

**UNIVERSITY OF PATRAS**

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**FIELD OF SPECIALISATION: COMMUNICATIONS**

**THESIS**

of the Electrical and Computer Engineering student  
of University of Patras

Grivas Efthymios Konstantinos

ID: 1047014

Subject

**«Establishing L2 connectivity between multiple  
isolated Openstack-based clouds»**

Supervisor

Denazis Spyridon

**Thesis ID:**

Patras, June 2020

# CONTENTS

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Cloud Computing.....</b>	<b>3</b>
<b>2.1 Definition.....</b>	<b>3</b>
<b>2.2 The benefits of Cloud Computing.....</b>	<b>3</b>
<b>2.3 Service Models.....</b>	<b>5</b>
2.3.1 Infrastructure as a Service (IaaS) .....	5
2.3.2 Platform as a Service (PaaS).....	7
2.3.3 Service as a Service (SaaS).....	8
2.3.4 Serverless.....	8
<b>2.4 Deployments Models .....</b>	<b>9</b>
2.4.1 Private cloud .....	9
2.4.2 Public cloud.....	9
2.4.3 Hybrid cloud .....	10
<b>3. Tunneling Protocols.....</b>	<b>11</b>
<b>3.1 Definition.....</b>	<b>11</b>
<b>3.2 Common Tunneling Protocols .....</b>	<b>11</b>
3.2.1 Generic Routing Encapsulation (GRE) .....	11
3.2.2 Virtual Extensible LAN (VXLAN).....	12
3.2.3 Geneve .....	13
3.2.4 Internet Protocol Security (IPsec).....	14
<b>3.3 Tunneling Example .....</b>	<b>16</b>
<b>4. Virtual Private Network (VPN).....</b>	<b>19</b>
<b>4.1 Definition.....</b>	<b>19</b>
<b>4.2 VPN Protocols.....</b>	<b>20</b>
4.2.1 OpenVPN.....	20
4.2.2 L2TP/IPsec .....	21
4.2.3 Point-to-Point Tunneling Protocol (PPTP) .....	21
4.2.4 Secure Socket Tunneling Protocol (SSTP) .....	22
4.2.5 Internet Key Exchange version 2 (IKEv2) .....	23
<b>4.3 OpenVPN Installation.....</b>	<b>24</b>
<b>5. Open vSwitch.....</b>	<b>28</b>
<b>5.1 The role of Open vSwitch .....</b>	<b>28</b>
<b>5.2 Features .....</b>	<b>29</b>
<b>5.3 Commands .....</b>	<b>30</b>

5.3.1	Ovs-vsctl .....	30
5.3.1.1	‘Show’ commands.....	30
5.3.1.2	Switch configuration commands .....	30
5.3.2	Ovs-ofctl.....	31
5.3.2.1	‘Show’ commands.....	31
5.3.2.2	Switch configuration commands .....	31
5.3.3	Ovs-dpctl .....	32
5.3.4	Ovs-appctl .....	32
5.4	Installation .....	32
6.	Openstack .....	33
6.1	What is Openstack? .....	33
6.2	Openstack Components .....	34
6.2.1	Compute .....	35
6.2.2	Hardware Lifecycle .....	36
6.2.3	Storage.....	36
6.2.4	Networking.....	36
6.2.5	Shared services.....	37
6.2.6	Orchestration .....	37
6.2.7	Workload Provisioning .....	38
6.2.8	Application Lifecycle.....	39
6.2.9	API Proxies.....	39
6.2.10	Web Frontend .....	39
6.3	Commands .....	41
6.3.1	Project management.....	41
6.3.2	Image management .....	41
6.3.3	Volume management.....	42
6.3.4	Server management.....	43
6.3.5	Network and subnet management.....	43
6.3.6	Router management .....	44
6.3.7	Security groups management .....	45
6.3.8	Volume management.....	46
6.3.9	Keypair management .....	46
6.4	Installation .....	47
6.5	Example.....	48
7.	Packer.....	51
7.1	Introduction .....	51
7.2	Advantages of using Packer .....	51

<b>7.3 Packer terminology .....</b>	<b>52</b>
7.3.1 Artifacts.....	52
7.3.2 Builds and builders.....	52
7.3.3 Commands .....	54
7.3.4 Post-processors .....	55
7.3.5 Provisioners.....	55
<b>7.4 Building a new image .....</b>	<b>55</b>
<b>8. The project.....</b>	<b>56</b>
8.1 Work analysis .....	56
8.2 Future Work .....	72
<b>FIGURE LIST .....</b>	<b>73</b>
<b>REFERENCES.....</b>	<b>74</b>

# ABSTRACT

Network functions virtualization (NFV) is a way to virtualize network services, such as routers, firewalls, and load balancers, that have traditionally been run on proprietary hardware. These services are packaged as virtual machines (VMs) on commodity hardware, which allows service providers to run their network on standard servers instead of proprietary ones. With NFV, there is no need to have dedicated hardware for each network function. NFV improves scalability and agility by allowing service providers to deliver new network services and applications on demand, without requiring additional hardware resources. The NFV architecture consists of:

Virtualized network functions (VNFs) are software applications that deliver network functions such as file sharing, directory services, and IP configuration.

Network functions virtualization infrastructure (NFVi) consists of the infrastructure components—compute, storage, networking—on a platform to support software, such as a hypervisor like KVM or a container management platform, needed to run network apps.

Management, automation and network orchestration (MANO) provides the framework for managing NFV infrastructure and provisioning new VNFs.

The purpose of the thesis is to provide layer 2 connectivity between two or more isolated cloud environments using pure software virtual switches. So, in fact a VNF has been developed. Switches are based on Open vSwitch, a proven solution for implementing software-based data planes. Goal is to package OVS in a way that can implement the gateway role for multiple data paths that may serve different purposes in each cloud.

The cloud provider used is Openstack. As each cloud topology is hidden behind a NAT topology with just one external IP exposed, an overlay VPN connection is needed. All the code files have been uploaded to Github in a package named Cloud-Connectivity (<https://github.com/thimiosgr/CloudConnectivity.git>). Two use-cases have been taken into consideration. The first one is the scenario of a user that wants to manage multiple openstack environments from a remote host. In that case the user must clone the “master” branch of this package from Github. The second case is the scenario of a user who works directly on a server that runs an Openstack environment and wants to configure only that single server. The “simple” branch covers this case.

To achieve connectivity between the clouds using the virtual IP’s of the clouds, tunnels must be set up. The tunneling protocol that is used is VXLAN (Virtual Extensible LAN) and it allows private network communications to be sent across a public network (such as the Internet) through a process called encapsulation. The switches and tunnels must be set up automatically with no human intervention. For this to be done, a program named Packer is used and it’s purpose it to create custom images. It

uses a base image and executes the necessary configuration on it so that it runs the services we need. For example, in this thesis the base image that was used is Ubuntu Server 16.04 cloud version. An example service that was added on this image is that when an instance that runs the image is booted for the first time, must fetch the VPN configuration files, connect to the VPN server and set up the switches and tunnels that allow communication with the rest of the clouds. A second custom image was also created, a simple HTTP server that allows us to show that HTTP traffic can be sent across the tunnels.

The infrastructure used is consisted of two servers, each one running Openstack. Openstack was installed with Devstack which is a series of extensible scripts used to quickly bring up a complete OpenStack environment based on the latest versions of everything from git master. The VPN server is running on a separate host, independent of the machines that are running the clouds and it uses the OpenVPN protocol. After the CloudConnectivity package is installed and executed on each server, a complete network topology consisted of five isolated from the outside world networks and one network able to reach the Internet is created. An instance attached to all networks is booted. This instance is performing the role of a gateway for the isolated networks, so that any instance belonging to them can communicate with the other cloud. Of course, this applies to the 2<sup>nd</sup> cloud as well. The switches and tunnels are brought up automatically. The tunnels use the VPN IP's as their endpoints, so they are set up between the two "gateways" of each cloud's internal networks. Finally if we try to ping an instance of the 2<sup>nd</sup> cloud from an instance running in the 1<sup>st</sup> cloud, we notice that the ICMP packets are delivered properly. To be sure that the packets reach their right destination we can use the "tcpdump" command to capture the packets arriving at the instance of the 2<sup>nd</sup> cloud.

Assuming a path MTU of 1500 bytes the Maximum Segment Size is equal to 1422 as shown in the below table. The VXLAN protocol uses UDP as its underlying transport protocol which inserts a header of 8 bytes. The original ICMP packet has an ICMP header of 8 bytes and an IP header of 20 bytes. The final packet contains an outer IP header of 20 bytes and an outer MAC header of 14 bytes. So the MSS drops to 1422 bytes.

# 1. Introduction

Network Function Virtualization (NFV) is the technology that virtualizes network functions such as router, switches etc. that traditionally run on dedicated hardware into building blocks. With NFV the need for explicit hardware no longer exists as every network function runs on a multi-purpose server. NFV improves scalability and flexibility by allowing service providers to provide new network services and custom applications, without the need for additional hardware resources. The NFV architecture consists of:

- Virtualized Network Functions – VNF which are software applications that provide network functions such as shared file usage and IP configuration.
- NFV Infrastructure – NFVI which consists of infrastructure elements such as storage and computing power. NFVI could be distributed across multiple data centers called NFVI-PoP's.
- MANagement and Network Orchestration (MANO) which manages and orchestrates the NFVI and provides new VNF's.

The purpose of this thesis is to provide layer 2 connectivity between multiple isolated Openstack-based clouds, using only virtual switches and tunnels. Switches are based on Open vSwitch (OVS), a proven solution for implementing software-based data planes. Goal is to package OVS in a way that can implement the gateway role for multiple data paths that may serve different purposes in each cloud. Layer 2 connectivity was chosen instead of layer 3, because layer 2 protocols work with MAC addresses and are faster than layer 3 ones. Moreover the layer 2 networks forward broadcast traffic to all the devices of the network compared to layer 3 networks which sometimes restrict it.

Nowadays, cloud computing has gained much reputation and is used by the majority of organizations. Some of them own their own cloud and some others rent the necessary infrastructure from a cloud provider. Most of the clouds are distributed all around the world in different data centers. There might be a VNF that is running on multiple clouds located in different countries. The VNF's machines must have connectivity so they can collaborate and run the application properly. So, there is the need for the cloud components to have connectivity with one another. The traditional way of achieving that is with dedicated hardware. But, with NFV technology these functions run as virtualized in a server.

After a brief introduction to NFV technology, one could easily assume that this thesis is implementing a VNF. Actually this thesis proposes a way of communication between the NFVi-PoPs by using tunnels. In more specific, we use virtual switches based on Open vSwitch. On every switch we set-up VXLAN tunnels whose endpoints

are Virtual Machines running on each cloud. The limitation of public IP's lead to the use of a VPN server which just makes our life easier by making the two clouds visible to one another. Having had some more public IP's in our disposal, then there would be no need for a VPN server.

In general, the object of this thesis is pretty useful especially as cloud computing is becoming more and more popular. A large number of companies is moving to cloud as it saves money and there is no need for maintenance by their side. So the connectivity between the components of each cloud is a very significant topic as far as speed, security and availability are concerned.



## 2. Cloud Computing

### 2.1 Definition

Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the client. The term is generally used to describe data centers available to many clients over the Internet. Large clouds, often have functions distributed over multiple locations from central servers. Cloud computing relies on sharing of resources to achieve coherence and economies of scale.

### 2.2 The benefits of Cloud Computing

Cloud computing has been around for approximately two decades and despite the data pointing to the business efficiencies, cost-benefits, and competitive advantages it holds, a large portion of the business community continues to operate without it. According to a study by the International Data Group, 69% of businesses are already using cloud technology in one capacity or another, and 18% say they plan to implement cloud-computing solutions at some point. At the same time, Dell reports that companies that invest in big data, cloud, mobility, and security enjoy up to 53% faster revenue growth than their competitors. As this data shows, an increasing number of tech-savvy businesses and industry leaders are recognizing the many benefits of the cloud-computing trend. But more than that, they are using this technology to more efficiently run their organizations, better serve their customers, and dramatically increase their overall profit margins. Here are some common reasons organizations are turning to cloud computing services. [14], [21]

Cost reduction: By using cloud infrastructure, businesses don't have to spend huge amounts on purchasing and maintain equipment. This drastically reduces capex costs. They don't have to build out a large data center to grow. Cloud also reduces costs related to downtime. Since downtime is rare in cloud systems, this means that no money will be spent on fixing potential issues related to downtime.

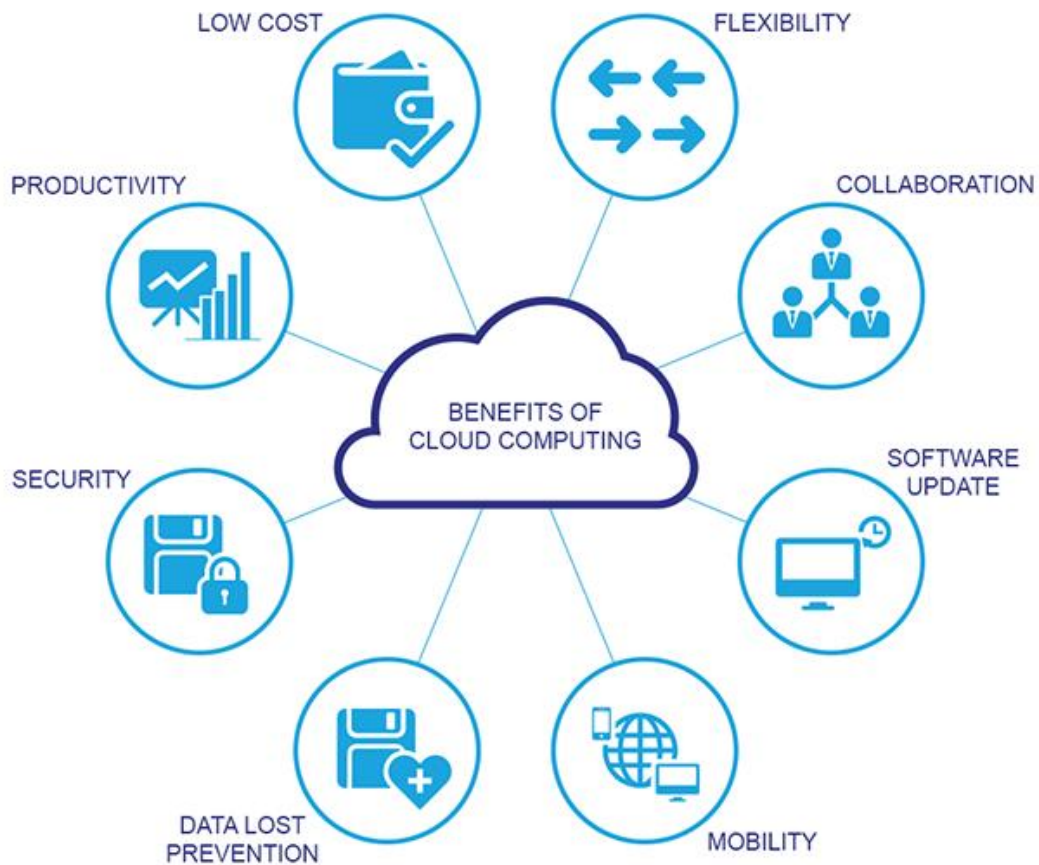
Data security: One of the major concerns of every business, regardless of size and industry, is the security of its data. Data breaches and other cybercrimes can devastate a company's revenue, customer loyalty and brand positioning. Cloud offers many advanced security features that guarantee that data is securely stored and handled. Cloud storage providers implement baseline protections for their platforms and the data they process, such authentication, access control, and encryption. From there, most enterprises supplement these protections with added security measures of their own to bolster cloud data protection and tighten access to sensitive information in the cloud.

Scalability: Different companies have different IT needs -- a large enterprise of 1000+ employees won't have the same IT requirements as a start-up. Using cloud is a great solution because it enables enterprise to efficiently -- and quickly -- scale up/down their IT departments, according to business demands. Cloud based solutions are ideal for businesses with growing or fluctuating bandwidth demands. If a business demands increase, they can easily increase their cloud capacity without having to invest in physical infrastructure. This level of agility can give businesses using cloud computing a real advantage over competitors. This scalability minimizes the risks associated with in-house operational issues and maintenance. The users have high-performance resources at their disposal with professional solutions and zero up-front investment.

Mobility: Cloud computing allows mobile access to corporate data via smartphones and devices, which is a great way to ensure that no one is ever left out of the loop. Staff with busy schedules, or who live a long way away from the corporate office, can use this feature to keep instantly up-to-date with clients and coworkers. Resources in the cloud can be easily stored, retrieved, recovered, or processed with just a couple of clicks. Users can get access to their works on-the-go, 24/7, via any devices of their choice, in any corner of the world as long as they stay connected to the internet. On top of that, all the upgrades and updates are done automatically, off-sight by the service providers. This saves time and team effort in maintaining the systems, tremendously reducing the IT team workloads.

Disaster recovery: Data loss is a major concern for all organizations, along with data security. Storing data in the cloud guarantees that data is always available, even if equipment like laptops or PCs, is damaged. Cloud-based services provide quick data recovery for all kinds of emergency scenarios -- from natural disasters to power outages. Cloud infrastructure can also help with loss prevention. If someone relies on traditional on-premises approach, all his data will be stored locally, on office computers. Despite their best efforts, computers can malfunction from various reasons -- from malware and viruses, to age-related hardware deterioration, to simple user error. But, if they upload their data to the cloud, it remains accessible for any computer with an internet connection, even if something happens to the work computer.

Control: Having control over sensitive data is vital to any company. You never know what can happen if a document gets into the wrong hands, even if it's just the hands of an untrained employee. Cloud enables complete visibility and control over data. It can be easily configured which users have what level of access to what data. This gives the Cloud's owner control, but it also streamlines work since staff will easily know what documents are assigned to them. It will also increase and ease collaboration. Since one version of the document can be worked on by different people, and there's no need to have copies of the same document in circulation.



*Figure 1. The benefits of Cloud Computing*

## **2.3 Service Models**

Cloud computing providers offer their "services" according to different models. Most cloud computing services fall into four broad categories: infrastructure as a service (IaaS), platform as a service (PaaS), serverless and software as a service (SaaS). These are sometimes called the cloud computing stack because they build on top of one another. In the next subchapters we will take a more detailed look into the Cloud's service models. For more information please refer to [8], [16].

### **2.3.1 Infrastructure as a Service (IaaS)**

Infrastructure as a service (IaaS) is an instant computing infrastructure, provisioned and managed over the internet. IaaS quickly scales up and down with demand, letting the clients pay only for what they use. It helps on avoiding the expense and complexity of buying and managing your own physical servers and other datacenter infrastructure. Each resource is offered as a separate service component, and the clients only need to rent a particular one for as long as they need it. A cloud computing service

provider manages the infrastructure, while the clients purchase, install, configure, and manage their own software—operating systems, middleware, and applications.

Typical things businesses do with IaaS include:

Test and development: Teams can quickly set up and dismantle test and development environments, bringing new applications to market faster. IaaS makes it quick and economical to scale up dev-test environments up and down.

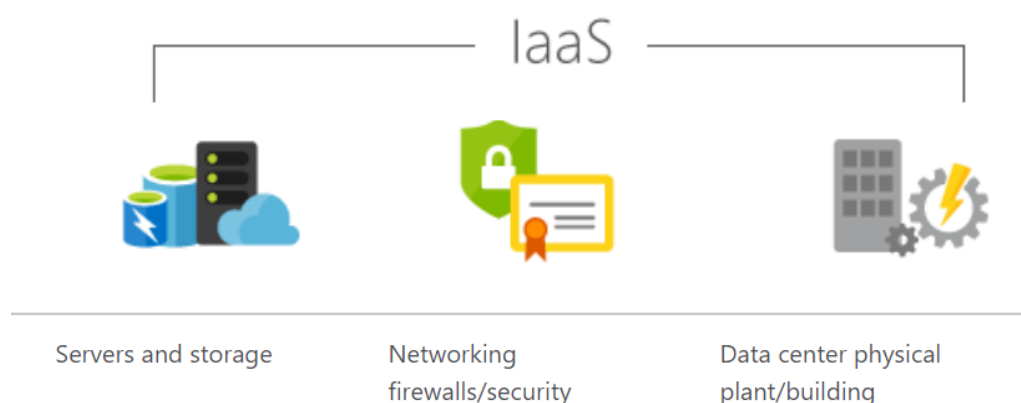
Website hosting: Running websites using IaaS can be less expensive than traditional web hosting.

Storage, backup and recovery: Organizations avoid the capital outlay for storage and complexity of storage management, which typically requires a skilled staff to manage data and meet legal and compliance requirements. IaaS is useful for handling unpredictable demand and steadily growing storage needs. It can also simplify planning and management of backup and recovery systems.

Web apps: IaaS provides all the infrastructure to support web apps, including storage, web and application servers and networking resources. Organizations can quickly deploy web apps on IaaS and easily scale infrastructure up and down when demand for the apps is unpredictable.

High-performance computing: High-performance computing (HPC) on supercomputers, computer grids or computer clusters helps solve complex problems involving millions of variables or calculations. Examples include earthquake and protein folding simulations, climate and weather predictions, financial modeling and evaluating product designs.

Big data analysis: Big data is a popular term for massive data sets that contain potentially valuable patterns, trends and associations. Mining data sets to locate or tease out these hidden patterns requires a huge amount of processing power, which IaaS economically provides.



*Figure 2. Infrastructure as a Service*

### 2.3.2 Platform as a Service (PaaS)

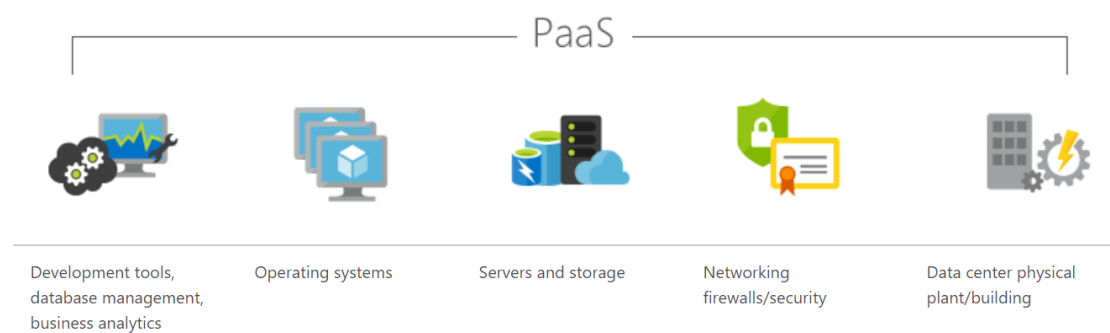
Platform as a service (PaaS) is a complete development and deployment environment in the cloud, with resources that enable delivering everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications. The clients purchase the resources they need from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection. Like IaaS, PaaS includes infrastructure—servers, storage and networking—but also middleware, development tools, business intelligence (BI) services, database management systems and more. PaaS is designed to support the complete web application lifecycle: building, testing, deploying, managing and updating. It allows to avoid the expense and complexity of buying and managing software licenses, the underlying application infrastructure and middleware, container orchestrators such as Kubernetes or the development tools and other resources. The users manage the applications and services they develop and the cloud service provider typically manages everything else.

Organizations typically use PaaS for these scenarios:

Development framework: PaaS provides a framework that developers can build upon to develop or customize cloud-based applications. Similar to the way that an Excel macro is created, PaaS lets developers create applications using built-in software components. Cloud features such as scalability, high-availability and multi-tenant capability are included, reducing the amount of coding that developers must do.

Analytics or business intelligence: Tools provided as a service with PaaS allow organizations to analyze and mine their data, finding insights and patterns and predicting outcomes to improve forecasting, product design decisions, investment returns and other business decisions.

Additional services: PaaS providers may offer other services that enhance applications, such as workflow, directory, security and scheduling.

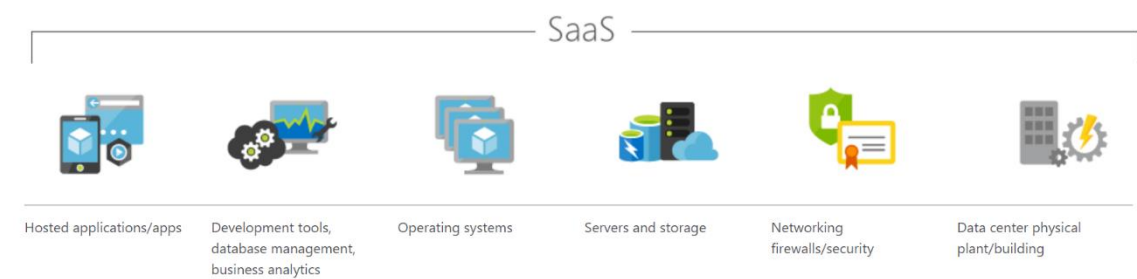


*Figure 3. Platform as a Service*

### 2.3.3 Service as a Service (SaaS)

Software as a service (SaaS) allows users to connect to and use cloud-based apps over the Internet. Common examples are email, calendaring and office tools. SaaS provides a complete software solution which the clients purchase on a pay-as-you-go basis from a cloud service provider. They rent the use of an app for their organization and their users connect to it over the Internet, usually with a web browser. All of the underlying infrastructure, middleware, app software and app data are located in the service provider's data center. The service provider manages the hardware and software and with the appropriate service agreement, will ensure the availability and the security of the app and data as well. SaaS allows an organization to get quickly up and running with an app at minimal upfront cost.

Common SaaS scenarios: Everyone who has used a web-based email service such as Outlook, Hotmail or Yahoo! Mail, has already used a form of SaaS. With these services, the users log into their account over the Internet, often from a web browser. The email software is located on the service provider's network and their messages are stored there as well. The email and stored messages can be accessed from a web browser on any computer or Internet-connected device. The previous examples are free services for personal use. For organizational use, an organization can rent productivity apps, such as email, collaboration and calendaring; and sophisticated business applications such as customer relationship management (CRM), enterprise resource planning (ERP) and document management. They pay for the use of these apps by subscription or according to the level of use.



*Figure 4. Service as a Service*

### 2.3.4 Serverless

Serverless computing enables developers to build applications faster by eliminating the need for them to manage infrastructure. With serverless applications, the cloud service provider automatically provisions, scales and manages the infrastructure required to run the code. In understanding the definition of serverless computing, it is important to note that servers are still running the code. The serverless name comes from the fact that the tasks associated with infrastructure provisioning and management are invisible to the developer. This approach enables developers to increase their focus on the business logic and deliver more value to the core of the business. Serverless computing helps teams increase their productivity and bring products to market

faster and it allows organizations to better optimize resources and stay focused on innovation.

## **2.4 Deployments Models**

### **2.4.1 Private cloud**

A private cloud refers to a cloud deployment model operated exclusively for a single organization, whether it is physically located at the company's on-site data center, or is managed and hosted by a third-party provider. In a private cloud, resources are not shared with other organizations, but this also means that the company using it is entirely responsible for its management, maintenance, and regular updates – which can also get significantly more expensive than public ones. Because this cloud deployment model is only accessible by a single company, there are less security concerns as all data is protected behind a firewall.

#### Advantages:

- ✓ More possibilities for customization of the cloud environment.
- ✓ Higher security and privacy as resources are not shared with others.
- ✓ Enhanced reliability and greater control over the server.

#### Disadvantages:

- ⊗ Accessing data from remote locations can be significantly more difficult.
- ⊗ High costs for investing in private cloud infrastructure.
- ⊗ Operating expenses as the company is responsible for maintenance.

### **2.4.2 Public cloud**

A cloud is called a "public cloud" when the services are rendered over a network that is open for public use. Public cloud services may be free. Technically there may be little or no difference between public and private cloud architecture, however, security consideration may be substantially different for services (applications, storage, and other resources) that are made available by a service provider for a public audience and when communication is effected over a non-trusted network. Generally, public cloud service providers like Amazon Web Services (AWS), IBM, Oracle, Microsoft, Google, and Alibaba own and operate the infrastructure at their data center and access is generally via the Internet.

#### Advantages:

- ✓ Reduces time in developing, testing and launching new products.
- ✓ Cost effectiveness – there is no need to invest in expensive infrastructures.

- ✓ “Pay-as-you-go” scalability – the users only pay for what they use.

Disadvantages:

- ⊗ Higher security risks due to vulnerabilities as a result from shared resources.
- ⊗ Network performance can suffer instabilities due to spikes in use.
- ⊗ Lack of customization – public clouds are usually less personalizable.

### **2.4.3 Hybrid cloud**

Hybrid Clouds, as their name suggests, are a combination of private and public cloud deployment models that are bound together to provide the benefits of both infrastructures to the company using them. By using a hybrid cloud, organizations are capable of moving data and applications between private and public clouds depending on their purposes. For example, in case of high-volume, less sensitive data that doesn't require strong security layers, public clouds are the better option as they provide more capacity to their users. However, if the company wants to store and manage sensitive data related to critical business operations, it is usually recommended to use a private cloud as it provides more security.

Advantages:

- ✓ Flexibility and control – the company can choose to allocate resources depending on each specific case.
- ✓ Cost effectiveness – as public clouds provide scalability, the clients only pay for the extra capacity if they need it.
- ✓ Enhanced organizational agility for developing and testing new applications.

Disadvantages:

- ⊗ Requires more maintenance, which can result in higher operating expenses for the company.
- ⊗ Initial costs for activating both infrastructures can be really high for many organizations.
- ⊗ Data and application integration can be challenging when building a hybrid cloud.

For more information please refer to [5].



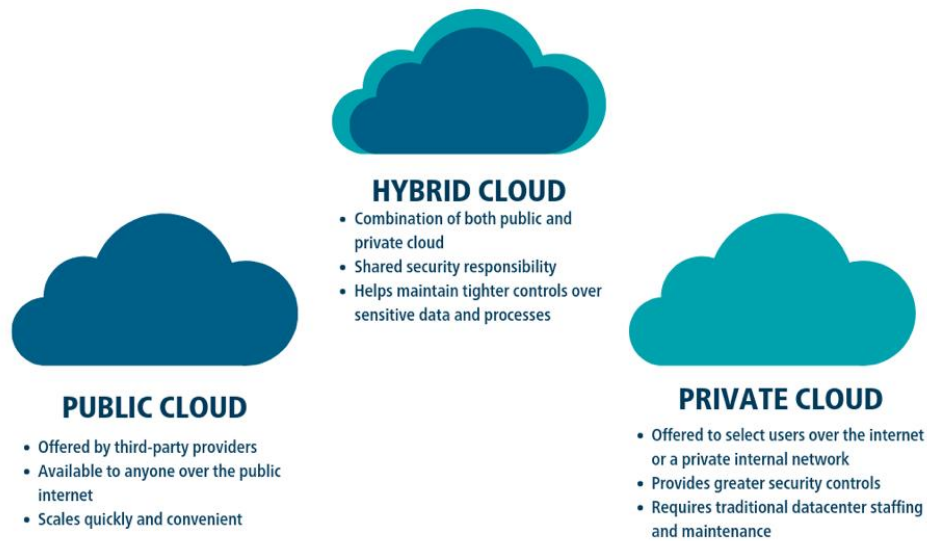


Figure 5. Deployment models. Source: <https://medium.com/>

## 3. Tunneling Protocols

### 3.1 Definition

A tunneling protocol is a communications protocol that allows the movement of data from one network to another. It involves allowing private network communications to be sent across a public network (such as the Internet) through a process called encapsulation. The tunneling protocol works by using the data portion of a packet (the payload) to carry the packets that actually provide the service. Tunneling uses a layered protocol model such as those of the OSI or TCP/IP protocol suite, but usually violates the layering when using the payload to carry a service not normally provided by the network. Typically, the delivery protocol operates at an equal or higher level in the layered model than the payload protocol.

In this chapter the most common tunneling protocols are explained. Furthermore, we will take a quick look at IPsec which is used to provide security over tunneling protocols and present an example of setting up a VXLAN tunnel using Open vSwitch which is described in more detail in chapter 5. For more information on every protocol presented please refer to [18].

### 3.2 Common Tunneling Protocols

#### 3.2.1 Generic Routing Encapsulation (GRE)

Generic Routing Encapsulation (GRE) is a tunneling protocol developed by Cisco Systems that can encapsulate a wide variety of network layer protocols inside virtual

point-to-point links or point-to-multipoint links over an Internet Protocol network. Unfortunately GRE does not encrypt the traffic providing no security. To be the tunnels more secure, GRE packet must be encapsulated into security protocols like IPsec.

Bits 0–3			4–12	13–15	16–31
C	K	S	Reserved0	Version	Protocol Type
Checksum (optional)					Reserved1 (optional)
Key (optional)					
Sequence Number (optional)					

Figure 6. GRE packet header structure

- C: Checksum bit. Set to 1 if a checksum is present.
- K: Key bit. Set to 1 if a key is present.
- S: Sequence number bit. Set to 1 if a sequence number is present.
- Reserved0: Reserved bits; set to 0.
- Version: GRE Version number; set to 0.
- Protocol Type: Indicates the ether protocol type of the encapsulated payload. (For IPv4, this would be hex 0800.)
- Checksum: Present if the C bit is set; contains the checksum for the GRE header and payload.
- Reserved1: Present if the C bit is set; is set to 0.
- Key: Present if the K bit is set; contains an application-specific key value.
- Sequence Number: Present if the S bit is set; contains a sequence number for the GRE packet. [19]

### 3.2.2 Virtual Extensible LAN (VXLAN)

Virtual Extensible LAN (VXLAN) is a network virtualization technology that attempts to address the scalability problems associated with large cloud computing deployments. It uses a VLAN-like encapsulation technique to encapsulate OSI layer 2 Ethernet frames within layer 4 UDP datagrams, using 4789 as the default IANA-assigned destination UDP port number. VXLAN endpoints, which terminate VXLAN tunnels and may be either virtual or physical switch ports, are known as VXLAN tunnel endpoints (VTEPs). VXLAN is an evolution of efforts to standardize on an overlay encapsulation protocol. It increases scalability up to 16 million logical networks and allows for layer 2 adjacency across IP networks. Multicast or unicast with head-end replication (HER) is used to flood broadcast, unknown unicast, and multicast (BUM) traffic. The VXLAN specification was originally created by VMware, Arista Networks and Cisco. Other backers of the VXLAN technology include Huawei, Broadcom, Citrix, Pica8, Big Switch Networks, Cumulus Networks, Dell EMC, Red Hat, Joyent, Juniper Networks, Ericsson, Mellanox, FreeBSD and OpenBSD.

VXLAN adds 50 to 54 bytes of additional header information to the original Ethernet frame. VXLAN uses UDP instead of TCP as the underlying transport protocol. The reason behind this is that good overlay networks must reduce latency and overhead to the minimum necessary. UDP adds no more header overhead than TCP, but less protocol overhead. Furthermore TCP adds overhead, and can create unintentional bandwidth bottlenecks. So, in practice UDP will give higher bandwidth and lower latency using fewer host resources, which is why it is used by VXLAN. [7], [20]

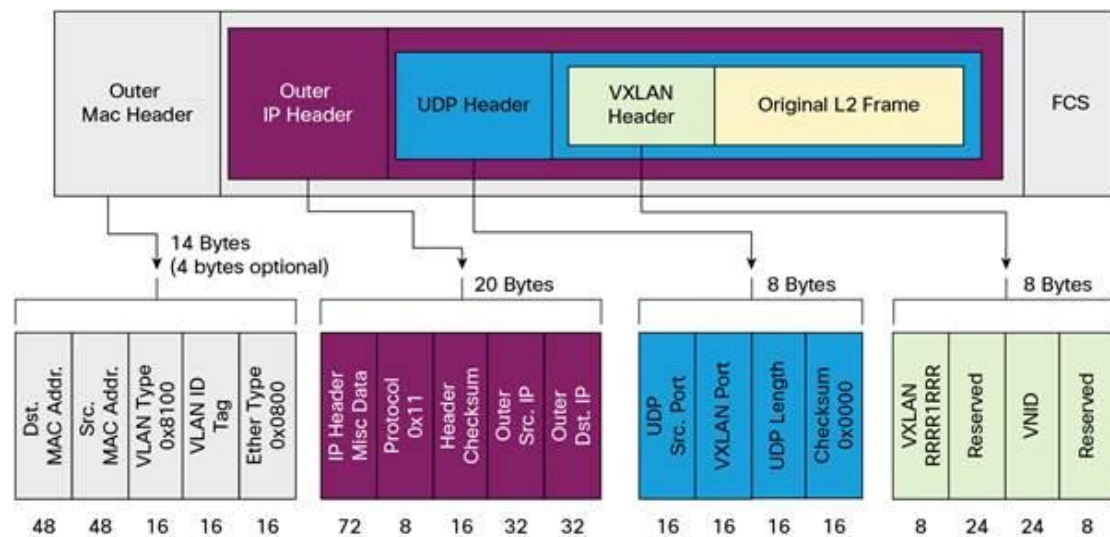


Figure 7. VXLAN packet format

### 3.2.3 Geneve

Generic Network Virtualization Encapsulation (GENEVE) supports all of the capabilities of VXLAN and was designed to overcome their perceived limitations. Many believe GENEVE could eventually replace these earlier formats entirely. The tunnel header looks like:



Figure 8. Geneve packet format

The packet format is very similar to VXLAN's. The main difference is that the GENEVE header is flexible. It's very easy to add new features by extending the header with a new Type-Length-Value (TLV) field.

### 3.2.4 Internet Protocol Security (IPsec)

IPsec is not dedicated to tunneling but it is used widely to provide security on top of the unsecure tunneling protocols. IPsec is a secure network protocol suite that authenticates and encrypts the packet of data to provide secure encrypted communication between two computers over an Internet Protocol (IP) network. It includes protocols for establishing mutual authentication between agents at the beginning of a session and negotiation of cryptographic keys to use during the session. IPsec can protect data flows between a pair of hosts (*host-to-host*), between a pair of security gateways (*network-to-network*), or between a security gateway and a host (*network-to-host*). It supports network-level peer authentication, data-origin authentication, data integrity, data confidentiality (encryption), and replay protection.

IPsec can be used in two modes, the Transport mode and the Tunnel mode. In Transport mode, only the payload of the IP packet is usually encrypted or authenticated. The routing is intact, since the IP header is neither modified nor encrypted; however, when the authentication header is used, the IP addresses cannot be modified by network address translation, as this always invalidates the hash value. The transport and application layers are always secured by a hash, so they cannot be modified in any way, for example by translating the port numbers. In Tunnel mode, the entire IP packet is encrypted and authenticated. It is then encapsulated into a new IP packet with a new IP header. Tunnel mode is used to create virtual private networks for network-to-network communications (e.g. between routers to link sites), host-to-network communications (e.g. remote user access) and host-to-host communications (e.g. private chat).

There is no explicit "Mode" field in IPsec: what distinguishes Transport mode from Tunnel mode is the next header field in the AH header. When the next-header value is *IP*, it means that this packet encapsulates an entire IP datagram (including the independent source and destination IP addresses that allow separate routing after de-encapsulation). This is Tunnel mode. Any other value (TCP, UDP, ICMP, etc.) means that it's Transport mode and is securing an endpoint-to-endpoint connection.

