

Report#5

Model Deployment:

MILESTONE 5: Model Deployment (1/2)

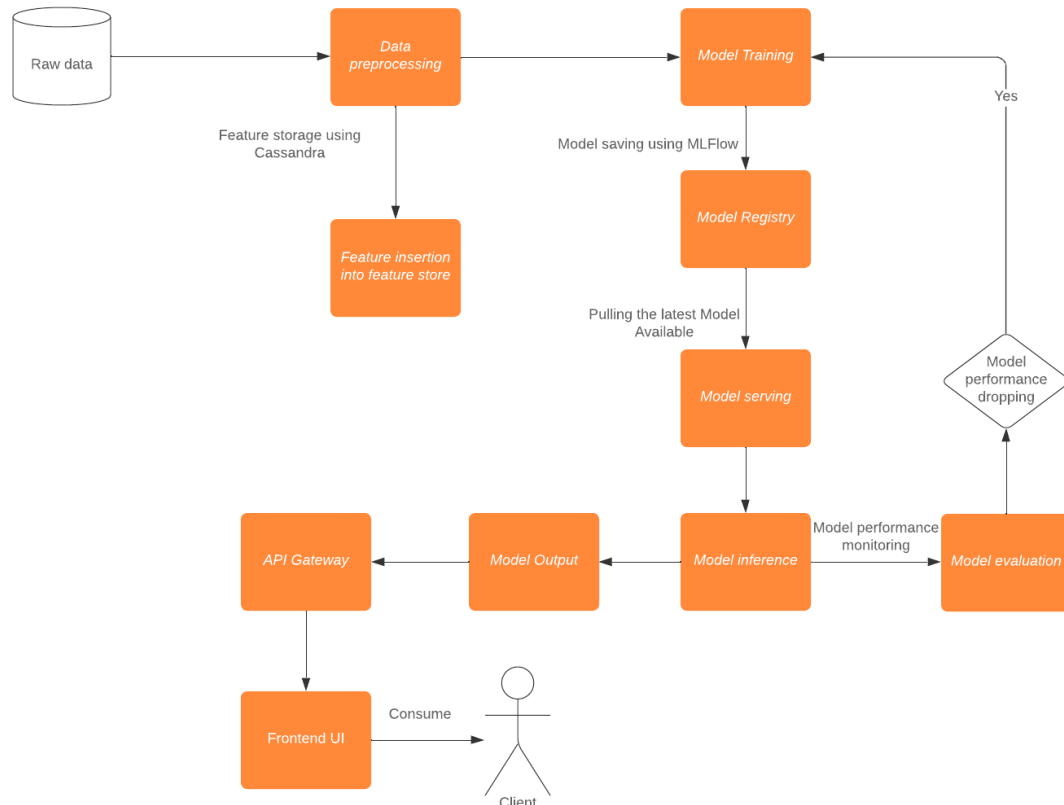
Goal: API development, packaging, deployment and serving

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.

ML System architecture: Employee review sentiment analysis



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.

```
main.py > predict_sentiment
1  from fastapi import FastAPI, HTTPException
2  from fastapi.middleware.cors import CORSMiddleware
3  from pydantic import BaseModel
4
5  from pipeline import model_inference_step
6
7  app = FastAPI()
8
9  app.add_middleware(
10     CORSMiddleware,
11     allow_origins=["*"],
12     allow_credentials=True,
13     allow_methods=["*"],
14     allow_headers=["*"],
15 )
16
17 class TextRequest(BaseModel):
18     text: str
19
20 @app.post("/predict")
21 def predict_sentiment(text_request: TextRequest):
22     text = text_request.text
23     predictions = model_inference_step([text]) # returns a list of predictions
24     return {"sentiment": predictions[0]}
25
26 if __name__ == "__main__":
27     import uvicorn
28     uvicorn.run(app, host="0.0.0.0", port=8000)
```

The execution:

The screenshot shows a REST client interface with the following details:

- URL:** `https://employee-reviews-model-yyxax4dhpq-no.a.run.app/predict`
- Method:** `POST` (indicated by the 'Send' button)
- Status:** `200 OK`
- Time:** `8.64 s`
- Size:** `24 B`
- Request Body (JSON):**

```
{  "text": "I love this text."}
```
- Response Body (JSON):**

```
{  "sentiment": "positive"}
```

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:

```
1 FROM tiangolo/uvicorn-gunicorn-fastapi:python3.9
2
3 ENV PYTHONDONTWRITEBYTECODE 1
4 ENV PYTHONUNBUFFERED 1
5
6 WORKDIR /app
7
8 COPY ./requirements.txt /app/requirements.txt
9
10 RUN pip install --no-cache-dir --upgrade -r /app/requirements.txt
11
12 COPY . /app
13
14 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
1 build==1.2.1
2 cassandra-driver==3.29.1
3 docker==6.1.3
4 fonttools==4.50.0
5 fqdn==1.5.1
6 frozenlist==1.4.1
7 fs==2.4.16
8 fsspec==2024.3.1
```

