Getting started with Apache Airflow

Building your first workflow

Ricardo Sueiras | Principal Open Source, Amazon Web Services

Workshop Agenda

From a local working directory

```
git clone https://github.com/094459/fossasia-airflow.git
```

Open up the README where the workshop will begin.

We will come back to this presentation during the workshop

Apache Airflow

orchestration

/ɔːkɪˈstreɪʃ(ə)n/

noun

1. the arrangement or scoring of music for orchestral performance.

2. the planning or coordination of the elements of a situation to produce a desired effect

workflow

/ˈwəːkfləʊ/

noun

1. the sequence of steps (tasks) involved in moving from the beginning to the end of a working process

DAG

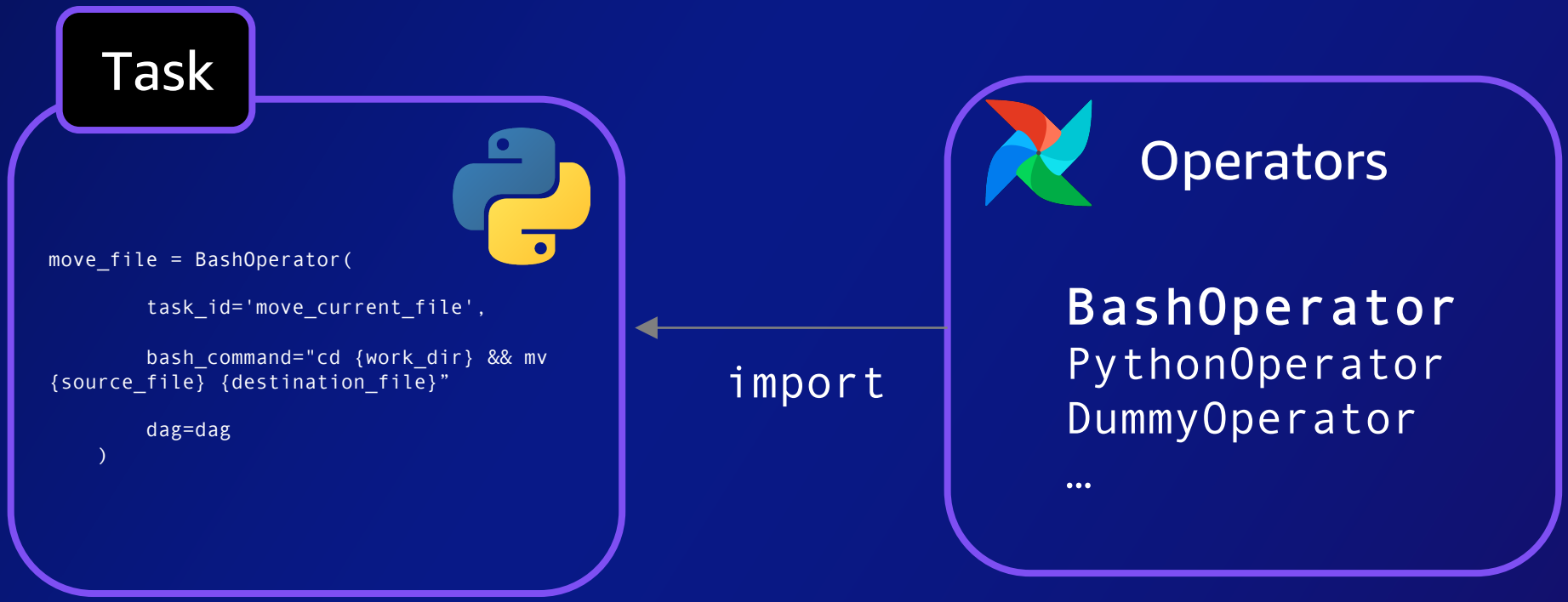
```
from airflow import DAG

DAG_ID = ‘daily_dw_ingest’

dag = DAG(
    dag_id=DAG_ID,
    default_args=default_args,
    description='First Apache Airflow DAG',
    schedule_interval=None,
    start_date=days_ago(2),
    tags=[bigdataeurope','demo'],
)
```

