enter network address - Dotted Decimal")
    start=input("Enter starting address")
    end=input("Enter ending address")
    ipver=input("Enter The IP version")
    subname=input("Enter Subnet Name")
    subnetcr= {
        "subnet": {
            "name": subname,
            "network_id": netid,
            "ip_version": ipver,
            "cidr": cidr,
            "allocation_pools": [{"start": start, "end": end}]
        }
    }
    ip4=('http://'+dose+'9696/v2.0/subnets')

    res4 = requests.post(ip4,
        headers={'content-type': 'application/json',
                'X-Auth-Token': token,
                },
        data=json.dumps(subnetcr)
    )
    print(res4.text)
```

PICTURE 18 CREATE SUBNET

In the POST request to create a new subnet the minimum information required are: Name, Network ID, IP version, the network address followed by subnet mask to identify the CIDR and if DHCP is enabled a starting and ending address. The data structure used again is a container in JSON format. The authentication is made utilizing again the fernet token from keystone.

4.4.6 ROUTER LIST

```
def router_list(dose):
    ip7=('http://'+dose+':9696/v2.0/routers.json?fields=id&fields=name')

    res7 = requests.get(ip7,
                        headers={'content-type': 'application/json',
                                'X-Auth-Token': token,
                                },
                        )
    print(res7.text)
```

PICTURE 19 LIST ROUTERS

With this GET request, all available routers are displayed. With the filter applied in the URL, only IDs and router names will be displayed.

4.4.7 ROUTER CREATE

```
def router_create(dose):
    rname=input("Please provide with new routers name: ")
    ntidr=input("Please Provide network to add to router: ")

    routercr= {
        "router": {
            "name": rname,
            "external_gateway_info": {
                "network_id": ntidr,
                "enable_snat": True,
            },
            "admin_state_up": True
        }
    }
    ip5=('http://'+dose+':9696/v2.0/routers')

    res5 = requests.post(ip5,
                        headers={'content-type': 'application/json',
                                'X-Auth-Token': token,
                                },
                        data=json.dumps(routercr)
                        )
    print(res5.text)
```

PICTURE 20 ROUTER CREATE

Creation of an external gateway requires the following parts: Network ID, state condition (Up/Down), NAT state, Name. The authentication to API end point is made with fernet token.

4.4.8 PORT LIST

```
def port_list(dose):
    ip10=('http://' +dose+' :9696/v2.0/ports.json?fields=id&fields=name')
    res10 = requests.get(ip10,
        headers={'content-type': 'application/json',
                 'X-Auth-Token': token,
                },
    )
    print(res10.text)
```

PICTURE 21 LISTS PORTS

4.4.9 PORT CREATE

```
def port_create(dose):
    pname=input("Please insert your Port Name ")
    netid=input("Enter network ID")
    devid=input("Please enter Device ID")
    portcr= {
        "port": {
            "name": pname,
            "network_id": netid,
            "device_id": devid,
        }
    }
    ip14=('http://' +dose+' :9696/v2.0/ports')
    res14 = requests.post(ip14,
        headers={'content-type': 'application/json',
                 'X-Auth-Token': token,
                },
        data=json.dumps(portcr)
    )
    print(res14.text)
```

PICTURE 22 PORT CREATE

4.4.10 SCRIPT STRUCTURE

To facilitate the usage of scripts, instead of separate execution, a simple decision making function is used so the user can call the script and perform all the available actions:

```

if (answer=="1"):
    network_create(ip)
elif(answer=="2"):
    network_list(ip)
elif (answer=="3"):
    subnet_create(ip)
elif (answer=="4"):
    router_create(ip)
elif (answer=="5"):
    router_list(ip)
elif (answer=="6"):
    subnet_list(ip)
elif (answer=="7"):
    port_list(ip)
elif (answer=="8"):
    port_create(ip)

```

PICTURE 23 IF STRUCTURE

CHAPTER 5

TESTING

In this testing will demonstrate the creation of a network, a router and then a subnet using Python scripts in Openstack. After the creation of the network components, will test connectivity creating a virtual machine inside OpenStack. The scripts will run from Compute Node.

5.1 GETTING THE AUTHENTICATION TOKEN.

To get the authentication Token, we will initiate the process running a simple bash script. This will take the randomly generated OpenStack password and insert it to main Python script that will be executed afterword.

The script:

“

As first part of the script we need to access the "identity" API and get the aut_token.

#In order to do that we will use a CLI script integrating the password to Python script


```
#!/bin/bash
```

```
#Getting the password - Currently working only in Controller
```

```
pass=$(sudo snap get microstack config.credentials.keystone-password)
```

```
echo "Your password is: "$pass
```

```
#Inserting the password as variable a in the python script
```

```
sed -i "4i a=""$pass"" script.py
```

```
#Running Python script to get the Auth Token
```

```
python3 script.py
```

“

```
root@compute:/home/bene# ./id.sh
Your password is: eNQLImj2pQyLD6NBpqhXyFb4dQ6dWLEN
Please Enter your username: admin
Please enter the controllers IP - Dotted decimal: 192.168.30.152
You got authenticated -- Your token is: gAAAAABgT0EJZ0mriHaXsu5spXnfrwkgwQ-DaKjy-Xb6naUpSZJaW_zXBVTCKjAu5TWV1CWdSJg
-CJdBxZli4lVUoxHTUmM3mHnke6UeTI6lL6X6ucdLzdxbjUTtdrS6cT5qZs2U97lFQp13v6rAH-kA86q4E8YjWxZAQe4IUJ9yZ0vbikAx0FA
To create a network press 1
To Display All Available Networks Press 2
To Create subnet Push 3
To create a router press 4
To List all available Routers press 5
To List All Available Subnets press 6
To List all available ports press 7
To create a new port press 8: [ ]
```

PICTURE 24 SCRIPT RUN

After the password is copied to main script, a username will be asked. In our case this is “admin”. After entering the username there is a prompt to enter controller’s IP address. If credentials are correct the Auth Key is displayed along with the “main menu” screen, asking to choose the next actions. (Picture 24)

In pictures bellow, there are the list of all the available networks, subnets, and ports in the deployment.

```
{"networks":[{"id":"283a23e4-870e-4f4d-97ee-26277da49bad","name":"test2_01_03"}, {"id":"3769c880-3aa3-4c8d-9d68-6e9ae
b18ccad","name":"external"}, {"id":"ab298f55-041c-478d-9132-35b2ef771127","name":"test"}, {"id":"eaf43c8e-2acb-4b58-ac
53-c2db6afa7eb9","name":"Benetatos_TEST"}]}
root@compute:/home/bene# [ ]
```

PICTURE 25 LIST OF AVAILABLE NETWORKS

```
{
  "routers": [
    {
      "id": "3f2a430c-fbc7-4772-a800-4687ale51855",
      "name": "GUI_Router"
    },
    {
      "id": "d8ce26fb-7779-449f-a34d-8922b5e60259",
      "name": "test-router"
    }
  ]
}
```

FIGURE 26 LIST OF AVAILABLE ROUTERS

```
{
  "subnets": [
    {
      "id": "060b5f3d-6fe8-40bd-a9a0-c8f4210b6386",
      "name": "external-subnet"
    },
    {
      "id": "5cbc2a08-ad5c-4152-b421-b3694343b0a0",
      "name": "test-subnet"
    },
    {
      "id": "a0befcbf-d42f-4f8c-a964-bf5907bc25d7",
      "name": "sub1"
    },
    {
      "id": "dcea05f4-ac9c-47fc-b515-00c0d2e59f8c",
      "name": "Testing_sub"
    }
  ]
}
```

FIGURE 27 LIST OF AVAILABLE SUBNETS

```
{
  "ports": [
    {
      "id": "147211b0-5755-4503-9850-a3618123d946",
      "name": "nic5"
    },
    {
      "id": "1a9f6a7c-9ba5-4e3b-9c0e-aca2a69b3155",
      "name": "nic4"
    },
    {
      "id": "25a77904-bb8a-4d66-9c55-88204df881ae",
      "name": ""
    },
    {
      "id": "3a07889a-835a-460f-933f-83fbd1982c04",
      "name": ""
    },
    {
      "id": "5ebc7157-7a56-4942-a086-27f50d432b30",
      "name": ""
    },
    {
      "id": "6606cb83-abdd-46f3-a53a-54ecfe62c496",
      "name": ""
    },
    {
      "id": "991fe051-17c1-4915-a35a-635f766af6f5",
      "name": ""
    },
    {
      "id": "b349890c-6b9a-4bf1-b5c6-b2fee813cfbb",
      "name": ""
    },
    {
      "id": "df34443a-e36c-4b9a-b571-18a8545ab5f4",
      "name": ""
    },
    {
      "id": "dfbbd37f-059f-4b66-a34e-368d109002e6",
      "name": ""
    },
    {
      "id": "efc8d246-93e0-4db5-a0bf-acdb3d537738",
      "name": ""
    },
    {
      "id": "fe40592a-a106-4f70-b65a-9bc9e9063dd8",
      "name": ""
    }
  ]
}
```

FIGURE 28 LIST OF AVAILABLE PORTS

Displaying 4 items

<input type="checkbox"/>	Name	Subnets Associated	Shared	External
<input type="checkbox"/>	test2_01_03	sub1 10.80.1.0/24	Yes	No
<input type="checkbox"/>	external	external-subnet 10.20.20.0/24	Yes	Yes
<input type="checkbox"/>	test	test-subnet 192.168.222.0/24	Yes	No
<input type="checkbox"/>	Benetatos_TEST	Testing_sub 192.168.13.0/24	Yes	No

Displaying 4 items

FIGURE 29 NETWORKS AS SHOWN FROM OPENSTACK HORIZON

5.2 CREATING NETWORK

Name: DEMO

Description: DEMO for the presentation

Domain: my-org.com

```
Please assign a name to networkDEMO
Please describe your networkDEMO network for presentation
Enter DNS domainmy-org.com
{"network":{"id":"226bfeb4-9e81-4035-aad7-a3a85e432786","name":"DEMO","tenant id":"ede3f1c0c53a47e68ed7da7bb6dc25dd","admin state up":true,"mtu":1400,"status":"ACTIVE","subnets":[],"shared":false,"project id":"ede3f1c0c53a47e68ed7da7bb6dc25dd","router:external":false,"provider network type":"geneve","provider:physical network":null,"provider:segmentation id":4,"availability zone hints":[],"is default":false,"availability zones":[],"ipv4 address scope":null,"ipv6 address scope":null,"description":"DEMO network for presentation","tags":[],"created at":"2021-03-15T11:47:49Z","updated at":"2021-03-15T11:47:49Z","revision number":1}}
```

FIGURE 30 DEMO NETWORK CREATED SUCCESSFULLY.

At this time this newly created network has no subnets associated to. At the next step, a new subnet will be created and attached to this network.

5.3 CREATE SUBNET

Name: DEMO_sub

CIDR: 172.16.0.0/24

Starting: 172.16.0.2 // **Ending:** 172.16.0.254

```
If you know you network's ID please press ENTER, To display the available networks press 1: 1
{"networks":[{"id":"226bfeb4-9e81-4035-aad7-a3a85e432786","name":"DEMO"}, {"id":"283a23e4-870e-4f4d-97ee-26277da49bad","name":"test2_01_03"},
{"id":"3769c880-3aa3-4c8d-9d68-6e9aeb18ccad","name":"external"}, {"id":"ab298f55-041c-478d-9132-35b2ef771127","name":"test"}, {"id":"eaf43c8
e-2acb-4b58-ac53-c2db6afa7eb9","name":"Benetatos_TEST"}]}
Please insert your Network ID: 226bfeb4-9e81-4035-aad7-a3a85e432786
Enter network address - Dotted Decimal172.16.0.0/24
Enter starting address172.16.0.2
Enter ending address172.16.0.254
Enter The IP version4
Enter Subnet NameDEMO_sub
{"subnet":{"id":"6500898c-6bff-4dcc-9683-452db1588e4f","name":"DEMO_sub","tenant_id":"ede3f1c0c53a47e68ed7da7bb6dc25dd","network_id":"226bf
eb4-9e81-4035-aad7-a3a85e432786","ip_version":4,"subnetpool_id":null,"enable_dhcp":true,"ipv6_ra_mode":null,"ipv6_address_mode":null,"gatew
ay_ip":"172.16.0.1","cidr":"172.16.0.0/24","allocation_pools":[{"start":"172.16.0.2","end":"172.16.0.254"}],"host_routes":[],"dns_nameserve
rs":[],"description":"","service_types":[],"tags":[],"created_at":"2021-03-15T11:53:42Z","updated_at":"2021-03-15T11:53:42Z","revision_numbe
r":0,"project_id":"ede3f1c0c53a47e68ed7da7bb6dc25dd"}}
```

PICTURE 31 CREATE SUBNET AND ASSOCIATE TO NETWORK

Networks

Displaying 5 items

<input type="checkbox"/>	Name	Subnets Associated	Shared	External
<input type="checkbox"/>	DEMO	DEMO_sub 172.16.0.0/24	No	No
<input type="checkbox"/>	test2_01_03	sub1 10.80.1.0/24	Yes	No

PICTURE 32 THE CREATED NETWORK FROM OPENSTACK HORIZON

5.4 CREATE NEW ROUTER – EXTERNAL NETWORK

Router is created on the external network (192.168.30.0/24). The given name is :”DEMO Router”.

