



Concordia Institute of Information Systems Engineering (CIISE)

INSE 6620 – Cloud Computing Security and Privacy

Project Report

OpenStack-Based Cloud Deployment and Network Security with Snort Detection

Submitted to:

Prof. Lingyu Wang

Submitted by:

Name	Student ID
Anab Abdi	40259956
Aniket Agarwal	40266485
Bhavika Kaura	40294564
Insha Singh	40258129
Kalyani Batle	40243967
Naman Verma	40294281

Summer 2024

1.0. Introduction

1.1. Abstract

This report details the installation and configuration of OpenStack, an open-source cloud computing platform having a single-node architecture, using DevStack, with a focus on enhancing security through the integration of Snort, an Intrusion Detection and Prevention System (IDPS) [1]. The project began by setting up Ubuntu 22.04 LTS on VirtualBox, followed by configuring DevStack to install the latest OpenStack version. After resolving authentication issues during the addition of an Ubuntu 16.04 Xenial image, virtual instances named "victim" and "attacker" were created to simulate network scenarios and test Snort's detection capabilities. Security groups were configured for ICMP and SSH access, and Snort was installed on the victim instance to monitor and detect attacks, such as TCP SYN flood, using custom rules. The project setup included detailed hardware and software specifications, and a meticulously designed networking environment with public, and private networks connected via a router. This report demonstrates the practical application of OpenStack in building a secure cloud environment, highlighting the importance of integrating robust security tools like Snort to safeguard against cyber threats.

1.2. Introduction to OpenStack

OpenStack is an open-source cloud computing platform that enables users to deploy and manage large-scale cloud environments. Key components in OpenStack include:

- **Keystone:** Provides identity services for user authentication and authorization.
- **Glance:** Manages and stores images for virtual machines.
- **Nova:** Handles computing resources and virtual machine management.
- **Placement:** Tracks resource inventories and manages resource allocation.
- **Cinder:** Provides block storage services.
- **Neutron:** Manages networking services and connectivity.
- **Horizon:** Offers a web-based dashboard to manage cloud services.

These components work together to deliver a robust and scalable cloud environment, facilitating easier application deployment and management while minimizing the need for physical hardware [1][2].

2.0. Installing OpenStack and deploying a virtual network

To install OpenStack, we utilized DevStack, a collection of scripts designed to set up a complete OpenStack environment based on the latest versions. The process began with the installation of *Ubuntu 22.04 LTS* on VirtualBox.

We started by creating a non-root user with sudo privileges (*stack* user). Following this, we installed DevStack by cloning the repository using the command:

```
git clone https://opendev.org/openstack/devstack
```

Next, we created a local.conf file at the root of the DevStack repository to preset essential passwords and configurations. The configuration is shown in the figure below:

```

stack@ubuntu-1: ~/devstack
ubuntu@victim: ~/snort_src/snort-2.9.20
GNU nano 6.2 local.conf
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=10.0.2.15

```

After configuring the ‘local.conf’ file, we initiated the installation by running the ‘./stack.sh’ script. Upon completion, we accessed the OpenStack dashboard via the ‘HOST_IP’ specified in the configuration file, using the username ‘admin’ and password defined in the ‘local.conf’.

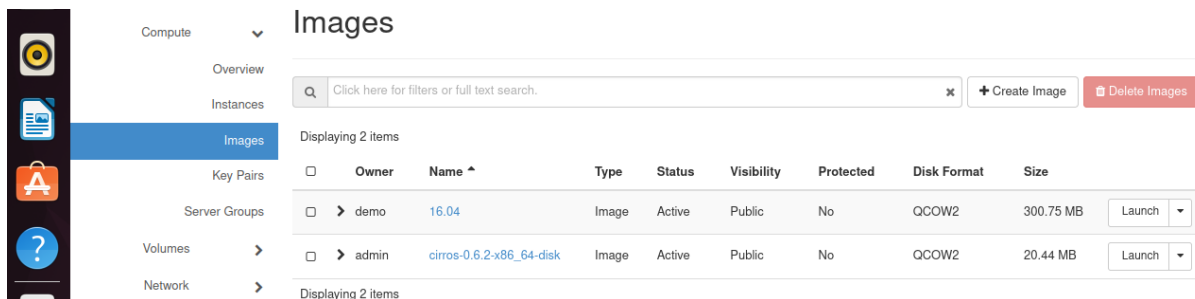
Although a Cirros image was already installed by default, we required an Ubuntu image for our instances to support tools like ‘hping3’ and ‘snort’. We downloaded the Ubuntu 16.04 Xenial image using the following command:

`wget https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img`

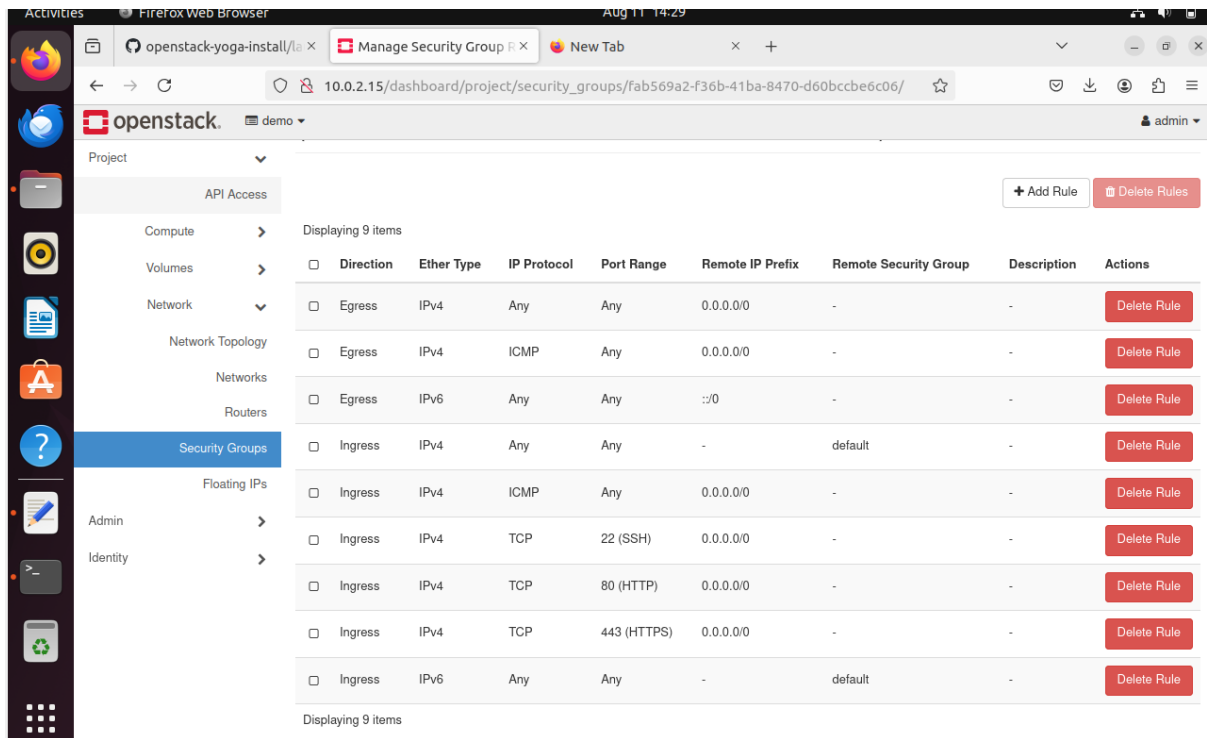
We stored this image in the OpenStack Glance image service using the following command:

`openstack --insecure image create --disk-format qcow2 --min-disk 8 --min-ram 512 --file xenial-server-cloudimg-amd64-disk1.img --public 16.04`

During this process, we encountered an error stating "missing value of URL required for auth-plugin password." To resolve this, we authenticated through the ‘openrc’ file by downloading it from the dashboard, saving its contents to a file, and sourcing it to authenticate. After reattempting the image installation, it succeeded.



With the Ubuntu image in place as shown in the above figure, we proceeded to create two instances named "victim" and "attacker." After launching the instances, we edited the ‘default’ security group responsible for traffic control by adding two rules: the first, an ICMP rule to allow ping and network-level connectivity, and the second, an SSH rule to enable access to the instances through port 22 via the SSH key created during the instance setup as shown in the below figure.



Once the instances were operational, we installed Snort on the victim instance to detect any attacks. To monitor for TCP SYN flood attacks, we wrote a custom rule that checks incoming packets, providing a robust security measure within the cloud environment.

We used single laptop to setup the OpenStack using VirtualBox and their specifications are as follows:

2.1. Hardware Specifications:

- Processor: 11th Gen Intel® Core™ i7-11800H, 2.30 GHz
- RAM: 32 GB
- Storage: 1 TB

2.2. Software Specifications:

- Virtualization Platform: VirtualBox was used to launch the Ubuntu 22.04 LTS operating system.
- Processors Allocated: 6
- Base Memory: 14 GB
- Storage Allocated: 50 GB
- IP address allocated to the instance running Ubuntu 22.04: 10.0.2.15

We used nested virtualization and hence we enabled hardware acceleration, nested paging and paravirtualization in VirtualBox settings. The specifications of OpenStack instances are as follows

2.3. OpenStack Instances Specifications:

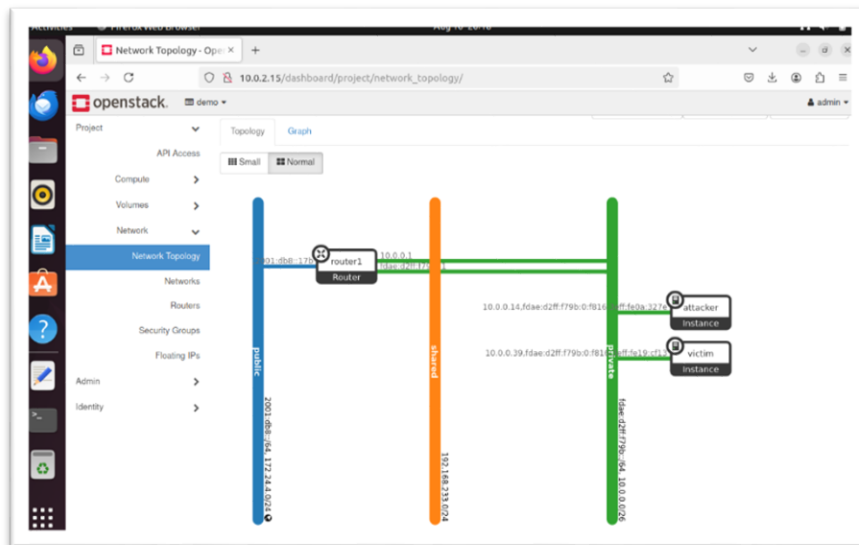
- 10.0.2.15 - horizon IP
- 2 instances-victim and attacker

Specifications	Victim	Attacker
Type	m1.small	ds1G
RAM	2GB	1 GB
Disk	20GB	10GB

VCPU	1	1
Floating IP	172.24.4.39	172.24.4.106
IP Address	10.0.0.39	10.0.0.14
Image Running	Ubuntu 16.04 LTS	Ubuntu 16.04 LTS

2.4. OpenStack Router (router1):

- Acts as the bridge between different networks (Public and Private).



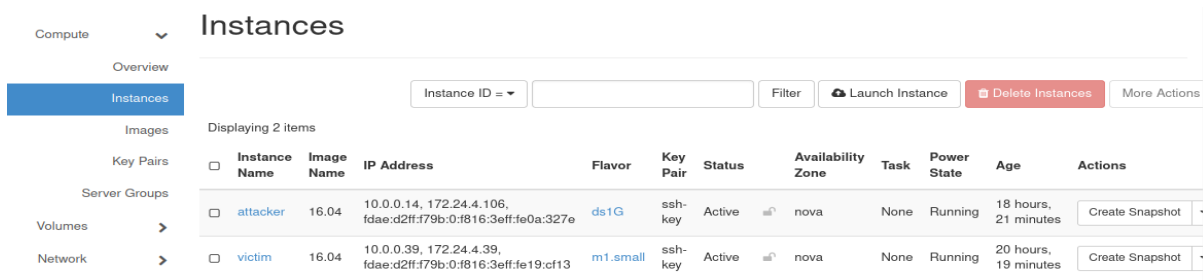
- It's connected to the public network (2001:db8::/64, 172.24.0/24) and the private network (fd4e:d2ff:f79b::/64, 10.0.0.0/26).

2.4. Networks:

- Public Network (blue):** Connected to router1 and accessible from the external internet.
- Shared Network (orange):** A network that may be used by multiple instances or tenants.
- Private Network (green):** This network is typically isolated and used for internal communication between instances.

2.5. Instances :

- Attacker Instance:** Connected to the private network (10.0.0.14).
- Victim Instance:** Connected to the private network (10.0.0.39).



The screenshot shows the OpenStack dashboard's 'Instances' page. On the left, a sidebar lists navigation options: Compute (selected), Overview, Instances, Images, Key Pairs, Server Groups, Volumes, and Network. The main content area shows a table of instances. Two instances are listed: 'attacker' and 'victim'. Both are in the 'Active' state, running on 'nova' availability zone, with 'None' task and 'Running' power state. The 'attacker' instance is 18 hours and 21 minutes old, while the 'victim' is 20 hours and 19 minutes old. Both have a 'Create Snapshot' action available.

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
attacker	16.04	10.0.0.14, 172.24.4.106, fd4e:d2ff:f79b:0:1816:3eff:fe0a:327e	ds1.G	ssh-key	Active	nova	None	Running	18 hours, 21 minutes	Create Snapshot
victim	16.04	10.0.0.39, 172.24.4.39, fd4e:d2ff:f79b:0:1816:3eff:fe19:cf13	m1.small	ssh-key	Active	nova	None	Running	20 hours, 19 minutes	Create Snapshot

The setup in the above figure shows a typical scenario where both an attacker and victim are in the same private network, which might be used to simulate attacks and test the detection capabilities of Snort. The shared and public networks allow for communication outside of the private network, and the router facilitates the traffic routing between these networks.

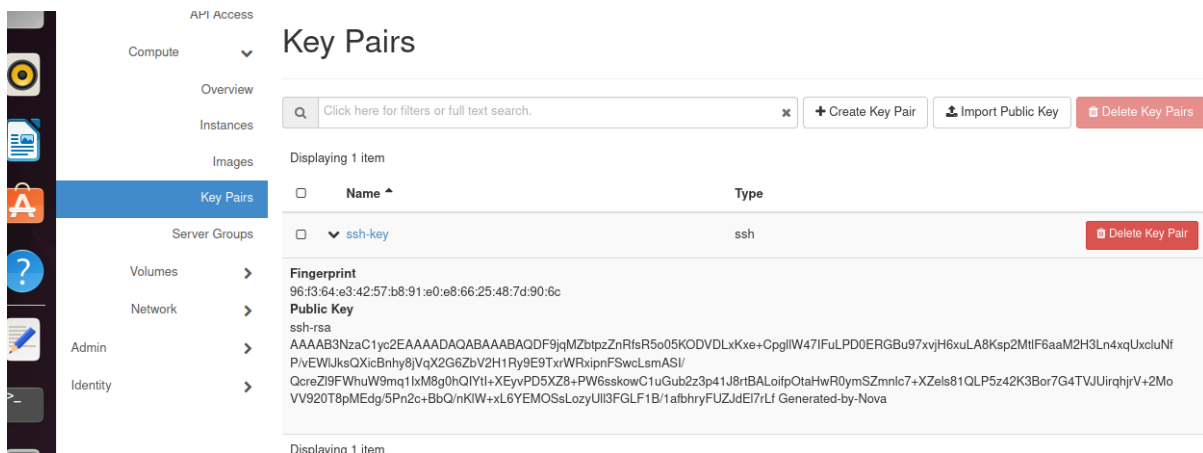
3.0 Attack Scenario:

In our setup, we simulated an attack using two instances: an attacker and a victim. The attack was executed as follows:

3.1 Establishing Connection:

OpenSSH Setup: We first ensured that OpenSSH was properly configured on both the attacker and victim instances to allow for secure remote connections.

Connection: We stored the public key generated by OpenStack using the key/pair feature to Ubuntu 22.04 machine and then we connected to each OpenStack instance using this key to verify connectivity and configure the environment for the attack and detection.



The screenshot shows the OpenStack dashboard's 'Key Pairs' page. The sidebar on the left is the same as the previous screenshot. The main content area shows a table with one key pair named 'ssh-key'. The key pair is of type 'ssh'. Below the table, the fingerprint and public key are displayed. The fingerprint is '96:33:64:e3:42:57:b8:91:e0:e8:66:25:48:7d:90:6c'. The public key is 'AAAAB3NzaC1yc2EAAAADAQABAAQDAF9jgMZbtpzZnRtsR5o05KODVDLxKxe+CpgllW47lFuLPD0ERGBu97xvjH6xuLA8Ksp2MtlF6aaM2H3Ln4xqUxcluNfPivEWJksQXicBnhy8jVqX2G6ZbV2H1Ry9E9TxrWRxipnFSwLsmASI/QcreZi9FWhuW9mq1lxM8g0hQIYt1+XEyvPD5XZ8+PW6sskowC1uGub2z3p41J8rtBALoitpOtaHwR0ymSZmnc7+XZels81QLP5z42K3Bor7G4TVJUIrqhjrV+2MoVV920T8pMEdg/5Pn2c+BbQ/nKIW+xl6YEMOSsLozyUII3FGLF1B/1atbhryFUZjdEi7rLf Generated-by-Nova'.

Name	Type
ssh-key	ssh

Fingerprint
96:33:64:e3:42:57:b8:91:e0:e8:66:25:48:7d:90:6c

Public Key
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDAF9jgMZbtpzZnRtsR5o05KODVDLxKxe+CpgllW47lFuLPD0ERGBu97xvjH6xuLA8Ksp2MtlF6aaM2H3Ln4xqUxcluNfPivEWJksQXicBnhy8jVqX2G6ZbV2H1Ry9E9TxrWRxipnFSwLsmASI/QcreZi9FWhuW9mq1lxM8g0hQIYt1+XEyvPD5XZ8+PW6sskowC1uGub2z3p41J8rtBALoitpOtaHwR0ymSZmnc7+XZels81QLP5z42K3Bor7G4TVJUIrqhjrV+2MoVV920T8pMEdg/5Pn2c+BbQ/nKIW+xl6YEMOSsLozyUII3FGLF1B/1atbhryFUZjdEi7rLf Generated-by-Nova

=> `ssh -i <ssh_key> user@<ip address>`

In the above command -

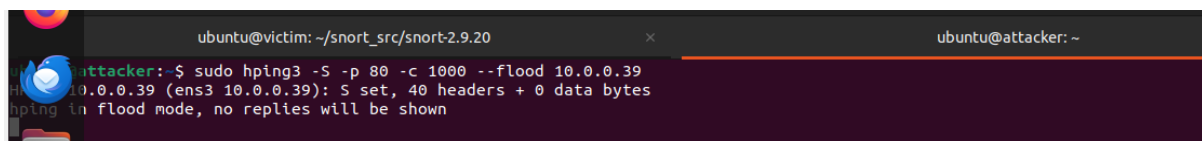
- *ssh_key* is the respective public key generated by OpenStack to connect to the instances whose private key is connected with the instances by the OpenStack using key/pair feature.
- *user* is the hostname of the instance to connect with.
- *ip_address* is the ip of the instance.

3.2 Launching the Attack:

Attack Tool: We used hping3, a network tool for crafting and analyzing TCP/IP packets, to simulate a TCP SYN flood attack.

Attack Description: We used TCP SYN flood attack which falls under the category of Denial-of-Service attack. TCP uses 3-way handshake mechanism to establish a connection between a client and a server which follows a SYN, SYN-ACK, ACK packets in the order. Now to perform the attack, we used hping3 to continuously flood the server (victim in this case) with SYN packets without waiting to reply for these packets which eventually increased the load on server and denied the server from serving the requests from any other source.

SYN Flood: From the attacker instance, we executed hping3 to generate a high volume of SYN packets targeting the victim instance.



```
ubuntu@victim: ~/snort_src/snort-2.9.20
ubuntu@attacker: ~
attacker:~$ sudo hping3 -S -p 80 -c 1000 --flood 10.0.0.39
hping3 10.0.0.39 (ens3 10.0.0.39): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

In the above figure:

- **-S** flag is used to specify hping3 to send only SYN packets to target 10.0.0.39
- **-p** is used to specify port which is 80
- **-c** is used to specify the packets count per second
- **--flood** is used to enter flood mode and continuously flood the target with SYN packets without waiting for replies.

From the above figure we can see that the attack was launched successfully from the attacker's instance to the victim's instance.

4.0 Detection Scenario:

Step 1: Installation of Prerequisites

We updated our system and installed necessary tools and libraries such as build-essential, libpcap-dev, libpcrc3-dev, libdumbnet-dev, zlib1g-dev, bison, flex, and libssl-dev [6].

Step 2: Installation of DAQ (Data Acquisition library)

We installed the DAQ source code from the official Snort website, extracted the tar file, compiled and installed DAQ on our victim instance. [6]

Step 3: Installation of Snort

We downloaded the Snort source code, extracted it, and configured the installation with the --enable-sourcefire option. Then we compiled and installed Snort.

Step 4: Configure Snort

We then created the necessary directories for Snort, such as `/etc/snort`, `/etc/snort/rules`, `/etc/snort/preproc_rules`, `/var/log/snort`, and `/usr/local/lib/snort_dynamicrules` and set up a user and group for Snort, and change the ownership of these directories to the Snort user [6].

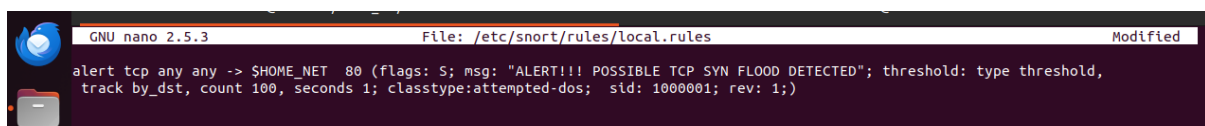
We edited the Snort configuration file located at `/etc/snort/snort.conf` to define paths, network variables, and rules according to our network setup.

Step 5: Test and Run Snort

We validated the Snort configuration using the command to check for any errors.

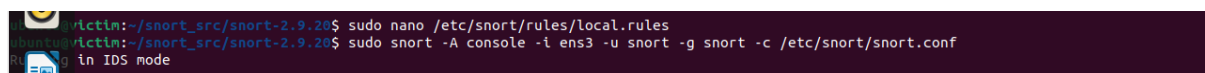
Step 6: Create and Manage Rules

We created our own custom rules in the `/etc/snort/rules/local.rules` directory [6].



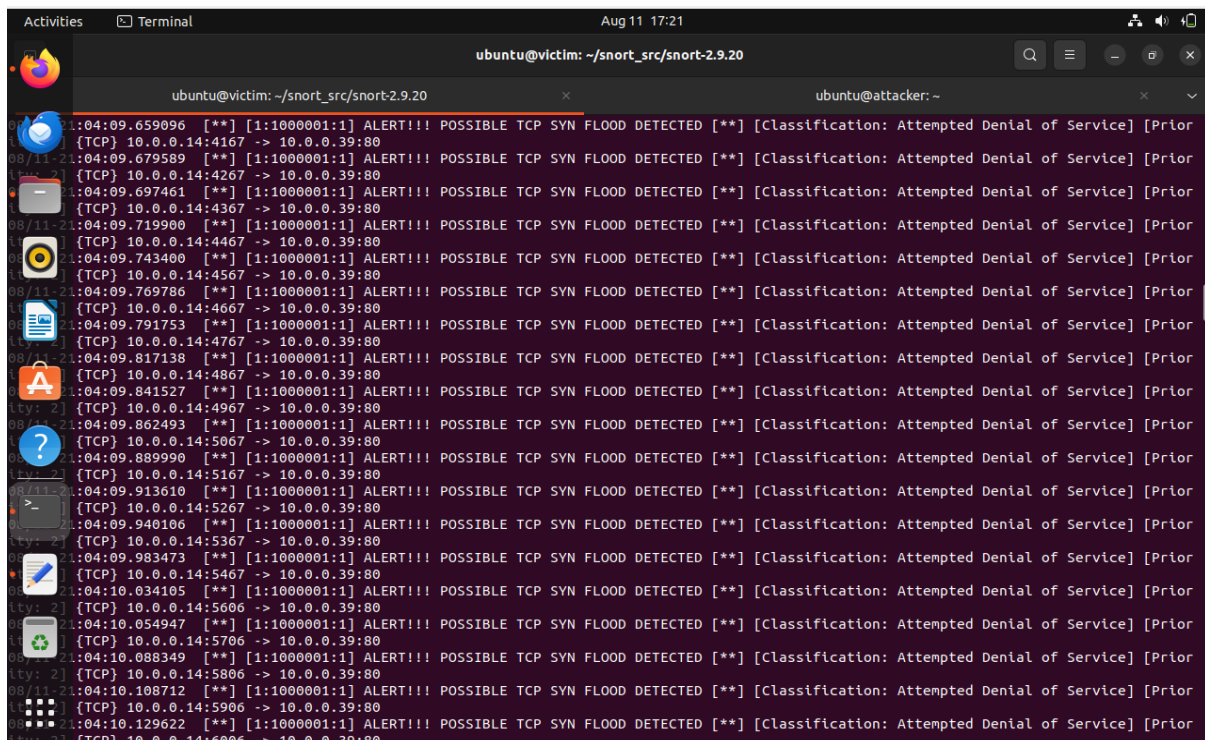
```
GNU nano 2.5.3 File: /etc/snort/rules/local.rules Modified
alert tcp any any -> $HOME_NET 80 (flags: S; msg: "ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED"; threshold: type threshold,
track by_dst, count 100, seconds 1; classtype:attempted-dos; sid: 1000001; rev: 1;)
```

The above SNORT rule listens for any tcp SYN packets coming from any source ip address to the victim on port 80 and starts alerting the victim when it detects packet count of 100 or more per second with respective alert message.



```
ubuntu@victim:~/snort_src/snort-2.9.20$ sudo nano /etc/snort/rules/local.rules
ubuntu@victim:~/snort_src/snort-2.9.20$ sudo snort -A console -i ens3 -u snort -g snort -c /etc/snort/snort.conf
in IDS mode
```

The above command enable SNORT to listen for incoming packet traffic on interface 'ens3' and outputs alert messages if any on console.



