**1. Justification for Choosing Google Cloud:**

I chose **Google Cloud** to deploy my e-commerce application due to several key advantages:

* **Cloud SQL** offers a managed MySQL service with automated backups, updates, and high availability, which is essential for ensuring data integrity and uptime for the e-commerce store.
* **Cloud Run** provides a serverless platform that automatically scales the backend (Java Spring Boot) and frontend (Angular) services based on traffic demand, ensuring efficient resource usage and cost optimization.
* Google Cloud's **VPC Serverless Connector** allows secure, private connections between services like Cloud Run and Cloud SQL, protecting the database from exposure to the public internet while enabling high-performance, low-latency access.
* Google Cloud's strong integration with popular development tools (like Docker and Kubernetes) ensures a streamlined CI/CD process.
* Competitive pricing and clear billing are also attractive for scaling projects.

**2. Implementation of Container Images:**

For the deployment, I used Docker to containerize both the Angular frontend and the Java Spring Boot backend services. Here’s a summary of the steps:

* **Backend (Java Spring Boot)**:
  + Created a Dockerfile that packages the Spring Boot application as a container image.
  + Used Maven to build the Java application and copied the output .jar file into the Docker image.
  + The image is hosted on Google Container Registry (GCR) and deployed via **Google Cloud Run**, which automatically manages scaling and load balancing.
* **Frontend (Angular)**:
  + Built the Angular application into static files, packaged in a Docker image with a lightweight Nginx web server to serve the static files.
  + Similarly, the image is stored in Google Container Registry and deployed to **Cloud Run**.
* **Database (Cloud SQL)**:
  + Configured **Cloud SQL** for MySQL with a private IP, ensuring security by connecting the database to the backend service using a **VPC Serverless Connector**.

Finally, I automated the deployment using GitLab CI/CD pipelines that build, tag, and push the Docker images to Google Container Registry and trigger deployment on **Google Cloud Run**.

**1. Backend (Java Spring Boot) - Dockerfile**

Dockerfile

# Use the official Maven image to compile the application

FROM maven:3.8.5-openjdk-17 AS build

WORKDIR /app

# Copy the project files

COPY . /app

# Compile the project

RUN mvn clean install

# Use a lightweight OpenJDK image to run the application

FROM openjdk:17-jdk-slim

WORKDIR /app

# Copy the compiled JAR file

COPY --from=build /app/target/\*.jar app.jar

# Expose port 8080

EXPOSE 8080

# Run the Spring Boot application

ENTRYPOINT ["java", "-jar", "app.jar"]

**2. Frontend (Angular) - Dockerfile**

Dockerfile

# Stage 1: Build Angular app

FROM node:18-alpine AS build

WORKDIR /app

# Copy package.json and install dependencies

COPY package\*.json ./

RUN npm install

# Copy the source code and build the app

COPY . .

RUN npm run build --prod

# Stage 2: Serve with Nginx

FROM nginx:alpine

COPY --from=build /app/dist/your-angular-app /usr/share/nginx/html

# Expose port 80

EXPOSE 80

# Start Nginx

CMD ["nginx", "-g", "daemon off;"]

**3. GitLab CI/CD Pipeline Configuration (.gitlab-ci.yml)**

yaml

stages:

- build

- deploy

# Backend: Java Spring Boot

build-backend:

stage: build

image: docker:24.0.5

services:

- docker:24.0.5-dind

script:

- docker build -t gcr.io/d424-436314/spring-app:latest -f Dockerfile .

- docker push gcr.io/d424-436314/spring-app:latest

only:

- main

# Frontend: Angular

build-frontend:

stage: build

image: docker:24.0.5

services:

- docker:24.0.5-dind

script:

- docker build -t gcr.io/d424-436314/angular-app:latest -f Dockerfile .

- docker push gcr.io/d424-436314/angular-app:latest

only:

- main

# Deploy backend to Cloud Run

deploy-backend:

stage: deploy

image: google/cloud-sdk:latest

script:

- gcloud auth configure-docker

- gcloud run deploy spring-app --image gcr.io/d424-436314/ecommerce-service:v1 --platform managed --region us-central1 --allow-unauthenticated --vpc-connector sql-connector

only:

- main

# Deploy frontend to Cloud Run

deploy-frontend:

stage: deploy

image: google/cloud-sdk:latest

script:

- gcloud auth configure-docker

- gcloud run deploy angular-app --image gcr.io/d424-436314/angular-app:latest --platform managed --region us-central1 --allow-unauthenticated

only:

- main

**4. VPC Serverless Connector Configuration**

To ensure secure communication between Cloud Run and Cloud SQL, you'll need to configure the VPC connector. This step is done outside the script, but here’s how to set it up:

1. Create a VPC connector:

bash

gcloud compute networks vpc-access connectors create sql-connector \

--region=us-central1 --range=10.8.0.0/28

1. To ensure Cloud Run service uses this connector during deployment: In the Cloud Run deployment commands, I added: --vpc-connector sql-connector to allow the backend to connect to the Cloud SQL instance.

**5. Database Connection (Application Properties for Java)**

Configure the Spring Boot application to connect to your Cloud SQL instance using the private IP:

properties

spring.datasource.url=jdbc:mysql://<PRIVATE\_IP\_OF\_CLOUD\_SQL>:3306/your-database

spring.datasource.username=your-username

spring.datasource.password=your-password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect