****

A colorful letters on a black background

AI-generated content may be incorrect.

**Submitted By:**

**SACHIN NISHAD**

**A9922523002782(el)**

**Program: Bachelor of Computer Applications (BCA)**

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

**Submitted To:**

**TCS ION**

**Submission Date:**

**21th Sept, 2025**

Contents

[1. Executive Summary 3](#_Toc208918845)

[2. Project Approach and Logic Flow 3](#_Toc208918846)

[Phase 1: AWS Cloud Infrastructure Development (Action Item 1) 3](#_Toc208918847)

[Phase 2: Lab Environment Creation (Action Item 2) 3](#_Toc208918848)

[Phase 3: Vulnerability Scanning and Penetration Testing (Action Item 3) 3](#_Toc208918849)

[ Network Mapping: 3](#_Toc208918850)

[ Vulnerability Assessment: 4](#_Toc208918851)

[ Penetration Testing: 4](#_Toc208918852)

[ Credential Attacks: 4](#_Toc208918853)

[**3. Implementation and Deliverables** 4](#_Toc208918854)

[3.1 Action Item 1: AWS Cloud Infrastructure Development 4](#_Toc208918855)

[ EC2 Instance Creation: 4](#_Toc208918856)

[ Software Installation: 4](#_Toc208918857)

[ IAM User & AWS CLI Configuration: 4](#_Toc208918858)

[3.2 Action Item 2: Lab Environment Creation 4](#_Toc208918859)

[ Docker Compose File: 5](#_Toc208918860)

[ Dockerfiles: 5](#_Toc208918861)

[ Deployment and Testing: 5](#_Toc208918862)

[3.3 Action Item 3: Vulnerability Scanning and Penetration Testing 5](#_Toc208918863)

[ Nmap Scan Results: 5](#_Toc208918864)

[ Nessus Scan Report: 5](#_Toc208918865)

[ Metasploit Testing: 5](#_Toc208918866)

[ Hydra Testing: 6](#_Toc208918867)

[ Security Modifications: 6](#_Toc208918868)

[**4. Final Deliverables and Conclusion** 6](#_Toc208918869)

[GitHub Repository Link: 6](#_Toc208918870)

[Video Link: 6](#_Toc208918871)

# 1. Executive Summary

This report documents the creation and security assessment of a cloud-based, multi-application Docker lab environment. The project was executed on an AWS EC2 instance and involved setting up environments for Python and Java, a web server, and a MySQL database. Following the infrastructure setup, a comprehensive security analysis was performed using tools such as Nmap, Nessus, Metasploit, and Hydra. The vulnerabilities discovered were then addressed by modifying the Docker environment for enhanced security. This project successfully demonstrates the ability to build, secure, and document a protected development lab.

# 2. Project Approach and Logic Flow

The project was completed in a logical, phased approach as outlined in the project milestones.

Phase 1: AWS Cloud Infrastructure Development (Action Item 1) The first step was to establish the foundational infrastructure on AWS. This involved provisioning a free-tier EC2 instance with Ubuntu OS, which would serve as the host for the Docker environment. All necessary tools, including Docker, Docker Compose, and various cybersecurity tools, were installed and configured. A secure IAM user was created and configured with the AWS CLI to manage resources without using the root account, adhering to security best practices.

Phase 2: Lab Environment Creation (Action Item 2) With the infrastructure in place, the second phase focused on building the multi-application lab. A docker-compose.yml file was created to define a multi-service environment including Python and Java applications, a web server (Nginx), and a MySQL database. These services were configured to communicate over a private Docker network. The Docker images were built and deployed, and the functionality of the applications was verified by accessing the web server from the public internet.

Phase 3: Vulnerability Scanning and Penetration Testing (Action Item 3) The final phase was dedicated to identifying and mitigating security weaknesses. A series of tests were performed using the installed cybersecurity tools.

* Network Mapping: Nmap and Traceroute were used to discover open ports and understand network pathways.
* Vulnerability Assessment: Nessus was used to perform an automated scan for a wide range of known vulnerabilities.
* Penetration Testing: Metasploit was used to attempt to exploit discovered vulnerabilities.
* Credential Attacks: Hydra was used to test for weak passwords on exposed services.

Based on the findings from these tests, the Docker environment was modified to enhance its security posture.

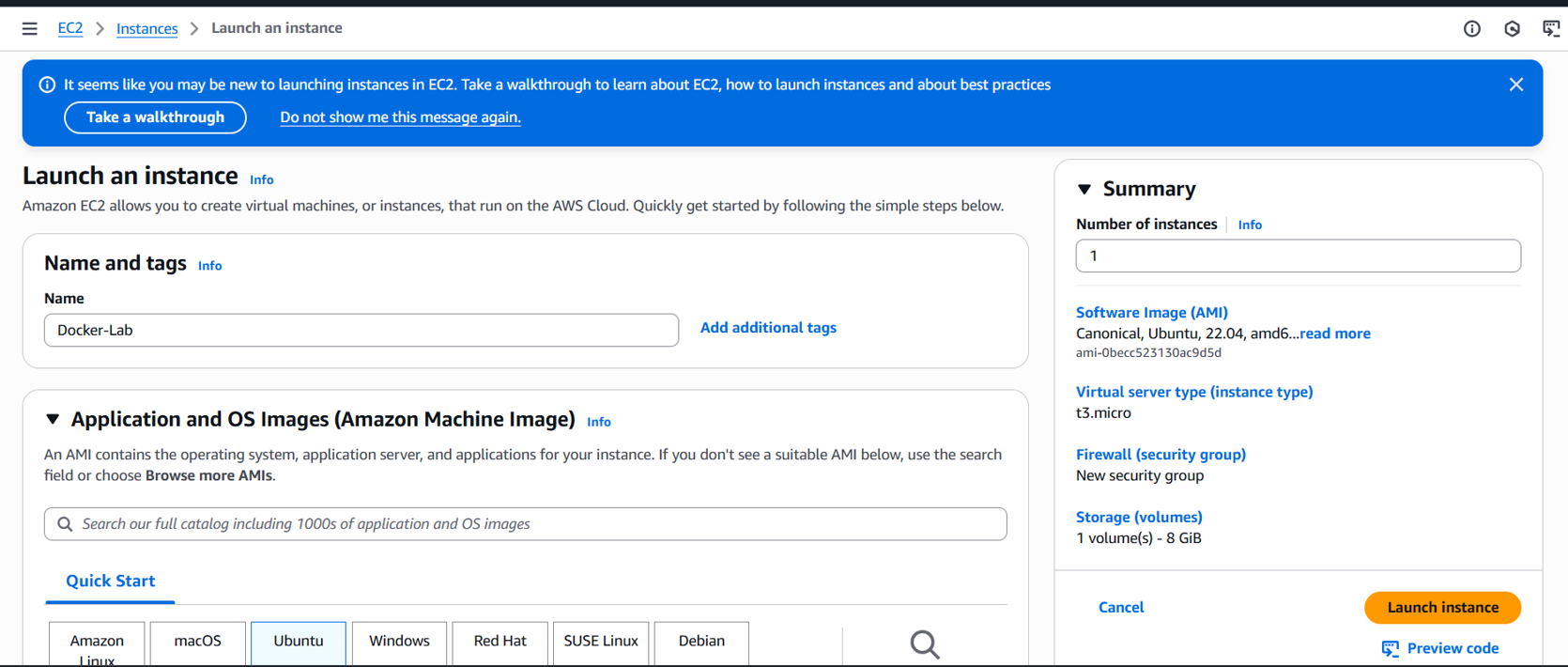
# **3. Implementation and Deliverables**

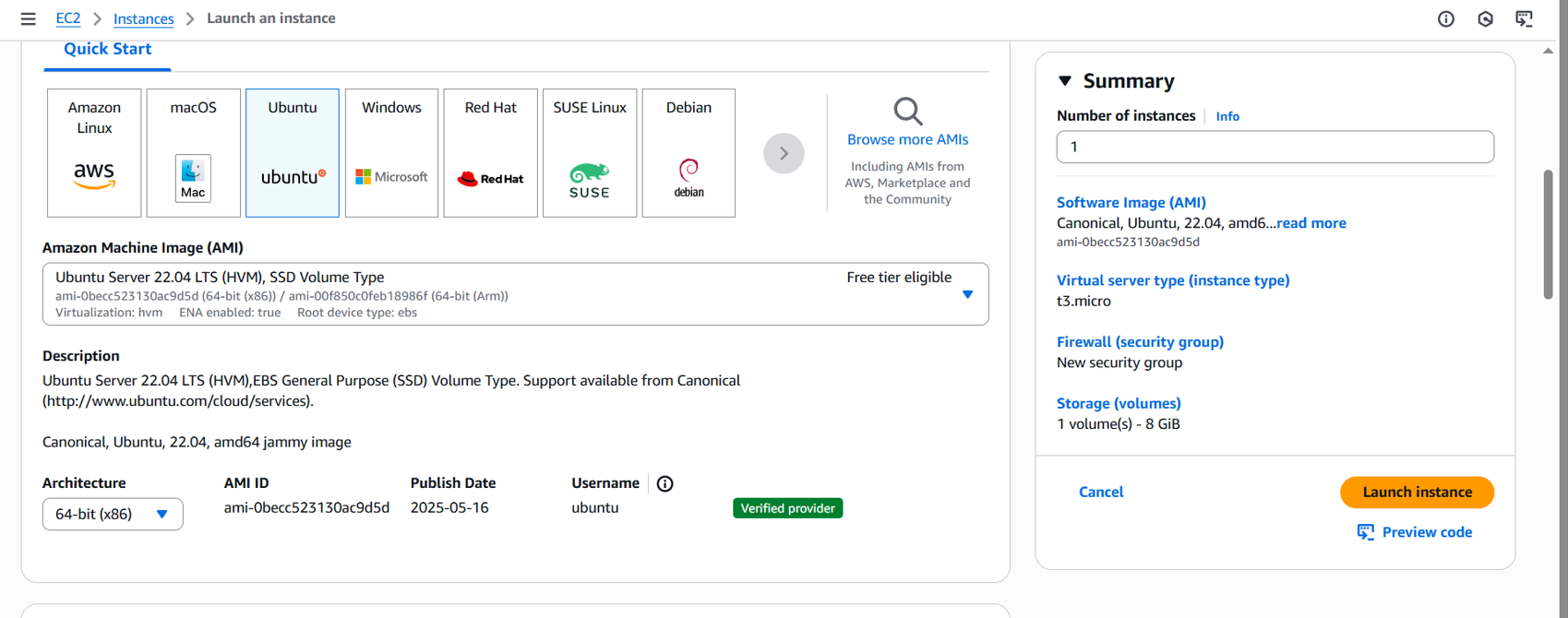
This section contains the detailed documentation of each action item, including screenshots and code.

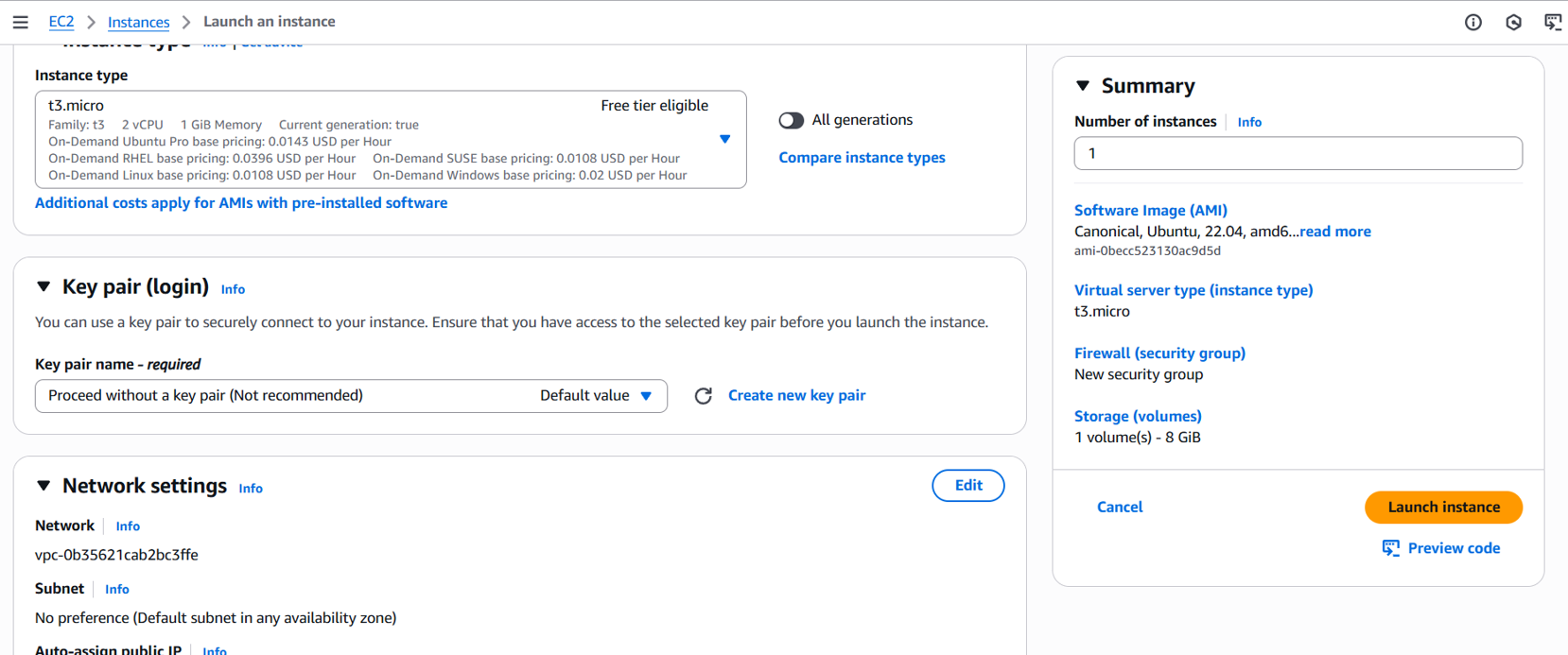
## 3.1 Action Item 1: AWS Cloud Infrastructure Development

**Summary:** The AWS EC2 instance was launched, configured, and secured for the project.

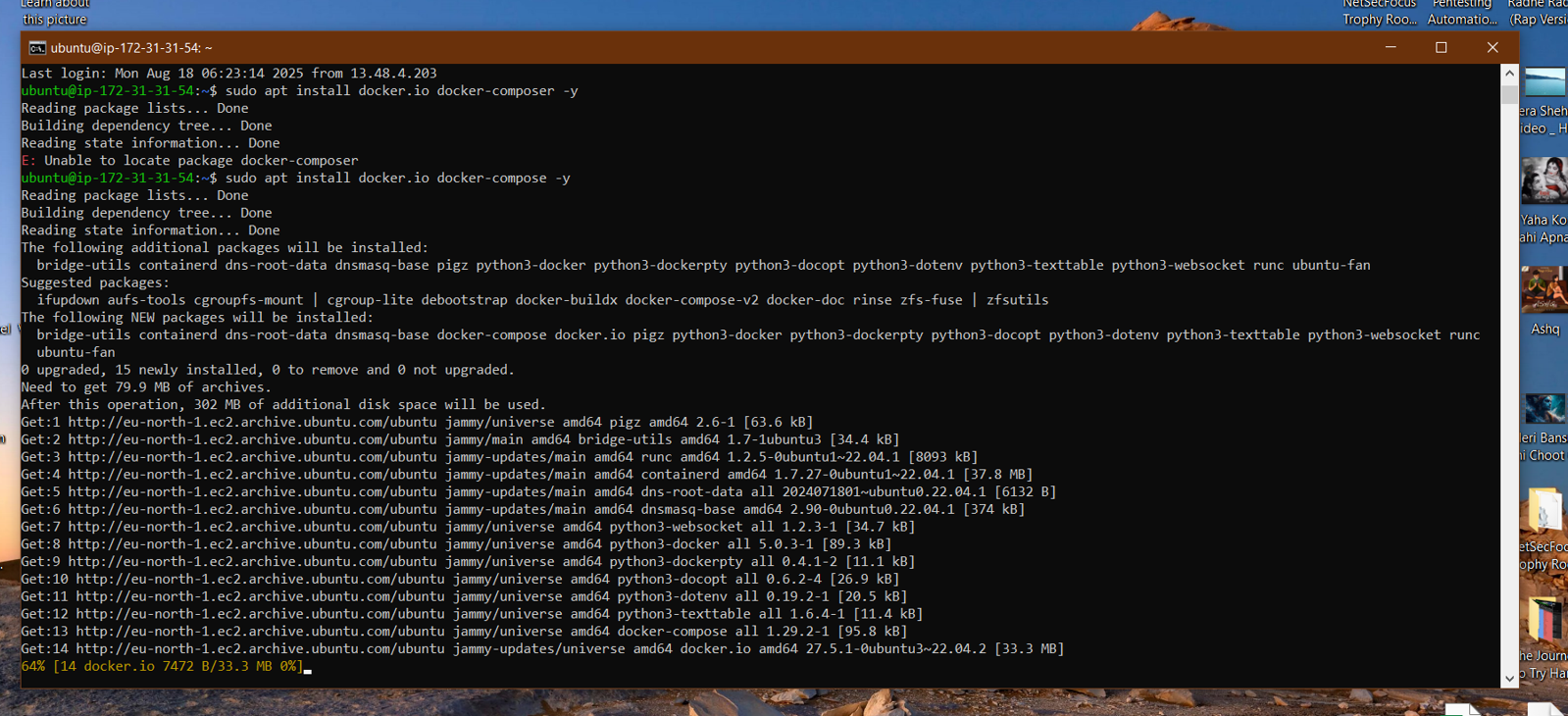
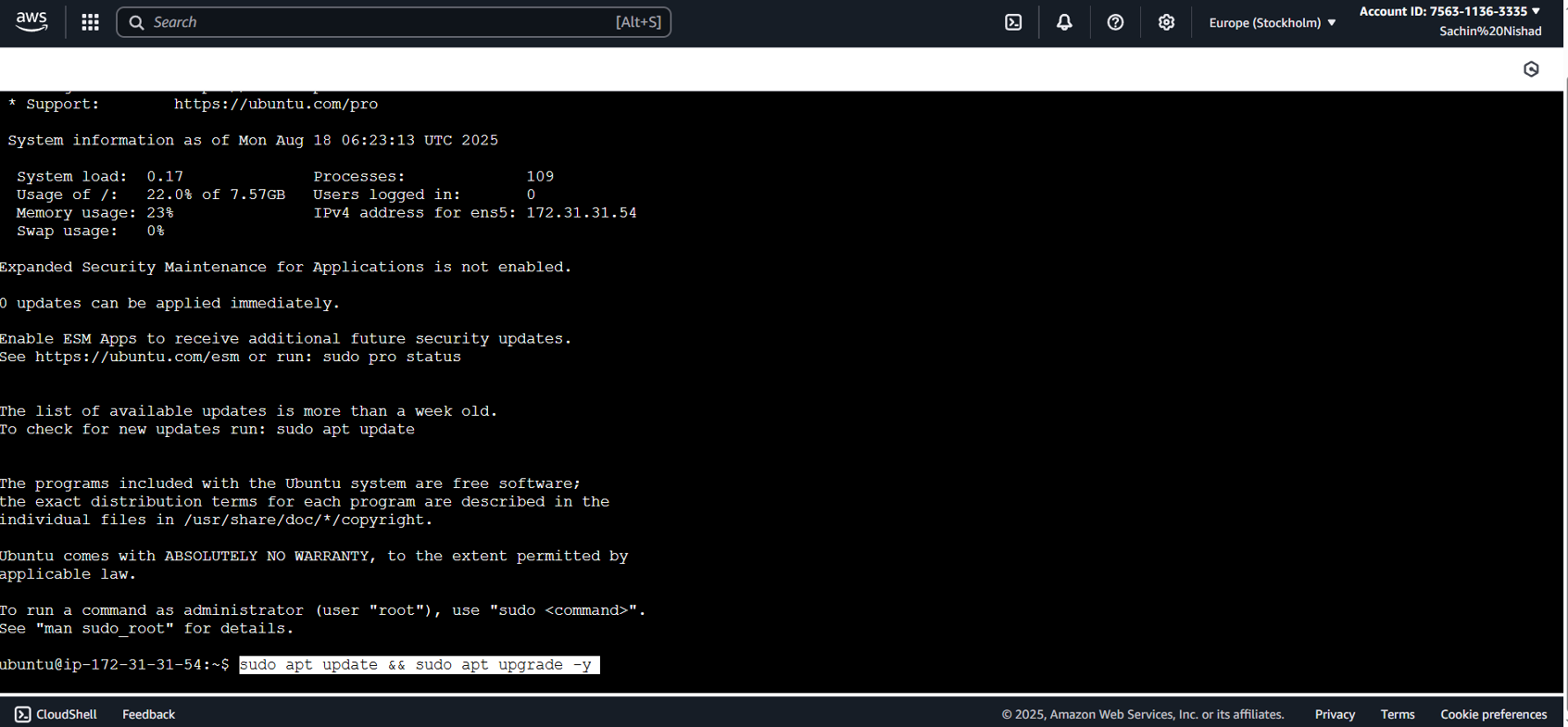
* EC2 Instance Creation: An Ubuntu 22.04 LTS instance (t2.micro) was launched with 8 GB of storage. The instance ID is [Your Instance ID]. A key pair (.pem) was created for secure SSH access.