```
Aug 11 17:21
ubuntu@victim: ~/snort_src/snort-2.9.20
ubuntu@victim:~/snort_src/snort-2.9.20
[04:09.659096] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4167 -> 10.0.0.39:80
[04:09.679589] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4267 -> 10.0.0.39:80
[04:09.697461] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4367 -> 10.0.0.39:80
[04:09.719900] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4467 -> 10.0.0.39:80
[04:09.743400] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4567 -> 10.0.0.39:80
[04:09.769786] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4667 -> 10.0.0.39:80
[04:09.791753] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4767 -> 10.0.0.39:80
[04:09.817138] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4867 -> 10.0.0.39:80
[04:09.841527] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:4967 -> 10.0.0.39:80
[04:09.862493] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5067 -> 10.0.0.39:80
[04:09.889990] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5167 -> 10.0.0.39:80
[04:09.913610] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5267 -> 10.0.0.39:80
[04:09.940106] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5367 -> 10.0.0.39:80
[04:09.983473] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5467 -> 10.0.0.39:80
[04:10.034105] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5567 -> 10.0.0.39:80
[04:10.054947] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5667 -> 10.0.0.39:80
[04:10.088349] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5767 -> 10.0.0.39:80
[04:10.108712] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5867 -> 10.0.0.39:80
[04:10.129622] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:5967 -> 10.0.0.39:80
[04:10.149990] [**] [1:1000001:1] ALERT!!! POSSIBLE TCP SYN FLOOD DETECTED [**] [Classification: Attempted Denial of Service] [Priority: High]
[TCP] 10.0.0.14:6067 -> 10.0.0.39:80
```


The above figure displays the alert messages of Snort we received when we performed the attack described on the victim instance. Snort successfully detected the TCP SYN flood attack.

5.0 Challenges and Solution

During the initial setup of the OpenStack environment, we encountered several issues that required a shift in the approach to successfully deploy and configure the cloud infrastructure.

5.1. Initial Setup Challenges with CentOS:

The initial goal of our project was to install OpenStack on CentOS in a VirtualBox environment. However, we soon encountered a major compatibility issue: CentOS could not support the network infrastructure required for OpenStack, mainly due to handling network services

The biggest challenge is that CentOS needs to disable NetworkManager to make OpenStack's network components work properly. Unfortunately, disabling NetworkManager on CentOS proved problematic as in the latest version required network services (such as network.service) were either unavailable or not working as expected. This caused problems with the network connectivity needed to provide OpenStack has been out of order, resulting in a connection failure.

Despite several attempts to manually configure network configuration and force the system to revert to default network configuration the configuration remained unstable. Lack of a properly configured network environment prevented us from deploying OpenStack implemented correctly, as the underlying network services required for communication between the controller, computer and storage and the nodes are not available

Due to these ongoing issues, we decided to move to Ubuntu 22.04 LTS, which offers better support for both OpenStack and DevStack. Ubuntu's network management tools are closely aligned with OpenStack's requirements, allowing deployment to continue without the network configuration issues we encountered with CentOS

5.2. Challenges with Glance Image and Snort Installation:

Having successfully switched to Ubuntu, we ran into another set-up problem. Initially, we used Cirros images in Glance for our examples. Although CirrOS is lightweight and useful for rapid testing, both the attacker and victim instances lacked the tools and libraries needed to support the Snort installation

This limitation in CirrOS became apparent when we tried to configure Snort for intrusion detection. The image's minimal environment did not support Snort or other required packages, causing multiple failures in the system. To overcome this, we switched to the Ubuntu 16.04 Xenial image, which provided the necessary compatibility and enabled Snort to install and configure correctly.

5.3. Multi-Node Deployment Challenges:

In our efforts to create a more scalable OpenStack environment, we also tested a three-node deployment on VirtualBox, with separate VMs for the controller, and two compute nodes. The installation process and errors faced are described further in the APPENDIX section of this report.

After exhausting troubleshooting options, we decided to simplify our setup by reverting to a single-node deployment using DevStack, which allowed us to focus on implementing and testing security features without the complexities of a multi-node setup.

5.4. DNS Resolution Problems: After we were able to successfully take the remote access of the attacker and victim server instances using SSH we were faces with yet another challenge.

In order to download the required tools, we needed to update the servers. The ‘*sudo apt update*’ command was failing but ping request to an IP was getting successful. So, the issue was with the DNS resolution as shown in below figure.

```
ubuntu@victim1:~$ sudo apt update
sudo: unable to resolve host victim1: Connection refused
Err:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Temporary failure resolving 'archive.ubuntu.com'
Err:2 http://security.ubuntu.com/ubuntu xenial-security InRelease
Temporary failure resolving 'security.ubuntu.com'
Err:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Temporary failure resolving 'archive.ubuntu.com'
Err:4 https://esm.ubuntu.com/infra/ubuntu xenial-infra-security InRelease
Could not resolve host: esm.ubuntu.com
Err:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Temporary failure resolving 'archive.ubuntu.com'
Err:6 https://esm.ubuntu.com/infra/ubuntu xenial-infra-updates InRelease
Could not resolve host: esm.ubuntu.com
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/xenial/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/xenial-updates/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/xenial-backports/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/xenial-security/InRelease Temporary failure resolving 'security.ubuntu.com'
W: Failed to fetch https://esm.ubuntu.com/infra/ubuntu/dists/xenial-infra-security/InRelease Could not resolve host: esm.ubuntu.com
W: Failed to fetch https://esm.ubuntu.com/infra/ubuntu/dists/xenial-infra-updates/InRelease Could not resolve host: esm.ubuntu.com
W: Some index files failed to download. They have been ignored, or old ones used instead.
```

We changed the nameserver specified in the /etc/resolv.conf file to 1.1.1.1 and 1.0.0.1 which are mapped to the public DNS servers of Cloudflare and it solved our issue as shown in the figure below:

