Project Report: Implementing OpenStack

Prepared by: Karim ELHoumaini
Supervised by: Chaker EL AMRANI

Table of Contents

Introduction

Cloud computing has become an essential part of modern computing, providing the flexibility, scalability, and reliability needed to run complex applications and services. OpenStack is an open-source cloud computing platform that offers Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) solutions. In this report, we will provide a detailed guide to implementing OpenStack and installing SaaS and IaaS.

Chaptre 1: Installing Openstack with devstack in Ubuntu server:

OpenStack is a free and open-source cloud computing platform that provides Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) solutions. In this project report, i provides a detailed guide to implementing OpenStack and installing SaaS and IaaS using Devstack in Ubuntu Server. The chapitre1 includes step-by-step instructions and screenshots for installing Ubuntu Server, configuring the network, and creating virtual machines. It also covers testing the OpenStack modules, such as Horizon, Nova, Glance, Neutron, Keystone, Cinder, and Placement.

Step 1: Installing Ubuntu Server for OpenStack Deployment

The first step in implementing OpenStack is to install Ubuntu Server on a physical or virtual machine. I recommend using Ubuntu Server 18.04 or later for this project in a virtual machine. For virtualization, I used Parallel Desktop 15.

To install Ubuntu Server, follow these steps:

1. Download the Ubuntu Server ISO file from the official website.
2. Create a bootable USB drive using the ISO file and a tool like Rufus or Etcher.
3. Insert the USB drive into the machine where you want to install Ubuntu Server and boot from the USB drive.
4. Follow the on-screen instructions to complete the installation process.

Once the installation is complete, you should have a working Ubuntu Server machine.

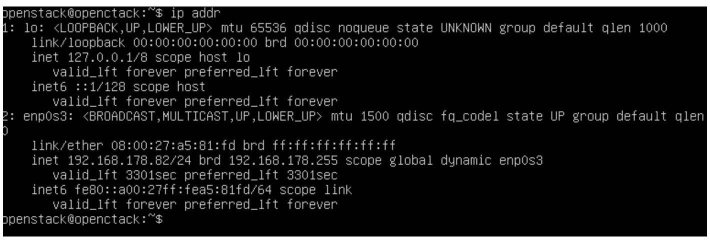
Step 2: Testing Connection Between Client and Server

Before proceeding with the OpenStack installation, it's important to test the connection between the client and server machines.

To test the connection, follow these steps:

1. Determine the IP address of the Ubuntu Server machine by running the `ifconfig` command in the terminal.
2. Determine the IP address of the client machine (e.g., macOS) by going to System Preferences > Network.
3. Open the terminal on the client machine and ping the IP address of the Ubuntu Server machine using the `ping` command.
4. If the ping is successful, use the `ssh` command to connect to the Ubuntu Server machine from the client machine.

ubuntu server address ip:



2. macOS ip address:

```
        status: inactive
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=400<CHANNEL_IO>
        ether a4:d1:8c:5e:c9:72
        inet6 fe80::104e:d156:e23a:6a82%en1 prefixlen 64 secured scopeid 0x5
        inet 192.168.178.154 netmask 0xffffff00 broadcast 192.168.178.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
```

3. pinging:

```
karim@Karims-MacBook-Pro ~ % ping 192.168.178.82
PING 192.168.178.82 (192.168.178.82): 56 data bytes
64 bytes from 192.168.178.82: icmp_seq=0 ttl=64 time=0.780 ms
64 bytes from 192.168.178.82: icmp_seq=1 ttl=64 time=0.338 ms
64 bytes from 192.168.178.82: icmp_seq=2 ttl=64 time=0.490 ms
64 bytes from 192.168.178.82: icmp_seq=3 ttl=64 time=0.551 ms
64 bytes from 192.168.178.82: icmp_seq=4 ttl=64 time=0.623 ms
64 bytes from 192.168.178.82: icmp_seq=5 ttl=64 time=0.678 ms
64 bytes from 192.168.178.82: icmp_seq=6 ttl=64 time=0.677 ms
64 bytes from 192.168.178.82: icmp_seq=7 ttl=64 time=0.789 ms
64 bytes from 192.168.178.82: icmp_seq=8 ttl=64 time=0.948 ms
64 bytes from 192.168.178.82: icmp_seq=9 ttl=64 time=0.536 ms
```

