DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM*, *SWEDEN 2019*

# Multi-Tenancy Security in Cloud Computing

Edge Computing and Distributed Cloud

**ALI SHOKROLLAHI YANCHESHMEH**

ERICSSON

**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE**

# Multi-Tenancy Security in Cloud Computing

Ali Shokrollahi Yancheshmeh

Master of Science Thesis

Communication Systems
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Examiner: Peter Sjödin
Supervisor: Markus Hidell

Ericsson
Supervisor: Christopher Price

Stockholm, Sweden
Dec 2019

# Abstract

With the advent of technology cloud computing has become the next generation of network computing where cloud computing can deliver both software and hardware as on-demand services over the Internet. Cloud computing has enabled small organizations to build web and mobile apps for millions of users by utilizing the concept of “pay-as-you-go” for applications, computing, network and storage resources as on-demand services. These services can be provided to the tenants in different categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In order to decrease the costs for the cloud users and increase resource utilization, cloud providers try to share the resources between different organizations (tenants) through a shared environment which is called Multi-Tenancy.

Even though multi-tenancy’s benefits are tremendous for both cloud providers and users, security and privacy concerns are the primary obstacles to Multi-Tenancy. Since Multi-Tenancy dramatically depends on resource sharing, many experts have suggested different approaches to secure Multi-Tenancy. One of the solutions is resource allocation and isolation techniques. In most cases, resource allocation techniques consider but are not sufficient for security. OpenStack community uses a method to isolate the resources in a Multi-Tenant environment. Even though this method is based on a smart filtering technique to segregate the resources in Compute nodes (the component that the instances are running on it in OpenStack), this method is not flawless. The problem comes up in the Cinder nodes where the resources are not isolated. This failure can be considered as a security concern for a Multi-Tenant environment in OpenStack.

In order to solve this problem, this project explores a method to secure Multi-Tenancy for both sides in the Compute node and for backend where Block Storage devices for the instances can be isolated as well.



## Keywords

Cloud computing, OpenStack, Multi-Tenancy, Security, Multi-Tenancy Isolation.

# Sammanfattning

Med tillkomsten av teknik har molnberäkning blivit nästa generation nätverksberäkning där molnberäkning kan leverera både mjukvara och hårdvara som on-demand-tjänster över Internet. Cloud computing har gjort det möjligt för små organisationer att bygga webb- och mobilappar för miljontals användare genom att använda begreppet ”pay-as-you-go” för applikationer, datoranläggningar, nätverks- och lagringsresurser som on-demand-tjänster. Dessa tjänster kan tillhandahållas hyresgästerna i olika kategorier: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) och Software as a Service (SaaS). För att minska kostnaderna för molnanvändarna och öka resursanvändningen, försöker molnleverantörer att dela resurserna mellan olika organisationer (hyresgäster) genom en delad miljö som kallas Multi-Tenancy.

Men fördelarna med flera hyresgäster är enorma för både molnleverantörer och användare, säkerhets- och integritetsfrågor är de främsta hindren för Multi-Tenancy. Eftersom Multi-Tenancy dramatiskt beror på resursdelning har många experter föreslagit olika metoder för att säkra Multi-Tenancy. En av lösningarna är resursallokering och isoleringstekniker. I de flesta fall beaktar resursallokeringstekniker men är inte tillräckliga för säkerhet. OpenStack community använder en metod för att isolera resurserna i en Multi-Tenant-miljö. Men denna metod är baserad på en smart filtreringsteknik för att separera resurserna i Compute-noder (komponenten som instansen körs på den i OpenStack), den här metoden är inte felfri. Problemet kommer upp i Cinder-noderna där resurserna inte är isolerade. Detta fel kan betraktas som ett säkerhetsproblem för en Multi-Tenant-miljö i OpenStack.

För att lösa detta problem försöker detta projekt säkra Multi-Tenancy för båda sidor i Compute-noden och för backend där Block Storage-enheter för instanserna också kan isoleras.



**Keywords**
Cloud computing, OpenStack, Multi-Tenancy, Security, Multi-Tenancy Isolation.

# Contents

# List of Figures

# List of Tables

# List of Acronyms and Abbreviations

This document requires readers to be familiar with certain terms and concepts. For clarity, we summarize some of these terms and give a short description of them before presenting them in the next sections.

| | |
|---|---|
| IaaS | Infrastructure as a Service |
| PaaS | Platform as a Service |
| SaaS | Software as a Service |
| CSP | Cloud Service Provider |
| IDS | Intrusion Detection System |
| IPS | Intrusion Prevention System |
| LXC | Linux Containers |
| TaaS | Tap as a Service |
| LOM | Light Out Management |
| IPMI | Intelligent Platform Management Interface |
| KVM | Kernel-based Virtual Machine |
| C-Groups | Control Groups |
| NAT | Network Address Translation |
| NTP | Network Time Protocol |
| SOA | Service-Oriented Architecture |
| CSA | Cloud Security Alliance |
| AOP | Aspect-Oriented Programming |
| OPNFV | Open Platform for NFV Project |
| NIST | National Institute of Standards and Technology |
| MTCEM | Multi-Tenant Trusted Computing Environment |

# 1 Introduction

Cloud computing is known as one of the most popular and widely exploited technologies that gives this opportunity to all small and big enterprises to access system resources via the internet. A wide range of users' needs such as data storage, processor power, and software via outside sources with the concepts of pay-per-use is fulfilled by Cloud Computing. It means customers (users) can use the resources as long as they pay for it as a tenant. Cloud computing brings great advantages for customers such as high flexibility and performance without requiring complicated maintenance tasks [27]. In order to take full advantage of cloud computing, Multi-Tenant architecture is designed with the goal of maximizing resource sharing among users. Not only Multi-Tenancy provides full resource utilization for the cloud providers, but also it decreases the cost for the clients. Multi-Tenancy can be described as an architectural structure that allows all resources to be shared by multiple users and sub-users at the same time [27].

Even though Multi-Tenancy brings many advantages both for service providers and customers, it is not flawless and it has its own security issues. Multi-Tenancy security issues are related to integrity and confidentiality risks in sharing resources in cloud computing. When multiple users are sharing the same resources, a malicious user can take the advantage to get access to all other users' resources by using some tricks [1].

Network security experts suggest different solutions to overcome Multi-Tenancy security issues. Some suggest using resource allocation techniques due to the nature of Multi-Tenancy [27]. Other security experts, on the other hand, are of the opinion that automated security control can be the best option for cloud providers to protect their network from malicious users. They offer Intrusion Detection System [10]. Host-based IDS and Network-based IDS are two types of IDS that can be deployed in a cloud environment.

However, for a Multi-Tenant environment, Network-based IDS cannot be useful since it can only address attacks from outsiders, not insiders [28]. Host-based IDS can be useful for checking inside attacks where both attackers and victims are located

in the same place. In order to solve this problem and avoid imposing security tasks to the customers, this project explores a method for isolating the tenants in a shared environment and shows the importance of the automation of that method where there are lots of users and nodes.

In the following, first, this report provides a general background of different aspects and areas such as problems, goals, and purpose of this thesis and its benefits and advantages. In the second chapter, all the urgent information will be provided in detail with graphs and figures. Chapter three describes the method and methodology which gives information about solving the problem and the utilized methods. Chapter four describes the implementation of the project, and it depicts the full resource isolation in OpenStack for a Multi-Tenant environment. Finally, in Chapter five and six results and conclusion will be shown. This project is ended up with two appendix parts in order to give technical details of the implementation.

## 1.1 Background

Cloud computing is an urgent need in the IT industry these days that it makes organizations needless of running data centers to run their applications without paying the high expenditure for buying or maintaining the hardware. Cloud computing can be deployed on four different types based on different needs: Public, Private, Community and Hybrid Cloud. Public cloud such as Amazon Web Services (AWS), Google Cloud Platform that are provided for general usage, Private Cloud that can be used for a single company, Community Cloud that is used by a group of users (companies) or Hybrid Cloud that can be a mix of other three models [17].

In addition the deployment models, Cloud Computing can be provided with different services; in Software as a Service (SaaS), cloud providers share software or application to multiple users over the internet different applications such as Google Apps can be an example for SaaS. In Platform as a Service (PaaS), a virtualized environment (platform) will be dedicated to developers (users), and a user can run its own applications on that virtualized environment. Google Apps Engine is one of the best examples for PaaS, Finally, in Infrastructure as a Service (IaaS), a pool of resources such as servers, routers, storage and switches are dedicated to a user. The user has the ability to compute, storing and network resourcing, and he can control these resources without managing the infrastructures.

OpenStack is one of the most famous IaaS providers [15]. OpenStack is an open-source cloud computing platform to implement Infrastructure as a Service with high scalability for public and private clouds. OpenStack controls computing, storage and networking resources throughout a data center. All the management and provisioning are through APIs with a common authentication mechanism [18], [19].

This project uses OpenStack as a cloud provider where all the resources are established in three nodes as the jump host, controller and compute nodes. As previously mentioned, in order to decrease the cost for users and full resource utilization, cloud providers share the dedicated resources between multiple users. For example in this project OpenStack is used as the cloud provider with a Compute node. Compute node runs VMs as the instances and every single VM or compute node can be shared between multiple users. To provide security between the users in a shared compute node the best way that is suggested by many security experts is using the resource allocation technique before running any instances by user [27], [29], [33].

As an example in a private cloud, just imagine an organization that has two different departments (A, B) where they are assigned to different tenants (projects). The problem arises when the application that is used by department A needs to be totally segregated from everyone else even department B. Therefore, Multi-Tenant Isolation can meet the security concerns instead of creating a cloud region for every department.

## 1.2 Problem

Multi-Tenancy has pros and cons; form one side it increases resource utilization and decreases the costs, but from the other side, it brings security and privacy concerns [1]. In a Multi-Tenant environment, users are separated from each other at the virtual level, but the hardware is not isolated and users share the hardware [3]. Some security specialists believe in using smart techniques for resource isolation and separation the tenants from each other to overcome security issues in Multi-Tenancy; The reason is to increase the security in Multi-Tenancy with doing resource allocation and make it harder and more costly for the attackers to investigate the network [1], [10].

As mentioned above, cloud computing can provide different services and in every service, Multi-Tenancy implies a different meaning. In IaaS, where this project uses the OpenStack as a cloud provider, just using resource utilization from the compute nodes cannot meet the security concerns because of using the same physical servers for the storage. So should have been omitted Multi-Tenancy due to the security flaws in the backend?

This project aims to answer that question.

Wayne Brown et al suggest the best precaution for making hypervisor and in general, IaaS secure and preventing attacks is maintaining and updating hypervisor software and implementing Intrusion Detection and Prevention System to have a permanent observation of the cloud environment [10]. Other perimeter security controls, like firewalls can be used as well, but due to the shared Multi-Tenancy nature of cloud computing it might be less effective. In OpenStack, it is possible to use two kinds of IDS, Network-based IDS, and Host-based IDS. NIDS tries to address attacks from outsiders and it has limited effectiveness against insider attacks.

HIDS can be effective but typically must be monitored and managed by the cloud users and every single user may use multiple shared instances [28]. The next question is, should we impose the security tasks to amateur users and ask them to secure the dedicated resources?

## 1.3 Purpose

The purpose of this project is to solve the security issues in Multi-Tenancy in cloud computing and avoid to burden users with security measurements. This project aims to bring one layer of security and privacy via dedicating a set group of Compute and Cinder nodes to a particular tenant and prevent the possibility of having malicious neighbors.

## 1.4 Goal

The goal of this project is to bring security for Multi-Tenancy via an isolation method for both sides in OpenStack; in compute nodes where the users want to lunch an instance, and for the backend where a volume (Block Storage) wants to attach to that instance to enable persistent storage.

### 1.4.1 Benefits, Ethics and Sustainability

This project plans to conserve Multi-Tenancy as one of the best properties of cloud computing in a secure way that users are not concerned about the privacy and security issues of a Multi-Tenant environment. Even though it imposes more complex configurations for the cloud service providers, it brings an automation method for doing the entire configuration with Ansible playbook. This paper not only meets the security requirements for both users and providers, but also decreases the costs for the users and increases the hardware utilization for providers through a secure Multi-Tenant area.

## 1.5 Methodology and Methods

The methodology that is used in this project will be qualitative research with the main work being an extensive literature study followed by a case study of the implementation at Ericsson. Moreover, research will be inductive where it uses [10] and [1] as facts and best practices for increasing the security in Multi-Tenancy by using isolation. The analysis will be qualitative to understand what needs to be done in order to validate the solution. The reason for selecting the qualitative method instead of quantitative is to find a suitable way to measure the effectiveness of isolation in both computing and storage instead of just isolation for computing.

## 1.6 Delimitations

This project mainly focuses on theoretical and practical implications for security issues in Multi-Tenancy in cloud computing where running automation of resource allocation can be the best solution to overcome the security issues of multi-tenancy in OpenStack. This project uses a cloud environment provided by OPNFV which is installed on bare-metal to implement OpenStack as IaaS. The architecture of OPNFV cloud is based on three nodes where one node is using the jump host for running the OpenStack commands, one node as the controller for running the most important services of OpenStack, such Horizon (web interface), Keystone, and Neutron; Compute node, where the instances are running on the hypervisor and Cinder as the OpenStack Block Storage service (Cinder is a software designed to create and manage a service that provides persistent data storage). This cloud environment is just using Virtual Machines and it doesn't consist of Nova Container Node for running the containers.

All those services are running in the LXC (Linux Containers), and in order to run the isolation method, the OpenStack features are enabled. It means that for allocating the compute nodes to the users, nova-scheduler filters are used [33]. For the backend, Cinder multi backend is enabled [34].

## 1.7 Outline

In the following, the Background section will describe all the essential requirements for the thesis where detailed information about cloud computing, OpenStack, multi-tenancy, security issues in Multi-Tenancy and Resource Isolation in OpenStack will be denoted.

# 2 Background

This chapter provides basic and detailed information about cloud computing, OpenStack, Multi-Tenancy in cloud computing. Additionally, this chapter describes security issues in Multi-Tenancy in cloud computing as the problem statement and its related works.

## 2.1 Cloud Computing

According to the National Institute of Standards and Technology (NIST), "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [14]. The cloud model consists of five essential characteristics, three service models, and four deployment models; you can see the general cloud architecture in Figure 2.1.

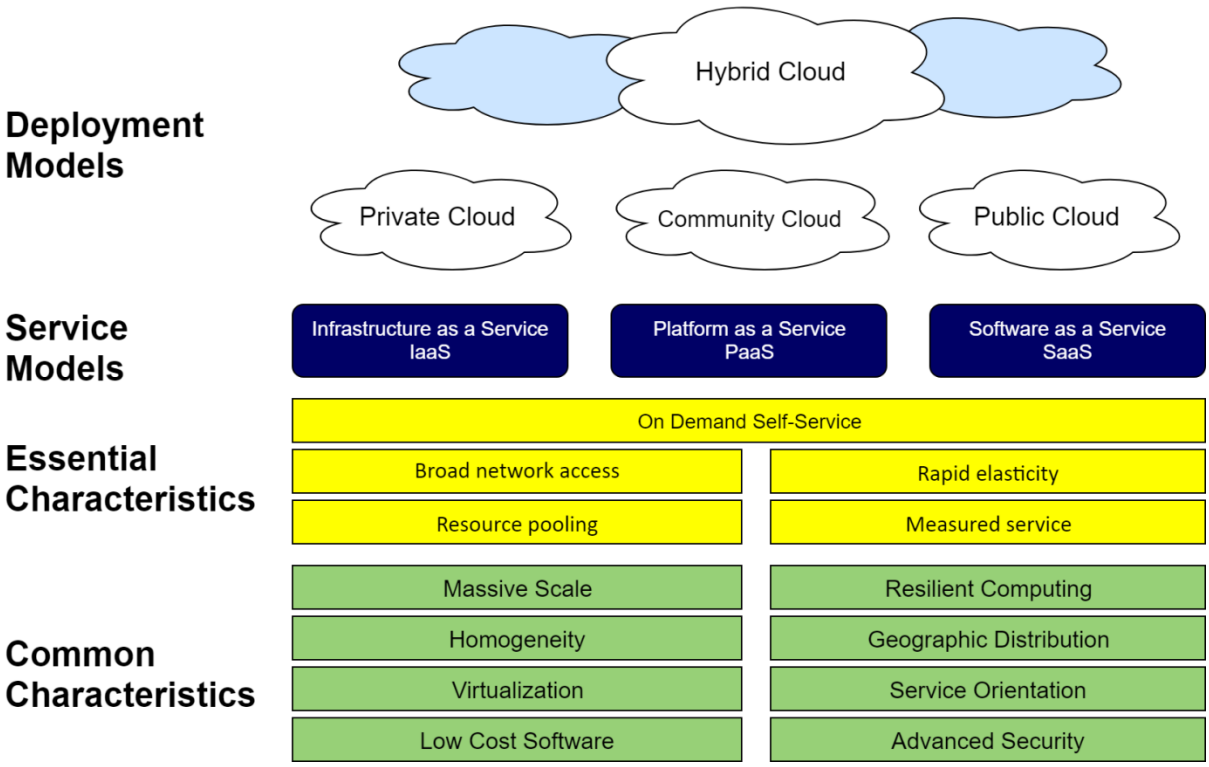



**Figure 2-1: cloud computing definition [17]**

### 2.1.1 Essential characteristics

**On-demand self-service:** The user can prepare computing features and capabilities on a one-way demand and automatically without human interaction in the middle [14].

**Broad network access:** features are ready to use, all entire network through standard mechanisms that rise via user's platforms such as laptops, mobiles, and tablets [14].

**Resource pooling:** The resources can be shared between multiple users in a Multi-Tenant model where according to the user's demand different physical and virtual resources can be shared between. The resources (such as storage and memory, network bandwidth, processors) can be dedicated to the users without any control or information about the resource location and resource equipment even though it may possible for the users to know about the location of the resources on a higher level such as country or location of the data center [14].

**Rapid elasticity:** the released capabilities should be flexible and automated to scale the demands quickly. In other words, the capabilities should be available at any time and any quantity [14].

**Measured service:** Cloud providers are the authority for control and optimizing the resources by using some metrics such as pay-per-use or charge-per-use according to the type of service. Moreover, resource usage and utilized services can be monitored and reported to both providers and consumers [14].

### 2.1.2 Deployment models

**Public cloud:** In this model, the cloud is provided for open and public utilization and it can be owned, managed and controlled by a company, university, government or a mix of them. In other words, Public clouds are available to general use and are owned by a third-party who offers the services. Third-parties store the data that is created and submitted by the users on the server [14]. In Public Cloud, resources are provided as a service to consumers via the internet (pay-per-usage-fee). There is no need for users to buy expensive hardware, and they can scale their usage on a

demand. The most important advantages of Public cloud are scalability, availability all the time. Talking about its shortcomings, security and privacy can be considered.

Reliability is one of the concerns about Public clouds, and it arises with the unknown data's location or the method that is used for storing data or the accessibility of data. Public Cloud's structure doesn't meet the specific organization's privacy and security concerns, so an organization needs to find out "is the selected Public cloud provider able to meet its security and privacy concerns" or not. The examples of Public cloud include Amazon Web Service (AWS), Microsoft Azure and Google Cloud Platform [15].

**Private cloud:** the private cloud is provided for a single enterprise or organization exclusively where multiple consumers can have access to the resources. A private cloud can be owned or managed by an organization or a third-party or both [14]. A private cloud is located in a data center of an organization or a company and only provides the services to the users inside the company. In comparison with the Public Cloud, Private cloud prepares more security and privacy, less complexity, and cost-saving in terms of the resources that the company consumes. It brings some un-used resources that can be shared with its partners in order to full resource usage. Private cloud computing provides better control over the infrastructure and computational resources. The most significant issue with the private cloud goes to the costs where a company needs to dedicate a huge budget for buying hardware, software, and stuffing [15].

**Community cloud:** community cloud is provided specifically for a group of users in an organization or enterprise who have the same concerns such as security rules or specific policies. This type of cloud can be owned, organized or controlled by one or more organizations or a third-party or a mix of them [14]. Community cloud can be considered as a mix of Public and Private cloud where it looks like a Private cloud, but the infrastructure and computational resources are shared between multiple consumers (organizations) with the same security and privacy concerns. This architecture provides less cost in comparison with the Private cloud since the dedicated budget is shared between multiple organs. Community cloud's consumers can pass on the management and control tasks to a trusted third-party, and decrease the cloud complexity by outsourcing the maintenance. Conversely, in comparison

with the public cloud, the Community cloud imposes more costs and fewer security abilities due to the shared bandwidth and storage between the consumers [15].

**Hybrid cloud:** It is a combination of two or more cloud models, comprising private and public or community by using standards and technologies that enable application and data portability where the user will stay as a unique entity. In other words, the Hybrid cloud includes two or more clouds that are unique entities, but they are bound together by a standardized or technology that enables data and application portability [14]. A Hybrid cloud is based on either "a vendor has a private cloud and forms a partnership with a public cloud provider or a public cloud provider forms a partnership with a vendor that provides private cloud platforms". In the Hybrid cloud, some resources can be hosted inside the cloud (in-host), and some other resources can be hosted as out-host.

Hybrid Clouds provide the benefits of public clouds such as cost and scale, and profits of Private cloud such as privacy and security. This kind of cloud refers to Hybrid IT. Hybrid clouds are usually used by organizations that are inclined to push some part of their tasks to public clouds either for cloud-bursting purposes or faster implementation. Since hybrid clouds are based on the company's requirements and the implementation structure, with no specific solution (model). Because hybrid environments include both private and public, some additional infrastructure security considerations come up. There is an urgent need for business planners who want to deploy hybrid clouds to know about different security requirements in order to mitigate security risks [15].

In general, it can be said, there is no much difference between different cloud deployment models, based on the organization's requirements where the Public cloud suffers from security and privacy, Private cloud benefits of good security but it is costly to establish. In the Hybrid cloud which is a mix of both, organizations use Public cloud for their regular data and Private cloud for their sensitive data. A community cloud is something between Public and Private Cloud where some organizations get to gather and make their own Private cloud.

### 2.1.3 Service models

As Figure 2.2 shows, Cloud Computing offers three types of services:

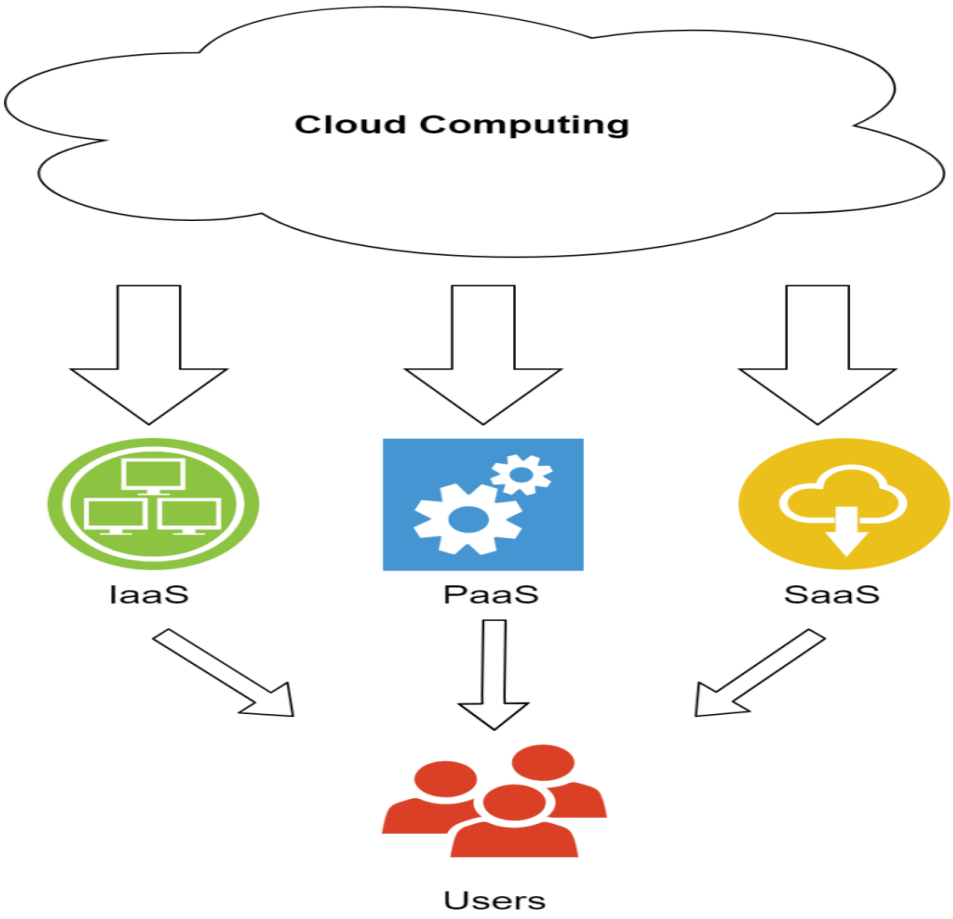



**Figure 2-2: Cloud Computing Services [13]**

**Software as a Service (SaaS):** In SaaS, CSP or Cloud Service Provider is responsible for providing the applications. In other words, an instance of the software will be provided to the users over the internet. In SaaS, the software has been shared between multiple users since it is considered as single-to-many. In SaaS, users are not able to control or monitor the infrastructures, so management and maintenance are centralized and is a duty of the cloud providers. SaaS is a network-based service, and cloud service providers need to make sure the service is up and running all the time. This service ensures the latest version of the software is available to the users where there is no need for users to buy expensive software. The examples of this service are Google Apps, NetSuite [8], [13], [14].

SaaS serves unique characteristics; first of all, users have access to the application via an internet-based interface which is typically run from a web browser. This feature easily provides scalability for adding new users. SaaS supports Multi-Tenancy where each user can share access to the software with the others, so users can opt either

increasing the scale for the cost or remain as a single tenant and have greater security and privacy. The SaaS model has a systematic model to support the software instead of maintenance and releasing patches to the subscribers, so users can use the benefits of the latest technological features provided by vendors without any disruption or cost for updating and upgrading.

The traditional method of using software is based on installing software on the user's computer locally and buying the license for authorization, but with the SaaS, consumers don't need to install the software locally and they just need to pay for the subscription (it also can be free), and software will be accessible via internet. The best example for SaaS is Google Docs, an online word processing application where consumers can access it via the internet and create their own document. In other words, Google provides an application that users can use but not alter directly. It looks like the traditional model, but is used from the internet [15].

**Platform as a Service (PaaS):** In PaaS, the entire hardware is dedicated to the user as a virtualized environment, and the user can run its own applications on that virtualized environment. PaaS supports scalability where users can ask for more hardware resources, moreover, it supports multiple programming languages that are available to developers (users) in different platforms. In PaaS, users are not allowed to control the resources such as server, network or host operating system, but users are able to manage their own infrastructure by using programming provided tools [8], [13], [14]. As a compare with SaaS, in SaaS, the application that is owned by a cloud provider is ready to use, but PaaS provides a platform to create and modify the applications where the PaaS model provides infrastructure as an operational and development environment for the deployment of the applications.

As an example, as one of the most famous development platforms, Google Apps Engine can be mentioned. Google Apps Engine helps developers by providing different tools like programming languages and APIs. From a security perspective, PaaS leverages the strong facilities of cryptography to secure storage and user's confidential data. These facilities provide domain security that protects user's data from unauthorized access in a cloud environment; in addition, PaaS provides a privacy feedback process that users will be informed about any risks that endanger their sensitive and confidential data [15].

**Infrastructure as a Service (IaaS):** In IaaS, a pool of resources that are necessary for running high-performance applications such as servers, routers, storage and switches are dedicated to the user. The user is able to prepare compute, storing and network resourcing and control these resources without managing the infrastructures. In other words, the user has control over the devices and the applications and interacts with the infrastructure, but functions are provisioned by the cloud service provider. In IaaS, an individual user is free to deploy and run every arbitrary software, containing applications or operating systems and control of selected networking components (e.g., host firewalls or host IDS) [8], [13], [14].

IaaS enjoys different standards and architectures from an organization to another, but one single solution is not designed for all. IaaS is a foundation for all delivery models; for example, Kubernetes can be installed on top of it in order to cluster the containers. IaaS is composed of different components: Physical and Virtual Servers, Storage Systems, Network connectivity and Network segmentation (Network blocs and virtual network areas), Network equipment (routers, switches, firewalls, etc.) , DHCP and DNS servers, virtualized platform, billing system, security equipment such as hardware-based or VM-based firewalls, Intrusion Detection and Prevention System. IaaS can be considered as a column for a cloud computing architecture where PaaS and SaaS are built on top of it. This architecture is clearly shown in Figure 2.3; as can be seen from Figure 2.3, on the left side the layered architecture is visible, and on the right side, different services and the related examples are shown [15].

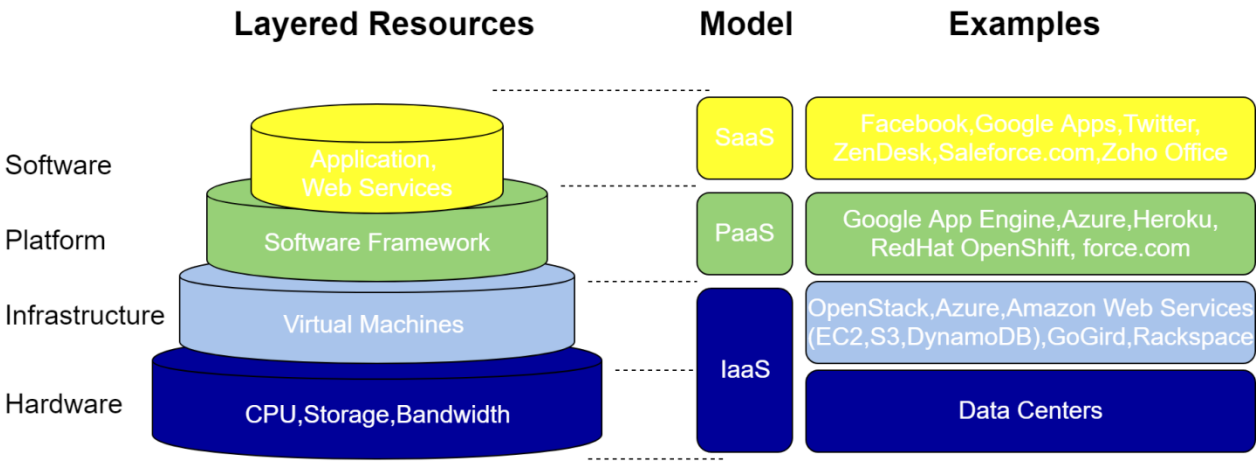



**Figure 2-3: Layered cloud computing models and examples [16]**

In general, what it differs PaaS from IaaS is, in PaaS user has no control over the virtualization instance or network configuration of the server, and from the other side, in PaaS, the user has no control on the hardware that application is running on it or application itself or network configuration [15].

This project aims to use OpenStack as an IaaS provider and talking more about its security issues regarding Multi-Tenancy. In the following, there is more information about OpenStack as an Infrastructure as a Service provider in cloud computing.

## 2.2 OpenStack

According to the OpenStack community, "OpenStack is a cloud operating system that controls large pools of computing, storage, and networking resources throughout a data center, all managed and provisioned through APIs with common authentication mechanisms" [18].

OpenStack is produced as scalable and adaptable open-source architecture for both public and private clouds with high performance. In cloud computing, all the resources such as storage, computing or network are provided to the end-users as services in the form of infrastructure, platform or software (IaaS, PaaS, and SaaS). OpenStack is an open-source cloud computing platform to implement Infrastructure as a Service with high scalability. More specifically, OpenStack is a combination of different software which is designed and developed by NASA and Rackspace in 2010 as an IaaS for both private and public clouds with high availability, scalability, reliability and performance [19].

Figure 2.4 denotes the OpenStack implementation from a high perspective. As Figure 2.4 shows, OpenStack provides resources including: storage, compute and network. These resources can be hosted as containers, VMs or bare metal. On the other side, Third-parties such as Kubernetes or Cloud FOUNDRY and etc can be installed on top of the dedicated instances. Access to these resources is either from a web user interface which is called Horizon or from the command line (OpenStack SDK).
As can be seen from Figure 2.4, OpenStack can provide Bare-metal, Virtual Machines, and containers as computing resources by using different drivers where it

uses Ironic for Bare-metal (Metal as a Service), Nova-compute for Virtual Machines and Nova-Docker for containers. An application can be deployed in three host types:

**Bare-metal:** There is no virtualization and entire hardware will be dedicated to the user as a service. OpenStack uses PXE, IPMI (Intelligent Platform Management Interface) and LOM (Light Out Management) for providing Bare-metal [20].

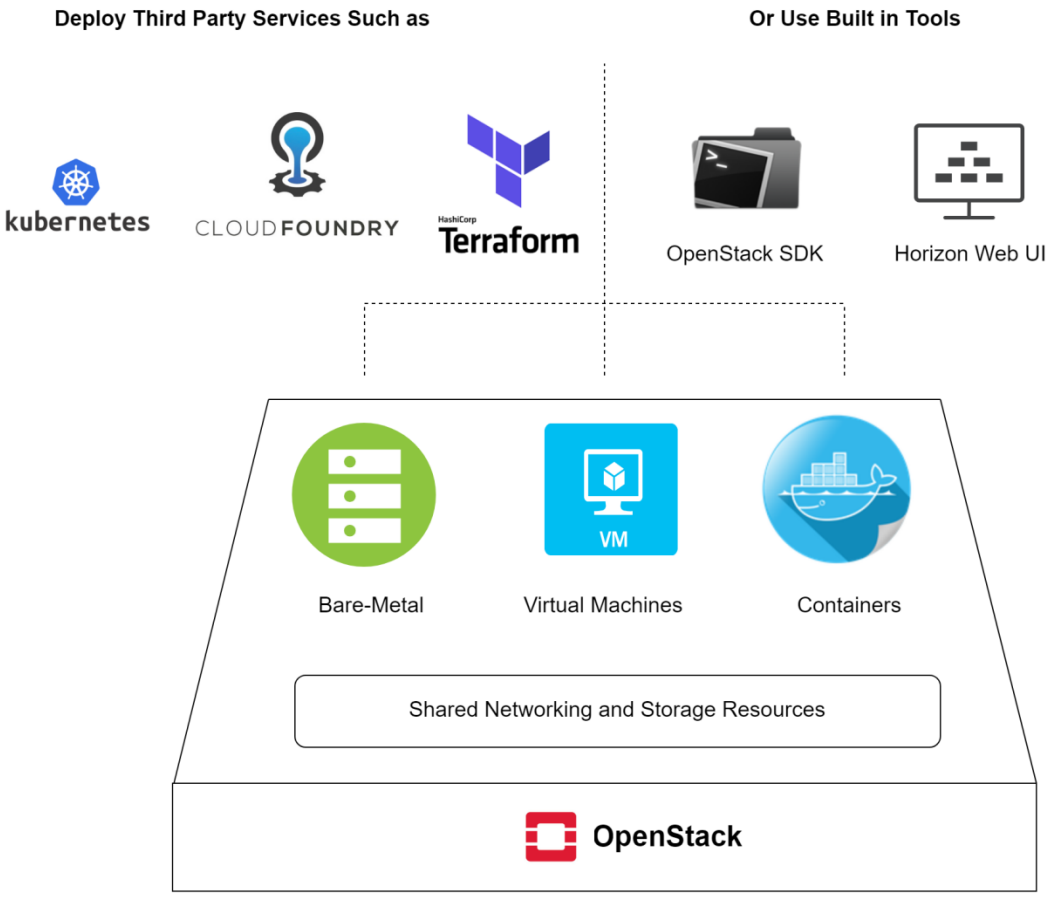



**Figure 2-4: OpenStack general view [18]**

**Virtual Machine:** VM is considered as the first traditional virtualization where a machine is delivered as a self-contained computer. This machine is running on the top of the Hypervisor layer that boots the Kernel of Operating System. In OpenStack, Nova-compute service uses KVM as a hypervisor for running the VM in Compute hosts [20].

**Containers:** containers are the light-weight approach of VMs, but with more speed. It is a way to isolate the resources by sharing the same OS's Kernel. In OpenStack,

Nova-Docker or LXC is responsible for provisioning containers in Compute hosts [20].

Figure 2.5 illustrates the differences between these computing resources and their features [20], [21]. As denoted by Figure 2.5, Bare-metal provides the entire Infrastructure (server hardware) to load, so applications (App 1) will be able to run on the hardware natively on the host and exploit all the resources. Bare-metal brings this option just for a single-tenant, and consequently, unused resources cannot be shared with the other tenants. In better words, Bare-metal doesn't support multi-tenancy. Besides that, Bare-metal increases the costs because of special extensions that are needed for access to hardware resources.

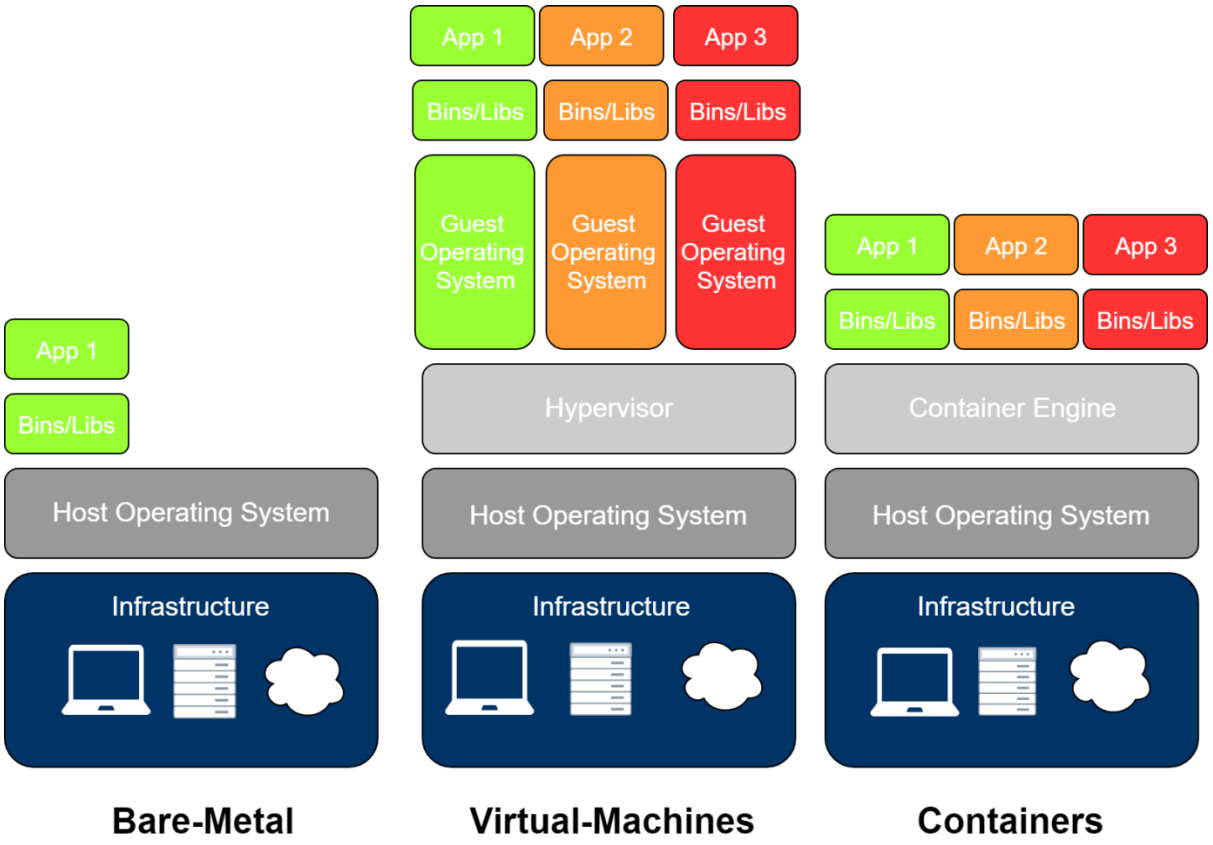



**Figure 2-5: Different Hosts in OpenStack [20], [21]**

Two other host types, Virtual Machines and Containers solve the limitations of Bare-metal via providing an extra layer (engine) which is called Hypervisor. Hypervisor which runs on top of server hardware (Infrastructure) runs the VMs and provides Guest Operating systems in a virtual environment. These virtual resources interact with hardware resources, so it brings load isolation and independent hardware

resources. There are two types of the hypervisor, Hypervisor type one or Bare-metal; in this type of hypervisor, VMs are running directly on top of hardware such as KVM, Xen, ESEXi. Type two hypervisor; VMs are running on top of the operating system that is running on top of hardware (example: VirtualBox, VM workstation).

In OpenStack by default, the Nova-compute node uses the KVM as a hypervisor type I (more specifically KVM-QEMU pair). The communication between OpenStack and KVM is managed and provided by Nova-libvirt. Even though VMs enjoy the isolation's benefits, data exchange between Hypervisor and guest operating systems is slow. In order to overcome this issue, KVM uses different methods to decrease overheads. First, Hardware Virtualization features in the processors. Second, KVM supports Para-virtual devices. Para-virtual devices are enabled via virtualization standard which is called Virtio. Virtio informs guest operating systems as only devices who know they are running on the virtual environment and working with the hypervisor.

Containers use the same process for isolation by modifying a shared kernel of the operating system, so there is no need to run a guest operating system. Instead of using a guest operating system by Control Groups (C-Group) and namespace (features of Linux) a workspace for them can be provided. Finally, it is lighter and faster than VMs because of the less time for booting. In OpenStack, the Nova-Docker driver runs the containers. These containers are composed of different tools and libraries that are needed for running an application on them [20].

OpenStack is composed of different open software which is called services. Figure 2.6 depicts the OpenStack architecture where you can see the OpenStack components and the interaction between them. These components can be divided into five different groups based on their characteristics: computing, networking and storing as the essential parts of cloud-computing for OpenStack. These five services are essential parts of OpenStack that can be installed based on the demands, so it is possible to install them all or just a few of them [19].

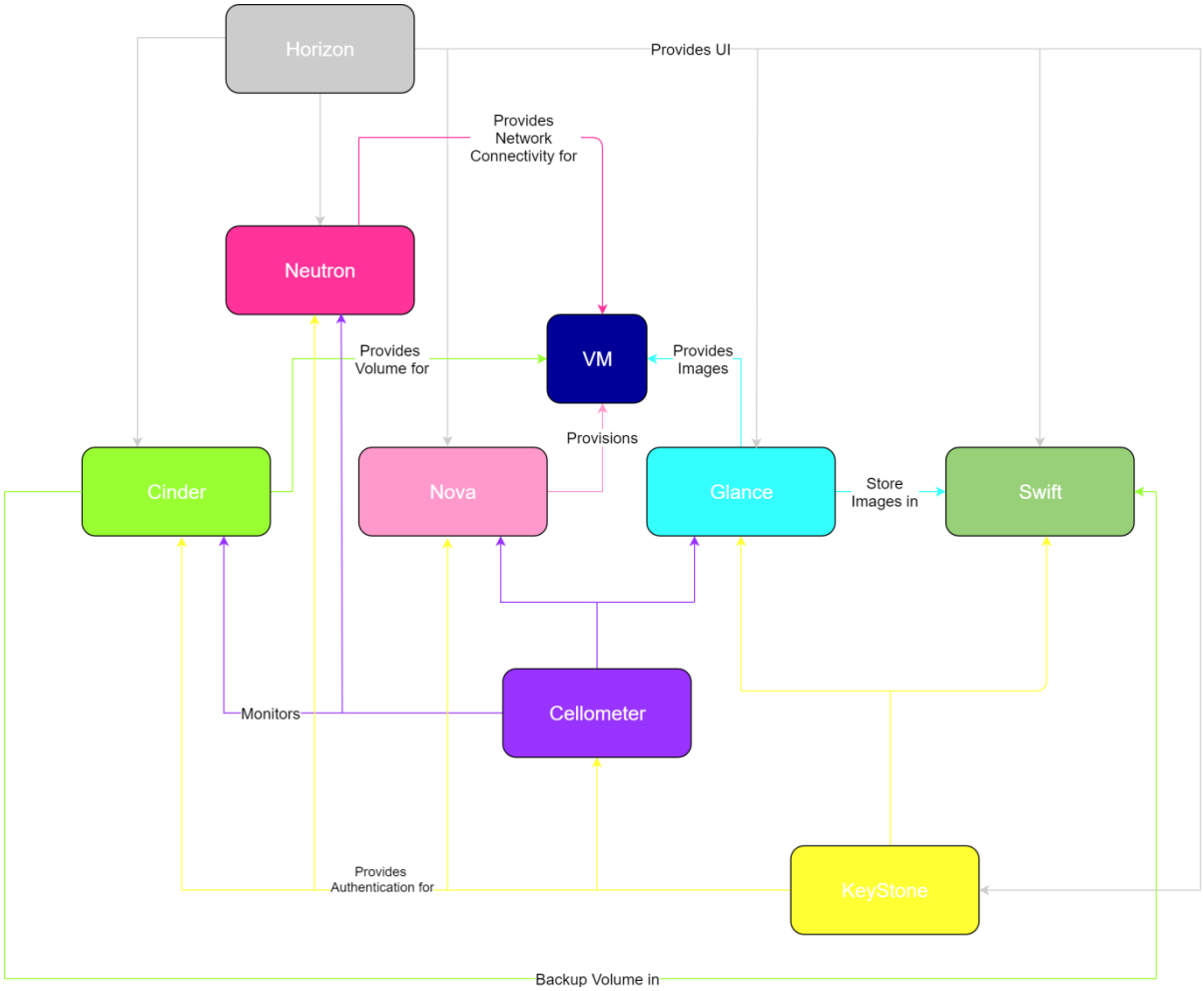



**Figure 2-6: OpenStack Conceptual Architecture [19]**

**Computing:** Computing in OpenStack is divided into two services: Nova and Glance. Nova or OpenStack Compute or Nova Compute node is an essential service for running the OpenStack. It provides virtual servers based on the demand on top of the hypervisor; in other words, this is a node or server for running the instances. Nova Compute node has an API to provide services for managing the instances in the cloud, such as access control, network management and orchestrating the instances. Glance or Image Service is another essential part of OpenStack that is responsible for creating, saving and retrieving an image of the instances (VMs). Glans uses an API for retrieving, saving and management of the metadata images and libraries [19].

**Networking:** Neutron is responsible for making a network connection between all other resources. By using the Plugin feature, this service has the capability to interact with different network technologies such as DHCP, IP, VLANs and other advanced topologies and policies. Due to its architecture, it is provided for the users to leverage

advanced frameworks such as Intrusion Detection System, Load Balancer and Virtual Private Network [19].

**Storing:** This service is composed of two different sub-services: Swift and Cinder. Swift or OpenStack Object Storage is responsible for saving and retrieving all the files. It allows users to store their files with high scalability. Cinder or OpenStack Block Storage provides block storages (volumes) to the virtual machines [19].

**Shared Resources:** This service is composed of Keystone, Horizon, and Ceilometer. Keystone provides authentication and authorization between the other services in OpenStack. It implies as a single point to unify policies, tokens catalogs and apply them to both users and services. Keystone also is an essential part of OpenStack that is totally urgent for the end-users. Horizon or OpenStack dashboard provides a modular web-application in order to manage the other services easily both for users and cloud admins. Lots of different management tasks such as running VMs, IP configurations and security group management. Ceilometer or OpenStack telemetry is a new service in OpenStack for checking and monitoring cloud services for metering data and billing [19].

**Supporting Services:** This service is composed of Database and Advanced Message Queue Protocol (AMQP) in order to have scalability. OpenStack by its default uses MySQL as a database for storing configuration and management of the information. MYSQL is installed on the controller node as the infrastructure node in OpenStack. AMQP is a service for different services to communicate with each other. By default, AMQP service uses RabbitMQ and supports Qpid and ZeroMQ as well. In this essential service, the broker uses a platform to send and receive the messages [19].

In this paper, the goal is to run VMs where it is based on using Nova-Compute even though after installation of the OpenStack nova-container will be available to run OpenStack's services. In order to run an instance or VM, there is a need for at least two nodes or hosts. The following part will explain more about the nodes (hosts) structure in a general view.

### 2.2.1 OpenStack Architecture

OpenStack has a distributed architecture; it means, it can be installed and configured in one or more than one host (computer). In other words, a single computer provides all the services such as Neutron, Cinder, Swift, etc or these services can be installed on different computers for load-balancing [22]. Today all the OpenStack services are installed on different computers with a multi-node architecture to have better scalability and performance.

In cloud computing, each cloud can have only one dashboard, one image store and one identity management, but it can have any number of compute instances and storage. These services can be hosted on different nodes and computers based on the cloud provider's policies and features [23]. As stated above, there is a need for at least two nodes for running OpenStack and its instances, but in order to have better capabilities, it is highly recommended to use three nodes architecture where OpenStack is composed of Controller node, Compute node and Network node.

**Controller node:** Controller node is a host computer which has a Linux as its operating system, and most of the OpenStack shared services are installed on this computer. The controller can have all of the non-compute resources or just a few of them, so the controller does not run the VM instances. The Controller has the Horizon (dashboard), image store and Keystone as the identity management service; moreover, OpenStack database, message broker and Neutron Server as the network management for Nova-compute are installed in controller [23].

**Compute node:** Compute node is a host computer with a Linux as its operating system. Compute node is responsible for running and managing the VMs, and it uses KVM as hypervisor by default and holds related services inside. Compute node has a Cinder volume service for installing and running the VMs. In OpenStack, it is possible to have more than one compute node [23].

**Network node:** Network node is a computer system that has Linux and has Neutron network service in order to provide virtual networking and connectivity between the instances. It uses Neutron Layer 3, NAT, NTP, tunnel and DHCP network services [23].

Figure 2.7 illustrates OpenStack with a three-node configuration. Figure 2.7 clearly shows different services that are basic or optional for running the OpenStack and finally the instances. In Figure 2.7, OpenStack only used one compute node which is titled compute node 1, but it is possible to have more than one compute node based on the demand and requirements. In this architecture, three nodes are using a common subnet called the management subnet. The Controller node and every compute node are sharing a separate common subnet called the data subnet. Each system is attached to the management network via the physical interface. The Network node and Compute node are attached to the data network through their physical interfaces [23].

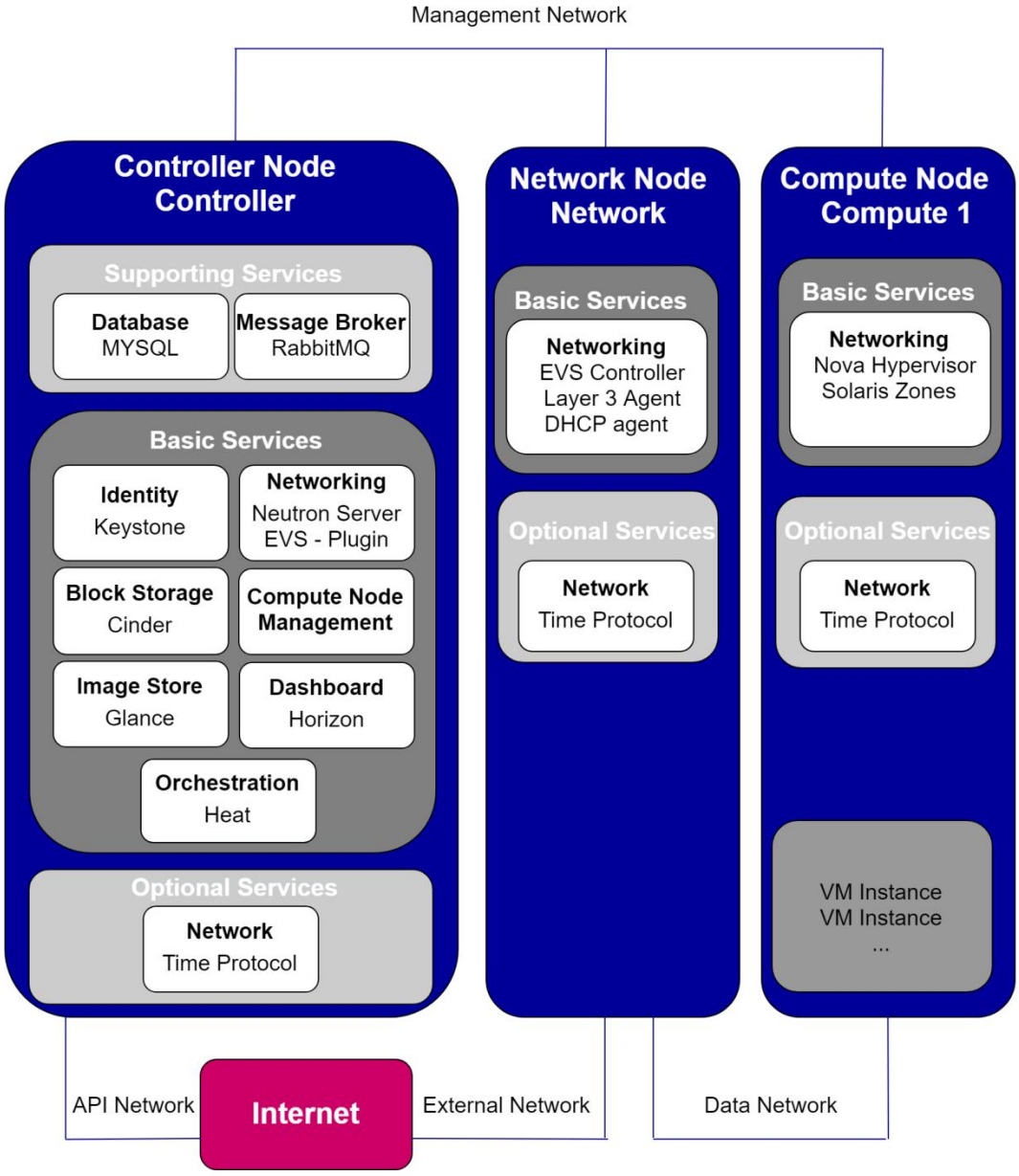



**Figure 2-7: OpenStack with Three-Node Configuration Architecture [23]**

For launching a VM Nova, compute uses nova-scheduler service in the compute node. Nova-scheduler uses some metrics to do the initial placement decisions such as checking availability of the resources based on the static information about resources, but it is not able to do management tasks such as load-balancing or power-management of the VMs. Nova-scheduler doesn't consider the current resources that are utilized for launching the VMs, and it saves all configurable options for modifying in a file (nova.conf).

## 2.3 Problem Statement

### 2.3.1 Multi-Tenancy in Cloud Computing

With the advent of cloud computing technology, cloud security has become a big issue in the cloud. Security needs to be considered as one of the most serious concerns for cloud customers such as enterprises and companies. That big issue mostly is driven by Multi-Tenancy that refers to sharing the resources in cloud computing that leads to integrity and confidentiality risks. In order to conquer the security issues in cloud computing and propose solutions, it is necessary to know more about the architecture of Multi-Tenancy, different attack vectors and attack surfaces [1].

Multi-Tenancy is a way of trying to achieve an economic gain in Cloud Computing by utilizing virtualization and resource sharing. Multi-Tenancy implies different meanings from different points of view and services. In SaaS, Multi-Tenancy implies; when two or more users use the same software or application that is provided by the Cloud Service Provider irrespective of the resources. In PaaS, Multi-Tenancy happens; when a platform or VM is shared between two users (Developers) or more. In IaaS, Multi-Tenancy happens when two or more VMs belong to different users are sharing the same resources (physical machines) [5], [8]. In order to have a better concept of multi-tenancy, Figure 2.8 shows the benefits of multi-tenancy.

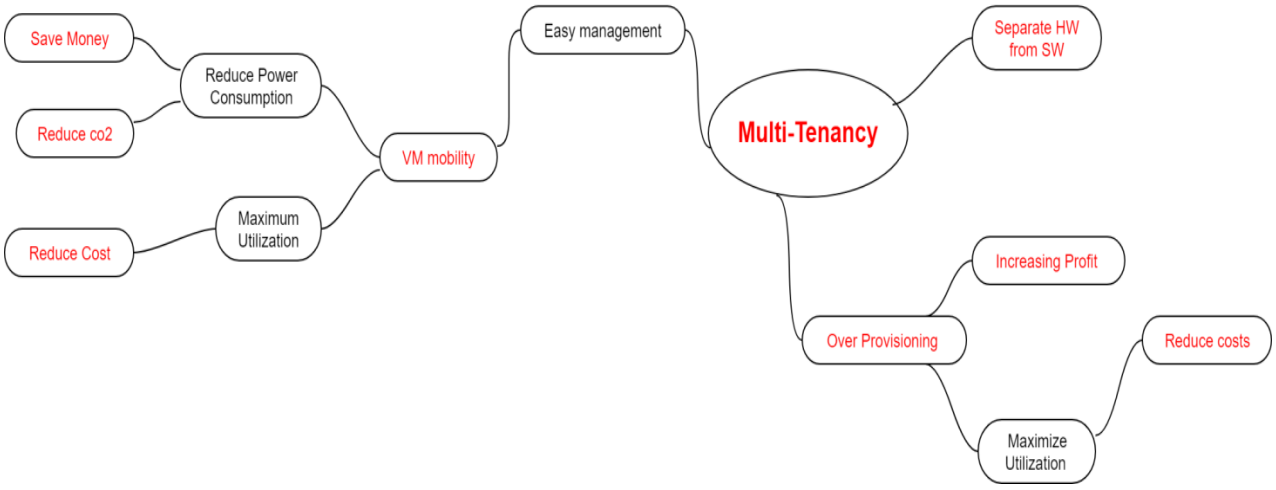



**Figure 2-8: Benefits of Multi-Tenancy tree [1]**

Figure 2.8 clearly shows all the possible benefits of Multi-Tenancy in cloud computing. As tree shows, all these benefits lead to either virtualization or resource sharing or a mix of them. In other words, it can be said "Multi-Tenancy = virtualization + resource sharing". By the way of the example for its benefits, the separation of hardware failures from software failures is possible by virtualization; or resource sharing brings a reduction in energy consumption that finally leads to a reduction of emission gasses and costs. These two inseparable features have a direct impact on VM mobility and over-provisioning. With VM mobility cloud providers are able to reallocate the VMs into clusters and minimize the number of the used servers through high resource utilization.

In addition, by over-provisioning cloud service providers are able to sell more than the resource's capacity to the users [1]. Multi-Tenancy has pros and cons; even though Multi-Tenancy is imagined as a great chance for the developers, security experts see the Multi-Tenancy as a vulnerability that can be considered as a threat to confidentiality. It is very important for the cloud service providers to try to keep both features, VM mobility and over provisioning and any security solution needs to consider those features [1].

Cloud computing can be defined as data center resource sharing through virtualization. Put it differently, multiple users use the same resource through different services. This system provides broad network access, scalability, and virtualization as the essential characteristics of cloud computing as a service to the customers (Pay-as-you-Go). The biggest challenge through the security in Multi-

Tenancy refers to the tradeoff between security and costs. Tim Watson says "…although one provider may offer a wonderful secure service and another may not, if the latter charges half the price, the majority of organizations will opt for it as they have no real way of telling the difference"[2].

Security has a significant role in cloud computing regardless of the service type (IaaS-PaaS-SaaS) that is dedicated to the users. By the way of example in the case of SaaS, cloud service providers provide software or application for users to use. In that case, if a malicious user gets access to the software's location, then the attacker has the potential to make inaccessible the resources for all other users. This type of attack can be extended for all other services [13]. In the following part, you can see the comparison between different services and the suggested solutions.

### 2.3.2 Multi-Tenancy Security Issues in Cloud Computing

There are many good reasons for Multi-Tenancy to be considered as a security threat in cloud computing; First of all, confidentiality and privacy can be menaced by multi-tenancy. Even though users are separated from each other at the virtual level, the hardware is not isolated and users share the hardware. Multi-Tenancy is relevant to multi-tasking in operating systems where multitasking or multi-processing share common processing resources such as a CPU. Multi-tenancy, like multitasking, directs to a large number of privacy and confidentiality threats [3].

Secondly, Subashini et al suggest using isolation for both in physical layer and application which this segregation needs to be enough intelligent to isolate the data from different users. Subashini believes the Intrusion of data of one host by a malicious user is possible due to the shared feature of multi-tenancy. This intrusion can be done either by hacking through the loopholes in the application or by injecting client code into the SaaS system [4].

Thirdly, Azeez et al describe security as the big challenge in a Multi-Tenant environment, and they tried to build a secure architecture for (SOA framework) where users are enabled to move their applications to a Multi-Tenant environment with almost changes to those applications [5]. To increase security in the cloud, the VR team has suggested a layer of security and depicts virtualization as an issue of the

hosting layer because of hosting different virtual machines in the same physical machine [6]. The last but not the least, Cloud Security Alliance (CSA) describes the problem with current security solutions for making a Multi-Tenant environment secure due to the separated location of the cloud environment. They do believe Multi-Tenancy has increased the potential of intrusion in the cloud [7].

The unique feature about Multi-Tenancy from a security perspective is in a Multi-Tenant environment both attackers and victims are on the same side, for example, they are sharing one VM. Figure 2.9 depicts the different cases of attacker and victim and networking between them.

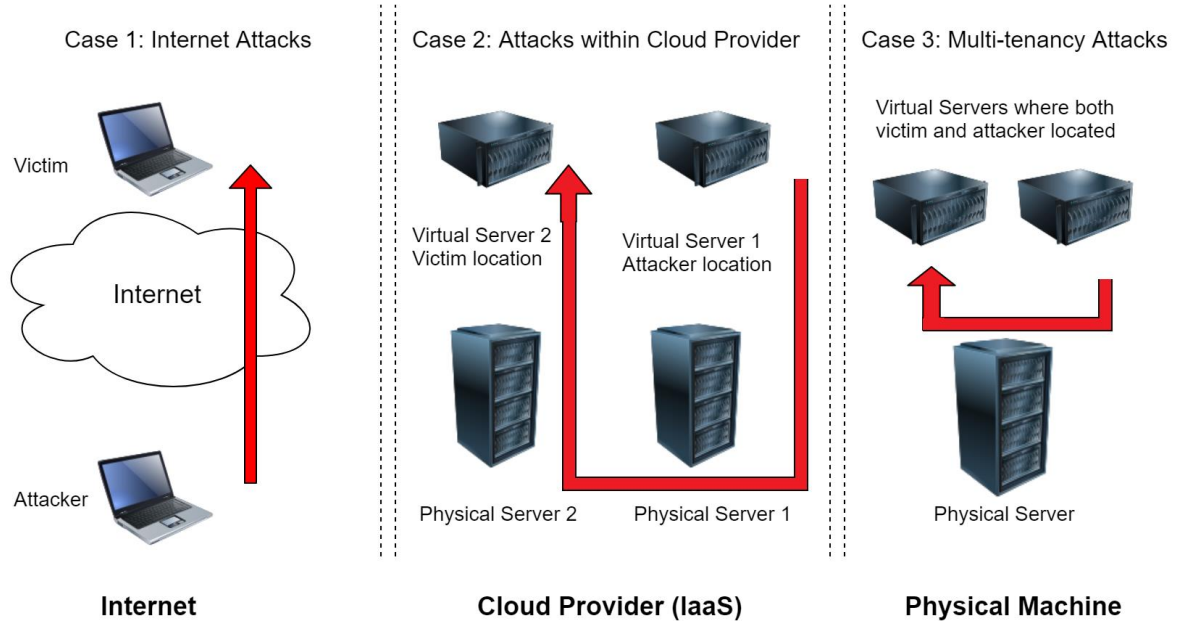



**Figure 2-9: Difference between Multi-Tenancy and other networks [1]**

As shown by Figure 2.9, in case one as the simplest case of cyber-attack, both attacker and victim are internet users. In order to prevent such attacks, users can use traditional cyber-security techniques. In case two both attackers and victims are the tenants of a cloud service provider, but they are located on separated physical servers. Due to the usage of a virtualization layer on top of physical servers, there is a need for utilization of virtual network security devices for providing security in the cloud or a way to isolate the shared resources. Case three clearly shows the problem that this project is trying to mention. As can be observed, case three illustrates Multi-Tenancy where both attackers and victims are cloud's tenants, and they are sharing the same physical server.

The network communication between the attack's VM and victim's VM through the physical server imposes one of the most security challenges in cloud computing. Something that makes it hard for security experts to overcome is in this case traffic will not leave the physical machine and it makes it harder to use virtualized network security devices that are used in case two [1].

## 2.4 Related Work

Shared Services as one of the most important concepts of cloud computing, implies a different meaning when different cloud computing services are used. In Sass; each instance of hosted application shares an instance of the object code, and any mistake in code structure or failure in memory can lead to data leakage or undesirable access to other user's data. One method for solving this failure is Aspect-Oriented Programming (AOP). With using AOP, clients are allowed to use different security methods such as authentication and authorization, encryption algorithm and cipher strength. It is possible by adding additional behavior to existing code [11].

In PaaS, every single tenant may consist of different layers of hosted methods across multiple physical servers. The risk in PaaS arises with the miss-configuration of the system. This risk can be decreased by using the dependency map for each tenant, so cloud service providers are obligated to provide a dynamic and updated mapping of infrastructure for each client's virtualized server. This is very helpful for solving network management problem that cloud service providers can use it to notify the tenants who are influenced by the crack [11].

In IaaS a single instance of hypervisor or a version of hypervisor is responsible for the control and partitioning of each hosted environment. On the other side, some recent exploits such as "Cloudburst" and "Blue Pill Project" have shown it is possible to let a guest in a VM move to the host and later on compromise the hypervisor via "rootkit-based"[12], [10]. In the next two parts, different solutions for securing the Multi-Tenancy in IaaS will be discussed.

### 2.4.1 Resource Isolation

In order to provide security in Multi-Tenancy in IaaS, the first task is to know how it is possible for Multi-Tenancy to be exploited. This question can be addressed by Saripalli et al [8] where they examined an attack that is done over Amazon cloud (EC2). In order to achieve this, attackers started to investigate the network, and later on, they followed their attack via a brute force attack. Multi-Tenancy did a great favor to the attackers where attackers were able to be the residence in their VM next to the victim's VM just by spending a trifle budget. Finally, attackers used the obtained information from the system and its characteristics to generate side-channel attacks and get the victim's VM information [1], [8].

According to the Amazon "The attack was a UDP amplification attack. In this attack, a UDP-based service is abused to attack others, wasting the bandwidth and computing resources" [10]. Saripalli et al [8] clearly show in a Multi-Tenant environment any malicious user can do the attack to other tenants, and above all, side-channel attacks cannot be detected by the hypervisor in the cloud. AlJahdali et al are of the opinion that Multi-Tenancy cannot be omitted due to the lack of security flaws. They believe, using smart techniques for resource isolation and separation the tenants from each other. In other words, they want to increase the security in Multi-Tenancy with doing resource allocation and make it harder and more costly for the attackers to investigate the network [1].

Wayne J et al believe the same, they do believe the main challenge with security and privacy in Multi-Tenancy is data isolation where the absence of effective bandwidth and traffic isolation increases the access chance for the attackers in the guise of tenants to jeopardize cloud environment. They introduce the side-channel attack as the most significant risk in cloud computing. A side-channel attack is a type of attack that works based on the obtained information from the system such as bandwidth monitoring or getting information from the hardware such as a hypervisor or other similar techniques. The only reason for that kind of attack is the lack of authorization for resource sharing the physical devices since unauthorized tenants can get the resources in the absence of control policies [10].

In order to have efficient data isolation Wayne j, et al introduces a Multi-Tenant trusted model which is titled "Multi-Tenant Trusted Computing Environment Model

(MTCEM)". MTCEM is based on the different security domains for isolating the user's data and access from each other more precisely, in IaaS or PaaS a given Host or Guest is allowed to belong to multiple security domains and use several security subjects via different security policies [10].

### 2.4.2 IDS & IPS

According to Wayne Brown et al, the best precaution for making hypervisor and in general IaaS secure and preventing attacks is maintaining and updating hypervisor software and implementing Intrusion Detection and Prevention System to have a permanent observant to the cloud environment. By using IDS and IPS cloud service providers are able to detect and alert the tenants from the malicious user's actions and do the needed actions [10].

### 2.4.3 Summary

This project will not examine IDS and IPS since it focuses on using the feature of OpenStack to bring the resource isolation between the tenants instead of using additional tools and imposing the security duties to the customers. In addition, most of the mentioned methods for resource isolation and allocation that are already done do not support OpenStack as the cloud provider. Mirantis produces a method for resource isolation in Multi-Tenancy in OpenStack even though they used their own filter that is called "PlacementFilter" to make the decision isolation policies by nova-scheduler [30]. Moreover, the OpenStack Community describes a method for isolating the resources, but it doesn't support the resource isolation for the backend(s) (where the block storage needs to be isolated) [41].

This project tries to implement a method to support resource isolation for both compute nodes and Cinder node. OpenStack provides some features that can be used to establish Multi-Tenancy isolation where specific tenants can be isolated in both from the instances and block storages. In order to achieve this goal, some features of OpenStack can be leveraged such as Host Aggregates, Availability Zone, Nova-Scheduler and Cinder quota. This project aims to use these features of OpenStack to provide a layer more of security and privacy for multi-tenancy. In the next section, Methodology and Methods; this project will explain its method for using the OpenStack different features to establish secure Multi-Tenant isolation.

# 3. Methodology and Method

This chapter provides an overview of the research method used in this project. Section 3.1 describes the research process, Section 3.2 explains the data collection technique used in the project. In Section 3.3, the experimental design of the project is presented and finally; Section 3.4 explains the validity and reliability of the collected data.

## 3.1 Research process

The various steps carried out in this research project are shown in order in Figure 3.1. The project started with gaining basic knowledge of cloud computing and the general advantage and disadvantage of moving to cloud computing. It was followed by learning OpenStack, Kubernetes and Multi-Tenancy in the cloud environment and its security issues. In parallel a case study was done at Ericsson to know more about Ericsson's cloud structure, and what has been done to provide Multi-Tenancy isolation for OpenStack.

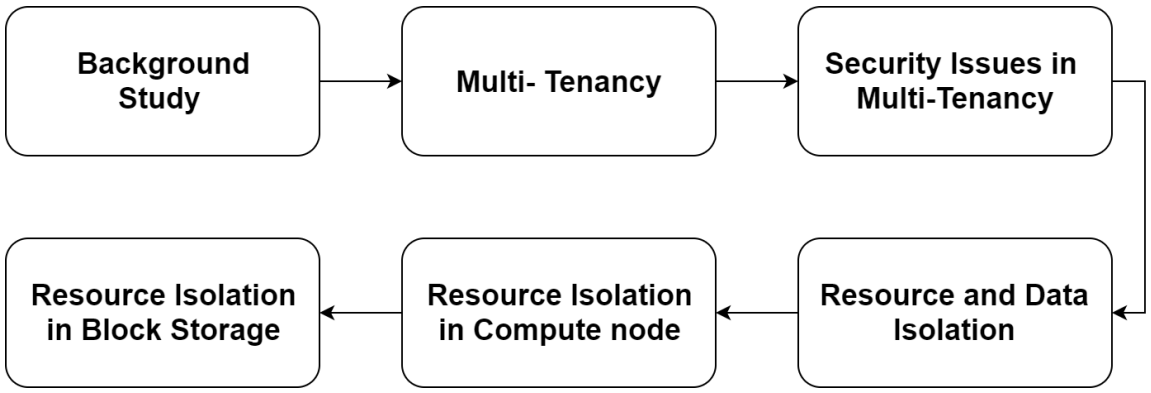



**Figure 3-1: Research process steps**

After the necessary background knowledge was acquired, the current implementation of Multi-Tenancy in both cloud computing and OpenStack had examined. It was followed by doing the Multi-Tenancy isolation for compute node and in next level was extended to the block storage. The first level of implementation acquired by different resources and the second level of resource isolation was gained

by a new method. It leads to the optimal path to increase security in Multi-Tenancy in OpenStack.

## 3.2 Data Collection

The data collection in this project was done in an inductive approach that it uses [10], [1] as facts and best practice for increasing the security in Multi-Tenancy by using isolation technique. The work was completed with basic theories of OpenStack features from different vendors or experts where all the collected data met the needs for the evaluation [29], [30].

## 3.3 Experimental Design

This section explains the environment experiments in this project were carried out. This includes software and the specific hardware that is used.

### 3.3.1 Hardware Platform

This project utilizes one x86_64 physical machine from OPNFV Linux Foundation with two different CPU sockets and 88 CPU cores. The CPU model was HP Enterprise. It has a 503 G of memory for RAM and 894 GB SSD hard disk.

### 3.3.2 Software Platform

The operating system that used throughout the experiments in this project was Ubuntu 16.04.5 LTS (Xenial Xerus). OpenStack Version Kilo is used. This cloud environment used LXC (Linux Containers) for running OpenStack services. The Scripts on the OpenStack is written in Python and moreover, this project used Ansible Playbook for making an automation which is used Python as the programming language as well.

## 3.4 Reliability and Validity

The information was gathered in the case study by doing discussions and interviews with experts who worked at Ericsson or OPNFV with OpenStack and then comparing the gathered information with the collected data from the literature study.

### 3.4.1 Reliability

Regarding the source of the information that this project used, it is important to mention that the information about future plans can be validated until the time that information was gathered.

### 3.4.2 Validity

There is a probability that if the same steps (in Chapter 4) were repeated again the results would be different or even some errors could become up due to the different environments. The other aspect of this project was because of the qualitative method that this project was based on; the method that was used could be different if someone another wanted to reach the same purpose.

# 4. Multi-Tenancy Isolation in OpenStack

As stated in the previous section, this project aims to provide a method to implement Multi-Tenancy isolation in OpenStack, and explains all the required levels and finally shows the importance of the automation for a complicated task like this. This chapter will cover the concept of Multi-Tenancy isolation and its implementation where it enables cloud providers to make force strict segregation of the tenants to assign to the resources. This project leverages three different services of OpenStack: identity (Keystone), Compute (Nova) and block storage (Cinder). Unlike the method is used by OpenStack Community that only focuses on the Compute and Keystone, this project shows how to implement block storage as well and its importance.

This method allows cloud administrators to allocate a group of compute nodes (Host Aggregates) and a group of Volumes to a specific tenant. It can avoid the possibility of noisy neighbors and above all will provide a layer of security and privacy in Multi-Tenancy in OpenStack [29].

## 4.1 Resource Isolation in OpenStack

Resource isolation is essential in a Multi-Tenant environment since all tenants are sharing the same resources. Resource isolation like authentication ensures that tenants only have access to their own resources but in a shared environment. In Multi-Tenancy, compute, storage and networking resources are shared by numerous tenants. It is therefore compulsory that separation of user's data is implemented.

In order to achieve, an authentication and authorization mechanism that controls the access and restricts tenants to only their resources needs to be implemented [42]. In Multi-Tenancy, each tenant needs to have its own individual secure computing resources, and with resource isolation, each tenant remains isolated and invisible to other tenants. Figure 4.1 shows the isolation scheme in OpenStack in general view.

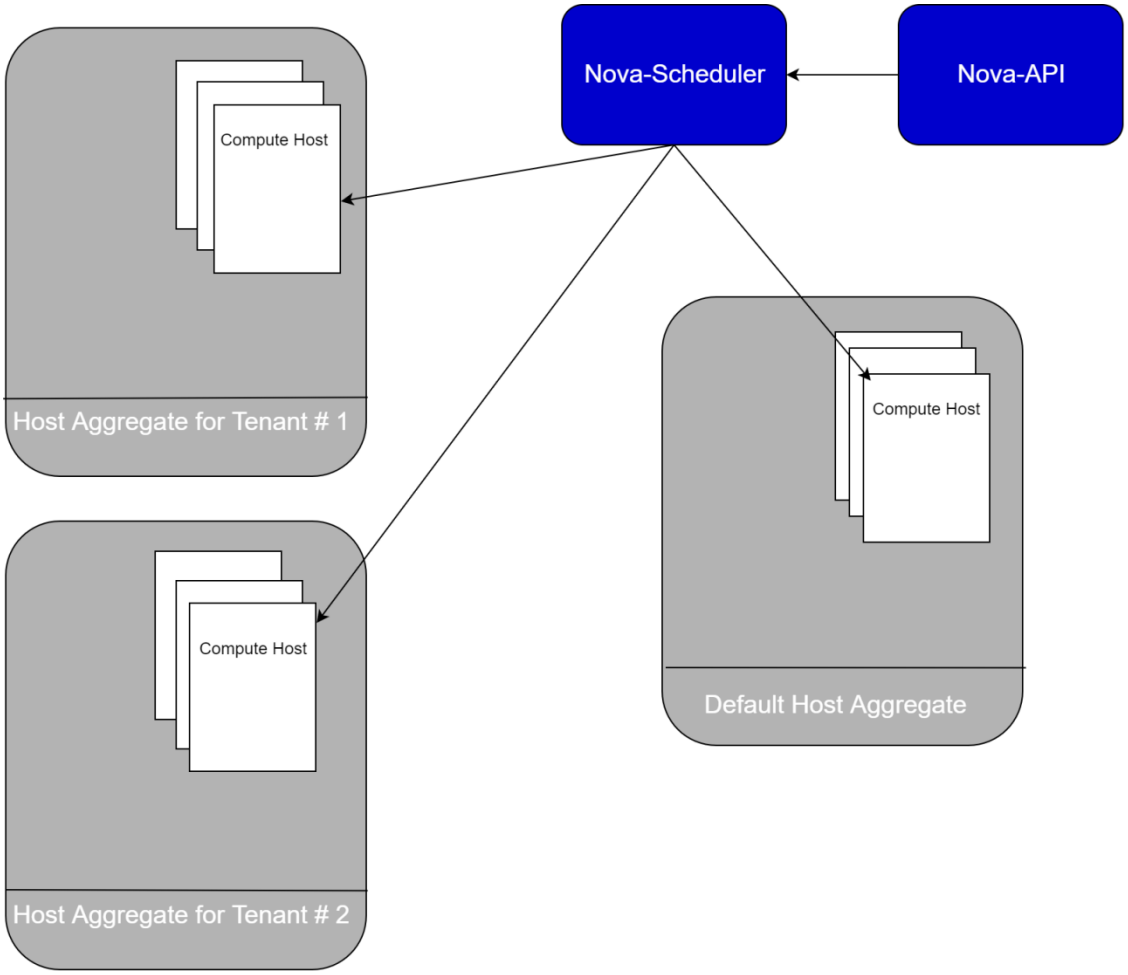



**Figure 4-1: Resource Isolation in OpenStack by OpenStack Community [30]**

## 4.2 Resource Isolation goals

The main goal of the resource isolation is to reserve an entire compute node only for the instances (VMs) of one tenant. In other words, making them isolated from the other instances (belong to other tenants) on the hardware level by using the authorizations mechanism. As can be shown by Figure 4.1, after resource isolation configuration, the creation of the tenants can be specified isolated or not isolated from the other tenants (Default Host Aggregate). In order to run VM instances for isolated tenants; only compute nodes that are grouped should be used. This is achieved by creating manually Host Aggregates.

The Nova- Scheduler decides to pick the proper Host Aggregate based on the filter policies. For the other tenants who are not isolated, Default Host Aggregate will be selected [30]. This method can be used for both private and public clouds in

OpenStack; by the way of the example, in a private cloud, an organization with two different departments that are using the shared resources of the same cloud. The concerns arise when a system or application on each department or organization needs to be totally segregated from the others. Multi-Tenant isolation helps to fulfill their concerns instead of creating a new cloud region or ignoring the multi-tenancy.

Multi-Tenancy Isolation can be used in a public cloud as well when OpenStack is used as a cloud provider; two or more organizations can share the compute nodes and run their virtual machines but in an isolated environment.

## 4.3 Resource Isolation Problems and Challenges in OpenStack

In OpenStack, the interaction between different controllers determines the allocation of the resources. The resources linked with resource allocation strategies are mostly schedulers and defined filters that are doing the authentication of the tenants. Different algorithms and methods are also developed for resource isolation and allocation in OpenStack, by OpenStack community or individual works. These algorithms are developed based on the strategies for allocating the resources on the group of compute nodes.

Considering Figure 4.1, OpenStack resource Isolation sets limits on the resources across multiple group of compute nodes (Host Aggregates), but it does not support the resource isolation for the volumes (Volumes are the Block Storage devices that are attached to instances to enable persistent storage) because it is not possible to create Host Aggregates for Cinder as well. Volume can be attached or detached to a running instance any time because for the block-storage there is no metadata to assign to a specific user. Moreover, we cannot use any specific filter for the cinder as well as something that is done for the Nova-scheduler. This can be considered as the first challenge. The second challenge that comes up; is how it is possible to assign a Host Aggregate and Cinder Quota to a specific tenant since they are totally run on different nodes (Computers) and have no direct connections (Cinder and Nova).

Most of the companies such as Mirantis who work in different areas of cloud computing and OpenStack with security problems as well as Multi-Tenancy isolation.

They emphasis, “the standards for sensitive tenants, isolated compute nodes” where they use only the “nova-scheduler” and more precisely “FilterScheduler” to do Multi-Tenant isolation [30].

The most significant problem here is; even though the resources are isolated in the compute node that VMs are running, different tenants are sharing the same resources in Cinder where the image of those VMs are stored. This can lead to data leakage and security and privacy concerns. An attacker can get the information from images of running VMs to do side-channel attacks.

## 4.4 Resource Isolation in OpenStack Implementation

This project exploits the new steps for the Cinder Backend to segregate them as well, to control which tenant can use the segregated Cinders and finally solve challenges. It is possible through the functionality part of Nova and Cinder which is called Cinder quota. By creating the Cinder quota cloud provider is able to control the access to the volumes as well for a specific tenant. This project is trying to use Volume types to provide the option to select which volume can be selected and manually create a direct link to the volumes. In order to have a backend Isolation, this project tries to run multiple-backend for the block storage and trying to authenticate the tenants.

In the following, all the essential information and different steps will be provided in more detail.

### Host Aggregates and Availability Zones

**Host Aggregates:** With the Host Aggregates cloud providers are able to group the hardware logically and do the partitioning in the deployment. Host aggregates are usually used to group the same hardware such as Compute nodes. This group of compute nodes gives us information that is known is metadata. Users can use this metadata to be authenticated and authorized when they want to lunch the instances because Host aggregates are only visible to the cloud administrators. For example, an instance can be launched in a specific compute host or hosts that we want. By using the metadata that specific hosts will understand this metadata is requested from the instance who wants to be launched on that compute node(s) [31], [32]. Figure 4.2

depicts an example of host aggregates to determine different groups of host aggregates. As Figure 4.2 shows, compute hosts can belong to more than one host aggregate.

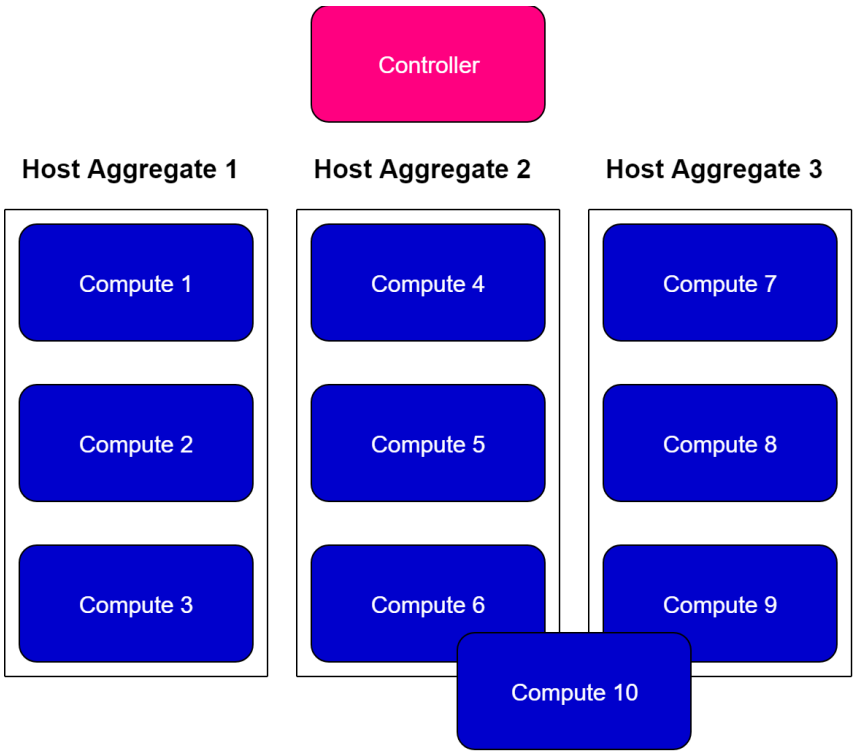



**Figure 4-2: Host Aggregates [31]**

**Availability Zones:** Unlike the Host Aggregates, Availability Zones are exposed to the end-users as a logical abstraction for partitioning cloud without any knowledge about physical infrastructure [32]. In other words, this is an abstraction for partitioning the compute resources that represents a group of hypervisors to the end-user for running the instances on them [31]. This abstraction is actually metadata that is attached to aggregate to be visible to end-users. Finally, it allows to schedule on a specific group of hosts who belongs to the aggregate [32]. With Host Aggregates, cloud administrators are able to create compute resources that are invisible to the end-user but can be grouped according to their purposes. The metadata will be added to an aggregate that matches a flavor's metadata, later on, that flavor will be used to launch the instance. The instance will run on the only compute hosts who are assigned to that aggregate [31].

As explained in Chapter 2, the responsible component for distribution of the instances across OpenStack cluster is nova-compute service, and more precisely nova-scheduler. Nova-scheduler is working based on filters to determine which instance is able to run in which host (host means every single node that nova-compute is running on it). The default filter for nova-scheduler is *filter_scheduler* that can be consisting of different filters and can be modified in *nova/nova.conf file*. Filter_scheduler supports two phases filtering and weighting to make decisions on where a new instance can be created [33]. In the filtering phase, a list of eligible hosts to respond to the request will be generated. Filtering is a binary, so it either accepts a host or rejects it. The accepted hosts will be ranked through passing different algorithms by the Weights phase to make a ranked list [24], [33].

Figure 4.3 shows two phases of nova-scheduler. As can be seen from Figure 4.3, hosts are chosen after the filtering at the first level and later on in the weighting level, they are ranked where Host 3 has the first priority and Host 1 has the last.

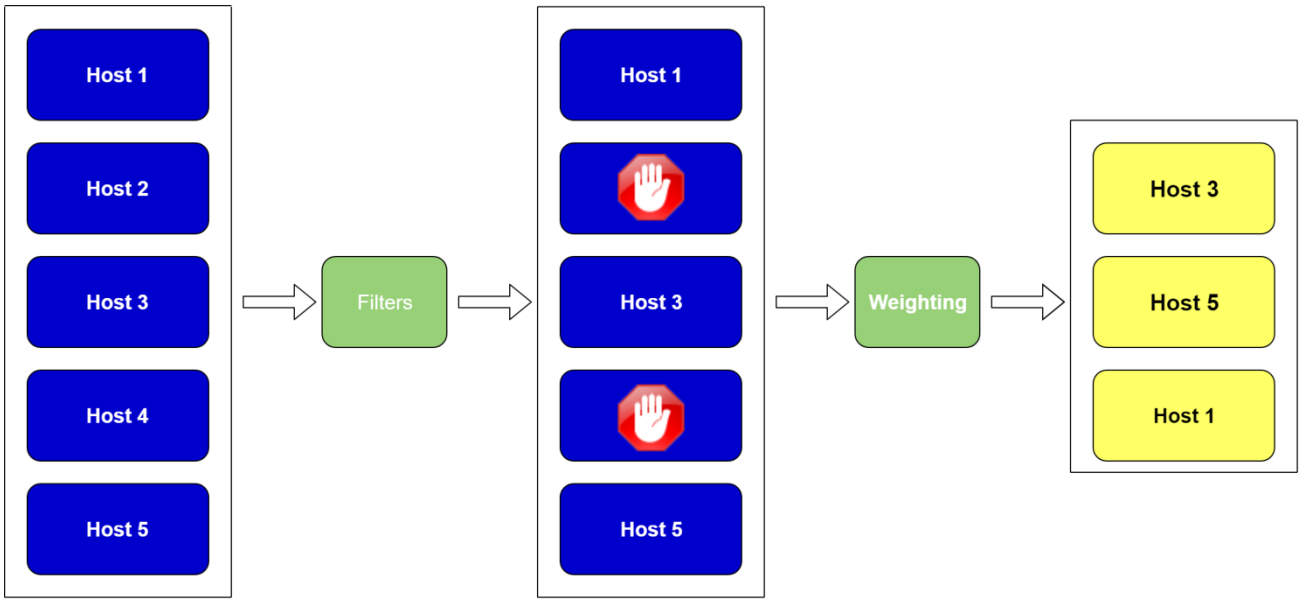



**Figure 4-3: Nova_scheduler filter [24], [33]**

The first task to enable the functionality is by adding additional nova-scheduler filters to the cloud. Two nova-scheduler filters need to be added: *AggregateInstanceExtraSpecsFilter and AggregateMultiTenancyIsolation*. By adding these filters, technically Multi-Tenancy isolation is completed. Multiple filters can be given, as a comma-separated list.

**AggregateInstanceExtraSpecsFilter:** This filter checks the defined properties for an instance with the admin-defined properties on a host aggregate. This filter works with specs that are restricted with aggregate_instance_extra_specs, but for backward compatibility, also it works with restricted specifications [33].

**AggregateMultiTenancyIsolation:** This filter ensures tenant(s) create the instances on specific host aggregates and availability zones [33].

After adding these filters, a virtual link between the tenant and host aggregates needs to be established. In order to do, it needs adding metadata tags to a specific component. In this case, metadata needs to be added to both host aggregate and custom flavor. By default, nova uses the defined metadata tag that is called filter_tenant_id for authorization.

By using this method, cloud providers are able to provide tenant IDs of the tenants that need to be isolated. Note that, if a host doesn't belong to an aggregate with metadata key, the host is able to create instances from all other tenants [29], [33]. As explained before, every created tenant can be coupled with a host aggregate with the provided ID, so if not, a new tenant will not be able to create any instances. That is why host aggregate needs to be established and not Availability zones (The Availability zones only allow for a compute node to be included) [36]. To wrap up, Nova-scheduler selects the suitable host aggregate and checks the available resources. As long as the resources are available, instances will be settled on the compute nodes.

As an implementation, this project uses the same process that is used for the Nova service for Cinder service (storage service). As mentioned above, it is not possible to create Host Aggregates for the block storages as well, but this project tries to use multiple block storage and proves it is possible to segregate Cinder backend and determine which tenant or tenants can use them. It is done via a built-in Cinder quota where it is considered as the functionality part of cinder and nova. Cinder quota provides authorization through the limitation of the allocation access to create volumes on a particular volume type [29] (Volumes are the Block Storage devices that are attached to instances to enable persistent storage. Volume can be attached or detached to a running instance any time) [34].

In order to segregate the block storage in OpenStack, multiple-storage-backend needs to be enabled for Cinder. Cinder allows adding multiple-backend configuration that is called “cinder-multiple-backend”. These different volumes can play the role of Host aggregates not for the flavors but for the block storages. Cinder-multiple-backend makes different volumes or external share storage devices available to the Cinder to connect. This ability can be done by enabling the enabled-backend flag in cinder.conf file. The manual scheduler defines where to send the volume based on the volume type that is sent in. [34].

After finishing the backend setup, there should be a direct link to the volumes; this is possible by creating and associating “volume types”. Volume types provide authentication by providing the option to select which volume can be selected. In other words, it makes it possible to separate the storage backend(s) and allows us to set volume types restrictions by Cinder quotas [35], [29]. Figure 4.4 shows all the described levels in eight steps.

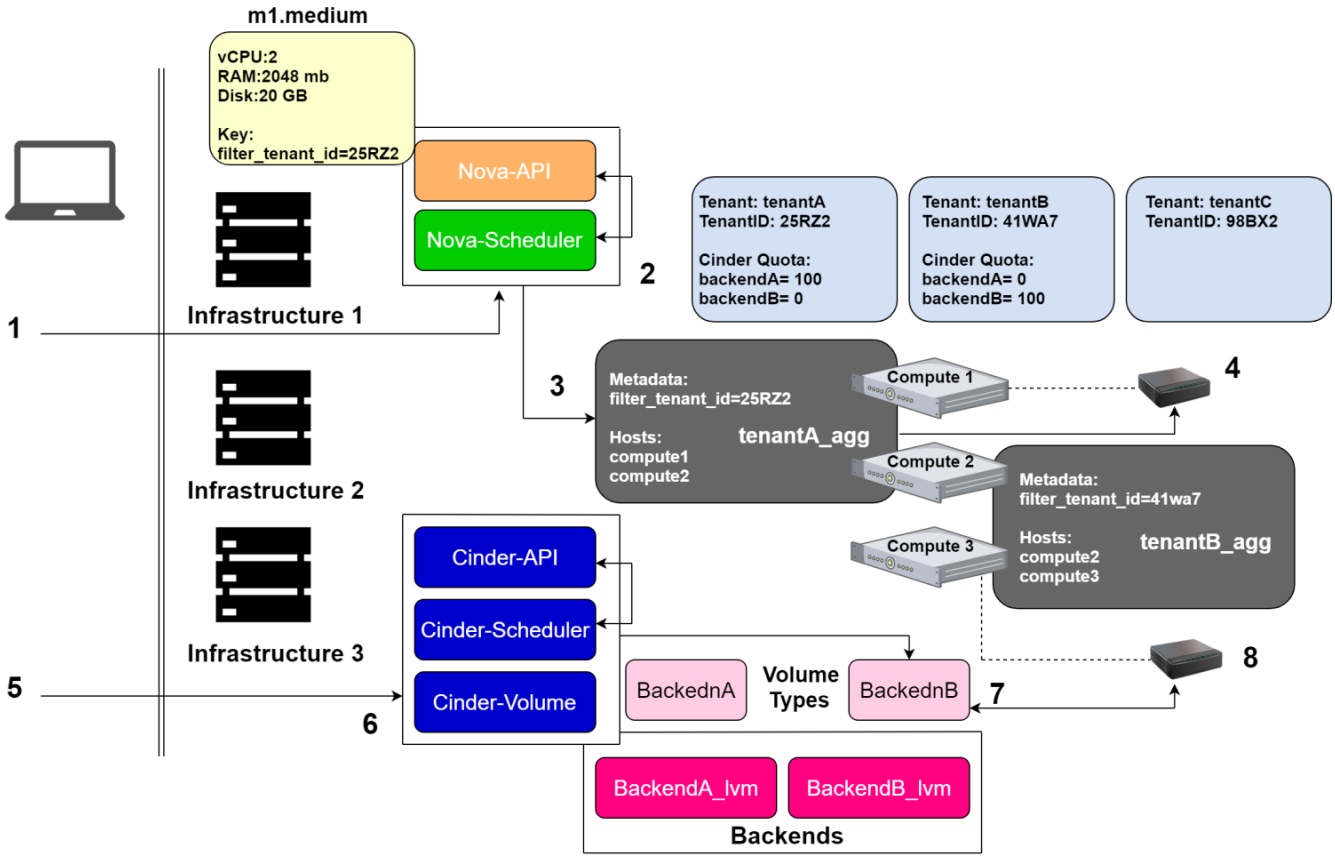



**Figure 4-4: Multi-Tenancy Isolation [29], [30]**

As Figure 4.4 denotes:

1. The end-user who belongs to tenantA sends a request to create an instance with flavor m1.medium and an image.
2. Nova-API receives the request and passes it to the nova-scheduler.
3. Nova-scheduler evaluates the request according to the defined filter, then according to the filter_tenant_id that is attached to the flavor in step 1, decides to which host aggregate is allowed to use.
4. After selecting the right host aggregate (tenantA_agg), now the scheduler is able to pass the request to the right compute node (compute 1 or 2) to setup the instance.
5. The end-user who belongs to tenantB sends a request to create a new volume.
6. Cinder-API receives the request and passes it to the Cinder-scheduler.
7. Cinder-scheduler evaluates the backend(s) on the cloud, and according to the --volume-type (backendB) as an ID, it will target specific volume.
8. The volume will be created on the volume-type named backend. The volume can be attached to the instance that is running on host aggregate tenantB_agg.

In Appendix A, this project will show the entire configuration for Multi-Tenancy isolation, and later on, it will show the configuration for making automation for those configurations via Ansible playbook in Appendix B.

# 5 Results and Analysis

This chapter combines the gathered information from literature and case study where it discussed in Chapter 2 and Chapter 4, and finally, a conclusion will be shown as to what important benefits for both Cloud Service Providers and customers. This analysis will focus on both Multi-Tenancy isolation just for compute node and the Full resource isolation method that is implemented by this project; Multi-Tenancy isolation for computing node and block storage, and its specific benefits.

## 5.1 Major results

This project can be discussed from two angles; according to the literature study and case study, there are two methods to improve the security in multi-tenancy. First, adding additional tools such as IDS and IPS to improve security. Second, using smart techniques to segregate the shared resources in OpenStack. In the following these two methods will be discussed and compared:

### 5.1.1 IDS & IPS vs. Multi-Tenancy Isolation

As mentioned earlier in Chapter 2.5.2, using security tools can be one of the suggested methods to overcome security issues in multi-tenancy. Two types of IDS can be used by OpenStack, Host-based IDS, and Network-based IDS. Both types of IDS have their own advantages and disadvantages. NIDS is used for checking, analyzing and monitoring the network traffic in order to protect a system or entire network. NIDS monitors all the inbound traffic and searches for any kind of suspicious behavior from outside of the network that comes inside. When a treat has been detected by the NIDS, NIDS can make different decisions such as notifying the network admins, blocking the source IP address from the sender based on the severity of the attack. Snort is one of the most famous NIDSs. This kind of IDS has limited effectiveness in multi-tenancy because it just monitors the incoming traffic from the outside of the cloud [26].

HIDS is used for internal monitoring in a system (computer), HIDS notifies the administrators if a system has been compromised or any malicious user tries to attack the machine. OSSEC, Samhain, Tripwire, and AIDE are examples of HIDS

[26]. Host-Based IDS can have better performance in Multi-Tenancy since it can be installed on different nodes such as Compute or even directly in VMs to check the tenant's behavior. Host IDS is effective, but it needs to be monitored and managed by users. This method will impose security tasks for customers who may not be experts in network security. In addition, skilled attackers can disable Host-IDS easily [28]. Moreover, in order to run IDS, some cloud resources should dedicate to run an HIDS Server.

Many good reasons can be considered for Multi-Tenancy Isolation to perform better than IDS. First of all, Multi-Tenancy Isolation uses the features of OpenStack and doesn't need any additional tools. Secondly, for running IDS some resources should be dedicated. For example, for running OSSEC, one of the nodes should run the OSSEC server to monitor other OSSEC agents. Last but not least, the main aim of cloud computing is to decrease the workload on the customer side. By using IDS customers need to do some security tasks as well.

### 5.1.2 Multi-Tenancy Isolation for Compute nodes and Backend

Proper full resource isolation improves the security and privacy of both compute nodes and Block Storages, and also it helps in avoiding the different kinds of transient bottlenecks involved in the OpenStack cloud.

Full resource isolation is important because:

- Full resource isolation helps to ensure that the quality of service standards is met.
- Full resource isolation helps to fully isolate the data streams between the tenants through the authentication and authorization of the tenants in the entire OpenStack environment.
- From a security perspective, full resource isolation allows ensuring a high standard of security by countering different denial-of-service attacks.

Different algorithms are developed for resource isolation by the OpenStack Community. These algorithms are developed based on the strategies for isolating the presented resources in Compute Service. To ensure good resource isolation in a cloud environment, this project has developed a technique where is able to do resource isolation in all the OpenStack services.

This separation enables computation and storage to scale independently, making it possible to provide Multi-Tenancy and isolation. Therefore, OpenStack Storage runs on separated hardware without network connectivity to OpenStack Compute. The primary functions of this are to manage the creation, attaching and detaching of the block devices. The consumer requires no knowledge of the type of back-end storage equipment or where it is located.

This project used the same algorithm for the resource isolation in Compute nodes and extended it to the Blok Storage as well to make full resource isolation in OpenStack. Different services are used to do full resource isolation, Table 5.1 shows the different services used for resource isolation in Compute nodes and Block Storage, and in the following the comparison between will be discussed.

**Table 5.1: Resource Isolation in Compute node and Block Storage**

| **Solutions** / **Services** | **Resource Isolation in Compute Node** | **Resource Isolation in Block Storage** |
|---|---|---|
| Node | Compute | Cinder |
| Scheduler | Nova-Scheduler | Cinder-Scheduler |
| Host Groups | Host Aggregates | Multiple Storage |
| Filter | *AggregateInstanceExtraSpecsFilter* *AggregateMultiTenancyIsol* | cinder.scheduler filter_scheduler FilterScheduler |
| Authentication | Metadata | Volume Type |
| Authorization | Filter-tenant-id | Cinder-quota |
| Partition | Flavor | Volume |

Table 5.1 shows the different services and attributes that are used for doing the full resource isolation in OpenStack. The attributes can be discussed from different angles; as evidenced by Table, for the Compute nodes, Host Aggregates are used and from the other side, Cinder nodes cannot be used for making Host Aggregates. In that case, this project tries to run multiple backend(s) for the Cinder node. Resource Isolation in Compute and Resource Isolation in Cinder can be compared from different perspectives:

- Both, Host Aggregates and multiple Cinder Backend(s) need to be created by Cloud administrators, and they are not visible to the end-users. Tenant users need to be authenticated and authorized to connect and use them.
- Both Nova-Scheduler and Cinder-Scheduler work based on the Filters, but they use different filters. Nova filters are made for resource isolation in multi-tenancy, but Cinder filters select the storage provider node on which to create the volume and it does not support resource isolation for storage nodes. The cinder filters are used just to select the storages and they do not do authentication. In order to solve this problem, this project used the method to do authentication manually.
- Host Aggregates provide Metadata that can be used by the user tenants to select the corresponded Host Aggregate through creating a virtual link to the Flavors when the Flavor uses the filter_tenant_id. Multiple-Storage does not provide any metadata for the end-user to be authenticated. The Volume Type creates a link to the backend(s) and Cinder quota allows or restricts different access to the different backend(s).
- Even though the authentication and authorization mechanism that used for the backend(s) does not any metadata in comparison with the authentication mechanism for the compute nodes, it provides authentication and authorization by setting restrictions to the allow access to the tenants.

As discussed above, although this method does not use metadata for the resource isolation in the backend(s), it can fully cover the Multi-Tenancy isolation in OpenStack. Scalability is one of the advantages of this method. Multi-Tenancy Isolation enables more scalability and makes software development and applications

more efficient through providing more capacity and objects that enhance performance when a new VM is added to the cloud.

Tables 5.2 and 5.3 show the different methods of Multi-Tenancy and their pros and cons. These tables denote the different properties and advantages and disadvantages of the different methods to overcome security issues in Multi-Tenancy in OpenStack.

As to be shown by the tables, although IPS and IDS provide permanent observation in the cloud it has lots of issues such as extra resources, increasing security tasks. Resource Isolation by OpenStack does not have IDS's problems but it can not fully support resource isolation in all the services and nodes where Full resource Isolation solves this issue in Table 5.2.

**Table 5.2: Comparison between different methods for Resource Isolation**

| Method / properties | Resource Isolation by OpenStack Community | Full Resource Isolation by this Project |
|---|---|---|
| Resource Isolation in Compute Node | √ | √ |
| Resource Isolation in Cinder node | × | √ |
| Authentication & Authorization Tenants in Compute nodes | √ | √ |
| Authentication & Authorization Tenants in Block Storage | × | √ |

**Table 5.3: Comparison between IDS & IPS and Full Resource Isolation**

| Method / Properties | IDS & IPS | Full Resource Isolation |
|---|---|---|
| Special Features | Permanent Observation in Cloud Environment | Authentication and Authorisation the Tenants |
| Additional Tools | × | √ |
| Accessible by Attackers | √ | × |
| Security task for end-users | √ | × |

## 5.3 Reliability Analysis

Regarding the source of the information that this project used, it is important to mention that the information about future plans can be validated until the time that information was gathered.

## 5.4 Validity Analysis

There is a probability that if the same steps (in Chapter 4) were repeated again the results would be different or even some errors could become up due to the different environments. The other aspect of this project was because of the qualitative method that this project was based on; the method that was used could be different if someone another wanted to reach the same purpose in another environment.

The other important feature of security is Availability. In this project, we imagined all the resources are available all the time to the users and we just did our implementation based on the Authentication and Authorization.

## 5.5 Discussion

This section debates the benefits of Multi-Tenancy in two directions first cost-saving and security and privacy.

### 5.5.1 Cost Saving

One of the huge benefits of Multi-Tenancy is providing the ability for lots of companies to reside in the same infrastructure. By using this method will never be any idle or unused resources running. Cost-saving is the main purpose of the Multi-Tenancy where different tenants can share the shared resources and pay less money. Multi-Tenancy has the ability to be extended for the block storage that is done by this project. By using the old method for doing the Multi-Tenancy this feature will be lost for the Cinder. Since each of the tenants is using the same nodes and resources, the cost of development and maintenance lowers down.

Another important factor that helps cost-saving is the fact that Multi-Tenant isolation doesn't need additional software or significant changes in codes. Because there is only one platform to support and maintain, it will be much easier for cloud service providers to deliver services and maintenance.

### 5.5.2 Security and Privacy

As stated in Chapter 2 (problem statement) the biggest issue with Multi-Tenancy is security. This project tries to overcome this issue through resource isolation. Protecting the tenants can be considered from different angles; first of all, the only cloud provider can assign tenants to Host Aggregates and Cinder Backend(s) that is possible with metadata and no one else has any access to the metadata. Secondly, each object or file functions as independent instances carrying a volume of controlled and restricted access policies by metadata that can be assigned to a specific Host Aggregates and Cinder Backend(s). All these steps are considered as CSP's duty and do not impose any security task to the cloud customers.

