Dockerized Microservices Application Report

# 1. Dockerfiles

Each component of the application (frontend and backend) has been containerized using Docker.   
The Dockerfiles define the steps required to build the respective Docker images for the services.

## Backend Dockerfile

FROM python:3.10-slim  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY . .  
CMD ["python", "app.py"]

## Frontend Dockerfile

FROM nginx:alpine  
COPY index.html /usr/share/nginx/html/index.html

# 2. Docker Compose Configuration

The docker-compose.yml file defines and configures multiple services including frontend, backend,  
a PostgreSQL database for application data, and Apache Airflow services (webserver, scheduler, and metadata database).  
All services are connected using a bridge network and necessary environment variables and volumes are defined.

# 3. Instructions for Deployment and Usage

To deploy and test the application:  
1. Ensure Docker and Docker Compose are installed.  
2. Place all necessary files and folders according to the project structure.  
3. Open a terminal and run the following command to build and start the containers:  
 docker-compose up --build  
4. Access the services using the following URLs:  
 - Frontend: http://localhost:8080  
 - Backend API: http://localhost:5001/api/data  
 - Airflow UI: http://localhost:8081  
5. Use the Airflow UI to trigger DAGs manually and monitor task execution.

# 4. Optional: Scaling and Load Balancing Strategies

To scale services:  
- Use Docker Compose to scale frontend and backend containers:  
 docker-compose up --scale backend=3 --scale frontend=2  
- Implement a reverse proxy like NGINX or Traefik for load balancing.  
- For production environments, consider using Kubernetes with horizontal pod autoscaling.

# 5. Challenges Faced

Several challenges were encountered during the implementation:  
- Port conflicts due to multiple PostgreSQL instances running simultaneously.  
- Failure of Airflow services to start until the metadata database was initialized with 'airflow db init'.  
- Inconsistent visibility of folders created via terminal in the IDE.  
- Data type mismatch between pandas and psycopg2 (e.g., numpy.int64).  
- Errors due to missing environment variables or misconfigured docker-compose.yml structure.  
- Manual creation of test users and Airflow connections were initially overlooked.