4. connecting using ssh from macOS to ubuntu:

```
karim@Karims-MacBook-Pro ~ % ssh openstack@192.168.178.82
The authenticity of host '192.168.178.82 (192.168.178.82)' can't be established.
ECDSA key fingerprint is SHA256:txBJilZHuYb8k6g3TqArCP+8cI+L1gKR6zyssbTCFOU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.178.82' (ECDSA) to the list of known hosts.
openstack@192.168.178.82's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-144-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed 08 Mar 2023 10:34:16 AM UTC

  System load:  0.0               Processes:               123
  Usage of /:   25.3% of 9.74GB   Users logged in:         1
  Memory usage: 3%                IPv4 address for enp0s3: 192.168.178.82
  Swap usage:   0%


Last login: Wed Mar  8 10:27:07 2023
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

openstack@openctack:~$
openstack@openctack:~$
```

As observed, the connection to our server machine was established successfully.

Step 3: Creating a User in Ubuntu

Next, we need to create a user in Ubuntu with sudo privileges. To create a user, follow these steps:

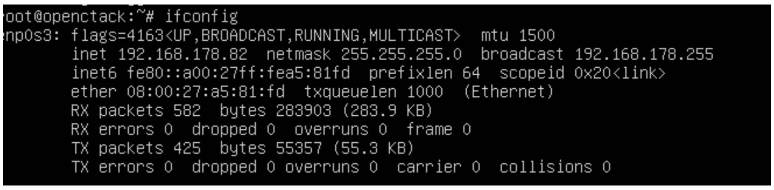
```
root@openctack:~# adduser stack
Adding user `stack' ...
Adding new group `stack' (1001) ...
Adding new user `stack' (1001) with group `stack' ...
Creating home directory `/home/stack' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for stack
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
root@openctack:~# _
```

We will create a user with no password by using the following command:

```
root@openctack:~# echo "stack ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
root@openctack:~# _
```

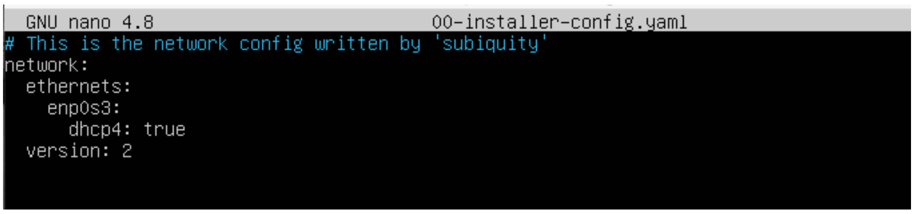
Step 4: Changing DHCP to Static IP Address

By default, Ubuntu Server is configured to use DHCP to obtain an IP address automatically. However, for OpenStack, we need to use a static IP address. To change the network configuration from DHCP to static, follow these steps:



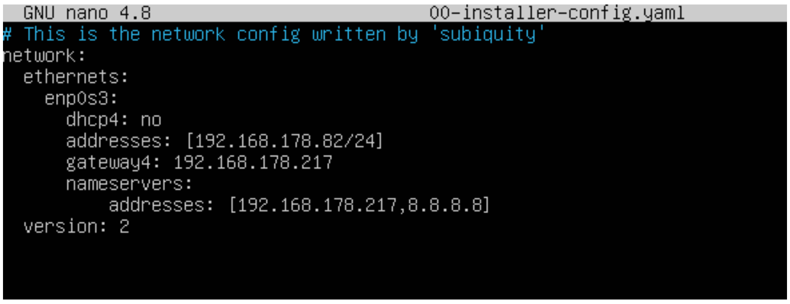
```
oot@openctack:~# ifconfig
np0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.178.82  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 fe80::a00:27ff:fea5:81fd  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:a5:81:fd  txqueuelen 1000  (Ethernet)
        RX packets 582  bytes 283903 (283.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 425  bytes 55357 (55.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Edit the netplan configuration file by running the command: `sudo vim /etc/netplan/01-netcfg.yaml`.