Authentication Header (AH) and Encapsulating Security Payload (ESP) are the two main wire-level protocols used by IPsec, and they authenticate (AH) and encrypt+authenticate (ESP) the data flowing over that connection. AH is used to authenticate (but not encrypt) IP traffic, and this serves the treble purpose of ensuring that we're really talking to who we think we are, detecting alteration of data while in transit, and to guard against replay by attackers who capture data from the wire and attempt to re-inject that data back onto the wire at a later date. ESP is primarily designed to provide encryption, authentication and protection services for the data that is being transferred in an IP network. ESP doesn't protect the packet header; however, in a tunnel mode if the entire packet is encapsulated within another packet as a payload/data packet, it can encrypt the entire packet residing inside another packet. [6]

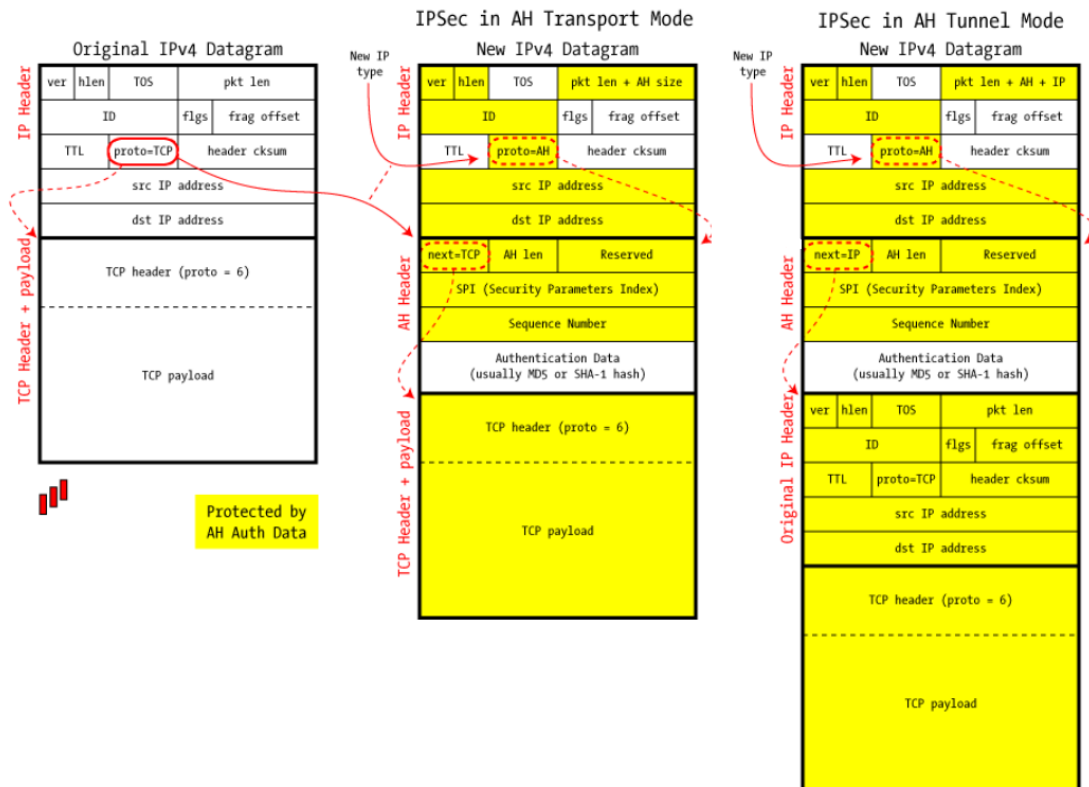


Figure 9. IPsec in AH Transport and Tunnel Mode

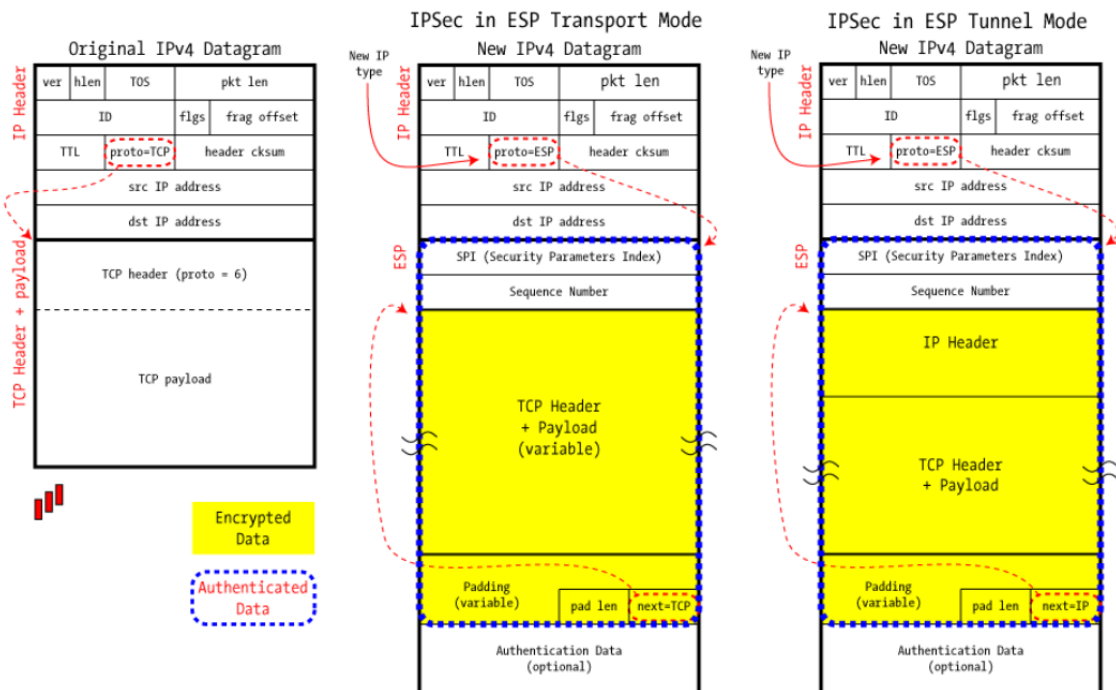
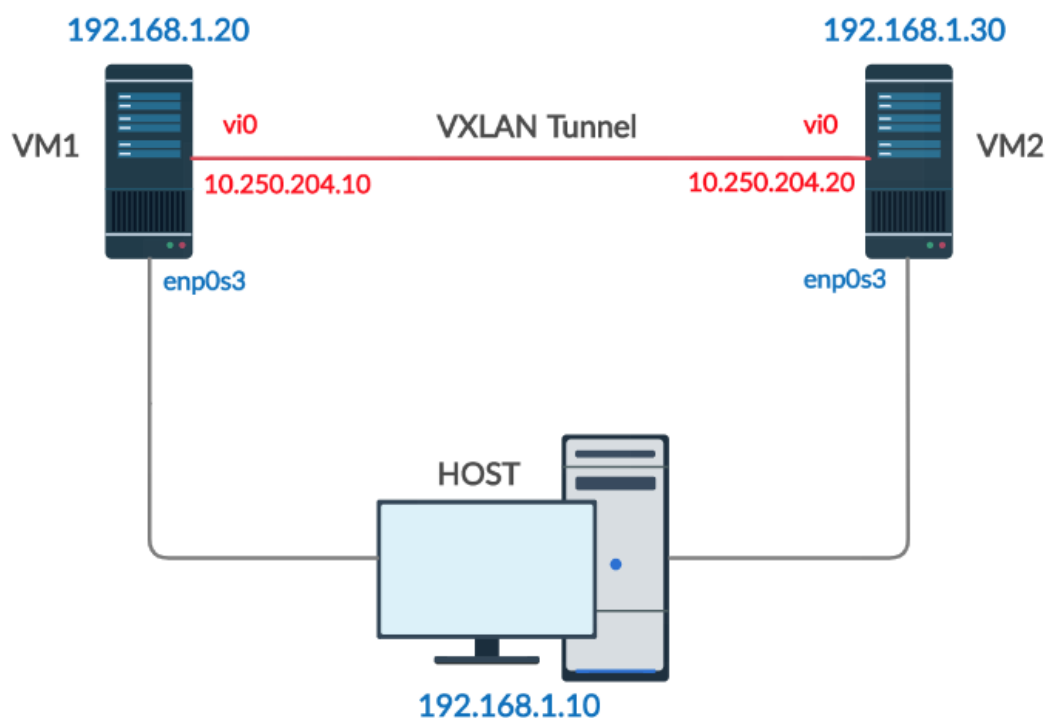


Figure 10. IPsec in ESP Transport and Tunnel Mode

### 3.3 Tunneling Example

To make tunneling protocols understandable, here is an example. Our setup is composed of two Virtual Machines (VM's) which were created with Oracle VM Virtual-box. Let's call them VM1 and VM2. The objective is to create a tunnel so that VM1 can ping VM2 using Open vSwitch. Our Host OS is Linux Ubuntu 18.04. The OS of the VM's is Ubuntu Server 16.04. There is no need for a graphical interface, so we keep it as light as we can. We will create a local interface bonded to a virtual switch and we will assign a local IP address, both on VM1 and VM2. The local IP addresses on the VM's must belong on the same subnet because there is no router between them. We will assume that our Internet Service Provider (ISP) uses DHCP, so the VM's have an interface that provides Internet connection. The primary network interface of the VM's is called `enp0s3` and the interface that will be used for the tunnel is called `vi0`. Our setup looks like Figure 11. Note: This example uses Open vSwitch which is described in detail in chapter 4.



*Figure 11. Example setup*

After installing Open vSwitch on both VM's (as described in chapter 4.4), we must run some configuration commands. We are going to use VXLAN as our tunneling protocol but the procedure is pretty much the same for GRE and Geneve.

Configuration on VM1:

```
$ sudo ovs-vsctl add-br br0
```

Creates a virtual bridge named 'br0'. We check if the bridge was created by running the command: `$ sudo ovs-vsctl show`

```
$ sudo ovs-vsctl add-port br0 vxlan0
```

Creates a port name 'vxlan0' and bonds it to br0.

```
$ sudo ovs-vsctl set Interface vxlan0 type=vxlan options:remote_ip=192.168.1.30
```

Specifies that the port 'vxlan0' is of type "VXLAN". An additional option is the 'remote\_ip' which is the IP of VM2 and is the other endpoint of the tunnel.

```
$ sudo ovs-vsctl add-port br0 vi0 -- set Interface vi0 type=internal
```

Creates an internal interface named 'vi0' which is bonded to the bridge 'br0'. We can run: `$ ifconfig vi0` to check if 'vi0' was properly created.

```
$ sudo ifconfig vi0 10.250.204.10/24 up
```

Assigns a local IP to the interface 'vi0'.

#### Configuration on VM2:

```
$ sudo ovs-vsctl add-br br0
```

```
$ sudo ovs-vsctl add-port br0 vxlan0
```

```
$ sudo ovs-vsctl set Interface vxlan0 type=vxlan options:remote_ip=192.168.1.20
```

```
$ sudo ovs-vsctl add-port br0 vi0 -- set Interface vi0 type=internal
```

```
$ sudo ifconfig vi0 10.250.204.20/24 up
```

By running: `$ ovs-vsctl show`, we are able to check if everything is set up correctly. The output of the command is similar to figures 12 and 13, for VM1 and VM2 respectively.

```

root@vm1:~# ovs-vsctl show
d24565ee-fa89-45be-8833-618b949cc575
    Bridge "br0"
        Port "vi0"
            Interface "vi0"
                type: internal
        Port "vxlan0"
            Interface "vxlan0"
                type: vxlan
                options: {remote_ip="192.168.1.30"}
    Port "br0"
        Interface "br0"
            type: internal
    ovs_version: "2.5.5"

```

*Figure 12. VM1 – “ovs-vsctl show” command output*

```

root@vm2:~# ovs-vsctl show
477ed1db-62e0-416b-ae62-4a8b92e5653a
    Bridge "br0"
        Port "vi0"
            Interface "vi0"
                type: internal
        Port "vxlan0"
            Interface "vxlan0"
                type: vxlan
                options: {remote_ip="192.168.1.20"}
    Port "br0"
        Interface "br0"
            type: internal
    ovs_version: "2.5.5"

```

*Figure 13. VM2 – “ovs-vsctl show” command output*

A simple ping test between the two machines should work. Let’s try to ping 10.250.204.20 with four ICMP packets from VM1.

```

root@vm1:~# ping -c4 10.250.204.20
PING 10.250.204.20 (10.250.204.20) 56(84) bytes of data.
64 bytes from 10.250.204.20: icmp_seq=1 ttl=64 time=0.433 ms
64 bytes from 10.250.204.20: icmp_seq=2 ttl=64 time=1.39 ms
64 bytes from 10.250.204.20: icmp_seq=3 ttl=64 time=1.38 ms
64 bytes from 10.250.204.20: icmp_seq=4 ttl=64 time=1.39 ms

--- 10.250.204.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.433/1.150/1.395/0.416 ms

```

*Figure 14. VM1 – “ping” command output*

And now let’s capture the packets on VM2 to make sure that they come from VM1. Using “tcpdump” we can see all the incoming and outgoing packets to every interface of the machine. We will adjust the command to capture only the ICMP packets on the



interface vi0, so that we get a more clear view and we don't get confused by other incoming traffic. We expect to capture eight packets. Four incoming from 10.250.204.10 and four outgoing to the same IP. Run:

```
$ tcpdump -I vi0 icmp
```

```
root@vm2:~# tcpdump -i vi0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on vi0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:13:25.699979 IP 10.250.204.10 > 10.250.204.20: ICMP echo request, id 1823, seq 1, length 64
11:13:25.700006 IP 10.250.204.20 > 10.250.204.10: ICMP echo reply, id 1823, seq 1, length 64
11:13:26.702981 IP 10.250.204.10 > 10.250.204.20: ICMP echo request, id 1823, seq 2, length 64
11:13:26.703046 IP 10.250.204.20 > 10.250.204.10: ICMP echo reply, id 1823, seq 2, length 64
11:13:27.705064 IP 10.250.204.10 > 10.250.204.20: ICMP echo request, id 1823, seq 3, length 64
11:13:27.705117 IP 10.250.204.20 > 10.250.204.10: ICMP echo reply, id 1823, seq 3, length 64
11:13:28.707608 IP 10.250.204.10 > 10.250.204.20: ICMP echo request, id 1823, seq 4, length 64
11:13:28.707696 IP 10.250.204.20 > 10.250.204.10: ICMP echo reply, id 1823, seq 4, length 64
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

*Figure 15. VM2 – “tcpdump” command output*

## 4. Virtual Private Network (VPN)

### 4.1 Definition

A virtual private network (VPN) extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network. Applications running on a computing device, e.g., a laptop, desktop, smartphone, across a VPN may therefore benefit from the functionality, security, and management of the private network. Encryption is a common, though not an inherent, part of a VPN connection. The user's initial IP address is replaced with one from the Virtual Private Network provider. Subscribers can obtain an IP address from any gateway city the VPN service provides. For instance, the user may live in San Francisco, but with a Virtual Private Network, he can appear to live in Amsterdam, New York, or any number of gateway cities. Virtual Private Networks are most often used by corporations to protect sensitive data. However, using a personal VPN is increasingly becoming more popular as more interactions that were previously face-to-face transition to the Internet.

In this thesis we are going to use a VPN as an overlay connection. That is necessary, as each cloud topology is hidden behind a NAT topology with just one external IP exposed. The protocol used is OpenVPN but in this chapter we are going to describe some of the most widely used VPN protocols. Also a quick guide on how to install OpenVPN on Ubuntu Server is included in chapter 4.3. [2]

## 4.2 VPN Protocols

Security is the main reason why corporations have used VPNs for years. VPNs use advanced encryption protocols and secure tunneling techniques to encapsulate all online data transfers. A VPN protocol determines exactly how data routes between the user's computer and the VPN server. The purpose of the tunneling protocol is to add a layer of security that protects each packet on its journey over the internet. The packet is traveling with the same transport protocol it would have used without the tunnel; this protocol defines how each computer sends and receives data over its ISP. Each inner packet still maintains the passenger protocol, such as internet protocol (IP), which defines how it travels on the LANs at each end of the tunnel. Protocols have different specifications, offering benefits to users in a range of circumstances. For instance, some prioritize speed, while others focus on privacy and security.

### 4.2.1 OpenVPN

OpenVPN is an open source VPN protocol. This means users can scrutinize its source code for vulnerabilities, or use it in other projects. OpenVPN has become one of the most important VPN protocols. As well as being open source, OpenVPN is also one of the most secure protocols. It allows users to protect their data using essentially unbreakable AES-256 bit key encryption (amongst others), with 2048-bit RSA authentication, and a 160-bit SHA1 hash algorithm. [2]

#### Advantages:

- ✓ It is a very secure protocol, being able to use 256-bit encryption keys and high-end ciphers.
- ✓ It can easily bypass any firewall it encounters.
- ✓ Since OpenVPN can use both TCP and UDP, it offers the user more control over the connections.
- ✓ It runs on a large number of platforms. Some examples include Windows, macOS, iOS, Android, Linux, routers, FreeBSD, OpenBSD, NetBSD, and Solaris.

#### Disadvantages:

- ⊗ Manually setting up the OpenVPN protocol can be rather difficult on some platforms.
- ⊗ Sometimes, there might be drops in connection speeds due to the strong encryption.
- ⊗ It requires third-party applications to run.

### 4.2.2 L2TP/IPsec

Layer 2 Tunnel Protocol is a very popular VPN protocol. L2TP is the successor to the depreciated PPTP developed by Microsoft, and L2F, developed by Cisco. However, L2TP doesn't actually provide any encryption or privacy itself. Accordingly, services that use L2TP are frequently bundled with security protocol IPsec. Once implemented, L2TP/IPsec becomes one of the most secure VPN connections available. Since the L2TP packet itself is wrapped and hidden within the IPsec packet, the original source and destination IP address is encrypted within the packet. [\[13\]](#), [\[17\]](#)

#### Advantages:

- ✓ It provides a very secure connection using AES-256 bit encryption.
- ✓ L2TP is readily available on many Windows and macOS platforms since it's built into them. It also works on many other devices and operating systems too.
- ✓ It is fairly easy to set up.

#### Disadvantages:

- ⊗ L2TP and L2TP/IPsec have been allegedly weakened or cracked by the National Security Agency of the United States (NSA).
- ⊗ Due to its double encapsulation feature, L2TP/IPsec tends to be a bit resource-intensive and not extremely fast.
- ⊗ L2TP can be blocked by NAT firewalls if it's not further configured to bypass them.
- ⊗ The protocol defaults to use UDP on port 500. This makes traffic easier to spot and block.

### 4.2.3 Point-to-Point Tunneling Protocol (PPTP)

Point-to-Point Tunneling Protocol is one of the oldest VPN protocols. It is still in use in some places, but the majority of services have long upgraded to faster and more secure protocols. PPTP was introduced way back in 1995. It was actually integrated with Windows 95, designed to work with dial-up connections. At the time, it was extremely useful. But the VPN technology has progressed, and PPTP is no longer secure. Governments and criminals cracked PPTP encryption long ago, making any data sent using the protocol unsecure. In fact, using PPTP to access sensitive information (like a bank account or credit card details) is a sure way of getting it stolen. However PPTP gives the best connection speeds, due to the lack of security features. [\[13\]](#), [\[17\]](#)

#### Advantages:

- ✓ It is a very fast VPN protocol.

- ✓ It is really easy to set up and configure on most operating systems and devices.
- ✓ Due to the protocol's high rate of cross-platform compatibility, a PPTP connection can be established on tons of platforms.

Disadvantages:

- ⊗ PPTP encryption is sub-par and not suitable for securing online data and traffic. The NSA has actually cracked PPTP traffic.
- ⊗ A PPTP connection can be exploited by cybercriminals with malicious attacks.
- ⊗ A router with PPTP Passthrough is usually required since PPTP doesn't natively work with NAT.
- ⊗ A PPTP connection can be blocked quite easily by firewalls.

#### **4.2.4 Secure Socket Tunneling Protocol (SSTP)**

SSTP is a VPN protocol that was developed by Microsoft, and introduced by them with Windows Vista. The protocol is designed to secure online data and traffic, and is considered a much safer option for Windows users than PPTP or L2TP/IPSec. It is often compared to OpenVPN thanks to the high level of security it offers. [13], [17]

Advantages:

- ✓ SSTP encryption offers a decent level of security, almost on par with OpenVPN.
- ✓ It is easy to configure on platforms it is built into.
- ✓ It is very difficult to block because it uses TCP port 443 (the same one HTTPS uses).
- ✓ SSTP offers good speeds if the user has enough bandwidth.
- ✓ It can bypass NAT firewalls.

Disadvantages:

- ⊗ SSTP is closed-source and solely owned by Microsoft.
- ⊗ It is available on a limited number of platforms – Windows, Linux, Android, and routers.
- ⊗ SSTP connections could be dropped if the network admin spots the SSTP header (which is possible to do since the protocol doesn't support authenticated web proxies).
- ⊗ It only works on TCP and is susceptible to the "TCP Meltdown" issue.

For more information please refer to [4].

### 4.2.5 Internet Key Exchange version 2 (IKEv2)

IKEv2 is a VPN encryption protocol that handles request and response actions. It assures the traffic is secure by establishing and handling the SA (Security Association) attribute within an authentication suite – usually IPsec since IKEv2 is basically based on it and built into it. IKEv2 was developed by Microsoft together with Cisco. [13], [17]

#### Advantages:

- ✓ IKEv2 security is quite strong since it supports multiple high-end ciphers.
- ✓ Despite its high security standard, IKEv2 offers fast online speeds.
- ✓ It can easily resist network changes due to its MOBIKE support, and can automatically restore dropped connections.
- ✓ It is natively available on BlackBerry devices, and can be configured on other mobile devices too.
- ✓ Setting up an IKEv2 VPN connection is relatively simple.

#### Disadvantages:

- ⊗ IKEv2 only uses UDP port 500, so a firewall or network admin could block it.
- ⊗ It doesn't offer as much cross-platform compatibility like other protocols (PPTP, L2TP, OpenVPN).

VPN Tunneling Protocol					
Features	PPTP	L2TP/IPSec	IKEv2/IPSec	SSTP	Open VPN
<i>Ease of setup</i>	Very easy	Easy	Easy	Easy	Tricky on its own, but easy if you use a good VPN
<i>Stability</i>	Struggles with most blocking software	Struggles with most blocking software	Struggles with some blocking software	Bypasses most blocking software	Bypasses most blocking software
<i>Encryption</i>	Basic	Strong	Very strong	Very strong	Very strong
<i>Speed</i>	Fast	A bit slow	Fast	Fast	Best performance
<i>Security</i>	Not secure	Secure but may be breakable	Very secure, but users are not free to access the codes to verify security claims	Very secure, but users are not free to access the codes to verify security claims	Very secure, and anyone is free to access the codes to verify security claims
<i>Compatibility</i>	Window, Mac OS, Linux, etc.	Window, Mac OS, Linux, etc.	Window, Mac OS, Linux, etc.	Windows only	Window, Mac OS, Linux, etc.

Figure 16. VPN protocols comparison. Source: <https://www.vpnmentor.com/blog/ultimate-guide-to-vpn-tunneling/>

### 4.3 OpenVPN Installation

The VPN server used for this thesis is running on a VM using Ubuntu 16.04 Server. The VM was created with Oracle VM Virtualbox. The amount of RAM used is 1 GB, the virtual NIC is in bridged mode with our physical Ethernet adapter and the graphics memory is set to 16 MB since we will not need graphics. The installation procedure is described below. We are going to use TCP protocol and port 443. It is supposed that the VM is up and running and the user has superuser privileges.

```
$ apt update && apt upgrade -y
```

Updates and upgrades the system.

```
$ apt-get install openvpn easy-rsa -y
```

Installs Openvpn and easy-rsa packages.

In order to issue trusted certificates, we need to set up our own simple certificate authority (CA).

```
$ make-cadir ~/openvpn-ca
```

Copies the easy-rsa template directory into home directory.

```
$ cd ~/openvpn-ca
```

Moves into the newly created directory to begin configuring the CA.

```
$ sudo nano vars
```

Edit:

```
export KEY_CONFIG=$EASY_RSA/openssl-1.0.0.cnf
```

```
export KEY_NAME="server"
```

```
$ source vars
```

Sources the vars file.

```
$ ./clean-all
```

This command assures that we are operating in a clean environment.

```
$ ./build-ca
```

Builds the root CA.

```
$ ./build-key-server server
```

Generates the OpenVPN server certificate and key pair.

```
$ ./build-dh
```

Generates a strong Diffie-Hellman key to use during key exchange.

```
$ openvpn --genkey --secret keys/ta.key
```

Generates an HMAC signature to strengthen the server's Transport Layer Security (TLS) integrity verification capabilities.

```
$ source vars
```

```
$ ./build-key client1
```

Produces credentials without a password, to aid in automated connections.

```
$ cd keys
```

```
$ sudo cp ca.crt server.crt server.key ta.key dh2048.pem  
/etc/openvpn
```

Copies the CA cert, the server cert and key, the HMAC signature and the Diffie-Hellman file in the openvpn directory.

```
$ gunzip -c /usr/share/doc/openvpn/examples/sample-con-  
fig-files/server.conf.gz | sudo tee /etc/open-  
vpn/server.conf
```

Copies and unzips a sample OpenVPN configuration file into configuration directory so that we can use it as a basis for our setup.

```
$ sudo nano /etc/openvpn/server.conf
```

Uncomment: *tls-auth ta.key 0, cipher AES-128-CBC, user nobody, group nogroup,  
uncomment push "dhcp-option DNS 208.67.222.222", push "dhcp-option DNS  
208.67.220.220"*

Add: *key-direction 0, auth SHA256*

```
$ sudo nano /etc/sysctl.conf
```

Uncomment: *net.ipv4.ip\_forward=1*

```
$ sudo nano /etc/ufw/before.rules
```

Add:

```
*nat  
:POSTROUTING ACCEPT [0:0]  
-A POSTROUTING -s 10.8.0.0/8 -o $NetworkInterface -j MASQUERADE  
COMMIT
```

```
$ sudo nano /etc/default/ufw
```

Edit: *DEFAULT\_FORWARD\_POLICY="ACCEPT"*

```
$ sudo ufw allow 443/tcp
```

Opens the UDP traffic to port 1194.

```
$ sudo ufw allow OpenSSH
```

Opens the SSH port.

```
$ sudo ufw disable
```

We need to disable and re-enable the UFW to load the changes from all the files we have modified.

```
$ sudo ufw enable
```

```
$ sudo systemctl start openvpn@server
```

Starts the OpenVPN server by specifying the configuration file name as an instance variable (in our case this file is server.conf).

```
$ sudo systemctl enable openvpn@server
```

Enables the service so that it starts automatically at boot.

```
$ mkdir -p ~/client-configs/files
```

Creates a directory structure within the home directory to store the files.

```
$ chmod 700 ~/client-configs/files
```

Locks down the permission in the inner directory, since all client configuration files will have the client keys embedded.

```
$ cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf ~/client-configs/base.conf
```

Copies an example client configuration into the new directory to be used as the base configuration.

```
$ nano ~/client-configs/base.conf
```

Uncomment: *user nobody, group nogroup*

Comment: *ca crt, cert client.crt, key client.key*

Edit: *remote EXTERNAL\_IP 1194, cipher AES-128-CBC*

Add: *auth SHA256, key-direction 1*



```
$ nano ~/client-configs/make_config.sh
```

We will create a simple script to compile our base configuration with the relevant certificate, key and encryption files.

Paste the following:

```
#!/bin/bash

# First argument: Client identifier

KEY_DIR=~/.openvpn-ca/keys
OUTPUT_DIR=~/.client-configs/files
BASE_CONFIG=~/.client-configs/base.conf

cat ${BASE_CONFIG} \
    <(echo -e '<ca>') \
    ${KEY_DIR}/ca.crt \
    <(echo -e '</ca>\n<cert>') \
    ${KEY_DIR}/${1}.crt \
    <(echo -e '</cert>\n<key>') \
    ${KEY_DIR}/${1}.key \
    <(echo -e '</key>\n<tls-auth>') \
    ${KEY_DIR}/ta.key \
    <(echo -e '</tls-auth>') \
    > ${OUTPUT_DIR}/${1}.ovpn
```

```
$ chmod 700 ~/client-configs/make_config.sh
```

Makes the file executable.

```
$ cd ~/client-configs
```

Move into the client-configs directory.

```
$ ./make_config.sh client1
```

We created a client certificate and key by running the `./build-key client1` command in the previous steps. Now, we use the script we created to generate a configuration file for those credentials. It is named “*client1.ovpn*” and is located under the `~/client-configs/files` directory.

That is all the configuration we need to do. The VPN server is now functional.

## 5. Open vSwitch

### 5.1 The role of Open vSwitch

Hypervisors need the ability to bridge traffic between VMs and with the outside world. On Linux-based hypervisors, this used to mean using the built-in L2 switch (the Linux bridge), which is fast and reliable. Virtualized environments require a new approach to switching. This new approach is Open vSwitch (OVS).

OVS, is an open-source implementation of a distributed virtual multilayer switch. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). OVS is targeted at multi-server virtualization deployments, a landscape for which the previous stack (the Linux bridge) is not well suited. These environments are often characterized by highly dynamic end-points, the maintenance of logical abstractions and (sometimes) integration with or offloading to special purpose switching hardware. Open vSwitch can operate both as a software-based network switch running within a virtual machine (VM) hypervisor, and as the control stack for dedicated switching hardware; as a result, it has been ported to multiple virtualization platforms, switching chipsets, and networking hardware accelerators.

Open vSwitch is the default switch in XenServer 6.0, the Xen Cloud Platform and also supports Xen, KVM, Proxmox VE and VirtualBox. It has also been integrated into many virtual management systems including Openstack openQRM, OpenNebula and oVirt. The kernel datapath is distributed with Linux, and packages are available for Ubuntu, Debian openSUSE and Fedora. Open vSwitch is also supported on FreeBSD and NetBSD. [9]

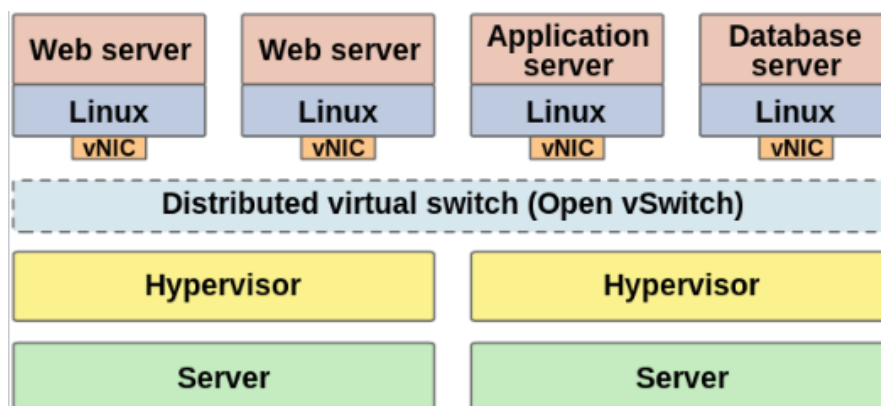


Figure 17. Open vSwitch deployed as a cross-server virtual network switch, distributed across multiple physical servers

## 5.2 Features

Open vSwitch supports a number of features that allow a network control system to respond and adapt as the environment changes. This includes simple accounting and visibility support such as NetFlow, IPFIX, and sFlow. But perhaps more useful, Open vSwitch supports a network state database (OVSDB) that supports remote triggers. Therefore, a piece of orchestration software can "watch" various aspects of the network and respond if/when they change. This is used heavily today, for example, to respond to and track VM migrations. Open vSwitch also supports OpenFlow as a method of exporting remote access to control traffic. There are a number of uses for this including global network discovery through inspection of discovery or link-state traffic (e.g. LLDP, CDP, OSPF, etc.). [9]

All supported features are presented below.

- Exposed communication between virtual machines, via NetFlow, sFlow, IP Flow Information Export (IPFIX), Switched Port Analyzer (SPAN), Remote Switched Port Analyzer (RSPAN), and port mirrors tunneled using Generic Routing Encapsulation (GRE)
- Link aggregation through the Link Aggregation Control Protocol (LACP, IEEE 802.1AX-2008)
- Standard 802.1Q virtual LAN (VLAN) model for network partitioning, with support for trunking
- Support for multicast snooping using versions 1, 2 and 3 of the Internet Group Management Protocol (IGMP)
- Support for the Shortest Path Bridging Media Access Control (SPBM) and associated basic support for the Link Layer Discovery Protocol (LLDP)
- Support for the Bidirectional Forwarding Detection (BFD) and 802.1ag link monitoring
- Support for the Spanning Tree Protocol (STP, IEEE 802.1D-1998) and Rapid Spanning Tree Protocol (RSTP, IEEE 802.1D-2004)
- Fine-grained quality of service (QoS) control for different applications, users, or data flows
- Support for the hierarchical fair-service curve (HFSC) queuing discipline (qdisc)
- Traffic policing at the level of virtual machine interface
- Network interface controller (NIC) bonding, with load balancing by source MAC addresses, active backups, and layer 4 hashin
- Support for the OpenFlow protocol, including various virtualization-related extensions

- Complete IPv6 (Internet Protocol version 6) support
- Support for multiple tunneling protocols, including RE, Virtual Extensible LAN (VXLAN), Stateless Transport Tunneling (STT) and Geneve, with additional support for layering over Internet Protocol Security (IPsec)
- Remote configuration protocol, with existing bindings for the C and Python programming languages
- Multi-table forwarding pipeline with a flow-caching engine

## 5.3 Commands

OVS is feature rich with different configuration commands, but the majority of the configuration and troubleshooting can be accomplished with the following 4 commands:

- **ovs-vsctl**: Used for configuring the ovs-vswitchd configuration database (known as ovs-db)
- **ovs-ofctl**: A command line tool for monitoring and administering OpenFlow switches
- **ovs-dpctl**: Used to administer Open vSwitch datapaths
- **ovs-appctl**: Used for querying and controlling Open vSwitch daemons

For more information please refer to: [\[15\]](#)

### 5.3.1 Ovs-vsctl

This tool is used for configuration and viewing OVS switch operations. Port configuration, bridge additions/deletions, bonding, and VLAN tagging are just some of the options that are available with this command. Below are the most useful ‘show’ commands and the common switch configuration commands.

#### 5.3.1.1 ‘Show’ commands

- **ovs-vsctl -v**: Prints the current version of Open vSwitch
- **ovs-vsctl show**: Prints a brief overview of the switch database configuration.
- **ovs-vsctl list-br**: Prints a list of configured bridges
- **ovs-vsctl list-ports <bridge>**: Prints a list of ports on a specific bridge.

#### 5.3.1.2 Switch configuration commands

- **ovs-vsctl add-br <bridge>**: Creates a bridge in the switch database.

- **ovs-vsctl add-port <bridge> <interface>** : Binds an interface (physical or virtual) to a bridge.
- **ovs-vsctl add-port <bridge> <interface> tag=<VLAN number>**: Converts port to an access port on specified VLAN (by default all OVS ports are VLAN trunks).
- **ovs-vsctl set interface <interface> type=patch options:peer=<interface>** : Used to create patch ports to connect two or more bridges together.

### 5.3.2 Ovs-ofctl

This tool is used for administering and monitoring OpenFlow switches. Even if OVS isn't configured for centralized administration, ovs-ofctl can be used to show the current state of OVS including features, configuration, and table entries. Below are the most common 'show' and switch configuration commands.

#### 5.3.2.1 'Show' commands

- **ovs-ofctl show <bridge>**: Shows OpenFlow features and port descriptions.
- **ovs-ofctl snoop <bridge>**: Snoops traffic to and from the bridge and prints to console.
- **ovs-ofctl dump-flows <bridge> <flow>**: Prints flow entries of specified bridge. With the flow specified, only the matching flow will be printed to console. If the flow is omitted, all flow entries of the bridge will be printed.
- **ovs-ofctl dump-ports-desc <bridge>**: Prints port statistics. This will show detailed information about interfaces in this bridge, include the state, peer, and speed information.
- **ovs-ofctl dump-tables-desc <bridge>**: Similar to above but prints the descriptions of tables belonging to the stated bridge.
- **ovs-ofctl dump-ports-desc**: this command is useful for viewing port connectivity.

#### 5.3.2.2 Switch configuration commands

- **ovs-ofctl add-flow <bridge> <flow>**: Add a static flow to the specified bridge. Useful in defining conditions for a flow (i.e. prioritize, drop, etc.).
- **ovs-ofctl del-flows <bridge> <flow>**: Delete the flow entries from flow table of stated bridge. If the flow is omitted, all flows in specified bridge will be deleted.

### 5.3.3 Ovs-dpctl

Ovs-dpctl is very similar to ovs-ofctl. They both show flow table entries. The flows that ovs-dpctl prints are always an exact match and reflect packets that have actually passed through the system within the last few seconds. Ovs-dpctl queries a kernel datapath and not an OpenFlow switch. This is why it's useful for debugging flow data.

### 5.3.4 Ovs-appctl

OVS is comprised of several daemons that manage and control an Open vSwitch switch. Ovs-appctl is a utility for managing these daemons at runtime. It is useful for configuring log module settings as well as viewing all OpenFlow flows, including hidden ones. Below are the most useful commands.

- **ovs-appctl bridge/dump-flows <bridge>**: Dumps OpenFlow flows, including hidden flows. Useful for troubleshooting in-band issues.
- **ovs-appctl dpif/dump-flows <bridge>**: Dumps datapath flows for only the specified bridge, regardless of the type.
- **ovs-appctl vlog/list**: Lists the known logging modules and their current levels
- **ovs-appctl vlog/set**: sets/changes the module log level.

## 5.4 Installation

As far as the installation of OVS is concerned, the steps are described below. The operating system used is Ubuntu Desktop 18.04 but it works on most Ubuntu versions.

Direct installation (through terminal):

```
$ sudo apt update
```

Downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies

```
$ sudo apt upgrade
```

Fetches new versions of packages existing on the machine if APT knows about these new versions by way of apt-get update.

```
$ sudo apt install openvswitch-switch
```

Installs Open vSwitch.