```
Choose: 4
Please provide with new routers name: DEMO Router
Please Provide network to add to router: 3769c880-3aa3-4c8d-9d68-6e9aeb18ccad
{"router": {"id": "93648b34-81d0-4516-87ae-e30b076f7eea", "name": "DEMO Router", "tenant_id": "ede3f1c0c53a47e68ed7da7bb6dc25dd", "admin_state_up": true, "status": "ACTIVE", "external_gateway_info": {"network_id": "3769c880-3aa3-4c8d-9d68-6e9aeb18ccad", "external_fixed_ips": [{"subnet_id": "060b5f3d-6fe8-40bd-a9a0-c8f4210b6386", "ip_address": "10.20.20.162"}]}, "enable_snat": true}, "description": "", "availability_zones": [], "availability_zone_hints": [], "routes": [], "tags": [], "created_at": "2021-03-16T06:58:27Z", "updated_at": "2021-03-16T06:58:28Z", "revision_number": 3, "project_id": "ede3f1c0c53a47e68ed7da7bb6dc25dd"}}
```

PICTURE 33 SUCCESSFUL ROUTER CREATION

Name	Status	External Network	Admin State	Availability Zones
GUI_Router	Active	external	UP	-
DEMO_Router	Active	external	UP	-
test-router	Active	external	UP	-

PICTURE 34 ROUTERS FROM HORIZON

5.5 CREATING INSTANCE IN OPENSTACK (FROM HORIZON)

To check the internal communication of the new built network components, we set up a Virtual Machine using OpenStack NOVA compute and the default “cirros” Unix image. In the VM we will give one vNIC attached in the DEMO network. The IP will be allocated automatically through DHCP.

Launch Instance ✕

Please provide the initial hostname for the instance, the availability zone where it will be deployed, and the instance count. Increase the Count to create multiple instances with the same settings. ?


Instance Name *

Description

Availability Zone

Count *

Total Instances (10 Max)



20%

■ 1 Current Usage
■ 1 Added
■ 8 Remaining

PICTURE 35 CREATING INSTANCE FROM HORIZON

Instances

Instance ID ▾

Displaying 2 Items

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
<input type="checkbox"/>	testing	cirros	10.20.20.148	m1.small	-	Active	nova	None	Running	0 minutes	<input type="button" value="Create Snapshot"/>
<input type="checkbox"/>	DEMO	cirros	172.16.0.108	m1.tiny	-	Active	nova	None	Running	1 minute	<input type="button" value="Create Snapshot"/>

Displaying 2 Items

PICTURE 36 INSTANCE (VM) CREATED SUCCESSFULLY

From controller Node, we add a static route for network 172.16.0.0/24 pointing as next hop IP address the 10.20.20.19 (router DEMO interface) and as exit interface the external bridge we have created (br-ex : 10.20.20.1). As per bellow screen shot traffic flows from Controller Node to the newly installed VM.

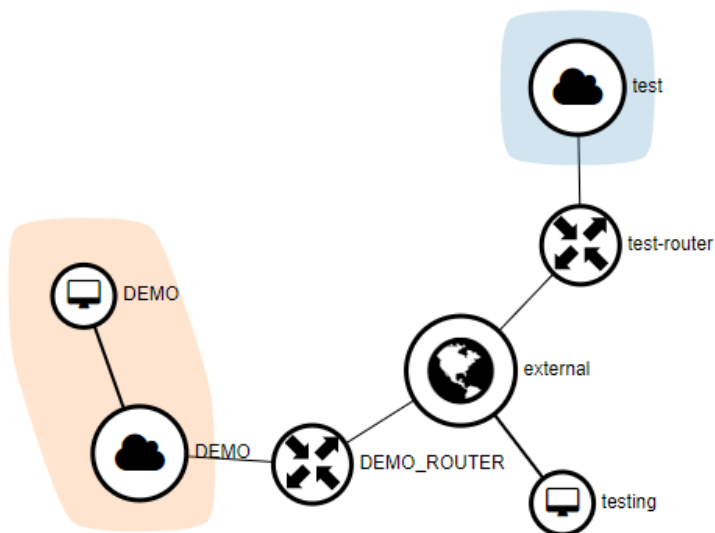
```

root@bene:/home/bene# ping 172.16.0.108
PING 172.16.0.108 (172.16.0.108) 56(84) bytes of data.
64 bytes from 172.16.0.108: icmp_seq=1 ttl=63 time=3.37 ms
64 bytes from 172.16.0.108: icmp_seq=2 ttl=63 time=1.68 ms
64 bytes from 172.16.0.108: icmp_seq=3 ttl=63 time=0.836 ms
64 bytes from 172.16.0.108: icmp_seq=4 ttl=63 time=0.703 ms
64 bytes from 172.16.0.108: icmp_seq=5 ttl=63 time=0.707 ms
64 bytes from 172.16.0.108: icmp_seq=6 ttl=63 time=0.571 ms

```

PICTURE 37 FROM CONTROL NODE TO VM IN OPENSTACK WITHIN THE 172.16.0.0/24 NETWORK

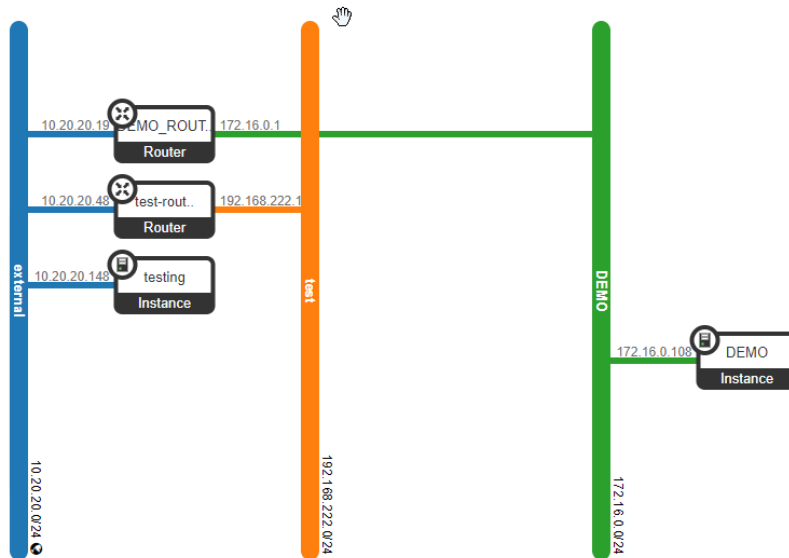
5.6 NETWORK DIAGRAM FROM OPENSTACK HORIZON



PICTURE 38 NETWORK DIAGRAM OF THE NEW BUILT COMPONENTS

From the diagram above, we can see DEMO_ROUTER that was created using the scripts, having two different interfaces. One in the external network subnet, so the traffic can be routed outside the tenant network too, and one with the network 172.16.0.0/24 (DEMO) that we have also created using the scripts.

The Virtual Machine DEMO has an interface in the DEMO network and subnet, also created using the Python Scripts.



PICTURE 39 NETWORK TOPOLOGY, VISUALIZED THROUGH OPENSTACK HORIZON

5.7 CHECKING NETWORK WITH IPERF

To check the newly created network, two more new VMs are created in the OpenStack, adding a vNIC to each one pointing the 172.16.0.0/24 network. Iperf3 is used in all testes.

To install iperf3 in the new instances, ubuntu 18.4 OS is installed in both. For the installation Canonical, ready-to-deploy images are used.



PICTURE 40 NEW NETWORK TOPOLOGY (2 VM ADDED)

All results are displayed in graphs, containing statistics for throughput as well as Latency in 3 main points.

- Communication between Virtual Machine and Control Node of Openstack
- Communication between 2 VM hosted in the stack.
- Communication between 2 VMs in Network already existed in Virtualization

For all measurements, statistics will be taken for 128, 256, 512, 1024 KB packets. The basic structure of the commands that will be used in iperf3 is:

Server Side:

```
#iperf3 -p 5203 -s -A 0
```

-p: Determines the port that iperf will be listening to

-s: Indicates that this box runs as Iperf3 server

-A: Set the CPU affinity

Client Side:

