Lab 9: Docker compose, volumes, and cache

Ahmed Baha Eddine Alimi

Assignment Report

SD-01

I. Questions to Answer：

1. Secret management is crucial for the security of applications. Explain how you would securely manage secrets for a containerized FastAPI application running in a production environment. Discuss any additional measures or tools you would use to enhance the security of the secret management process.

- Use a Secrets Manager (e.g., AWS Secrets Manager, HashiCorp Vault, or Azure Key Vault) to store and retrieve secrets dynamically.
- Environment Variables (via Docker/Kubernetes secrets) for runtime injection—avoid hardcoding.
- Encryption for secrets at rest and in transit (TLS, KMS).
- Least Privilege Access—restrict who/apps can access secrets (IAM, RBAC).
- Rotate Secrets Regularly—automate rotation via tools like Vault or AWS Secrets Manager.
- Audit Logs—track access/changes to secrets for security compliance.

Key Tools: Vault, AWS Secrets Manager, SOPS, Kubernetes External Secrets.

2. Using the infrastructure configured in "Task 5: More advanced use case", demonstrate how to scale the web service to run three instances of the FastAPI application (Simply add one more instance). Configure your first name as the SERVER_ID of this new instance

```
allimi@allimi-VirtualBox:~/myapp$ nano docker-compose.yml
```

allimi@allimi-VirtualBox: ~/myapp

GNU nano 7.2 docker-compose.yml *

```
version: '3.8'
services:
  db:
    image: postgres:13
    volumes:
      - ./postgres-init:/docker-entrypoint-initdb.d
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}

  web1:
    build: ./app
    volumes:
      - ./app:/app
    environment:
      DATABASE_URL: ${DATABASE_URL}
      SERVER_ID: SERVER-1
    depends_on:
      - db

  web2:
    build: ./app
    volumes:
      - ./app:/app
    environment:
      DATABASE_URL: ${DATABASE_URL}
      SERVER_ID: SERVER-2
    depends_on:
      - db

  web3:
    build: ./app
    volumes:
      - ./app:/app
    environment:
      DATABASE_URL: ${DATABASE_URL}
      SERVER_ID: AhmedBahaEddine
    depends_on:
      - db

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./nginx:/etc/nginx/conf.d
    depends_on:
      - web1
      - web2
      - web3

volumes:
  postgres_data:
```

```
allimi@allimi-VirtualBox:~/myapp/nginx$ nano default.conf
```

