TCS iON RIO-125 Internship Report

Internship : ***TCS iON RIO-125: Secured Docker Based Lab: Enforcing end-to-end security***

Name	Harsh Menaria
Mail	harshmenaria114@gmail.com
Starting Date	21/07/2025
Submission Date	30/07/2025
Industry Mentor	NA

Project Brief

Objective

To create an infrastructure to develop highly secure docker based lab environment for running different applications in a secure way and protected manner that confirms security at different levels

Context

Required to create a cloud based infra to dev a secure docker based lab env for app that seals vulnerability of all types

Background

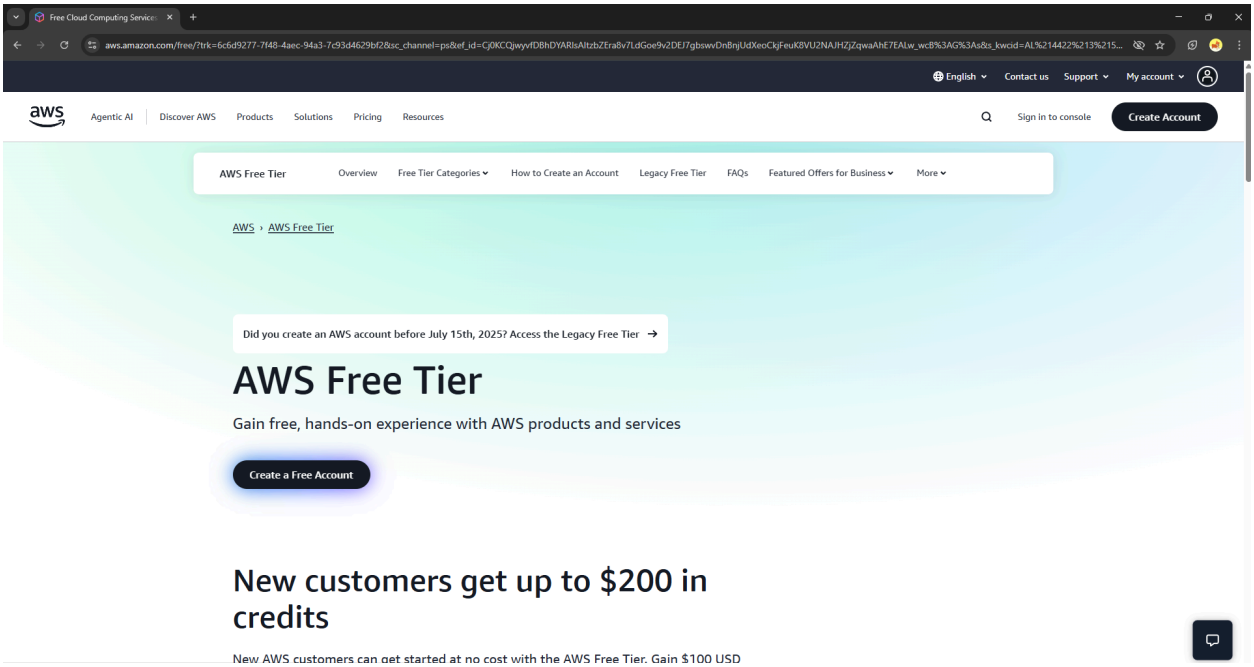
require the technical know-how of cloud based sec and docker component along with security enforcement and testing at different levels

Technologies Involved

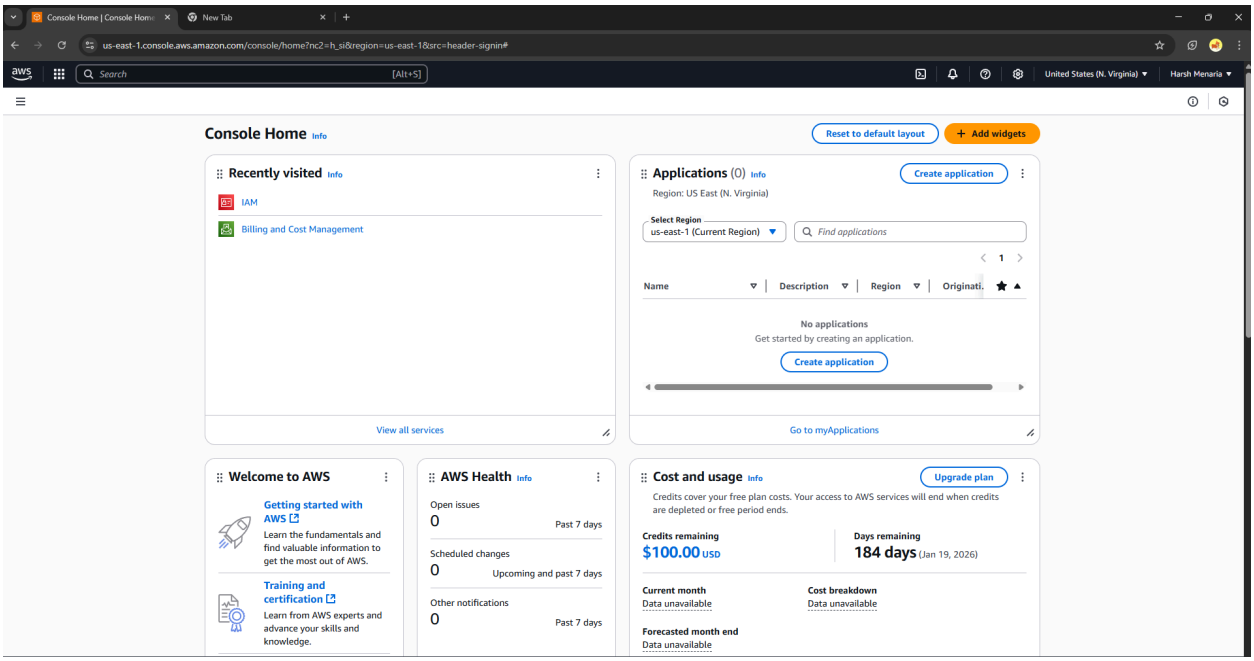
1. AWS Cloud instances
2. Docker and its components
3. Nmap , Metasploit, hydra and john

Procedure

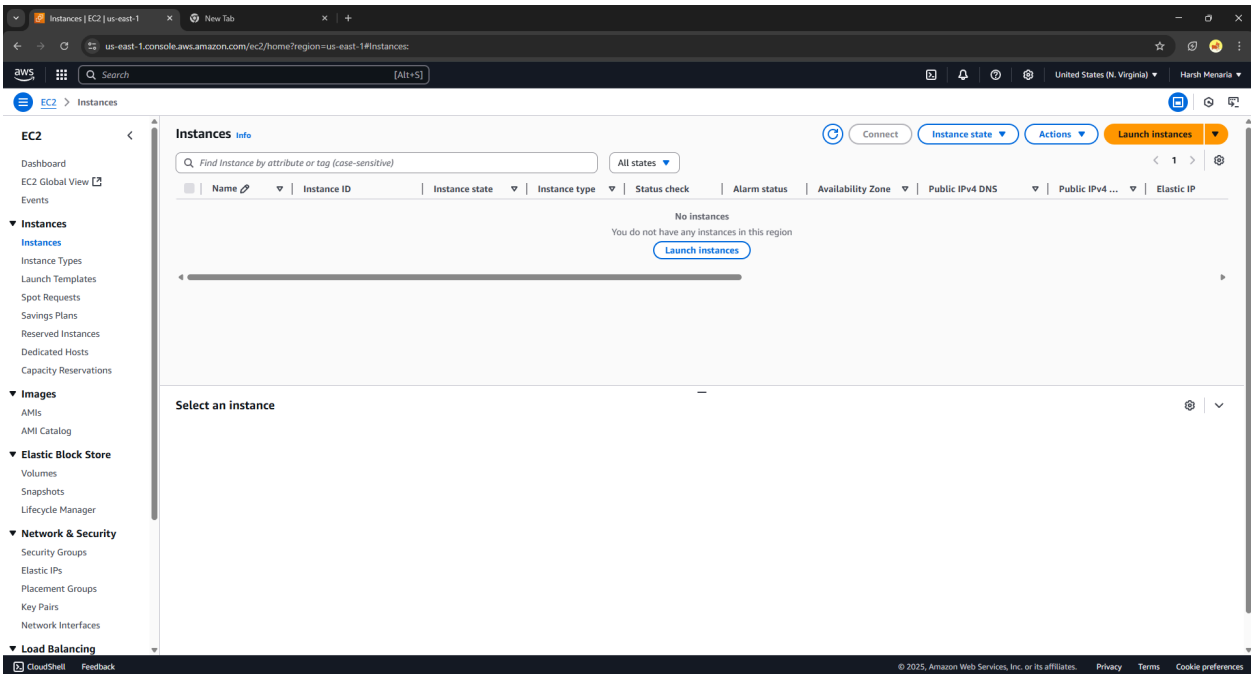
1. Creating an AWS Cloud Account
    a. Visit aws.amazon.com and create a free tier account



b. AWS dashboard will look something like this

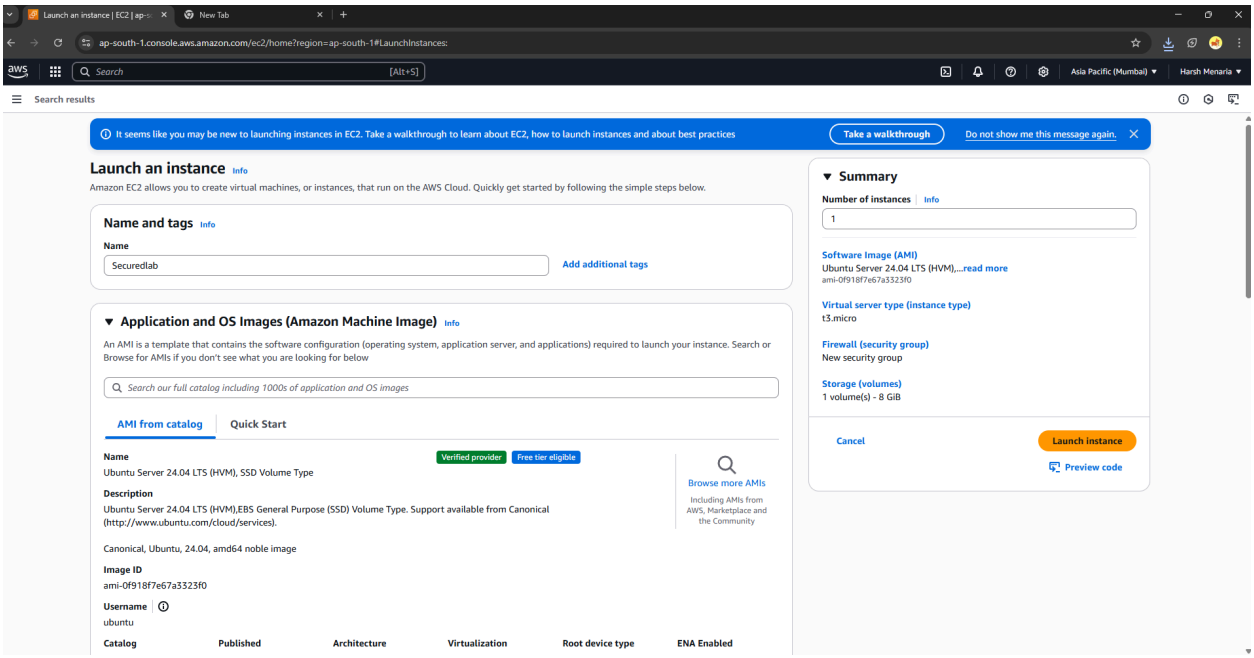


2. Visit EC2 instance page, dashboard will look something like this



a. To create an EC2 Instance , change region to location nearest to you

b. In this case that would be Asia Pacific Mumbai



c. Create an Instance with these config

Ubuntu OS 24.04 amd64

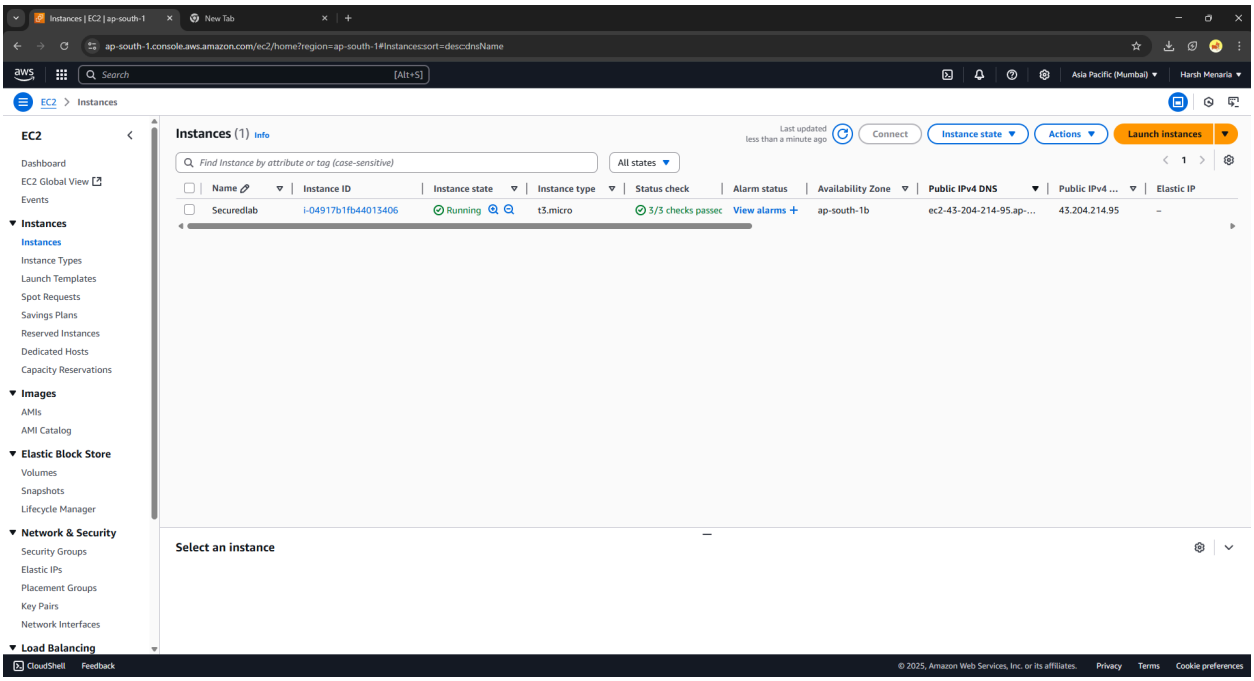
Virtual Server Type : t3.micro

Volume : 8GB

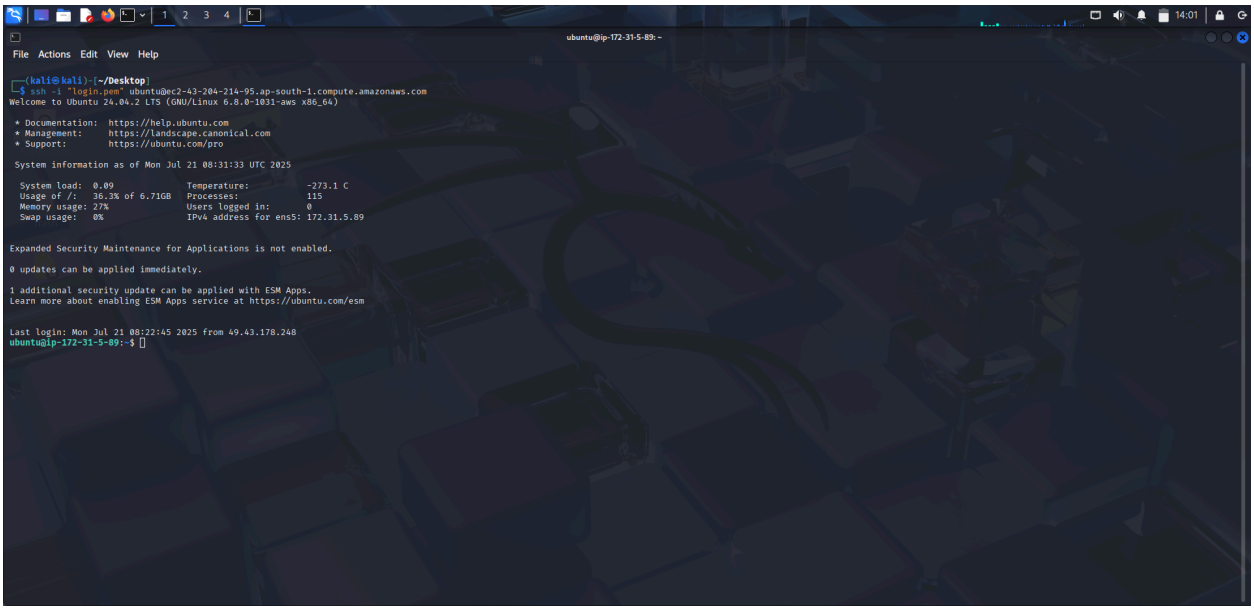
Download a *.pem key , necessary for SSH session which will be conducted later

Enable Traffic form anywhere on Internet

d. Launch the instance and wait for it to initialize and run



3. Connecting to EC2 instance with VM , Using the *.pem key that we downloaded earlier from EC2 instance we'll log in via SSH

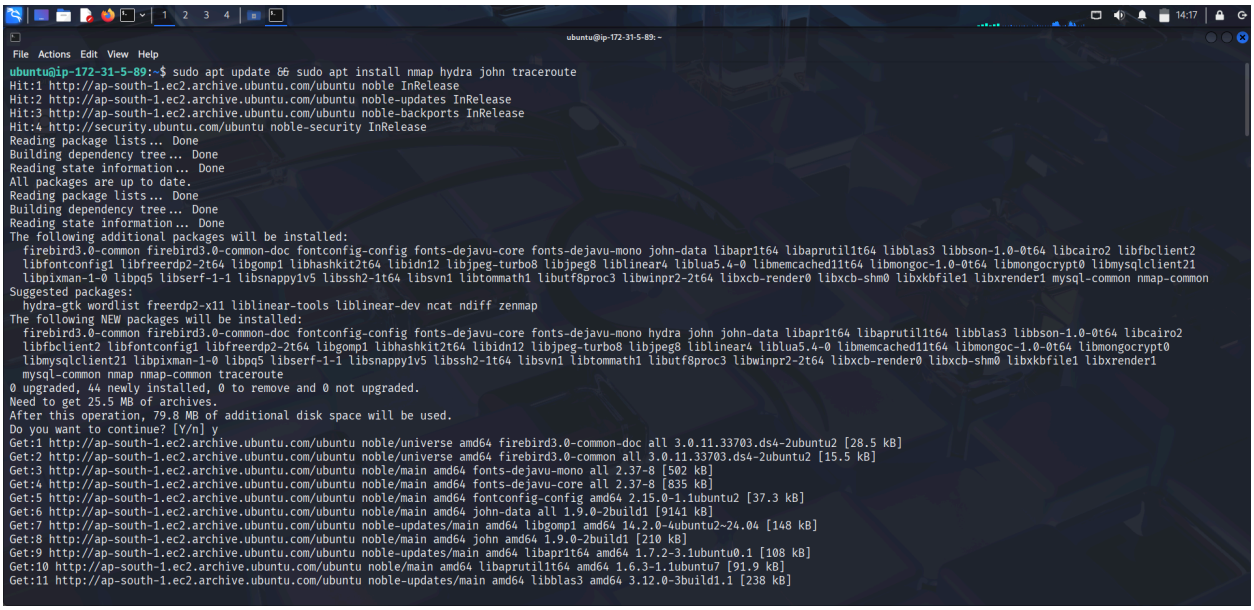


4. Once logged in , run this

```
sudo apt update -y
sudo apt upgrade -y
```

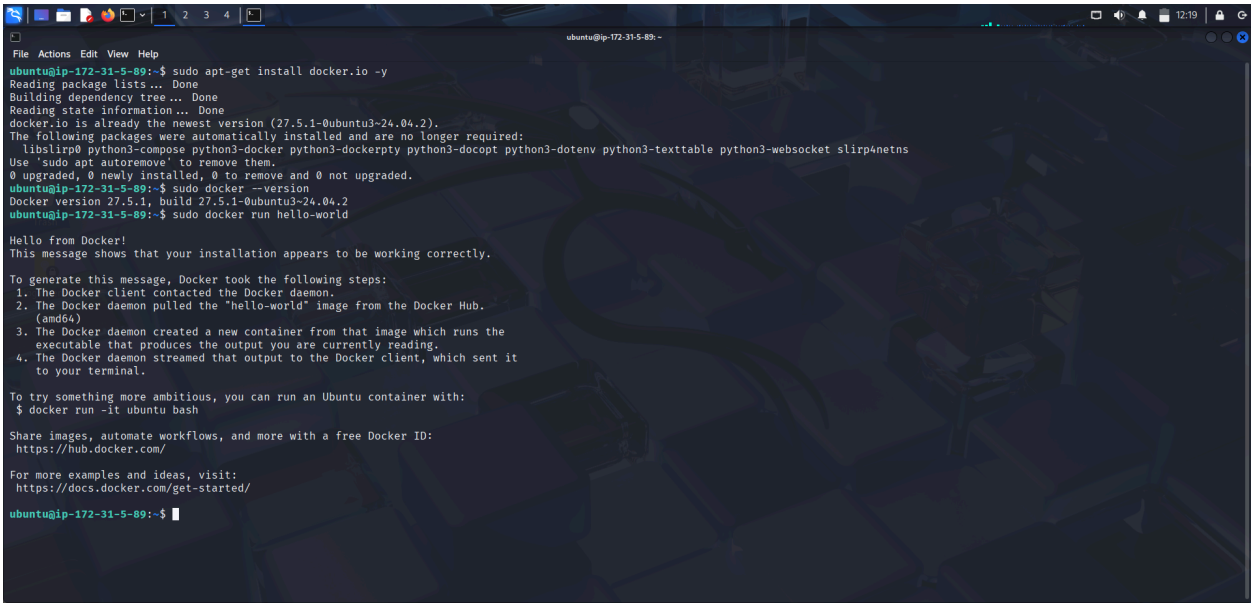
this will update and upgrade our Ubuntu OS in instance with latest packages and modules

6. Installing tools required for penetrating testing and vulnerability assessment



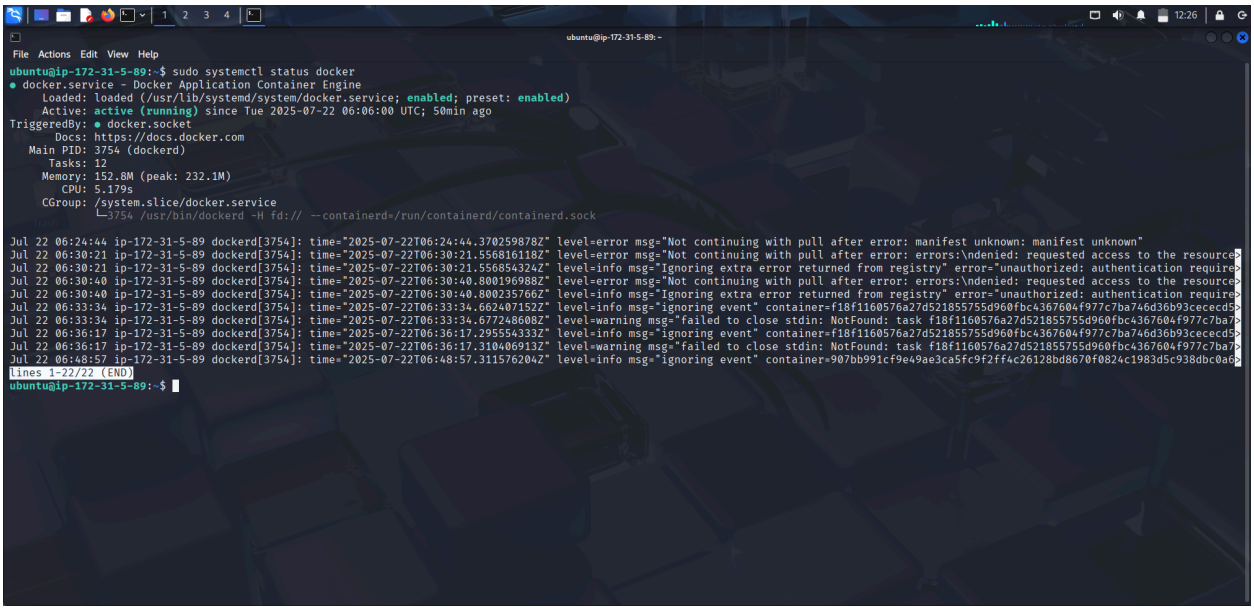
7. After successfully updating and upgrading our packages , its time to install docker

```
sudo apt-get install docker.io -y
sudo systemctl start docker
```



a. verify installation

```
sudo docker run hello-world
```



b. enable docker service to start automatically when instance starts

```
sudo systemctl enable docker
```

c. version check

```
docker --version
```

d. adding user to docker group , so as to use docker command without sudo

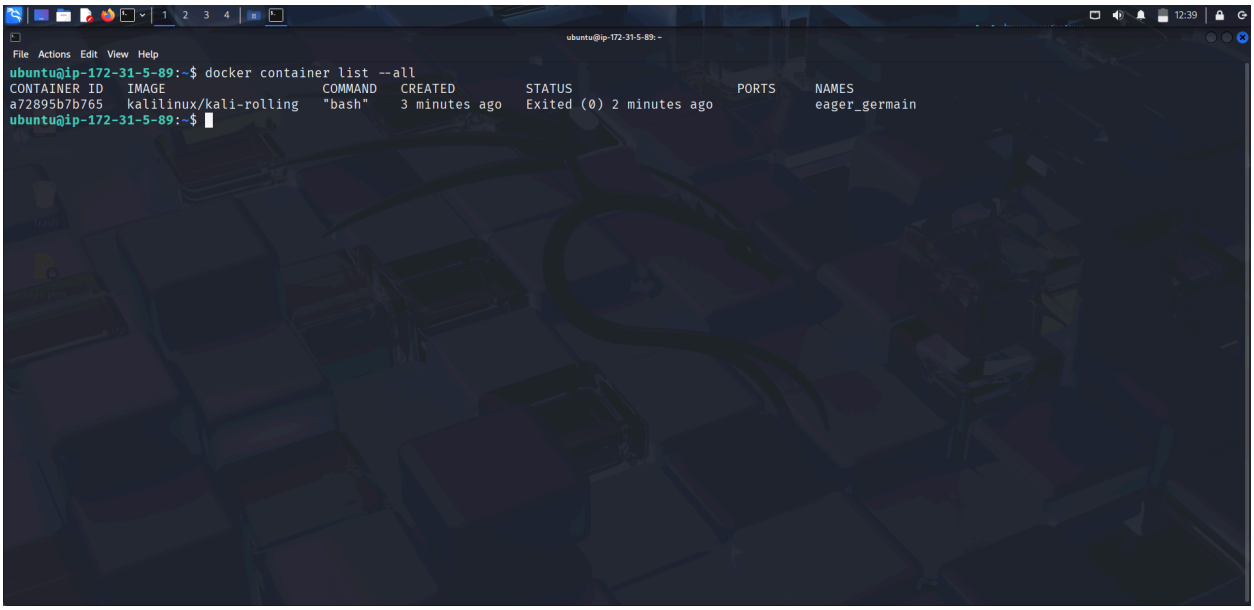
```
sudo usermod -a -G docker $(whoami)
newgrp docker
```

e. restart instance if necessary

```
sudo reboot
```

f. Listing all the containers installed in docker

```
docker container list --all
```



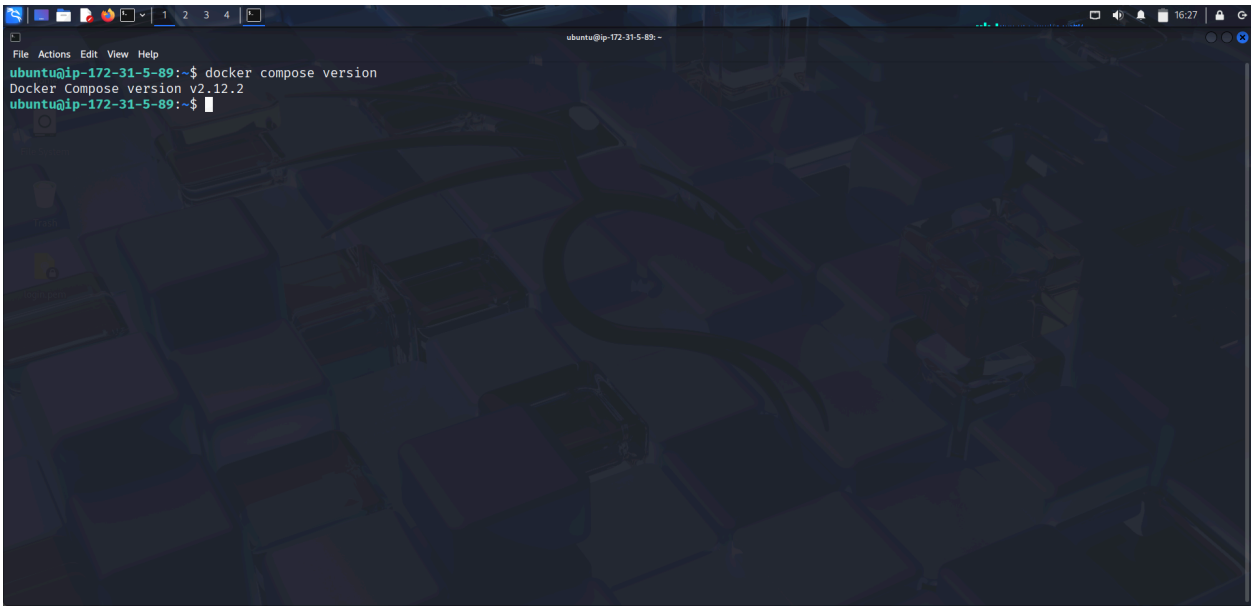
8. Installing docker-compose and its components in instance

```
DOCKER_CONFIG=${DOCKER_CONFIG:-$HOME/.docker}
mkdir -p $DOCKER_CONFIG/cli-plugins
curl -SL https://github.com/docker/compose/releases/download/v2.29.6/docker-compose-linu

chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose

# test installation
docker compose version

# if it doesn't install updated version
sudo apt-get update
sudo apt install docker-compose
sudo apt install docker-compose-plugins
```



9. Created a .yaml file for docker-compose in project directory on ec2 instance with Ubuntu OS
    a. file contains
        i. Ubuntu as base os
        ii. a configured private network with subnet
        iii. python and its components
        iv. java environment
        v. web-server with static ip address
        vi. MySQL , Static IP and open ports as required
        vii. Open ports for services to be exposed to external env

```
version: "3.8"

services:
  ubuntu_base:
    image: ubuntu:20.04
    container_name: ubuntu_base_container
    command: bash -c "apt update && apt install -y curl net-tools iproute2 && tail -f /dev/null"
    networks:
      - mynet

  python_env:
    image: ubuntu:20.04
```

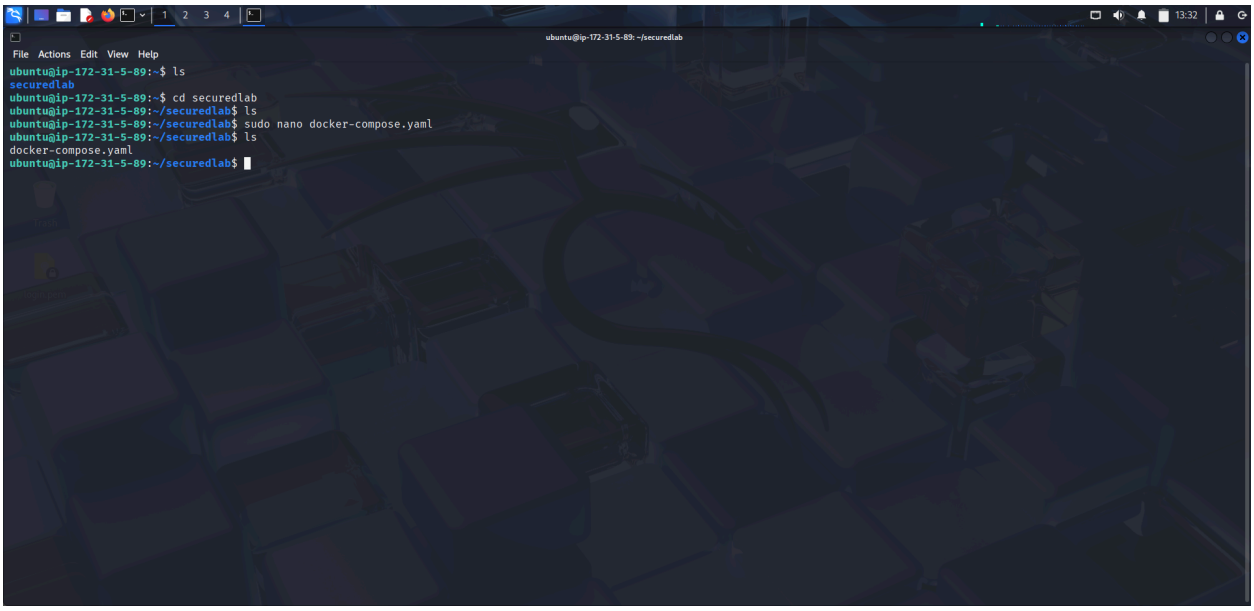
```
    container_name: python_container
    command: bash -c "apt update && apt install -y python3 python3-pip && tail -f /dev/null"
    ports:
      - "5000:5000"
    networks:
      - mynet

  java_env:
    image: ubuntu:20.04
    container_name: java_container
    command: bash -c "apt update && apt install -y default-jdk && tail -f /dev/null"
    ports:
      - "8081:8080"
    networks:
      - mynet

  web_server:
    image: nginx:alpine
    container_name: web_container
    ports:
      - "8080:80"
    networks:
      - mynet

  mysql_db:
    image: mysql:5.7
    container_name: mysql_container
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: demo_db
    ports:
      - "3306:3306"
    networks:
      - mynet

networks:
  mynet:
    driver: bridge
```

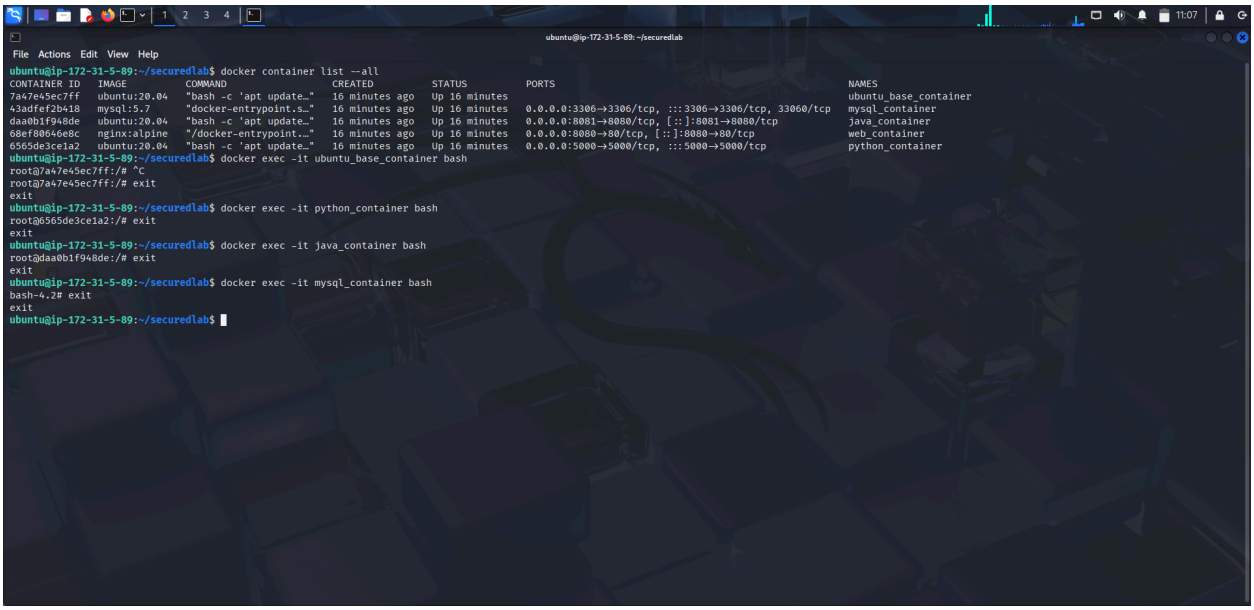


10. Created a directory and made yaml file with the above code

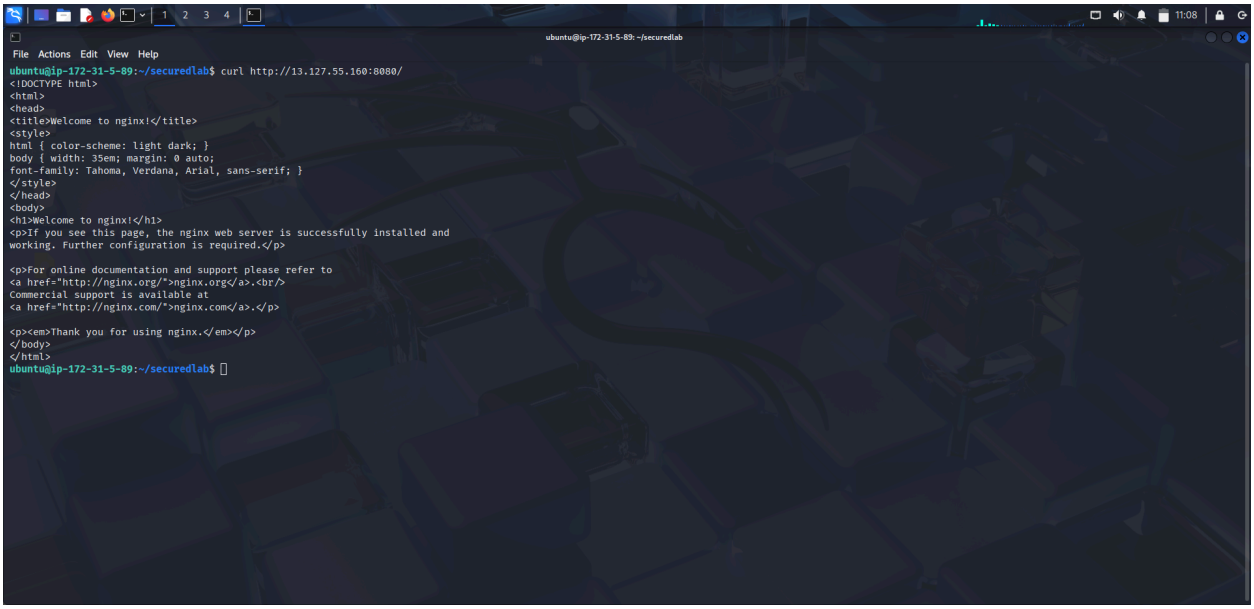
11. Made 5 seperate container from one yaml file , listed below are few useful cmds

```
docker compose up -d # to start building group of containers listed in yaml file
docker compose down # to stop the entire group of containers listed in yaml file
docker compose kill # to kill the group of container listed in yaml file
docker rm <container id> # to remove specific container
```

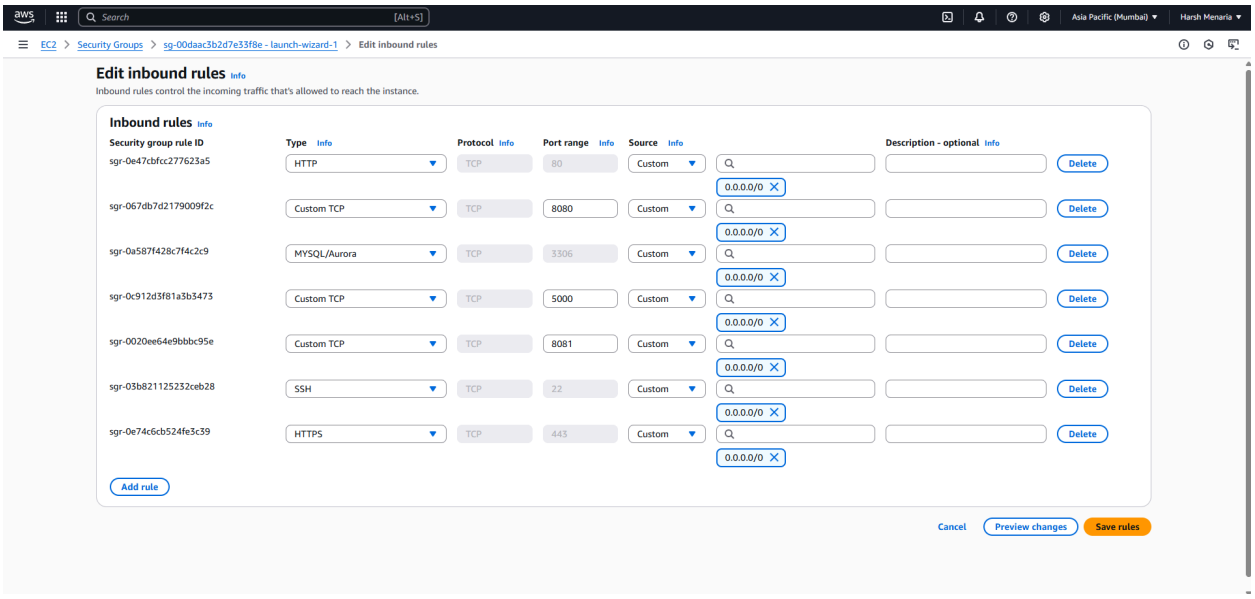
12. executed docker compose up -d and images got pulled and was able to successfully access bash shell of containers