```
# iperf3 -f m -c [iperf3 server address] -p 5203 -t 31 -i 31 -O 1 -M [value] -l [value] -P 8 -T 0 -A 0
```

-f: Shows the measurement rate (m stands for Mbit)

-c: Indicates that this is a iperf3 client

-p: Communication port used

-t: Time in seconds for transmit

-i: Interval between reports (eg show results every 1 second)

-O: Omit seconds to avoid TCP slow start

-M: Sets the TCP Maximum segments size

-l: Change buffer size for read and write

-P: Number of simultaneous connection to the server

-T: Time to Live

-A: Set the CPU affinity

-u: Utilize UDP instead of TCP

5.7.1 LATENCY

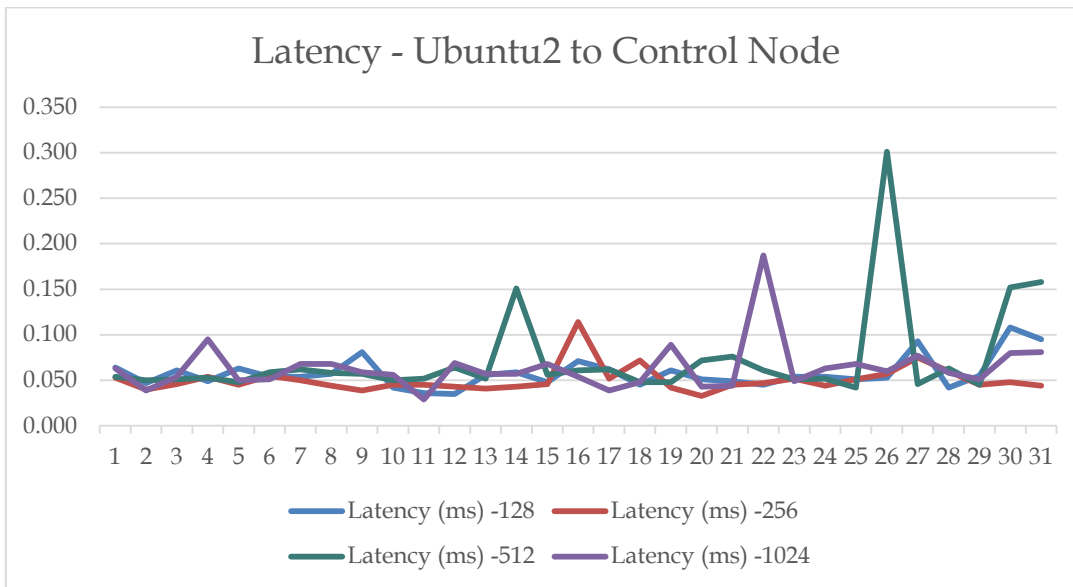


FIGURE 3 LATENCY GRAPH MEASURING CONNECTION LATENCY BETWEEN CONTROL NODE (10.20.20.1) AND UBUNTU1 INSTANCE (172.16.0.220). THE PROTOCOL USED FOR THIS IS UDP. AS IT SEEMS, THERE IS NOW SIGNIFICANT AND VISUAL DIFFERENCE BETWEEN MEASUREMENTS OF DIFFERENT SIZES.

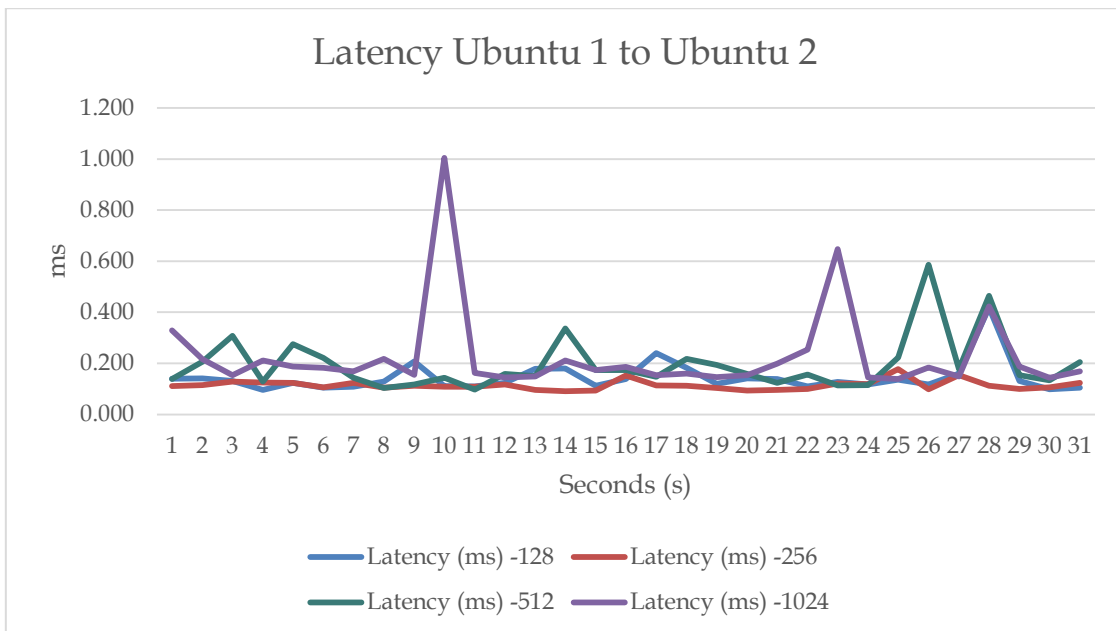


FIGURE 4 LATENCY GRAPH BETWEEN UBUNTU 1 AND UBUNTU 2 VPM. UDP IS ALSO USED. AT 1024 KB THERE IS A SLIGHT DIFFERENCE THAT IS VISIBLE THROUGH THE CHART TOO

In Figure 3, is displayed the latency between instance “Ubuntu 2” with IP address 172.16.0.220 and the bridge interface that lays on the controller (10.20.20.1). The choice of Ubuntu 2 instance is not relevant, as all created instances in the stack, return the same network results and statistics.

Using UDP, the results displayed in the chart are divided in four different sizes starting with 128 KB, 256 KB, 512 KB and finally 1024 KB. The testing period is 31 sec, with the polling interval to be 1 second.

Comparing results, there is no significant difference between the sizes. The average latency remains around 0.05 ms. The peaks that are displayed are random and it seems that there is no connection to the size of the data transmitted.

In Figure 4, is displayed the latency between two instances created in the stack, both having network interfaces that lay in 172.16.0.0/24 network. More specific instance Ubuntu1 (172.16.0.183) communicates with instance Ubuntu2 (172.16.0.220). In this communication none of controller’s interface are used.

Again, data of different sizes (128 KB, 256 KB, 512 KB, 1024 KB) are used. Testing period is 31 seconds and polling interval remains at 1 second. Comparing the results, there is no significant variation between the latency, with average value to be around 0.150ms . We can see some random peaks, but it seems there is no visible connection with packet sizes.

Comparing Figure 3 and Figure 4 together we can figure that there is a significant difference. In the communication between the two virtual machines inside the stack there is bigger latency. This is probably caused due to miss – optimization of the flat networks inside the created vlan.

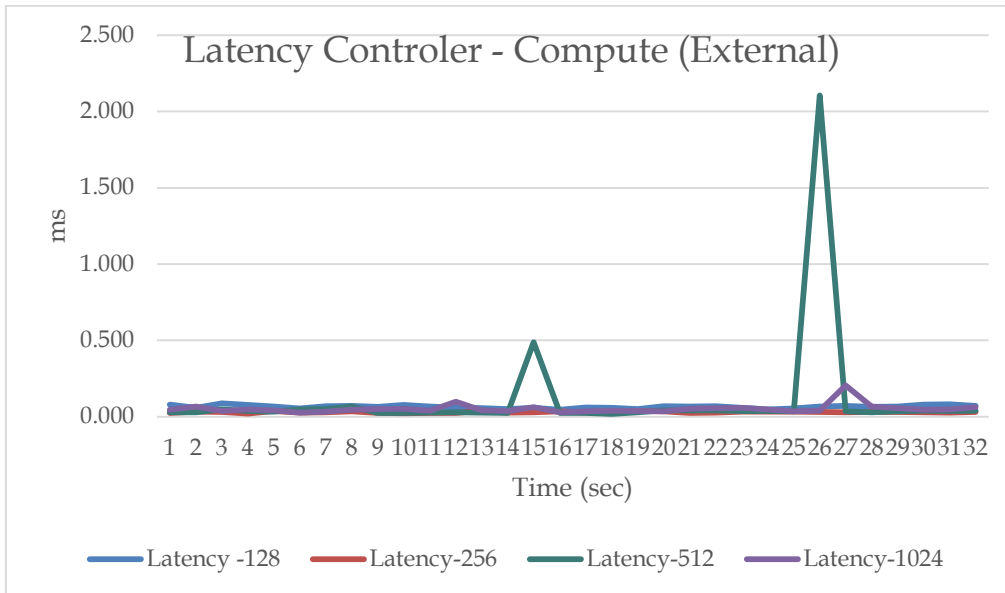


FIGURE 5 LATENCY IN MS IN CONNECTION BETWEEN CONTROLLER AND COMPUTE NODE, THROUGH EXTERNAL NETWORK 192.168.30.0/24 THAT WAS ALREADY HOSTED IN THE VIRTUALIZATION

Comparing the results of latency in ms, for the connection between Control Node and Compute Node, utilizing network interfaces attached to network 192.168.30.0/24, that was already hosted in virtualization environment, we can see that there is no significant difference between the results we already collected measuring connections inside the OpenStack environment. The values seem to be stable. We can see a peak around 25 and 27 sec but this could be considered as random.

5.7.2 THROUGHPUT

Throughput will be measured in the connection between the Control Node with IP 192.168.30.47 (in the External network) and 172.16.0.220 vNIC, attached to VM ubuntu2. Also throughput is measured in the connection between the two Virtual Machines hosted in OpenStack (Ubuntu1 and Ubuntu2). The measurements will contain packets of 128, 256, 512, 1024 KB.

For testing purposes, a figure measuring Throughput between 2 virtual machines in an existing network inside virtualization will also be displayed.

The mathematical formula used is:

$$\frac{(\text{sender}(\text{upload}) + \text{receiver}(\text{download}))}{2}$$

$$\text{Result} * \text{Conversion Multiplier}$$

Target Test (L3 length)	L3 overhead multiplier	L2 overhead multiplier
128	1.6842	1.8684
256	1.2549	1.3235
512	1.1130	1.1435
1024	1.0535	1.0679
1500	1.0359	1.0456
9000	1.0058	1.0074

PICTURE 41 CONVERSIONAL MULTIPLIER

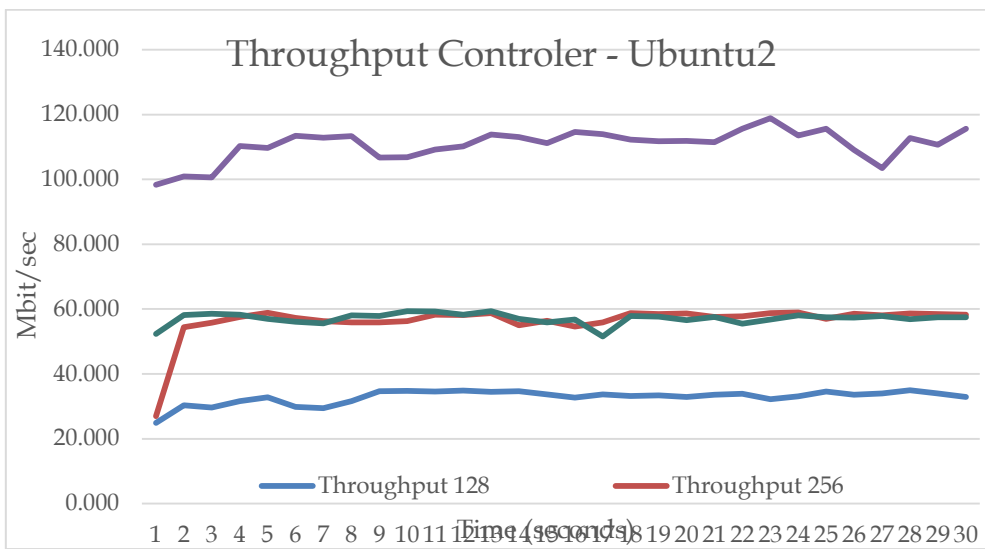