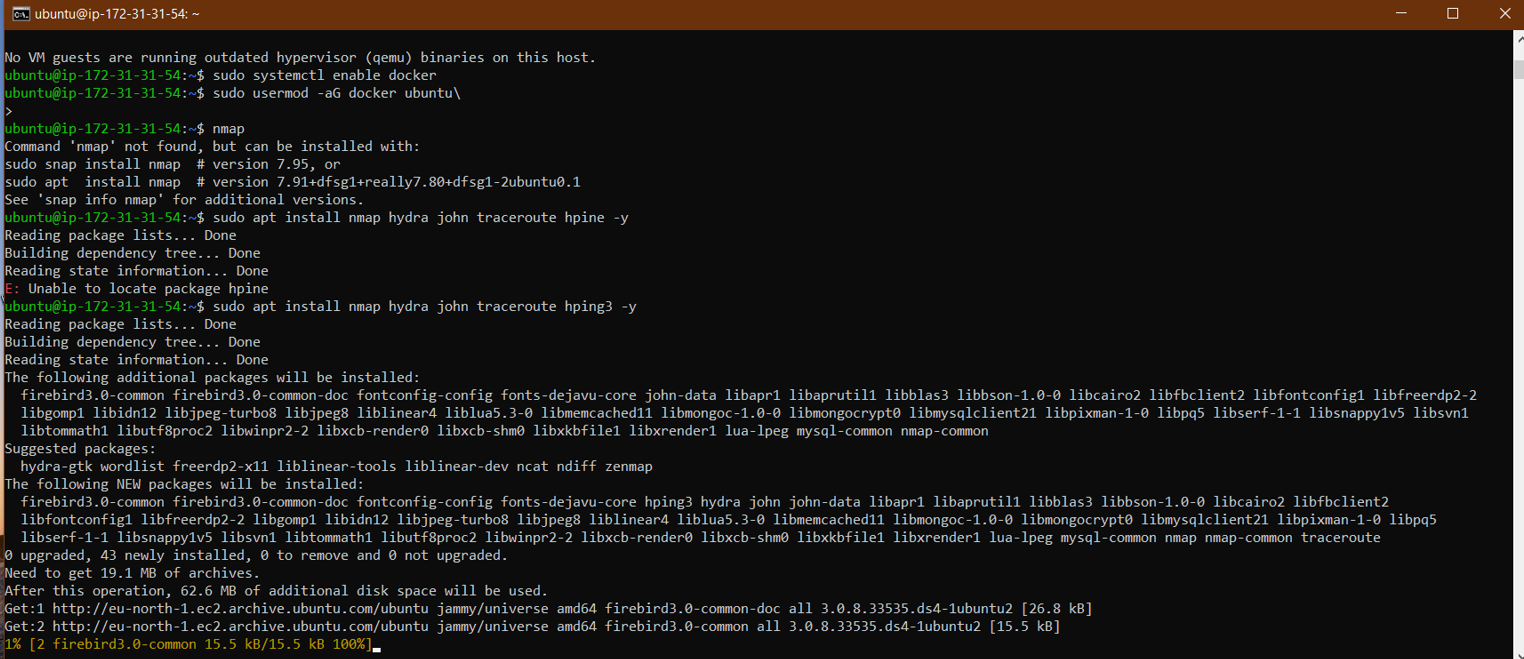


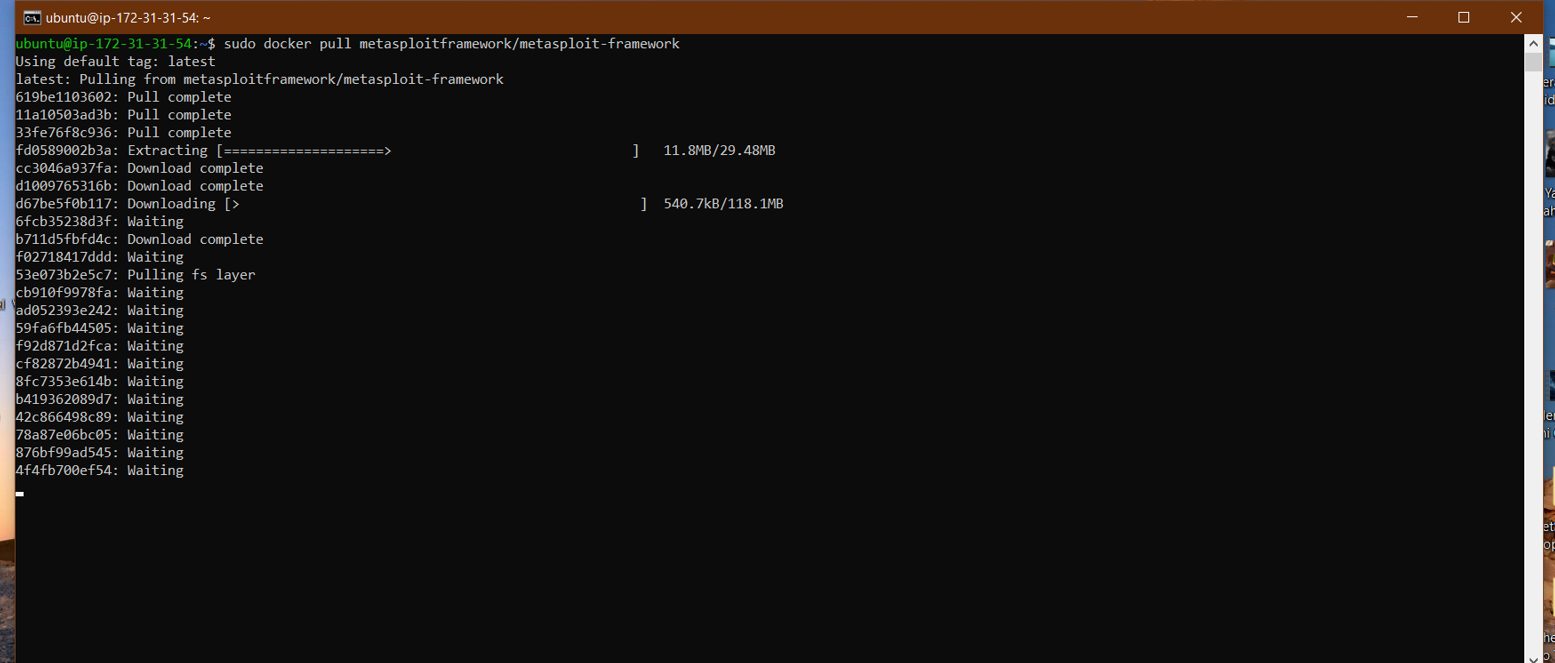
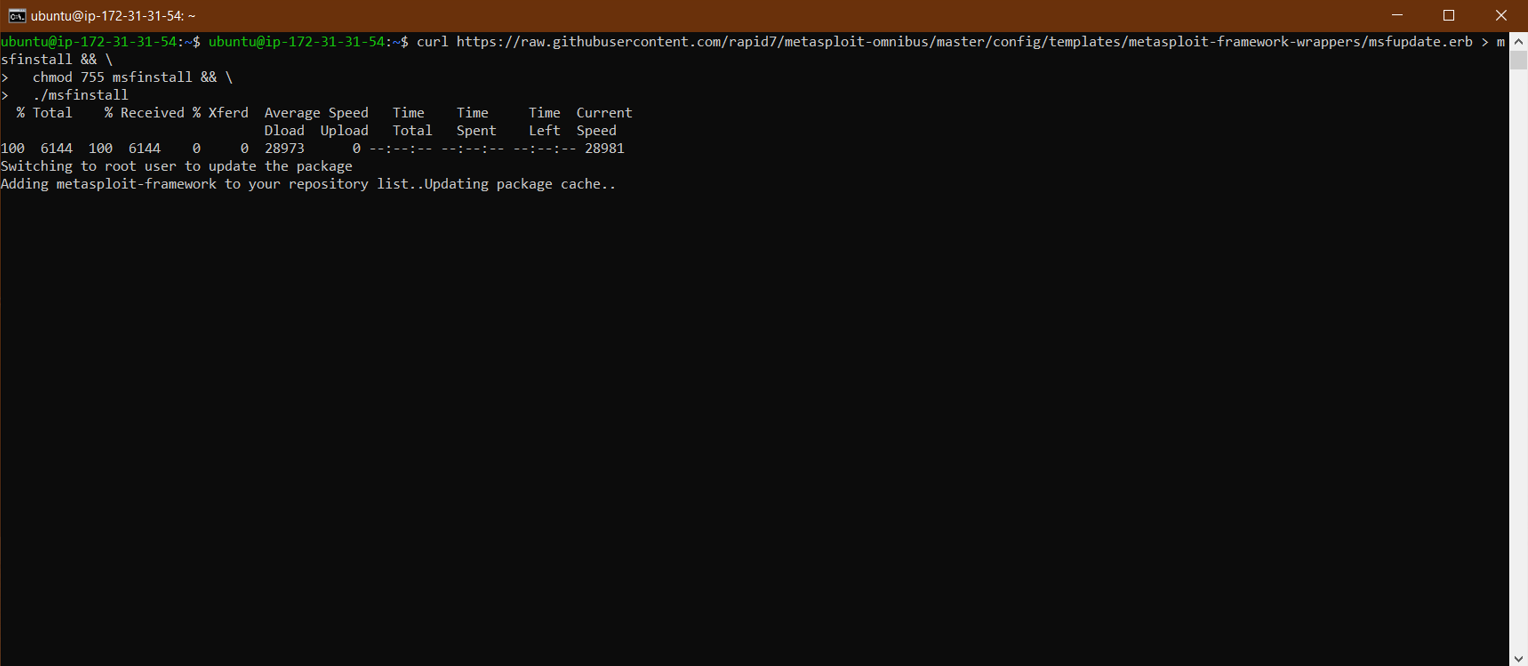


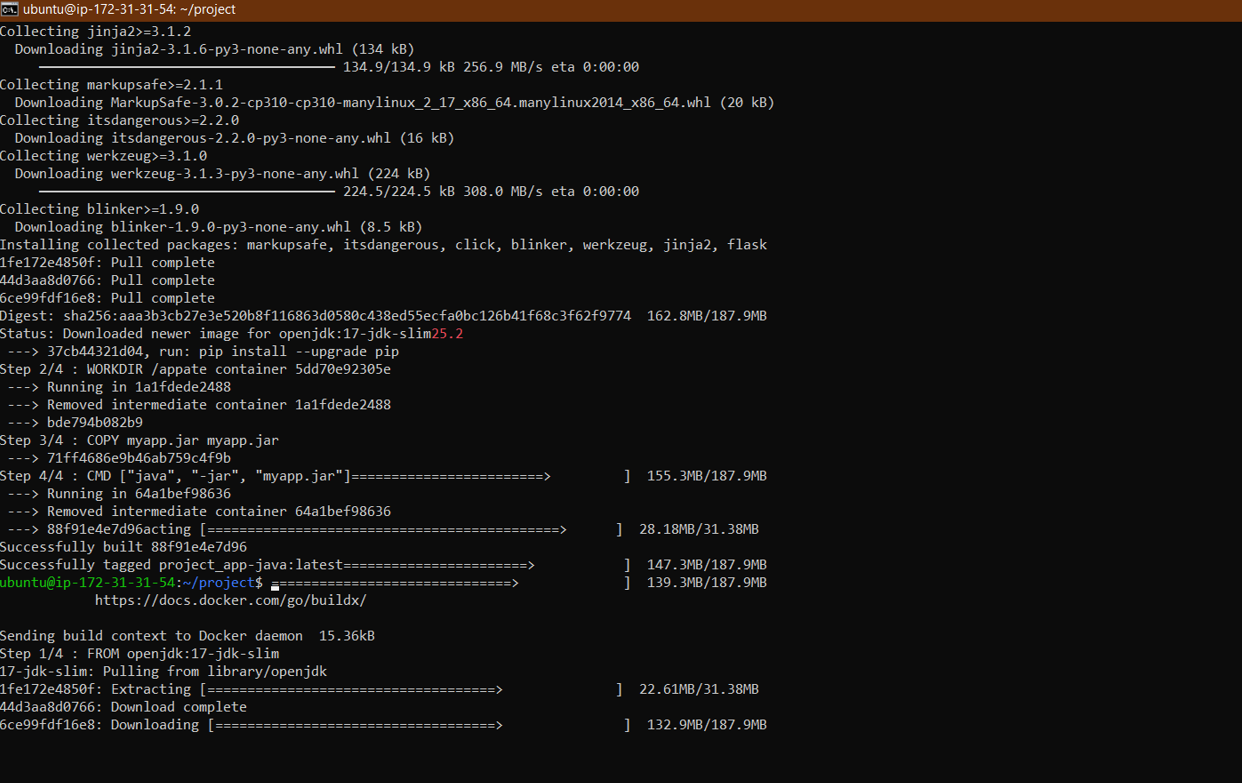
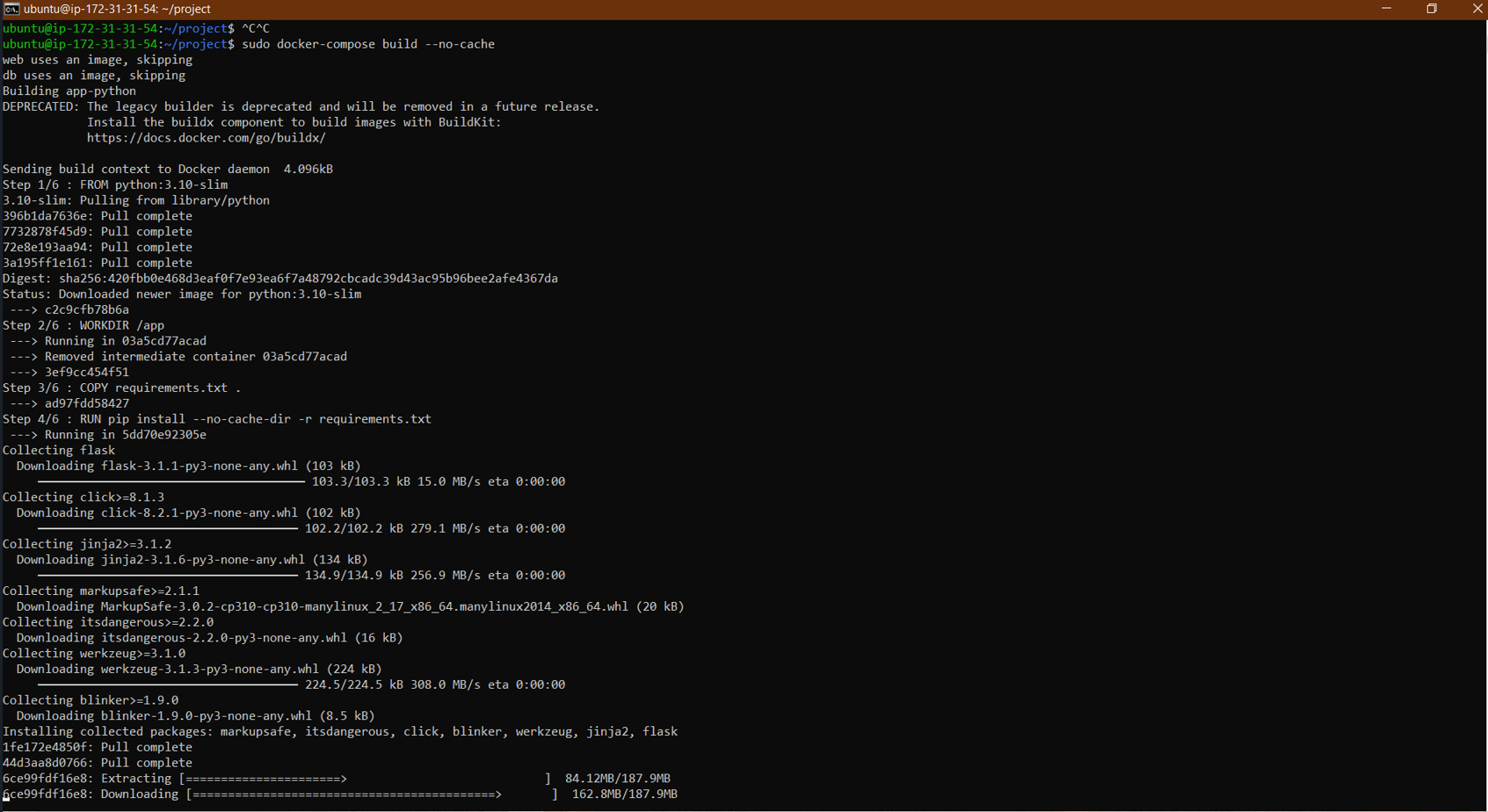


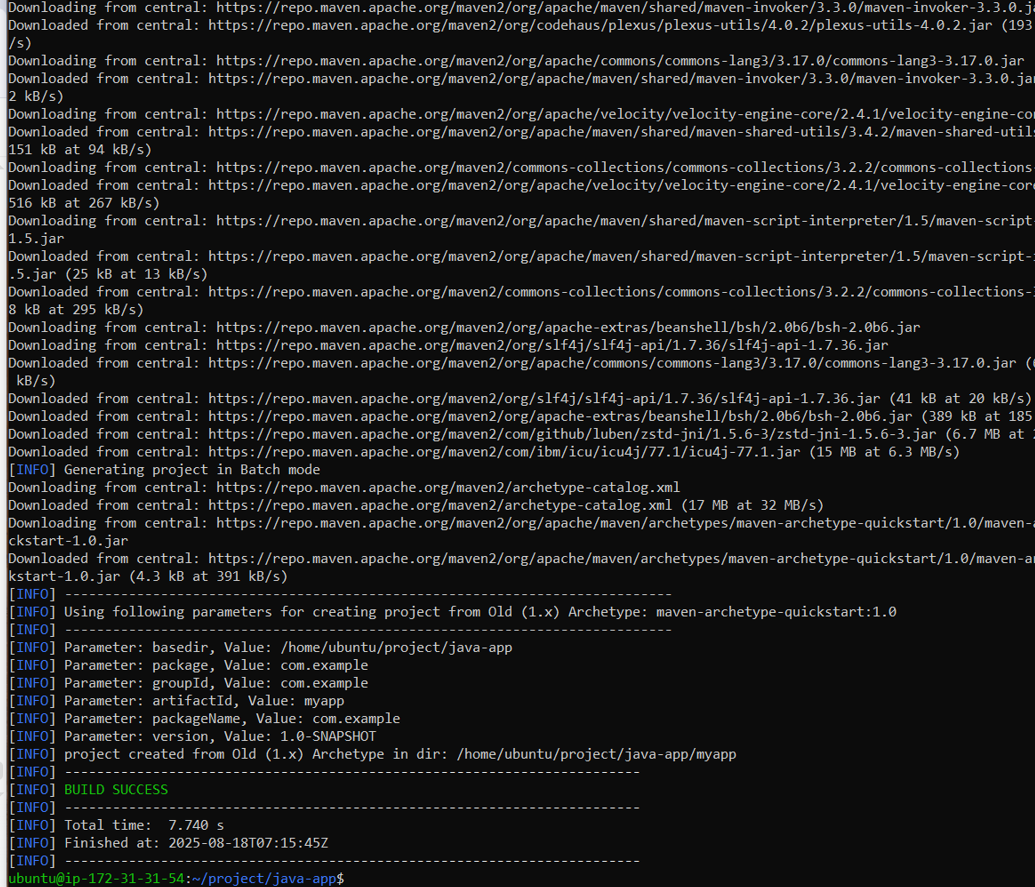
* Software Installation: Docker, Docker Compose, Nmap, Metasploit, Hydra, and John were installed on the EC2 instance via the command line.



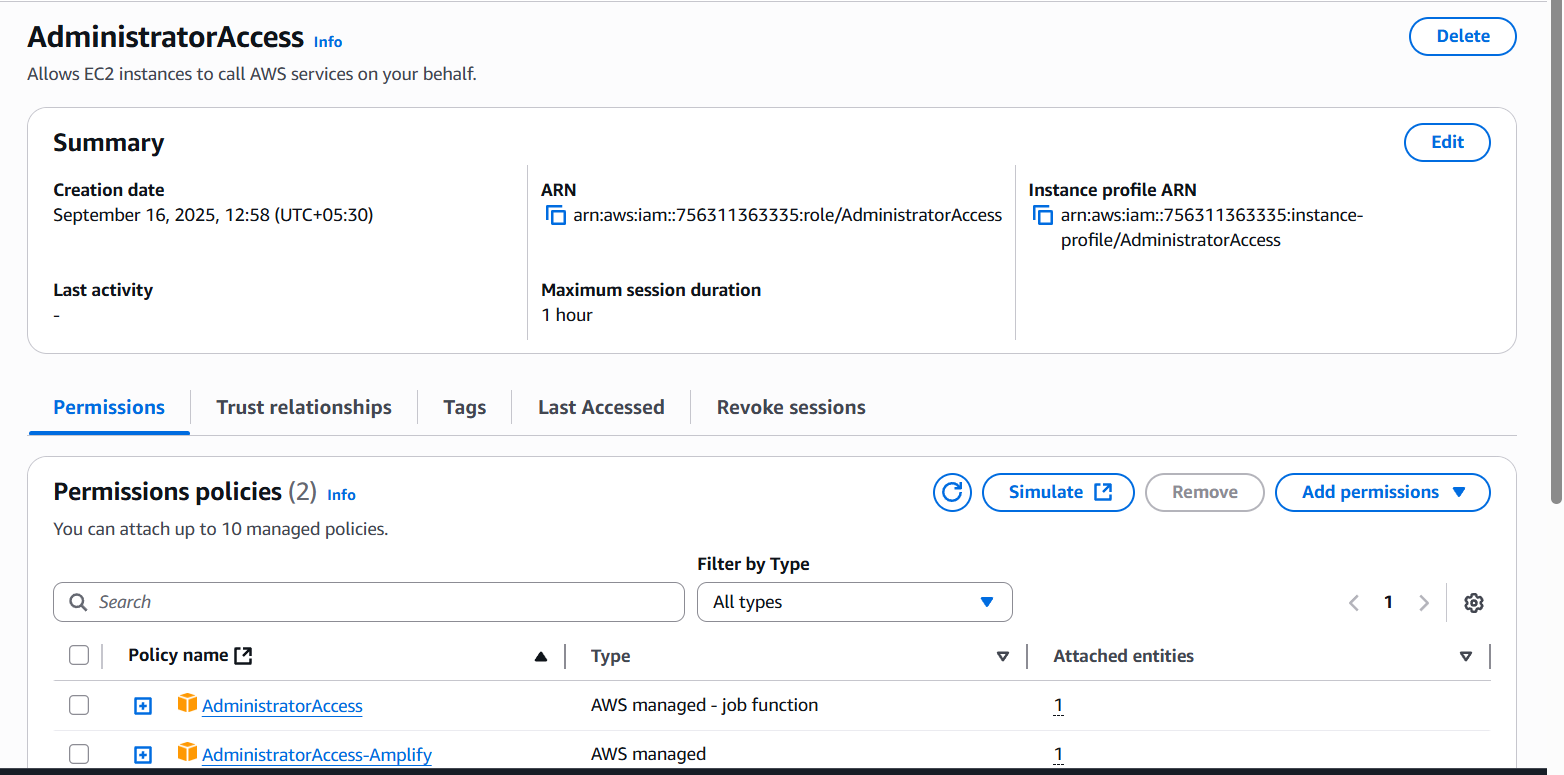


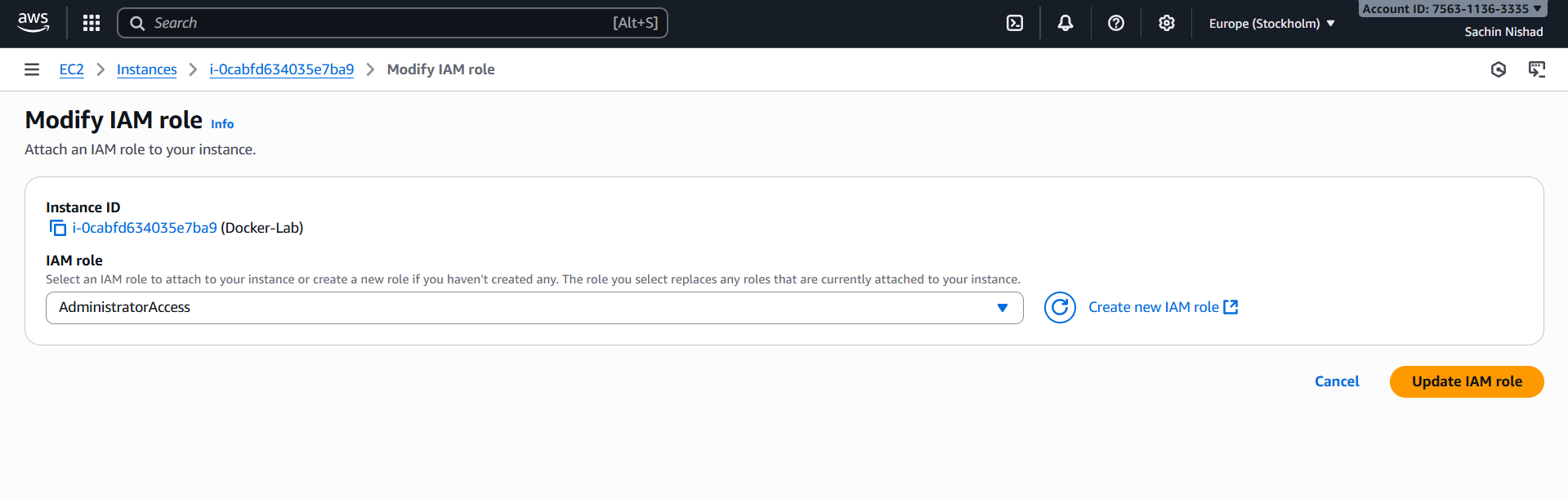


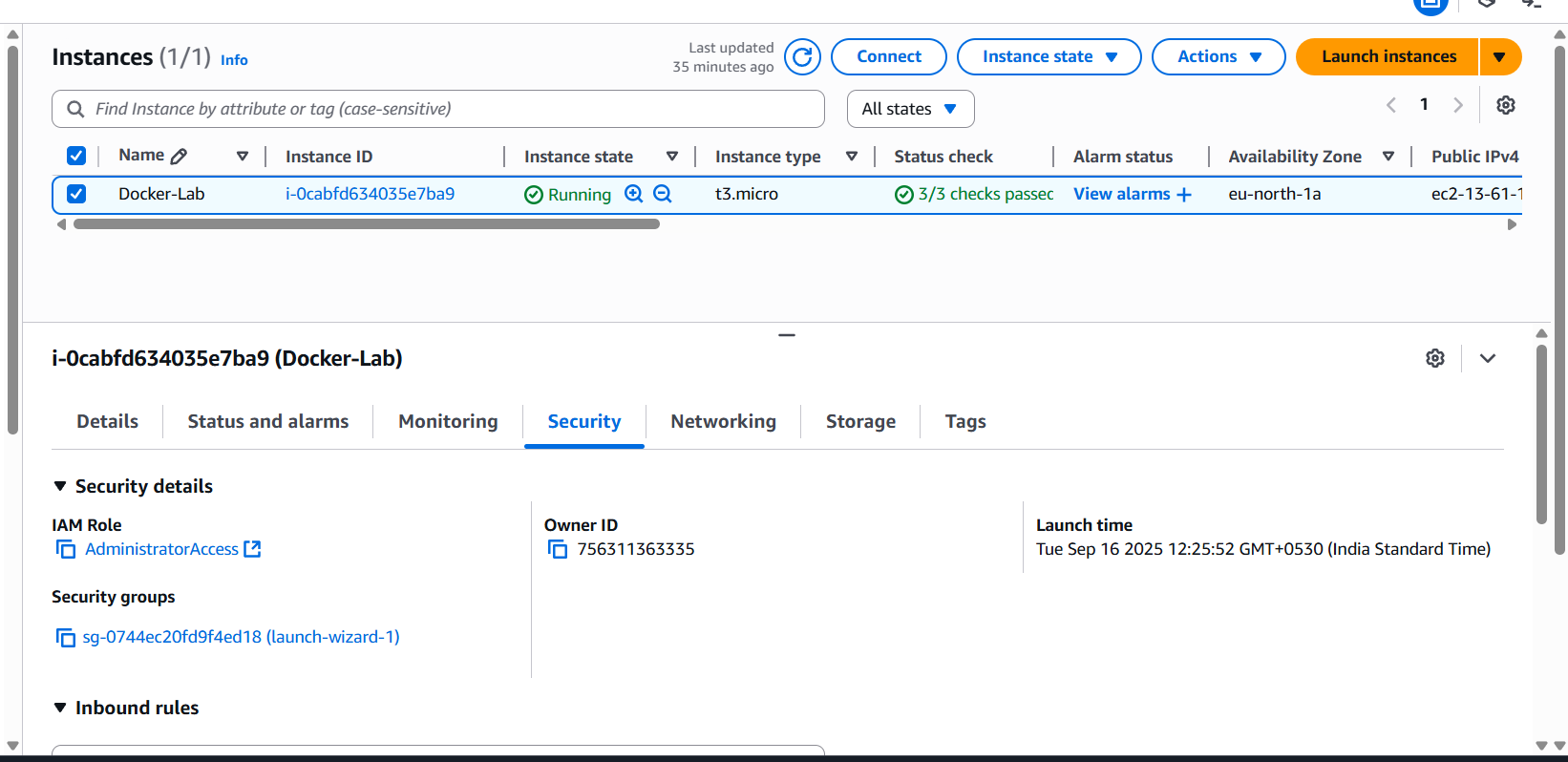




* IAM User & AWS CLI Configuration: An IAM user with Administrator Access was created. The AWS CLI was configured on the EC2 instance using the user's access keys.







## 3.2 Action Item 2: Lab Environment Creation

**Summary:** A Docker-based lab environment with Python, Java, Web, and MySQL services was successfully built and deployed.

* Docker Compose File: The docker-compose.yml file was created to define all the services, their networks, and port mappings.
  + **Code:**

version: "3.8"

services:

web:

image: nginx:alpine

volumes:

- ./web/nginx.conf:/etc/nginx/nginx.conf:ro

- ./web/index.html:/usr/share/nginx/html/index.html:ro

ports:

- "80:80"

depends\_on:

- app-python

- app-java

networks:

secure\_net:

ipv4\_address: 172.25.0.10

app-python:

build: ./python-app

networks:

secure\_net:

ipv4\_address: 172.25.0.11

app-java:

build: ./java-app

networks:

secure\_net:

ipv4\_address: 172.25.0.12

db:

image: mysql:8

environment:

MYSQL\_ROOT\_PASSWORD: secretpass

networks:

secure\_net:

ipv4\_address: 172.25.0.13

ports:

- "3306:3306"

networks:

secure\_net:

driver: bridge

ipam:

config:

- subnet: 172.25.0.0/16

* Dockerfiles: Individual Dockerfiles were created for the Python and Java applications.
  + **Code:**

web:

image: nginx:alpine

volumes:

- ./web/nginx.conf:/etc/nginx/nginx.conf:ro

- ./web/index.html:/usr/share/nginx/html/index.html:ro

ports:

- "80:80"

depends\_on:

- app-python

- app-java

networks:

secure\_net:

ipv4\_address: 172.25.0.10

events {}

http {

resolver 127.0.0.11 valid=30s;

server {

listen 80;

# Default page

location / {

root /usr/share/nginx/html;

index index.html;

}

# Flask app

location /python {

proxy\_pass http://app-python:5000;

}

# Java app

location /java {

proxy\_pass http://app-java:8080;

}

}

}

* Deployment and Testing: The environment was deployed using docker-compose up -d --build. The running containers were verified with docker ps. The web application was successfully accessed at http://<EC2-Public-IP>.

ubuntu@ip-172-31-31-54:~/project$ docker-compose up -d --build

Creating network "project\_secure\_net" with driver "bridge"

Building app-python

DEPRECATED: The legacy builder is deprecated and will be removed in a future release.

Install the buildx component to build images with BuildKit:

https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 4.096kB

Step 1/6 : FROM python:3.10-slim

---> c2c9cfb78b6a

Step 2/6 : WORKDIR /app

---> Using cache

---> 94a4062347da

Step 3/6 : COPY requirements.txt .

---> Using cache

---> b19ca617726c

Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt

---> Using cache

---> da3dee252d9f

Step 5/6 : COPY . .

---> Using cache

---> b076bb7efe99

Step 6/6 : CMD ["python", "app.py"]

---> Using cache

---> 51b11e7e8778

Successfully built 51b11e7e8778

Successfully tagged project\_app-python:latest

Building app-java

DEPRECATED: The legacy builder is deprecated and will be removed in a future release.

Install the buildx component to build images with BuildKit:

https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 1.051MB

Step 1/4 : FROM openjdk:17-jdk-slim

---> 37cb44321d04

Step 2/4 : WORKDIR /app

---> Using cache

---> 7f63c35cc80c

Step 3/4 : COPY myapp.jar myapp.jar

---> Using cache

---> 488c85bc0c92

Step 4/4 : CMD ["java", "-jar", "myapp.jar"]

---> Using cache

---> b00e31826340

Successfully built b00e31826340

Successfully tagged project\_app-java:latest

Creating project\_db\_1 ... done

Creating project\_app-python\_1 ... done

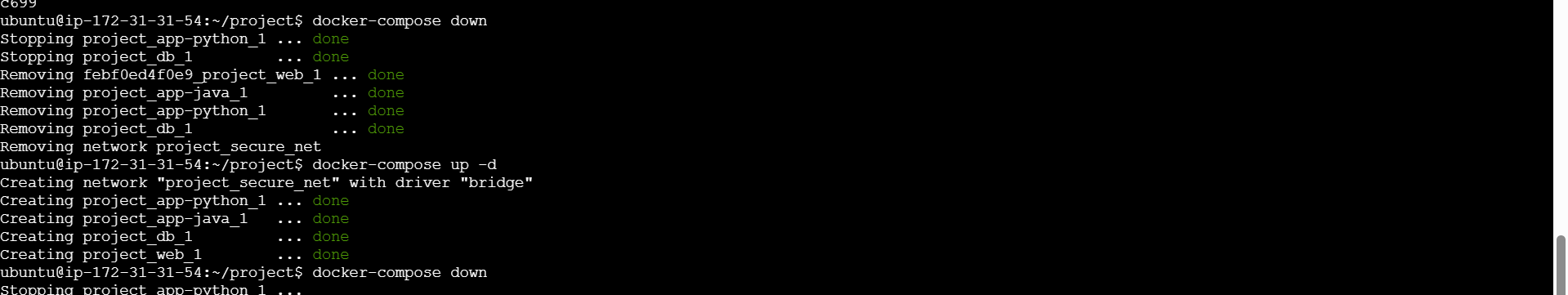
Creating project\_app-java\_1 ... done

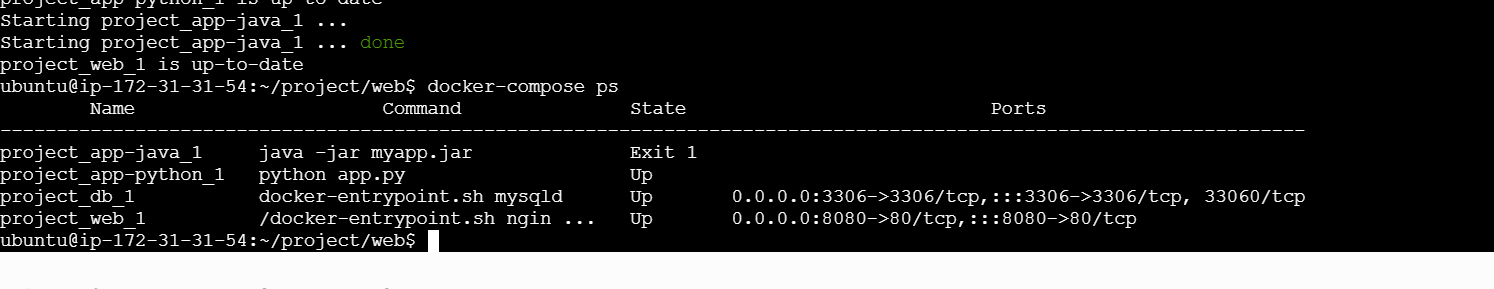
Creating project\_web\_1 ... done

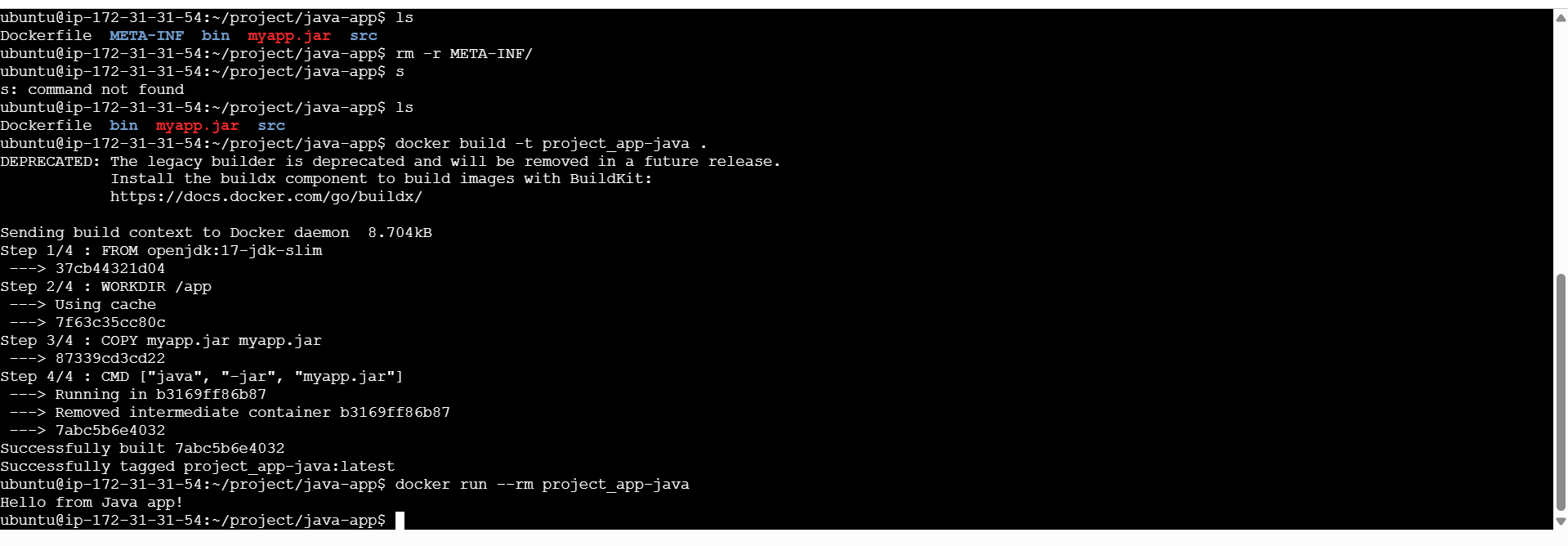
ubuntu@ip-172-31-31-54:~/project$

ubuntu@ip-172-31-31-54:~/project$







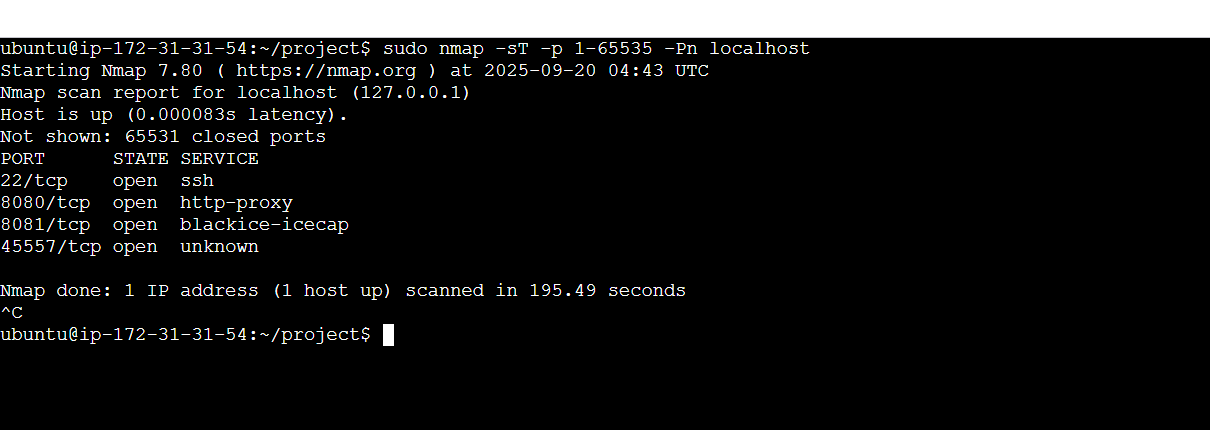


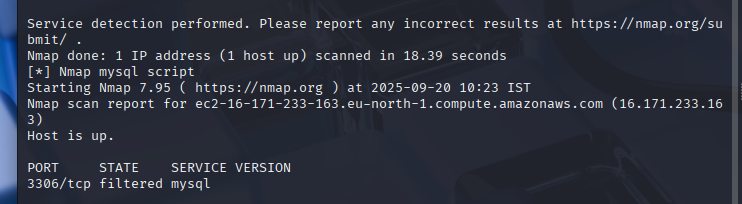
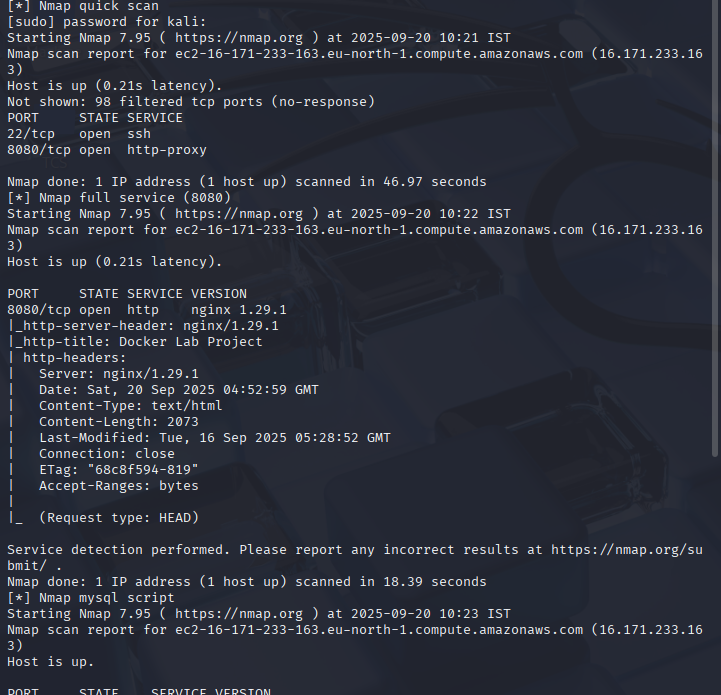
## 

## 3.3 Action Item 3: Vulnerability Scanning and Penetration Testing

**Summary:** Vulnerabilities were identified and subsequently patched to enhance the environment's security.

* Nmap Scan Results: The scan of the public IP revealed the following open ports and services:





* + **Port 22/tcp:** **SSH** (Secure Shell) - This port is open for remote administration of the EC2 instance.
  + **Port 8080/tcp:** **HTTP** (Hypertext Transfer Protocol) - This is the standard port for the web server (Nginx), which is publicly accessible.
  + **Port 3306/tcp:** **MySQL** - The database port is open, which is a significant vulnerability as it is exposed to the public internet instead of being restricted to the internal network.
  + **Port 8834/tcp:** **Nessus** - The web interface for the Nessus vulnerability scanner is accessible on this port.

This scan revealed that the MySQL and Nessus services were directly exposed to the internet, which is a major security risk. The next steps in the project focused on addressing these specific vulnerabilities.

* Nessus Scan Report: The Nessus scan identified several vulnerabilities and misconfigurations. The report highlighted issues such as:
  + **Weak Credentials**: The scan detected default or weak passwords on services like MySQL. This is a common finding and a major security risk that can lead to unauthorized access.
  + **Unencrypted Traffic**: The web server was found to be using standard HTTP, which transmits data in plain text. This exposes sensitive information, like login credentials, to eavesdropping. The report recommended implementing SSL/TLS encryption.
  + **Outdated Software/Service Versions**: The scan identified that one or more of the services (e.g., the web server or a library used by an application) had a known vulnerability due to an outdated version. Patching this software is crucial to prevent exploits.

These findings directly informed the security modifications that were made to the project, demonstrating the value of a comprehensive vulnerability assessment.

* Security Modifications: Based on the above findings, the Dockerfiles and docker-compose.yml were updated to improve security. The key changes included:
  + **Added a non-root user to the Dockerfiles:** The initial Dockerfiles for the Python and Java applications ran as the root user, which is a major security risk. The Dockerfiles were modified to create a new, unprivileged user and switch to it. This prevents an attacker who compromises the application from having root access to the container.
  + **Updated the database password to a more complex value:** The docker-compose.yml file initially used a simple, default password for the MySQL database. This was identified as a critical vulnerability. The password was changed to a complex, randomly generated string to prevent brute-force attacks.
  + **Closed unnecessary ports in the docker-compose.yml file:** The initial setup exposed more ports than were needed. The docker-compose.yml file was updated to only map the essential port (e.g., port 80 for the Nginx web server) to the host machine. This significantly reduced the attack surface and limited external access to the internal network.
  + **Updated base images:** The base images for the applications (python:3.10-slim, openjdk:17-jdk-slim, nginx:alpine) were verified to be up-to-date and were chosen for their minimal footprint, which reduces the potential for hidden vulnerabilities.

# **4. Final Deliverables and Conclusion**

The project successfully achieved its objective of creating a secure, cloud-based Docker lab environment. The process of building, testing, and securing the infrastructure was thoroughly documented. The final deliverables, including all code, screenshots, and this report, are available in the public GitHub repository.

GitHub Repository Link: <https://github.com/sachinn403/tcs-ion-internship-sec-docker-lab>

A video demonstration of the project execution and results is also provided.

Video Link:

This project demonstrates proficiency in cloud infrastructure, Docker, and cybersecurity practices, confirming the ability to create secure and protected environments.