**Project Highlights**

* ETL pipeline using airflow (usually use this step in data engineering)
* Feature store implementation
* ML monitoring

**Workflow**

Database setup (In GCP) 🡪 Project setup 🡪 ETL pipeline (Use Airflow and store data in Postgres SQL)🡪 Data Ingestion (Get data from Postgres SQL)🡪 Feature Store (Use Redis) 🡪 Data processing with feature storing 🡪 Model Training 🡪 Training Pipeline 🡪 Data and code versioning 🡪 User app building 🡪 Data Drift modelling (use alibi detect and evidently AI) 🡪 ML Monitoring (Prometheus and Grafana)

**Project Setup**

* Navigate to GCP a bucket under the project (Make sure to unclick the “Enforce public access prevention on this bucket” option. Otherwise, we cannot access this bucket folder)
* Then upload the csv file to that newly create bucket.
* Navigate to VS code and start build the virtual environment
  + Before making any codes, we must create a virtual environment first
  + For that, we must create a folder in our local machine
  + Start the terminal of this folder and run below codes
    - python -m venv venv
    - source venv/bin/activate
    - Make below sub folders and python script
      * templates
      * src (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * config (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * pipeline (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * artifacts
      * static
      * setup.py (Project management code will be here). Add below code to this setup.py file

A computer screen shot of a program code

AI-generated content may be incorrect.

* Run this code now: pip install -e .
* Then come to the “src” folder. Under that folder, create a python file called “logger.py” and “custom\_exception.py” files

**Data Ingestion**

**Below it mentioned the steps that we must complete in Google cloud environment before starting the data ingestion part**

* First, we must create a service account. Before making the service account we must install google cloud CLI to our local machine. [Link](https://cloud.google.com/sdk/docs/install)
* Then navigate to GCP 🡪 IAM Admin 🡪 Service Accounts 🡪 create new service account 🡪 give a name to the service account 🡪 under role, select “storage admin” 🡪 under role, select “storage object viewer” 🡪 under role, select “owner” 🡪 Done
* Go to buckets 🡪 select the bucket name 🡪 click three dots 🡪 Edit access 🡪 Add principal 🡪 Under new principal tab, add the service account 🡪 again give the same roles (storage admin and storage object viewer) and click save
* Again, navigate to the service account 🡪 Now we must create a key 🡪 Click three dots 🡪 manage keys 🡪 Add key 🡪 create new key 🡪 JSON 🡪 Create

**ETL Part (Extract data from GCP using Airflow and load that data into Postgres SQL)**

* This is more related to data engineering role. What we are doing here is, create an ETL tool using Airflow. That means we are use this ETL tool to upload our data to Postgres SQL server. Then we retrieve those data to our VS code environment to do the model training.
* First, we must install astro airflow CLI
* After installing, run below code in the project root directory to initialize the astro

>> astro dev init

* Now, paste the google cloud Json file that downloaded previously into the “include” folder. Rename the Json file name as “gcp-key-json,json”
* Navigate to .astro/config.yaml file. Update that yaml file
* Navigate to “Dockerfile” and add below two lines to it

FROM quay.io/astronomer/astro-runtime:7.3.0

USER root

RUN apt-get update && \

apt-get install -y libpq-dev && \

rm -rf /var/lib/apt/lists/\*

USER astro

RUN pip install --no-cache-dir apache-airflow-providers-google psycopg2-binary

* Then in terminal, run (it will create a container for astro)

>> astro dev start

* Default password and username for the Apache airflow dashboard is “admin”. Add username and password
* Now go to admin tab 🡪 connection. First, we must make two endpoints. The first endpoint is for GCP bucket because we are getting data from GCP bucket. Also, we need to create another endpoint for PostgreSQL because we retrieve data from that
* Click + sign 🡪 Connection ID is “google\_cloud\_default” 🡪 Connection type is “Google Cloud” 🡪 Add the key file path as /usr/local/airflow/include/gcp-key.json 🡪 Add scope as <https://www.googleapis.com/auth/cloud-platform> --> click save
* Click + sign 🡪 Connection ID is “postgres\_default” 🡪 Connection type is “Postgres” 🡪 Host should be the container name (we can get this docker container name by clicking the + sign of the container in docker desktop. It has multiple containers running. We can see them by clicking dropdown icon. We must click the container called “postgres-1”. And copy its name. 🡪 Database, login and password should be “postgres” (by default it is “postgres” 🡪 Port should be 5432 (check this in docker desktop under the Postgres container)🡪 click save
* Navigate to VS code 🡪 Now we must create a new Dag file 🡪 head to dags folder 🡪 under that, make a new file called “extract\_data\_from\_gcp.py” 🡪 Update it
* Navigate to airflow dashboard now 🡪 Go to DAGs 🡪 Now we can see out newly created Dag ID 🡪 Select the run icon 🡪 then it will trigger the event
* Now we can see the SQL table (the only issue is its running inside the container which we cannot see using our local machine. So, we must download tool called “Dbeaver”. Download Dbeaver tool to our machine
* Navigate to Dbeaver now 🡪 Go to Database tab 🡪 Click new database connection 🡪 Select the database, which in our case is Postgres 🡪 click next 🡪 give the password of the database (in our case its “postgres”) 🡪 Additionally check the username and database name as well 🡪 Test the connection 🡪 Finish 🡪 Now we can see the tables

**Load data from Postgres SQL to the Project**

* Navigate to VS code. Under the config folder, create a file called “database\_config.py”. Add all the credentials of the postgres SQL into that.
* Under the src folder, create a file called “data\_ingestion.py”. Update it
* Go to the terminal and run

>> python src/data\_ingestion.py

**Building the feature store using REDIS**

* We cannot use REDIS directly in our local machine. For that we must make a Linux environment. Therefore, the best solution is run REDIS as a docker container.
* Run below two lines in the terminal,

>> docker pull redis

>> docker run -d --name redis-container -p 6379:6379 redis

* Go to src folder and create a new file called “feature\_store.py”. Update the code

**Data processing and model training**

* Go to src folder and create a new file called “data\_preprocessing.py”. Update the code
* Go to the terminal and run

>> python src/data\_processing.py

Or

>> python -m src.data\_processing

* Go to src folder and create a new file called “model\_training.py”. Update the code
* Go to the terminal and run

>> python src/model\_training.py

**Training pipeline and data/code versioning**

* Go to pipeline folder and create a new file called “training\_pipeline.py”. Update the code
* Go to the terminal and run

>> python pipeline/ training\_pipeline.py

* Navigate to GitHub and make a repo. Push all the files into that folder (GitHub will use for the code and data versioning)

>> git init

>> git branch -M main

>> git remote add origin [https:<<repo\_link>>.git](https://github.com/JThilinaDK123/mlops-project-02.git)

>> git add .

>> git commit -m "first commit"

>> git push origin main

**ML Monitoring (Data Drifting)**

* In this project, it will use alibi-detect python library to do the data drifting. The test will use in this project is Kolmogorov Smirnov.
* To check the data drift, we need two types of data
  + Reference data (Data that used in model training)
  + Current data (Data input by user)
* The only place, that we must update is “application.py”

**ML Monitoring (Grafana / Prometheus)**

* Prometheus use to extract metrics data. Grafana will as the dashboard tool.
* We must create another docker container for this task
* First, we must create a file called “docker-compose.yml” in our root file. Update the code.
* Then, we must create a file called “prometheus.yml” in our root file. Update the code
* Now run below code. (Grafana and Prometheus run in the same container)

>> docker compose up -d (this code use to run the container)

* We can access the Grafana:
  + - localhost:3000
* We can access the Prometheus:
  + - localhost:9090
* After updating the “application.py”, then run

>> python application.py

>> docker restart prometheus

* Finally, we can access each tool like below
  + - localhost:5000 🡪 Flask application
    - localhost:5000/metrics 🡪 Flask application metrics
    - localhost:3000 🡪 Grafana
    - localhost:9090 🡪 Prometheus
* The only issue that we are having is all these metrics show in text format. Therefore, we must do some changes in the Grafana dashboard
* Navigate to Grafana dashboard using port link 🡪 Under Connections 🡪 Data Sources 🡪 Add data Sources 🡪 Select Prometheus 🡪 Set the Prometheus URL (here we cannot add “<http://localhost/9090/>” because Prometheus run inside a docker container. Therefore, we must replace “localhost” by Prometheus container name which is “prometheus”. Therefore, the updated link should be “http://prometheus:9090 “) 🡪 Save and test
* Navigate to the Grafana dashboards 🡪 create dashboard 🡪 add visualizations 🡪 select the data source as prometheus 🡪 Select the builder option 🡪 select the metric 🡪 it will display all the default and custom metrics 🡪 select the metric and hit “run query” 🡪 Make sure to save the dashboard as well

**Final Steps:**

* Make sure the Application is running
* After making all the changes, run “docker restart prometheus”
* Check with localhost:9090, whether the prometheus is up and running