# 6 Conclusions and Future work

This chapter will denote the arrived at the conclusion as well as the limitations of the study, the future work, and reflections that need to be done in order to get a better understanding of the subject.

## 6.1 Conclusions

As time goes on, cloud computing has become one of the most popular and widely exploited technologies that provides the chance for all small and big enterprises to access system resources via the internet. The final aim of cloud computing is to provide a pool of resources for all enterprises to have their own data center in a virtual way. This can be considered as the best practice for small companies to be developed through cost reduction. This cost reduction can be extended with Multi-Tenancy where the pool of resources is shared between more than one customer. Multi-Tenancy has both pros and cons; from the positive angle it leads to cost-saving for the customers and increase in resource utilization, but it has its drawbacks in security and privacy.

This project as a master thesis has aimed to overcome security issues with Multi-Tenancy in OpenStack. It uses some suggested solutions by different experts and academic works, and finally, it uses it is own method to fully support the customers via resource isolation. This project uses some features of OpenStack to develop its idea, and implement a new method. For example, Mirantis implements its own filter (PlacementFilter) in nova-scheduler instead of using the default filter of OpenStack, but from the other side, this project extends resource isolation for backend(s) as well that is not implemented by Mirantis.

It might be a good idea for researchers who are interested in this topic to work on new filters and compare the results with default filters by OpenStack. Moreover, using Host Aggregate was the work area that this project used however it could be interesting to find a way for using Availability Zone to compare with Host Aggregate.

## 6.2 Limitations

The biggest limitation can be considered as a lack of quantitative data. For the purposes of this thesis project, it would have been preferable to actually have a quantitative data, but not doing so was a necessary decision due to the project goal by the Ericsson Company for setting up such a test environment.

## 6.3 Future Work

The first task for the future would be to collect some quantitative data regarding using a new filter for nova-scheduler and compare it with *AggregateInstanceExtraSpecsFilter and AggregateMultiTenancyIsolation* filters. It can lead to interesting results or even improved for better performance.

The other task that can be done for future work is to work on Multi-Tenancy in Kubernetes. Multi-Tenancy in Kubernetes can be examined from two angles; first, it can be examined how Multi-Tenancy in the infrastructure layer (OpenStack) has effects on the Kubernetes clusters. Conversely, how Multi-Tenancy in Kubernetes has an effect on VMs in the infrastructure layer.

Above all, the best practice for future work is to implement new filters for compute nodes, also implement new filters for the backend(s) where the block storages are running can be an interesting topic to work.

The other future works that can be considered is using Multi-Tenancy in a different cloud environment and compare the results. This project only works on OpenStack although lots of cloud providers use AWS or Azure or Google Cloud. It is a good idea for the cloud developers to try the same method that this project used for Multi-Tenancy isolation in their own environment.

## 6.4 Reflections

This project tries to solve the security issue in cloud computing. It yields benefits for both companies who are developing their cloud and end-users who are using this environment. By using this method cloud service providers will allocate fewer budgets for maintaining their cloud environment. In addition, a secure Multi-Tenant

cloud encourages more enterprise to run a startup to decrease the costs. Improvement the resource utilization will lead to a decrease in energy consumption and finally less emission green-house gasses.

# References

[1]. AlJahdali, Hussain, et al. "Multi-Tenancy in cloud computing." 2014 IEEE 8th International Symposium on Service-Oriented System Engineering. IEEE, 2014.
[2]. Khorshed, Md Tanzim, ABM Shawkat Ali, and Saleh A. Wasimi. "A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing." Future Generation computer systems 28.6 (2012): 833-851.
[3]. Zissis, Dimitrios, and Dimitrios Lekkas. "Addressing cloud computing security issues." Future Generation computer systems 28.3 (2012): 583-592.
[4]. Subashini, Subashini, and Veeraruna Kavitha. "A survey on security issues in service delivery models of cloud computing." *Journal of network and computer applications* 34.1 (2011): 1-11.
[5]. Azeez, Afkham, et al. "Multi-Tenant SOA middleware for cloud computing." 2010 IEEE 3rd international conference on cloud computing. IEEE, 2010.
[6]. Team, Verizon RISK. "2015 data breach investigations report." (2015).
[7]. Pearson, Siani, and Azzedine Benameur. "Privacy, security and trust issues arising from cloud computing." 2010 IEEE Second International Conference on Cloud Computing Technology and Science. IEEE, 2010.
[8]. Saripalli, Prasad, and Ben Walters. "Quirc: A quantitative impact and risk assessment framework for cloud security." 2010 IEEE 3rd international conference on cloud computing. Ieee, 2010.
[9]. Amazon EC2 Abuse Report, https://security.stackexchange.com/questions/195164/amazon-ec2-abuse-report, Jul/2019.
[10]. Brown, Wayne J., Vince Anderson, and Qing Tan. "Multitenancy-security risks and countermeasures." 2012 15th International Conference on Network-Based Information Systems. IEEE, 2012.
[11]. Tsai, Wei-Tek, and Qihong Shao. "Role-based access-control using reference ontology in clouds." 2011 Tenth International Symposium on Autonomous Decentralized Systems. IEEE, 2011.
[12]. John Rhoton, *Cloud Computing Explained Second Edition,* Recursive Publishing, 2011
[13]. Jasti, Amarnath, et al. "Security in Multi-Tenancy cloud." *44th Annual 2010 IEEE International Carnahan Conference on Security Technology*. IEEE, 2010.
[14]. Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).

[15]. Goyal, Sumit. "Public vs private vs hybrid vs community-cloud computing: a critical review." International Journal of Computer Network and Information Security 6.3 (2014): 20.
[16]. Rashid Mijumbi, Joan Serrat, Network Function Virtualization: State-of-the-Art and Research Challenges, https://www.semanticscholar.org/paper/Network-Function-Virtualization%3A-State-of-the-Art-Mijumbi-Serrat/fb38375adf84f909e7784f9736192aafc3c9f7a9/figure/4 , Jul/2019
[17]. Baiju Joseph, Cloud testing, https://www.slideshare.net/baijuglad/cloud-testing-16617639, Jul/2019
[18]. WHAT IS OPENSTACK?, https://www.OpenStack.org/software/, Jul/2019
[19]. Rosado, Tiago, and Jorge Bernardino. "An overview of OpenStackck architecture." *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM, 2014.
[20]. Kominos, Charalampos Gavriil, Nicolas Seyvet, and Konstantinos Vandikas. "Bare-metal, virtual machines and containers in OpenStack." *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IEEE, 2017.
[21]. Docker Containers vs. Virtual Machines, https://www.aquasec.com/wiki/display/containers/Docker+Containers+vs.+Virtual+Machines , Jul/2019
[22]. Sehgal, Anuj. "Introduction to OpenStack." *Running a Cloud Computing Infrastructure with OpenStack, University of Luxembourg* (2012).
[23]. Installing Across Multiple Systems for a Multi-node Havana OpenStack Configuration,https://docs.oracle.com/cd/E36784_01/html/E54155/installmulti.html#scrolltoc , Jul/2019
[24]. Sahasrabudhe, Shalmali Suhas, and Shilpa S. Sonawani. "ComparinOpenStackck aVMwarere." *2014 International Conference on Advances in Electronics Computers and Communications*. IEEE, 2014.
[25]. Ashoor, Asmaa Shaker, and Sharad Gore. "Difference between intrusion detection system (IDS) and intrusion prevention system (IPS)." *International Conference on Network Security and Applications*. Springer, Berlin, Heidelberg, 2011.
[26]. Singh, Amrit Pal, and Manik Deep Singh. "Analysis of host-based and network-based intrusion detection system." *IJ Computer Network and Information Security* 8 (2014): 41-47.

[27]. Karataş, Gözde, et al. "Multi-Tenant architectures in the cloud: a systematic mapping study." *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, 2017.
[28]. Nikolai, Jason, and Yong Wang. "Hypervisor-based cloud intrusion detection system." *2014 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2014.
[29]. Bentley, Walter. *OpenStack Administration with Ansible*. Packt Publishing Ltd, 2016.
[30]. Roman Bogorodskiy, Placement control and multi-tenancy isolation with OpenStack Cloud: Bare Metal Provisioning, Part 2, https://www.mirantis.com/blog/baremetal-provisioning-multi-tenancy-placement-control-isolation/ , Aug/2019.
[31]. Jackson, Kevin, Cody Bunch, and Egle Sigler. "*OpenStack cloud computing cookbook*". Packt Publishing Ltd, 2015.
[32]. Host Aggregates, Availability Zones (AZs), https://docs.OpenStack.org/nova/rocky/user/aggregates.html , Aug/2019.
[33]. Compute schedulers, https://docs.OpenStack.org/nova/rocky/admin/configuration/schedulers.html#host-aggregates , Aug/2019.
[34]. Cinder-multi-backend, https://wiki.OpenStack.org/wiki/Cinder-multi-backend, Aug/2019.
[35]. Manage volumes and volume types , https://docs.OpenStack.org/horizon/latest/admin/manage-volumes.html , Aug/2019.
[36]. Belmiro Moreira, Multi-tenancy isolation with aggregates, https://blueprints.launchpad.net/nova/+spec/multi-tenancy-aggregates , Aug/2019.
[37]. SCHEDULE HOSTS AND CELLS, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/6/html/Administration_Guide/section-scheduler.html , Sep/2019.
[38]. Create and associate a volume type, https://docs.OpenStack.org/juno/config-reference/content/coraid_creating_associating_volume_type.html , Sep/2019.
[39]. 3.4. MANAGE HOST AGGREGATES , https://access.redhat.com/documentation/en-

US/Red_Hat_Enterprise_Linux_OpenStack_Platform/6/html/Administration_Guide/section-host-aggregates.html#table-aggregate-keys , Sep/2019.
[40]. Manage Block Storage service quotas, https://docs.OpenStack.org/newton/admin-guide/cli-cinder-quotas.html , Sep/2019.
[41]. Kyle Bai kairen, OpenStack Multi-Tenant Isolatio, https://github.com/kairen/openstack-handbook/blob/master/management/openstack-multi-tenant-isolation.md, Sep/2019.
[42]. Odun-Ayo, Isaac, et al. "Cloud multi-tenancy: Issues and developments." *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 2017.
[43]. Margaret Rouse,Ansible Playbook, https://searchitoperations.techtarget.com/definition/Ansible-playbook , Oct/ 2019.
[44]. edureka!, DevOps Interview Questions and Answers | DevOps Tutorial | DevOps Training | Edureka, https://www.youtube.com/watch?v=clZgb8GA6xI&t=3426s , Oct/2019.

# Appendix A

This section provides all the information for Multi-Tenancy isolation steps that support resource isolation in all the dedicated resources in OpenStack. This section will show all OpenStack CLI commands that are used in this Project.

As stated earlier, OpenStack is composed of different services that can be deployed on different nodes. This project uses the server from OPNFV community where it uses LXC (Linux Containers) for running the services. Figure A.1 illustrates the OpenStack structure in OPNFV server.

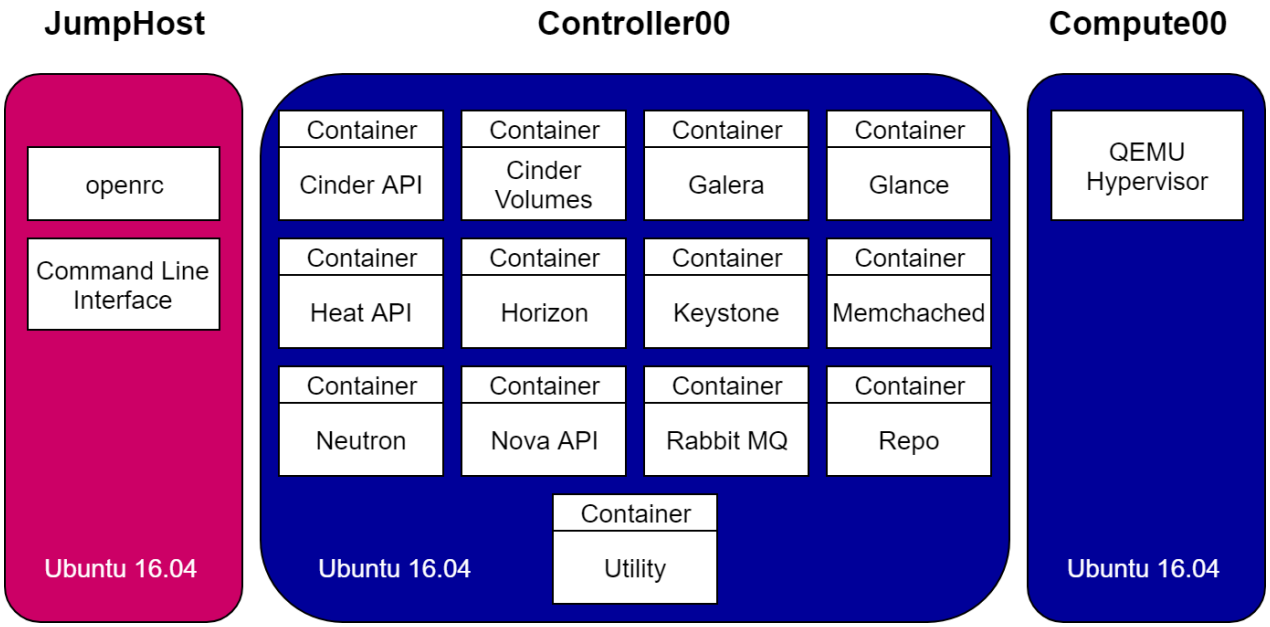



**Figure A-1: OpenStack in OPNFV Server**

Figure A.1 denotes the OpenStack architecture in a general view. As highlighted the , Controller node hosts all the services of OpenStack, and compute node is a node for running the VMs. JumpHost that hosts the openrc file, holds all the credentials for running the OpenStack CLI commands. All the nodes are using Ubuntu 16.04 as the operating system and they are in the same subnet. Controller uses HAProxy as the proxy and load-balancer in order to forward the request to the API endpoints. Figure A.1 shows only one compute node however we can have more than one compute node for running the Multi-Tenancy isolation or testing different scenarios. Moreover, this

project didn't use a single server (node) for the Cinder and all the storage will be stored in Cinder Container.

The first step for running the Multi-Tenancy isolation is to add the filters in nova-scheduler for changing the schedule policies. In the controller node, by running the "lxc-ls –f" command a list of all the containers can be shown. Every container can be accessed via either SSH or grep. For example, the nova container can be accessed via below command:

# lxc-attach -n $(lxc-ls -f | grep nova_api_container | awk '{ print $1 }')

Any changes need to restart the corresponded services to make them work. In order to change the nova-scheduler filter policies, the /etc/nova/nova.conf needs to be changed.

In [filter-shceduler] section, AggregateInstanceExtraSpecsFilter and AggregateMultiTenancyIsolation need to be added to the enabled_filters

```
[filter_scheduler]
max_instances_per_host = 50
max_io_ops_per_host = 10
ram_weight_multiplier = 5.0
available_filters = nova.scheduler.filters.all_filters
enabled_filters = AggregateInstanceExtraSpecsFilter,AggregateMultiTenancyIsolation, RetryFilter, AvailabilityZoneFilter, RamFilter, AggregateRamFilter, ComputeFilter, AggregateCoreFilter, DiskFilter, Aggr
egateDiskFilter, AggregateNumInstancesFilter, AggregateIoOpsFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter, ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter, NUMATopologyFilter
host_subset_size = 10
weight_classes = nova.scheduler.weights.all_weighers
tracks_instance_changes = True
```

To enable new changes in nova.conf below command needs to be run

# service nova-scheduler restart

By adding two filters, nova is obligated to work based on host aggregate's metadata and filter policies. AggregateInstanceExtraSpecsFilter ensures host aggregate's metadata and flavor's metadata are matched, and AggregateMultiTenancyIsolation ensures a host with specified filter_tenant_id can only hold the instances that belong to that tenant [29], [37].

In order to do Multi-Tenant isolation for the backend where the block storage can be isolated as well, Cinder multiple-storage can be added by adding enabled_backend(s) flag into the /etc/cinder/cinder.conf in cinder_volumes_container.

```
[backend_defaults]
target_helper = tgtadm

enabled_backends=backendA-lvm,backendB-lvm
[backendA-lvm]
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
volume_group=cinder-volumes

[backendB-lvm]
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDrivervolume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI_2
volume_group=cinder-volumes2
```

By running the below command cinder_volume service will be restarted and the entire configuration will be changed:

# service cinder-volume restart

After that, corresponded individual volume types to the backend need to be created. In order to, in jumphost or utility container the below commands should be used [29], [38]:

# cinder type-create <volume type name>

# cinder type-key <volume type name> set volume_backend_name= <backend name>

The above steps are considered as the pre-configuration for Multi-Tenancy isolation. In the following, the configurations that are needed to be applied for the tenant will be shown.

**Create a new Tenant (Project):** after pre-configuration; the first step for creating Multi-Tenant isolation is to create tenant or use the defined tenants. Tenant ID (project ID) plays a significant role in the next steps. All the tenants on OpenStack can be seen with “openstack project list” command, or a new tenant can be created with “openstack project create <project name>”.

**Create new Host Aggregates:** Host aggregates can only be managed by cloud administrators, and host aggregates assign compute hosts to associated metadata. The metadata is used to provide information for nova-scheduler to use, moreover; the specified metadata in a host aggregate will restrict the usage of a host to any instance that has the same specified metadata in its flavor [39]. Host aggregate can be created by either:

# OpenStack aggregate create <host aggregate name>

Or

# nova aggregate-create <host aggregate name>

**Adding hosts to Host Aggregates:** by creating host aggregates in the previous step, now in this step compute nodes should be added to the host aggregates. This project uses two compute nodes compute00 and compute01 for checking different scenarios. Hosts can be added to any aggregates with using the following commands:

# OpenStack aggregate add host < host aggregate ID> <host name>

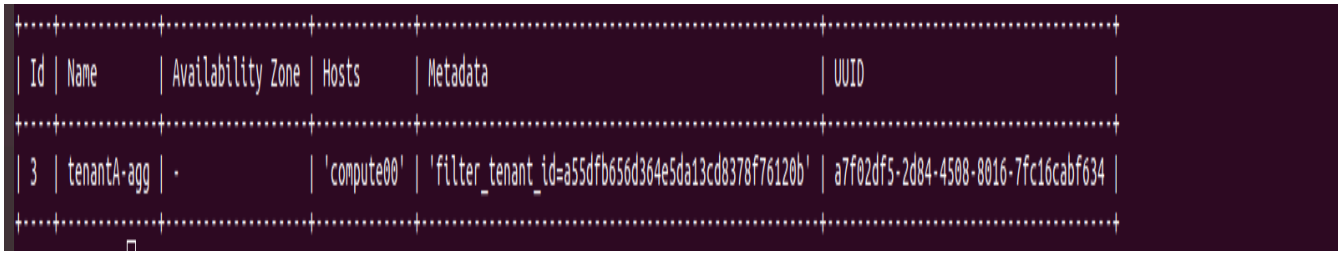
Or

# nova aggregate-add-host < host aggregate ID> <host name>

**Update Host Aggregate's metadata:** As the last part of host aggregate configuration, there is a need to update the host aggregate's metadata with filter_tenant_id. If filter_tenant_id is specified, the aggregate will only include this tenant according to the AggregateMultiTenancyIsolation filter that is set on the compute-scheduler. By this step, the tenant and Host Aggregate will be linked together [29], [39]. This step is done via below command:

# nova aggregate-set-metadata < host aggregate ID>  filter_tenant_id=<tenant ID>

The final result should be like this:



| Id | Name | Availability Zone | Hosts | Metadata | UUID |
|---|---|---|---|---|---|
| 3 | tenantA-agg | - | 'compute00' | 'filter_tenant_id=a55dfb656d364e5da13cd8378f76120b' | a7f02df5-2d84-4508-8016-7fc16cabf634 |

**Flavor:** Flavor is a part of the Compute node where instances are running with the allocated resources. With Multi-Tenancy isolation, Nova-scheduler will direct the cloud user's requests to the corresponded Host Aggregates and Compute nodes. The tenant filter accomplished this feature easily, so custom flavors needs to be created for each tenants with their tenant ID as a key to separate them from each other. In order to create a custom flavor following commands can be use:

# OpenStack flavor create --id <number> --ram <number> --disk <number> --vcpus <number> < flavor name>  --is-public false

Or

# nova flavor-create <flavor name> <id> <ram> <disk> <vcpus> --ispublic false

In the above commands, the flavor is not public and it is not accessible by the other tenants. The created flavor needs to be assigned to one specific tenant:

# nova flavor-access-add <flavor name> <tenant ID>

By running the above command, the flavor is limited to a single tenant, so only can be accessible by the users of that tenant. As the last part of the flavor, filter_tenant_ID needs to be added to the flavor. The key directs nova-scheduler to the appropriate host aggregate to allocate the resources when it is sending a request to boot the instance:

# nova flavor-key <flavor name> set filter_tenant_id=<tenant ID>

The final output should be like below:

```
+----------------------------+-----------------------------------------------------+
| Property                   | Value                                               |
+----------------------------+-----------------------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                                   |
| description                | -                                                   |
| disk                       | 20                                                  |
| extra_specs                | {"filter_tenant_id": "a55dfb656d364e5da13cd8378f76120b"} |
| id                         | 2ea98284-fa5a-4c10-a3ea-a9cfe89e75e2                |
| name                       | m1.custom.small                                     |
| os-flavor-access:is_public | False                                               |
| ram                        | 2000                                                |
| rxtx_factor                | 1.0                                                 |
| swap                       |                                                     |
| vcpus                      | 2                                                   |
+----------------------------+-----------------------------------------------------+
```

In order to have a short recap for debated steps and Figure 4.4, when a cloud user sends a request to create an instance on a specific flavor, nova-API receives the request and will send it to the nova-scheduler. Nova-scheduler evaluates the request and makes its decision. *AggregateInstanceExtraSpecsFilter* checks the metadata of flavor and host aggregate's metadata to specify which host aggregate should be selected. With the filter_tenant_id key, *AggregateMultiTenancyIsolation* specifies which host should be selected for running the instances [29], [39].

With enabled_backend(s), the cinder backend is linked to the volume types. It leads to having control to access the volume types using the cinder quota, volume types can be a part of the quota option. This is can be done with setting the values to restrict access to the volume type (by default all the volume types are listed as a quota option for all tenants) so by setting limitation, we can define a method manually for making backend exclusive for a specific tenant that has a specific host aggregate. The process will be shown below [29], [34], [40]:

# cinder quota-update --volumes <number of volumes> --volume-type <volume type> <tenantID>

The number of volumes specifies allow or limit access to the volume type where 100 allows access to the volume type and 0 will restricts access to the volume type. By passing all the steps Full Multi-Tenancy Isolation will be completed however doing all these steps can be hard to configure in a large cloud environment. In order to solve this complication and avoiding any mistakes, this project makes automation with Ansible playbook. All the steps for making configurations of the playbook and the Github link will be provided in Appendix B.

# Appendix B

This chapter describes a method for making automation for all the steps mentioned in Appendix A. With automation all the tasks can be done easily that helps to avoid any configuration mistake by cloud administrators. This project uses Ansible Playbook for doing the automation. In the following first general information about Ansible structure will be provided which is followed by Ansible Playbook code description.

## Ansible

Ansible is an open-source configuration manager and automation platform that is invented by the Red Hat community. Ansible uses YAML templates to program repetitive tasks automatically. In other words, it replaces the manual commands or scripts with automated repeatable codes. Ansible uses either modules or commands to push application code or programs to run in physical servers, virtual machines or cloud instances. By using Ansible, users are also able to invert the setup to pull architecture [43]. Figure B.1 shows the Ansible scheme.

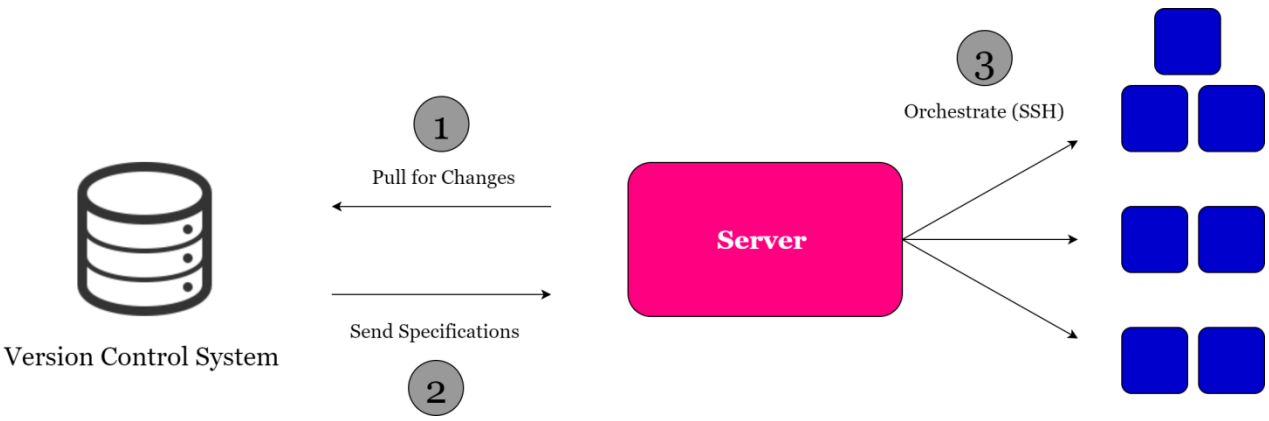



**Figure B-1: Ansible Scheme [44]**

As illustrated by Figure B.1, Ansible Server holds all the Ansible Playbooks that can be the Version Control System. Ansible commands push the changes to the other nodes with remote SSH. By using SSH, Ansible is able to run the commands in remote nodes after login. By using playbook it is easy to run a command in lots of

remote nodes. In that case, Ansible Playbook uses SSH to login to remote hosts according to the inventory file, and runs different tasks and finally reports back the status of the tasks that it was successful or not [44].

Playbook is composed of plays that are combined of modules. Ansible playbook will be executed by ansible-playbook command against target hosts. The hosts are listed in a file that is called inventory with respective their IP addresses [43].

## Ansible Playbook for Multi–Tenancy Resource Isolation

In this section, an Ansible playbook and role will be created to run Multi-Tenancy isolation in OpenStack. Before starting, it needs to be mentioned all the steps for the pre-configuration in both compute nodes and block storage mentioned in Appendix A need to be done. The other pre-configuration is to set utility_container and its IP address in controller00 in hosts file in etc/hosts.
With running the command “ansible-playbook –i hosts playbook.yml”.


Playbook will call hosts file and utility_container as the inventory and run all the roles that are dictated in tenant-isolation directory. All the commands are located in file roles/tenant-isolation/tasks/main.yml.
This project imagined the tenant is already created, with the below task the tenant ID of the project will be captured.

```
- name: getting tennat ID

  shell: openstack --os-cloud default project list | awk '/ {{
tenant_name }} / {print$2}'
  register: tent1
```

This command uses shell command with --os-cloud in order to do the authentication to run the commands as a cloud administrator.
The next two tasks create a new Host Aggregate and after that, it will register the Host Aggregate ID to use within the role.

```
- name: creating new host aggregate

  command: openstack --os-cloud default aggregate create {{
agg_name }}



 - name: getting aggregate ID
   shell: openstack --os-cloud default aggregate list | awk '/
{{ agg_name }} / {print $2}'
   register: aggre1
```

The next task adds compute nodes to the new Host Aggregate. The following task needs to be repeated for each host if cloud admin wants to add.

```
 - name: adding hosts to host aggregate
   command: openstack --os-cloud default aggregate add host {{
aggre1.stdout }} {{ item.name }}
   with_items: compute
```

The next task updates host aggregate with metadata tag by using filter_tenant_id

```
 - name: update hos aggregate metadata
   shell: source /root/openrc && nova --insecure aggregate-
set-metadata {{ aggre1.stdout }} filter_tenant_id={{
tent1.stdout }}
   args:
       executable: /bin/bash
```

Via following commands a flavor will be created to use by a tenant and with the filter_tenant_id it is possible to assign a flavor to a specific Host Aggregate. For creating a flavor we are also able to use Ansible OpenStack modules that is mentioned in the following as well.

```
 - name:  create a custom flavor
```

```
   shell: source /root/openrc && nova --insecure flavor-create
{{ flavor_name }} auto {{ flavor_ram }} {{ flavor_disk }} {{
flavor_cpu }} --is-public false

   args:
      executable: /bin/bash


    os_nova_flavor:
    cloud: default
    state: present
    name: '{{ flavor_name }}'
    ram: 1024
    vcpus: 2
    disk: 10
    is_public: no


 - name: access to custom flavor for tenant
   shell: source /root/openrc && nova --insecure flavor-
access-add {{ flavor_name }} {{ tent1.stdout }}
   args:
     executable: /bin/bash

 - name: Add flavor key to new flavor
    shell: source /root/openrc && nova --insecure flavor-key
{{ flavor_name }} set filter_tenant_id={{ tent1.stdout }}
    args:
      executable: /bin/bash
```

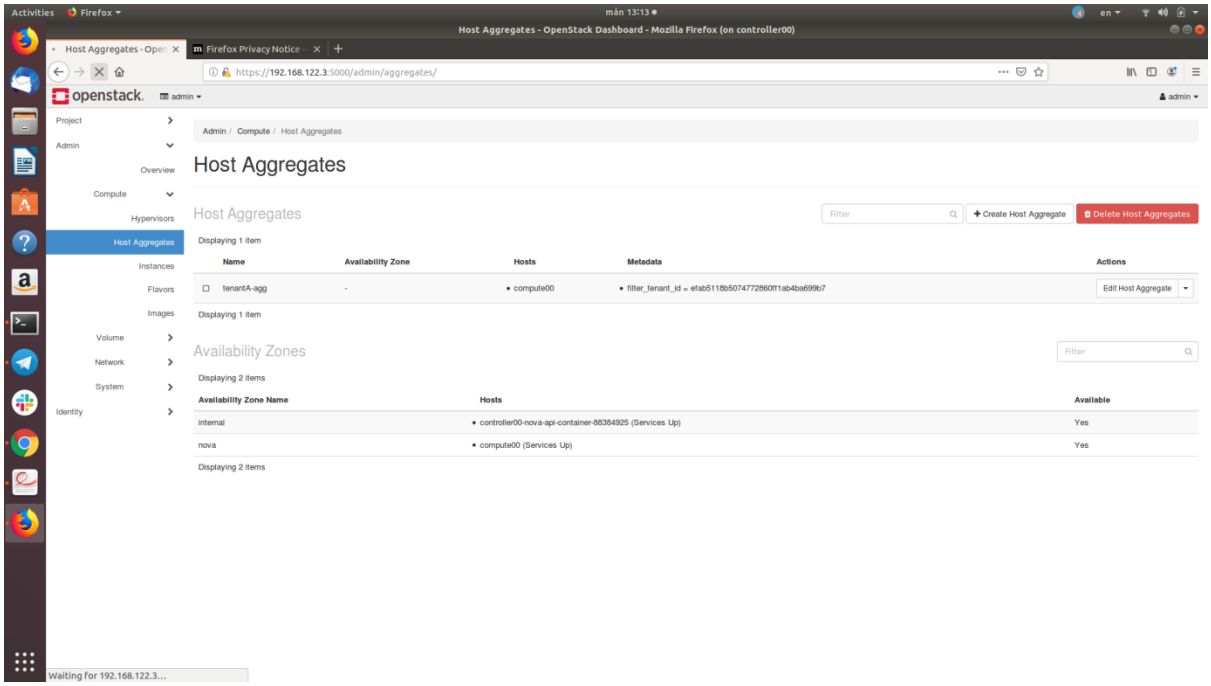
The next two tasks isolate the Cinder controlled storage to OpenStack:

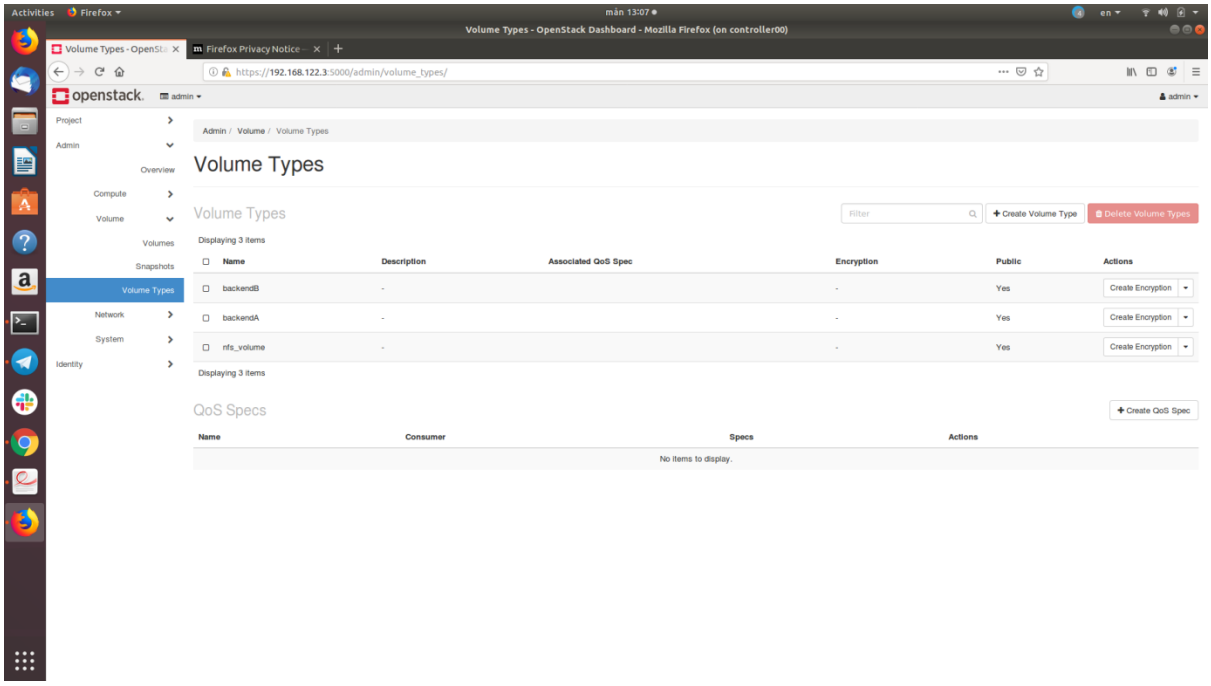
```
- name: Update default Cinder quota to allow volume type
access

  shell: source /root/openrc && cinder --insecure quota-update
--volumes 100 --volume-type backendB {{ tent1.stdout }}
    with_items: allow_vol_type
    args:
      executable: /bin/bash
    with_items: allow_vol_type


 - name: Update default Cinder quota to restrict volume type
access
   shell: source /root/openrc && cinder --insecure quota-
update --volumes 0 --volume-type backendB {{ tent1.stdout }}
   args:
     executable: /bin/bash
```

After running Ansible playbook, resource isolation is done for compute nodes and for the block storages as well. The following figures show the OpenStack Dashboard after running the Ansible Playbook.




The Ansible playbook is accessible via below Github link:
https://github.com/ali-sh1363/Multi-Tenancy-Isoaltion-OpenStack

TRITA-EECS-EX-2019:781