

Airflow Providers and Integrations:

Analytics:

- Apache Spark
- Apache Superset
- Google Analytics
- Adobe Analytics
- Microsoft Power BI
- Databricks

Preprocessing:

- Apache NiFi
- Qubole
- Apache Kafka
- Talend
- Apache Flink
- Apache Storm

Reports:

- Tableau
- Microsoft Power BI
- QlikView
- Google Data Studio
- SAP Crystal Reports

ETL:

- Apache NiFi
- Apache Beam
- Talend
- Glue
- Azure Data Factory
- Informatica

Storage:

- Amazon S3
- Google Cloud Storage
- Microsoft Azure Blob Storage
- Apache Cassandra
- MongoDB
- Snowflake

Real-time:

- Apache Flink
- Apache Storm
- Apache Kafka
- Apache Ignite
- Google Cloud Dataflow

Workflow:

- Apache Airflow
- Apache Oozie
- Apache Beam
- Luigi
- Azkaban

ML:

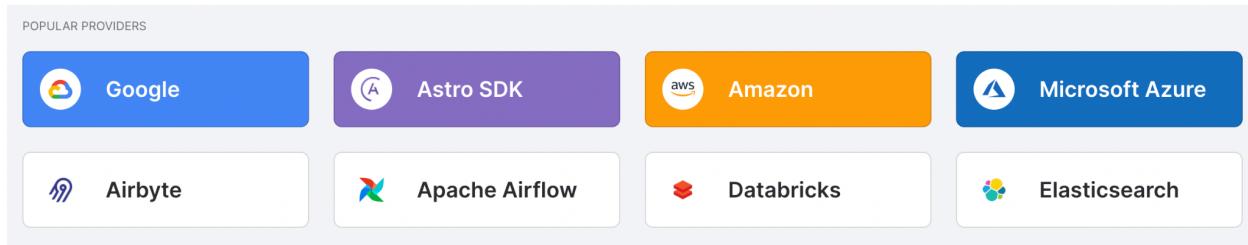
- TensorFlow
- PyTorch
- Scikit-learn
- Apache Spark MLlib
- Microsoft Azure ML

Monitor:

- Prometheus
- Grafana
- Nagios
- Datadog
- New Relic

Cloud Integrations

Popular Providers :



More Providers : <https://registry.astronomer.io/providers>

Google Cloud Service :

Composer - Cloud Airflow Architecture
Google Kubernetes Cluster

A screenshot of the Google Cloud Composer interface. The top navigation bar shows 'Google Cloud' and 'gcp-dataeng-demos'. The main area is titled 'Composer' and shows a list of environments. One environment, 'composer-demo', is selected and highlighted with a green checkmark. The table columns include Name, Location, Composer version, Airflow version, Creation time, Update time, Airflow webserver, DAG list, Logs, DAGs folder, and Labels. The 'DAGs' link under 'Airflow webserver' is underlined, indicating it's a live link.

A screenshot of the Google Cloud Storage interface. The top navigation bar shows 'Google Cloud' and 'gcp-dataeng-demos'. The main area is titled 'Bucket details' for 'us-central1-composer-demo-197652a3-bucket'. The 'OBJECTS' tab is selected. The table shows a single object named 'airflow_monitoring.py'. The table columns include Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, Encryption, and Retention expiry. The 'Show deleted data' checkbox is checked.

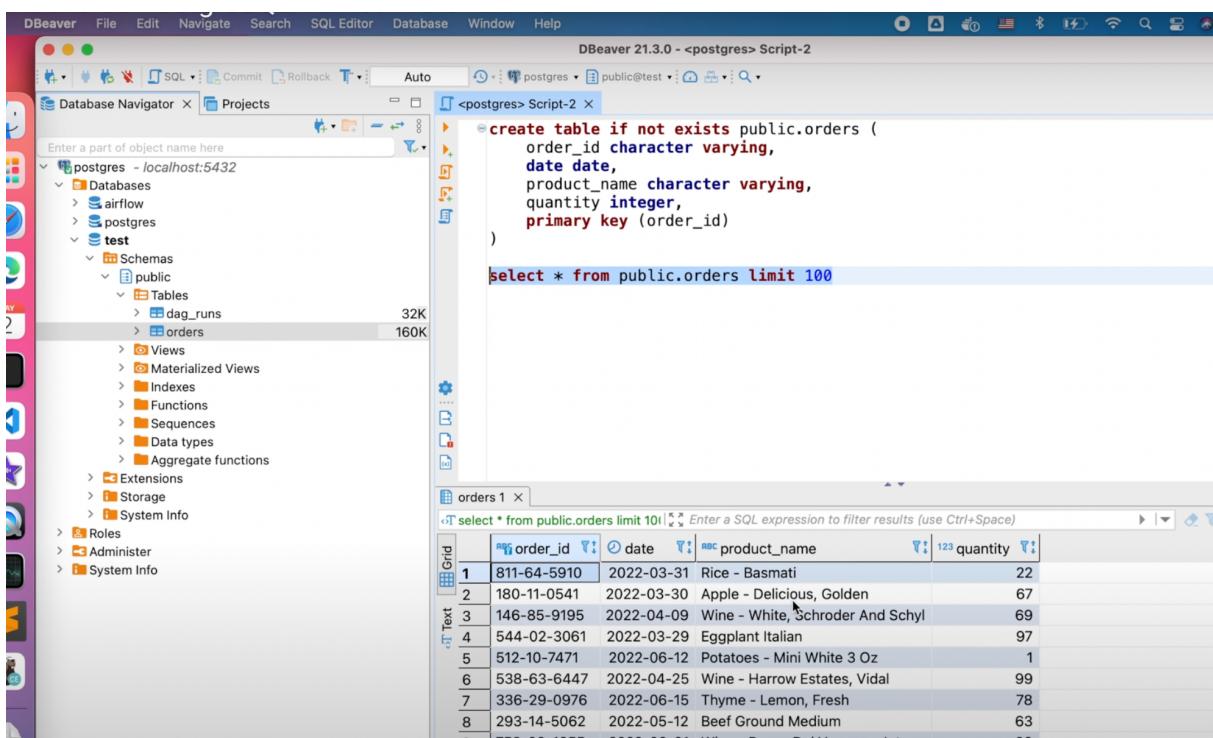
Amazon AWS :

EC2 Hosting Airflow

S3 Instances for Data Lake

Use Case :

Airflow Hooks S3 PostgreSQL [Postgres - Text - Upload to S3]



order_id	date	product_name	quantity
1	2022-03-31	Rice - Basmati	22
2	2022-03-30	Apple - Delicious, Golden	67
3	2022-04-09	Wine - White, Schrader And Schyl	69
4	2022-03-29	Eggplant Italian	97
5	2022-06-12	Potatoes - Mini White 3 Oz	1
6	2022-04-25	Wine - Harrow Estates, Vidal	99
7	2022-06-15	Thyme - Lemon, Fresh	78
8	2022-05-12	Beef Ground Medium	63
9	2022-06-01	Wine - Rosso Del Veronese Igt	82

```
load_file(self, filename: str, key: str, bucket_name: Optional[str] = None,  
replace: bool = False, encrypt: bool = False, gzip: bool = False, acl_policy:  
Optional[str] = None) [source]
```

Loads a local file to S3

Parameters

- **filename (str)** -- name of the file to load.
- **key (str)** -- S3 key that will point to the file
- **bucket_name (str)** -- Name of the bucket in which to store the file

Airflow AWS S3 Sensor Operator:

```
) as dag:  
    task1 = S3KeySensor(  
        task_id='sensor_minio_s3',  
        bucket_name='airflow',  
        bucket_key='data.csv',  
        aws_conn_id='minio_conn',  
        mode='poke',  
        poke_interval=5,  
        timeout=30  
)
```

Microsoft Azure :

Databricks

Steps:

