Ezgi Subaşı 20050111015
Instructor: Çağdaş Evren Gerede

CLOUD COMPUTING PROJECT REPORT

1. Project Overview

My project delivers a cloud-native sentiment analysis service using a microservice-based architecture. The main goals were to create a scalable, portable, and stateless API that accepts user input text, analyzes its sentiment using a Hugging Face transformer model, and optionally logs results in a PostgreSQL database. All components are containerized using Docker, following best practices in cloud-native and twelve-factor app design.

2. Key Components and Used Technologies

2.1.Flask API:

Lightweight Python framework used to expose a RESTful endpoint (`/analyze`).

Accepts JSON input and returns sentiment classification results.

Stateless by design—each API call is independent.

2.2.Hugging Face Transformers:

Pre-trained model (e.g., distilbert-base-uncased-finetuned-sst-2-english) used for sentiment analysis.

Integrated via the pipeline("sentiment-analysis") interface for simplicity and performance.

Model is loaded at container startup to minimize inference time per request.

2.3.PostgreSQL Database:

Stores user queries and sentiment predictions.

Treated as a "backing service" per twelve-factor methodology: replaceable and decoupled from application logic.

Accessed via SQL from the Flask container.

2.4.Docker and Docker Compose

Each component (API, DB) runs in its own container.

Docker Compose orchestrates multi-container deployment.

Environment variables are used for configuration, supporting deployment to any cloud platform.

3. Cloud-Native Design Features

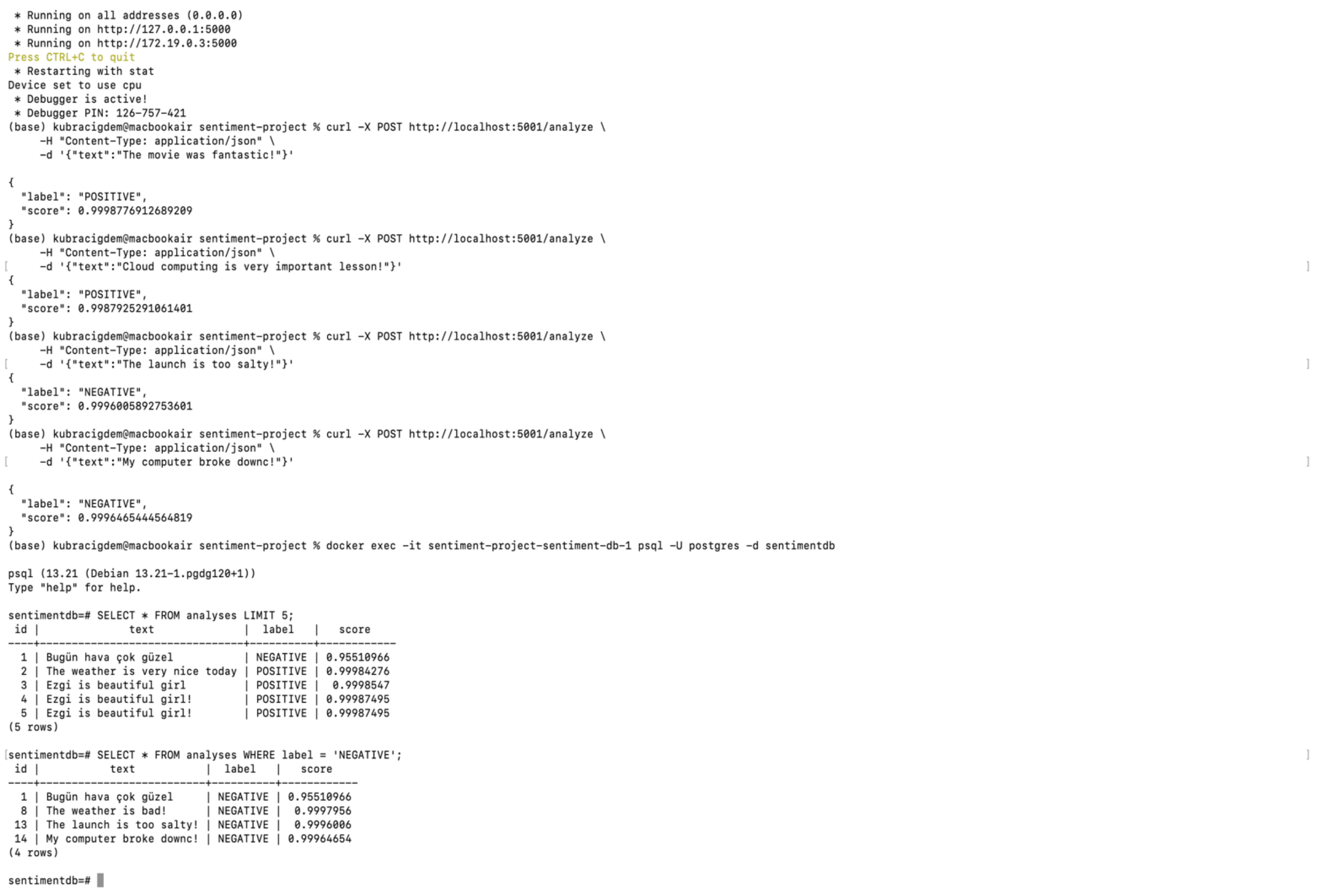
3.1 Stateless Architecture: The Flask service doesn't store session data; all state is handled externally (PostgreSQL). This enables easy horizontal scaling and resilience.