allimi@allimi-VirtualBox: ~/myapp/nginx

GNU nano 7.2 default.conf *

```
upstream webapp {
    server web1:8000;
    server web2:8000;
    server web3:8000;  # Added new web instance
}

server {
    listen 80;

    location / {
        proxy_pass http://webapp;
    }
}
```

```
allimi@allimi-VirtualBox:~/myapp$ sudo docker-compose up --build -d
[sudo] password for allimi:
[+] Building 4.3s (18/24)
 => [myapp-web1 internal] load build definition from Dockerfile           0.5s
 => => transferring dockerfile: 244B                                      0.1s
 => [myapp-web2 internal] load build definition from Dockerfile           0.8s
 => => transferring dockerfile: 244B                                      0.1s
 => [myapp-web3 internal] load metadata for docker.io/library/python:3.9-  1.9s
 => [myapp-web3 internal] load build definition from Dockerfile           0.3s
 => => transferring dockerfile: 244B                                      0.0s
 => [myapp-web3 internal] load .dockerignore                              0.3s
 => => transferring context: 2B                                           0.0s
 => [myapp-web2 internal] load .dockerignore                              0.4s
 => => transferring context: 2B                                           0.0s
 => [myapp-web1 internal] load .dockerignore                              0.3s
 => => transferring context: 2B                                           0.0s
 => [myapp-web1 1/5] FROM docker.io/library/python:3.9-slim@sha256:e52ca5  0.0s
 => [myapp-web3 internal] load build context                              0.6s
 => => transferring context: 2.90kB                                       0.0s
 => [myapp-web2 internal] load build context                              0.7s
 => => transferring context: 2.90kB                                       0.1s
 => [myapp-web1 internal] load build context                              0.7s
 => => transferring context: 2.90kB                                       0.1s
 => CACHED [myapp-web3 2/5] WORKDIR /app                                  0.0s
 => CACHED [myapp-web3 3/5] COPY ./requirements.txt requirements.txt      0.0s
 => CACHED [myapp-web3 4/5] RUN pip install --no-cache-dir -r requirement  0.0s
 => CACHED [myapp-web1 5/5] COPY ./ /app                                  0.0s
 => [myapp-web2] exporting to image                                       0.2s
 => => exporting layers                                                   0.0s
 => => writing image sha256:aa43f235d9a16526f68b2eec0c5eeeebf161282f839dd  0.1s
 => => naming to docker.io/library/myapp-web2                             0.0s
 => [myapp-web1] exporting to image                                       0.2s
 => => exporting layers                                                   0.0s
 => => writing image sha256:6399d68a94e203e94ac876f62b830e74171fcd6a7309d  0.1s
 => => naming to docker.io/library/myapp-web1                             0.1s
 => [myapp-web3] exporting to image                                       0.2s
 => => exporting layers                                                   0.0s
 => => writing image sha256:79cfe1f214da391716ccf493abfef5325fdb1a356a3b4  0.1s
 => => naming to docker.io/library/myapp-web3                             0.1s
[+] Running 5/5
 ⠿ Container myapp-db-1     Started                                                                  10.7s
 ⠿ Container myapp-web3-1   Started                                                                  16.4s
 ⠿ Container myapp-web1-1   Started                                                                  16.7s
 ⠿ Container myapp-web2-1   Started                                                                  16.4s
 ⠿ Container myapp-nginx-1  Started                                                                  15.0s
```

```
allimi@allimi-VirtualBox:~/myapp$ sudo docker-compose ps
NAME                IMAGE               COMMAND                  SERVICE             CREATED             STATUS              PORTS
myapp-db-1          postgres:13         "docker-entrypoint.s…"   db                  34 seconds ago      Up 22 seconds       5432/tcp
myapp-nginx-1       nginx:latest        "/docker-entrypoint.…"   nginx               28 seconds ago      Up 12 seconds       0.0.0.0:80->80/tcp, :::80->80/tcp
myapp-web1-1        myapp-web1          "uvicorn main:app --…"   web1                33 seconds ago      Up 16 seconds
myapp-web2-1        myapp-web2          "uvicorn main:app --…"   web2                33 seconds ago      Up 16 seconds
myapp-web3-1        myapp-web3          "uvicorn main:app --…"   web3                33 seconds ago      Up 16 seconds
```

```
allimi@allimi-VirtualBox:~/myapp$ curl http://localhost
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}allimi@allimi-VirtualBox:~/myapp$ for i in {1..10}; do curl http://localhost; echo ""; done for
i in {1..10}; do curl http://localhost; echo ""; done
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-2","note":"Hello from Dockerized Postgres!"}
{"server_id":"AhmedBahaEddine","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-1","note":"Hello from Dockerized Postgres!"}
{"server_id":"SERVER-2","note":"Hello from Dockerized Postgres!"}
{"server_id":"AhmedBahaEddine","note":"Hello from Dockerized Postgres!"}
allimi@allimi-VirtualBox:~/myapp$
```

3. Design a Docker deployment that utilizes volumes for persistent data storage and manipulation. Your project will consist of a database container and a secondary container tasked with periodically backing up the database to a volume.

```
allimi@allimi-VirtualBox:~$ mkdir docker-postgres-backup
allimi@allimi-VirtualBox:~$ cd docker-postgres-backup
```

```
allimi@allimi-VirtualBox:~/docker-postgres-backup$ nano docker-compose.yml
```

allimi@allimi-VirtualBox: ~/docker-postgres-backup

GNU nano 7.2 docker-compose.yml *

```
version: '3.8'

services:
  postgres:
    image: postgres:15
    container_name: my_postgres
    restart: always
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydatabase
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - my_network

  backup:
    build: ./backup
    container_name: db_backup
    restart: always
    volumes:
      - backup_data:/backup
      - db_data:/var/lib/postgresql/data:ro
    depends_on:
      - postgres
    networks:
      - my_network

volumes:
  db_data:
  backup_data:

networks:
  my_network:
```

```
allimi@allimi-VirtualBox:~/docker-postgres-backup$ mkdir backup
allimi@allimi-VirtualBox:~/docker-postgres-backup$ cd backup
```

```
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$ nano Dockerfile
```

allimi@allimi-VirtualBox: ~/docker-postgres-backup/backup

GNU nano 7.2 Dockerfile *

```
FROM alpine:latest

RUN apk add --no-cache postgresql-client \
    && apk add --no-cache bash \
    && apk add --no-cache crond

COPY backup.sh /backup.sh
COPY crontab /etc/crontabs/root

RUN chmod +x /backup.sh

CMD ["crond", "-f"]
```

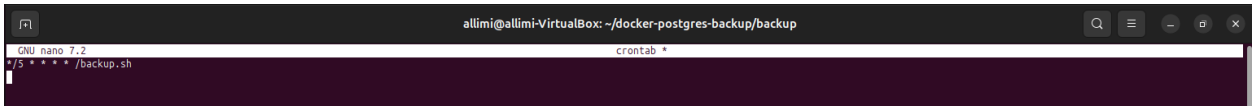
```
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$ nano backup.sh
```

allimi@allimi-VirtualBox: ~/docker-postgres-backup/backup

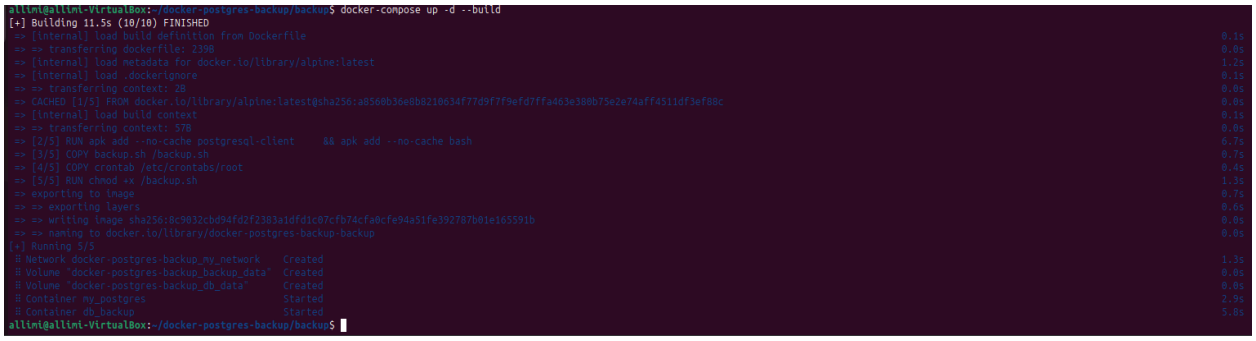
GNU nano 7.2 backup.sh

```
#!/bin/bash
PGPASSWORD=mypassword pg_dump -h my_postgres -U myuser -d mydatabase --no-owner --no-privileges --format=p > /backup/db_backup_$(date +"%Y-%m-%d_%H-%M-%S").sql
```

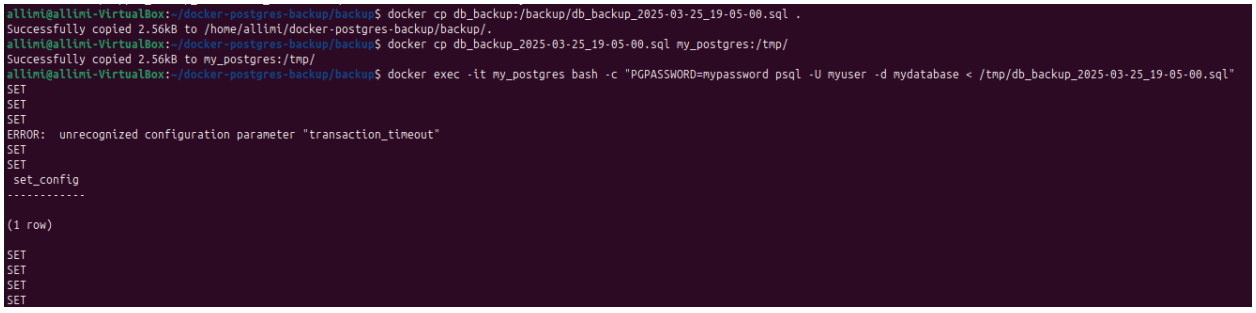
```
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$ nano crontab
```




```
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$ cd ..
allimi@allimi-VirtualBox:~/docker-postgres-backup$ docker-compose up -d --build
```




```
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS      NAMES
709b53ea86bf   docker-postgres-backup-backup   "crond -f"               3 minutes ago   Up 2 minutes              db_backup
346085a84e20   postgres:15                     "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   5432/tcp   my_postgres
allimi@allimi-VirtualBox:~/docker-postgres-backup/backup$
```



4. Explore and discuss the significance of container orchestration in managing large-scale, distributed applications. Highlight key features such orchestration systems provide for automation, scaling, and managing containerized applications' lifecycle. List some widely-used orchestration tools.

Container Orchestration for Large-Scale Apps it aims to automate deployment, scaling, and management of distributed containerized apps, it Ensures reliability, efficiency, and scalability for cloud-native apps.

Key Benefits:

- Automation: Deploys, schedules, and manages containers dynamically.
- Scaling: Auto-scales horizontally/vertically based on demand.
- High Availability: Self-heals failed containers and balances load.

- Zero-Downtime Updates: Supports rolling updates and rollbacks.
- Security: Manages secrets, RBAC, and network policies.

Top Orchestration Tools:

- Kubernetes (Most powerful, industry standard)
- Docker Swarm (Simple, built into Docker)
- Amazon ECS (AWS-integrated, managed)
- Nomad (Lightweight, multi-cloud)