```
ubuntu@victim1: ~
GNU nano 2.5.3 File: /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
# DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 1.1.1.1
nameserver 1.0.0.1
```

```
ubuntu@victim1:~$ sudo apt update
sudo: unable to resolve host victim1: Connection refused
ubuntu@victim1:~$ sudo apt update
sudo: unable to resolve host victim1
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [106 kB]
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [106 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [106 kB]
Get:5 https://esm.ubuntu.com/infra/ubuntu xenial-infra-security InRelease [7,521 B]
Get:6 https://esm.ubuntu.com/infra/ubuntu xenial-infra-updates InRelease [7,472 B]
Get:7 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [7,532 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial/universe Translation-en [4,354 kB]
Get:9 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [144 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial/multiverse Translation-en [106 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [1,269 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial-updates/main Translation-en [303 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [1,171 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/universe Translation-en [334 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 Packages [21.6 kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse Translation-en [8,440 B]
Get:17 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [913 kB]
Get:18 http://security.ubuntu.com/ubuntu xenial-security/main Translation-en [211 kB]
Get:19 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [740 kB]
Get:20 http://security.ubuntu.com/ubuntu xenial-security/universe Translation-en [203 kB]
Get:21 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [7,864 B]
Get:22 http://security.ubuntu.com/ubuntu xenial-security/multiverse Translation-en [2,672 B]
Get:23 http://archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [10.2 kB]
Get:24 http://archive.ubuntu.com/ubuntu xenial-backports/main Translation-en [4,456 B]
Get:25 http://archive.ubuntu.com/ubuntu xenial-backports/universe amd64 Packages [11.5 kB]
Get:26 http://archive.ubuntu.com/ubuntu xenial-backports/universe Translation-en [4,476 B]
Get:27 https://esm.ubuntu.com/infra/ubuntu xenial-infra-security/main amd64 Packages [970 kB]
Get:28 https://esm.ubuntu.com/infra/ubuntu xenial-infra-updates/main amd64 Packages [4,980 B]
Fetched 18.7 MB in 2d 1h 17min 30s (105 B/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
12 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@victim1:~$
```

6.0 References

- [1] <https://docs.openstack.org/devstack/latest/>
- [2] <https://www.openstack.org/software/>
- [3] <https://www.fortinet.com/resources/cyberglossary/openstack>
- [4] <https://www.eurovps.com/blog/openstack-cloud-benefits-challenges/>
- [5] <https://bugs.launchpad.net/keystone/+bug/2042744>
- [6] <https://upcloud.com/resources/tutorials/install-snort-ubuntu>

7.0 APPENDIX

7.1. Installation of Bobcat, OpenStack 2023.2

In this installation, we attempted a minimal deployment for Bobcat, OpenStack 2023.2. This release was selected because although it is not under development any more, it is still maintained. This means bugs have been fixed and most solutions are available online.

We attempted this deployment on Oracle VirtualBox with one controller (controller) and two compute nodes (compute1, compute2). The VM specifications were as follows:

- Storage: 25GB for each VM.
- RAM: 4GB each, with 2 VCPUs.
- OS: Ubuntu 22.04 LTS

We launched controller and compute1, configured the network, and confirmed that the two nodes communicated (ping) with each other and the Internet. Compute2 was spun up later by cloning compute1.

On the controller, we successfully configured and verified operation of chrony (clock synchronization), MariaDB, RabbitMQ, Memcached, and Etc. In addition, this deployment required identity service (Keystone), image service (Glance), placement service (Placement), compute service (Nova), networking service (Neutron), and the dashboard (Horizon).

1. Keystone identity service

This was done on controller. We installed keystone package and added configurations to keystone.conf file. However, when trying to populate the service database, an error was thrown:

```

root@Controller01:/home/inse6620# su -s /bin/sh -c "keystone-manage db_sync" keystone
Exception ignored in: <function _removeHandlerRef at 0x75418854c430>
Traceback (most recent call last):
  File "/usr/lib/python3.10/logging/__init__.py", line 846, in _removeHandlerRef
  File "/usr/lib/python3.10/logging/__init__.py", line 226, in _acquireLock
  File "/usr/lib/python3.10/threading.py", line 164, in acquire
  File "/usr/lib/python3/dist-packages/eventlet/green/thread.py", line 34, in get_ident
AttributeError: 'NoneType' object has no attribute 'getcurrent'
root@Controller01:/home/inse6620# nano /usr/lib/python3/dist-packages/eventlet/green/thread.py

```

We found a fix for this under the reported keystone bugs [5] that involved making a try-except construction for the `get_ident` function in `thread.py` as shown below:

```

root@controller: /home/inse6620
GNU nano 6.2 /usr/lib/python3/dist-packages/eventlet/green/thread.py
def get_ident(gr=None):
    try:
        if gr is None:
            return id(greenlet.getcurrent())
        else:
            return id(gr)
    except:
        return id(gr)

```

Following this, we successfully populated the database and configured Apache HTTP server.

2. Glance Image service

This was done on controller. We successfully installed glance package, edited `glance-api.conf` file and populated the database for this service. An image of Ubuntu 16.04 was successfully uploaded.

3. Placement service

This was done on controller node only. The `placement-api` package was installed, the `placement.conf` file edited, and we verified the successful operation of the service.

4. Nova compute service

This was done on both the controller and `compute1`. After installing the packages and confirming that all nova services were running on both controller and `compute1`, we verified the operation by successfully registering the compute node on the controller using the “`nova-manage discover_hosts`” command.

At this point, we cloned the `compute1` VM to create `compute2`. All hostnames and IP were edited as required in the three nodes. However, on starting the nova-compute service an error was thrown:

```

oot support detected
2024-08-07 22:43:13.615 1131 INFO nova.virt.node [None req-20118c51-262b-4ba3-aa98-5c18d8b8e677 - - - - -] Determined node
identity a1e76713-bd2a-44b6-b763-a34cab30a98a from /var/lib/nova/compute_id
2024-08-07 22:43:13.630 1131 ERROR oslo_service.service [None req-20118c51-262b-4ba3-aa98-5c18d8b8e677 - - - - -] Error sta
rting thread: nova.exception.InvalidConfiguration: My compute node a1e76713-bd2a-44b6-b763-a34cab30a98a has hypervisor_hostn
ame compute but virt driver reports it should be compute2. Possible rename detected, refusing to start!

```

We determined that this error was as a result of nova-compute service picking the ID of `compute1`. Since we had already started the service on `compute1`, the ID was stored in `/var/lib/nova/compute_id` and was copied during the cloning. Examination of the error revealed that the VM’s hostname was tied to the `compute_id`. As such, hypervisor in `compute2` was picking the hostname of `compute1` instead of `compute2`. We fixed this error by removing the `compute_id` in `compute2` using “`rm /var/lib/nova/compute_id`” and restarting the nova-compute service, which then ran successfully.

5. Neutron networking service

We selected Networking Option 2: Self-service networks. This option allows a regular, non-privileged account to create and manage virtual networks without involving an administrator. Neutron was configured on both the controller and compute nodes. We also created a bridge for neutron to manage network traffic through open vswitch bridges. All neutron services were restarted successfully.

6. Horizon dashboard

This was done on controller node. We successfully installed openstack-dashboard package, edited the local_settings.py file as required and reloaded the apache2.service web server configurations. To verify, we visited <http://10.0.0.11/horizon> and successfully logged in as admin and demo users.

7. Launching an instance

We successfully created a provider network and subnet. We encountered an error when trying to use the demo account to create a private network. The demo user (non-privileged account) had no permissions to create a network. We therefore created a policy.yaml file in neutron on the controller to allow non-privileged accounts to create their own networks, subnets and routers.

```
GNU nano 6.2 /etc/neutron/policy.yaml
"create_network": "rule:admin_or_owner or rule:member"
"create_subnet": "rule:admin_or_owner or rule:member"
"create_router": "rule:admin_or_owner or rule:member"
```

We were then able to successfully create a subnet with CIDR 192.168.100.0/24 and a router that would provide NAT services.

```
root@controller:/home/inse6620# openstack port list --router router
+-----+-----+-----+-----+-----+
| ID | Name | MAC Address | Fixed IP Addresses | Status |
+-----+-----+-----+-----+-----+
| 785e717f-2516-44ea-aa51-1567e696e8a3 | | fa:16:3e:f7:b7:62 | ip_address='192.168.100.1', subnet_id='25ff2b04-9e09-46d4-a7af-cd1d189c60dc' | DOWN |
| 9b36d323-6729-4c74-83e6-439d44da56c5 | | fa:16:3e:1c:c3:d8 | ip_address='203.0.113.162', subnet_id='42ead0f9-a5d9-4a37-87fa-fa060a871b1f' | ACTIVE |
+-----+-----+-----+-----+-----+
root@controller:/home/inse6620# ip netns
qrouter-0b7968e4-1a51-4180-963e-f2759b3ae147 (id: 2)
qdhcp-6bafc928-002e-4a5e-8d82-0e6449e54d5b (id: 1)
qdhcp-9d7a9c59-a728-4679-bbfe-f711984c21d2 (id: 0)
root@controller:/home/inse6620#
```

As seen in the above, we noticed that the private network was DOWN. This impacted our ability to deploy instances through our horizon dashboard and through CLI, even though we were able to create m1.nano and m1.tiny flavors through Horizon dashboard. Error logs showed the following error:

```
2024-08-12 12:05:04.377 2985 ERROR neutron.plugins.ml2.managers [req-1b8b0f65-d60e-477c-a955-8e0282183a5b req-b6562c72-944d-46a6-99b8-c31753ef
fe5f cf21d800cee2447eaa0a3b66bebb2f 8b0709c619f24cd998f3af38ee2b5609 - - default default] Failed to bind port cc93180f-50e7-417d-ab76-b6fc01
353505 on host compute1 for vnic_type normal using segments [{'id': '24c65104-7879-4faa-a013-1546f2c786c8', 'network_type': 'vxlan', 'physical
network': None, 'segmentation id': 696, 'network id': '3753050e-72da-408b-9909-d23a088b31d9'}]
```

We tried to change the vnic_type to different options including 'direct' and 'direct physical' but the error persisted. We verified our configurations in neutron.conf, nova.conf, and openvswitch_agent.ini. We also checked that the bridge and the networks were configured correctly and working.

At this point, we could not find any solutions to making the network status ACTIVE. We therefore decided to stick with the single-node architecture for our OpenStack deployment.