



INSTITUTE OF BUSINESS ADMINISTRATION
BIG DATA ANALYTICS

PROJECT

DATA LAKE USING APACHE NIFI, MINIO (OBJECTS), POSTGRES (RELATIONAL DATABASE), AIRFLOW(DAGS), APACHE SUPERSET FOR DASHBOARD

SUBMITTED BY

Zeerak Ajmal Kakar
Gohar Mumtaz
Noor Mustufa Bhutto

About the Data:

The dataset is about Airline passenger satisfaction. It consists of ratings given by the passengers on different services provided by the Airline such as cleanliness, wifi-service, leg room, online boarding, food and drink, luggage handling among others. The dataset has passengers divided into types such as loyal and disloyal customer, and also the kind of class they take Economy, or business class. The data was collected from different sources and transformed into one. The size of the data is 1.7 Gigabytes and has more than 5000000 + rows and more than 35 columns. The data format is CSV. Using this dataset, we aim to design a data lake with data streaming into different components and then getting outputs in the form of dashboards to gain meaningful insights from the data. We also used objected-based text data to feed into MinIO which is then connected to Apache Nifi and managed through airflow.

The Data Pipeline:

We used 7 different tools to solve our big data problem.

1. MINIO

To stream object based data into it and then connect it to Apache Nifi.

2. Apache Nifi

It connects the two data streams (Relational and object based) and defines rules for data processing and distribution.

3. Apache Registry

To store and manage different programs on Nifi and version control of the different data flows and create templates or rules that can be used repeatedly for different scenarios.

4. pgAdmin

Administration of the Relational Database (PostgreSQL)

5. PostgreSQL

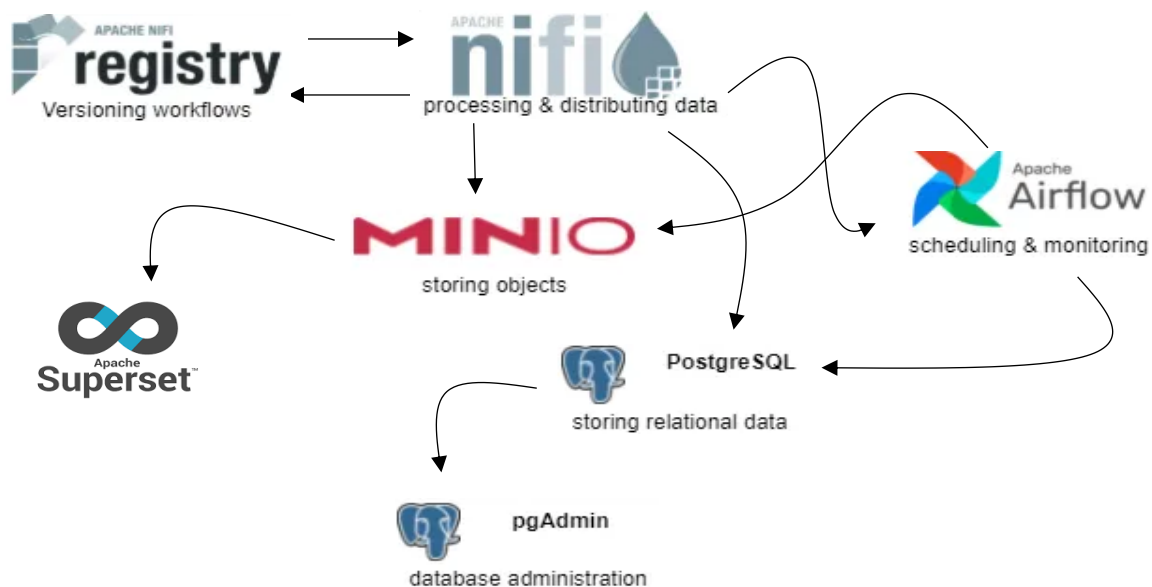
Ingest and analyze Relational Database.

6. Apache Airflow

To schedule workflows, define and manage data pipelines using DAGs that consists of tasks and dependencies.

7. Apache Superset

To design and publish dashboards to gain insightful information about the data.



















































The Required Containers

The first step was to pull and run all the required containers.

We used the following images to pull the containers.

- a) Airflow: `puckel/docker-airflow:1.10.9`
- b) Zookeeper: `bitnami/zookeeper:3.7.0`
- c) NiFi: `apache/nifi:1.14.0`
- d) NiFi registry: `apache/nifi-registry:1.15.0`
- e) MinIO: `bitnami/minio:2021`
- f) Postgres: `postgres:14-bullseye`
- g) pgadmin: `dpape/pgadmin4:6.1`
- h) `apache/superset:latest`

As per our data pipeline we pulled all necessary images using docker and ran the containers. The list of all running containers can be seen below.

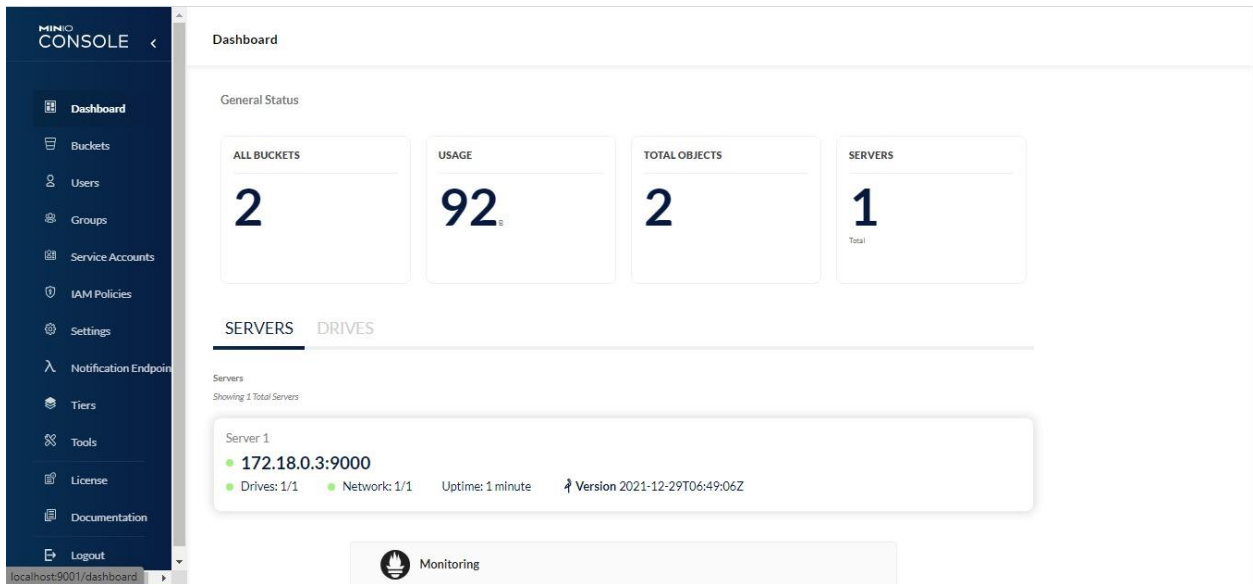
<input type="checkbox"/>	 superset e24428fad248 	apache/superset	Running	8088:8088 	30 minutes ag 		
<input type="checkbox"/>	 minio_container 0767b8264a7b 	bitnami/minio:2021	Exited (255)	9000:9000  9001:9001 			
<input type="checkbox"/>	 registry_container f911bad27536 	apache/nifi-registry:1.15.0	Exited (137)	18080:18080 			
<input type="checkbox"/>	 airflow_container 6c01f00599f5 	puckel/docker-airflow:1.10.9	Exited (137)	8085:8080 			
<input type="checkbox"/>	 nifi_container d5e194799eae 	apache/nifi:1.14.0	Exited (137)	8091:8080 			
<input type="checkbox"/>	 postgres_container cb1c8e4078b2 	postgres:14-bullseye	Running	5432:5432 	33 minutes ag 		
<input type="checkbox"/>	 pgadmin_container a605df53702b 	dpape/pgadmin4:6.1	Running	5050:80 	33 minutes ag 		
<input type="checkbox"/>	 zookeeper_container db4e011fbc78 	bitnami/zookeeper:3.7.0	Running		33 minutes ag 		

Building the Data Pipeline (Steps)

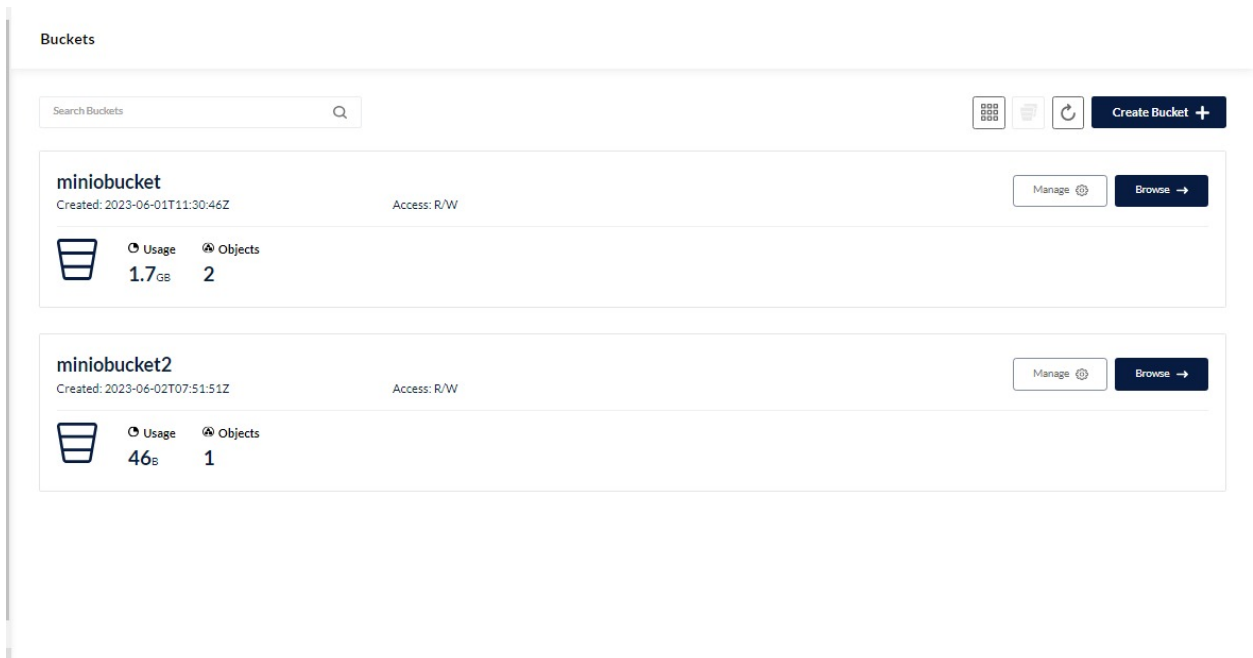
A. MinIO locally hosted object storage (AWS S3)

The first step was to setup the configuration of MinIO for object-based storage. We created buckets of unstructured text based data MinIO's scalability option for storing and organizing the large amounts of data. This data is then connected with Apache Nifi for data processing and Distribution.

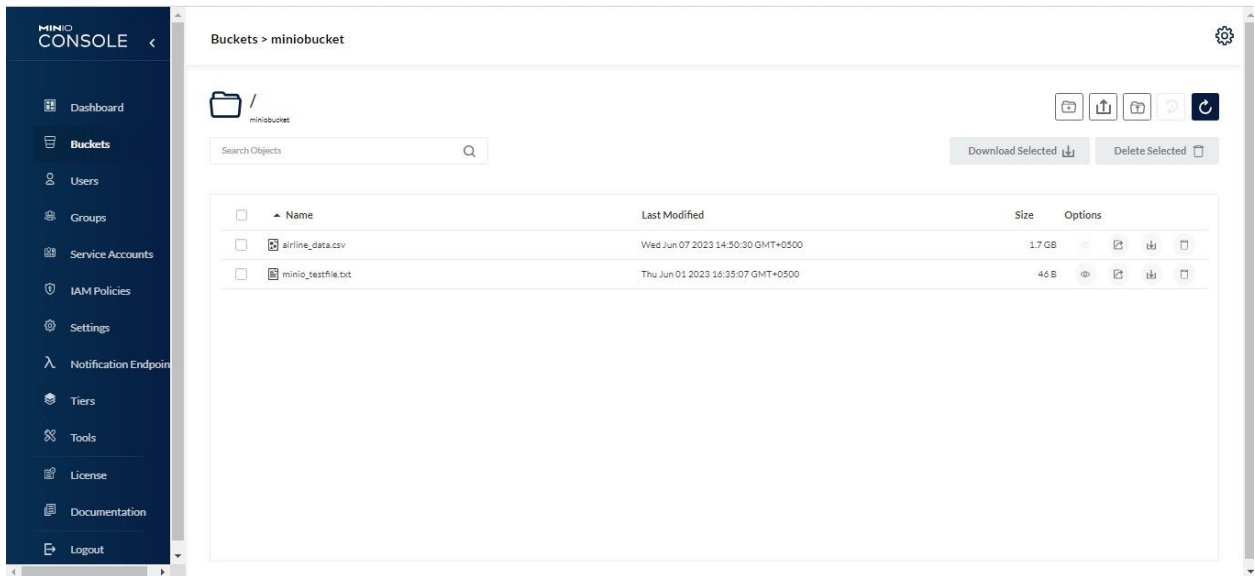
We can see the main dashboard display of the MinIO dashboard which displays the number of buckets, memory usage, total number of objects and the services being used.



Below we can see the created buckets. We initially created a test bucket to test on a small size of data, then we added the actual data file of our use-case.



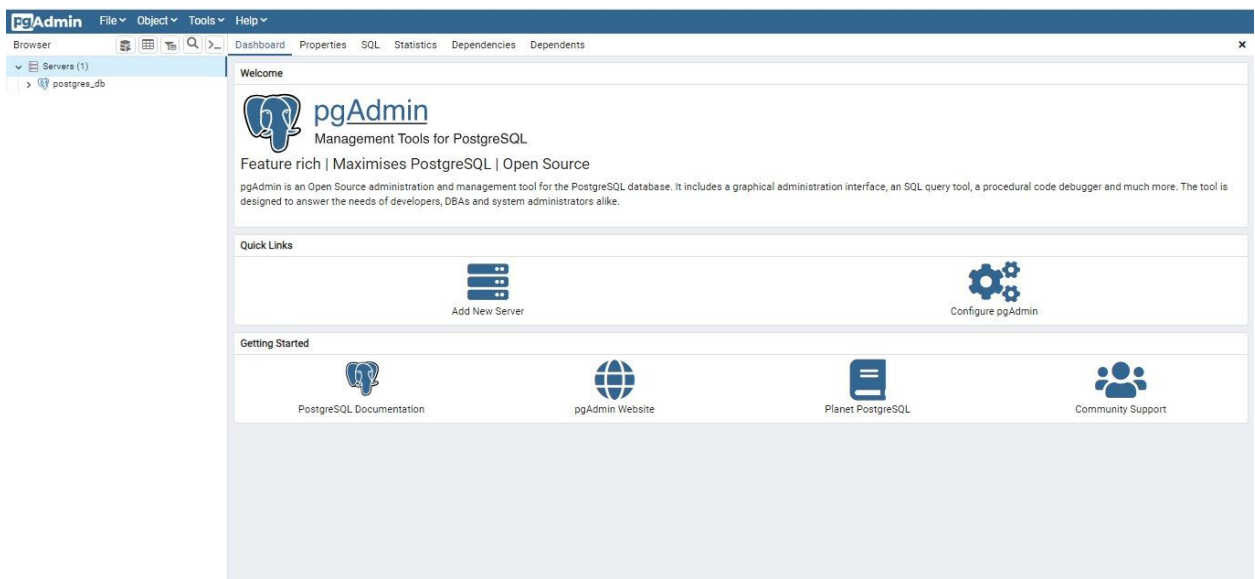
When we go inside the data bucket we can see the contents of the bucket. This created bucket contains two files, a test data and our final use-case data which was 1.7 GB.



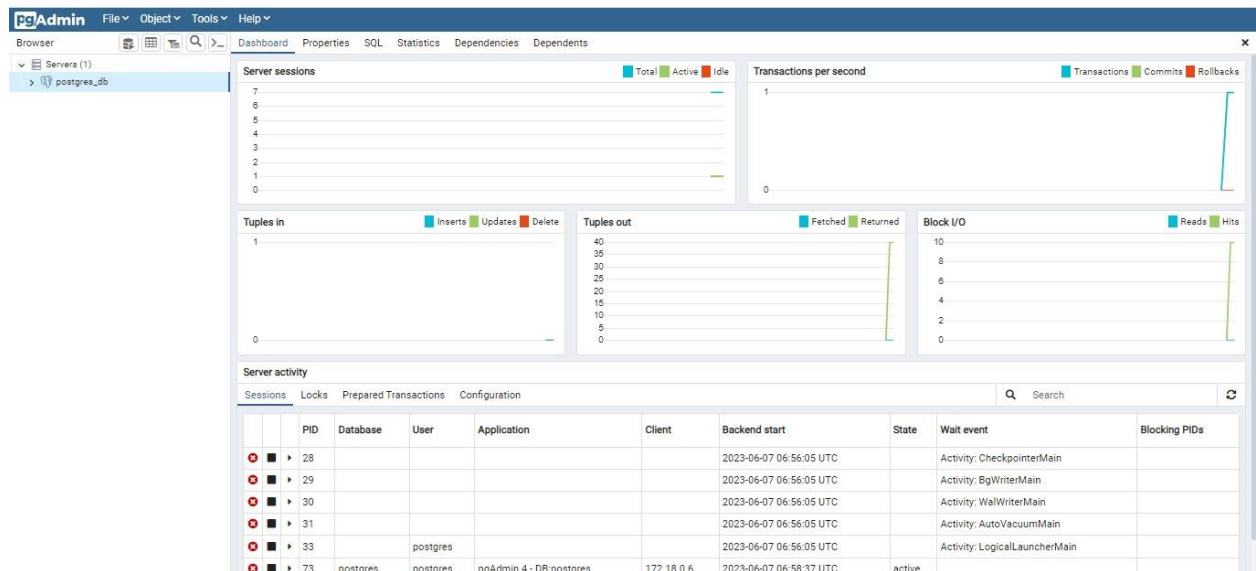
B. pgAdmin for PostgreSQL Administration

Alongside MinIO, we simultaneously set up the pgAdmin for parallel processing. We hosted its – web version using docker and setup the database administration to manage our PostgreSQL relational database.

This is the main interface of the pgAdmin running on our localhost where we can see our dashboard and setup our services and connections. This also allows us object-relational database management.



This is the dashboard of pgAdmin, where we created our server for the PostgreSQL database. We can see our server sessions and the activities as well. In the left pane we can see our running servers.

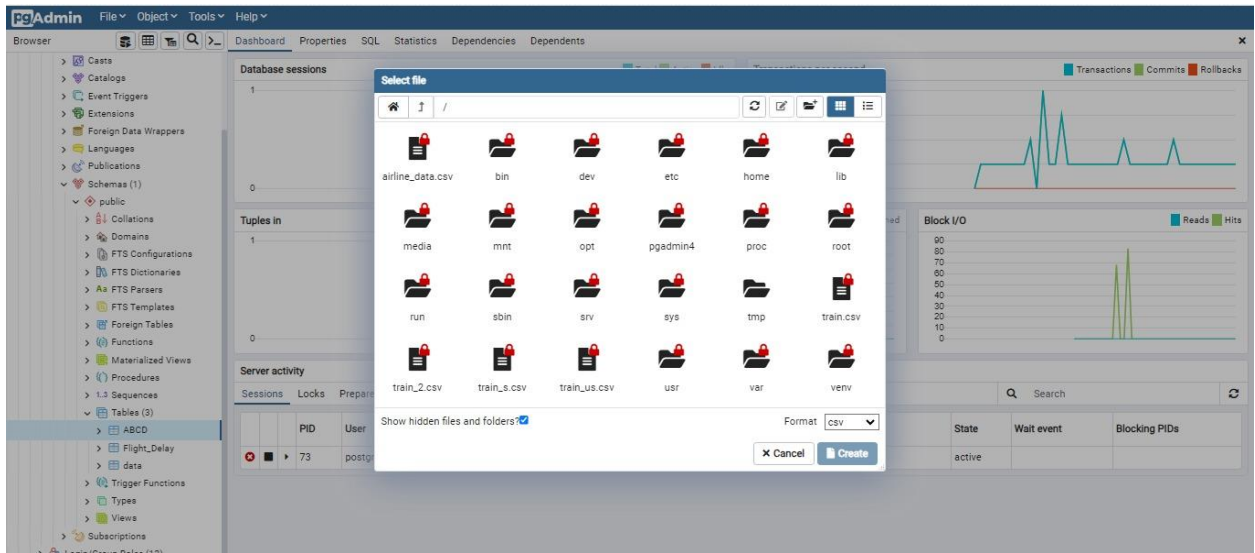


C. PostgreSQL to manage Relational Database (Object)

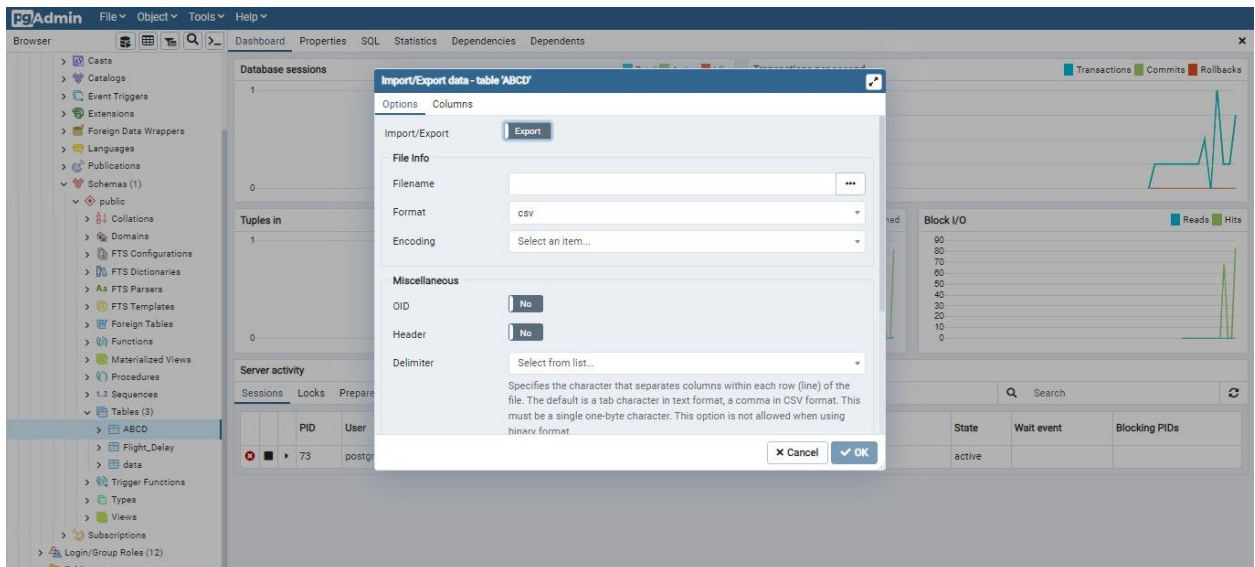
In order to create the relation database, first we copy the file into the tmp folder of the pgAdmin using docker command so that it can be accessed inside the pgAdmin container.

```
E:\IBA\2nd_Semester\BigDataAnalytics\BDA\Project>docker cp E:\IBA\2nd_Semester\BigDataAnalytics\BDA\Project\train_us.csv pgadmin_container:train_us.csv
```

Then run the pgAdmin container on the local host and we can see the user interface where we can view the contents of the tmp folder. Here we find our copied file so that we can import into the pgAdmin container which will be used for further processing.



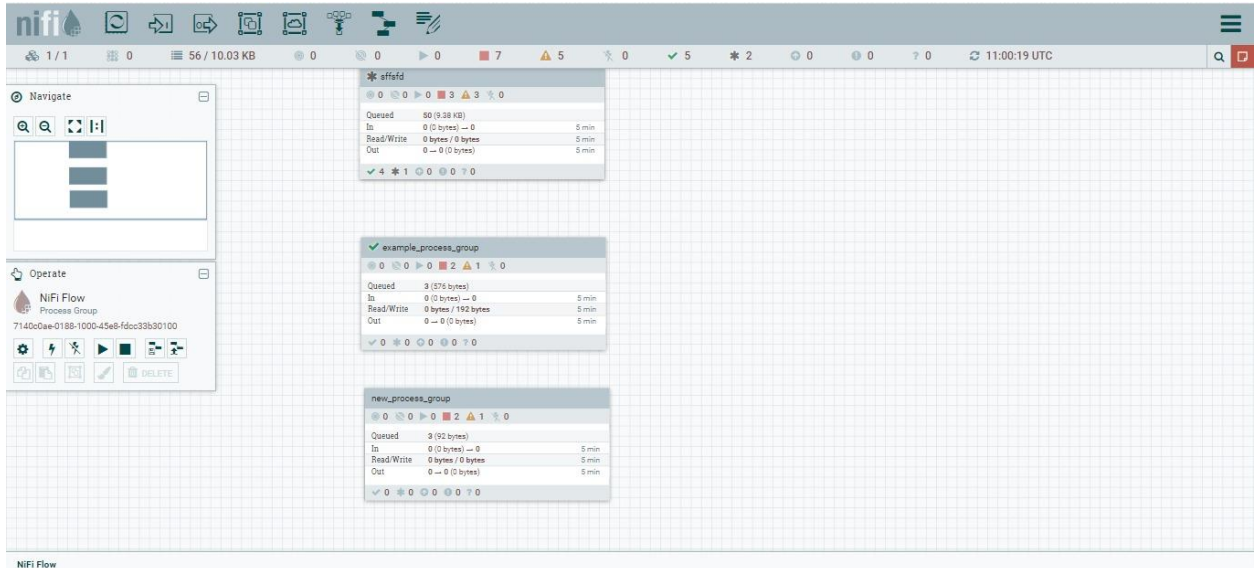
After we select the file, we get this pop up where we can select import the file into the pgAdmin container where we can then perform further operations such as database creation, schema creation and then table creation using the CRUD commands.



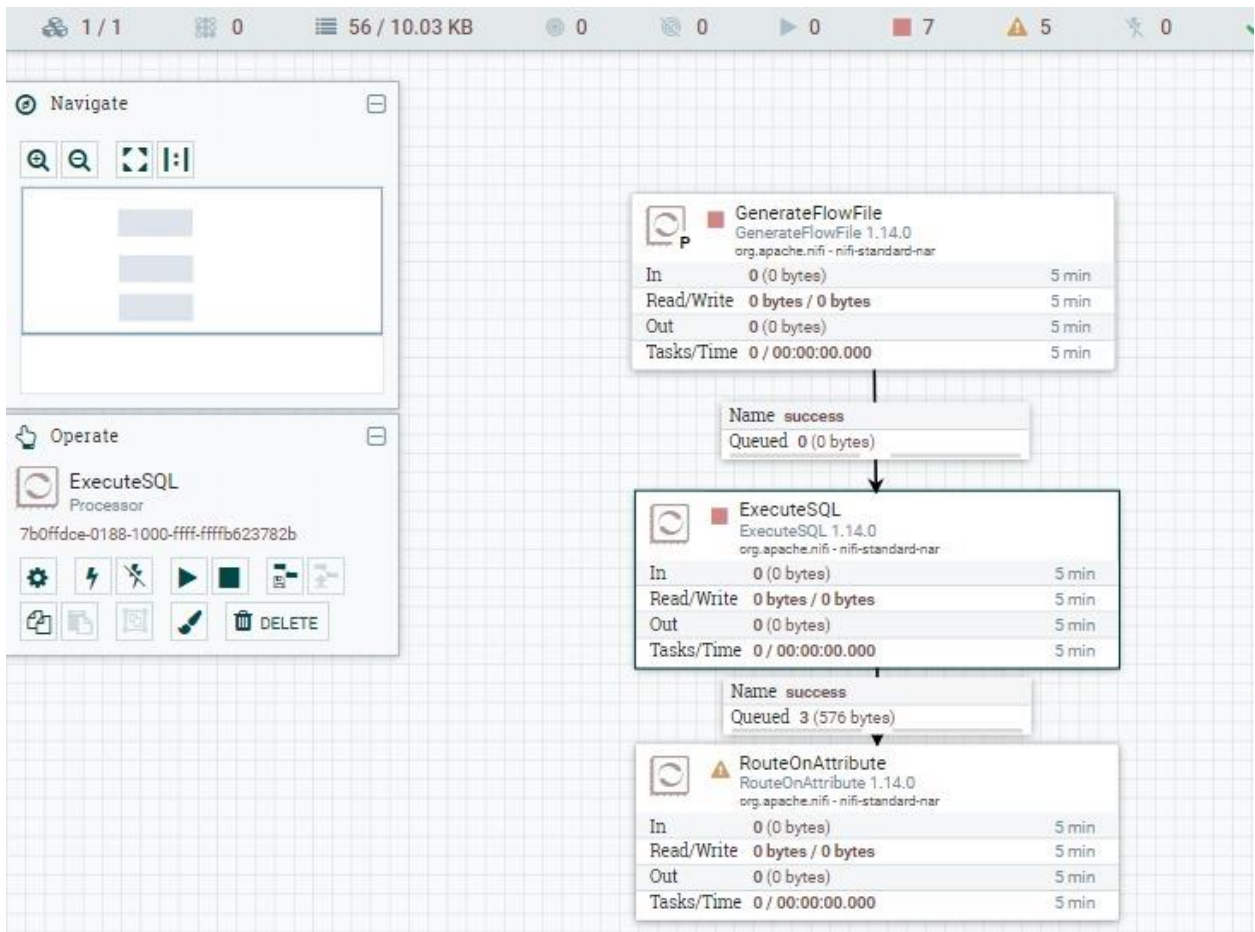
D. Apache Nifi for Data Processing and Distribution

First we define the relationship between the different data streams using Apache Nifi by defining the process groups.

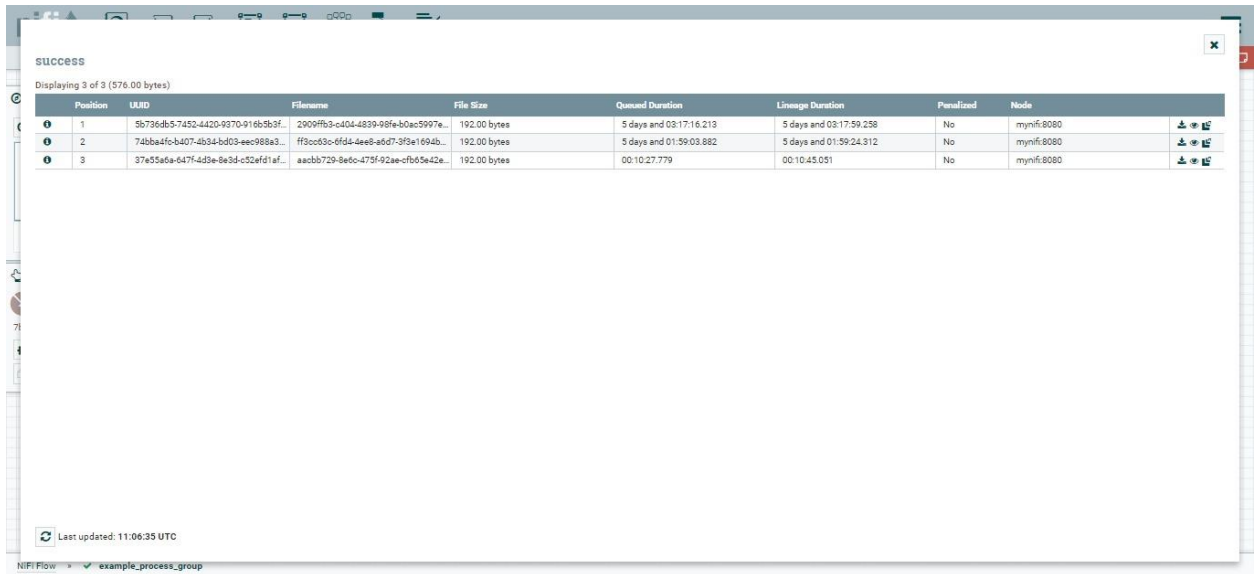
We defined two process groups one for relational database PostgreSQL and the other for object-based MINIO.



Inside the First process group we have SQL Process defined where we first generate a flow file and then connect to Execute SQL where we can access the file. We can also define other rules here and design the template that how this relational database interacts with other data streams flowing into the network.



Here we can see the created log for the activities for PostgreSQL where we can edit and view the contents of each activity log.



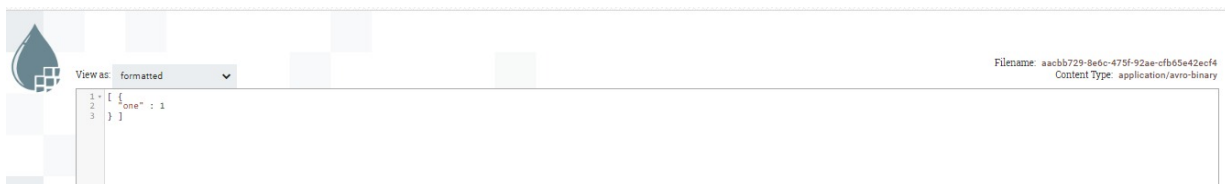
Displaying 3 of 3 (576.00 bytes)

Position	UUID	Filename	File Size	Queue Duration	Lineage Duration	Parallelized	Node
1	5b736db5-7452-4420-9370-916b5b3f...	2009ff93-c404-4839-98fe-b0ac5997e...	192.00 bytes	5 days and 03:17:16.213	5 days and 03:17:59.258	No	my/nifi:8080
2	74dbba4f-b407-4b54-bd03-ee0988a3...	ff3c0d3c-6fda-dee8-a6d7-3f3e1694b...	192.00 bytes	5 days and 01:59:03.882	5 days and 01:59:24.312	No	my/nifi:8080
3	37e55a6a-647f-4d3e-8e3d-c5caf61af...	aacbb729-8e6c-475f-92ae-cfb65e42e...	192.00 bytes	00:10:27.779	00:10:43.051	No	my/nifi:8080

Last updated: 11:06:35 UTC

Nifi Flow > example_process_group

This displays the Postgres connection to Nifi where an activity tag can be seen.

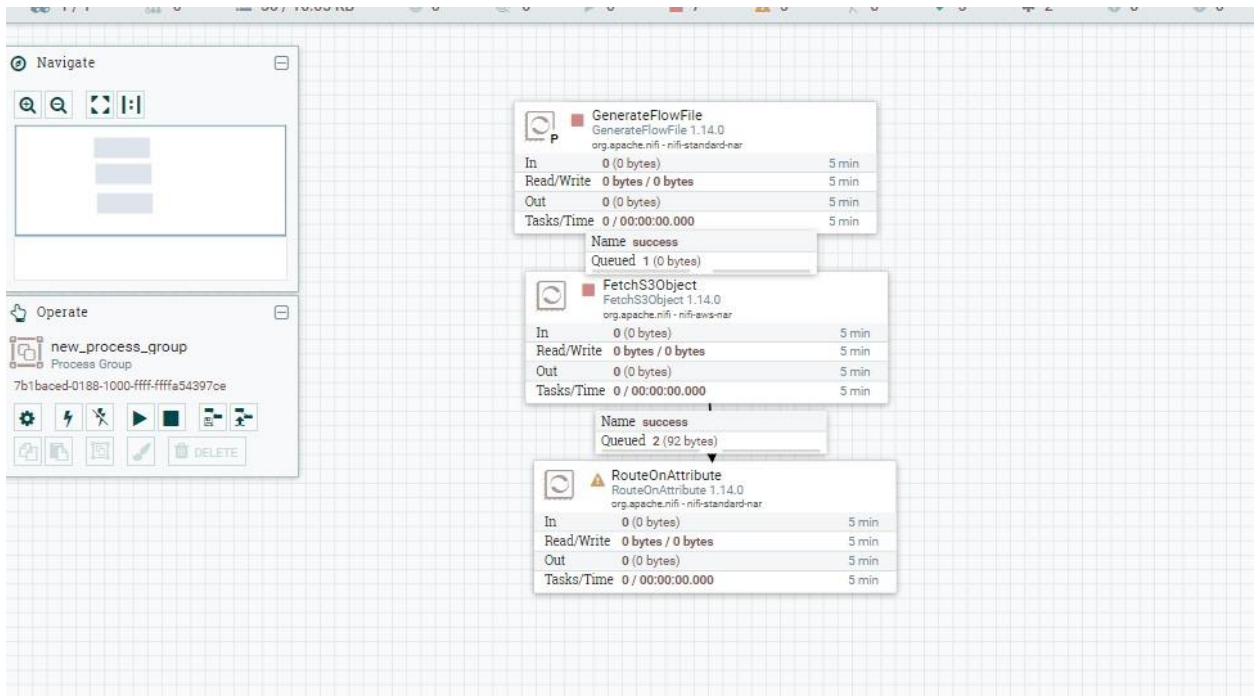


View as: formatted

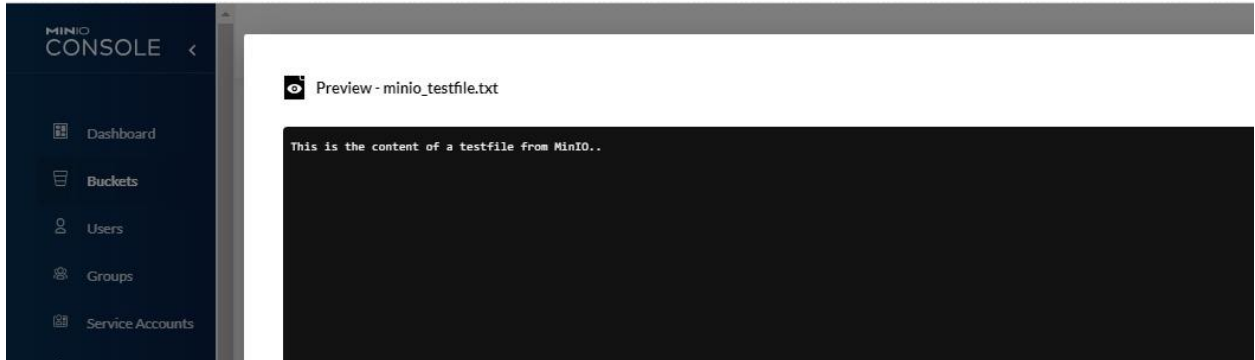
Filename: aacbb729-8e6c-475f-92ae-cfb65e42ecf4
Content Type: application/avro-binary

```
1. [ {  
2.   "one" : 1  
3. } ]
```

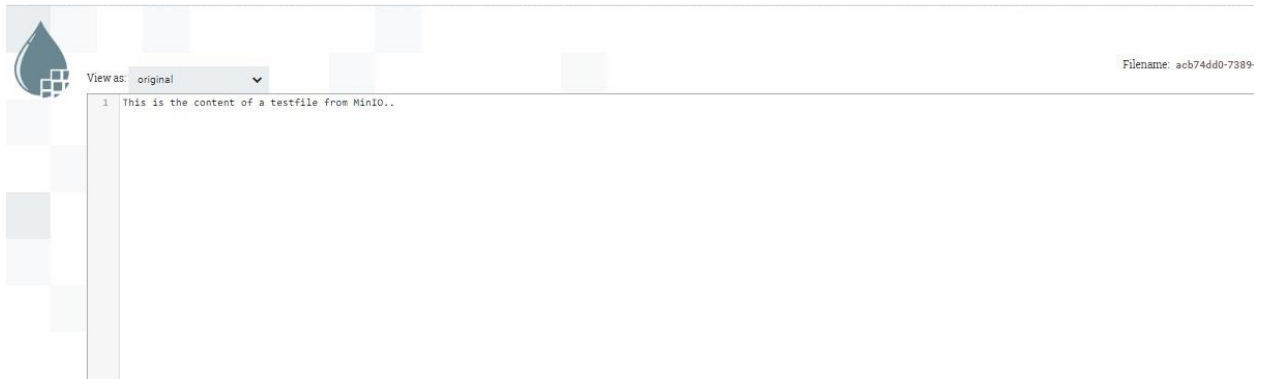
Upon entering the second process group we can see the object-based process for our text file where we first generate flow file and then fetch the related object and execute it in real time. These are rules defined so that every time an object is detected it follows this designed template for further processes.



This is the actual object file which can be seen in the MINIO container, we will establish a connection with this file inside Apache NIFI by generating flow and then executing to perform other functions.



Here we can see the MINIO file fetched inside Apache Nifi as per the designed flow inside the process groups.




E. Apache Registry to Store and Manage

Created registry to store and manage different programs on nifi

F. Apache Airflow to monitor workflows.

Directed acyclic graphs created for Minio / Nifi / Postgres

Airflow

DAGs

Data Profiling ▾

Browse ▾

Admin ▾
































Docs ▾

About ▾

2023-06-07 17:06:24 UTC

DAGs

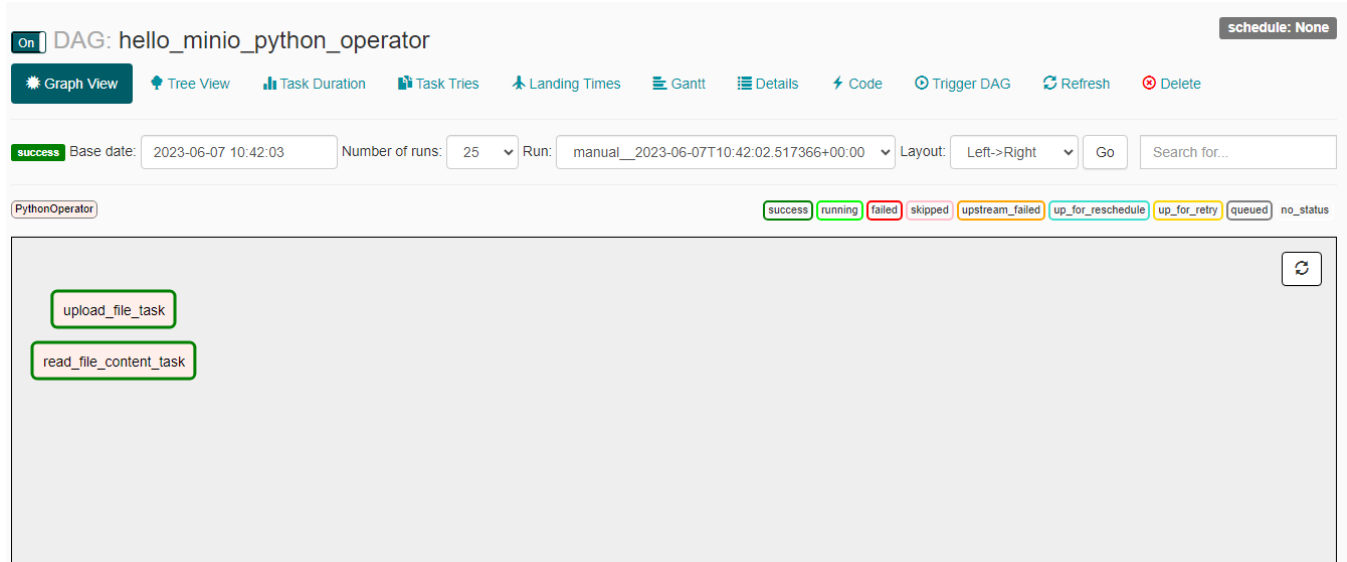
Search:

		DAG	Schedule	Owner	Recent Tasks 	Last Run 	DAG Runs 	Links
	<div>On</div>	hello_minio_python_operator	<div>None</div>	airflow	<div><div>2</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	2023-06-07 10:42 	<div><div>5</div><div>4</div></div>	<div></div>
	<div>On</div>	hello_nifi	<div>0 3 * * *</div>	airflow	<div><div>2</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	2023-06-07 07:52 	<div><div>3</div><div>1</div></div>	<div></div>
	<div>On</div>	hello_postgres_postgres_operator	<div>None</div>	airflow	<div><div>2</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	2023-06-07 17:04 	<div><div>4</div><div>7</div></div>	<div></div>

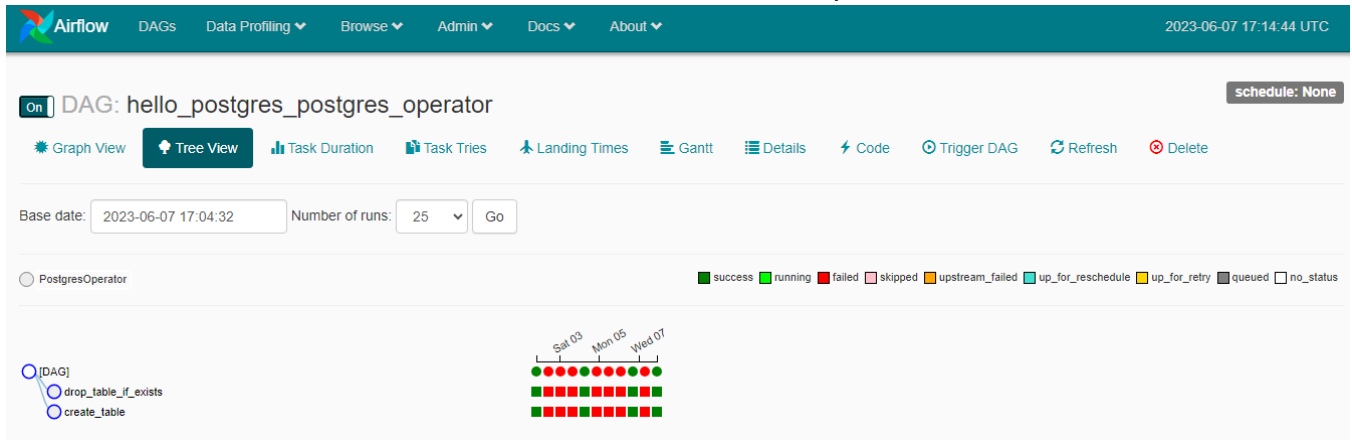
Connections with ports to all different containers

<input type="checkbox"/>		myminio_connection	s3				
<input type="checkbox"/>		mynifi_connection	postgres	http://mynifi	8080		
<input type="checkbox"/>		mypostgres_connection	postgres	mypostgres	5432		

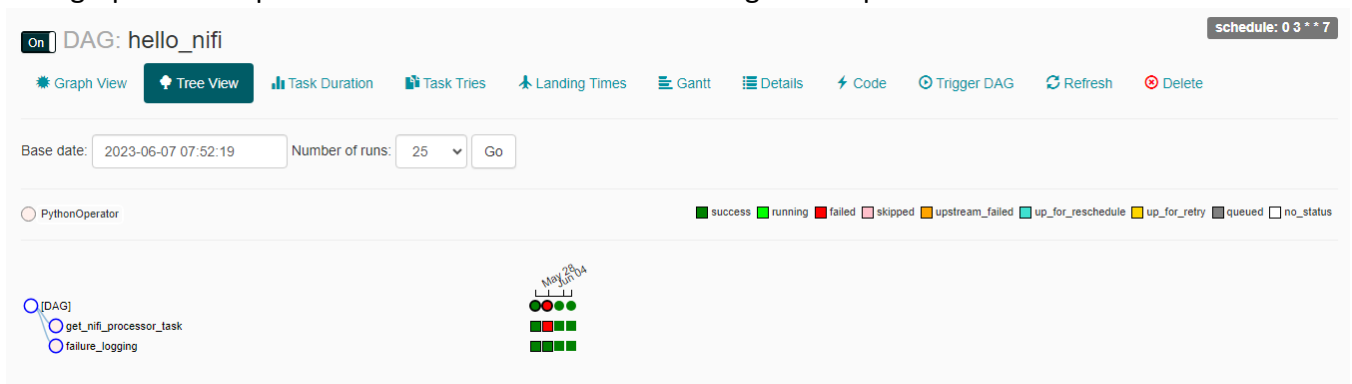
A successful execution where a DAG uploads a file and reads file contents.

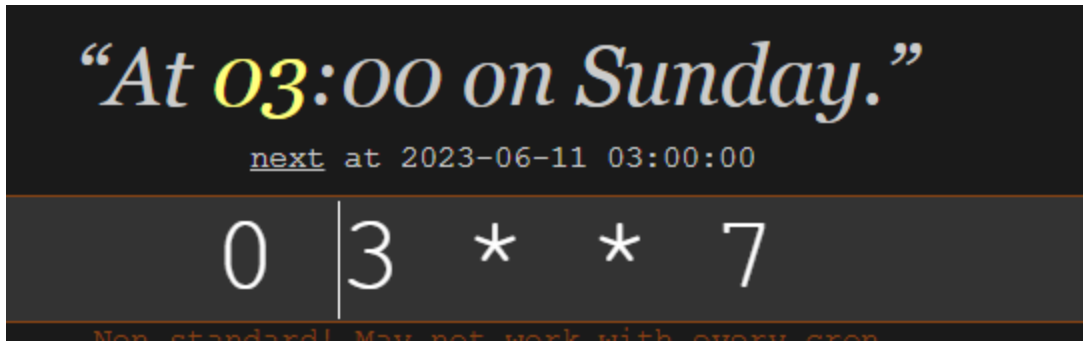


DAG which creates a table but first it removes a table if it already exists



DAG graph for NIFI processor. There is a scheduler running cron expression





Airflow orchestrating workflow on MINIO for uploading file task logs show successful.

Airflow DAGs Data Profiling ▼ Browse ▼ Admin ▼ Docs ▼ About ▼ 2023-06-07 17:08:01 UTC

Task Instance: upload_file_task 2023-06-07 10:42:02

Task Instance Details Rendered Template Log XCom

Log by attempts

1 Toggle wrap Jump to end

```
*** Reading local file: /usr/local/airflow/logs/hello_minio_python_operator/upload_file_task/2023-06-07T10:42:02.517366+00:00/1.log
[2023-06-07 10:42:06,870] {{taskinstance.py:655}} INFO - Dependencies all met for <TaskInstance: hello_minio_python_operator.upload_file_task 2023-06-07T10:42:02.517366+00:00 [queued]
[2023-06-07 10:42:06,875] {{taskinstance.py:655}} INFO - Dependencies all met for <TaskInstance: hello_minio_python_operator.upload_file_task 2023-06-07T10:42:02.517366+00:00 [queued]
[2023-06-07 10:42:06,876] {{taskinstance.py:866}} INFO -
-----
[2023-06-07 10:42:06,877] {{taskinstance.py:867}} INFO - Starting attempt 1 of 1
[2023-06-07 10:42:06,877] {{taskinstance.py:868}} INFO -
-----
[2023-06-07 10:42:06,886] {{taskinstance.py:887}} INFO - Executing <Task(PythonOperator): upload_file_task> on 2023-06-07T10:42:02.517366+00:00
[2023-06-07 10:42:06,889] {{standard_task_runner.py:53}} INFO - Started process 869 to run task
[2023-06-07 10:42:06,942] {{logging_mixin.py:112}} INFO - Running %s on host %s <TaskInstance: hello_minio_python_operator.upload_file_task 2023-06-07T10:42:02.517366+00:00 [running]>
[2023-06-07 10:42:07,119] {{logging_mixin.py:112}} INFO - [2023-06-07 10:42:07,119] {{S3_hook.py:201}} INFO - Not Found
[2023-06-07 10:42:07,244] {{python_operator.py:114}} INFO - Done. Returned value was: None
[2023-06-07 10:42:07,247] {{taskinstance.py:1048}} INFO - Marking task as SUCCESS.dag_id=hello_minio_python_operator, task_id=upload_file_task, execution_date=20230607T104202, start_d
[2023-06-07 10:42:16,865] {{logging_mixin.py:112}} INFO - [2023-06-07 10:42:16,865] {{local_task_job.py:103}} INFO - Task exited with return code 0
```

G. Apache Superset for Dashboard

After pulling the container and running Apache Superset on localhost we setup the data source for dashboards. We selected PostgreSQL as our data source here.

Connect a database

×

STEP 1 OF 3

Select a database to connect

PostgreSQL

Presto

MySQL

SQLite

Or choose from a list of other databases we support:

SUPPORTED DATABASES

Choose a database...

Want to add a new database?

Any databases that allow connections via SQL Alchemy

We connect the data source by defining the credentials and the database name and host.

Connect a database

×

STEP 2 OF 3

Enter the required PostgreSQL credentials

Need help? [Learn more about connecting to PostgreSQL..](#)

HOST *

PORT *

postgres

5432

DATABASE NAME *

postgres

Copy the name of the database you are trying to connect to.

USERNAME *

postgres

PASSWORD

.....

DISPLAY NAME *

PostgreSQL

Pick a nickname for how the database will display in Superset.

ADDITIONAL PARAMETERS

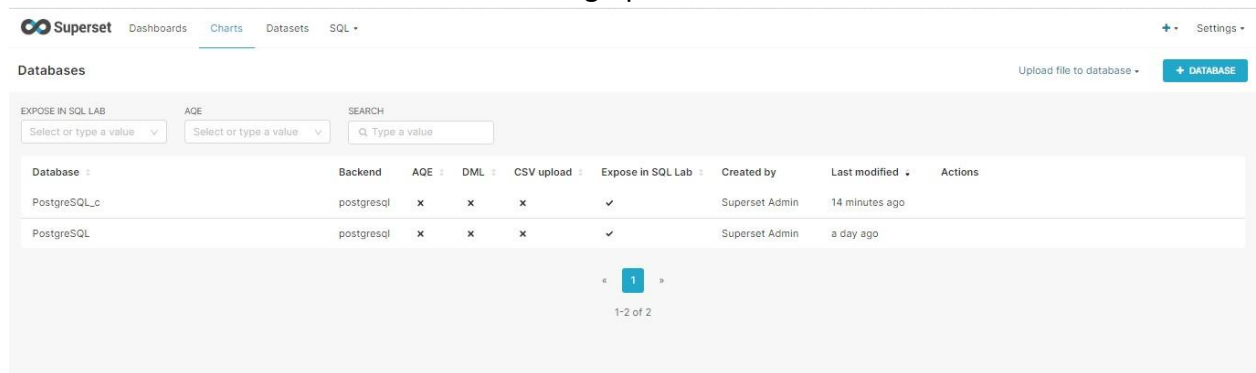
e.g. param1=value1¶m2=value2

[Add additional custom parameters](#)

BACK

CONNECT

Once we are done with defining the data source, we can see the databases inside the Superset which can now be used to create charts and graphs.



We have created a dashboard in Apache Superset using the Air Flight Satisfaction dataset. The purpose of this dashboard is to provide insights into the total number of records, as well as the breakdown of data by gender and satisfaction, and class and customer satisfaction.

To begin, we added the Air Flight Satisfaction dataset as a data source in Apache Superset. This dataset contains information about air flight satisfaction and is stored in a compatible format. we accessed the Superset UI and navigated to the "Data" menu, where I selected "Databases" and added a new database connection with the details of my dataset.

Once the data source was set up, we proceeded to create a new dashboard. I named it accordingly and saved it for future reference. This dashboard serves as a container for the visualizations I wanted to include.

Next, we added a chart to the dashboard by clicking on the "+" button within the dashboard view. I chose the appropriate visualization type to display the total number of records. In this case, I opted for a "Number" chart. I made sure to select the Air Flight Satisfaction dataset as the data source for this chart.

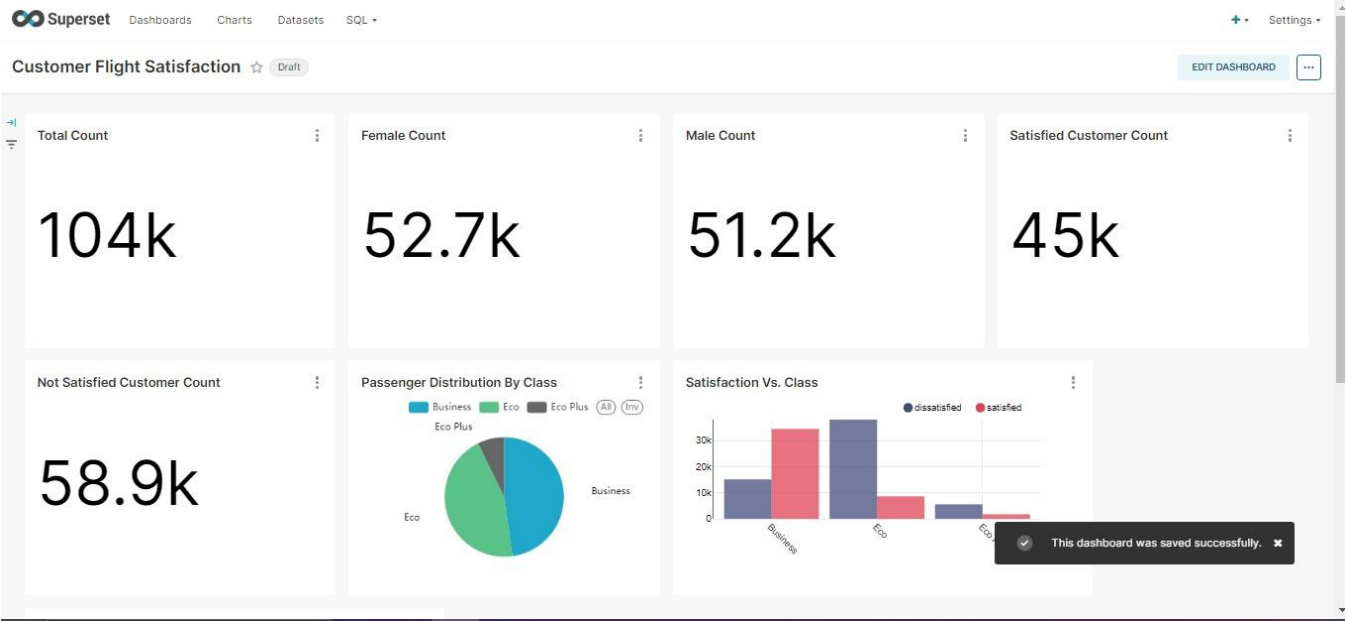
To calculate the total number of records, we set the metric as "COUNT(*)" within the chart configuration. This allowed me to obtain an accurate count of the records in the dataset.

To provide further insights, we added filters to the dashboard. Specifically, I included separate filters for gender and satisfaction to enable users to explore the data based on their specific criteria. By selecting different values for these filters, users can refine the results and gain deeper insights into the dataset.

To further enhance the dashboard, we added a second chart to display the breakdown of data by class and customer satisfaction. Following the same steps as before, we selected an appropriate visualization type, configured the metrics accordingly, and added the necessary filters.

After arranging the charts within the dashboard view to my satisfaction, we saved the changes and published the dashboard. This ensures that the insights and visualizations are available for

future analysis and sharing with other stakeholders.



Superset Dashboards Charts Datasets SQL

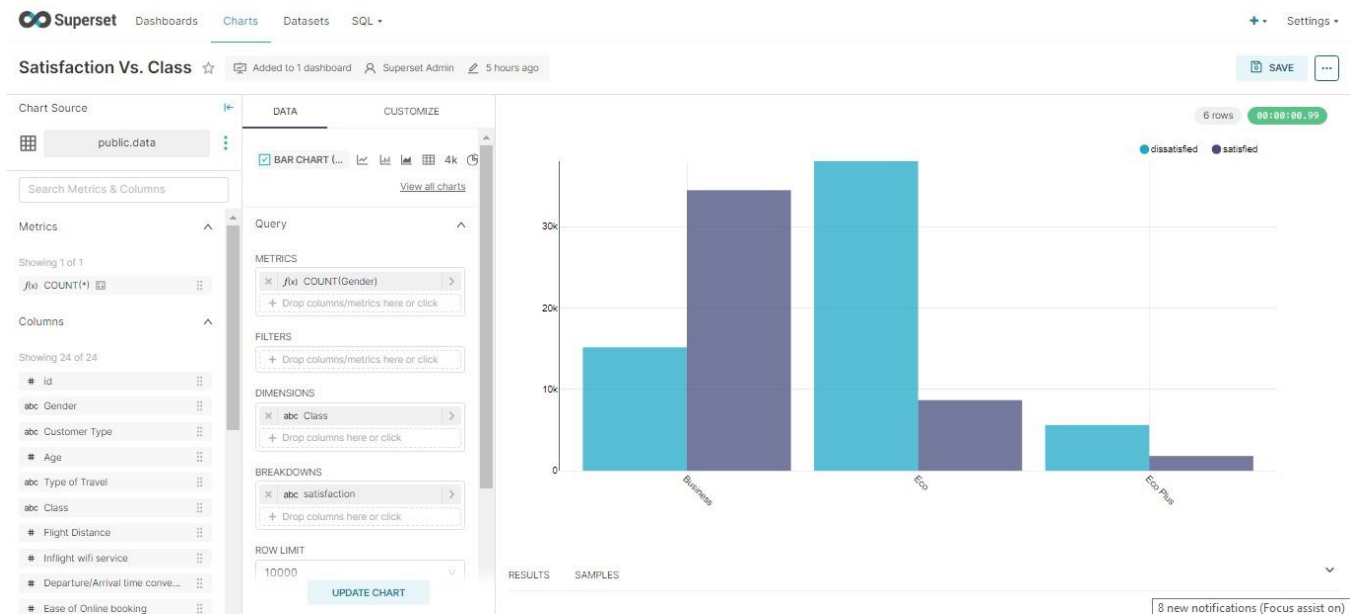
Charts BULK SELECT + CHART

OWNER: Select or type a value
CREATED BY: Select or type a value
CHART TYPE: Select or type a value
DATASET: Select or type a value
DASHBOARDS: Select or type a value
FAVORITE: Select or type a value
CERTIFIED: Select or type a value

SEARCH: Q Type a value

Chart	Visualization type	Dataset	Modified by	Last modified	Created by	Actions
☆ Satisfaction Vs. Class	Bar Chart (legacy)	public.data	Superset Admin	5 hours ago	Superset Admin	
☆ Passenger Distribution By Class	Pie Chart	public.data	Superset Admin	7 hours ago	Superset Admin	
☆ Passenger Distribution By Class	Pie Chart	public.data	Superset Admin	7 hours ago	Superset Admin	
☆ Not Satisfied Customer Count	Big Number	public.data	Superset Admin	8 hours ago	Superset Admin	
☆ Satisfied Customer Count	Big Number	public.data	Superset Admin	8 hours ago	Superset Admin	
☆ Male Count	Big Number	public.data	Superset Admin	8 hours ago	Superset Admin	
☆ Female Count	Big Number	public.data	Superset Admin	10 hours ago	Superset Admin	
☆ Total Count	Big Number	public.data	Superset Admin	10 hours ago	Superset Admin	
☆ Distribution	Bar Chart	public.data	Superset Admin	11 hours ago	Superset Admin	

1



Concluding Remarks

In conclusion, our big data project involved various tasks, including pulling images and running containers, which proved to be relatively straightforward and easily accomplished. However, the real challenge arose when it came to integrating and configuring these components into a cohesive pipeline. Creating seamless bridges between the different technologies in our stack required careful consideration and in-depth understanding. Nonetheless, through diligent effort and collaboration, we were able to overcome these challenges and successfully build a holistic pipeline for our project. For a comprehensive set of commands and instructions, please refer to the separate file attached to this report.