```
$ sudo ovs-vswitchd
```

Starts the OVS daemon.

## 6. Openstack

### 6.1 What is Openstack?

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication mechanisms. A dashboard is also available, giving administrators control while empowering their users to provision resources through a web interface. Beyond standard infrastructure-as-a-service functionality, additional components provide orchestration, fault management and service management amongst other services to ensure high availability of user applications. OpenStack began in 2010 as a joint project of Rackspace Hosting and NASA. As of 2012, it is managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012 to promote OpenStack software and its community. More than 500 companies have joined the project.

In this thesis Openstack is used as the Cloud provider. Openstack will be used through Devstack. Devstack is a series of extensible scripts used to quickly bring up a complete OpenStack environment based on the latest versions of everything from git master. It is used interactively as a development environment and as the basis for much of the OpenStack project's functional testing. This chapter includes a description of Openstack components, commands, a quick installation guide and an example of launching a simple instance.

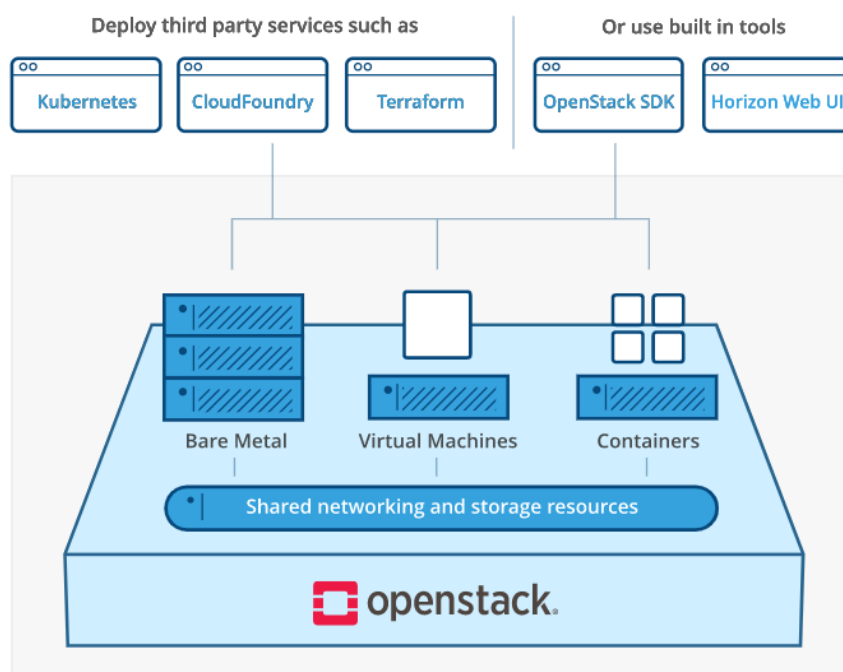
































Figure 18. Openstack

## 6.2 Openstack Components

Openstack consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center. [10]

	<b>NOVA</b>	Compute service
	<b>ZUN</b>	Containers service
	<b>QINLING</b>	Functions service
	<b>IRONIC</b>	Bare Metal Provisioning service
	<b>CYBORG</b>	Lifecycle management of accelerators
	<b>SWIFT</b>	Object store
	<b>CINDER</b>	Block Storage
	<b>MANILA</b>	Shared filesystems
	<b>NEUTRON</b>	Networking
	<b>OCTAVIA</b>	Load balancer
	<b>DESIGNATE</b>	DNS service
	<b>KEYSTONE</b>	Identity service
	<b>PLACEMENT</b>	Placement service
	<b>GLANCE</b>	Image service
	<b>BARBICAN</b>	Key management
	<b>KARBOR</b>	Application Data Protection as a Service
	<b>SEARCHLIGHT</b>	Indexing and Search
	<b>HEAT</b>	Orchestration
	<b>SENLIN</b>	Clustering service



	<b>MISTRAL</b>	Workflow service
	<b>ZAQAR</b>	Messaging service
	<b>BLAZAR</b>	Resource reservation service
	<b>AODH</b>	Alarming service
	<b>MAGNUM</b>	Container Orchestration Engine Provisioning
	<b>SAHARA</b>	Big Data Processing Framework Provisioning
	<b>TROVE</b>	Database as a service
	<b>MASAKARI</b>	Instances High Availability service
	<b>MURANO</b>	Application Catalog
	<b>SOLUM</b>	Software Development Lifecycle Automation
	<b>FREEZER</b>	Backup, Restore and Disaster Recovery
	<b>EC2API</b>	EC2 API proxy
	<b>HORIZON</b>	Dashboard

*Table 1. Openstack components*

### 6.2.1 Compute

**Nova:** Nova is the OpenStack project that provides a way to provision compute instances (aka virtual servers). Nova supports creating virtual machines, bare metal servers (through the use of ironic), and has limited support for system containers. Nova runs as a set of daemons on top of existing Linux servers to provide that service. Nova is written in Python. It uses many external Python libraries such as Eventlet (concurrent networking library), Kombu (AMQP messaging framework), and SQLAlchemy (SQL toolkit and Object Relational Mapper). Nova is designed to be horizontally scalable. Rather than switching to larger servers, we procure more servers and simply install identically configured services.

**Zun:** Provides an OpenStack API for launching and managing containers backed by different container technologies. Different from Magnum, Zun is for users who want to treat containers as OpenStack-managed resource. Containers managed by Zun

are supposed to be integrated well with other OpenStack resources, such as Neutron network and Cinder volume. Users are provided a simplified APIs to manage containers without the need to explore the complexities of different container technologies.

**Qinling:** Provides a platform to support serverless functions.

### 6.2.2 Hardware Lifecycle

**Ironic:** To implement services and associated libraries to provide massively scalable, on demand, self service access to compute resources, including bare metal, virtual machines, and containers.

**Cyborg:** Provides a general purpose management framework for accelerators (including GPUs, FPGAs, ASIC-based devices, etc.)

### 6.2.3 Storage

**Swift:** Swift is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply. It's built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound.

**Cinder:** It is a Block Storage service. It virtualizes the management of block storage devices and provides end users with a self service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device. This is done through the use of either a reference implementation (LVM) or plugin drivers for other storage.

**Manila:** Provides coordinated access to shared or distributed file systems.

### 6.2.4 Networking

**Neutron:** Neutron is an OpenStack project to provide “network connectivity as a service” between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova). It implements the OpenStack Networking API. It manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in the OpenStack environment. OpenStack Networking enables projects to create advanced virtual network topologies which may include services such as a firewall, and a virtual private network (VPN). Neutron allows dedicated static IP addresses or DHCP. It also allows Floating IP addresses to let traffic be dynamically rerouted.

**Octavia:** It is an open source, operator-scale load balancing solution designed to work with OpenStack. Octavia was borne out of the Neutron LBaaS project, and starting with the Liberty release of OpenStack, Octavia has become the reference implementation for Neutron LBaaS version 2. Octavia accomplishes its delivery of load balancing services by managing a fleet of virtual machines, containers, or bare metal

servers —collectively known as amphorae— which it spins up on demand. This on-demand, horizontal scaling feature differentiates Octavia from other load balancing solutions, thereby making Octavia truly suited “for the cloud.”

**Designate:** It provides DNS-as-a-service for OpenStack

### 6.2.5 Shared services

**Keystone:** Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. It is the common authentication system across the cloud operating system. Keystone can integrate with directory services like LDAP. It supports standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. The OpenStack keystone service catalog allows API clients to dynamically discover and navigate to cloud services.

**Placement:** Placement is an OpenStack service that provides an HTTP API for tracking cloud resource inventories and usages to help other services effectively manage and allocate their resources.

**Glance:** Glance image services include discovering, registering, and retrieving virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project.

**Barbican:** It is the OpenStack Key Manager service. It provides secure storage, provisioning and management of secret data, such as passwords, encryption keys, X.509 Certificates and raw binary data.

**Karbor:** Karbor deals with protecting the Data and Meta-Data that comprises an OpenStack-deployed application against loss/damage (e.g. backup, replication) - not to be confused with Application Security or DLP. It does that by providing a standard framework of APIs and services that enables vendors to introduce various data protection services into a coherent and unified flow for the user.

**Searchlight:** The Searchlight project provides indexing and search capabilities across OpenStack resources. Its goal is to achieve high performance and flexible querying combined with near real-time indexing. It uses Elasticsearch, a real-time distributed indexing and search engine built on Apache Lucene, but adds OpenStack authentication and Role Based Access Control to provide appropriate protection of data.

### 6.2.6 Orchestration

**Heat:** Orchestrates the infrastructure resources for a cloud application based on templates in the form of text files that can be treated like code. Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API. It also

provides an autoscaling service that integrates with the OpenStack Telemetry services, so a scaling group can be included as a resource in a template.

**Senlin:** Senlin is a clustering service for OpenStack clouds. It creates and operates clusters of homogeneous objects exposed by other OpenStack services. The goal is to make orchestration of collections of similar objects easier.

**Mistral:** It is a workflow service. Most business processes consist of multiple distinct interconnected steps that need to be executed in a particular order in a distributed environment. One can describe such process as a set of tasks and task relations (via YAML-based language) and upload such description to Mistral so that it takes care of state management, correct execution order, parallelism, synchronization and high availability.

**Zaqar:** Zaqar is a multi-tenant cloud messaging service for web and mobile developers. The service features a fully RESTful API, which developers can use to send messages between various components of their SaaS and mobile applications. Underlying this API is an efficient messaging engine designed with scalability and security in mind. Other OpenStack components can integrate with Zaqar to surface events to end users and to communicate with guest agents that run in the "over-cloud" layer. Cloud operators can leverage Zaqar to provide equivalents of SQS and SNS to their customers.

**Blazar:** It is a resource reservation service for OpenStack. Blazar enables users to reserve a specific type/amount of resources for a specific time period and it leases these resources to users based on their reservations.

**AODH:** Aodh's goal is to enable the ability to trigger actions based on defined rules against sample or event data collected by Ceilometer.

### 6.2.7 Workload Provisioning

**Magnum:** Magnum makes container orchestration engines such as Docker Swarm, Kubernetes, and Apache Mesos available as first class resources in OpenStack. It uses Heat to orchestrate an OS image which contains Docker and Kubernetes and runs that image in either virtual machines or bare metal in a cluster configuration.

**Sahara:** The Sahara project aims to provide users with a simple means to provision data processing frameworks (such as Hadoop, Spark and Storm) on OpenStack. This is accomplished by specifying configuration parameters such as the framework version, cluster topology, node hardware details and more.

**Trove:** It is a database-as-a-service provisioning relational and non-relational database engines.

## 6.2.8 Application Lifecycle

**Masakari:** Masakari provides Instances High Availability Service for OpenStack clouds by automatically recovering failed Instances. Currently, Masakari can recover KVM-based Virtual Machine(VM)s from failure events such as VM process down, provisioning process down, and nova-compute host failure. Masakari also provides an API service to manage and control the automated rescue mechanism.

**Murano:** It enables application developers and cloud administrators to publish various cloud-ready applications in a browsable catalog. Cloud users -- including inexperienced ones -- can then use the catalog to compose reliable application environments with the push of a button. Murano uses OpenStack Heat to orchestrate infrastructure resources for the application.

**Solum:** To make cloud services easier to consume and integrate with the application development process by automating the source-to-image process, and simplifying app-centric deployment.

**Freezer:** Freezer is a distributed backup, restore and disaster recovery as a service platform. It is designed to be multi OS (Linux, Windows, OSX...), focused on providing efficiency and flexibility for block based backups, file based incremental backups, point-in-time actions, jobs synchronization (i.e. backup synchronization over multiple nodes) and many other features.

## 6.2.9 API Proxies

**EC2API:** It Provides an EC2-compatible API to OpenStack Nova.

## 6.2.10 Web Frontend

**Horizon:** Horizon is the canonical implementation of OpenStack's dashboard, which is extensible and provides a web based user interface to OpenStack services.

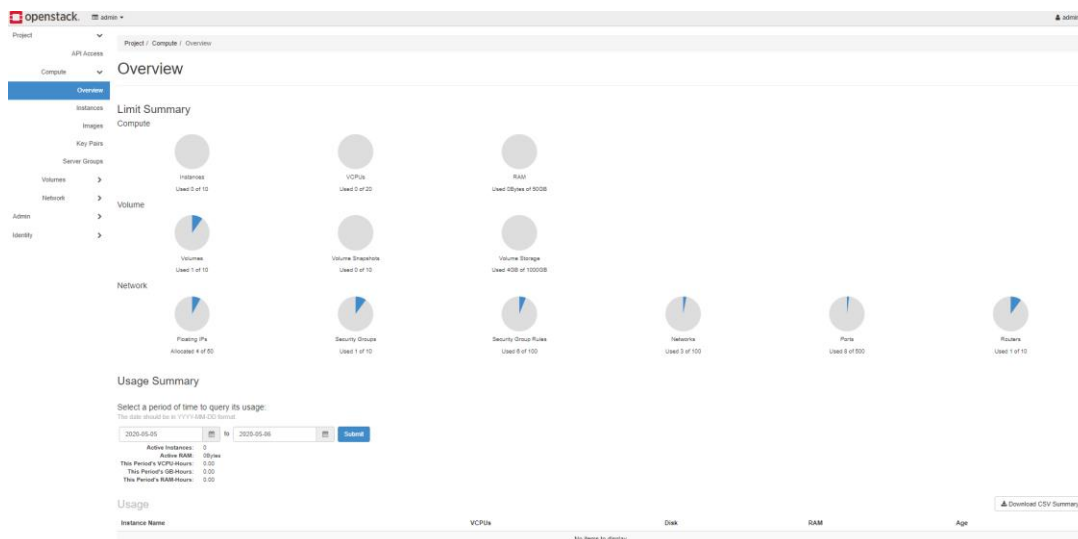


Figure 19. Openstack dashboard

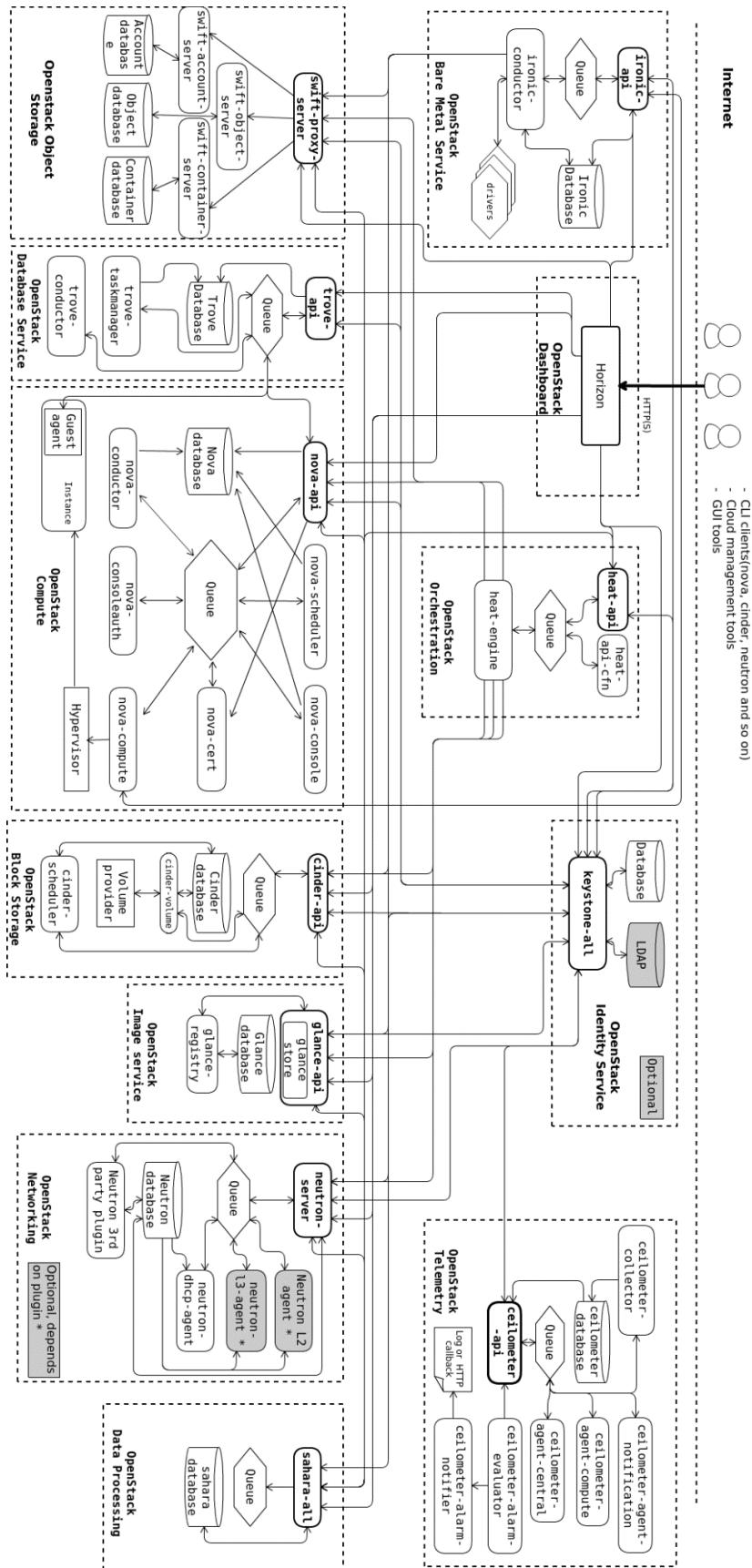


Figure 20. Openstack architecture

## 6.3 Commands

One way to interact with Openstack is the dashboard. This is a pretty straightforward way and there is no need for explanation. Another way is through the terminal. Openstack has many built-in commands that allow the user to manage every detail. In this section the most common used commands are explained. Also a simple example of creating a new project, a new image and launching a server is presented.

For one to be able to manage Openstack from terminal, must source the Openstack file with the cloud's credentials. This file can be easily downloaded from the Dashboard at the top right corner. After it is downloaded, run the following command, providing the cloud's password.

```
$ source /path/to/the/file
```

Let's have now a close look at some Openstack commands that are enough for a simple use of the cloud. Most of these commands have a lot options accompanying them but we will present them at the simplest way. For more information refer to the official site of Openstack. [\[11\]](#)

### 6.3.1 Project management

This set of commands is used for project managements (creation, deletion, listing etc.).

Create new project:

```
$ openstack project create <name>
```

Delete project:

```
$ openstack project delete <project>
```

List projects:

```
$ openstack project list
```

Display project details:

```
$ openstack project show <project>
```

Set project properties:

```
$ openstack project set <project>
```

Unset project properties:

```
$ openstack project unset <project>
```

### 6.3.2 Image management

This set of commands is used for image management (creation, deletion, display image details etc.).

Create/upload an image:

```
$ openstack image create <image-name>
```

Delete an image:

```
$ openstack image delete <image>
```

List available images:

```
$ openstack image list
```

Associate project with image:

```
$ openstack image add project <image> <project>
```

Disassociate project with image:

```
$ openstack image remove project <image> <project>
```

Display image details:

```
$ openstack image show <image>
```

Set image properties:

```
$ openstack image set <image>
```

Unset image properties:

```
$ openstack image unset <image>
```

### **6.3.3 Volume management**

This set of commands is used for volume management (creation, deletion, listing etc.).

Create new volume:

```
$ openstack volume create <name>
```

Delete volume:

```
$ openstack volume delete <volume>
```

List volumes:

```
$ openstack volume list
```

Migrate volume to a new host:

```
$ openstack volume migrate --host <host> <volume>
```



Display volume details:

```
$ openstack volume show <volume>
```

Set volume properties:

```
$ openstack volume set <volume>
```

Unset volume properties:

```
$ openstack volume unset <volume>
```

### 6.3.4 Server management

This set of commands is used for server management (creation, deletion, pausing, re-booting etc.).

Create a new server:

```
$ openstack server create <server-name>
```

Delete an existing server:

```
$ openstack server delete <server>
```

Add or remove floating IP from server:

```
$ openstack server add/remove floating ip <server> <ip-address>
```

Add or remove volume from server:

```
$ openstack server add/remove volume <server> <volume>
```

Add or remove security group from server:

```
$ openstack server add/remove security group <server>
```

List servers:

```
$ openstack server list
```

Migrate server to a new host:

```
$ openstack server migrate --host <host> <server>
```

Start/unpause/pause/stop a server:

```
$ openstack server start/unpause/pause/stop <server>
```

Perform a hard or soft server reboot:

```
$ openstack server reboot [--hard | --soft] <server>
```

### 6.3.5 Network and subnet management

This set of commands is used for managing networks and subnets (creation, deletion, listing etc.). A network is an isolated Layer 2 networking segment. There are two types of networks, project and provider networks. Project networks are fully isolated and are not shared with other projects. Provider networks map to existing physical networks in the data center and provide external network access for servers and other resources. Only an OpenStack administrator can create provider networks. Networks can be connected via routers. A subnet is a block of IP addresses and associated configuration state. Subnets are used to allocate IP addresses when new ports are created on a network.

Create network/subnet:

```
$ openstack network/subnet create <name>
```

Delete network/subnet:

```
$ openstack network/subnet delete <network/subnet>
```

List networks/subnets:

```
$ openstack network/subnet list
```

Display network/subnet details:

```
$ openstack network/subnet show <network/subnet>
```

Set network/subnet properties:

```
$ openstack network set <network/subnet>
```

Unset network/subnet properties:

```
$ openstack network unset <network/subnet>
```

Of course, these commands take a lot of arguments such as defining the network's IP pool or DHCP server. These are critical for the network/subnet creation and must not be undeclared.

### 6.3.6 Router management

This set of commands is used for router management (creation, deletion, listing etc.). A router is a logical component that forwards data packets between networks. It also provides Layer 3 and NAT forwarding to provide external network access for servers on project networks.

Create new router:

```
$ openstack router create <name>
```

Delete router:

```
$ openstack router delete <router>
```

Add or remove a port to/from a router:

```
$ openstack router add/remove port <router> <port>
```

Add or remove a subnet to/from a router:

```
$ openstack router add/remove <router> <subnet>
```

List routers:

```
$ openstack router list
```

Display router details:

```
$ openstack router show <router>
```

Set router properties:

```
$ openstack router set <router>
```

Unset router properties:

```
$ openstack router unset <router>
```

### 6.3.7 Security groups management

This set of commands is used for managing the security groups (creation, deletion, listing etc.). A security group acts as a virtual firewall for servers and other resources on a network. It is a container for security group rules which specify the network access rules.

Create new security group:

```
$ openstack security group create <name>
```

Delete security group:

```
$ openstack security group delete <security group>
```

List security groups:

```
$ openstack security group list
```

Display security group's details:

```
$ openstack security group show <security group>
```

Set security group's properties:

```
$ openstack security group set <security group>
```

### 6.3.8 Volume management

This set of commands is used for floating IP management (creation, deletion, listing etc.). Floating IP's are used so that the instances are accessible outside the Openstack network. These IP's are usually allocated from Openstack's external network pool.

Create new floating IP:

```
$ openstack floating ip create <network>
```

Delete floating IP:

```
$ openstack floating ip delete <floating-ip>
```

List floating IP's:

```
$ openstack floating ip list
```

Display floating IP details:

```
$ openstack floating IP show <floating-ip>
```

Set floating IP properties:

```
$ openstack floating ip set <floating-ip>
```

Unset floating IP properties:

```
$ openstack volume unset <floating-ip>
```

### 6.3.9 Keypair management

This set of commands is used for keypair management (creation, deletion, listing etc.). Keypair is the public key of an OpenSSH key pair to be used for access to created servers.

Create new keypair:

```
$ openstack keypair create <name>
```

Delete keypair:

```
$ openstack keypair delete <key>
```

List key fingerprints:

```
$ openstack keypair list
```

Display key details:

```
$ openstack keypair show <key>
```

## 6.4 Installation

In this chapter, a short guide for installing Openstack with Devstack on Ubuntu 18.04 LTS will be presented. The requirements are: a) fresh installation of Ubuntu, b) minimum memory of 4 GB, c) at least 2 vCPU's, d) storage capacity of 10 GB, e) Internet connection, f) user with superuser privileges. We will cover a minimal installation of Devstack including the activation of Heat services. Open a new terminal and execute the following commands.

```
$ apt update && apt upgrade -y
```

Updates the system.

```
$ init 6
```

Reboots the system.

```
$ useradd -s /bin/bash -d /opt/stack -m stack
```

Creates a new user for running Devstack.

```
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee  
/etc/sudoers.d/stack
```

Give the new user superuser privileges.

```
$ su - stack
```

Switches to user stack.

```
$ sudo apt -y install git
```

Installs git package so that we can clone Devstack in the next step.

```
$ git clone https://opendev.org/openstack-dev/devstack -b  
master
```

Clones into Devstack.

```
$ cd devstack
```

The next step is to edit the main configuration file of Devstack named *local.conf*. In this file we declare the cloud's password and the services that we need to be deployed. In this example we just install some API's and the Heat plug-in.

```
$ nano local.conf
```

Paste:

```
[[local|localrc]]

# Password for KeyStone, Database, RabbitMQ and Service
ADMIN_PASSWORD=openstack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD

# Host IP - get the Server's/VM's IP address from ip addr command
HOST_IP=$(HOST_IP)

#Enable heat services
enable_service h-eng h-api h-api-cfn h-api-cw
enable_plugin heat https://opendev.org/openstack/heat master
```

---

**\$ ./stack.sh**

That step is going to last about 12-15 minutes, depending on the PC and the Internet connection. When it is finished the cloud can be accessed from any browser, typing:

[https://HOST\\_IP/dashboard](https://HOST_IP/dashboard)

For more information on how to install Devstack, visit: <https://docs.openstack.org/devstack/latest/>

## 6.5 Example

In this example, we will use the Openstack cloud to give the reader a more clear view of the commands that are presented in chapter 6.3. The setup is consisted of one PC running Ubuntu Desktop 18.04 LTS and Openstack has been installed using Devstack. By default, when Devstack is installed, it comes with three projects (admin, alt-demo and demo), one image (cirros-0.4.0-x86\_64-disk), a public network (172.24.4.0/24), an internal network (10.0.0.0/24) and a security group. In this example we will use the admin project and the public network to assign a floating IP to the instance so we can have access to it from Host's terminal (via SSH).

We source the credential's file as explained in the beginning of chapter 4.3. The next step is to upload a new image. In this example we will use Ubuntu Server 16.04 at its cloud version. Download it by running:

```
$ wget https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk
```

To upload it to the cloud, we run:

```
$ openstack image create --disk-format qcow2 --min-disk 1 --min-ram 64 --file /path/to/image xenial1
```

If we run: `$ openstack image list` then the output should be similar to figure 21.

ID	Name	Status
be837dc7-2430-4daa-9279-3589eb88f9c3	cirros-0.4.0-x86_64-disk	active
e9ad70c4-438e-483b-bff7-931b2181bad6	xenial1	active

Figure 21. Command output

Now let's create a keypair so that we can SSH into the server we will create in the next steps.

```
$ keypair create KEYPAIR
```

That command automatically saves the key that was just created, on the PC. The next step is to change the permission of that file and add the key to the SSH keys. That can be done by running:

```
$ chmod 0700 KEYPAIR.pem
```

```
$ ssh-add KEYPAIR.pem
```

To verify the key creation we run: `$ openstack key list`

Name	Fingerprint
KEYPAIR	01:b6:a3:ca:83:06:96:93:45:64:f6:d8:1f:68:77:f6

Figure 22. Command output

Now let's edit Devstack's default security group and allow SSH connection.

```
$ openstack security group rule create --protocol ssh --ingress default
```

As the image we downloaded has not a default password that we can use to log-in, we must configure it. For this to happen we will use cloud-config. Cloud-config files are special scripts designed to be run by the cloud-init process. These are generally used for initial configuration on the very first boot of a server. Launch a terminal on Host and run:

```
$ cat >user-data <<EOF
#cloud-config
password: ubuntu
chpasswd: { expire: False }
ssh_pwauth: True
EOF
```

We have created a file named user-data which declares that the password for the instance will be “ubuntu”. For more information about the cloud-init package refer to:

<https://help.ubuntu.com/community/CloudInit>

We are ready now to launch a server.

```
$ openstack server create --image xenial1 --flavor
ml.tiny --security-group default --user-data user-data --
key-name KEYPAIR --network internal testingserver
```

The output is similar to figure 23.

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	
OS-EXT-SRV-ATTR:host	None
OS-EXT-SRV-ATTR:hypervisor_hostname	None
OS-EXT-SRV-ATTR:instance_name	
OS-EXT-STS:power_state	NOSTATE
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	None
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	
adminPass	sGS8nAbnJ2YX
config_drive	
created	2020-05-05T08:43:19Z
flavor	ml.tiny (1)
hostId	
id	5ffbd260-3ecc-4607-8893-8187ab53763a
image	xenial1 (e9ad70c4-438e-483b-bff7-931b2181bad6)
key_name	KEYPAIR
name	testingserver
progress	0
project_id	11cc4c4cf9854776a6cc1245939a305d
properties	
security_groups	name='default'
status	BUILD
updated	2020-05-05T08:43:18Z
user_id	f2ec52a3b9bf4809a59a5d6b10213429
volumes_attached	

*Figure 23. Server creation output*

So the newly created server has now an IP from Openstack’s internal network. If we want to access the instance we must add a floating IP from the external network. Firstly, we must create a floating IP from the public network and then we must assign it to the running server.

```
$ openstack floating ip create public
```

The output contains an IP that belongs to the 172.24.4.0/24 network. We are going to use this IP at the next command so that we assign it to the server.

```
$ openstack server add floating ip <ip> testing server
```

Finally let’s try to SSH into the server from the Host.



```
$ ssh ubuntu@<ip>
```

Accept the fingerprint and use the password to SSH. That's it. The instance is now accessible not only from the dashboard, but from the Host too.

## 7. Packer

### 7.1 Introduction

Packer is an open source tool for creating identical machine images for multiple platforms from a single source configuration. Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms in parallel. A machine image is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines. Machine image formats change for each platform. Some examples include AMIs for EC2, VMDK/VMX files for VMware, OVF exports for VirtualBox, etc. [12]

Pre-baked machine images have a lot of advantages, but most have been unable to benefit from them because images have been too tedious to create and manage. There were either no existing tools to automate the creation of machine images or they had too high of a learning curve. The result is that, prior to Packer, creating machine images threatened the agility of operations teams, and therefore aren't used, despite the massive benefits. Packer changes all of this. Packer is easy to use and automates the creation of any type of machine image.

### 7.2 Advantages of using Packer

**Super fast infrastructure deployment:** Packer images allows the users to launch completely provisioned and configured machines in seconds, rather than several minutes or hours. This benefits not only production, but development as well, since development virtual machines can also be launched in seconds, without waiting for a typically much longer provisioning time.

**Multi-provider portability:** Because Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox. Each environment is running an identical machine image, giving ultimate portability.

**Improved stability:** Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they'll be caught early, rather than several minutes after a machine is launched.

**Greater testability:** After a machine image is built, that machine image can be quickly launched and smoke tested to verify that things appear to be working. If they are, we can be confident that any other machines launched from that image will function properly.

## 7.3 Packer terminology

There are a handful of terms used throughout the Packer documentation. This chapter documents all the terminology required to understand and use Packer. Packer uses templates. Templates are JSON files which define one or more builds by configuring the various components of Packer. Templates are given to Packer commands which will take the template and actually run the builds within it, producing any resulting machine images.

### 7.3.1 Artifacts

Artifacts are the results of a single build, and are usually a set of IDs or files to represent a machine image. Every builder produces a single artifact. As an example, in the case of the Amazon EC2 builder, the artifact is a set of AMI IDs (one per region). For the VMware builder, the artifact is a directory of files comprising the created virtual machine.

### 7.3.2 Builds and builders

Builds are a single task that eventually produces an image for a single platform. Multiple builds run in parallel. Builders are components of Packer that are able to create a machine image for a single platform. Builders read in some configuration and use that to run and generate a machine image. A builder is invoked as part of a build in order to create the actual resulting images. Example builders include VirtualBox, VMware, and Amazon EC2. Builders can be created and added to Packer in the form of plugins.

As we will use Openstack as our cloud provider, let's see how Packer builders are used with Openstack. The builder takes a source image, runs any provisioning necessary on the image after launching it, then creates a new reusable image. This reusable image can then be used as the foundation of new servers that are launched within OpenStack. The builder will create temporary keypairs that provide temporary access to the server while the image is being created. This simplifies configuration quite a bit. The builder does not manage images. Once it creates an image, it

is up to the user to use it or delete it. There are many configuration options available for the builder. Below are presented the required parameters for the builder file.

- **username** (string) - The username or id used to connect to the OpenStack service. If not specified, Packer will use the environment variable `OS_USERNAME` or `OS_USERID`, if set. This is not required if using access token or application credential instead of password, or if using `cloud.yaml`.
- **password** (string) - The password used to connect to the OpenStack service. If not specified, Packer will use the environment variables `OS_PASSWORD`, if set. This is not required if using access token or application credential instead of password, or if using `cloud.yaml`.
- **identity\_endpoint** (string) - The URL to the OpenStack Identity service. If not specified, Packer will use the environment variables `OS_AUTH_URL`, if set.
- **image\_name** (string) - The name of the resulting image
- **source\_image** (string) - The ID or full URL to the base image to use. This is the image that will be used to launch a new server and provision it. Unless the user specifies completely custom SSH settings, the source image must have cloud-init installed so that the keypair gets assigned properly.
- **source\_image\_name** (string) - The name of the base image to use. This is an alternative way of providing `source_image` and only either of them can be specified.
- **source\_image\_filter** (ImageFilter) - Filters used to populate filter options. Example:
- **flavor** (string) - The ID, name, or full URL for the desired flavor for the server to be created.

Here is a simple example of a builder's configuration that can be used in a JSON file to build a new image in Openstack environment. It will be explained in more detail in the *Templates* chapter.

```
{
  "type": "openstack",
  "identity_endpoint": "http://<devstack-ip>:5000/v3",
  "username": "admin",
  "password": "<admin password>",
  "ssh_username": "root",
  "image_name": "Example image",
  "source_image": "<image id>",
  "flavor": "m1.tiny"
}
```

### 7.3.3 Commands

Packer is controlled using a command-line interface. All interaction with Packer is done via the packer tool. Like many other command-line tools, the packer tool takes a subcommand to execute, and that subcommand may have additional options as well. Subcommands are executed with `packer SUBCOMMAND`, where "SUB-COMMAND" is the actual command to be executed. The subcommands are quite a few and are presented in detail below.

**packer build:** Takes a template and runs all the builds within it in order to generate a set of artifacts. The various builds specified within a template are executed in parallel, unless otherwise specified. And the artifacts that are created will be outputted at the end of the build. Example usage:

```
$ packer build example_template.json
```

**packer console:** Allows the user to experiment with Packer variable interpolations. He may access variables in the Packer config he called the console with, or provide variables when he calls console using the `-var` or `-var-file` command line options. Example usage:

```
$ packer console example_template.json
```

**packer fix:** Takes a template and finds backwards incompatible parts of it and brings it up to date so it can be used with the latest version of Packer. Example usage:

```
$ packer fix old.json > new.json
```

**packer inspect:** Takes a template and outputs the various components a template defines. This can help the user quickly learn about a template without having to dive into the JSON itself. The command will tell the user things like what variables a template accepts, the builders it defines, the provisioners it defines and the order they'll run, and more. Example usage:

```
$ packer inspect example_template.json
```

**packer validate:** It is used to validate the syntax and configuration of a template. The command will return a zero exit status on success, and a non-zero exit status on failure. Additionally, if a template doesn't validate, any error messages will be outputted.

```
$ packer validate example_template.json
```

### 7.3.4 Post-processors

The post-processor section within a template configures any post-processing that will be done to images built by the builders. Examples of post-processing would be compressing files, uploading artifacts, etc. Post-processors are optional. If no post-processors are defined within a template, then no post-processing will be done to the image. The resulting artifact of a build is just the image outputted by the builder. For each post-processor definition, Packer will take the result of each of the defined builders and send it through the post-processors. This means that if one post-processor and two builders are defined in a template, the post-processor will run twice (once for each builder), by default.

### 7.3.5 Provisioners

Within the template, the provisioners section contains an array of all the provisioners that Packer should use to install and configure software within running machines prior to turning them into machine images. Provisioners are optional. If no provisioners are defined within a template, then no software other than the defaults will be installed within the resulting machine images. This is not typical, however, since much of the value of Packer is to produce multiple identical images of pre-configured software. Below is a simple example. The first object transfers a file from the host system to the new image and the second object executes some commands on the shell.

```
{
  "type": "file",
  "source": "/usr/local/bin/examplescript.sh",
  "destination": "/home/ubuntu/script.sh",
}
{
  "type": "shell",
  "inline": [
    "sudo apt update && sudo apt upgrade -y"
  ]
}
```

## 7.4 Building a new image

In this example we are going to build a new image using Packer through a JSON file. We will use as base image the same image we used at subchapter 5.5 which is Ubuntu 16.04 Cloud version.

```

{
  "builders": [
    {
      "type": "openstack",
      "ssh_username": "ubuntu",
      "identity_endpoint": "http://150.140.186.115/identity",
      "image_name": "packerimage",
      "source_image": "e9ad70c4-438e-483b-bff7-931b2181bad6",
      "flavor": "m1.tiny",
      "networks": [
        "3e42296e-cdbf-4099-a4c4-3ea4df87535c"
      ],
      "use_floating_ip": true,
      "floating_ip_pool": "public"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "inline": [
        "sudo apt update && sudo apt upgrade -y"
      ]
    }
  ]
}

```

In the builders part, we declared that the Cloud type is Openstack, the username that will be used by Packer to SSH into the instance is “ubuntu” and we have also provided a name for the new image, the ID of the source image and some more information about flavor, networking and floating IP’s. In the provisioners part there is a simple configuration that Packer needs to do to the source image. It just updates and upgrades it so that the new image will be fully up-to-date. But we can obviously run any command through the shell.

## 8. The project

### 8.1 Work analysis

The purpose of the thesis is to provide L2 connectivity between two or more clouds using pure software virtual switches and tunnels. We are going to use two clouds but the project can be used for more. In this chapter we are going to sum up all the previous chapters to end up in the final result.

The setup is composed of two PC’s, each one running Devstack on Ubuntu 18.04 LTS. So Openstack is going to be the cloud provider. Devstack was installed as described in chapter 6.4. Moreover, there is one VPN server running at an external IP.

The purpose of the VPN is to provide an overlay connection between the instances of each cloud so that they seem to belong in the same network.

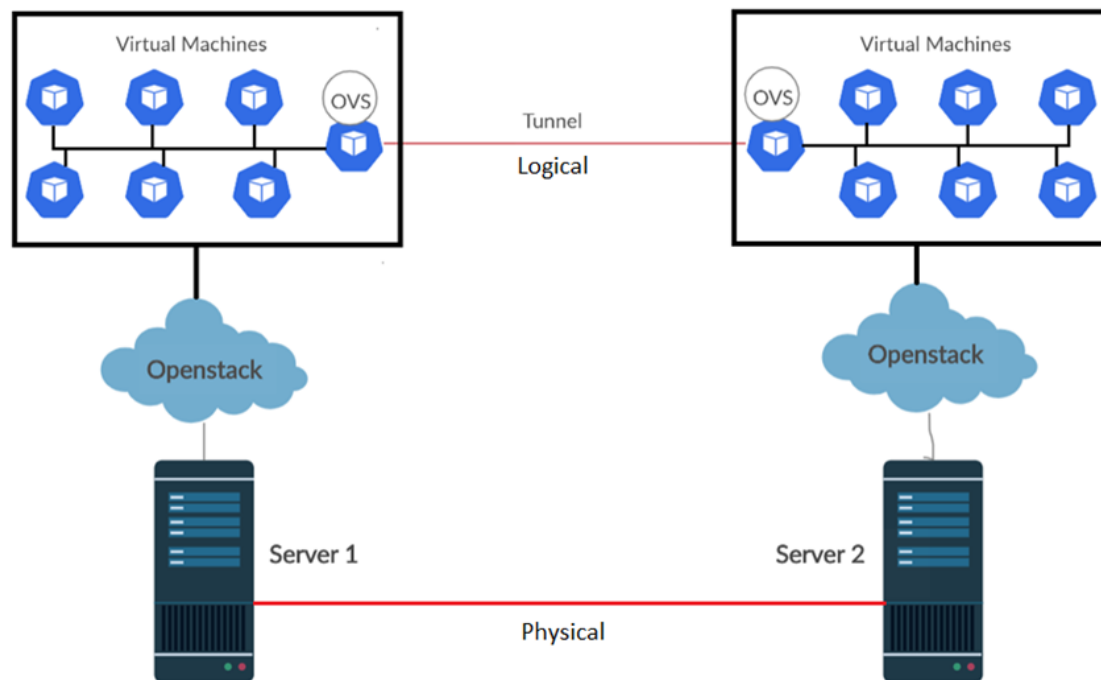


Figure 24. The setup

Open vSwitch is deployed in one VM in each Devstack environment. For this to be done automatically we have created a custom image using Packer. Practically, there are two services that are starting each time the custom image is booted. Packer is also used to create a second custom image (a simpler one), which is an HTTP server listening on port 8000. We will deploy one HTTP server on every internal network of Devstack to test if HTTP traffic can be moved between the clouds through the tunnels.

All the code files can be found on Github following this link: <https://github.com/thimiosgr/CloudConnectivity>. There are two branches on this project. The first one is called “master” and it is used if someone wants to configure a lot of Devstack environments remotely. There is one file called *manager.sh* under the ‘CloudConnectivity/management’ directory which is a shell script that runs everything on each cloud just by passing it the IP’s of the clouds. The second branch is called “simple” and it just contains the files that are needed for the user to execute them directly on the host that runs the cloud. The script that does the job is called *cloudconnectivity.sh* and is located under the ‘CloudConnectivity’ directory.

So let’s get into something more practical and follow the procedure from the beginning. We suppose that the git package from the previous section is cloned on our PC so we will use the “master” branch. Access the ‘CloudConnectivity’ directory that

was created and list all the files that it contains. The *'readme.md'* file just contains some information about this project. The *'cloud-configuration'* directory contains a file for the initial configuration on some instances that will be booted on the cloud. It sets them a custom password so that we can SSH into them and show what will be achieved in the end of this chapter.

#### Cloud configuration file:

```
#cloud-config
password: ubuntu
chpasswd: { expire: False }
ssh_pwauth: True
```

The *'credentials'* directory contains each cloud's credentials and also some files that pass variables to the main script such as the source image that Packer will use, the VPN server's IP etc. The variables are set to default and someone can easily change these and pass their own.

#### Variables file:

```
export OPENSTACK_IP="150.140.186.115"
export VPN_IP="2.87.164.197"
export FILENAME="client1"
export PASSWD="ops"
export PUBLIC_NETWORK="public"
export IMAGE_NAME="xenial1"
```

#### Cloud credentials file:

```
#!/usr/bin/env bash
export OS_AUTH_URL=http://150.140.186.115/identity
export OS_PROJECT_ID=a9211a8546104bbebfcd2a2c077bcbd6
export OS_PROJECT_NAME="admin"
export OS_USER_DOMAIN_NAME="Default"
if [ -z "$OS_USER_DOMAIN_NAME" ]; then unset OS_USER_DOMAIN_NAME; fi
export OS_PROJECT_DOMAIN_ID="default"
if [ -
z "$OS_PROJECT_DOMAIN_ID" ]; then unset OS_PROJECT_DOMAIN_ID; fi
unset OS_TENANT_ID
unset OS_TENANT_NAME
export OS_USERNAME="admin"
export OS_PASSWORD=$1
export OS_REGION_NAME="RegionOne"
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
```



The `'exec'` directory contains a script that is the main script of the package and runs almost everything on the clouds. The `services` directory contains all the services that Packer is passing to the custom images and that must be run on boot. Finally, the `templates` directory is composed of the templates (JSON files) that describe to Packer how to create the custom images. The “simple” branch slightly differs but almost all the files are the same.



Figure 25. "simple" branch use-case

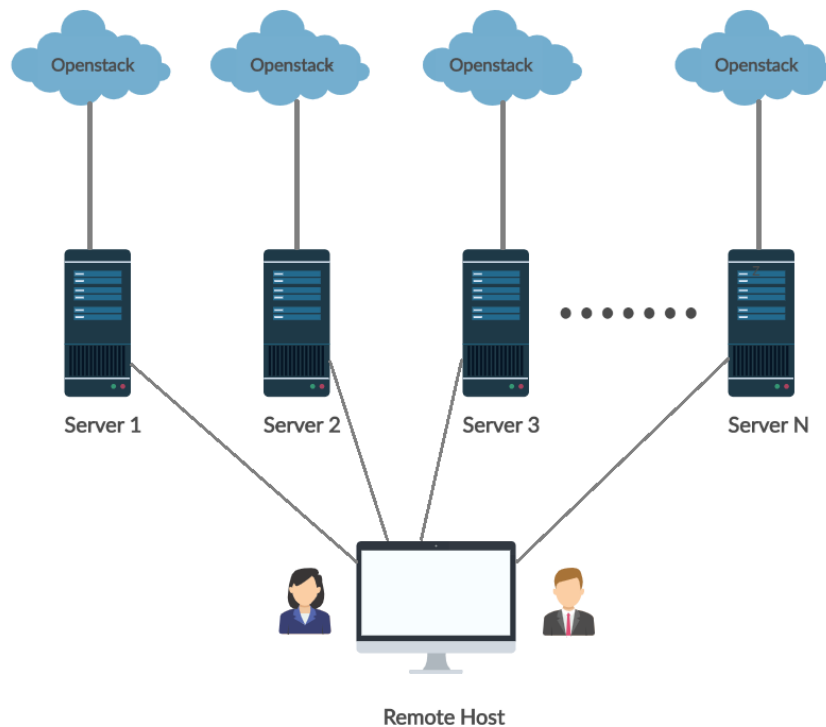


Figure 26. "master" branch use-case

One command is going to do all the job for us.

```
$ ./CloudConnectivity/management/manager.sh -ips CLOUD-IP
```

The cloud IP's must be passed to the script separated by commas. After the IP's are checked, the package is installed and ran in each cloud. The main script called "cloud-connectivity.sh" is creating almost everything. Firstly it checks if some required packages such as Packer are installed. If not, it downloads and installs them.

```
# Checking if Packer is installed.
if ! [ $(command -v packer) ] ; then
    echo "Installing Packer..."
    export VER="1.5.5"
    wget https://releases.hashicorp.com/packer/${VER}/packer_${VER}_linux_amd64.zip
    unzip packer_${VER}_linux_amd64.zip
    sudo mv packer /usr/local/bin
    printf "\033[0;32mInstalled Packer.\033[0m\n"
else
    echo "Packer is already installed. Not installing."
fi

# Checking if command-line JSON processor jq is installed.
if ! dpkg -s jq >/dev/null 2>&1; then
    echo "Installing jq."
    sudo apt-get install jq -y
    printf "\033[0;32mInstalled jq.\033[0m\n"
else
    echo "Jq is already installed. Not installing."
fi

# Checking if moreutils package is installed.
if ! dpkg -s moreutils >/dev/null 2>&1; then
    echo "Installing moreutils..."
    sudo apt-get install moreutils -y
    printf "\033[0;32mInstalled moreutils\033[0m\n"
else
    echo "Package moreutils is already installed. Not installing."
fi
```

Then it creates the full network topology that is going to be needed. It is consisted of one primary network that has access to the Internet and five completely isolated networks. On the primary network we are going to run the OVS machine and on the other's some example instances.

```

# Openstack networks configuration
PRIMARY_NETWORK_ID=$(openstack network create primary_network --
provider-network-type vxlan | grep " id " | awk '{print $4}' -)
INTERNAL_NETWORK1_ID=$(openstack network create internal_network1 --
provider-network-type vxlan --disable-port-
security | grep " id " | awk '{print $4}' -)
INTERNAL_NETWORK2_ID=$(openstack network create internal_network2 --
provider-network-type vxlan --disable-port-
security | grep " id " | awk '{print $4}' -)
INTERNAL_NETWORK3_ID=$(openstack network create internal_network3 --
provider-network-type vxlan --disable-port-
security | grep " id " | awk '{print $4}' -)
INTERNAL_NETWORK4_ID=$(openstack network create internal_network4 --
provider-network-type vxlan --disable-port-
security | grep " id " | awk '{print $4}' -)
INTERNAL_NETWORK5_ID=$(openstack network create internal_network5 --
provider-network-type vxlan --disable-port-
security | grep " id " | awk '{print $4}' -)
ROUTER_ID=$(openstack router create ROUTER | grep " id " | awk '{pri
nt $4}' -)
PRIMARY_NETWORK_SUBNET_ID=$(openstack subnet create primary_network_
subnet --network $PRIMARY_NETWORK_ID --subnet-range 192.168.0.0/24 -
-dhcp --dns-nameserver 8.8.8.8 --
gateway 192.168.0.1 | grep " id " | awk '{print $4}' -)

openstack subnet create internal_network1_subnet--
network $INTERNAL_NETWORK1_ID --subnet-range 192.168.1.0/24 --dhcp -
-gateway none > /dev/null 2>&1
openstack subnet create internal_network2_subnet --
network $INTERNAL_NETWORK2_ID --subnet-range 192.168.2.0/24 --dhcp -
-gateway none > /dev/null 2>&1
openstack subnet create internal_network3_subnet --
network $INTERNAL_NETWORK3_ID --subnet-range 192.168.3.0/24 --dhcp -
-gateway none > /dev/null 2>&1
openstack subnet create internal_network4_subnet --
network $INTERNAL_NETWORK4_ID --subnet-range 192.168.4.0/24 --dhcp -
-gateway none > /dev/null 2>&1
openstack subnet create internal_network5_subnet --
network $INTERNAL_NETWORK5_ID --subnet-range 192.168.5.0/24 --dhcp -
-gateway none > /dev/null 2>&1
openstack router set $ROUTER_ID -external
gateway $PUBLIC_NETWORK > /dev/null 2>&1
openstack router add subnet $ROUTER_ID $PRIMARY_NETWORK_SUBNET_ID >
/dev/null 2>&1

```

After that it edits the JSON files according to the user's preferences so that the images can be built correctly. When Packer is finished, then an instance running Open

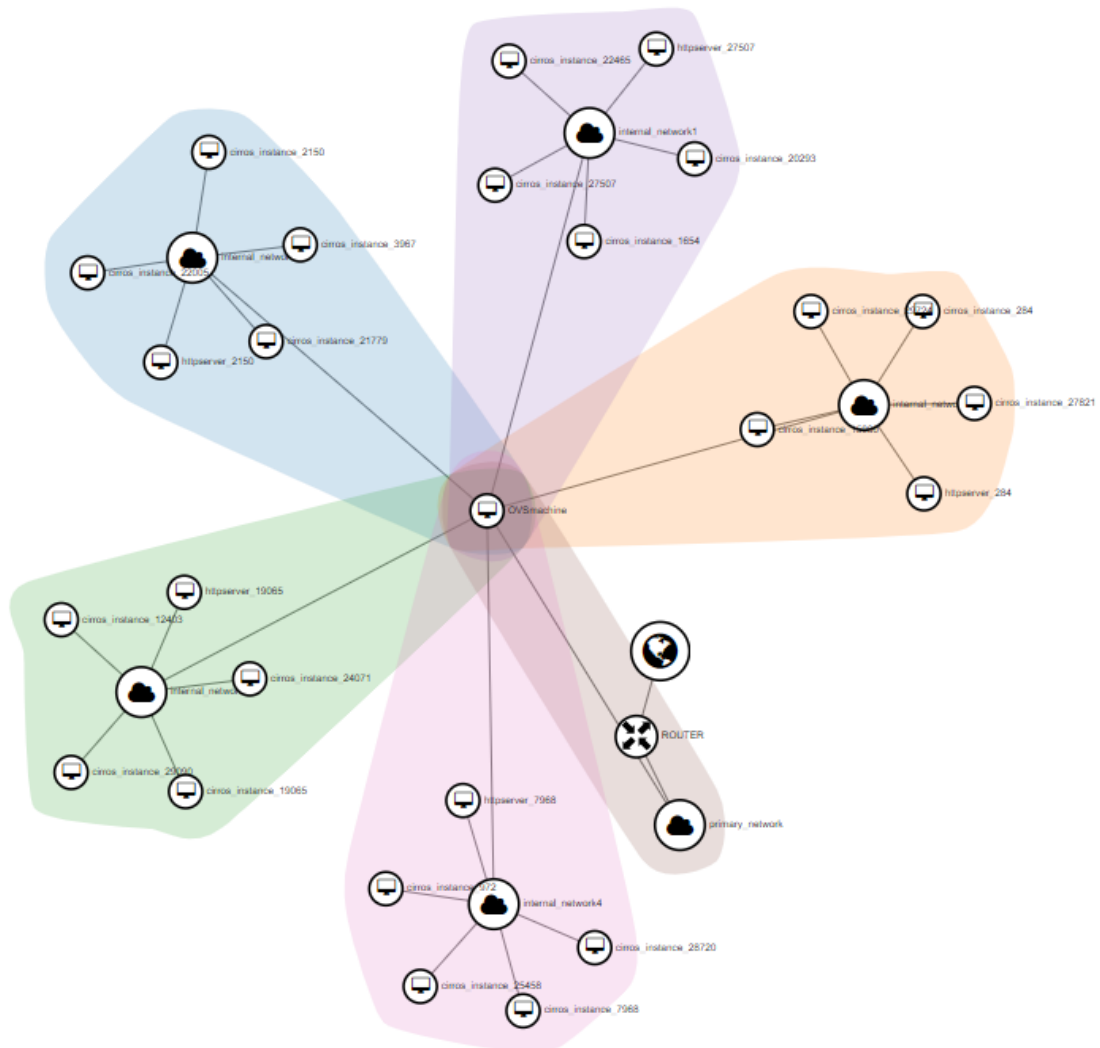
vSwitch is booted. The instance is automatically connected to the VPN server and creates tunnels to communicate with the respective instance of the second cloud. Furthermore, five instances are booted on each internal network. Four of them are using a “cirros” image which is a simple image just for cloud testing. The fifth one is an httpserver using the second custom image that Packer created so that we pass some HTTP traffic to the other side.

```
echo "Building images... This might take some time, depending on your hardware and your Internet connection."
packer build ${THE_PATH}/templates/ovsimage.json > /dev/null 2>&1
packer build ${THE_PATH}/templates/httpserverimage.json > /dev/null 2>&1
printf "\033[0;32mCreated images: OVSimage,SimpleHTTPserver.\n\033[0mRun 'openstack image list' for confirmation.\n"

printf "\nCreating server for Open vSwitch..."
SERVER_ID=$(openstack server create --image OVSimage --
flavor m1.heat_int --key-name KEYPAIR --user-data ${THE_PATH}/cloud-
configuration/user-data.txt --network ${PRIMARY_NETWORK_ID} --
network ${INTERNAL_NETWORK1_ID} --network ${INTERNAL_NETWORK2_ID} --
network ${INTERNAL_NETWORK3_ID} --network ${INTERNAL_NETWORK4_ID} --
network ${INTERNAL_NETWORK5_ID} OVSmachine | grep " id " | awk '{pri
nt $4}' -)
printf "\033[0;32m Done\033[0m\n"

printf "Creating instances on each internal network..."
COUNTER=0
while [ "$COUNTER" -lt "${#OPENSTACK_ARR[@]}" ];
do
    for i in $(seq 1 4)
    do
        RANDOM_INTEGER=$(echo $((1 + RANDOM)))
        openstack server create --image cirros-0.4.0-x86_64-disk --
flavor m1.nano --
network ${OPENSTACK_ARR[COUNTER]} "cirros_instance_${RANDOM_INTEGER}
" > /dev/null 2>&1
    done
    openstack server create --image SimpleHTTPserver --
flavor m1.heat_int --
network ${OPENSTACK_ARR[COUNTER]} "httpserver_${RANDOM_INTEGER}" > /
dev/null 2>&1
    COUNTER=$((COUNTER+1))
done
printf "\033[0;32m Done\033[0m\n"
```

By accessing the Dashboard we can see everything that was created. The network topology looks like *Figure 27*.



*Figure 27. Network topology*

We can clearly see the OVS machine in the middle of the figure. It is connected on all the networks so that it can be used as a gateway for the internal networks.

OVSmachine	OVSimage	<b>internal_network4</b>	192.168.4.7
		<b>internal_network5</b>	192.168.5.92
		<b>internal_network2</b>	192.168.2.133
		<b>internal_network3</b>	192.168.3.46
		<b>internal_network1</b>	192.168.1.91
		<b>primary_network</b>	192.168.0.149, 172.24.4.162

*Figure 28. OVS machine*

The router has two attached interfaces, one on the provider network and one on the primary network so that the OVS machine can have access to the Internet. At last, some more instances have been created on each internal network.

The networks that were created as explained in the previous paragraph are the following:

<a href="#">internal_network2</a>	<b>internal_network2_subnet</b> 192.168.2.0/24
<a href="#">internal_network3</a>	<b>internal_network3_subnet</b> 192.168.3.0/24
<a href="#">public</a>	<b>public-subnet</b> 172.24.4.0/24 <b>ipv6-public-subnet</b> 2001:db8::/64
<a href="#">internal_network5</a>	<b>internal_network5_subnet</b> 192.168.5.0/24
<a href="#">net1</a>	<b>net1</b> 192.168.100.0/24
<a href="#">internal_network1</a>	<b>internal_network1_subnet</b> 192.168.1.0/24
<a href="#">primary_network</a>	<b>primary_network_subnet</b> 192.168.0.0/24
<a href="#">internal_network4</a>	<b>internal_network4_subnet</b> 192.168.4.0/24

*Figure 29. Networks*

Someone can easily notice that in *Figure 28* is clearly shown that the OVS machine is running an image called OVSImage. But that is not a known image to us, right? As you can easily imagine, this is one of the images that was created by Packer.

<a href="#">OVSImage</a>	Image	Active	Private	No	QCOW2	1.18 GB
<a href="#">SimpleHTTPserver</a>	Image	Active	Private	No	QCOW2	1.21 GB

*Figure 30. Packer images*

The second image is the one we referred to earlier and is just an HTTP server. So, let's see in more detail how these images were built. The user is passing to the script the source image on which the custom image will be created from and also the ID of a network that has access to the Internet. He must also pass the flavor that is going to be used. All these parameters are passed into the JSON files that will be used by Packer. As already explained these templates are located under the “*templates*” directory.

The first template called “*ovsimage.json*” is apparently used for the Open vSwitch image. It passes four files that are used for services starting at boot time. It also

installs Open vSwitch and does some more configuration. Let's have a closer look at the services so that they can be more understandable. The first service called "*networkconf.service*" is scheduled to run right after the `network.target` has been reached and is running a script located in the "`/usr/local/bin`" directory. That script is called "*networkconfiguration.sh*" and it just raises the interfaces that are attached on each internal network. After that it reboots the system so that the changes are applied. All the services for this image can be found under the "*services/ovs-machine*" directory. Example for one internal network:

```
#!/bin/bash

FILE=/home/ubuntu/temp

if ! [[ -f "$FILE" ]]; then

    sudo bash -c "echo auto ens4 >> /etc/network/interfaces.d/50-cloud-init.cfg"
    sudo bash -c "echo iface ens4 inet dhcp >> /etc/network/interfaces.d/50-cloud-init.cfg"
    touch $FILE
    reboot

fi
```

#### Service file:

```
[Unit]

After=network.target

[Service]

ExecStart=/usr/local/bin/networkconfiguration.sh

[Install]

WantedBy=default.target
```

The second service is called "*tunneling.service*" and is also scheduled to run after the network has been configured. It executes another script also located under the "`/usr/local/bin`" directory. It is called "*tunnelcreator.sh*" and it's job is to fetch the VPN configuration files, to create all the Open vSwitch switches and tunnels and to connect to the VPN. Example for one internal network:

```

#!/bin/bash

VPN_IP=91.140.33.10
USERNAME="client1"
sudo mkdir /home/ubuntu/${USERNAME}
CHECK_FILE=/home/ubuntu/${USERNAME}/ca.crt
FILE=/home/ubuntu/temp

if [[ -f "$FILE" ]]; then
    if ! [ -f "${CHECK_FILE}" ]; then

        # IP route configuration
        ip route del default
        ip route add default via 192.168.0.1

        # Fetching the VPN files
        wget http://${VPN_IP}/${USERNAME}/ca.crt -
P /home/ubuntu/${USERNAME}/
        wget http://${VPN_IP}/${USERNAME}/${USERNAME}.crt -
P /home/ubuntu/${USERNAME}/
        wget http://${VPN_IP}/${USERNAME}/${USERNAME}.key -
P /home/ubuntu/${USERNAME}/
        wget http://${VPN_IP}/${USERNAME}/${USERNAME}.ovpn -
P /home/ubuntu/${USERNAME}/
        wget http://${VPN_IP}/${USERNAME}/vtep.sh -
P /home/ubuntu/${USERNAME}/
        VTEP_IP=$(head -n1 /home/ubuntu/${USERNAME}/vtep.sh)

        # Finding the internal network's IP's
        IN_NET_IP1=$(ip a | awk '/ens4/{getline;getline; print}' | awk
-F/ '{print $1}' - | awk '{print $2}' -)/24"

        # Hashing the IP's
        IP1_MD5=$(md5sum <<<"${IN_NET_IP1}" | cut -c 1-10)

        # OpenvSwitch configuration
        sudo ovs-vsctl add-br "br${IP1_MD5}"
        ip addr flush dev ens4
        sudo ovs-vsctl add-port "br${IP1_MD5}" ens4
        sudo ovs-vsctl add-port "br${IP1_MD5}" "vxlan${IP1_MD5}" -
- set interface "vxlan${IP1_MD5}" type=vxlan options:remote_ip=${VTEP_I
P} options:key=2000
        ip addr add ${IN_NET_IP1} dev "br${IP1_MD5}"
        ip link set "br${IP1_MD5}" up
        sudo openvpn /home/ubuntu/${USERNAME}/client1.ovpn
    fi
fi

```



Service file:

```
[Unit]
After=network.target

[Service]
ExecStart=/usr/local/bin/tunnelcreator.sh

[Install]
WantedBy=default.target
```

The second template called “*httpserver.json*” is used for the second image. It passes the files of one single service which opens the port 8000 for HTTP traffic. The only configuration is a simple update of the system and the installation of the python package. The service file runs a very simple script called “*httpserver.sh*” that can be found in the “*/usr/local/bin*” directory. Example:

```
#!/bin/bash

python -m SimpleHTTPServer
```

Service file:

```
[Unit]
After=network.target

[Service]
ExecStart=/usr/local/bin/httpserver.sh

[Install]
WantedBy=default.target
```

Now let’s SSH into the one OVS machine and see what exactly has been done. To do so we must associate a floating IP from the provider network to this instance. As we explained earlier there is one service that creates the tunnels that end up at the other cloud. By running the appropriate Open vSwitch command we can see an output similar to the one below. The tunnels and switches are created correctly with each one having a unique name and key based on an Ubuntu built-in hash function called “*md5sum*”.

```

ubuntu@ovsmachine:~$ sudo ovs-vsctl show
Bridge "brbca8c1369d"
  Port "vxlanbca8c1369d"
    Interface "vxlanbca8c1369d"
      type: vxlan
      options: {key="2002", remote_ip="10.8.0.40"}
  Port "ens6"
    Interface "ens6"
  Port "brbca8c1369d"
    Interface "brbca8c1369d"
      type: internal
Bridge "brdc603aaf57"
  Port "ens7"
    Interface "ens7"
  Port "brdc603aaf57"
    Interface "brdc603aaf57"
      type: internal
  Port "vxlandc603aaf57"
    Interface "vxlandc603aaf57"
      type: vxlan
      options: {key="2003", remote_ip="10.8.0.40"}
Bridge "br54acaeb056"
  Port "vxlan54acaeb056"
    Interface "vxlan54acaeb056"
      type: vxlan
      options: {key="2001", remote_ip="10.8.0.40"}
  Port "ens5"
    Interface "ens5"
  Port "br54acaeb056"
    Interface "br54acaeb056"
      type: internal
Bridge "brced47a1b26"
  Port "vxlanced47a1b26"
    Interface "vxlanced47a1b26"
      type: vxlan
      options: {key="2000", remote_ip="10.8.0.40"}
  Port "brced47a1b26"
    Interface "brced47a1b26"
      type: internal
  Port "ens4"
    Interface "ens4"
ovs_version: "2.5.5"

```

By running `$ ifconfig` we can spot the VPN interface. It has an IP that belongs to no one of the Openstack networks. Each instance running the OVSimage gets to have an interface like that because it automatically connects to the VPN server.

```
ubuntu@ovsmachine:~$ ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.8.0.20  P-t-P:255.255.255.255  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:6453 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5997 errors:0 dropped:457 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:1286304 (1.2 MB)  TX bytes:1225444 (1.2 MB)
```

The VPN server that we are using is configured so that the IP's given to the instances are static. So the OVS machine running in the first cloud is owning the 10.8.0.20 IP address and the OVS machine running in the second cloud is owning the 10.8.0.40 IP address. In that way we make our life easier when it comes to configuring the remote IP section in the Open vSwitch tunnels.

The networking section of the one OVS machine is displayed in the Figure 31. The second OVS machine has the following IP's:

		<b>primary_network</b> 192.168.0.46, 172.24.4.96
		<b>internal_network3</b> 192.168.3.220
		<b>internal_network4</b> 192.168.4.151
		<b>internal_network1</b> 192.168.1.44
		<b>internal_network2</b> 192.168.2.193
		<b>internal_network5</b> 192.168.5.2
<b>OVSmachine</b>	<b>OVSimage</b>	

*Figure 31. OVS machine networking*

The connection through VPN has been established. So let's try a simple ping test from the first machine to the second using the IP's of "*internal\_network2*". Simultaneously we capture the incoming ICMP packets on the other machine.

```
ubuntu@ovsmachine:~$ ping 192.168.2.193
PING 192.168.2.193 (192.168.2.193) 56(84) bytes of data.
64 bytes from 192.168.2.193: icmp_seq=1 ttl=64 time=60.9 ms
64 bytes from 192.168.2.193: icmp_seq=2 ttl=64 time=60.7 ms
64 bytes from 192.168.2.193: icmp_seq=3 ttl=64 time=59.5 ms
64 bytes from 192.168.2.193: icmp_seq=4 ttl=64 time=77.9 ms
^C
--- 192.168.2.193 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 59.520/64.791/77.980/7.637 ms
```

*Figure 32. ICMP packets going to OVS machine of the 2<sup>nd</sup> Openstack*

```

ubuntu@ovsmachine:~$ sudo tcpdump -i any icmp
sudo: unable to resolve host ovsmachine
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
11:46:30.550955 IP 192.168.2.133 > 192.168.2.193: ICMP echo request, id 2640, seq 1, length 64
11:46:30.551352 IP 192.168.2.133 > 192.168.2.193: ICMP echo request, id 2640, seq 1, length 64
11:46:30.551369 IP 192.168.2.193 > 192.168.2.133: ICMP echo reply, id 2640, seq 1, length 64
11:46:30.551431 IP 192.168.2.193 > 192.168.2.133: ICMP echo reply, id 2640, seq 1, length 64
11:46:31.552291 IP 192.168.2.133 > 192.168.2.193: ICMP echo request, id 2640, seq 2, length 64
11:46:31.552305 IP 192.168.2.133 > 192.168.2.193: ICMP echo request, id 2640, seq 2, length 64
11:46:31.552314 IP 192.168.2.193 > 192.168.2.133: ICMP echo reply, id 2640, seq 2, length 64
11:46:31.552316 IP 192.168.2.193 > 192.168.2.133: ICMP echo reply, id 2640, seq 2, length 64
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel

```

*Figure 33. ICMP packets captured at the OVS machine of the 2<sup>nd</sup> Openstack*

This ping will work with any IP addresses of any network. Of course any instance running on any internal network of the first cloud can communicate with any instance running on the same network of the second cloud and vice versa. As a second example we will try to get a file from an HTTP server running on the second cloud and having been attached to the “*internal\_network5*”.

httpserver_16777	SimpleHTTPserver	192.168.5.217
------------------	------------------	---------------

*Figure 34. HTTP server*

The server is displayed in the above figure. We log in a random cirros instance of the first cloud which is running on the respective “*internal\_network5*” and try to get a file. As displayed in Figure 36, our attempt is successful so the connectivity between the clouds has been achieved.

cirros_instance_24343	cirros-0.4.0-x86_64-disk	192.168.5.224
-----------------------	--------------------------	---------------

*Figure 35. Cirros instance*

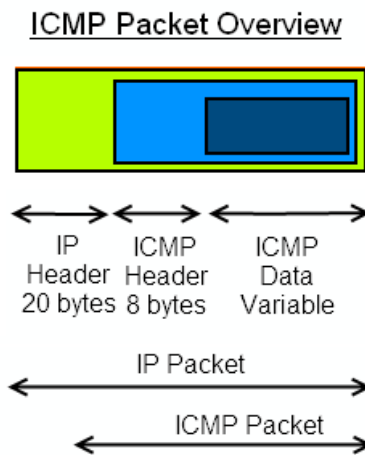
```

$ ls
$ wget http://192.168.5.217:8000
Connecting to 192.168.5.217:8000 (192.168.5.217:8000)
index.html 100% |*****| 970 0:00:00 ETA
$ ls
index.html

```

*Figure 36. Cirros instance console*

Finally we must test the MSS (Maximum Segment Size) which is the size of the largest segment that can be transferred between the clouds. The VXLAN packet format looks like that in Figure 7. The original ICMP packet format is displayed in Figure 37. So, let's do the math.



*Figure 37. ICMP packet overview*

The MTU path equals to 1500 bytes. The VXLAN protocol uses UDP as its underlying transport protocol which inserts a header of 8 bytes. The original ICMP packet has an ICMP header of 8 bytes and an IP header of 20 bytes. The final packet contains an outer IP header of 20 bytes and an outer MAC header of 14 bytes. So the MSS drops to 1422 bytes.

<u>Packet part</u>	<u>Number of bytes</u>
ICMP HEADER	8
ORIGINAL IP HEADER	20
VXLAN HEADER	8
OUTER UDP HEADER	8
OUTER IP HEADER	20
OUTER MAC HEADER	14
PAYLOAD	$1500 - 8 - 20 - 8 - 8 - 20 - 14 = \mathbf{1422}$

*Table 2. Maximum Segment Size*

## 8.2 Future Work

The limitation of the public IP's lead us to the use of a VPN server. Having had some more public IP's on our disposal or direct access to a public network there would be no need for a VPN server.

There are some things that can be implemented on this thesis to extend it's functionality. For example, someone could use an SND controller (like ONOS or OpenDayLight) to automatically provision the tunnels on the OVS switches. Moreover, the whole topology (OVS, SDN control, etc.) can be orchestrated over Kubernetes which is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. OVS may be running locally on worker or as a container itself but ideally everything could be packaged under a custom Central Registration Depository (CRD).

## FIGURE LIST

FIGURE 1. THE BENEFITS OF CLOUD COMPUTING .....	5
FIGURE 2. INFRASTRUCTURE AS A SERVICE .....	6
FIGURE 3. PLATFORM AS A SERVICE .....	7
FIGURE 4. SERVICE AS A SERVICE .....	8
FIGURE 5. DEPLOYMENT MODELS. SOURCE: <a href="https://medium.com/">HTTPS://MEDIUM.COM/</a> .....	11
FIGURE 6. GRE PACKET HEADER STRUCTURE .....	12
FIGURE 7. VXLAN PACKET FORMAT .....	13
FIGURE 8. GENEVE PACKET FORMAT .....	13
FIGURE 9. IPSEC IN AH TRANSPORT AND TUNNEL MODE .....	15
FIGURE 10. IPSEC IN ESP TRANSPORT AND TUNNEL MODE .....	15
FIGURE 11. EXAMPLE SETUP .....	16
FIGURE 12. VM1 – “OVS-VSCTL SHOW” COMMAND OUTPUT .....	18
FIGURE 13. VM2 – “OVS-VSCTL SHOW” COMMAND OUTPUT .....	18
FIGURE 14. VM1 – “PING” COMMAND OUTPUT .....	18
FIGURE 15. VM2 – “TCPDUMP” COMMAND OUTPUT .....	19
FIGURE 16. VPN PROTOCOLS COMPARISON. SOURCE: <a href="https://www.vpnmentor.com/blog/ultimate-guide-to-vpn-tunneling/">HTTPS://WWW.VPNMENTOR.COM/BLOG/ULTIMATE-GUIDE-TO-VPN-TUNNELING/</a> .....	23
FIGURE 17. OPEN VSWITCH DEPLOYED AS A CROSS-SERVER VIRTUAL NETWORK SWITCH, DISTRIBUTED ACROSS MULTIPLE PHYSICAL SERVERS .....	28
FIGURE 18. OPENSTACK .....	33
FIGURE 19. OPENSTACK DASHBOARD .....	39
FIGURE 20. OPENSTACK ARCHITECTURE .....	40
FIGURE 21. COMMAND OUTPUT .....	49
FIGURE 22. COMMAND OUTPUT .....	49
FIGURE 23. SERVER CREATION OUTPUT .....	50
FIGURE 24. THE SETUP .....	57
FIGURE 25. "SIMPLE" BRANCH USE-CASE .....	59
FIGURE 26. “MASTER” BRANCH USE-CASE .....	59
FIGURE 27. NETWORK TOPOLOGY .....	63
FIGURE 28. OVS MACHINE .....	63
FIGURE 29. NETWORKS .....	64
FIGURE 30. PACKER IMAGES .....	64
FIGURE 31. OVS MACHINE NETWORKING .....	69
FIGURE 32. ICMP PACKETS GOING TO OVS MACHINE OF THE 2 <sup>ND</sup> OPENSTACK .....	69
FIGURE 33. ICMP PACKETS CAPTURED AT THE OVS MACHINE OF THE 2 <sup>ND</sup> OPENSTACK .....	70
FIGURE 34. HTTP SERVER .....	70
FIGURE 35. CIRROS INSTANCE .....	70
FIGURE 36. CIRROS INSTANCE CONSOLE .....	70
FIGURE 37. ICMP PACKET OVERVIEW .....	71

## REFERENCES

- [1] James Kurose, Keith Ross. (2013). *Computer Networking - A Top-Down Approach 6th Edition*. New Jersey: Addison-Wesley.
- [2] Eric F Crist, Jan Just Keijser. (2015). *Mastering OpenVPN*. Packt Publishing.
- [3] George Lagogiannis, Aris Aleksopoulos. (2016). *Telecommunications and Computer Networking 10th Edition*. Athens: Gialos.
- [4] Gerardus Blokdijk . (2019). *IPsec VPN A Complete Guide*.
- [5] Ani Miteva. (2018, March 21). *Cloud Worldwide Services*. Retrieved from [www.cloudworldwideservices.com/en/cloud-deployment-models-differences/](http://www.cloudworldwideservices.com/en/cloud-deployment-models-differences/)
- [6] Friedl, Steve. (2005, August 24). *Unixwiz*. Retrieved from [www.unixwiz.net/techtips/iguide-ipsec.html?fbclid=IwAR3CEZBXv7ajNp5bfYocXLCG2upSdDrV-9LysLwt-aLJzgo8V\\_uJgvLJkpl](http://www.unixwiz.net/techtips/iguide-ipsec.html?fbclid=IwAR3CEZBXv7ajNp5bfYocXLCG2upSdDrV-9LysLwt-aLJzgo8V_uJgvLJkpl)
- [7] Juniper Networks. (2019, July 11). *Juniper Networks*. Retrieved from [www.juniper.net: www.juniper.net/documentation/en\\_US/junos/topics/topic-map/sdn-vxlan.html](http://www.juniper.net: www.juniper.net/documentation/en_US/junos/topics/topic-map/sdn-vxlan.html)
- [8] Microsoft. (n.d.). *Microsoft Azure*. Retrieved from [azure.microsoft.com/en-in/overview/what-is-cloud-computing/#cloud-computing-models](http://azure.microsoft.com/en-in/overview/what-is-cloud-computing/#cloud-computing-models)
- [9] Open vSwitch. (n.d.). Retrieved from [www.openvswitch.org](http://www.openvswitch.org)
- [10] Openstack. (n.d.). *Openstack*. Retrieved from [www.openstack.org/software/project-navigator/openstack-components#openstack-services](http://www.openstack.org/software/project-navigator/openstack-components#openstack-services)
- [11] Openstack. (n.d.). *Openstack*. Retrieved from [docs.openstack.org/python-openstackclient/pike/cli/command-list.html](http://docs.openstack.org/python-openstackclient/pike/cli/command-list.html), [www.openstack.org/software/project-navigator/openstack-components#openstack-services](http://www.openstack.org/software/project-navigator/openstack-components#openstack-services)
- [12] Packer. (n.d.). *Packer*. Retrieved from [www.packer.io](http://www.packer.io)
- [13] Phillips Gavin. (2017, October 12). *Makeuseof*. Retrieved from [www.makeuseof.com/tag/major-vpn-protocols-explained/](http://www.makeuseof.com/tag/major-vpn-protocols-explained/)
- [14] Ray J Rafaels. (2015). *Cloud Computing: From Beginning to End*.
- [15] *The Random Security Guy*. (n.d.). Retrieved from [therandomsecurityguy.com/openvswitch-cheat-sheet/](http://therandomsecurityguy.com/openvswitch-cheat-sheet/)
- [16] Thomas Erl. (2014). *Cloud Computing: Concepts, Technology & Architecture*. Service Tech Press.
- [17] Tim Mocan. (2019, February 1). *Cactus VPN*. Retrieved from [www.cactusvpn.com/beginners-guide-to-vpn/what-is-sstp/](http://www.cactusvpn.com/beginners-guide-to-vpn/what-is-sstp/), [www.cactusvpn.com/beginners-guide-to-vpn/what-is-pptp/](http://www.cactusvpn.com/beginners-guide-to-vpn/what-is-pptp/), [www.cactusvpn.com/beginners-guide-to-vpn/what-is-l2tp/](http://www.cactusvpn.com/beginners-guide-to-vpn/what-is-l2tp/), [www.cactusvpn.com/beginners-guide-to-vpn/what-is-ikev2/](http://www.cactusvpn.com/beginners-guide-to-vpn/what-is-ikev2/)



- [18]Wikipedia. (2020, May 20). *Wikipedia*. Retrieved from [en.wikipedia.org/wiki/Tunneling\\_protocol#Common\\_tunneling\\_protocols](https://en.wikipedia.org/wiki/Tunneling_protocol#Common_tunneling_protocols)
- [19]*wikipedia.org*. (2020, April 9). Retrieved from Wikipedia: [en.wikipedia.org/wiki/Generic\\_Routing\\_Encapsulation](https://en.wikipedia.org/wiki/Generic_Routing_Encapsulation)
- [20]Wordpress. (2015, February 2). Retrieved from [skminhaj.wordpress.com/2016/02/15/vxlan-encapsulation-and-packet-format](https://skminhaj.wordpress.com/2016/02/15/vxlan-encapsulation-and-packet-format)
- [21]*Salesforce*. (n.d.). Retrieved from [www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/](https://www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/)