Tugas Sesi 9 Bootcamp DevOps Engineer DigitalSkola

Deskripsi:

Sebuah startup ingin membangun aplikasi berbasis web dengan arsitektur berikut:

1. Frontend: Menggunakan Nginx.

2. Backend: API sederhana berbasis Python.

3. Database: PostgreSQL untuk menyimpan data aplikasi.

4. Networking: Backend dan database harus saling terhubung dengan jaringan.

5. Volume: Database PostgreSQL memerlukan penyimpanan data yang persisten.

Tugas Peserta:

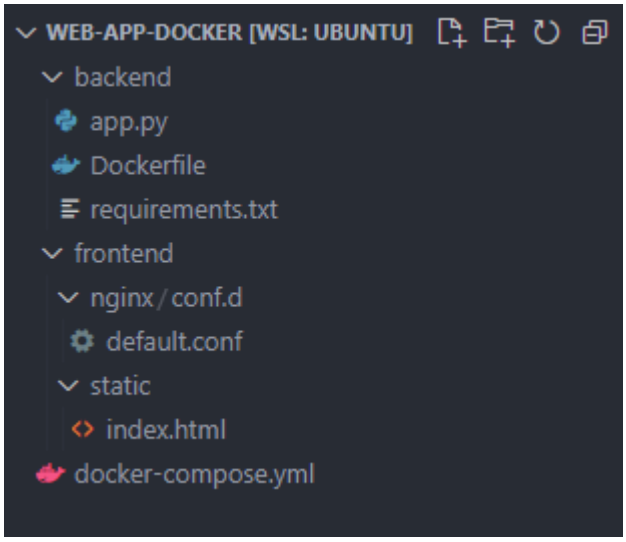
1. Buat Docker Compose file untuk menjalankan semua service.

2. Pastikan Nginx dapat diakses melalui browser.

3. Gunakan Docker Volume untuk menyimpan data PostgreSQL.

4. Pastikan semua container saling terhubung melalui jaringan.

Keseluruhan direktori untuk aplikasi web yang dibuat ini adalah sebagai berikut:



Pertama, buat file **docker-compose.yml** di direktori **web-app-docker**. File ini akan mendefinisikan semua layanan (services) yang dibutuhkan aplikasi web beserta konfigurasi terkait. Isinya adalah sebagai berikut:

```
# version: "3.9"

services:
  frontend:
    container_name: nginx-frontend
```

```yaml
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./frontend/nginx/conf.d:/etc/nginx/conf.d
      - ./frontend/static:/var/www/html
    depends_on:
      - backend
    networks:
      - app-network

  backend:
    container_name: python-backend
    # image: python:3.9-slim-buster
    build: ./backend
    ports:
    - "8080:8080"
    volumes:
      - ./backend:/app
    working_dir: /app
    command: python app.py
    environment:
      POSTGRES_HOST: postgres-db
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: app_db
      POSTGRES_PORT: 5432
    depends_on:
      - db
    networks:
      - app-network

  db:
    container_name: postgres-db
    image: postgres:13-alpine
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: app_db
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - app-network

networks:
  app-network:
```

```
    driver: bridge

volumes:
  db-data:
```

1. # version: "3.9"

- Mencantumkan atribut version di file dokcer-compose.yaml akan memunculkan warning berikut:

  WARN[0000] /home/kaisenberg/Sesi 9/web-app-docker/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

  Sehingga version dapat tidak dicantumkan saja (dihapus) ataupun ditambahkan # agar menjadi comment.

2. services: → Mendefinisikan layanan-layanan (containers) yang akan dijalankan dalam aplikasi ini. Setiap layanan merepresentasikan komponen terpisah dari aplikasi web.

2.1. frontend: → Menjalankan Nginx sebagai server web untuk menyajikan file statis frontend dan berpotensi sebagai reverse proxy untuk backend API.

- **container_name: nginx-frontend** → Memberikan nama spesifik "nginx-frontend" pada container ini, memudahkan identifikasi.
- **image: nginx:latest** → Menggunakan image Docker resmi nginx:latest dari Docker Hub. Ini akan mengambil image Nginx versi terbaru (latest).
- **ports**: → Konfigurasi pemetaan port.
  - **- "80:80"** → Memetakan port 80 pada host (komputer local) ke port 80 di dalam container Nginx. Ini memungkinkan akses aplikasi melalui http://localhost di browser.
- **volumes:** → Konfigurasi volume mounts (berbagi direktori antara host dan container).
  - **- ./frontend/nginx/conf.d:/etc/nginx/conf.d** → Mem-mount direktori konfigurasi Nginx kustom dari host (./frontend/nginx/conf.d) ke direktori konfigurasi Nginx di container (/etc/nginx/conf.d).
  - **- ./frontend/static:/var/www/html** → Mem-mount direktori yang berisi file statis frontend dari host (./frontend/static) ke direktori root dokumen Nginx di container (/var/www/html).
- **depends_on: - backend** → Menentukan bahwa layanan frontend akan dimulai setelah layanan backend berjalan.
- **networks: - app-network** → Menghubungkan container Nginx ke jaringan app-network.

2.2. backend: → Menjalankan backend API aplikasi menggunakan Python.

- **container_name: python-backend** → Memberikan nama spesifik "python-backend" pada container ini.
- **build: ./backend** → Menginstruksikan Docker Compose untuk **membangun image backend dari Dockerfile yang ada di direktori ./backend**. Ini memungkinkan kustomisasi image backend, seperti instalasi library Python (Flask, psycopg2). Image python yang langsung diambil dari docker hub tidak menyertakan library Python tambahan seperti flask dan psycopg2 secara default. Sehingga menggunakan Dockerfile menjadi salah satu solusi.
- **ports:** →Konfigurasi pemetaan port.
    - **- "8080:8080"** → Memetakan port 8080 pada host ke port 8080 di container backend. Ini memungkinkan akses langsung ke backend API melalui http://localhost:8080.
- **volumes:** → Konfigurasi volume mount.
    - **- ./backend:/app** → Mem-mount direktori kode backend dari host (./backend) ke direktori /app di container. Ini memungkinkan sinkronisasi kode antara host dan container.
- **working_dir: /app** → Menetapkan direktori kerja di dalam container menjadi /app. Perintah command dan perintah lainnya akan dijalankan dari direktori ini.
- **command: python app.py** → Perintah yang dijalankan saat container backend dimulai. Ini akan menjalankan aplikasi Python backend (file app.py).
- **environment:** → Konfigurasi environment variables untuk container backend. Variabel-variabel ini digunakan untuk konfigurasi koneksi database.
    - **POSTGRES_HOST: postgres-db** → Host database, menggunakan nama container db (PostgreSQL) dalam jaringan Docker.
    - **POSTGRES_USER: user** →Username database PostgreSQL.
    - **POSTGRES_PASSWORD: password** →Password database PostgreSQL.
    - **POSTGRES_DB: app_db** → Nama database PostgreSQL yang akan digunakan.
    - **POSTGRES_PORT: 5432** → Port database PostgreSQL (default: 5432).
- **depends_on: - db** → Menentukan bahwa layanan backend akan dimulai setelah layanan db (PostgreSQL) berjalan.
- **networks: - app-network** → Menghubungkan container backend ke jaringan app-network.

2.3. db: → Menjalankan database PostgreSQL untuk menyimpan data aplikasi.

- **container_name: postgres-db** → Memberikan nama spesifik "postgres-db" pada container database.
- **image: postgres:13-alpine** → Menggunakan image Docker postgres:13-alpine dari Docker Hub. Ini adalah image PostgreSQL versi 13 berbasis Alpine Linux yang lebih ringan.
- **environment:** → Konfigurasi environment variables untuk database PostgreSQL.
    - **POSTGRES_USER: user** → Mengatur username administrator PostgreSQL menjadi "user".
    - **POSTGRES_PASSWORD: password** → Mengatur password untuk user "user" menjadi "password".
    - **POSTGRES_DB: app_db** → Mengatur nama database awal yang akan dibuat menjadi "app_db".
- **volumes:** → Konfigurasi volume mount untuk persistence data.
    - **- db-data:/var/lib/postgresql/data** → Mem-mount **Docker Volume** bernama db-data ke direktori data PostgreSQL di dalam container (/var/lib/postgresql/data). Ini memastikan data database tetap ada meskipun container dihentikan atau dihapus.
- **ports:** → Konfigurasi pemetaan port untuk database.
    - **- "5432:5432"** →Memetakan port 5432 pada host ke port 5432 di container database. Ini memungkinkan akses database dari host menggunakan client PostgreSQL pada port 5432.
- **networks: - app-network** → Menghubungkan container database ke jaringan app-network.

3. networks: → Mendefinisikan jaringan Docker yang digunakan untuk menghubungkan semua layanan aplikasi.

- **app-network:** → Mendefinisikan jaringan dengan nama "app-network".
    - **driver: bridge** → Menggunakan driver jaringan bridge. Ini adalah driver jaringan default untuk Docker Compose dan menciptakan jaringan internal terisolasi. Container dalam jaringan yang sama dapat berkomunikasi menggunakan nama container sebagai hostname (contoh: postgres-db dapat diakses dari python-backend).

4. volumes: → Mendefinisikan Docker Volumes bernama yang digunakan untuk persistence data.

- **db-data:** → Mendefinisikan Docker Volume bernama "db-data". Volume ini digunakan oleh layanan db untuk menyimpan data PostgreSQL secara persisten. Docker mengelola volume ini, dan datanya akan tetap ada bahkan setelah container database dihapus.

Kedua, buat direktori **frontend** di dalam direktori **web-app-docker.** Isi dari direktori ini adalah configurasi nginx dan file **index.html**. Buat direktori **nginx** danf **nginx/conf.d** di dalam direktori **frontend**. Kemudian, buat file konfigurasi Nginx bernama **default.conf** di dalam direktori **nginx/conf.d**. Buat file **nginx/conf.d/default.conf** dengan konten berikut:

```
server {
    listen 80;
    index index.html;
    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

server { ... }: Mendefinisikan blok server Nginx.

- **listen 80;** → Nginx akan mendengarkan pada port 80.
- **index index.html;** → Menetapkan index.html sebagai file index default.
- **root /var/www/html;** → Menetapkan direktori root untuk file statis menjadi /var/www/html (sesuai dengan volume mount yang didefinisikan di docker-compose.yml).
- **location / { ... }** → Mendefinisikan konfigurasi untuk semua request ke path root (/).
    - **try_files $uri $uri/ /index.html;** →Mencoba mencari file yang diminta ($uri), jika tidak ditemukan, mencoba mencari direktori ($uri/), jika masih tidak ditemukan, mengembalikan index.html. Ini umum digunakan untuk aplikasi frontend single-page application (SPA).

Selanjutnya, buat direktori **static** di dalam direktori **forntend** dan buat file **index.html** di dalamnya. File **index.html** dibuat sederhana saja.

```
<!DOCTYPE html>
<html>
<head>
    <title>Selamat Datang!</title>
</head>
<body>
    <h1>Halo dari Nginx!</h1>
    <p>Aplikasi web berhasil dijalankan dengan Docker Compose.</p>
</body>
</html>
```

Ketiga, aplikasi backend dibuat menggunakan python dibuat di dalam direktori **backend** dengan nama file **app.py.** Konten backend/app.py dibuat dengan sederhana untuk koneksi ke database PostgreSQL:

```
from flask import Flask
from flask import jsonify
import os
import psycopg2

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello from Backend API!"

@app.route('/db_test')
def db_test():
    # Baca environment variable individual
    db_host = os.environ.get('POSTGRES_HOST')
    db_user = os.environ.get('POSTGRES_USER')
    db_password = os.environ.get('POSTGRES_PASSWORD')
    db_name = os.environ.get('POSTGRES_DB')
    db_port = os.environ.get('POSTGRES_PORT') # Port tetap string, psycopg2
akan handle konversi ke integer

    # Validasi environment variable (opsional, tapi bagus untuk debugging)
    if not all([db_host, db_user, db_password, db_name]):
        return jsonify({"error": "Missing required database environment
variables"}), 500

    try:
        # Koneksi ke database menggunakan parameter individual
        conn = psycopg2.connect(
            host=db_host,
            user=db_user,
            password=db_password,
            database=db_name,
            port=db_port
        )
        cur = conn.cursor()
        cur.execute("SELECT 1")
        result = cur.fetchone()
        cur.close()
        conn.close()
        return jsonify({"database_connection": "success", "result": result})
    except Exception as e:
        return jsonify({"database_connection": "failed", "error": str(e)}),
500
```

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Dockerfile dibuat juga di dalam direktori **backend** yang digunakan untuk membuat image aplikasi python dengan library flask dan psycopg2 adalah sebagai berikut:

```
FROM python:3.9-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

File **requirement.txt**:

```
flask
psycopg2-binary
```

Keempat, jalankan docker compose. Buka terminal, pastikan berada di direktori **web-app-docker** yang berisi file **docker-compose.yml**. Jalankan perintah **docker-compose up -d** untuk memulai semua service:

```
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker-compose up -d
```

Opsi **-d** untuk menjalankan container dalam mode detached (background).

Tunggu beberapa saat hingga semua container selesai dijalankan.

```
[+] Running 5/5
 ✓ backend                                Built
 ✓ Network web-app-docker_app-network     Created
 ✓ Container postgres-db                  Started
 ✓ Container python-backend               Started
 ✓ Container nginx-frontend               Started
```

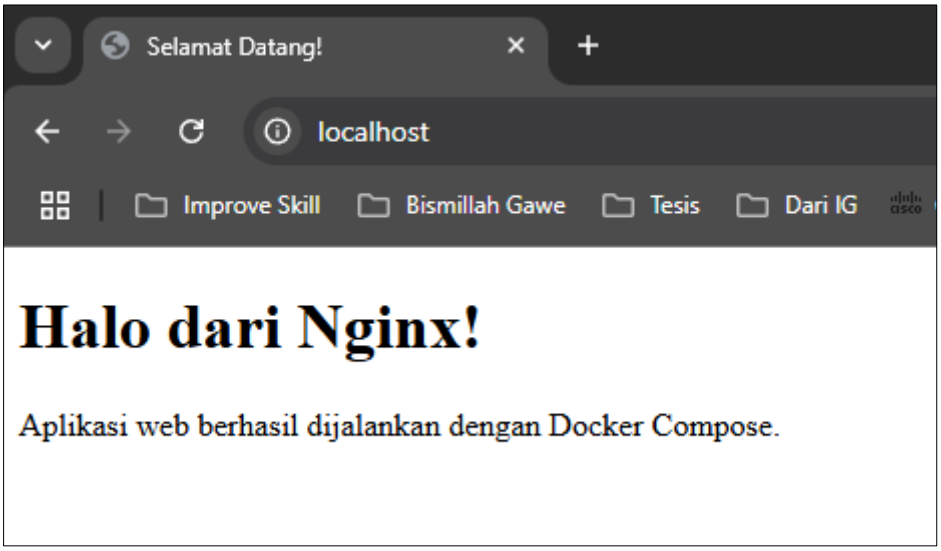
Lalu, dapat melihat log dengan perintah **docker-compose logs**.

Verifikasi aplikasi web yang telah dibuat dengan Docker Compose.

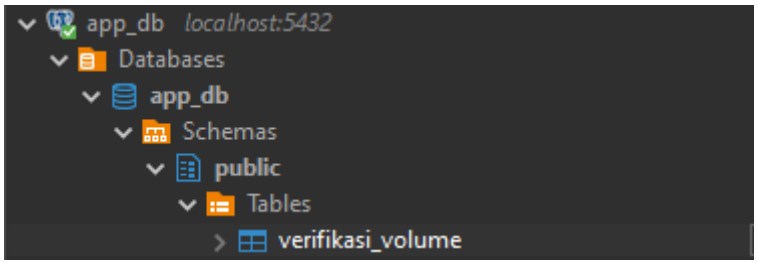
1. **Docker Compose file**: File **docker-compose.yml** telah dibuat dan mendefinisikan semua service (nginx-frontend, pyhton-backend, postgres-db), volume, network, dan konfigurasi lainnya.
   Dengan perintah **docker ps**, dapat dilihat container yang telah dibuat dan berjalan.

```
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                    NAMES
647ca047d076   nginx:latest           "/docker-entrypoint.…"   25 minutes ago   Up 25 minutes   0.0.0.0:80->80/tcp       nginx-frontend
84ea3f76cfca   web-app-docker-backend "python app.py"          25 minutes ago   Up 25 minutes   0.0.0.0:8080->8080/tcp   python-backend
c82d323f638e   postgres:13-alpine     "docker-entrypoint.s…"   25 minutes ago   Up 25 minutes   0.0.0.0:5432->5432/tcp   postgres-db
```

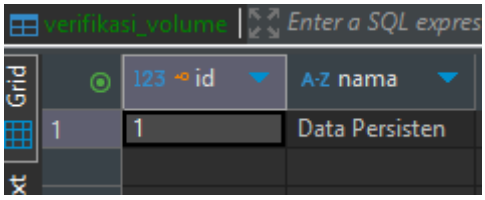
2. **Nginx diakses melalui browser**: Dapat mengakses Nginx melalui http://localhost dan melihat halaman statis.



3. **Docker Volume untuk PostgreSQL**: Volume bernama **db-data** telah didefinisikan dan digunakan oleh service db untuk persistence data PostgreSQL. Data database akan tetap ada meskipun container database di-stop atau dihapus.
   Untuk mem-verifikasinya, coba buat satu tabel bernama **verifikasi_volume.** Dengan tool DBeaver terlihat tabel berikut



   Boleh diinsert isi datanya:



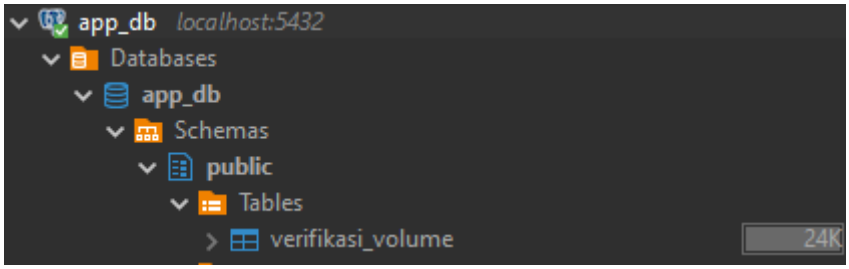
   Hentikan dan hapus container postgres-db dengan perintah **docker-compose stop db** lalu **docker-compose rm db.**

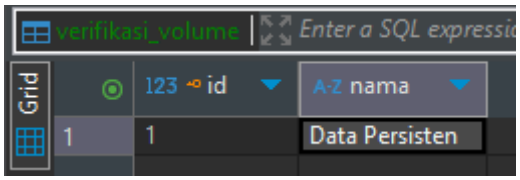
```
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker-compose stop db
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker-compose rm db
? Going to remove postgres-db Yes
[+] Removing 1/1
 ✓ Container postgres-db  Removed
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker container list -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED       STATUS       PORTS                    NAMES
647ca047d076   nginx:latest           "/docker-entrypoint.…"   2 hours ago   Up 2 hours   0.0.0.0:80->80/tcp       nginx-frontend
84ea3f76cfca   web-app-docker-backend "python app.py"          2 hours ago   Up 2 hours   0.0.0.0:8080->8080/tcp   python-backend
```

Jalankan perintah **docker-compose up -d** untuk menghidupkan kembali semua service Docker Compose.

```
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker-compose up -d
[+] Running 3/3
 ✓ Container postgres-db      Started
 ✓ Container python-backend   Running
 ✓ Container nginx-frontend   Running
```

Buka kembali DBeaver dan koneksi ke database app_db dan lakukan koneksi ulang. Tabel **verifikasi_volume** dan isi datanya tetap ada.





4. **Semua container terhubung melalui jaringan**: Semua service (nginx-frontend, pyhton-backend, postgres-db) terhubung ke network app-network, memungkinkan mereka berkomunikasi satu sama lain. Backend dapat terhubung ke database menggunakan hostname db (nama service database) dalam URL koneksi.
   Dengan perintah **docker network ls**, dapat dilihat network yang dibuat dengan docker compose telah ada di dalam list.

```
kaisenberg@MENOMEN:~/Sesi 9/web-app-docker$ docker network ls
NETWORK ID     NAME                          DRIVER    SCOPE
c79a6f94e539   bridge                        bridge    local
b0cb7ffed1de   host                          host      local
bb2d6fb097f9   my_app_net                    bridge    local
e0bf60ea5b32   none                          null      local
b02843fc9c69   web-app-docker_app-network    bridge    local
```

   Lalu dengan perintah **docker network inspect web-app-docker_app-network**, dapat dilihat configurasi network dan telah tercantum ketiga container dari service nginx-frontend, pyhton-backend, postgres-db telah terhubung dalam network ini.

```
[
  {
    "Name": "web-app-docker_app-network",
    "Id": "4881924cb5eea5638f665d08f85870f670c53e729a2f675d39ce16876cab3f79",
    "Created": "2025-03-03T06:01:44.635652859Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "1f7e2786f20f91266445a16ffcb2cda58ae542109a87b67402f59a1bcaa2f8e3": {
        "Name": "python-backend",
        "EndpointID":
"393ee64cafd3b42e1a733b0224c0ac01f21f1bed7cb793931642b7401dd2cd2e",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
      },
      "3c47711794e75702966bc8c9bdfb348731622e85133eb56cdf705fa400b664cb": {
        "Name": "nginx-frontend",
        "EndpointID":
"b0539843643c9fda2bfff97ae41ae7229916d813824c619e86425c5cdb15f051",
        "MacAddress": "02:42:ac:13:00:04",
        "IPv4Address": "172.19.0.4/16",
        "IPv6Address": ""
      },
      "7779dee76bdfc0e43d766ee30b5cfc08f11098fa2404e690185d4f7ac2821318": {
        "Name": "postgres-db",
        "EndpointID":
"7a88db23857374e329d1b7f0552257dae6592ede0efcea8b72e48a323f6858bc",
        "MacAddress": "02:42:ac:13:00:02",
```

```
            "IPv4Address": "172.19.0.2/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {
        "com.docker.compose.config-hash":
"4a6e71b53640a9d8cc19f43ea725257e9572d6adf671f5bf6fbd2eaef35469bb",
        "com.docker.compose.network": "app-network",
        "com.docker.compose.project": "web-app-docker",
        "com.docker.compose.version": "2.32.4"
    }
  }
]
```