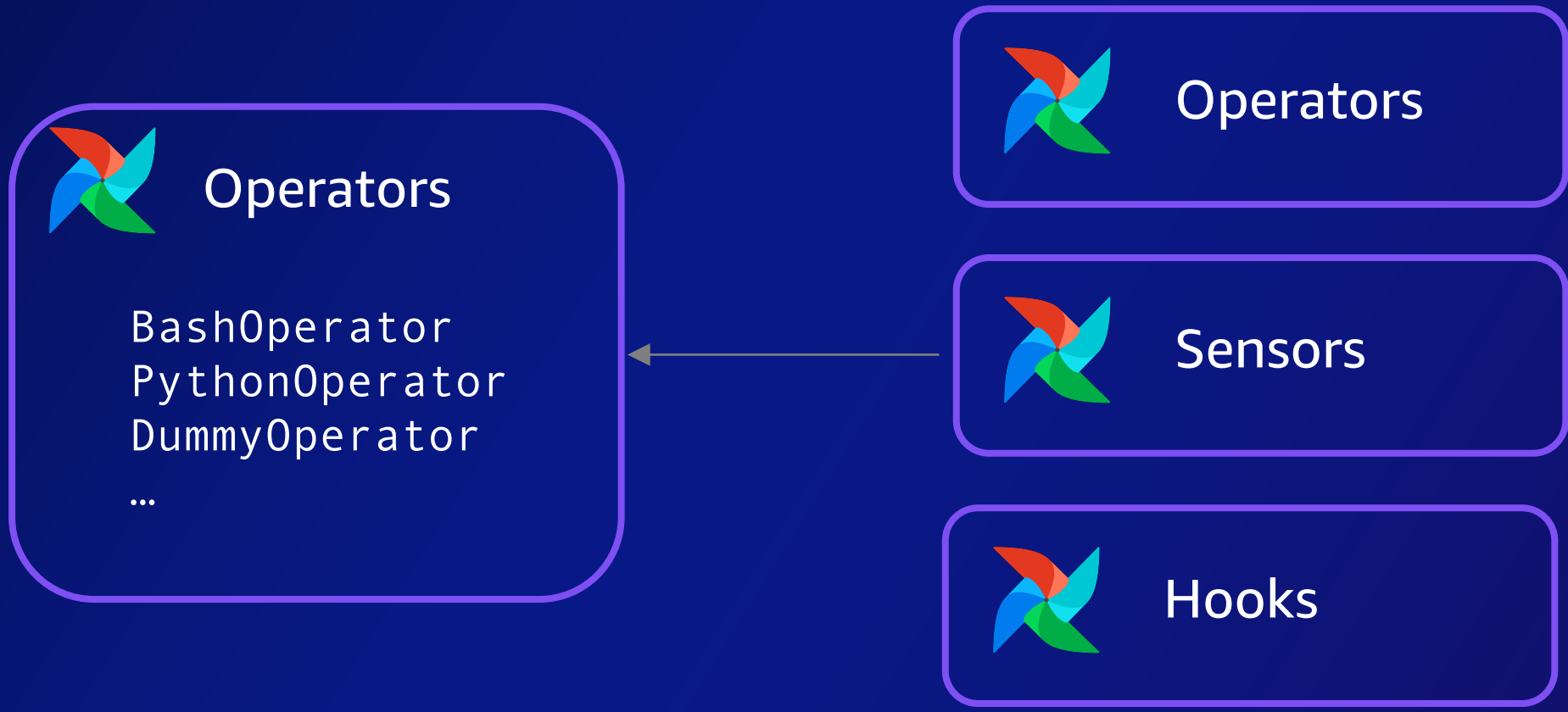
dag_id=daily_dw_ingest

Task

```
move_file = BashOperator(
        task_id='move_current_file',
        bash_command="cd {work_dir} && mv {source_file} {destination_file}”
        dag=dag
    )
```

task_id=move_current_file





Use Airflow Connections

Operators

- Airbyte
- Alibaba
- Amazon
- Apache Beam
- Apache Cassandra
- Apache Drill
- Apache Druid
- Apache HDFS
- Apache Hive
- Apache Kylin
- Apache Livy
- Apache Pig
- Apache Pinot
- Apache Spark
- Apache Sqoop
- Asana
- Celery
- IBM Cloudant
- Kubernetes
- Databricks
- Datadog
- Dingding
- Discord
- Docker
- Elasticsearch
- Exasol
- Facebook
- File Transfer Protocol (FTP)
- Github
- Google
- gRPC
- Hashicorp
- Hypertext Transfer Protocol (HTTP)
- Influx DB
- Internet Message Access Protocol (IMAP)
- Java Database Connectivity (JDBC)
- Jenkins
- Jira
- Microsoft Azure
- Microsoft PowerShell Remoting Protocol (PSRP)
- Microsoft SQL Server (MSSQL)
- Windows Remote Management (WinRM)
- MongoDB
- MySQL
- Neo4J
- ODBC
- OpenFaaS
- Opsgenie
- Oracle
- Pagerduty
- Papermill
- Plexus
- PostgreSQL
- Presto
- Qubole
- Redis
- Salesforce
- Samba
- Segment
- Sendgrid
- SFTP
- Singularity
- Slack
- Snowflake
- SQLite
- SSH
- Tableau
- Telegram
- Trino
- Vertica
- Yandex
- Zendesk

DAG

Task

```
from airflow import DAG
from datetime import datetime
from
airflow.providers.amazon.aws.operators.ecs
import ECSOperator

default_args = { 'owner': 'ubuntu',
'start_date': datetime(2019, 8, 14),
'retry_delay': timedelta(seconds=60*60) }
..
..
..
```

task_id=copy_data

Task

```
from airflow import DAG
from datetime import datetime
from
airflow.providers.amazon.aws.operators.ecs
import ECSOperator

default_args = { 'owner': 'ubuntu',
'start_date': datetime(2019, 8, 14),
'retry_delay': timedelta(seconds=60*60) }
..
..
..
```

task_id=store_raw_data

Task

```
from airflow import DAG
from datetime import datetime
from
airflow.providers.amazon.aws.operators.ecs
import ECSOperator

default_args = { 'owner': 'ubuntu',
'start_date': datetime(2019, 8, 14),
'retry_delay': timedelta(seconds=60*60) }
..
..
..
```

task_id=clean data

Task

```
from airflow import DAG
from datetime import datetime
from
airflow.providers.amazon.aws.operators.ecs
import ECSOperator

default_args = { 'owner': 'ubuntu',
'start_date': datetime(2019, 8, 14),
'retry_delay': timedelta(seconds=60*60) }
..
..
..
```

task_id=process_data

Task

```
from airflow import DAG
from datetime import datetime
from
airflow.providers.amazon.aws.operators.ecs
import ECSOperator

default_args = { 'owner': 'ubuntu',
'start_date': datetime(2019, 8, 14),
'retry_delay': timedelta(seconds=60*60) }
..
..
..
```