```
  GNU nano 4.8                    00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: true
  version: 2
```

to



```
  GNU nano 4.8                    00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.178.82/24]
      gateway4: 192.168.178.217
      nameservers:
          addresses: [192.168.178.217,8.8.8.8]
  version: 2
```

Once you have successfully created the user ‘stack’ and assigned sudo privileges, switch to the user using the command. `su - stack`

Step 5: Installing OpenStack with DevStack

Step 5 of the project report covers the installation of OpenStack using DevStack, a set of scripts and utilities that automate the process of installing and configuring OpenStack on a single machine for development purposes. The report provides a detailed guide on how to install DevStack on Ubuntu Server, configure the required settings, and start the installation process. This includes setting up the local.conf file, configuring the network settings, and starting the installation process. By the end of this step, readers will have a functional OpenStack installation on their machine for testing and development purposes.

Using git, clone devstack’s git repository as shown.

```
git clone https://git.openstack.org/openstack-dev/devstack
```

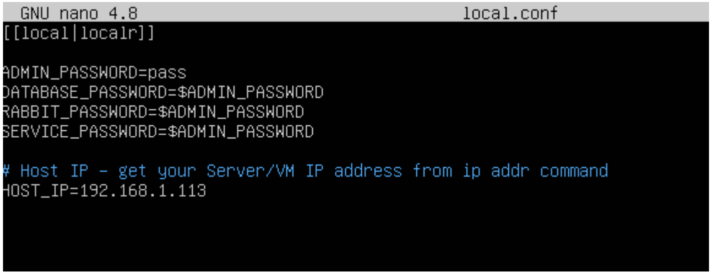
Create devstack configuration file

Inavigate to the devstack directory.

```
cd devstack
```

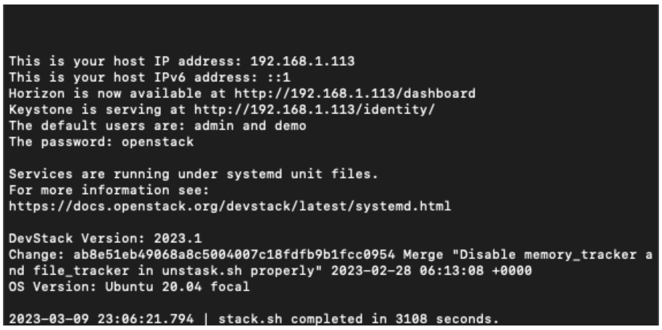
Then we create a `local.conf` configuration file.

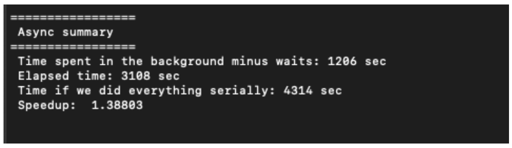
```
vim local.conf
```



launching the OpenStack installation with Devstack

To launch the OpenStack installation using Devstack, we need to execute the command "./stack.sh". During the installation process, we will be prompted to choose a password, and we recommend setting it as "openstack". After that, we need to wait for the installation to be completed.

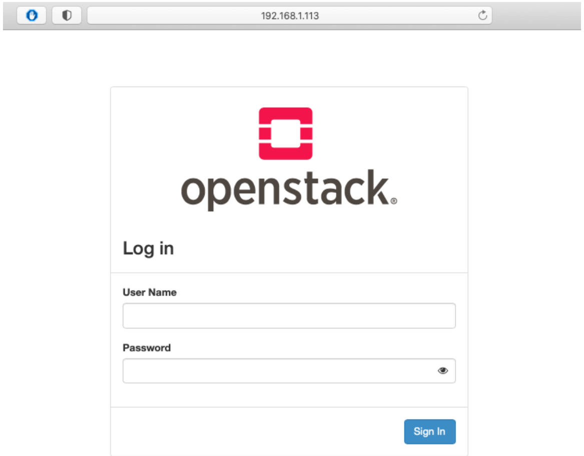




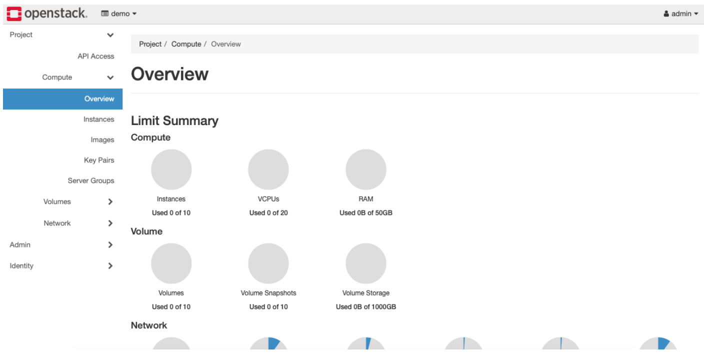
after openstack installed with the default features (modules)

- Horizon – OpenStack Dashboard
- Nova – Compute Service
- Glance – Image Service
- Neutron – Network Service
- Keystone – Identity Service
- Cinder – Block Storage Service
- Placement – Placement API

we test Horizon by login from our client machine to the address ip of the server using web browser Safari:



then we login using admin account:



Before testing all openstack modules features , let's define these modules:

- Horizon - OpenStack Dashboard: Horizon is the web-based dashboard interface for OpenStack. It provides an intuitive, graphical interface for users to manage OpenStack resources, such as instances, images, and volumes, as well as to view usage and billing information.
- Nova - Compute Service: Nova is the OpenStack service that provides virtual machine (VM) and bare metal instance management. It allows users to launch and manage instances on demand, using a range of hypervisors and operating systems.
- Glance - Image Service: Glance is the OpenStack service that provides image management functionality. It allows users to discover, register, and retrieve virtual machine images, as well as create and manage snapshots of running instances.
- Neutron - Network Service: Neutron is the OpenStack service that provides networking functionality. It allows users to create and manage virtual networks, routers, subnets, and ports, as well as to connect instances to networks.
- Keystone - Identity Service: Keystone is the OpenStack service that provides identity and authentication services. It allows users to authenticate to the OpenStack environment and manage their roles and permissions, as well as to manage service accounts and service catalog.
- Cinder - Block Storage Service: Cinder is the OpenStack service that provides block storage functionality. It allows users to create and manage persistent storage volumes and attach them to instances, as well as to create and manage volume snapshots.
- Placement - Placement API: Placement is the OpenStack service that provides the Placement API, which allows the OpenStack scheduler to find and allocate resources for instances. It manages the inventory of available resources and tracks the usage of resources by instances, in order to optimize resource allocation.

Testing:

1. Horizon has been tested by connecting to it using the web browser Safari

2. we test Glance by listing the available images in my OpenStack environment. This will confirm that the Glance image service is working properly.

```
stack@openstack:~$
stack@openstack:~$ openstack image list --os-username=admin --os-password=openst
ack --os-project-name=admin --os-project-domain-name=Default --os-user-domain-na
me=Default --os-auth-url=http://192.168.1.113/identity/v3

+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| f6605ea6-7ee3-46f8-a2dd-38c68f8980a8 | cirros-0.5.2-x86_64-disk | active |
+--------------------------------------+--------------------------+--------+
```

3. we test keystone by listing the available projects in my OpenStack environment. This will confirm that the Keystone identity service is working properly.

```
stack@openstack:~$ openstack project list --os-username=admin --os-password=open
stack --os-project-name=admin --os-project-domain-name=Default --os-user-domain-
name=Default --os-auth-url=http://192.168.1.113/identity/v3
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 6d6aa0e1c9684600bff6c972a1b1c09d | service            |
| 934daf8f50b34160b257b124b4307de8 | demo               |
| a39ac9ee6d924190adad6bf0de33c772 | invisible_to_admin |
| ce5c293d86404db7ba4de4e6623a40fc | alt_demo           |
| f683158ec4ac43c6b35daaa562c605aa | admin              |
+----------------------------------+--------------------+
```

4. we test Placement (Placement API) by listing the available hypervisors in my OpenStack environment.

```
stack@openstack:~$ openstack hypervisor list --os-username=admin --os-password=o
penstack --os-project-name=admin --os-project-domain-name=Default --os-user-doma
in-name=Default --os-auth-url=http://192.168.1.113/identity/v3
+--------------------------------------+---------------------+-----------------+
---------------+-------+
| ID                                   | Hypervisor Hostname | Hypervisor Type |
 Host IP       | State |
+--------------------------------------+---------------------+-----------------+
---------------+-------+
| 30f7731e-84a1-4cb3-a70b-4258d3541fae | openstack           | QEMU            |
 192.168.1.113 | up    |
+--------------------------------------+---------------------+-----------------+
---------------+-------+
```

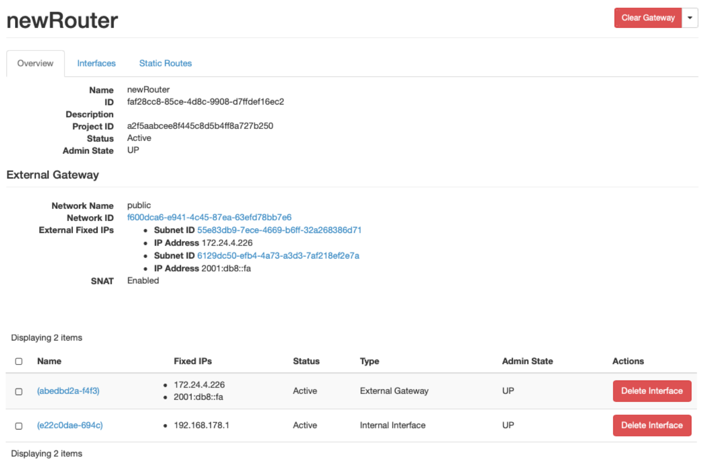
we will make sure of the state of other features by creating and implementing the IAAS and SAAS in the next chapitre :

Chapter 2: Implementation of IaaS and SaaS on OpenStack

To create instances, we need to configure the OpenStack network, router, security groups, and keypairs to allow SSH and ICMP access.

Here are the steps we followed:

1. We created a new router with two interfaces.



newRouter

Clear Gateway

Overview | Interfaces | Static Routes

Name newRouter
ID faf28cc8-85ce-4d8c-9908-d7ffdef16ec2
Description
Project ID a2f5aabcee8f445c8d5b4ff8a727b250
Status Active
Admin State UP

External Gateway

Network Name public
Network ID f600dca6-e941-4c45-87ea-63efd78bb7e6
External Fixed IPs
- **Subnet ID** 55e83db9-7ece-4669-b6ff-32a268386d71
- **IP Address** 172.24.4.226
- **Subnet ID** 6129dc50-efb4-4a73-a3d3-7af218ef2e7a
- **IP Address** 2001:db8::fa

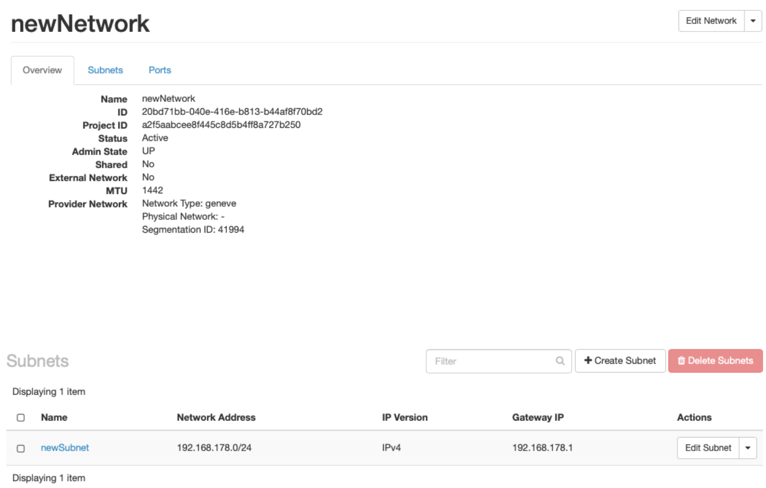
SNAT Enabled

Displaying 2 items

Name	Fixed IPs	Status	Type	Admin State	Actions
(abedbd2a-f4f3)	• 172.24.4.226 • 2001:db8::fa	Active	External Gateway	UP	Delete Interface
(e22c0dae-694c)	• 192.168.178.1	Active	Internal Interface	UP	Delete Interface

Displaying 2 items

2. Next, we created a network and a subnet.



newNetwork

Edit Network

Overview | Subnets | Ports

Name newNetwork
ID 20bd71bb-040e-416e-b813-b44af8f70bd2
Project ID a2f5aabcee8f445c8d5b4ff8a727b250
Status Active
Admin State UP
Shared No
External Network No
MTU 1442
Provider Network Network Type: geneve
Physical Network: -
Segmentation ID: 41994

Subnets

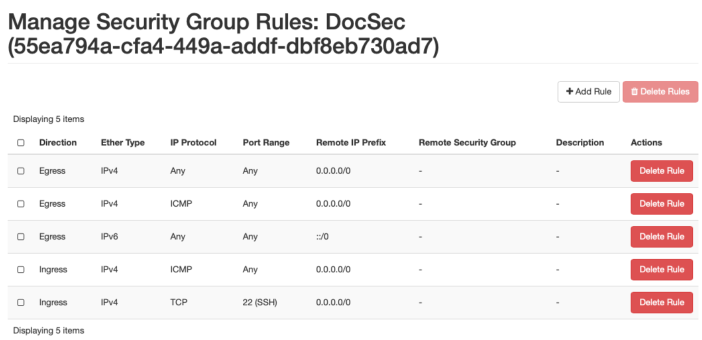
Filter | Create Subnet | Delete Subnets

Displaying 1 item

Name	Network Address	IP Version	Gateway IP	Actions
newSubnet	192.168.178.0/24	IPv4	192.168.178.1	Edit Subnet

Displaying 1 item

3. We then created a security group with rules that allow SSH and ICMP.



Manage Security Group Rules: DocSec (55ea794a-cfa4-449a-addf-dbf8eb730ad7)

Add Rule | Delete Rules

Displaying 5 items

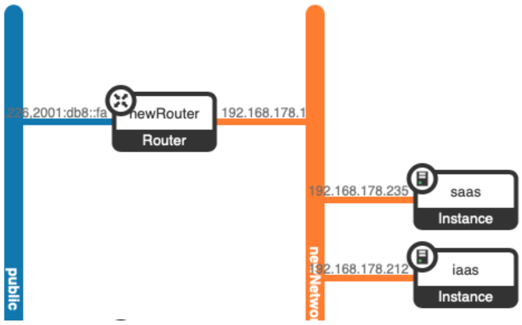
Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
Egress	IPv4	Any	Any	0.0.0.0/0	-	-	Delete Rule
Egress	IPv4	ICMP	Any	0.0.0.0/0	-	-	Delete Rule
Egress	IPv6	Any	Any	::/0	-	-	Delete Rule
Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	-	Delete Rule
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Delete Rule

Displaying 5 items

4. Finally, we allocated a new floating IP to our IaaS and SaaS instances.

172.24.4.220	iaas 192.168.178.212	public	Active	Disassociate
172.24.4.85	saas 192.168.178.235	public	Active	Disassociate

5. We then reviewed the results in the network topology.



Implementing IaaS :

IaaS stands for Infrastructure-as-a-Service. This refers to the provisioning of infrastructure resources (such as compute, storage, and networking) on a pay-as-you-go basis. In this report, the creation of a virtual machine on OpenStack is an example of IaaS

To create an IaaS, we followed these steps:

1. We created an instance called IaaS based on the Cirros image.

☐	iaas	cirros-0.5.2-x86_64-disk	192.168.178.212, 172.24.4.220	m1.tiny	-	Active		nova	None	Running	3 days, 2 hours	Create Snapshot ▾

2. We configured the instance settings with the following configuration:

Specs

Flavor Name	m1.tiny
Flavor ID	1
RAM	512MB
VCPUs	1 VCPU
Disk	1GB

IP Addresses

newNetwork	192.168.178.212, 172.24.4.220

Security Groups

DocSec	ALLOW IPv4 icmp to 0.0.0.0/0 ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv4 icmp from 0.0.0.0/0 ALLOW IPv6 to ::/0 ALLOW IPv4 22/tcp from 0.0.0.0/0
default	ALLOW IPv4 icmp to 192.168.178.154/24 ALLOW IPv4 from default ALLOW IPv6 to ::/0 ALLOW IPv4 22/tcp from 192.168.178.154/24 ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv6 from default

3. We attempted to connect to it using SSH from our machine.

```
stack@openstack:~$ ssh cirros@172.24.4.220
cirros@172.24.4.220's password:
$
```

Implementing SaaS :

SaaS refers to the creation of a custom webpage on the Apache server that simulates a Software-as-a-Service (SaaS) application. This is done by configuring the default webpage located in the directory /var/www/html/ to include forms, buttons, and other interactive elements. The resulting webpage can then be accessed by users and provides them with a simulated SaaS experience.

Here are the steps we followed:

1. We started by creating an image based on the Ubuntu server. This involved selecting the Ubuntu server as the base image and configuring any necessary settings. Once the image was created, we named it "ubuntu".

☐	› admin	ubuntu	Image	Active	Public	No	QCOW2	251.63 MB	Launch ▾

2. Next, we created an instance from the ubuntu Image that we had just created. We named this instance "SaaS " and configured its settings as needed.

☐	saas	ubuntu	192.168.178.235, 172.24.4.85	ds1G	key	Active		nova	None	Running	2 days, 14 hours	Create Snapshot ▾

Using the specified settings

Specs

Flavor Name	ds1G
Flavor ID	d2
RAM	1GB
VCPUs	1 VCPU
Disk	10GB

IP Addresses

newNetwork	192.168.178.235, 172.24.4.85

Security Groups

DocSec	ALLOW IPv4 icmp to 0.0.0.0/0 ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv4 icmp from 0.0.0.0/0 ALLOW IPv6 to ::/0 ALLOW IPv4 22/tcp from 0.0.0.0/0

3. Once the SaaS Instance was up and running, we connected to it via ssh from our local machine. This involved opening up a terminal window and using the ssh command to connect to the instance's IP address.

```
stack@openstack:~$ ssh -i key.pem ubuntu@172.24.4.85
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 3.13.0-169-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Thu Mar 16 01:07:53 UTC 2023

  System load:  0.77              Processes:           70
  Usage of /:   8.5% of 9.81GB    Users logged in:     0
  Memory usage: 5%                IP address for eth0: 192.168.178.235
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

0 updates can be installed immediately.
0 of these updates are security updates.


Last login: Thu Mar 16 01:08:08 2023 from 172.24.4.1
```

then we create a custom webPage that simulate a SaaS, Here are the general steps i followed :

1. After the installation was complete, we started the Apache service by running the command "sudo systemctl start apache2" in the terminal. This launched the web server and made it available for use.
2. With the web server up and running, we then navigated to the default web page directory located at "/var/www/html/". Here, we customized the default web page to simulate a SaaS application by including forms, buttons, and other interactive

elements. We used HTML, CSS, and JavaScript to build the custom web page.

3. Finally, we tested the custom web page by navigating to the instance's IP address in a web browser. We were able to interact with the web page and confirm that it was functioning correctly.

> Note : Due to the lack of a graphical interface on the Ubuntu server, we were unable to test the web page after installation. To overcome this limitation, there are several options available. One way is to connect to the instance from another machine with internet access. Alternatively, we can install a web browser that runs on a terminal or use a terminal-based SaaS, such as Tmux. Tmux is a terminal multiplexer that enables you to run multiple terminal sessions within a single window. It also allows you to detach and re-attach sessions, ensuring that you can continue working on them even if you disconnect from the server or shut down your computer.

Chapter 3:

Chapter 3 of this project report covers the simulation of a cloud environment on OpenStack using CloudSim. The goal of this chapter is to answer several key questions, including the number of physical hosts present in the OpenStack cloud environment, the number of virtual machines running on each physical host, the amount of RAM and CPU capacity available for each physical host, and any specific performance or optimization requirements for the simulation. By the end of this chapter, readers will have a better understanding of how to simulate a cloud environment on OpenStack using CloudSim for testing and development purposes.

Simulating our Cloud envirenemt using CloudSim:

In order to simulate a cloud environment on OpenStack using CloudSim, several questions need to be answered first. These questions include:

1. How many physical hosts (servers) are present in the OpenStack cloud environment?
2. How many virtual machines are currently running on each physical host?
3. What is the amount of RAM and CPU capacity available for each physical host?
4. Are there any specific performance or optimization requirements for the simulation?

After determining the necessary information, the simulation can be created. In this particular scenario :

1. 1 physical host which is an Ubuntu server with 7GB of RAM, 2 CPUs, and 25GB of storage
2. There are 2 virtual machines running on this physical host, Including:
3. Cirrus with 512MB of RAM, 1 CPU, and 1GB of storage,
4. Ubuntu LTS with 1GB of RAM, 1 CPU, and 10GB of storage.

so the CloudSim code will be as follow :

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class OpenStackCloudSimulation {
    private static final int NUM_HOSTS = 1; // Number of physical hosts
    private static final int NUM_VMS = 2; // Number of virtual machines
    private static final int RAM_SIZE = 7000; // Amount of RAM in the physical host in MB
```

```
    private static final int CPU_MIPS = 2000; // MIPS capacity of the physical host
    private static final int BANDWIDTH = 100000; // Bandwidth between hosts in Mbps

    public static void main(String[] args) {
        try {
            int num_user = 1;   // number of cloud users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace GridSim events

            CloudSim.init(num_user, calendar, trace_flag);

            // Create datacenter
            Datacenter datacenter = createDatacenter("OpenStack");

            // Create broker
            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();

            // Create virtual machines
            List<Cloudlet> cloudletList = new LinkedList<>();
            List<Vm> vmList = new ArrayList<>();
            for(int i=0; i<NUM_VMS; i++) {
                Vm vm = createVM(i, brokerId);
                vmList.add(vm);

                // Create cloudlets and bind them to the VMs
                Cloudlet cloudlet = createCloudlet(i, brokerId);
                cloudlet.setVmId(vm.getId());
                cloudletList.add(cloudlet);
            }

            broker.submitVmList(vmList);
            broker.submitCloudletList(cloudletList);

            CloudSim.startSimulation();

            List<Cloudlet> newList = broker.getCloudletReceivedList();
            CloudSim.stopSimulation();

            printCloudletList(newList);

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error in simulation");
        }
    }

    private static Datacenter createDatacenter(String name) {
        List<Host> hostList = new ArrayList<>();

        // Create the host
        for(int i=0; i<NUM_HOSTS; i++) {
            List<Pe> peList = new ArrayList<>();
            peList.add(new Pe(0, new PeProvisionerSimple(CPU_MIPS)));

            Host host = new Host(i, new RamProvisionerSimple(RAM_SIZE), new BwProvisionerSimple(BANDWIDTH), 250000, peList, new VmSche
            hostList.add(host);
        }

        String arch = "x86";      // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
        double time_zone = 10.0;         // time zone this resource located
        double cost = 3.0;              // the cost of using processing in this resource
        double costPerMem = 0.05;        // the cost of using memory in this resource
        double costPerStorage = 0.1;
        double costPerBw = 0.1; // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<>(); // we are not adding SAN devices by now
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os, vmm, hostList, time_zone, cost, costPerMem, costPe

    // Finally, create the datacenter
    Datacenter datacenter = null;
    try {
        datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return datacenter;
}

private static DatacenterBroker createBroker() {
    DatacenterBroker broker = null;
    try {
        broker = new DatacenterBroker("Broker");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
```

```
    }
    return broker;
}

private static Vm createVM(int id, int brokerId) {
    int mips = CPU_MIPS;
    long size = 10000; // image size (MB)
    int ram = 1024; // vm memory (MB)
    int bw = 1000;

    Vm vm = new Vm(id, brokerId, mips, 1, ram, bw, size, "Xen", new CloudletSchedulerTimeShared());
    return vm;
}

private static Cloudlet createCloudlet(int id, int brokerId) {
    long length = 40000; // cloudlet length
    int pesNumber = 1;
    long fileSize = 300;
    long outputSize = 300;
    UtilizationModel utilizationModel = new UtilizationModelStochastic();

    Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel
    cloudlet.setUserId(brokerId);
    return cloudlet;
}

private static void printCloudletList(List<Cloudlet> list) {
    int size = list.size();
    Cloudlet cloudlet;
    String indent = "    ";
    System.out.println();
    System.out.println("========== OUTPUT ==========");
    System.out.println("Cloudlet ID" + indent + "STATUS" + indent + "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "

    DecimalFormat dft = new DecimalFormat("###.##");
    for (int i = 0; i < size; i++) {
        cloudlet = list.get(i);
        System.out.print(indent + cloudlet.getCloudletId() + indent + indent);

        if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
            System.out.print("SUCCESS");

            System.out.println(indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() + indent + i
        }
    }
}
}
```

Output:

```
Run:  OpenStackCloudSimulation
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        20        0.1        20.1

Process finished with exit code 0
```

Conclusion

In conclusion, this project report has provided an overview of the implementation of OpenStack, including the various components of the platform and their functions. The report has also covered the testing of OpenStack, the implementation of IaaS and SaaS, and the simulation of a cloud environment using CloudSim. Through this report, we have demonstrated the capabilities of OpenStack and its potential for use in various industries. The use of CloudSim has also highlighted the importance of simulation in testing and development. Overall, the implementation of OpenStack has shown great promise in advancing cloud computing technology and providing a more efficient and flexible infrastructure for organizations.