**Problem - Business Objectives**

We receive the raw data every day at 23:30 CET. We need to process the data and make it available in Oracle DB. Furthermore, need to update a few other summary tables which will be eventually consumed by visualization apps. Visualization application will be refreshed everyday at 03:00 AM CET.

**Task to Accomplish:**

1. Create pipeline for the above business case to support end-end.

2. Visualization tools need the below KPIs to show to business stakeholders.

- How many users are there?

- every user has made how many requests.

- Display total number of successful request.

**Given Data Dictionary**

| **Data/Field** | **Description** |
| --- | --- |
| **127.0.0.1** | *This is the IP address of the client (remote host) which made the request to the server.* |
| **-** | *The "hyphen" in the output indicates that the requested piece of information is not available. In this case, the information that is not available is the RFC 1413 identity of the client determined by identd on the clients machine.* |
| **2134** | *This is the userid of the person requesting the document as determined by HTTP authentication.* |
| **[10/Oct/2000:13:55:36 -0700]** | *The time that the request was received. The format is: [day/month/year:hour:minute:second zone]* |
| **"GET /apache\_pb.gif HTTP/1.0"** | *The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache\_pb.gif, and third, the client used the protocol HTTP/1.0.* |
| **200** | *This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5). The full list of possible status codes can be found in the HTTP specification (RFC2616 section 10)* |
| **2326** | *The last part indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be "-". To log "0" for no content, use %B instead.* |

**Solution Approach**

This pipeline leverages the power of Docker containers to run Apache Airflow, Apache Spark, and PostgreSQL.Airflow Docker image has been used for scheduling the pipeline and the Spark Docker image for processing data. For data storage, I have utilized PostgreSQL, which is included in the Airflow Docker image.PostgreSQL has been chosen over other relational databases such as Oracle because it comes pre-installed in the Airflow Docker image, making it a convenient and efficient choice for local. By containerizing these technologies, we can easily manage dependencies and ensure consistency across environments. Additionally, this approach enables us to scale our infrastructure efficiently, allowing us to process large datasets quickly and accurately.

**ETL Pipeline Design approach**





**Source System** – Since the sample input data was not available, based on the data dictionary, log file has been generated in date format log file which has been scheduled in airflow DAG as a data extract pipeline for 23:30 everyday.

**Data Processing** – In this pipeline data processing has been done on Pyspark and processed data is been load in Postgres database and the pipeline has been scheduled in airflow for data processing and load the processed data into Postgres database

**View** – On top of Database two views has been created with the logic of distinct user, who made request and the total number of successful request has been made by the user, which can feed data in visualization tool

**Note** – Same approach can be done for Oracle database

All the services used here like spark, airflow, Postgres are running individually inside docker container

**Description on Docker-compose.yaml**

command - Docker-compose up -d (to start the docker container)

This Docker Compose file sets up a data pipeline consisting of several services running in containers.

* The first service is PostgreSQL, which is used as the database for Apache Airflow. The PostgreSQL container is configured with a volume that maps to a SQL script for initializing the database.
* The second service is Apache Airflow, which is built using a Dockerfile that defines the Airflow environment. The Airflow container is connected to the PostgreSQL container through a network and is configured with environment variables that specify the database connection. Additionally, the Airflow container is configured with volumes that map to DAGs and data directories. Running on a port - 8080 in localhost
* The third service is Adminer, which is a web-based database management tool. The Adminer container is connected to the PostgreSQL container through a network and is accessible through a specified port. Running on a port - 8081 in localhost
* The fourth and fifth services are Apache Spark, with one service acting as the master and the other as a worker. Both Spark containers are built using the same Dockerfile and are connected to the same network. The Spark master container is configured with environment variables that specify the master mode and the host and port for the Spark master. The Spark worker container is configured with environment variables that specify the worker mode and the URL for the Spark master. Additionally, the Spark worker container is configured with volumes that map to data and source code directories. Running on a port - 8082 in localhost

Overall, this Docker Compose file sets up a complete data pipeline environment that includes scheduling, processing, and storage capabilities.

**Steps to run this pipeline**

Steps to run the pipeline through Makefile

1. env\_up - this command will do the docker-compose up -d

after doing this all the services will be up and running in the docker container, and this will spin up 3 web server

* <http://localhost:8081/> - for airflow
* <http://localhost:8083/> - spark
* <http://localhost:8082/> - adminer for postgres

1. We can directly run the the airflow dags, which has two dags one for data extract and one for data process and load to database, data extract dag has been scheduled for 23: 30 CET and data process dag has been scheduled for 23:40 CET
2. We can visualize the output from dag in the adminer, where views has also been created

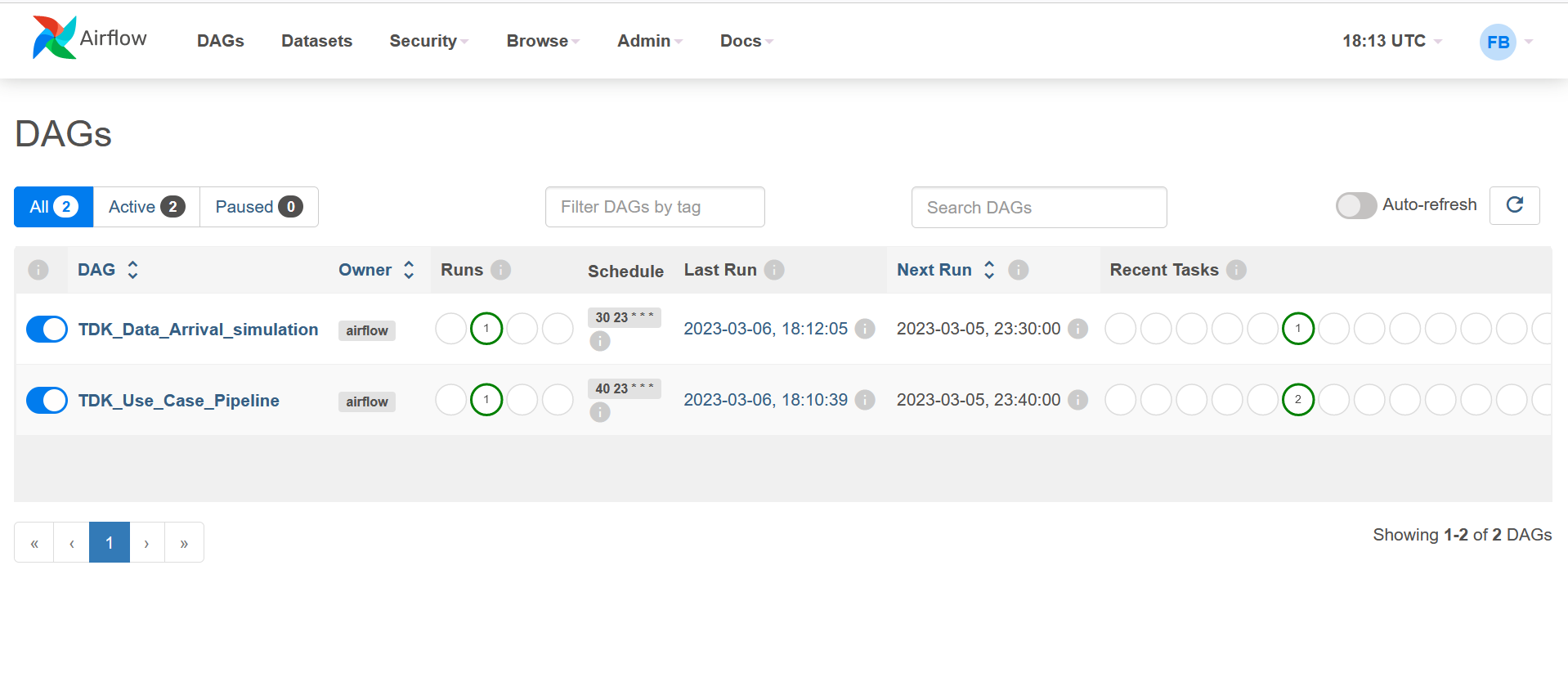
Manually run without airflow

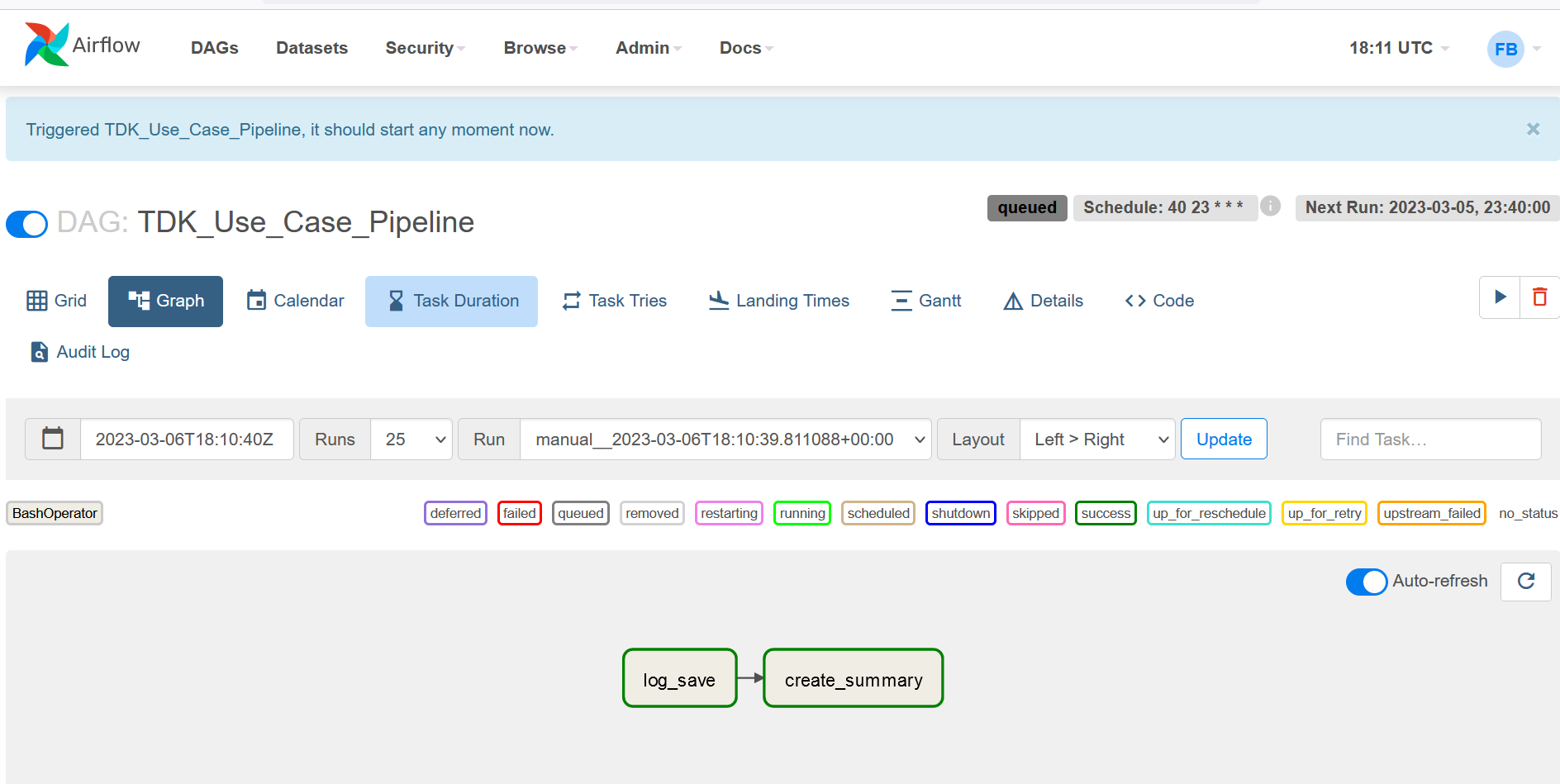
This pipeline can manually run through makefile

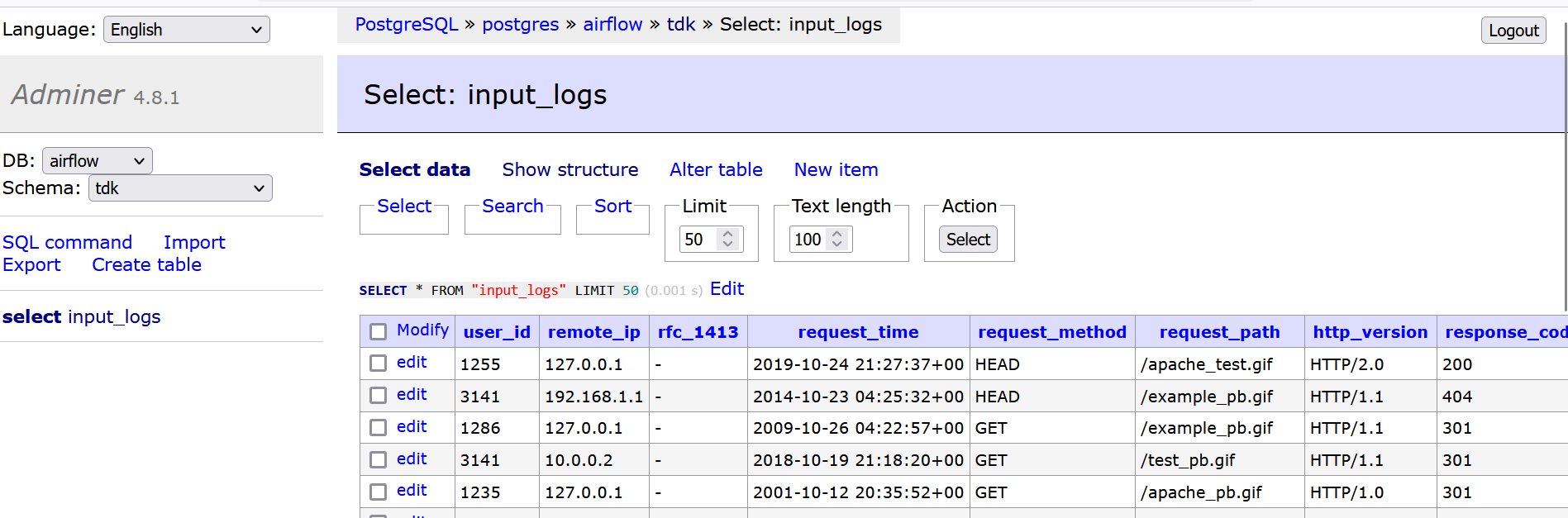
* make env\_up
* python gererate\_log.py - to generate the input file
* make env run - this will execute the spark-submit command for data process and load
* make env\_down - this will do docker-compose down
* make env\_clean - this will clean all the existing docker image for spark worker, master and airflow

docker rmi spark-master-img spark-worker-img airflow-img

This pipeline also covered some basic test cases





****

End of the file