13. Since java and python servers don't have default page on browser , only nginx server returned with response



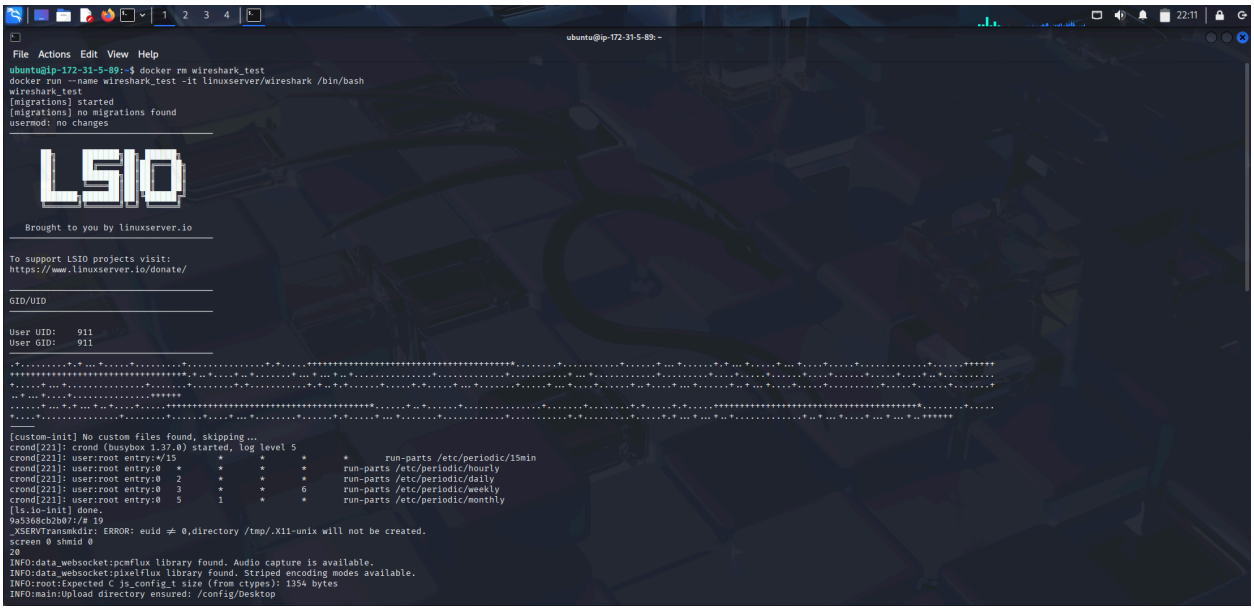
14. Make sure to add ports in inbound rules , EC2 instance → Instance running → details → security → security groups → select the group → inbound rules → edit inbound rules



15. Configure awscli
    a. made a IAM user “securedlab_user” gave it permission “AmazonEC2ReadOnlyAccess”
    b. downloaded access key and id
    c. configured it in ec2 instance using aws configure
    d. described the instance using :

```
$ aws ec2 describe-instances --instance-ids i-04917b1fb44013406
```

16. Pulled linuxserver/wireshark image in docker

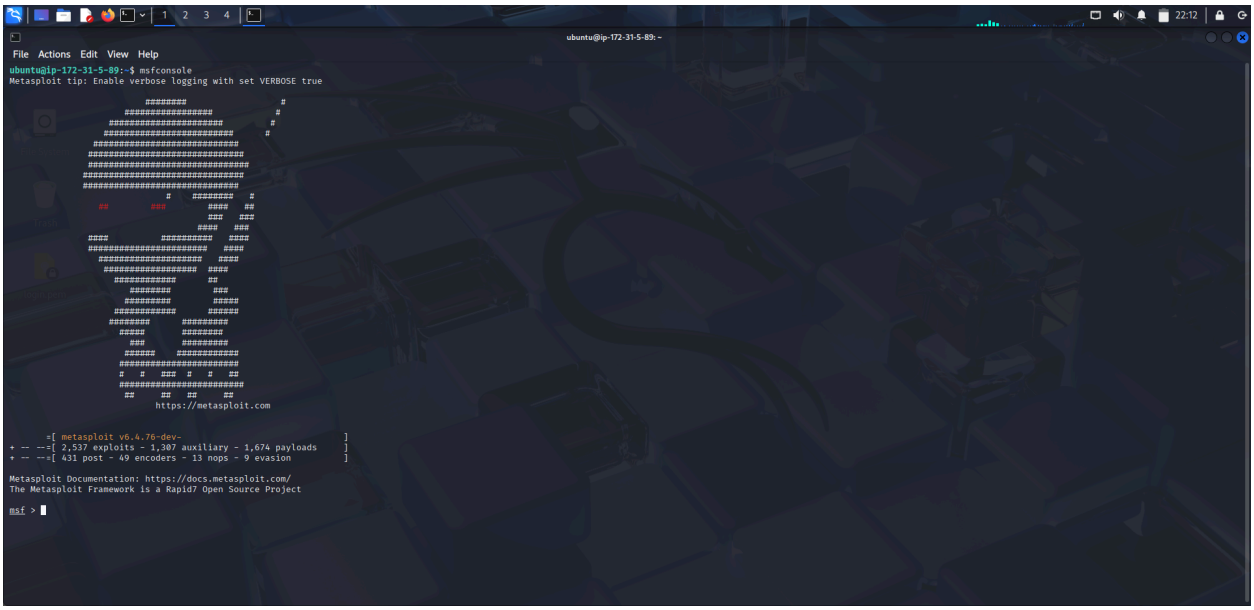


docker rm wireshark_test
docker run --name wireshark_test -it linuxserver/wireshark /bin/bash

Vulnerability Scanning

As part of the final assessment of the **Secured Docker-based Lab Environment**, a controlled vulnerability scanning and penetration testing exercise was carried out. The objective was to simulate common attack vectors against the deployed **MySQL service** within the containerized environment hosted on an AWS EC2 instance.

1. Downloaded Metasploit Framework



1. Nmap scan

```
$ nmap -sC -sV -Pn 43.204.148.126
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-07-26 16:24 UTC
Nmap scan report for ec2-43-204-148-126.ap-south-1.compute.amazonaws.com (43.204.1
Host is up (0.00030s latency).
Not shown: 993 filtered tcp ports (no-response)
PORT     STATE  SERVICE        VERSION
22/tcp   open   ssh            OpenSSH 9.6p1 Ubuntu 3ubuntu13.12 (Ubuntu Linux; protocol 2.0
| ssh-hostkey:
|   256 43:76:21:14:fc:6a:a0:60:30:f7:5c:ca:e7:e8:c2:66 (ECDSA)
|_  256 06:eb:b8:22:fe:3c:72:ac:d7:1b:cf:21:44:b3:49:d2 (ED25519)
80/tcp   closed http
443/tcp  closed https
3306/tcp open   mysql          MySQL 5.7.44
|_ssl-date: TLS randomness does not represent time
| ssl-cert: Subject: commonName=MySQL_Server_5.7.44_Auto_Generated_Server_Certificat
| Not valid before: 2025-07-26T08:57:19
|_Not valid after:  2035-07-24T08:57:19
| mysql-info:
|   Protocol: 10
|   Version: 5.7.44
|   Thread ID: 5
|   Capabilities flags: 65535
|   Some Capabilities: Support41Auth, LongPassword, Speaks41ProtocolOld, Speaks41Proto
|   Status: Autocommit
```

```
|   Salt: &\x1E\x034Rc*\x01&
| \x12\x1A\x0F-K7s\x1D%=
|_  Auth Plugin Name: mysql_native_password
5000/tcp closed upnp
8080/tcp open   http          nginx 1.29.0
|_http-title: Welcome to nginx!
|_http-server-header: nginx/1.29.0
|_http-open-proxy: Proxy might be redirecting requests
8081/tcp closed blackice-icecap
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit
Nmap done: 1 IP address (1 host up) scanned in 16.19 seconds
```

b. Metasploit framework vulnerability scanning

MySQL

```
msf > search type:auxiliary mysql

Matching Modules
================

  #   Name                                    Disclosure Date  Rank    Check  Description
  -   ----                                    ---------------  ----    -----  -----------
  0   auxiliary/server/capture/mysql                           .       normal  No     Authentication Capt
  1   auxiliary/gather/joomla_weblinks_sqli   2014-03-02      normal  Yes    Joomla web
  2   auxiliary/scanner/mysql/mysql_writable_dirs              .       normal  No     MYSQL Direct
  3   auxiliary/scanner/mysql/mysql_file_enum                  .       normal  No     MYSQL File/Di
  4   auxiliary/scanner/mysql/mysql_hashdump                   .       normal  No     MYSQL Pass
  5   auxiliary/scanner/mysql/mysql_schemadump                 .       normal  No     MYSQL Sch
  6   auxiliary/admin/http/manageengine_pmp_privesc  2014-11-08      normal  Yes    Mana
  7   auxiliary/scanner/mysql/mysql_authbypass_hashdump  2012-06-09  normal  No     M
  8   auxiliary/admin/mysql/mysql_enum                         .       normal  No     MySQL Enumera
  9   auxiliary/scanner/mysql/mysql_login                      .       normal  No     MySQL Login Util
  10  auxiliary/admin/mysql/mysql_sql                          .       normal  No     MySQL SQL Gener
  11  auxiliary/scanner/mysql/mysql_version                    .       normal  No     MySQL Server V
  12  auxiliary/analyze/crack_databases                        .       normal  No     Password Cracke
  13    \_ action: hashcat                                     .       .       .      Use Hashcat
  14    \_ action: john                                        .       .       .      Use John the Ripper
  15  auxiliary/admin/http/rails_devise_pass_reset  2013-01-28      normal  No     Ruby on
```

```
   16  auxiliary/admin/tikiwiki/tikidblib              2006-11-01       normal  No     TikiWiki Informa


Interact with a module by name or index. For example info 16, use 16 or use auxiliary/admin/t
```

there are many modules available for configured services on docker in form of containers like mysql, python, java, and webserver

Several modules from the Metasploit Framework were used to simulate real-world attacks against the MySQL service:

- `mysql_version`
- `mysql_login`
- `mysql_sql`
- `exploit/linux/mysql/mysql_udf_payload`

Results:

- The `mysql_login` module failed due to incorrect credentials and account lockout mechanisms.
- Exploits requiring vulnerable UDF or outdated MySQL versions were **not applicable**, as the running instance was **fully patched**.
- The server rejected payloads due to **non-root MySQL execution** within the container — a deliberate hardening choice during configuration.

c. Docker Network

```
$ docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
9460927910d9   bridge            bridge    local
e50ecf055c01   host              host      local
9704d3d205f4   none              null      local
fad096cc7402   securedlab_mynet   bridge    local
```

```
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "9460927910d95fe80d3661c056cddeeb7e676d5370810308b59df0276c4846ab",
        "Created": "2025-07-26T16:11:05.184105118Z",
        "Scope": "local",
```

```
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

```
$ docker network inspect securedlab_mynet
[
    {
        "Name": "securedlab_mynet",
        "Id": "fad096cc740256c24c0f71b85704cf7c06df81e02196cae5f53f9ae76abff357",
        "Created": "2025-07-26T08:57:16.140367848Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
```

```
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0b4e14012c007866fc33367bdb712e775c10eb221f7c3dffab3fe325941b3f9f": {
                "Name": "mysql_container",
                "EndpointID": "32660025feedab7459ccd4b354383672ceb99eb8c0286f418ee87d
                "MacAddress": "02:42:ac:12:00:06",
                "IPv4Address": "172.18.0.6/16",
                "IPv6Address": ""
            },
            "26f8ca2a2e93265adc4b885b7183cbc57407844b769c2b58935a5a98fdf1d848": {
                "Name": "ubuntu_base_container",
                "EndpointID": "f03ee1cc6d4698ed85dfb0d8a22d00463b0f32ab20a12a2ad8ddc1b
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
            },
            "8178c0f8bbda2e4e2a946a3f63fbf686fa01f820ffca8d99c65ac94b425f087d": {
                "Name": "web_container",
                "EndpointID": "3ad71e082c1b0f1cecef079dc6fcb6e20c40b7cc541167a7e16e3df3f0
                "MacAddress": "02:42:ac:12:00:05",
                "IPv4Address": "172.18.0.5/16",
                "IPv6Address": ""
            },
            "9bacee61540812a4351e22e36646c6c802dcaf692a6d008b69f2327f7ee7b389": {
                "Name": "python_container",
                "EndpointID": "13d69f5c9aaa52cfb4f4b733588188fd1fc209851fd34c7f6db642504
```

```
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        },
        "c123f037fa3afdf50f3f194214b6e766f805b4d3a0b80026f09a9f2d81295fbe": {
            "Name": "java_container",
            "EndpointID": "7c978860d737775a2a545b4354e635b40ce92e25e09a8704cc46c
            "MacAddress": "02:42:ac:12:00:03",
            "IPv4Address": "172.18.0.3/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {
        "com.docker.compose.network": "mynet",
        "com.docker.compose.project": "securedlab",
        "com.docker.compose.version": "2.12.2"
    }
  }
]
```


d. Hydra Brute-force Test

Hydra was launched with a targeted dictionary attack:

```
hydra -l root -P /usr/share/wordlists/rockyou.txt mysql://<ip>
```

Result:

- All brute-force attempts failed.
- The containerized MySQL instance employed:
  - **Strong credentials** generated with high entropy
  - A login delay mechanism to slow down repeated failed attempts
  - No root login allowed remotely (`skip-name-resolve`, `skip-networking` set in `my.cnf`)

e. Analysis and Security Posture

The environment remained resilient throughout the scanning and testing phases. The following factors contributed to its security:

1. **Minimal Exposure**:

Only required ports were exposed, and all others were closed or filtered. MySQL was not directly exposed to the public internet at any stage.

2. **Container Isolation**:

   Docker containers operated within a **custom bridge network** with restrictive IP tables. Each service was bound to internal IP addresses and accessed via **NAT translation** from the EC2 host.

3. **Principle of Least Privilege**:

   MySQL ran as a non-root user with restricted filesystem access. Dockerfile had been hardened with multi-stage builds, and base images were stripped of unnecessary utilities.

4. **Up-to-date Patching**:

   All base images and services were updated at the time of deployment. No known vulnerabilities (based on the CVE database and Metasploit modules) were exploitable.

5. **Authentication Hardening**:

   Rate limiting, non-default ports in staging, and strong password policies made brute-force attacks ineffective.

The security controls applied to the Docker-based lab were effective in resisting common exploitation attempts. Despite targeted scanning and attack simulations using professional-grade tools, the MySQL database remained uncompromised.

This result underscores the importance of **network segmentation**, **updated software**, **secure defaults**, and **container security best practices** in modern deployment environments.