Lesson 07 Demo 05

Securing Microservices Using Docker

Objective: To set up and secure microservices using Docker for improved deployment and management in a containerized environment

Tools required: Docker, Docker Compose, Python 3.x, and Flask

Prerequisites: None

Steps to be followed:

1. Create microservices
2. Create a requirements file for dependencies
3. Create a Dockerfile for each microservice
4. Create a Docker compose file and run the setup

Step 1: Create microservices

1.1 Switch to the root user using the following command:
sudo su

```
sakshiguptasimp@ip-172-31-32-167:~$ sudo su
root@ip-172-31-32-167:/home/sakshiguptasimp#
```

1.2 Create a directory for the microservices using the following command:
mkdir microservices-lab/

```
root@ip-172-31-32-167:/home/sakshiguptasimp# mkdir microservices-lab/
```

1.3 Navigate inside the created directory using the following command:
cd microservices-lab

```
root@ip-172-31-32-167:/home/sakshiguptasimp# cd microservices-lab
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab#
```

1.4 Create two directories, **service-a** and **service-b**, for the respective microservices using the following commands:
mkdir service-a
mkdir service-b

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# mkdir service-a
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# mkdir service-b
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab#
```

1.5 Create a Python file in the **service-a** directory using the following command:
vi service-a/app.py

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-a/app.py
```

1.6 Set up a simple Flask web application for **service-a** using the following code:
from flask import Flask

from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def hello_world():
secret = os.getenv('SERVICE_A_SECRET', 'default_secret')
return f'Hello from Service A! Secret: {secret}'

if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000)

```
from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def hello_world():
    secret = os.getenv('SERVICE_A_SECRET', 'default_secret')
    return f'Hello from Service A! Secret: {secret}'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

The use of environment variables, particularly for secrets management (SERVICE_A_SECRET), is a good practice for handling sensitive information.

1.7 Create a Python file in the **service-b** directory using the following command:
vi service-b/app.py

```
root@ip-172-31-32-167:/home/sakshiguptasimp# vi service-b/app.py
```

1.8 Set up a simple Flask web application for **service-b** using the following code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service B!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service B!'

@app.route('/health')
def health():
    return 'OK', 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service B!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service B!'

@app.route('/health')
def health():
    return 'OK', 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 2: Create a requirements file for dependencies

2.1 Create a requirements file in the **service-a** directory using the following command:
vi service-a/requirements.txt

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-a/requirements.txt
```

2.2 Specify the Flask version required for **service-a** using the following code:
Flask==2.0.1

```
Flask==2.0.1
~
```

2.3 Create a requirements file in the **service-b** directory using the following command:
vi service-b/requirements.txt

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-b/requirements.txt
```

2.4 Specify the Flask version required for **service-b** using the following code:
Flask==2.0.1

```
Flask==2.0.1
~
```

Step 3: Create a Dockerfile for each microservice

3.1 Create a Dockerfile in service-a directory using the following command:
vi service-a/Dockerfile

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-a/Dockerfile
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab#
```

3.2 Enter the following code in the created Dockerfile:
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
~
~
~
```

This Dockerfile sets up a container using the Python 3.8-slim image, sets the working directory to **/app**, installs dependencies from **requirements.txt**, copies the current directory contents to the container, and runs **app.py** with Python.

3.3 Create a Dockerfile in **service-b** directory using the following command:
vi service-b/Dockerfile

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-b/Dockerfile
```

3.4 Enter the following code in the created Dockerfile:

FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
~
~
~
~
```

This Dockerfile creates a container using Python 3.8-slim, sets the working directory to **/app**, copies **requirements.txt,** installs dependencies, copies all files from the current directory, and specifies the command to run **app.py** using Python.

Step 4: Create a Docker compose file and run the setup

4.1 Create a Docker compose file using the following command:

vi docker-compose.yml

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi docker-compose.yml
```

4.2 Add the following code in the created **docker-compose** file:

version: '3.8'

services:
service-a:
build: ./service-a
ports:
- "5000:5000"
secrets:
- service_a_secret
networks:
- frontend
deploy:
resources:
limits:

```
        memory: 100M
    restart_policy:
      condition: on-failure
  security_opt:
    - no-new-privileges:true

 service-b:
  build: ./service-b
  ports:
    - "5001:5000"
  networks:
    - backend
  deploy:
    resources:
      limits:
        memory: 100M
    restart_policy:
      condition: on-failure
  security_opt:
    - no-new-privileges:true

secrets:
 service_a_secret:
  file: ./secrets/service-a-secret.txt

networks:
 frontend:
 backend:
```

```yaml
version: '3.8'

services:
  service-a:
    build: ./service-a
    ports:
      - "5000:5000"
    secrets:
      - service_a_secret
    networks:
      - frontend
    deploy:
      resources:
        limits:
          memory: 100M
      restart_policy:
        condition: on-failure
    security_opt:
      - no-new-privileges:true

  service-b:
    build: ./service-b
    ports:
      - "5001:5000"
    networks:
      - backend
    deploy:
```

Setting memory limits (resources: limits: memory: 100M) in the Docker Compose file helps prevent resource exhaustion attacks.
The use of security_opt with no-new-privileges:true enhances security by preventing privilege escalation within containers.

4.3 Build and run the setup using the following command:
docker-compose up --build

```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# docker-compose up --build
WARNING: Service "service-a" uses an undefined secret file "/home/sakshiguptasimp/microservices-lab/secrets/ser
 should be created "/home/sakshiguptasimp/microservices-lab/secrets/service-a-secret.txt"
Creating network "microservices-lab_frontend" with the default driver
Creating network "microservices-lab_backend" with the default driver
Building service-a
```

```
Building service-b
[+] Building 3.6s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 183B
 => [internal] load metadata for docker.io/library/python:3.8-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:c177b5b444d6913678d80bd26af131187de166cd68ac6660
 => [internal] load build context
 => => transferring context: 242B
 => CACHED [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt requirements.txt
 => [4/5] RUN pip install -r requirements.txt
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:d151f73f63c27d43d63c0017bddb87000dd3e9ff4b3b9691e022a83e7d852d48
 => => naming to docker.io/library/microservices-lab_service-b
Creating microservices-lab_service-b_1 ...
Creating microservices-lab_service-a_1 ... error

Creating microservices-lab_service-b_1 ... done
not exist: /home/sakshiguptasimp/microservices-lab/secrets/service-a-secret.txt

ERROR: for service-a  Cannot create container for service service-a: invalid mount config for type "bind
shiguptasimp/microservices-lab/secrets/service-a-secret.txt
ERROR: Encountered errors while bringing up the project.
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab#
```

By following these steps, you have successfully set up and secured microservices using Docker for improved deployment and management in a containerized environment.