Security+ Container Deployment Project Report.

Step-by-Step Actions:

1. System Setup

I began by setting up an Ubuntu VM in VirtualBox with a host-only network. Once the VM was running, I ensured that internet access was available to allow package downloads and updates.

2. System Update and Upgrade

I updated and upgraded the system packages to make sure the VM had the latest security patches and software updates:

sudo apt update

sudo apt upgrade -y

This minimized vulnerabilities and kept the environment stable.

3. Installed Utilities

Next, I installed the required utilities that I would need for the project:

sudo apt install -y curl nano ufw apt-transport-https ca-certificates software-properties-common gnupg lsb-release

This provided me with tools for file editing, downloading, firewall management, and repository handling.

4. Installed Docker Engine and Docker Compose

I added Docker’s GPG key, repository, and then installed Docker Engine and Docker Compose. I also configured Docker so that it could run without sudo by adding my user to the Docker group:

sudo usermod -aG docker $USER

newgrp docker

This allowed me to work more efficiently with the principle of least privilege in mind.

5. Created Project Files

I created a new directory for my project and wrote a custom index.html file containing a Security+ welcome message: Welcome to Security+, from Bayo. This was the webpage that Nginx would serve.

6. Created the Dockerfile

I wrote a Dockerfile using the nginx:alpine image. In it, I copied my index.html into Nginx’s web directory and exposed port 8080 so the site would be accessible from the VM:

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/

EXPOSE 8080

7. Built and Ran the Container

I built the Docker image and ran the container, mapping port 8080 on the VM to port 80 inside the container:

docker build -t securityplus-nginx .

docker run -d -p 8080:80 securityplus-nginx

docker ps

This confirmed the container was running successfully.

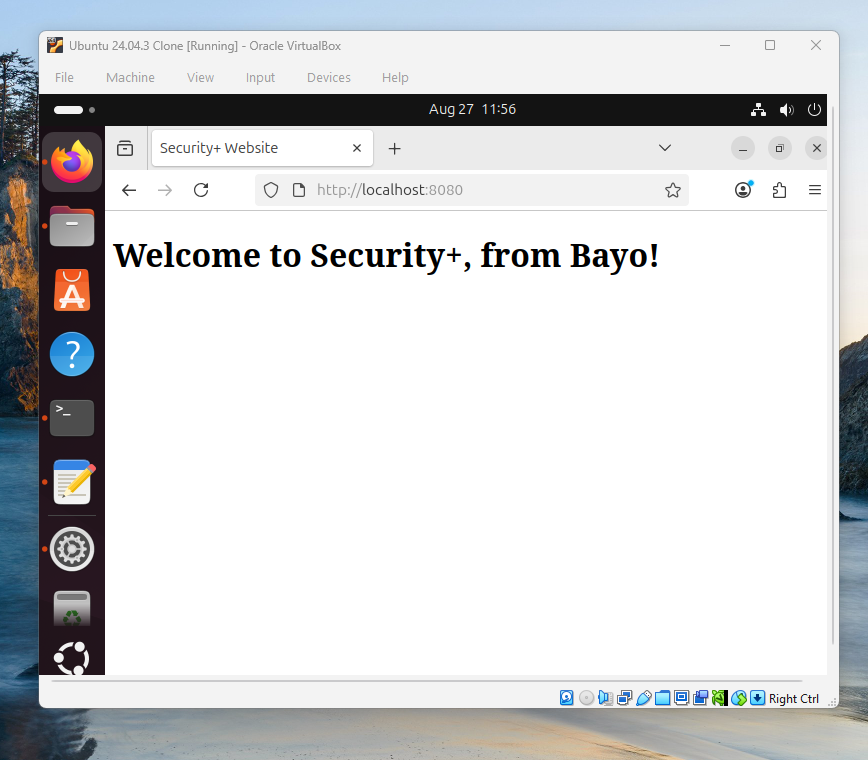
A screenshot of a computer

AI-generated content may be incorrect.

8. Verified Website Locally

From within VM1, I tested the website by visiting:

<http://localhost:8080>



I successfully saw the Security+ welcome message in my browser.

9. Configured OpenSSH

I installed and configured OpenSSH to enable secure remote access to the VM:

sudo apt install -y openssh-server

sudo systemctl enable ssh

sudo systemctl start ssh

sudo systemctl status ssh

A screenshot of a computer

AI-generated content may be incorrect.

10. Configured the Firewall (UFW)

I set up UFW to allow only SSH and HTTP (port 8080) traffic:

sudo ufw allow ssh

sudo ufw allow 8080/tcp

sudo ufw enable

sudo ufw status

This enforced network-level least privilege and secured my VM.

A screenshot of a computer

AI-generated content may be incorrect.

11. Accessed the Website from Another VM

Finally, I found the IP address of VM1 in the host-only network:

ip a | grep inet

Then, from VM2, I accessed the site using:

http://192.168.56.101:8080

This confirmed that my containerized web application was accessible remotely.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Reflections:

1. Advantages of Containerized Environments

Containerized environments like nginx:alpine provide several benefits:

- Isolation: Each container runs independently, which prevents conflicts between applications.

- Security: By isolating services, containers reduce the chance of one service compromising another.

- Reproducibility: Containers make it easy to recreate the same environment across different systems, ensuring consistency.

- Minimized Attack Surface: Containers run only the essential components required for the service, unlike a full host system, which reduces potential entry points for attackers.

2. Research and Accessing the Website from Another VM

To figure out how to access my hosted website from another VM, I researched VirtualBox networking modes and firewall configurations. I discovered that using a host-only network allowed VMs to communicate directly. The main challenge I faced was ensuring that Nginx was listening on all interfaces (0.0.0.0) rather than just localhost. Once I reconfigured this and allowed port 8080 through UFW, I was able to connect from VM2 successfully.

3. Challenging Commands and Strategies

The commands I found most challenging were related to building Docker images and running containers with the correct port mappings. To ensure accuracy, I carefully read error messages, double-checked the Docker documentation, and frequently used ‘docker ps' to confirm the container status. For UFW, I tested the steps one by one using 'ufw status' to verify that the rules were active.

[Security+ Container Deployment Project Report. | by Bayoakins | Aug, 2025 | Medium](https://medium.com/@bayoakins_68523/security-container-deployment-project-report-fd9e55dfccff)