3.2. Containerization & Portability: Docker ensures consistent environments across development, testing, and production. The architecture supports deployment to AWS, GCP, Azure, or on-prem infrastructure.

3.3. Twelve-Factor Compliance: The application follows principles like externalized configuration, dependency isolation, disposable processes, and logging to stdout. The database is an attached resource that can be dynamically swapped.

3.4. CI/CD Ready: Docker images can be versioned, tested, and deployed via automated pipelines. Also, I uploaded project to Github.

4. Project Workflow



```
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.19.0.3:5000
Press CTRL+C to quit
 * Restarting with stat
Device set to use cpu
 * Debugger is active!
 * Debugger PIN: 126-757-421
(base) kubracigdem@macbookair sentiment-project % curl -X POST http://localhost:5001/analyze \
     -H "Content-Type: application/json" \
     -d '{"text":"The movie was fantastic!"}'

{
  "label": "POSITIVE",
  "score": 0.9998776912689209
}
(base) kubracigdem@macbookair sentiment-project % curl -X POST http://localhost:5001/analyze \
     -H "Content-Type: application/json" \
     -d '{"text":"Cloud computing is very important lesson!"}'
{
  "label": "POSITIVE",
  "score": 0.9987925291061401
}
(base) kubracigdem@macbookair sentiment-project % curl -X POST http://localhost:5001/analyze \
     -H "Content-Type: application/json" \
     -d '{"text":"The launch is too salty!"}'
{
  "label": "NEGATIVE",
  "score": 0.9996005892753601
}
(base) kubracigdem@macbookair sentiment-project % curl -X POST http://localhost:5001/analyze \
     -H "Content-Type: application/json" \
     -d '{"text":"My computer broke downc!"}'

{
  "label": "NEGATIVE",
  "score": 0.9996465444564819
}
(base) kubracigdem@macbookair sentiment-project % docker exec -it sentiment-project-sentiment-db-1 psql -U postgres -d sentimentdb

psql (13.21 (Debian 13.21-1.pgdg120+1))
Type "help" for help.

sentimentdb=# SELECT * FROM analyses LIMIT 5;
 id |              text               |  label   |   score
----+---------------------------------+----------+------------
  1 | Bugün hava çok güzel            | NEGATIVE | 0.95510966
  2 | The weather is very nice today  | POSITIVE | 0.99984276
  3 | Ezgi is beautiful girl          | POSITIVE |  0.9998547
  4 | Ezgi is beautiful girl!         | POSITIVE | 0.99987495
  5 | Ezgi is beautiful girl!         | POSITIVE | 0.99987495
(5 rows)

sentimentdb=# SELECT * FROM analyses WHERE label = 'NEGATIVE';
 id |           text           |  label   |   score
----+--------------------------+----------+------------
  1 | Bugün hava çok güzel     | NEGATIVE | 0.95510966
  8 | The weather is bad!      | NEGATIVE |  0.9997956
 13 | The launch is too salty! | NEGATIVE |  0.9996006
 14 | My computer broke downc! | NEGATIVE | 0.99964654
(4 rows)

sentimentdb=# 
```

- Client sends a POST request to /analyze with text input.
- Flask parses the input and sends it to the sentiment model.
- The model returns a label (POSITIVE/NEGATIVE) and confidence score.
- Flask returns this result as JSON and optionally logs it to PostgreSQL.

Input format:

```
curl -X POST http://localhost:5001/analyze -H "Content-Type: application/json" -d
'{"text":"The movie was fantastic!"}'
```

User entire input which she want to analyze.

My Project Video: https://youtu.be/ZtfdNo6C5Nk