Report#5

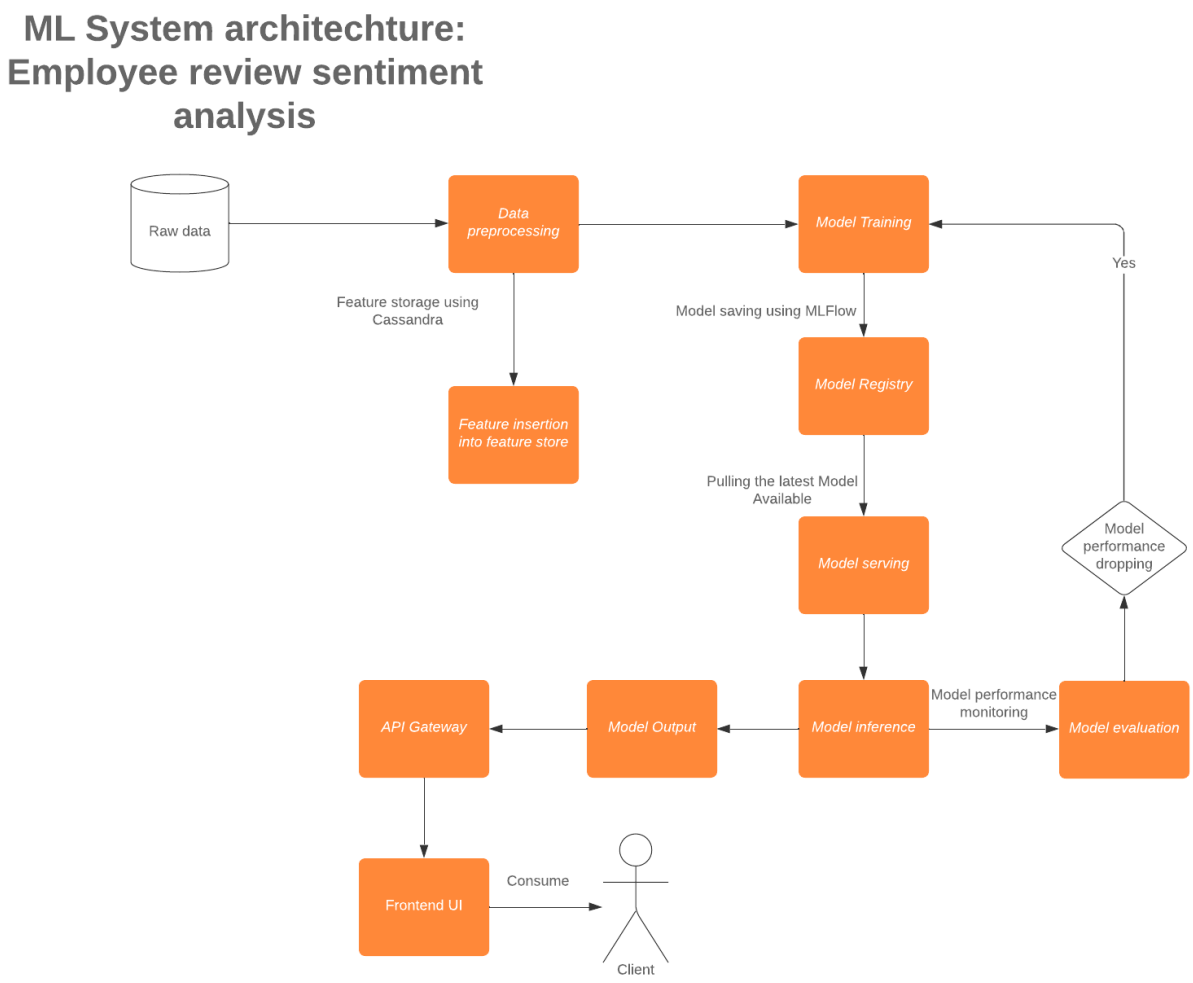
Model Deployment:

MILESTONE 5: Model Deployment (1/2)
Goal: API development, packaging, deployment and serving

Task	Tool suggestion
ML system architecture	The documentation should include a drawing using MS Visio or other
Packaging and containerization	Docker and Docker Compose
ML service deployment	Huggingface, ZenML-Seldon
Model serving	ZenML, MLFlow, TFX serving, FastAPI
Fron-end client	Streamlit, Gradio, FastAPI, Flask and Pytest, React

ML System Architecture:

This is the ML system architecture that was adopted during the development and the design of this ML project.

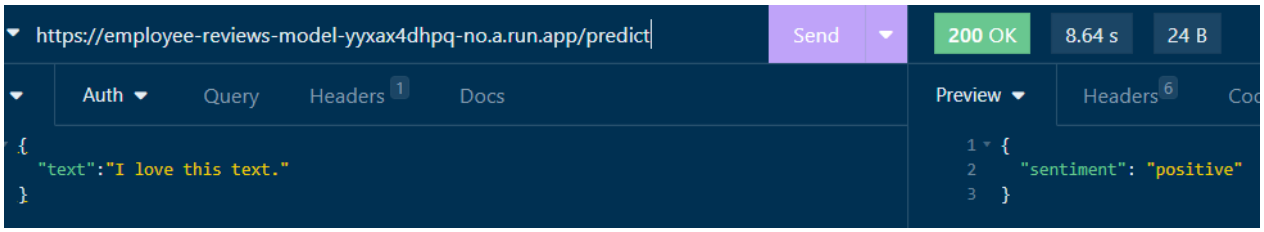


Model Serving using Fast API:

This part of the code serves as the point of interaction of the clients (Through a user interface) with the model inference, it provides a REST API that accepts and responds with json.

```python
main.py > predict_sentiment
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel

from pipeline import model_inference_step

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class TextRequest(BaseModel):
    text: str

@app.post("/predict")
def predict_sentiment(text_request: TextRequest):
    text = text_request.text
    predictions = model_inference_step([text]) # returns a list of predictions
    return {"sentiment": predictions[0]}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

The execution:



Packaging and containerization:

To package and containerize our application we need to create a docker file that contains a base container that the app would work on plus a requirements file that contains all the dependencies that are needed for the app to run.

The Dockerfile:

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.9

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY ./requirements.txt /app/requirements.txt

RUN pip install --no-cache-dir --upgrade -r /app/requirements.txt

COPY . /app

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

The requirements:

This file contains all the dependencies needed for the app to run.

```
requirements.txt
build==1.2.1
cassandra-driver==3.29.1
docker==6.1.3
fonttools==4.50.0
fqdn==1.5.1
frozenlist==1.4.1
fs==2.4.16
fsspec==2024.3.1
```

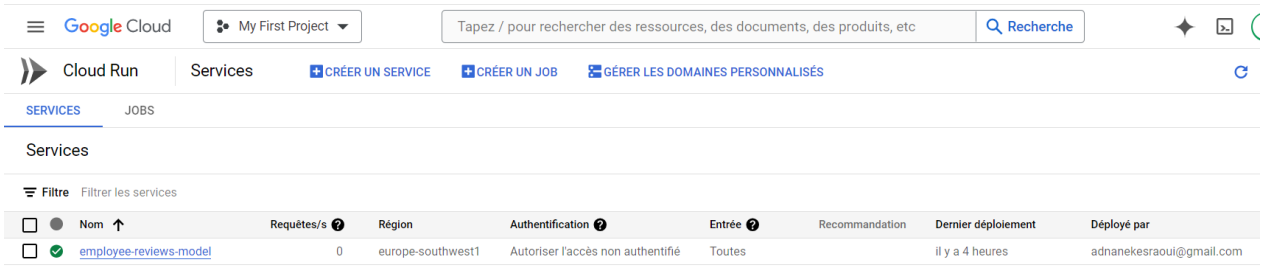
Docker build command:

This is the command that saves a docker image of the app:

docker build --no-cache -t employee-reviews .

Model Deployment:

Regarding the deployment of the app, google's Cloud Run was the chosen technology since it is an Iaas that provides free serverless hosting. It hosts our containarized Model API.



Front-end Client:

This is the code for the front-end client:

sentiment-app > src > JS SentimentPredictor.js > ...

```javascript
import axios from 'axios';
import React, { useState } from 'react';

function SentimentPredictor() {
  const [text, setText] = useState('');
  const [sentiment, setSentiment] = useState(null);

  const handleInputChange = (event) => {
    setText(event.target.value);
  };

  const handleSubmit = async (event) => {
    event.preventDefault();
    try {
      const response = await axios.post(' https://employee-reviews-model-yyxax4dhpq-no.a.run.app/predict', { text });
      setSentiment(response.data.sentiment);
    } catch (error) {
      console.error('Error fetching sentiment:', error);
      setSentiment('Error fetching sentiment');
    }
  };

  return (
    <div>
      <h1>Sentiment Predictor</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={text}
          onChange={handleInputChange}
          placeholder="Enter text here..."
        />
        <button type="submit">Predict Sentiment</button>
      </form>
      {sentiment !== null && <h2>Sentiment: {sentiment}</h2>}
    </div>
  );
}
```

The interface: