Paper Title: Exploring Apache Airflow for Enhanced Workflow Management and Error Handling in Scheduling Apache Spark Jobs: A Case Study in Stock Prediction

Course: MET CS777 A1
Assignment: Term Paper Proposal
Team Members: Chuong Nguyen – Zhen Cao

Abstract: This paper investigates the utility of Apache Airflow for scheduling Apache Spark jobs, emphasizing workflow optimization and error management. Focusing on stock prediction as a case study, we explore Airflow's Directed Acyclic Graphs (DAGs) and demonstrate their effectiveness in automating task scheduling and error handling. By integrating data collection, preprocessing, and machine learning model deployment, we showcase how Airflow streamlines workflow management, facilitating more informed investment decisions. Through this research, we highlight the potential of Airflow in enhancing efficiency and reliability in big data analytics workflows.

Table of Contents

I. Introduction

A. Background

Apache Airflow is a management platform first engineered by Airbnb in 2014 to support internal work. The platform offers a more effective and efficient solution for dealing with complex data engineering workflows. Specifically, it allows pipelines with multiple tasks and stages to be scheduled and arranged using Directed Acyclic Graphs (DAGs). DAG designers and users can robustly schedule, edit, debug, or create plans for every task within the pipeline in case of failure or error (Airflow, 2024).

The Airflow project was initiated in late 2014 and has been an open-source project since day one. The software was first made public on Airbnb's GitHub with the first commit in June 2015. Subsequently, the project became part of the Apache Software Foundation in 2016, and updates are still being released every three months. (Airflow, 2024)

B. Motivation

Has Moore's Law truly ended? After a long 60-year reign through the digital era and the evolution of the World Wide Web, the assertion about the development of technology is now being questioned, as it may no longer be able to keep up with the burgeoning of data (Tardi, 2024). The theory is not being questioned due to any invalid impacts, but rather because the explosion of data has created an enormous demand for data storage and processing power.

In particular, with the foundation from the previous era, the World Wide Web advanced into the new century at a rapid pace. Web 2.0 facilitated social connectivity worldwide, and Web 3.0 offers even more variety in access points. This means that people can not only reach others from everywhere around the globe but also from anywhere, such as on a computer, cell phone, or tablet. Many companies foresaw the rising demand and innovated great services to satisfy these new ideas, such as buying books from the comfort of one's couch, connecting with friends thousands of miles away, or advertising houses to tourists with whom they have no connection. To support these advanced technologies, distributed computing, cloud services, and diverse data sources had to be established in the back end, in other words, data decentralization came up introducing new complexity and challenges in managing and orchestrating data workflow. As a result, companies and their best scientists had to come up with better tools and strategies to maintain phenomenal growth or risk being left behind.

Among all, Apache Airflow, created by Airbnb, is one of the most popular workflow management platforms in the data engineering and data science communities. The vast majority of companies from various industries are adopting and utilizing Apache Airflow to improve their workflows, demonstrating the power of this software platform.

C. Objectives

In this paper, we will explore and showcase the benefits of Apache Airflow through a simple project aimed at predicting potential stocks using data from Yahoo Finance. The process begins with downloading data from the financial platform's API using a Python library called Yfinance. The data is expected to be downloaded daily to track the performance of the stocks and saved in two separate databases on a local machine in PostgreSQL. Several machine learning models will be trained and then utilized to flag any upward patterns found from each day of data.

II. Apache Airflow

A. Overview of Workflow Management Systems

In general, Workflow Management is a systematic approach to optimizing and ensuring a series of steps required to complete a specific project are consistently and efficiently executed (Pratt, 2022). In a big data analysis project, the pipeline starts with collecting a large amount of data from one or multiple sources. Depending on the scale and requirements of the project, the raw data will need to be transformed and processed before being saved to a new database or added to an existing one on a local machine or on a cloud storage service, ready to be used in machine learning models or business intelligence applications.

B. Apache Airflow

In such complex pipelines, Apache Airflow stands out as a powerful system that provides dynamic and robust scheduling. Additionally, it offers a user-friendly platform for monitoring and management. It also remains highly up-to-date for being open-source and written in Python which is one of the most popular programming languages.

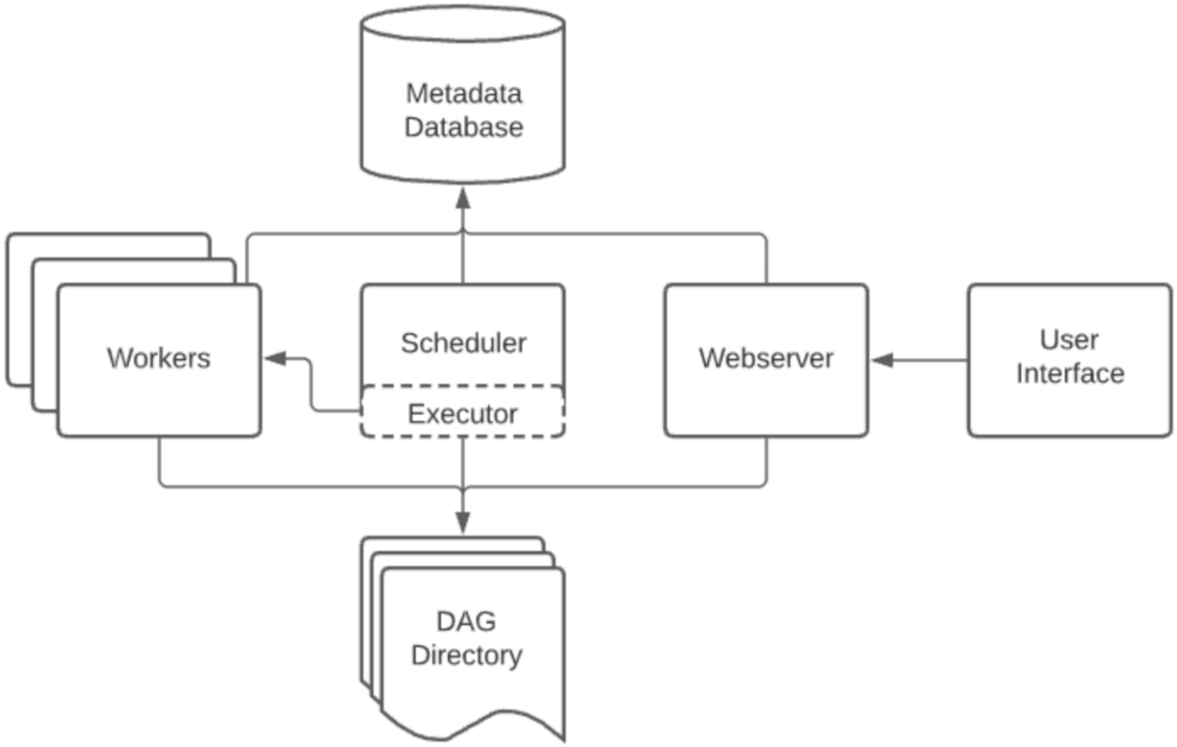


Image 1.1 – Core components in Apache Airflow infrastructure (Docs, Airflow components, 2024)

In Apache Airflow, the infrastructure includes four core components that work in a system to orchestrate and manage workflows seamlessly. First and foremost is the **Webserver**, a Flask server powered by Gunicorn, which serves as the gateway to the Airflow UI. Here, users can visualize, monitor, and manage their workflows with ease. The **Scheduler**, on the other hand, operates behind the scenes as a daemon, responsible for the critical task of job scheduling. As a multi-threaded Python process, it determines the timing and execution location for each task within the workflow. Supporting these functionalities is the **Database**, serving as the repository for all metadata related to Directed Acyclic Graphs (DAGs) and tasks. This could be any local DBMS including Postgres, MySQL, SQLite, etc., or cloud environment but it requires extra setups and third-party software. Lastly, the **Executor** serves as the engine for executing tasks within the scheduler, ensuring seamless workflow execution under diverse conditions.

Although Airflow has SequentialExecutor configured by default which runs locally and does not support parallelism, other executors support parallel task operation, thus, the platform may have multiple **workers**. As a result, efficiency issues may arise when a task instance occupies a worker while awaiting their upstream tasks or some certain conditions, leading to the worker becoming idle and wasting resources. In such cases, Airflow offers Deferrable operators and triggers that can free up the workers for other tasks. Together, these components form the backbone of Airflow's infrastructure, enabling efficient and reliable workflow management for organizations of all sizes. (Docs, Airflow components, 2024)

The heart of the workflow orchestration is encapsulated in an Airflow Directed Acyclic Graph (DAG). A DAG serves as a dynamic pipeline where users can define a set of tasks that are organized in an order. Because a DAG is always a directed graph, there can be as many tasks as needed but each task must have at least one connection to either the task in front of it (upstream) or after it (downstream) (Airflow, 2024). Of course, tasks must not point back to themselves to avoid creating infinite loops. See the example in Image 1.2.

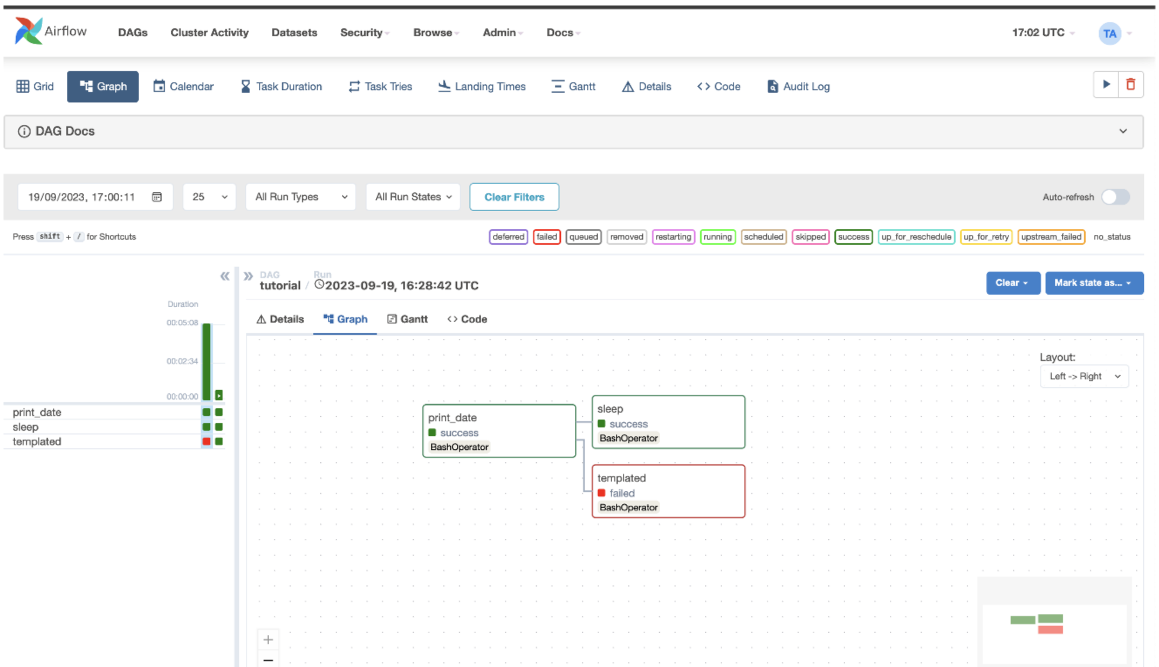


Image 1.2 – The status of the "demo" DAG in the web interface (Airflow, 2024)

All tasks in DAG can be visualized in Airflow UI similarly to a tree structure with each task as a node and the dependencies between them are vertices (Docs, Introduction to Airflow DAGs, 2024). What to do in each task at a specific point is determined by what operator it calls. There are various Airflow operators such as *PythonOperator*, *Bash Operator*, *KubenetesPodOperator*, *SnowflakeOperator*, etc. This means tasks in a DAG can be executed on multiple platforms and work with data from multiple sources.

Furthermore, a DAG can have its tasks run on different servers depending on which executor the user uses. To facilitate that, engineers came up with *Xcom*, which is a special way to push and pull for small amounts of data between tasks. For large data, *Xcom* can carry the link to the cloud bucket (e.g. S3 link) where the full data is saved. (Airflow, 2024)

C. Limitations

One of the most misleading myths about Airflow is that it is a tool to streamline data processing or simply handle data streams on a large scale (Yousry, 2023). This misconception is believed to originate from its capability to execute sophisticated multi-step pipelines. While the DAG structure holds all the power of Airflow, it also harbors an Achilles heel. The lack of a powerful mechanism to transfer data between tasks makes the workflow system weak against large-scale data. Users must save the data either on a local machine or in cloud storage, as Xcom was never designed to handle anything as large as a data frame. In other words, tasks in DAGs are completely isolated, and streaming data between them is extremely costly. Thus, unless the data has already been processed or there will be another tool to integrate, Airflow should not be the first option for streamlining data processing.

D. Conclusion

In conclusion, Apache Airflow emerges as a robust and versatile solution for orchestrating complex workflows, offering dynamic scheduling, user-friendly monitoring, and management capabilities. In the next section, we will introduce a practical application of Apache Airflow by showcasing a simple stock prediction pipeline, and highlighting its effectiveness in real-world scenarios.

III. Demo: Stock Prediction

A. Overview of the demo

To explore and demonstrate Apache Airflow platform, we implemented a stock prediction pipeline. The pipeline included 6 steps:

- Step 1 - Query stock information:
  - Download stock data (APPL) from Yahoo Finance using the *yfinance* library.
  - Connect to a local DBMS (Postgres).
  - Create a table if not exist.
  - Clean the data by replacing null values and formatting data types for further processing.
  - Saves the cleaned data into a PostgreSQL database table named *stock_info*.
  - Push the new batch of data to XCom.
- Task 2 – Subset the data for ML models:
  - Pull the raw stock data from XCom.
  - Filter to include only necessary columns ['Stock_Name', 'Date', 'Open', 'Close'.
  - Save the data back into the PostgreSQL database table named *processed_stock_data*.
- Sensor - Check if models exist in the S3 bucket using Minio.
  - Check if the trained models exist via Minio using S3KeySensor.
  - If not, wait until the models are saved in the S3 bucket.
- Tasks 3, 4, 5 - Load models and run predictions.
  - Models: Linear regression, logistic regression, LSTM.
  - Apply each model to the new data and return predictions.
  - Results are in binary values – 1 for good performance and 0 for bad performance.
- Task 6 - Make investment decisions

- Combine results from tasks 3, 4, and 5.
- Decide outcome based on whether at least 2 out of 3 models return good performance.

B. Dataset

Our program will download the specified stocks dataset from Yahoo Finance. Here is an overview of the dataset:

- **stock_name**: The name of the stock on Yahoo Finance. For example, "apple" corresponds to the stock symbol "AAPL", and "Nvidia" corresponds to "NVDA".
- **date**: The date of the stock data entry.
- **open**: The opening price of the stock, which is the price at which the stock begins trading when the market opens for the day.
- **high**: The highest price of the stock during the trading day.
- **low**: The lowest price of the stock during the trading day.
- **close**: The closing price of the stock, which is the final price at which the stock is traded on the given trading day.
- **volume**: The total number of shares or contracts traded for the stock on that day.
- **adj_close**: The adjusted closing price of the stock, which is the final price adjusted before the next trading day.
- **short_ma**: The short-term moving average of a financial metric.
- **long_ma**: The long-term moving average of a financial metric.

This dataset provides valuable information about stock prices and trading volumes over time, along with additional calculated metrics such as moving averages.

stock_name	date	year	month	day	weekday	week_number	year_week	open	high	low	close	volume	adj_close	return	short_ma	long_ma
AAPL	2023-04-05	2023	4	5	Wednesday	14	2023-14	164.74	165.05	161.8	163.76	51511700	162.89	0	162.89	162.89
AAPL	2023-04-06	2023	4	6	Thursday	14	2023-14	162.43	164.96	162	164.66	45390100	163.79	0	163.79	163.79
AAPL	2023-04-10	2023	4	10	Monday	15	2023-15	161.42	162.03	160.08	162.03	47716900	161.17	0	161.17	161.17
AAPL	2023-04-11	2023	4	11	Tuesday	15	2023-15	162.35	162.36	160.51	160.8	47644200	159.95	0	159.95	159.95
AAPL	2023-04-12	2023	4	12	Wednesday	15	2023-15	161.22	162.06	159.78	160.1	50133100	159.25	0	159.25	159.25
AAPL	2023-04-13	2023	4	13	Thursday	15	2023-15	161.63	165.8	161.42	165.56	68445600	164.68	0	164.68	164.68
AAPL	2023-04-14	2023	4	14	Friday	15	2023-15	164.59	166.32	163.82	165.21	49386500	164.33	0	164.33	164.33
AAPL	2023-04-17	2023	4	17	Monday	16	2023-16	165.09	165.39	164.03	165.23	41516200	164.35	0	164.35	164.35
AAPL	2023-04-18	2023	4	18	Tuesday	16	2023-16	166.1	167.41	165.65	166.47	49923000	165.59	0	165.59	165.59
AAPL	2023-04-19	2023	4	19	Wednesday	16	2023-16	165.8	168.16	165.54	167.63	47720200	166.74	0	166.74	166.74
AAPL	2023-04-20	2023	4	20	Thursday	16	2023-16	166.09	167.87	165.56	166.65	52456400	165.77	0	165.77	165.77
AAPL	2023-04-21	2023	4	21	Friday	16	2023-16	165.05	166.45	164.49	165.02	58337300	164.14	0	164.14	164.14
AAPL	2023-04-24	2023	4	24	Monday	17	2023-17	165	165.6	163.89	165.33	41949600	164.45	0	164.45	164.45
AAPL	2023-04-25	2023	4	25	Tuesday	17	2023-17	165.19	166.31	163.73	163.77	48714100	162.9	0	162.9	162.9
AAPL	2023-04-26	2023	4	26	Wednesday	17	2023-17	163.06	165.28	162.8	163.76	45498800	162.89	0	162.89	162.89
AAPL	2023-04-27	2023	4	27	Thursday	17	2023-17	165.19	168.56	165.19	168.41	64902300	167.52	0	167.52	167.52
AAPL	2023-04-28	2023	4	28	Friday	17	2023-17	168.49	169.85	167.88	169.68	55209200	168.78	0	168.78	168.78
AAPL	2023-05-01	2023	5	1	Monday	18	2023-18	169.28	170.45	168.64	169.59	52472900	168.69	0	168.69	168.69
AAPL	2023-05-02	2023	5	2	Tuesday	18	2023-18	170.09	170.35	167.54	168.54	48425700	167.65	0	167.65	167.65
AAPL	2023-05-03	2023	5	3	Wednesday	18	2023-18	169.5	170.92	167.16	167.45	65136000	166.56	0	166.56	166.56
AAPL	2023-05-04	2023	5	4	Thursday	18	2023-18	164.89	167.04	164.31	165.79	81235400	164.91	0	164.91	164.91
AAPL	2023-05-05	2023	5	5	Friday	18	2023-18	170.98	174.3	170.76	173.57	113316400	172.65	0	172.65	172.65

C. Environment Setups

The demo is implemented in macOS. We used Apache Airflow version 2.8.3 in Docker 4.27.2 and programmed in Python 3.8.

First, in docker-cmopose.yaml replace image and add ports: 5432:5432 to map the container's internal port 5432 to port 5432 on the host machine.

```yaml
# image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.8.3}
image: ${AIRFLOW_IMAGE_NAME:-extending_airflow:latest}

  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    ports:
      - 5432:5432
    healthcheck:
      test: [ "CMD", "pg_isready", "-U", "airflow" ]
      interval: 10s
      retries: 5
      start_period: 5s
    restart: always
```

Then, we builds a Docker image named **extending_airflow** and tags it as **latest**. It searches for a file named **Dockerfile** in the current directory and follows the instructions provided in that file to construct the image. Each command in the **Dockerfile**, such as **FROM**, **COPY**, **RUN**, etc., is executed to create the image with the required environment and application. Once the build is complete, the resulting image is labeled with the specified tag, which can be **latest** to indicate it as the most recent version.

dockerfile

```
FROM apache/airflow:2.8.3
COPY requirements.txt /requirements.txt
RUN pip install --user --upgrade pip
RUN pip install --no-cache-dir --user -r /requirements.txt
```

requirements.txt

```
pandas==2.0.3
yfinance==0.2.31
psycopg2-binary
scikit-learn==1.3.2
numpy==1.23.5
keras==2.13.1
tensorflow==2.13.1
```

```
docker build . --tag extending_airflow:latest
```

These screenshots show docker environment has been set up and connected to Airflow.

1.2.3

```
docker compose up -d
```

```
(venv) caozhen@dongdongs-3rd-MacBook-Pro AirFlowDemo % docker compose up -d
WARN[0000] The "AIRFLOW_UID" variable is not set. Defaulting to a blank string.
WARN[0000] The "AIRFLOW_UID" variable is not set. Defaulting to a blank string.
[+] Running 8/9
 ⠏ Network airflowdemo_default                  Created
 ✔ Container airflowdemo-postgres-1             Healthy
 ✔ Container airflowdemo-redis-1                Healthy
 ✔ Container airflowdemo-airflow-init-1         Exited
 ✔ Container airflowdemo-airflow-worker1-1      Started
 ✔ Container airflowdemo-airflow-worker2-1      Started
 ✔ Container airflowdemo-airflow-scheduler-1    Started
 ✔ Container airflowdemo-airflow-webserver-1    Started
 ✔ Container airflowdemo-airflow-triggerer-1    Started
```

```
docker ps
```

```
(venv) caozhen@dongdongs-3rd-MacBook-Pro AirFlowDemo % docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS                                 PORTS                    NAMES
01297688cf25   extending_airflow:latest  "/usr/bin/dumb-init …"   2 minutes ago   Up About a minute (health: starting)   0.0.0.0:8080->8080/tcp   airflowdemo-airflow-webserver-1
bc0d38bf32f0   extending_airflow:latest  "/usr/bin/dumb-init …"   2 minutes ago   Up About a minute (healthy)            8080/tcp                 airflowdemo-airflow-scheduler-1
b38c6a975bd4   extending_airflow:latest  "/usr/bin/dumb-init …"   2 minutes ago   Up About a minute (healthy)            8080/tcp                 airflowdemo-airflow-worker1-1
2523d3bc0580   extending_airflow:latest  "/usr/bin/dumb-init …"   2 minutes ago   Up About a minute (healthy)            8080/tcp                 airflowdemo-airflow-triggerer-1
9ed808849eb3   extending_airflow:latest  "/usr/bin/dumb-init …"   2 minutes ago   Up About a minute (healthy)            8080/tcp                 airflowdemo-airflow-worker2-1
ef112170e4bc   postgres:13               "docker-entrypoint.s…"   2 minutes ago   Up About a minute (healthy)            0.0.0.0:5432->5432/tcp   airflowdemo-postgres-1
78cefcea0715   redis:latest              "docker-entrypoint.s…"   2 minutes ago   Up About a minute (healthy)            6379/tcp                 airflowdemo-redis-1
```

We also establish the connection to Minio to call our pre-trained models that were stored in S3 bucket.

1.3 minio

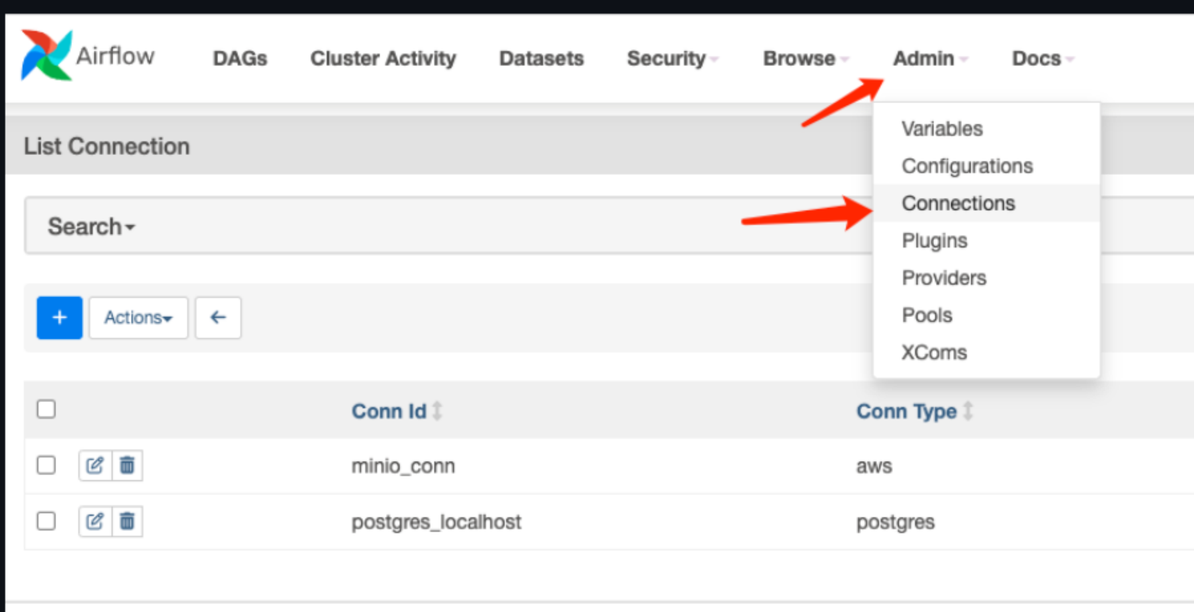
```
mkdir -p ~/minio/data

docker run \
   -p 9000:9000 \
   -p 9001:9001 \
   --name minio \
   -v ~/minio/data:/data \
   -e "MINIO_ROOT_USER=ROOTNAME" \
   -e "MINIO_ROOT_PASSWORD=CHANGEME123" \
   quay.io/minio/minio server /data --console-address ":9001"
```

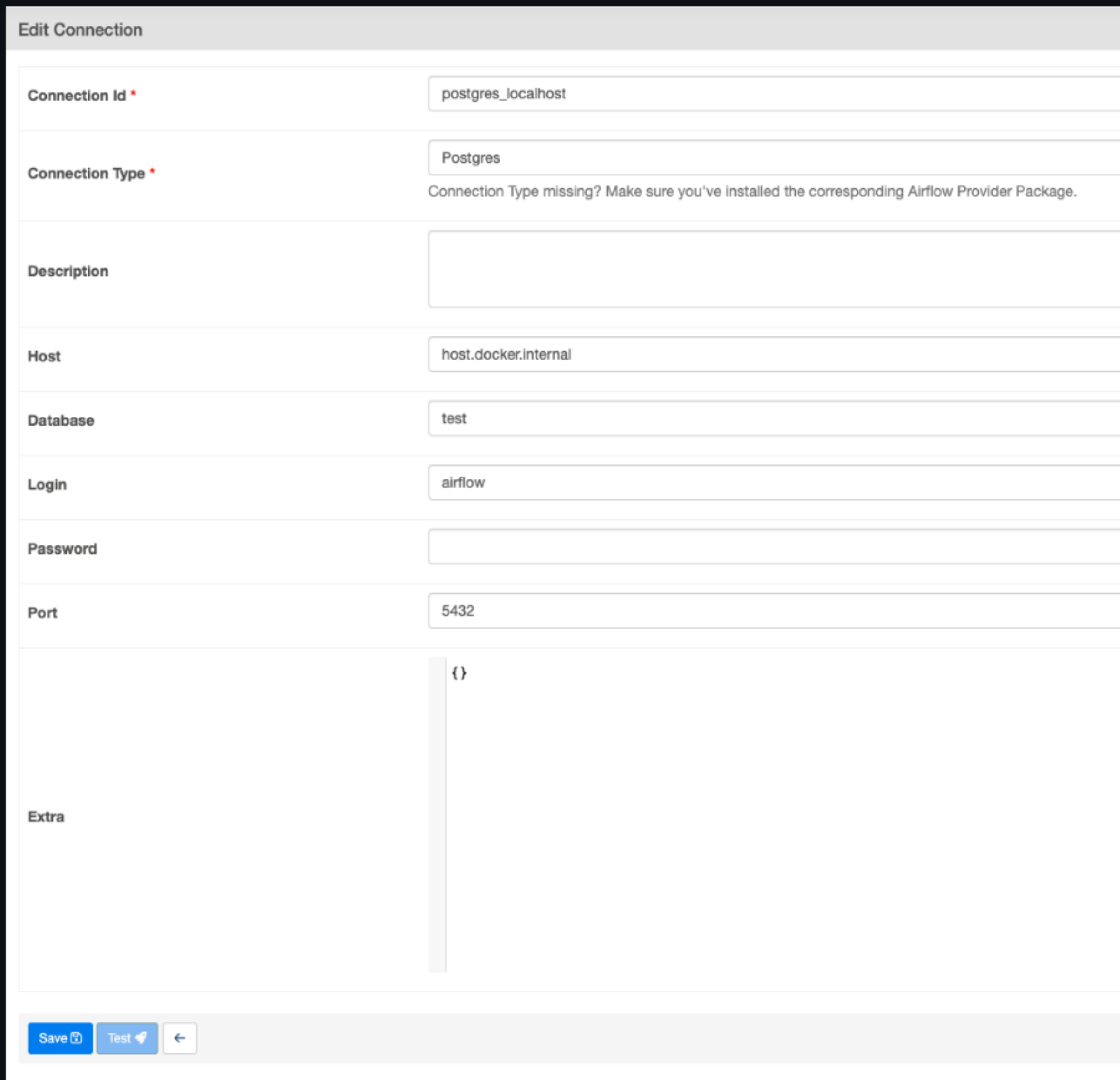
```
(venv) caozhen@dongdongs-3rd-MacBook-Pro AirFlowDemo % docker ps
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS                    PORTS                              NAMES
01297688cf25   extending_airflow:latest    "/usr/bin/dumb-init …"   8 minutes ago    Up 7 minutes (healthy)    0.0.0.0:8080->8080/tcp             airflowdemo-airflow-webserver-1
bc0d30bf32f0   extending_airflow:latest    "/usr/bin/dumb-init …"   8 minutes ago    Up 7 minutes (healthy)    8080/tcp                           airflowdemo-airflow-scheduler-1
b38c6a975bd4   extending_airflow:latest    "/usr/bin/dumb-init …"   8 minutes ago    Up 7 minutes (healthy)    8080/tcp                           airflowdemo-airflow-worker1-1
2523d3bc0580   extending_airflow:latest    "/usr/bin/dumb-init …"   8 minutes ago    Up 7 minutes (healthy)    8080/tcp                           airflowdemo-airflow-triggerer-1
9ed800849eb3   extending_airflow:latest    "/usr/bin/dumb-init …"   8 minutes ago    Up 7 minutes (healthy)    8080/tcp                           airflowdemo-airflow-worker2-1
ef112170e4bc   postgres:13                 "docker-entrypoint.s…"   8 minutes ago    Up 8 minutes (healthy)    0.0.0.0:5432->5432/tcp             airflowdemo-postgres-1
78cefcea0715   redis:latest                "docker-entrypoint.s…"   8 minutes ago    Up 8 minutes (healthy)    6379/tcp                           airflowdemo-redis-1
dc1edcb72343   quay.io/minio/minio         "/usr/bin/docker-ent…"   6 days ago       Up 10 minutes             0.0.0.0:9000-9001->9000-9001/tcp   minio
```

In Airflow UI, we connect to PostgreSQL DBMS in Admin tab.

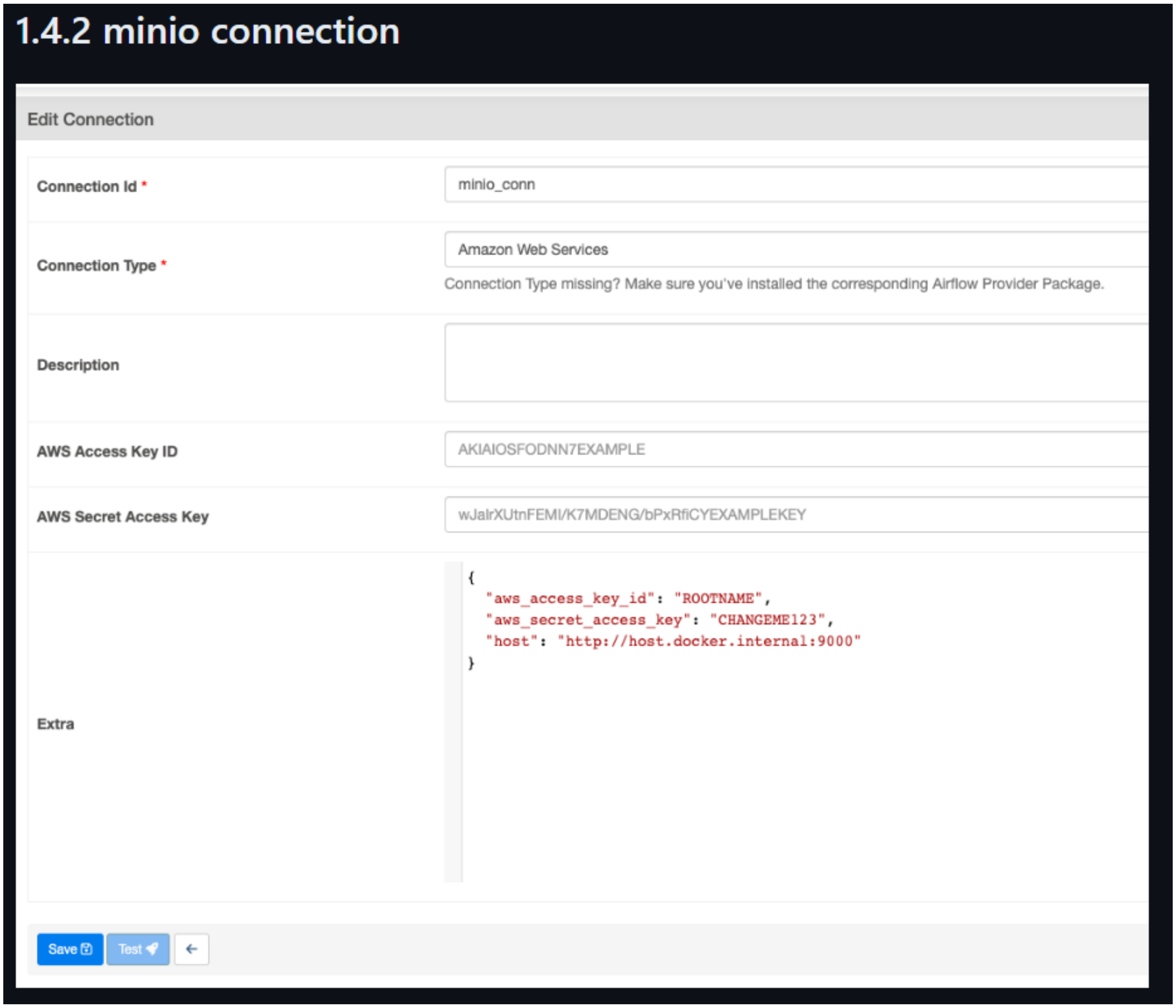
1.4 create connection using Airflow UI



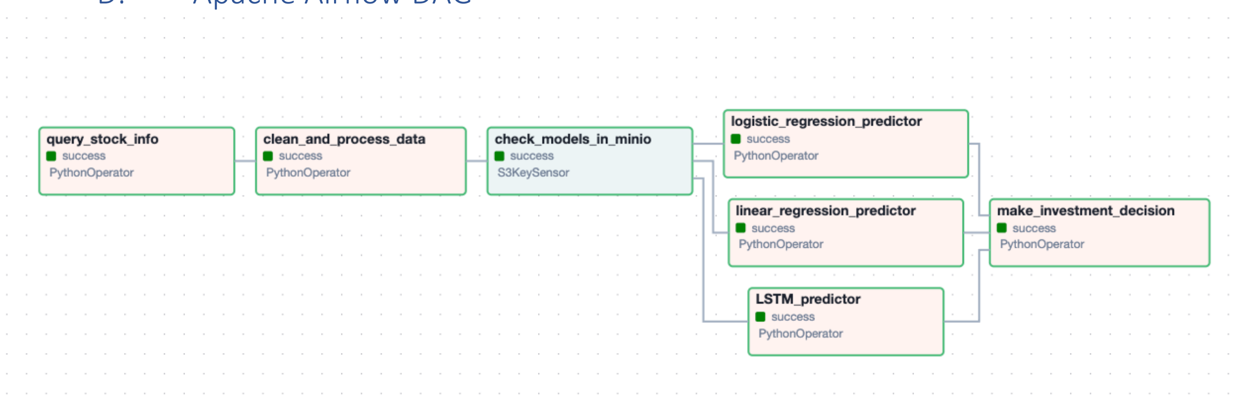
1.4.1 postgres connection



Establishing connection to Minio.



D. Apache Airflow DAG

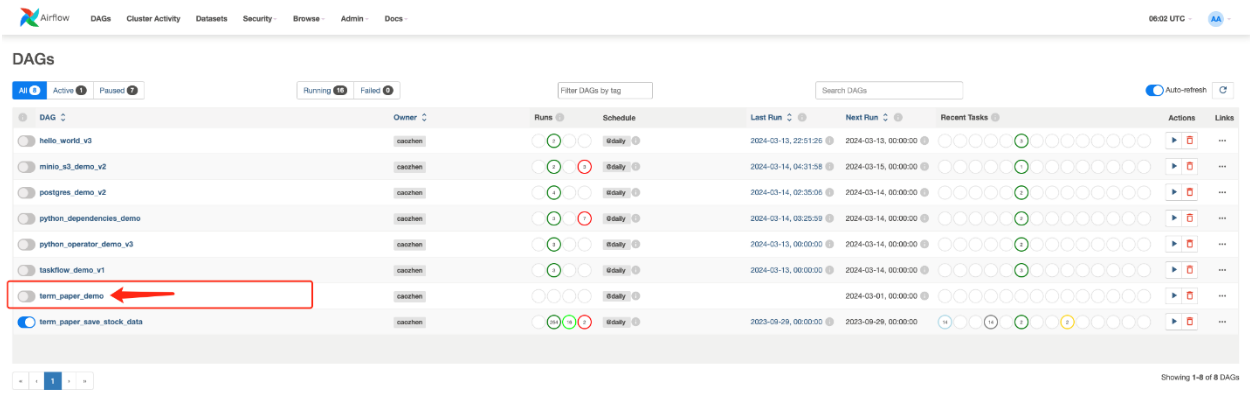


Declaring task dependencies:

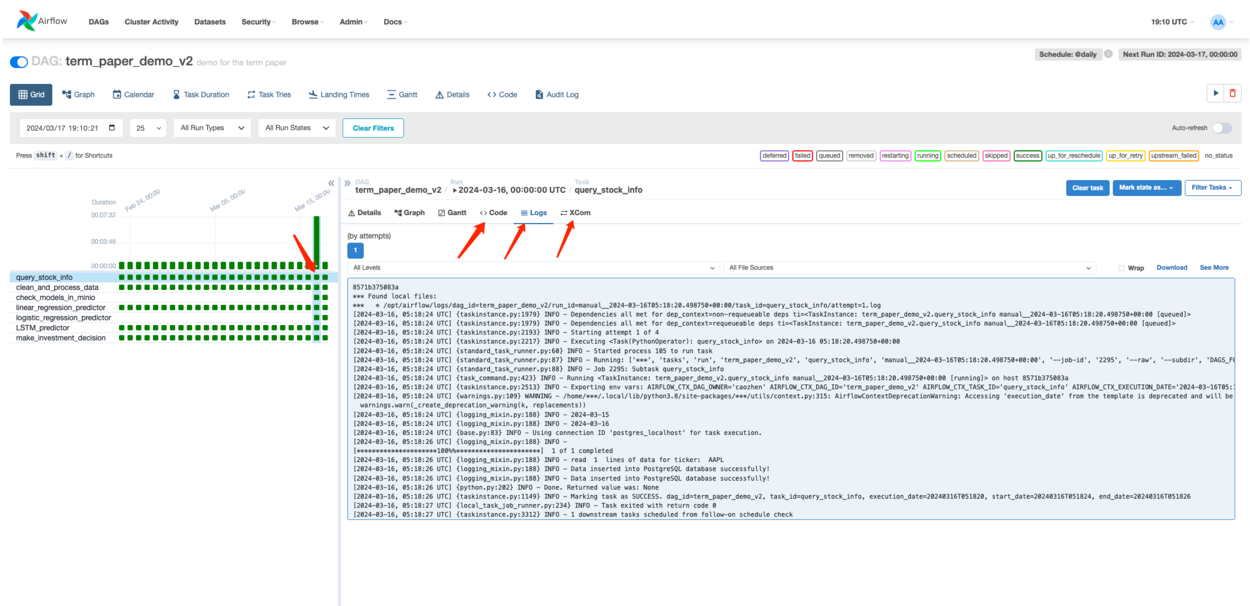
task1 >> task2 >> model_sensor >> [task3, task4, task5] >> task6

E. Running DAG in Airflow UI

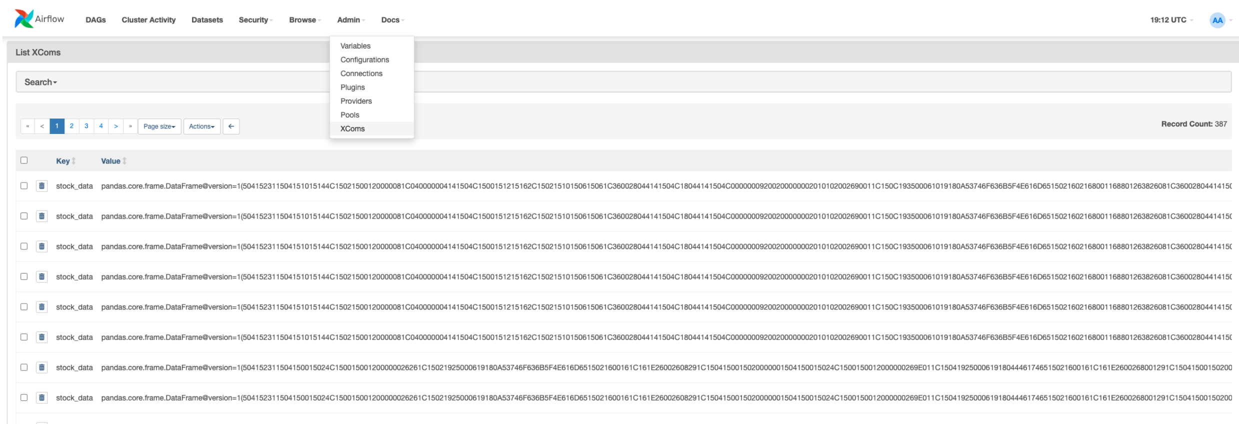
DAG found in directory and was ready to start.



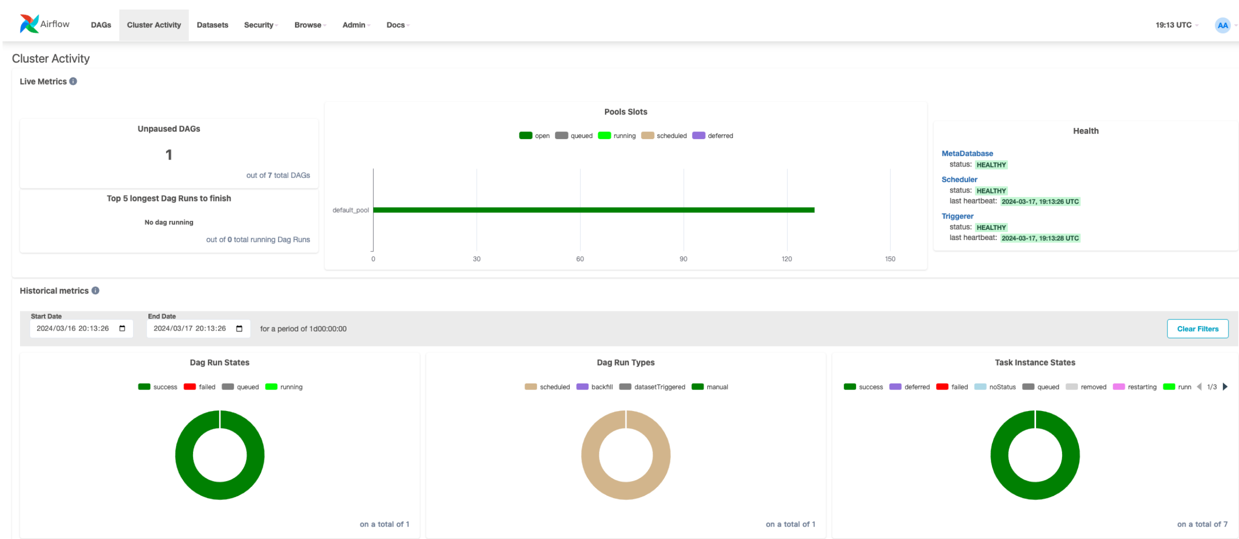
DAG ran successfully in Grid view with logs.



Immediate results of the raw stock data saved in XCom.



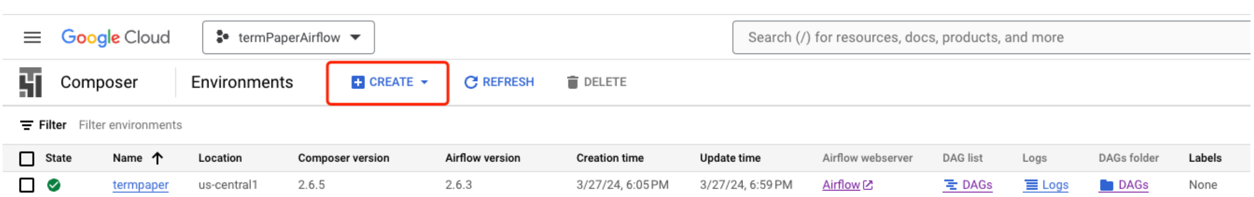
Monitor the workflow in the Cluster activity tab.



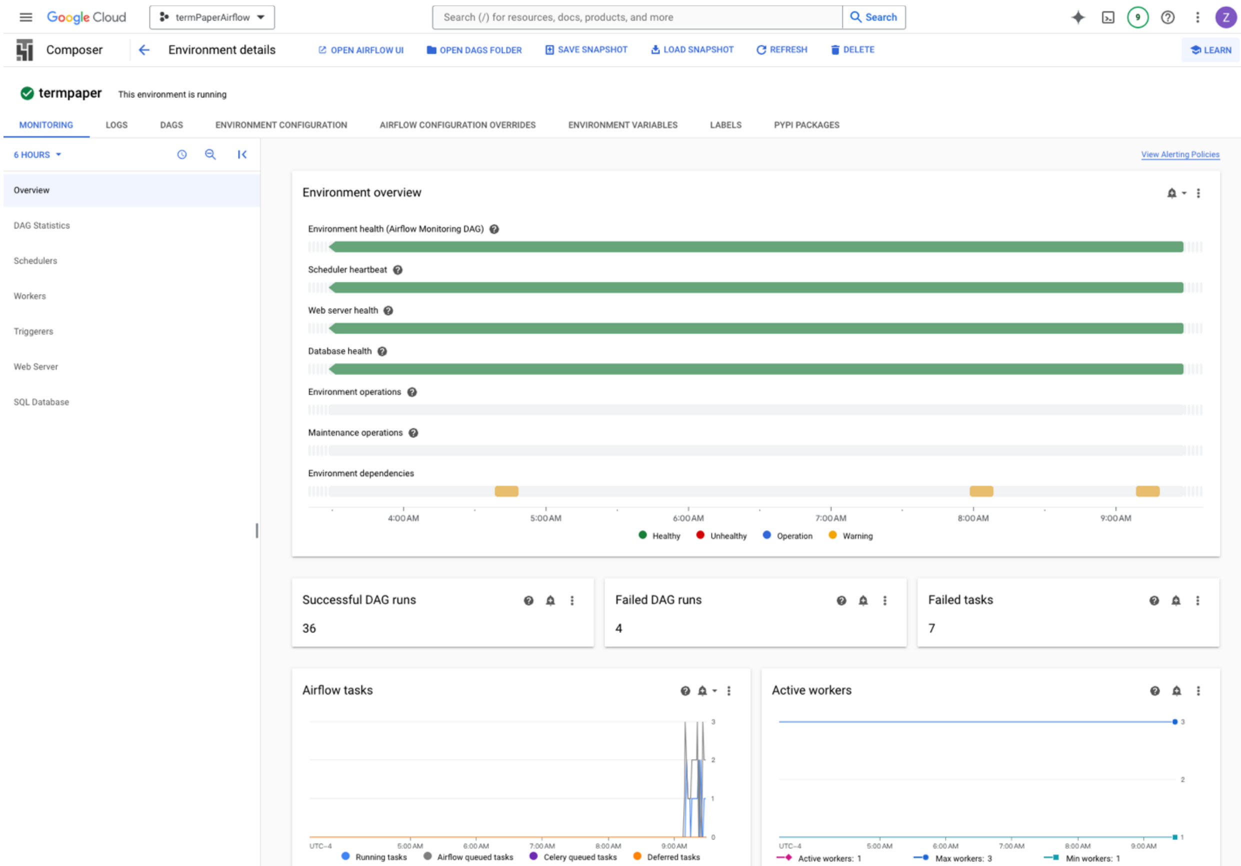
F. Apache Airflow in Google Cloud

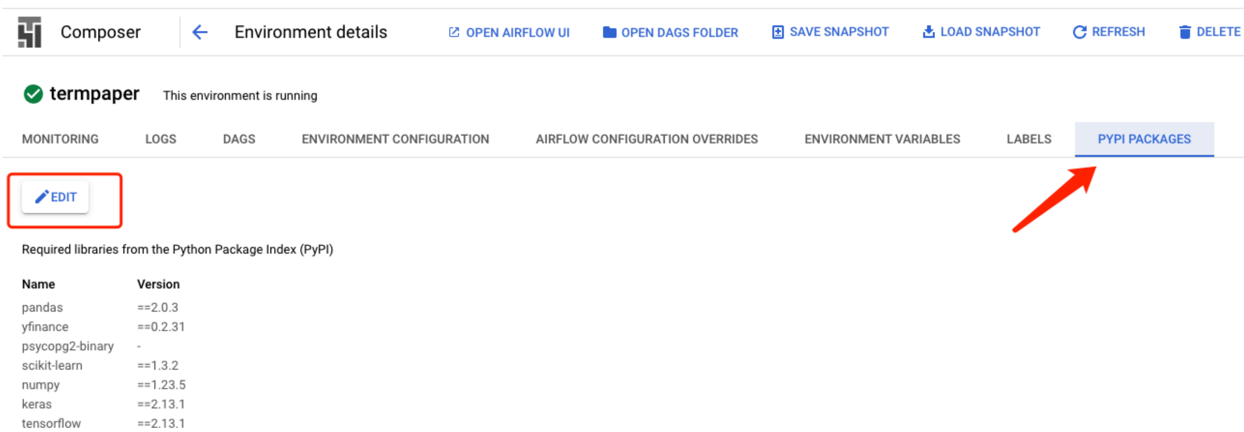
I. Create composer (airflow on GC)

https://console.cloud.google.com/composer/



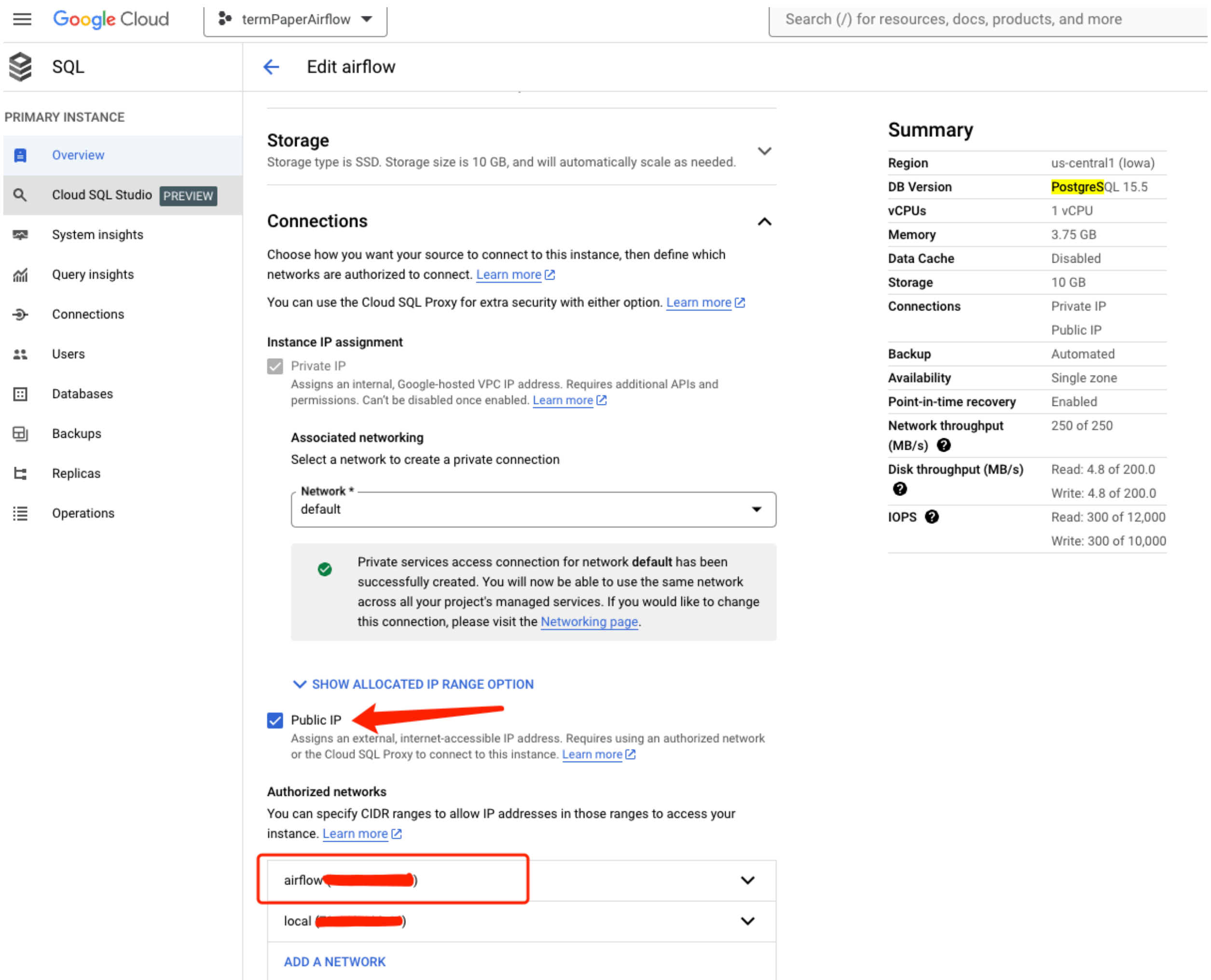
- Create composer 2.



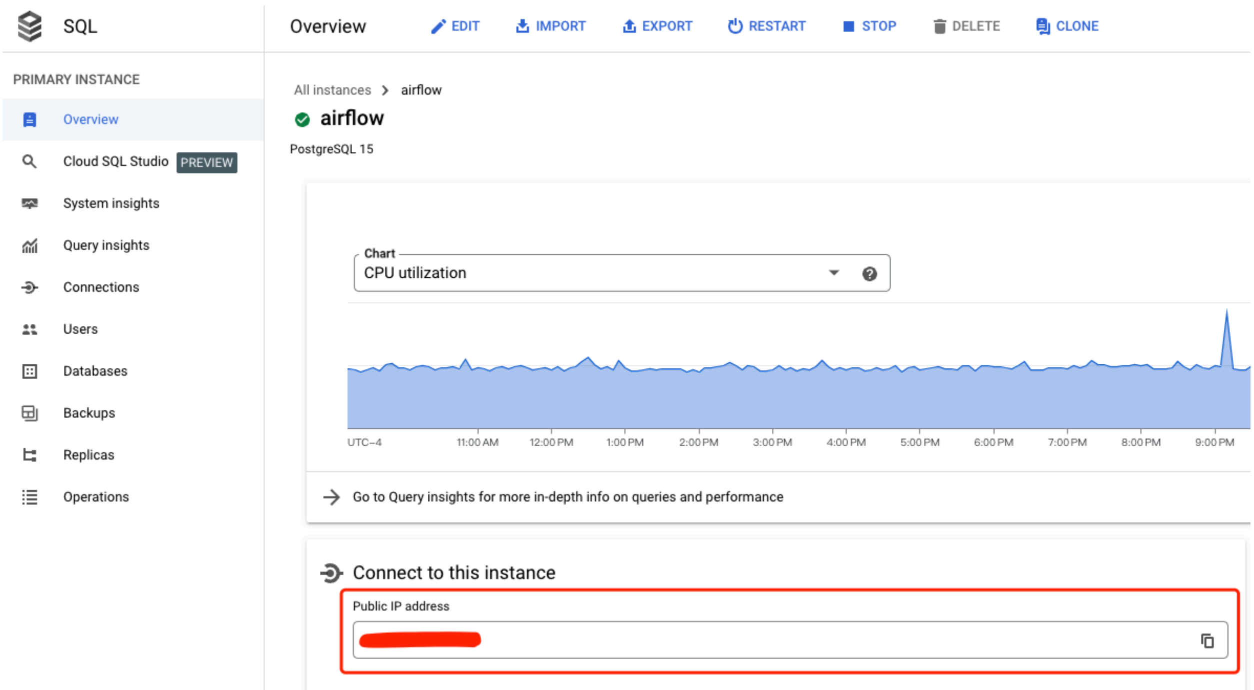


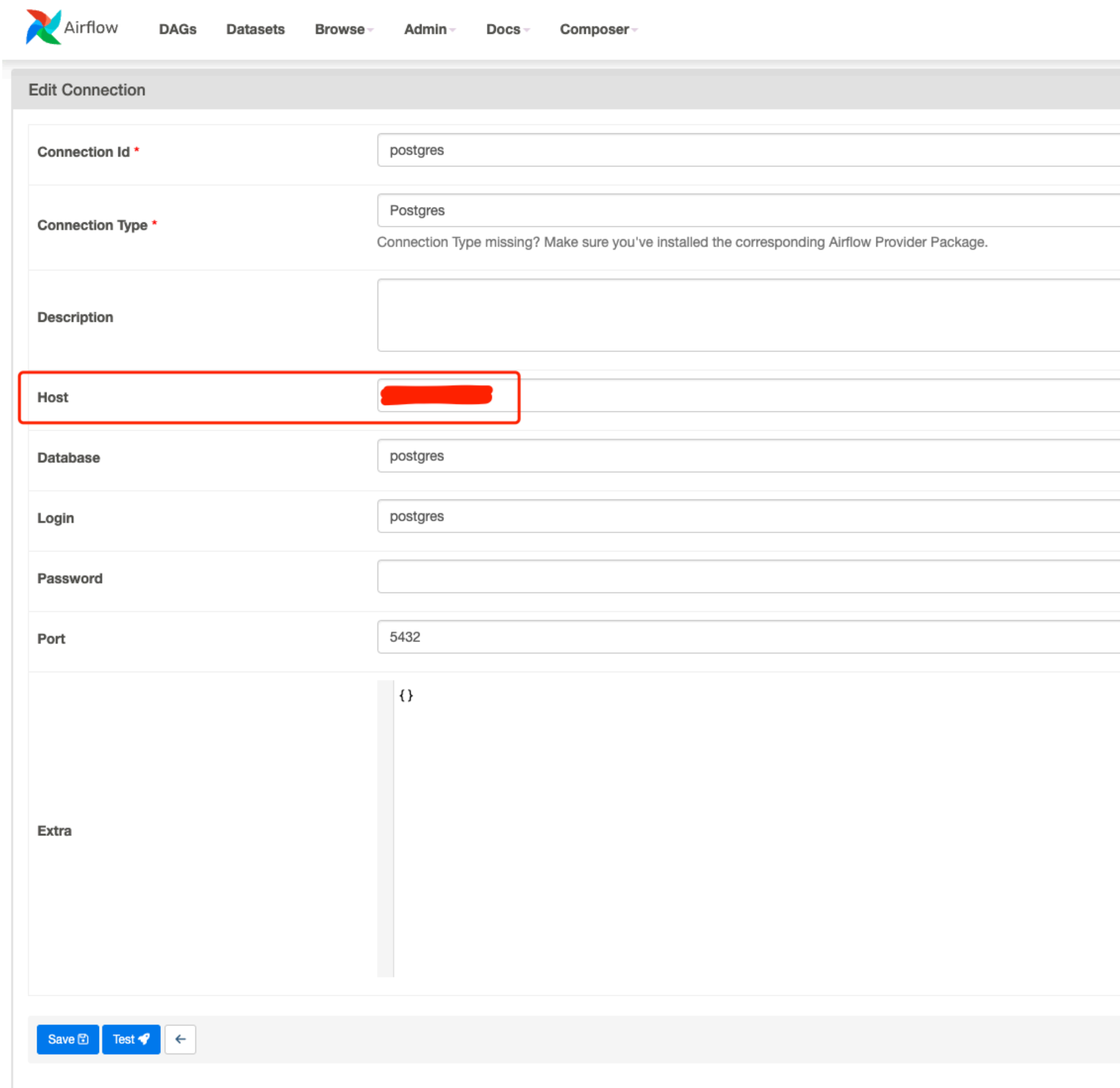
- You can add python packages using PYPI. Then composer will download for you automatically.

II. Create Postgres database



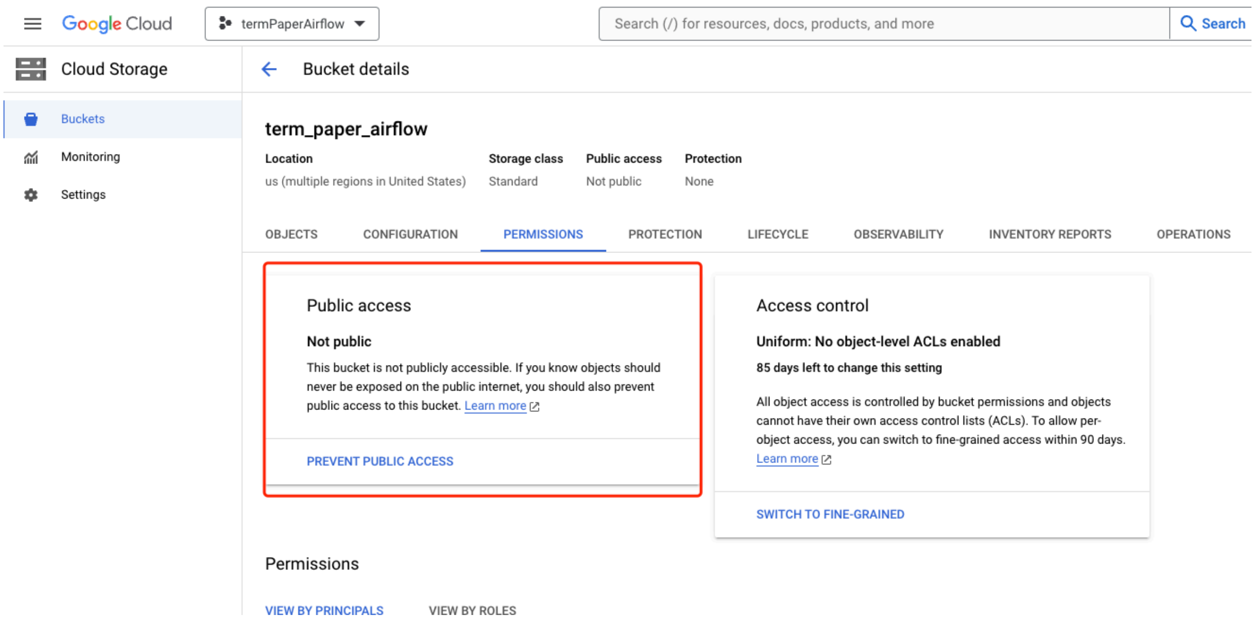
You need to add your airflow IP to authorized networks here.


SQL
Overview
EDIT
IMPORT
EXPORT
RESTART
STOP
DELETE
CLONE
PRIMARY INSTANCE
Overview
Cloud SQL Studio
PREVIEW
System insights
Query insights
Connections
Users
Databases
Backups
Replicas
Operations
All instances
airflow
airflow
PostgreSQL 15
Chart
CPU utilization
UTC-4
11:00 AM
12:00 PM
1:00 PM
2:00 PM
3:00 PM
4:00 PM
5:00 PM
6:00 PM
7:00 PM
8:00 PM
9:00 PM
Go to Query insights for more in-depth info on queries and performance
Connect to this instance
Public IP address



And create connection in Airflow UI using the Postgres database public IP address.

III. Google Cloud Storage



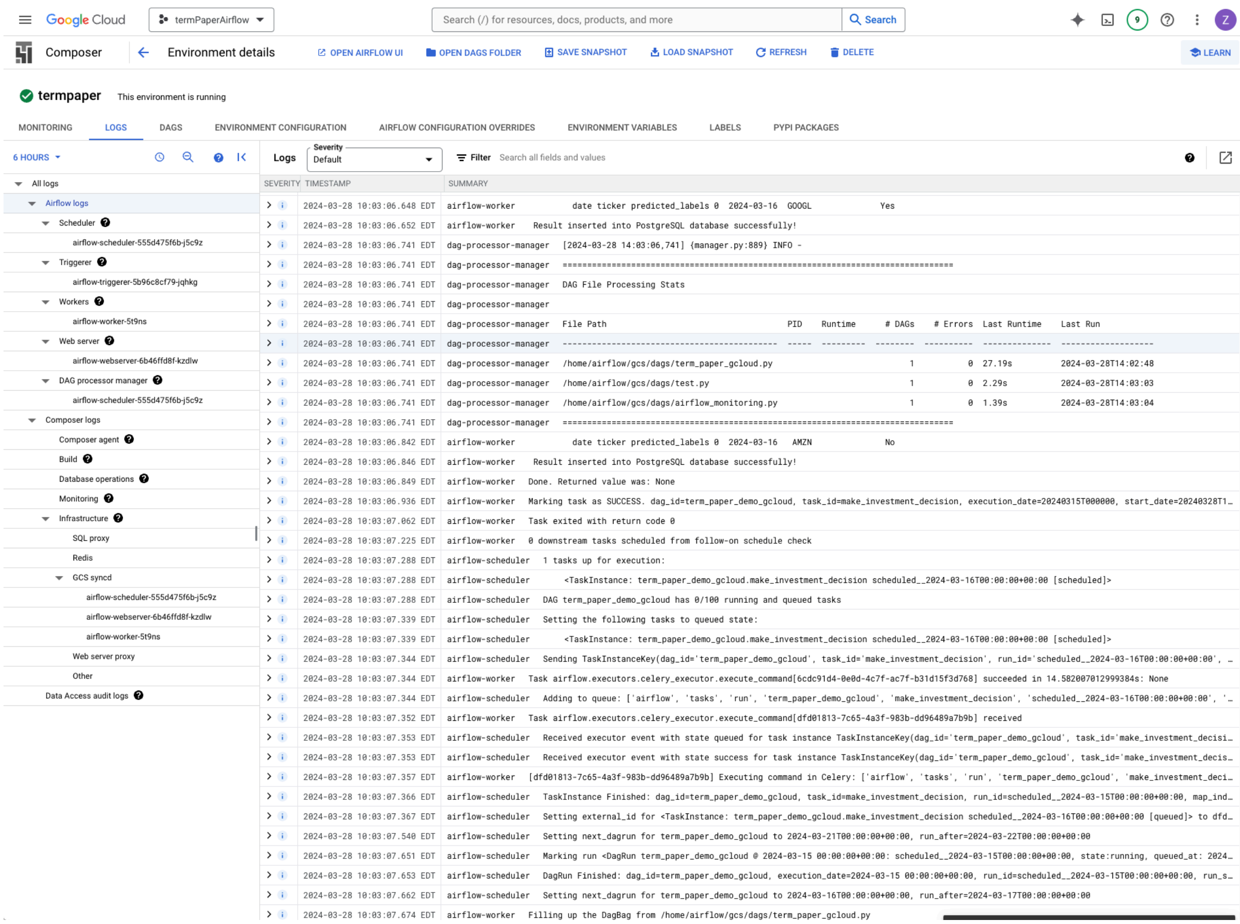
Here allow public access to the bucket and save the models in the bucket.

```
gcs_hook = GoogleCloudStorageHook(google_cloud_conn_id)
# First download .h5 model file from bucket and then load model
model_stream = gcs_hook.download(bucket_name=bucket_name, object_name=model_key,
filename=f'/tmp/{model_key}')
```

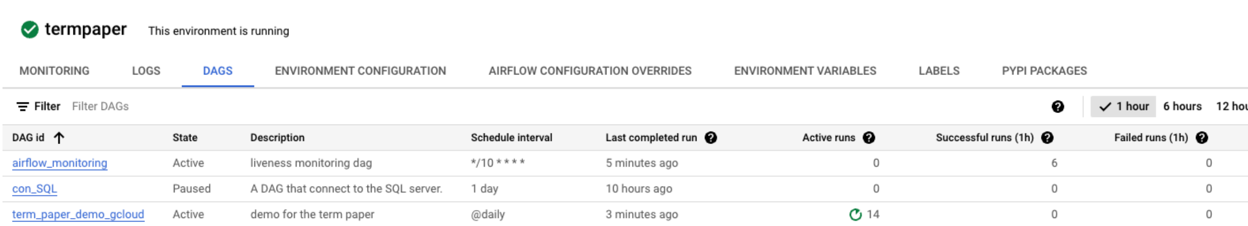
Because we use GoogleCloudStorageHook to download the model.

IV. Composer Dashboard

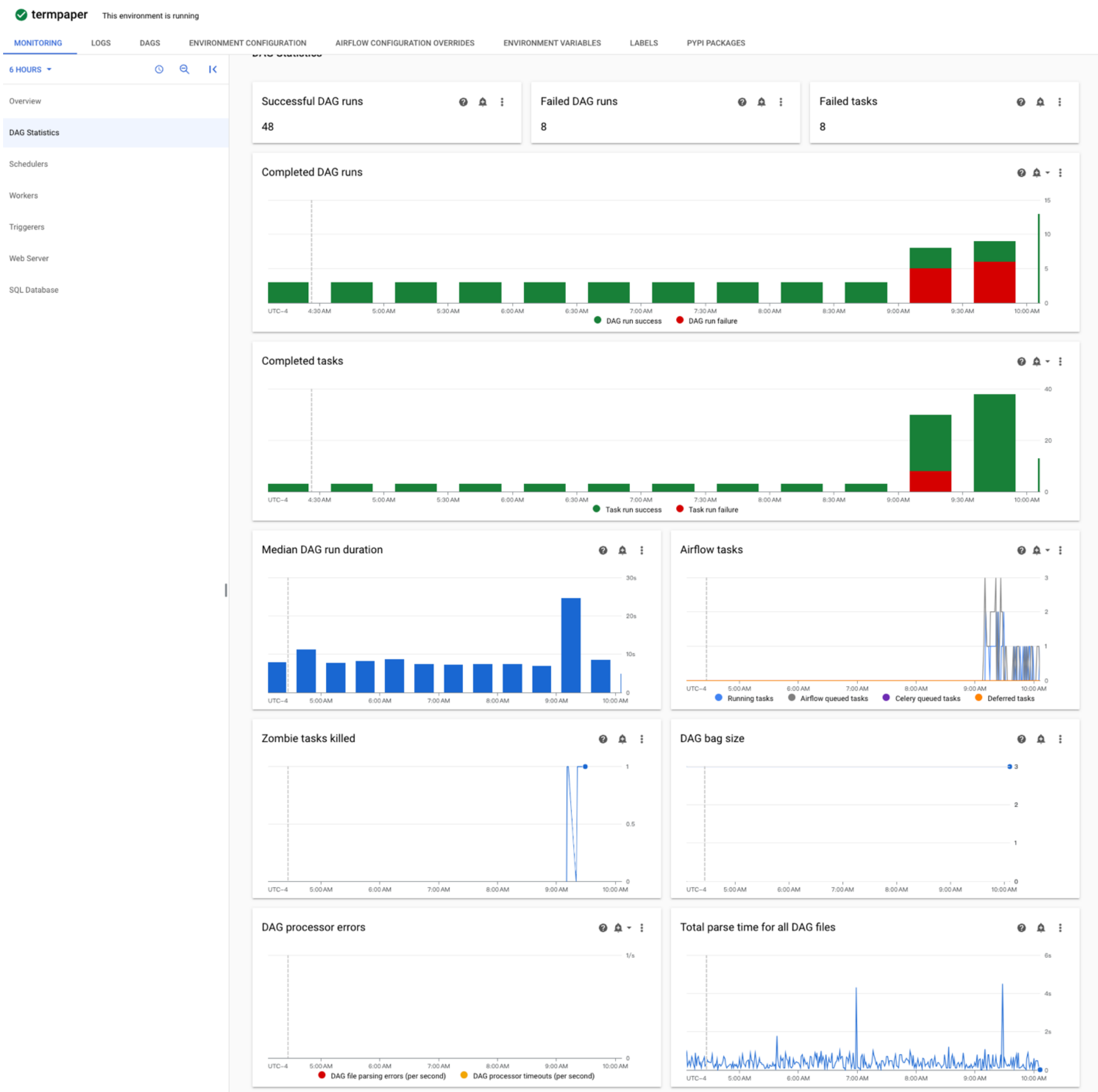
Logs:



Dags:



Monitor:


termpaper
This environment is running
MONITORING
LOGS
DAGS
ENVIRONMENT CONFIGURATION
AIRFLOW CONFIGURATION OVERRIDES
ENVIRONMENT VARIABLES
LABELS
PYPI PACKAGES
6 HOURS
Overview
DAG Statistics
Schedulers
Workers
Triggerers
Web Server
SQL Database
Successful DAG runs
48
Failed DAG runs
8
Failed tasks
8
Completed DAG runs
DAG run success
DAG run failure
Completed tasks
Task run success
Task run failure
Median DAG run duration
Airflow tasks
Running tasks
Airflow queued tasks
Celery queued tasks
Deferred tasks
Zombie tasks killed
DAG bag size
DAG processor errors
DAG file parsing errors (per second)
DAG processor timeouts (per second)
Total parse time for all DAG files

G. Results

Predictions are stored back into the PostgreSQL database. We can query the database to retrieve these predictions and compare them with the actual results. Below is an example query.

SELECT psd.date,
psd.stock_name,
psd.open,
psd.close,
sp.prediction,
(psd.close - psd.open) AS earning
FROM processed_stock_data psd
JOIN stock_prediction sp ON psd.date = sp.date AND psd.stock_name = sp.stock_name
WHERE psd.stock_name = 'NVDA';

date	stock_name	open	close	prediction	earning
2024-03-08	NVDA	951.38	875.28	Yes	-76.10000000000002
2024-03-11	NVDA	864.29	857.74	Yes	-6.5499999999999545
2024-03-12	NVDA	880.49	919.13	No	38.639999999999986
2024-03-13	NVDA	910.55	908.88	No	-1.669999999999959
2024-03-14	NVDA	895.77	879.44	No	-16.329999999999927
2024-03-15	NVDA	869.3	878.37	Yes	9.07000000000005
2024-03-18	NVDA	903.88	884.55	Yes	-19.33000000000004
2024-03-19	NVDA	867	893.98	No	26.980000000000018
2024-03-20	NVDA	897.97	903.72	No	5.75

date	stock_name	open	close	prediction	earning
2024-03-08	GOOGL	134.21	135.41	Yes	1.1999999999999886
2024-03-11	GOOGL	136.13	137.67	Yes	1.539999999999992
2024-03-12	GOOGL	137.03	138.5	Yes	1.4699999999999989
2024-03-13	GOOGL	139	139.79	Yes	0.789999999999992
2024-03-14	GOOGL	141.19	143.1	Yes	1.9099999999999966
2024-03-15	GOOGL	142.5	141.18	Yes	-1.3199999999999932
2024-03-18	GOOGL	148.61	147.68	Yes	-0.9300000000000068
2024-03-19	GOOGL	148.16	147.03	Yes	-1.1299999999999955
2024-03-20	GOOGL	148	148.74	Yes	0.7400000000000091

date	stock_name	open	close	prediction	earning
2024-03-08	AMZN	176.44	175.35	Yes	-1.0900000000000034
2024-03-11	AMZN	174.31	171.96	No	-2.3499999999999943
2024-03-12	AMZN	173.5	175.39	No	1.8899999999999864
2024-03-13	AMZN	175.9	176.56	Yes	0.6599999999999966
2024-03-14	AMZN	177.69	178.75	No	1.0600000000000023
2024-03-15	AMZN	176.64	174.42	Yes	-2.219999999999999
2024-03-18	AMZN	175.8	174.48	No	-1.3200000000000216
2024-03-19	AMZN	174.22	175.9	Yes	1.6800000000000068
2024-03-20	AMZN	176.14	178.15	Yes	2.0100000000000193

The rectangles highlight correct predictions when the labels "No" indicate negative earnings, and "Yes" indicate positive earnings.

IV. Conclusion

In conclusion, this paper has provided an in-depth exploration of Apache Airflow's capabilities in managing complex workflows, with a specific focus on scheduling Apache Spark jobs for stock prediction tasks. By leveraging Airflow's Directed Acyclic Graphs (DAGs), we demonstrated how workflow optimization and error handling can be effectively implemented. Through the case study of stock prediction, we showcased the seamless integration of data collection, preprocessing, model deployment, and investment decision-making within Airflow. Through this term paper, we explored the potential of Airflow in enhancing workflow efficiency and reliability in big data analytics scenarios. As companies continue in advancing through the challenges of handling large volumes of data, Apache Airflow proves to be a robust solution for orchestrating complex workflows and driving insights from data analysis tasks.

V. References

Airflow, A. (2024, March 17). *https://airflow.apache.org/*. Retrieved from https://airflow.apache.org/: https://airflow.apache.org/

Buyya, R., & Rodriguez, M. A. (2017, 1 1). Scientific Workflow Management System for Clouds. In I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, & B. Maxim, *Software Architecture for Big Data and the Cloud* (pp. 367-387). Morgan Kaufmann. Retrieved from ScienceDirect: https://www.sciencedirect.com/topics/computer-science/workflow-management-system

Docs, A. (2024, 03 19). *Airflow components*. Retrieved from Astronomer Docs: https://docs.astronomer.io/learn/airflow-components

Docs, A. (2024, 03 19). *Introduction to Airflow DAGs*. Retrieved from Astronomer Docs: https://docs.astronomer.io/learn/dags#:~:text=is%20a%20DAG%3F-,%E2%80%8B,tasks%20in%20the%20Airflow%20UI.

Ibrahim, S., He, B., & Zhou, A. C. (2016). Chapter 18 - eScience and Big Data Workflows in Clouds: A Taxonomy and Survey. In R. Buyya, A. V. Dastjerdi, & R. N. Calheiros, *Big Data* (pp. 431-455). Melbourne, Autralia: Morgan Kaufmann.

Nematpour, M. (2020, 11 26). *Using Airflow to Schedule Spark Jobs*. Retrieved from medium: https://medium.com/swlh/using-airflow-to-schedule-spark-jobs-811becf3a960

Poola, D., Salehi, M. A., Ramamohanarao, K., & Buyya, R. (2017). A Taxonomy and Survey of Fault-Tolerant Workflow Management Systems in Cloud and Distributed Computing Environments. In I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, & B. Maxim, *Software Architecture for Big Data and the Cloud* (pp. 285-320). Morgan Kaufmann.

Pratt, M. K. (2022, April 01). *What is workflow management?* Retrieved from TechTarget: https://www.techtarget.com/searchcio/definition/workflow-management#:~:text=Workflow%20management%20is%20the%20discipline,completed%20correctly%2C%20consistently%20and%20efficiently.

restack.io. (2024, 02 24). *restack.io*. Retrieved from Apache Airflow and Spark Integration: https://www.restack.io/docs/airflow-knowledge-apache-spark-example-jobs

Tardi, C. (2024, February 11). *What Is Moore's Law and Is It Still True?* Retrieved from Investopedia: https://www.investopedia.com/terms/m/mooreslaw.asp

Yousry, A. (2023, May 25). *Don’t Use Apache Airflow in That Way*. Retrieved from Medium: https://medium.com/illumination/what-apache-airflow-is-not-e9dc9722500b#:~:text=Airflow%20is%20not%20a%20Data%20Streaming%20Tool&text=While%20Airflow%20can%20be%20used,a%20database%20or%20file%20system.