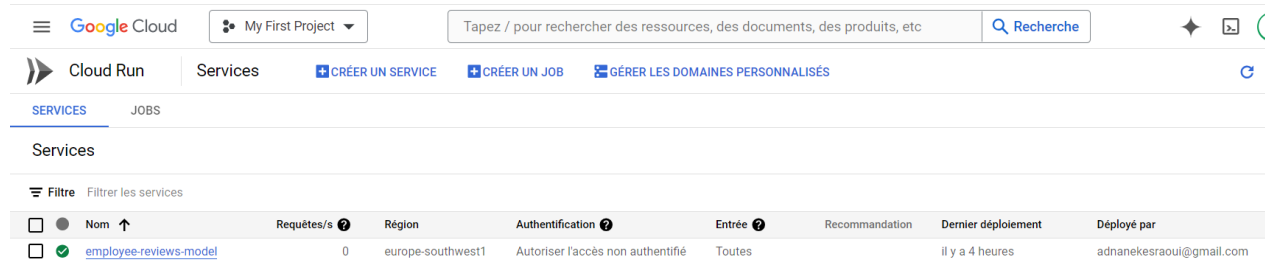
Docker build command:

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 containerized Model API.



The screenshot shows the Google Cloud console interface. At the top, there's a search bar and navigation links for 'Cloud Run', 'Services', 'CRÉER UN SERVICE', 'CRÉER UN JOB', and 'GÉRER LES DOMAINES PERSONNALISÉS'. Below this, the 'Services' section is active, displaying a table of deployed services. The table has columns for 'Nom', 'Requêtes/s', 'Région', 'Authentification', 'Entrée', 'Recommandation', 'Dernier déploiement', and 'Déployé par'. One service, 'employee-reviews-model', is listed with 0 requests/s, in the 'europe-southwest1' region, with 'Autoriser l'accès non authentifié' authentication, and a deployment time of 'il y a 4 heures' by 'adnanekesraoui@gmail.com'.

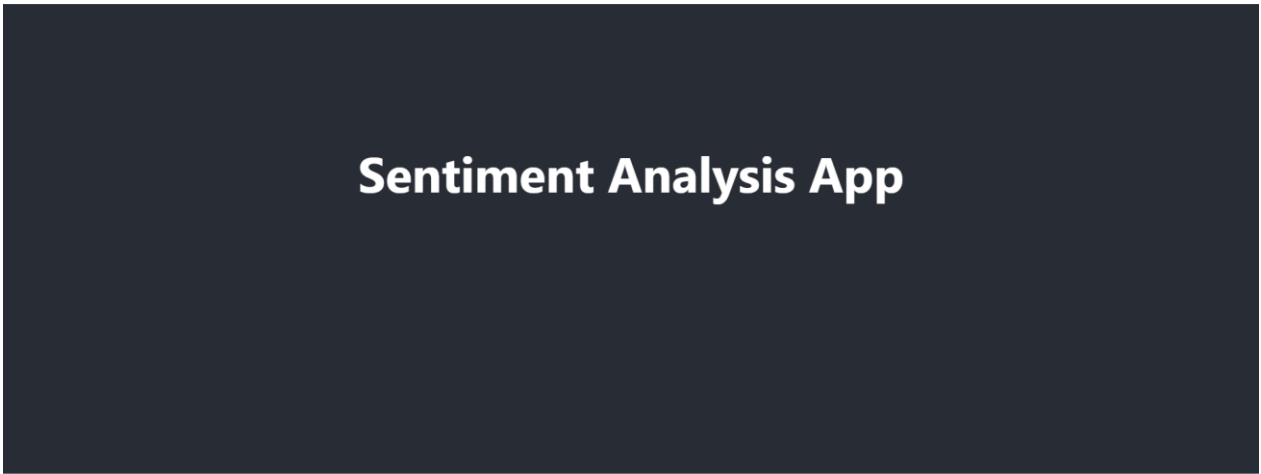
Nom	Requêtes/s	Région	Authentification	Entrée	Recommandation	Dernier déploiement	Déployé par
employee-reviews-model	0	europe-southwest1	Autoriser l'accès non authentifié	Toutes		il y a 4 heures	adnanekesraoui@gmail.com

Front-end Client:

This is the code for the front-end client:

```
sentiment-app > src > JS SentimentPredictor.js > ...
1 import axios from 'axios';
2 import React, { useState } from 'react';
3
4 function SentimentPredictor() {
5   const [text, setText] = useState('');
6   const [sentiment, setSentiment] = useState(null);
7
8   const handleInputChange = (event) => {
9     setText(event.target.value);
10  };
11
12  const handleSubmit = async (event) => {
13    event.preventDefault();
14    try {
15      const response = await axios.post('https://employee-reviews-model-yyxax4dhpq-no.a.run.app/predict', { text });
16      setSentiment(response.data.sentiment);
17    } catch (error) {
18      console.error('Error fetching sentiment:', error);
19      setSentiment('Error fetching sentiment');
20    }
21  };
22
23  return (
24    <div>
25      <h1>Sentiment Predictor</h1>
26      <form onSubmit={handleSubmit}>
27        <input
28          type="text"
29          value={text}
30          onChange={handleInputChange}
31          placeholder="Enter text here..."
32        />
33        <button type="submit">Predict Sentiment</button>
34      </form>
35      {sentiment !== null && <h2>Sentiment: {sentiment}</h2>}
36    </div>
37  );
38 }
```

The interface:



Sentiment Predictor

Enter text here...

Predict Sentiment

Activate Windows
Go to Settings to activate Windows.