FIGURE 6 THROUGHPUT FROM CONTROLLER TO UBUNTU2 VM CREATED AND HOSTED IN THE STACK. MEASUREMENTS FOR 128, 256, 512, 1024 KB

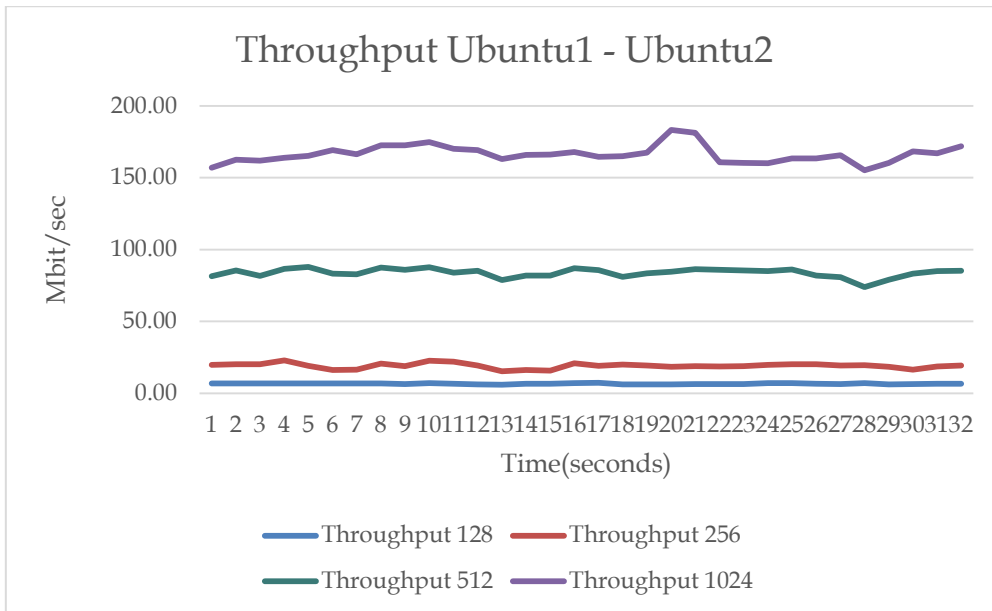


FIGURE 7 THROUGHPUT FROM UBUNTU1 TO UBUNTU2 VM CREATED AND HOSTED IN THE STACK. MEASUREMENTS FOR 128, 256, 512, 1024 KB

In figure 5, throughput of the connection Controller-Ubuntu2 is displayed. We measure the throughput in Mbps for 31 seconds time period. Different colored line indicates different packet size starting from 128 KB until 1024. For 256 KB and 512 KB the values are stable and almost equal during the testing period. On the other hand, we can see that, when the packet size is 1024 KB there is a great increase in throughput. Finally for 128 KB we have the lowest values in throughput, indicated with the blue line. For all the measurements the values do not show any significant peak.

In figure 6, throughput of the connection Ubuntu1-Ubuntu2 is displayed. We measure the throughput in Mbps for 31 seconds time period. Different colored line indicates different packet size starting from 128 KB until 1024. For 128KB and 256KB packets, we see less throughput than the Controller-Ubuntu1 connection displayed in figure 5. For 512 KB packet size, there is a significant increase as well as in 1024 that we have the maximum values, slightly increased in contrast with the Controller - Ubuntu1 connection. During the testing period, measurements for all packet sizes are stable without peaks.

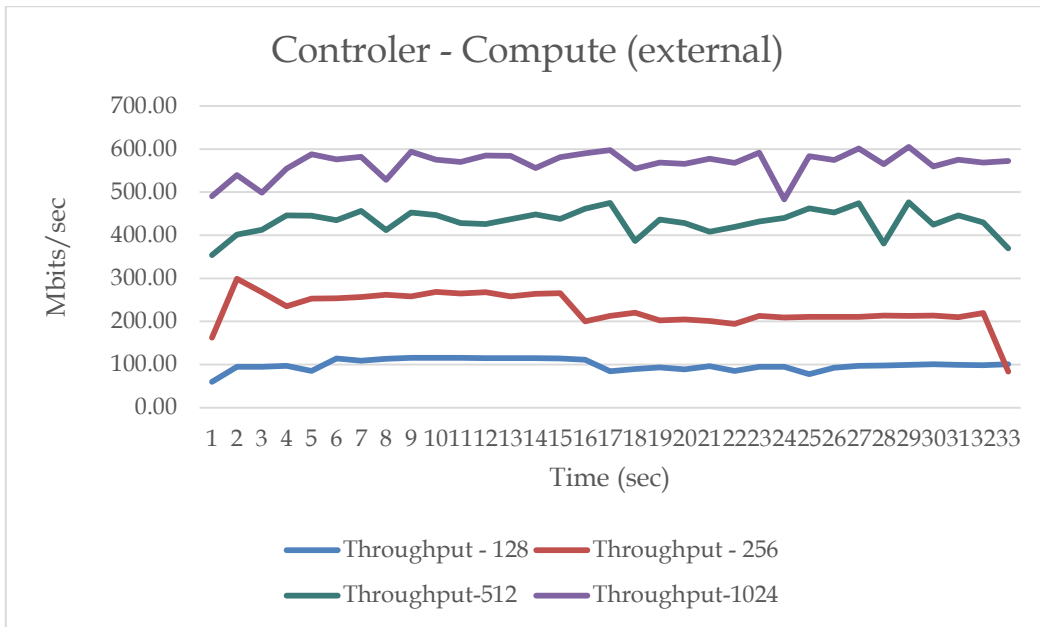


FIGURE 8 THROUGHPUT IN CONNECTION BETWEEN CONTROLLER AND COMPUTE NODE, THROUGH EXTERNAL NETWORK 192.168.30.0/24 THAT WAS ALREADY HOSTED IN THE VIRTUALIZATION

Comparing the throughput results between 2 nodes hosted in an already existed network inside virtualization, we can see great difference in bandwidth values with our measurements in networks hosted in OpenStack. This could be a result of poor optimization of network properties inside the stack and is a great opportunity for further studies.

CHAPTER 6

CONCLUSION – DISCUSSION

6.1 DISCUSSION

6.1.1 PYTHON FOR AUTOMATIONS

Trying to answer the main questions that concern this research, scripting and implementing automations based on those scripts are a good practice in Cloud, both Private and Public. Large

Cloud Providers like Amazon or Microsoft suggest the use of scripts and Command line environments to better, faster, and more efficient manage the cloud resources.

This conclusion also applies in Private Cloud implementations like OpenStack. From the results of this research, we can see that Python, as a universal programming and scripting language is maybe the best choice at that time, as it has an easy syntax, that helps others to better understand, adjust and re implement the scripts regardless of the initial environment.

The popularity that Python has gained, combined with the large and active community behind the project gives almost endless possibilities, including ready to go libraries event scripts and community support in every step of the implementation. Python ability to interact with JSON format files also make it a good choice.

As (MIHĂILĂ, BĂLAN, CURPEN, & SANDU, 2017) wrote in “**Network Automation and Abstraction using Python Programming Methods**”, Python is the best choice for new and experienced programmers to implement Network Automations and Abstractions, as it reduce time for equipment configuration and offer easier and more efficient maintenance. In the same article there is a great reference in the potential security benefits that automation implementation through scripting can offer.

In an older article from (Kumar, Rakesh, Charu, Jain, & Kumar Jangir, 2014) , we can find the same conclusions about Python Scripts, specialized in OpenStack management, giving a great example on how easier and more efficient is to deploy complex Openstack architecture, using automations and scripts.

6.1.2 OPENSTACK API INTERACTION

Openstack has many ways to interact with, including a web GUI, CLI and of course gives RESTFUL API compatibility for programmatic access and management. In this research we utilize API calls to run Python Scripts and manage the deployed environment. This is an efficient and fast way to massively deploy, manage and do maintenance work, from any host inside the virtualization. OpenStack mainly interacts with JSON configuration files. Python, using JSON libraries is a good choice to achieve API calls and communication with the stack. As API calls is the most common way in interacting with OpenStack so far, there is community support and technical knowledge sharing that facilitate the process.

6.1.3 COMPARISON OF NETWORKS CREATED IN OPENSTACK WITH THE PREEXISTING IN THE VIRTUALIZATION.

Comparing the measurements for network performance inside and outside the OpenStack deployment we can see that the preexisting networks inside the Virtualization environment, which was the base of our project, seem to have better throughput, returning results with stational significant different from the networks that were created inside the OpenStack.

Even if Latency in both tests was at the same level and there was zero packet loss, we can ignore the different values in Bandwidth and in throughput, as displayed in the given Figures. Many

variables may have led to that results, including potential lack in optimization in the deployed networks and network devices such as routers, virtual switches, and virtual machines. Despite the difference in values, the deployed networks inside the OpenStack environment continues to be considered as functional and stable and that is what gives the opportunity for further research regarding the configuration and the optimization of these networks.

As mentioned by (Saghir, 2019), latency and bandwidth issues inside OpenStack, are well known issues, both in Layer 2 and Layer 3 perspective, especially if classic Neutron Networking component and routing methods are used. In contrary, researcher purpose the use of Neutron DVR and OpenDayLight as a better alternative in OpenStack deployments.

6.2 REFLECTIONS

In this research we manage to demonstrate with success that using automations in a new or a pre-existing virtualized environment, built with open stack is an easy and pretty efficient way to interact, manage and support your deployments. While the architecture of the deployment grows and becomes more and more complex, having everything under control can be very difficult and time consuming if there are no automations set up.

Python with the proper libraries installed proved to be one of the best solutions that can help automate your environment. As OpenStack uses JSON config files as core configuration option, Python 3 that is used in this research, gives the wanted results in a fast, managed and at the same time very understandable way. Even if Python, is not widely used with openstack, developers and architectures prefers the interaction using the OpenStack CLI, it gives endless possibilities because gives the possibility to interconnect the stack with other applications and event to develop full management applications based on the need of each customer individually.

This research also demonstrates and proves that the use of RESTFUL APIs end points of OpenStack remains the most suitable way to communicate with the stack. Confirming our initial hypothesis, the web interface (Horizon) is easier but has some disadvantages especially if we are deploying complex compute and network architecture. API calls gives developers and architectures the ability to remotely access and support stack operations without the need to access the web interface. This is really important as gives the power to other devices, networks or individuals to communicate with the stack.

As for the networks created in Openstack, it is proved that they depend on the underlying hardware as well as the configurations inside the stack. Neutron Network component of Openstack, demands years of practical experience and good and thorough planning in order to create and manage your networks efficiently. In this research we follow a basic configuration, keeping a lot of settings and parameters in their default values, this is probably the reason that we were not able to fully and completely achieve to demonstrate all the abilities of OpenStack Networking. A great example for that is that we were not able to fully test the created network (in this case 172.16.0.0/24) as an external network inside the virtualization.

6.2.1 LIMITATIONS

- The resources of the virtualization. To better test Openstack, we may need resources (compute power, network, memory) that was difficult to get from the preexisting cyber range environment
- Known version mismatches between the OpenStack image that is used and the Virtualization environment that we host our stack, definitely hardened our work
- Limitations in the configurations of the image used to deploy OpenStack may lead to limited results, especially when deploying networks and network components.

6.3 FUTURE WORKS

Starting from the topics discussed and deployed in this research, there are many possibilities for further research.

- Development of a full management application written in Python or Java that utilizes the API endpoints of OpenStack, giving a full experience in the user. This maybe be tested in comparisons with the classical web GUI management purposed by OpenStack.
- Research and better optimization of created networks to achieve better performance
- Deployment of the OpenStack in Public Cloud Infrastructure, trying to interconnect with virtual networking components like an AWS VPC or a VNET in Azure
- Test scripts in a massive, Production – like deployment to check how efficient the scripts are in larger and more complex needs.

BIBLIOGRAPHY

Costa, G., Russo, E., & Armando, A. (2015). Automating the Generation of Cyber Range Virtual. *DIBRIS*.

ENISA. (2017). *Priorities for EU research - Analysis of the ECSO Strategic Research*. Brussels: ENISA.

European Cyber Security Organization (ECS). (2020, Μάρτιος). Understanding Cyber Ranges: From Hype to Reality. σ. 31.

- Gabriele Costa, E. R. (2019). *Automating the Generation of Cyber Range Virtual*. Università degli Studi di Genova.
- Hochstein, L. (2013). IBM. Ανάκτηση από Python APIs: The best-kept secret of OpenStack: <https://developer.ibm.com/depmodels/cloud/articles/cl-openstack-pythonapis/>
- Jaison Paul Mulerikkal, P. Y. (2020). *A Comparative Study of OpenStack and CloudStack*.
- Karjalainen, M., & Siponen, M. (2011, August). Toward a New Meta-Theory for Designing Information Systems (IS) Security Training Approaches. *Journal for Associations for Information Systems*.
- Kaspersky LAB. (2019). *Kaspersky*. Ανάκτηση από What is Cyber Security?: <https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security>
- Kumar, R., Rakesh, N., Charu, S., Jain, K., & Kumar Jangir, S. (2014). Open Source Solution for Cloud. *International Journal of Computer Science and Mobile Computing*.
- Lima, S., Rocha, A., & Licinio, R. (2019, May). An overview of OpenStack architecture: a message queuing services node. *Cluster Computing : The Journal of Networks, Software Tools and Applications*.
- Litvinski, O., & Abdelouahed, G. (2013). Openstack scheduler evaluation using design of experiment approach. *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*. IEEE.
- McAfee. (2013). *THE ECONOMIC IMPACT OF CYBERCRIME AND CYBER ESPIONAGE*. McAfee - Center for Strategic and Studies.
- MIHĂILĂ, P., BĂLAN, T., CURPEN, R., & SANDU, F. (2017). Network Automation and Abstraction using Python.
- MIT Lincoln Laboratory Lexington United States. (2016). Advanced Tools for Cyber Ranges. *DEFENSE TECHNICAL INFORMATION CENTER*.
- NIST - Cyber Range Project Team. (2020). The Cyber Range: A Guide. σ. 17.
- OpenStack. (2018). *OpenStack*. Ανάκτηση από OpenStack API Documentation: <https://docs.openstack.org/api-quick-start/index.html>
- OpenStack Docs. (2019). *OpenStack: OpenStack Networking*. Ανάκτηση από docs.openstack.org: <https://docs.openstack.org/ocata/networking-guide/intro-os-networking.html>
- Peter Mell, T. G. (2013). *The NIST Definition of Cloud*.
- Rosenstein, M., & Corvese, F. (2018). Secure Architecture for the Range-Level Command and Control System.
- Saghir, A. (2019). Performance Evaluation of OpenStack Networking Technologies.
- SEFRAOUI, O., AISSAOUI, M., & ELEULDJ, M. (2012). OpenStack: Toward an Open-Source Solution for Cloud Computing. *International Journal of Computer Applications*.
- Tantardini, M., Leva, F., Tajoli, L., & Piccardi, C. (2015). Comparing methods for comparing Networks. *Scientific Reports*.
- Van, V., Le Minh Chi, Nguyen Quoc Long, & Dac-Nhuong L. (2015). *A Performance Analysis of OpenStack Open-source*.

- Wang, L., & Zhang, D. (2017). Research on OpenStack of open source cloud computing in colleges and universities' computer room. *3rd International Conference on Advances in Energy, Environment and Chemical Engineering*. IOP Publishing.
- Zhihong, T., Yu, C., Shen, S., Xiaoxia, Y., Lihua, Y., & Xiang, C. (2018). *A Real-Time Correlation of Host-Level Events in Cyber Range Service for Smart Campus*. IEEE.