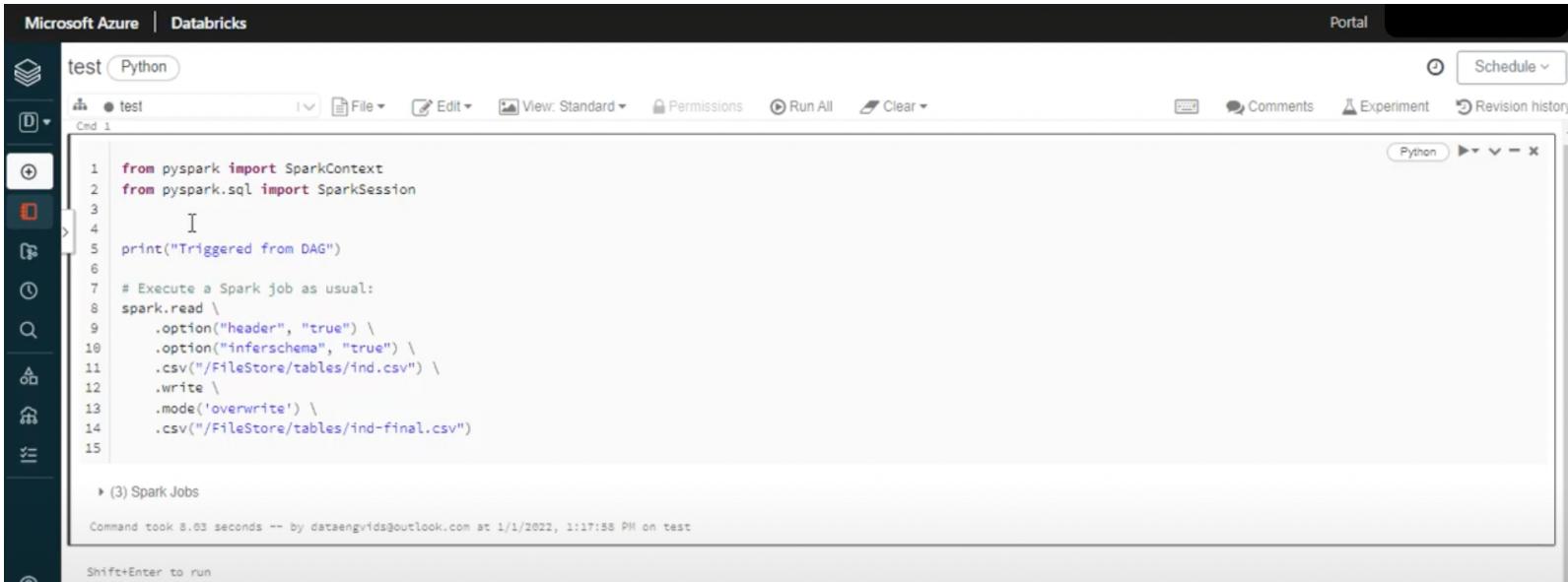
Create Connection

Import DatabricksSubmitRunOperator

Input Cluster details { name, versions, confs }

Notebook JSON { Cluster Details, Notebook Path }

Initiate Task { T_id, Conn_id, dag, notebook JSON }



The screenshot shows a Microsoft Azure Databricks notebook interface. The top navigation bar includes 'Microsoft Azure' and 'Databricks' on the left, and 'Portal' on the right. The main area is a code editor for a Python notebook named 'test'. The code cell contains the following Python script:

```
1 from pyspark import SparkContext
2 from pyspark.sql import SparkSession
3
4 I
5 print("Triggered from DAG")
6
7 # Execute a Spark job as usual:
8 spark.read \
9     .option("header", "true") \
10    .option("inverschema", "true") \
11    .csv("/FileStore/tables/ind.csv") \
12    .write \
13    .mode('overwrite') \
14    .csv("/FileStore/tables/ind-final.csv")
15
```

Below the code, a log message indicates the command took 8.63 seconds and was run by dataengvids@outlook.com on 1/1/2022 at 1:17:58 PM. A note at the bottom says 'Shift+Enter to run'.

More Providers and APIs :

1) Spotify Playlist :

Get Recently Played Spotify Playlist into DB:

https://github.com/karolina-sowinska/free-data-engineering-course-for-beginners/blob/master/dags/spotify_dag.py

2) Slack Notification:

Send Task Notification Via Slack

https://github.com/eywoon/airflow_demo/tree/master/dags

3) Podcast Summary:

A pipeline that will download podcast episodes and automatically transcribe them using speech recognition.

Store our results in a SQLite database that we can easily query.

https://github.com/dataquestio/project-walkthroughs/blob/master/podcast_summary/podcast_summary.py

4) Monitor Apache Airflow:

StatsD | Prometheus | Grafan

statsd-exporter: Service that aggregates and sends airflow statsd metrics to Prometheus metrics

Prometheus: Monitoring and alerting toolkit that runs Prometheus service that scrapes airflow metrics from statsd-exporter

Grafana: Runs Grafana instance for our final dashboard visualization and monitoring that allows you to query, visualize, alert on, and understand

<https://github.com/maxcotec/Apache-Airflow/tree/bc06530aa06e063c8c8c9f7ccd5fff8e17c16cc3>

5) Python Data Analysis ETL Pipeline :

https://github.com/hnawaz007/pythondataanalysis/blob/main/ETL%20Pipeline/automate_etl_with_airflow.py

Task :

Create a data processing pipeline that takes data from a CSV file, transforms it, and stores the results in a PostgreSQL database. The pipeline should run every day at midnight, and we want to be notified by email if there are any errors or if the pipeline fails.

Possible Steps:

1. A `file_sensor` task that waits for a CSV file to be uploaded to a specified directory.
2. A `data_processing` task that reads the CSV file, applies some transformations, and stores the results in a PostgreSQL database.
3. An `email_notification` task that sends an email notification if there are any errors or if the pipeline fails.
- 4.

Task 2 :

Create an Airflow data processing pipeline that extracts data from a MySQL database, cleans and transforms it using Apache Spark, loads it into a PostgreSQL database, triggers an ML model training script using Databricks, and sends a notification email with a report of the pipeline's status using the email operator.

Possible Steps:

1. Create a connection in Airflow for the MySQL and PostgreSQL databases, as well as the Databricks workspace.
2. Define a DAG with a schedule_interval of daily at midnight.
3. Create a task to extract data from the MySQL database using the MySqlOperator.
4. Create a Spark job using the SparkSubmitOperator to clean and transform the extracted data.
5. Create a task to load the transformed data into the PostgreSQL database using the PostgresOperator.
6. Create a task to trigger the ML model training script using the DatabricksSubmitRunOperator.
7. Define a task to send an email notification using the EmailOperator if there are any errors or if the pipeline fails.