task_id=move_to_datawarehouse

Import python libraries

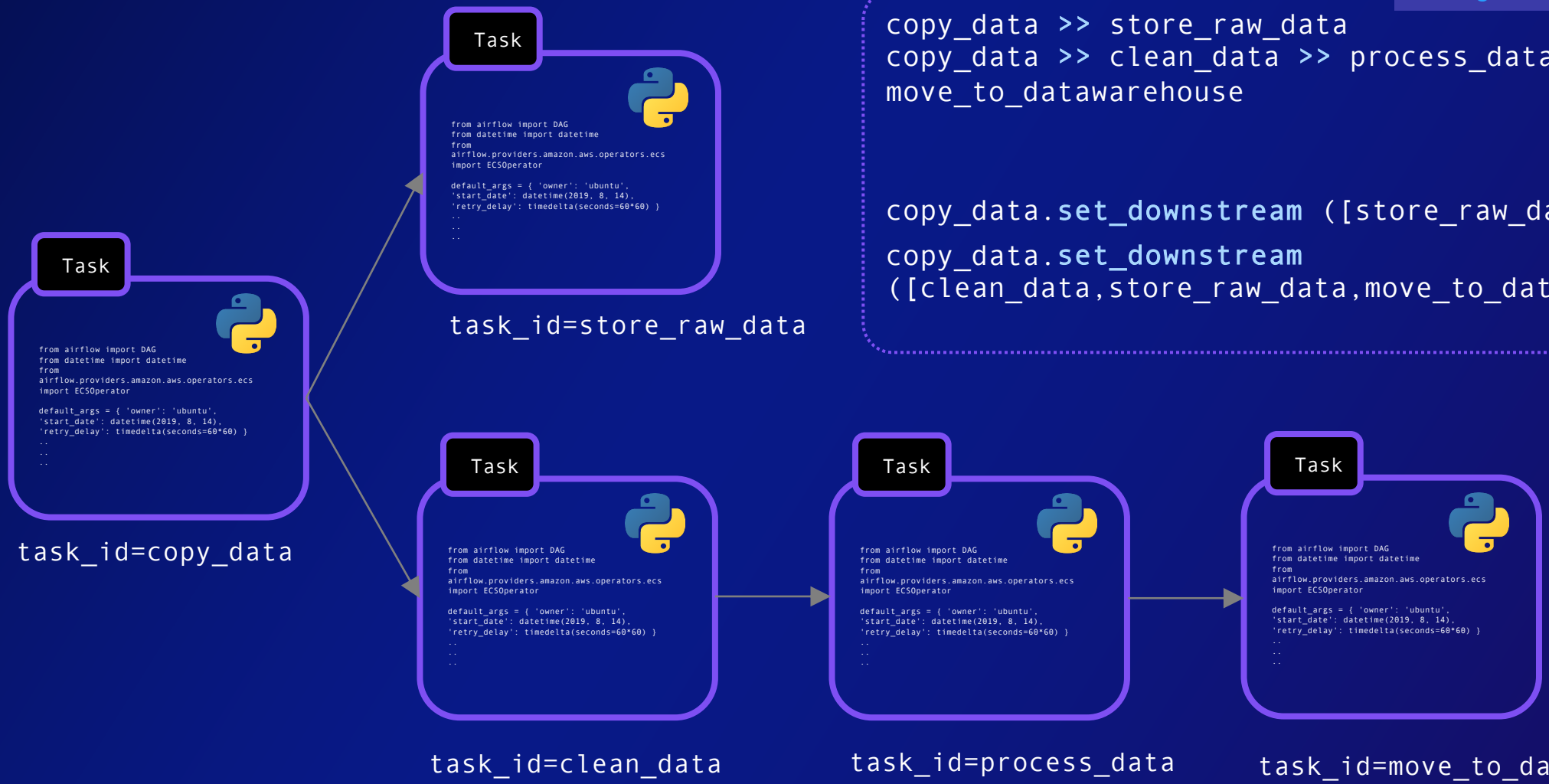
Define standard settings

Define workflow name

Define workflow settings

```
from airflow import DAG

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['ricsue@amazon.com'],
    'email_on_failure': False,
    'email_on_retry': False
}

DAG_ID = 'daily_dw_ingest'

dag = DAG(
    dag_id=DAG_ID,
    default_args=default_args,
    description='First Apache Airflow DAG',
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['devcon','demo'],
)
```

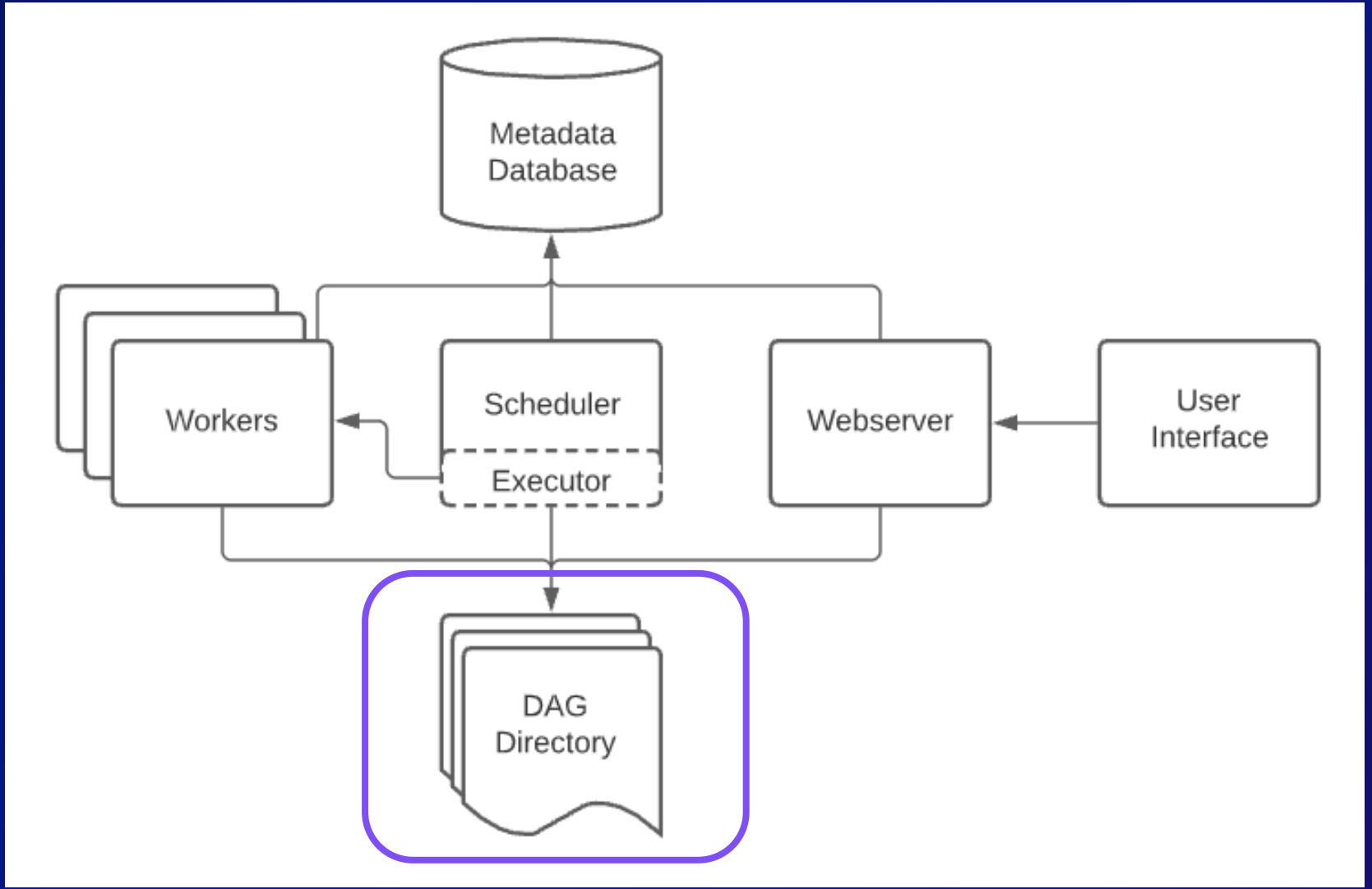
Control flow



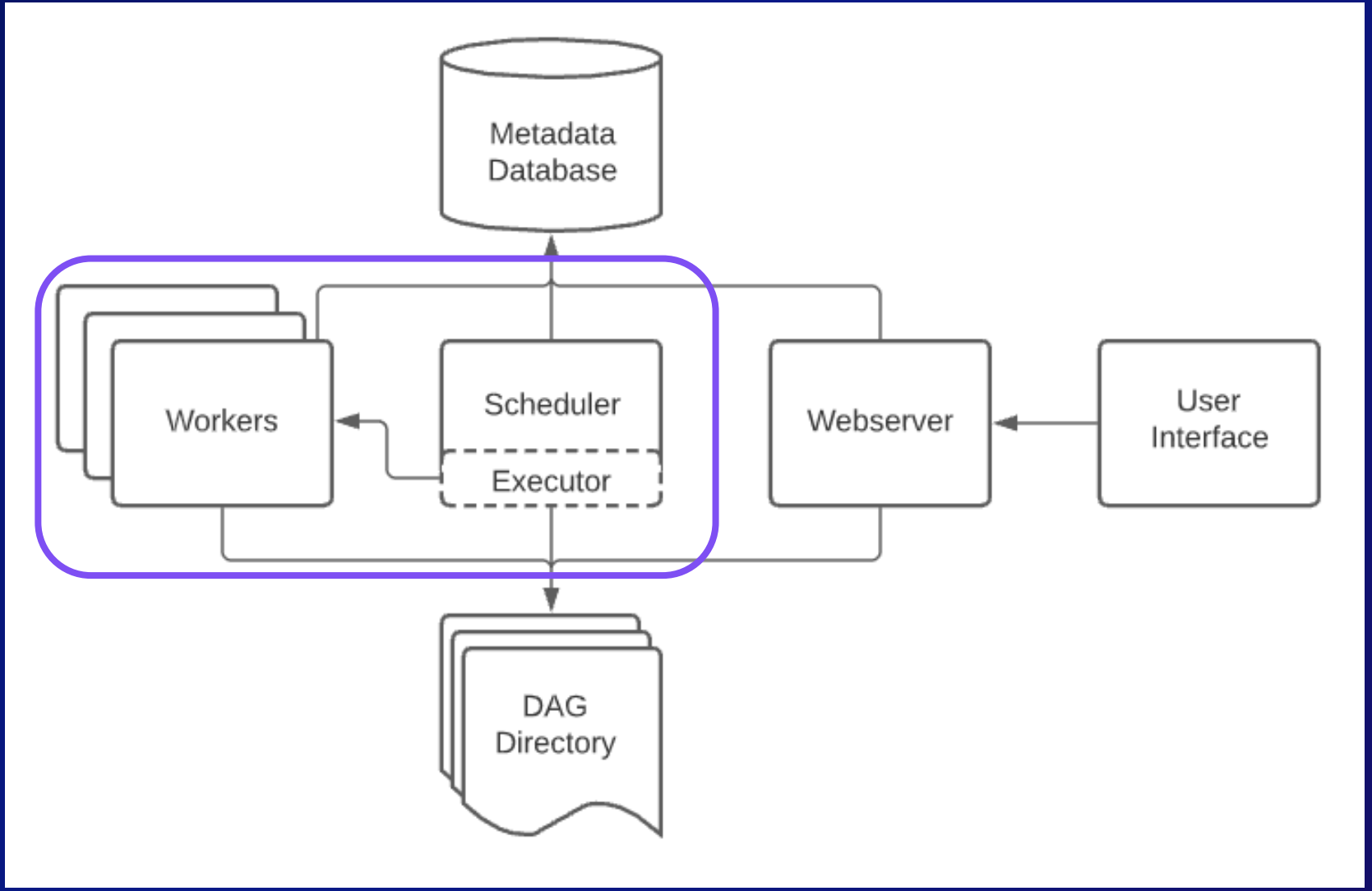
Defining task dependencies

```
copy_data >> store_raw_data
copy_data >> clean_data >> process_data >>
move_to_datawarehouse


copy_data.set_downstream ([store_raw_data])
copy_data.set_downstream
([clean_data,store_raw_data,move_to_datawarehouse])
```

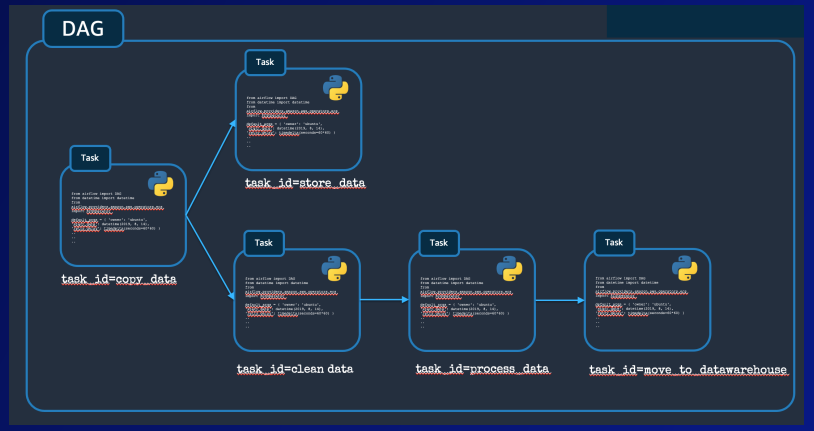
Deploying your DAGs



Apache Airflow Scheduler and Workers



Scheduling our Workflows (DAGs)

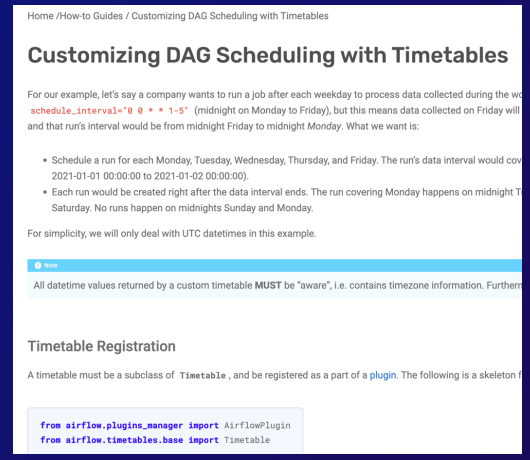



```
dag = DAG(
        dag_id="daily_dw_ingest ",
        schedule_interval=None,
        start_date=datetime.datetime(2022, 2, 1),
        catchup=False,
        tags=["example"],
    )
```

schedule_interval

schedule_interval="/10 * * * *" – every 10 min*
*schedule_interval="0 */2 * * *" – every 2 hours*
*schedule_interval="0 */1 * * *"* - every hour
schedule_interval="*/5 * * * *" – every 5 mins

To provide more scheduling flexibility, determining when a DAG should run is now done with **Timetables**.


Home /How-to Guides / Customizing DAG Scheduling with Timetables

Customizing DAG Scheduling with Timetables

For our example, let's say a company wants to run a job after each weekday to process data collected during the wo
schedule_interval="0 0 * * 1-5" (midnight on Monday to Friday), but this means data collected on Friday will
and that run's interval would be from midnight Friday to midnight *Monday*. What we want is:

- Schedule a run for each Monday, Tuesday, Wednesday, Thursday, and Friday. The run's data interval would cov
2021-01-01 00:00:00 to 2021-01-02 00:00:00).
- Each run would be created right after the data interval ends. The run covering Monday happens on midnight T
Saturday. No runs happen on midnights Sunday and Monday.

For simplicity, we will only deal with UTC datetimes in this example.

Note

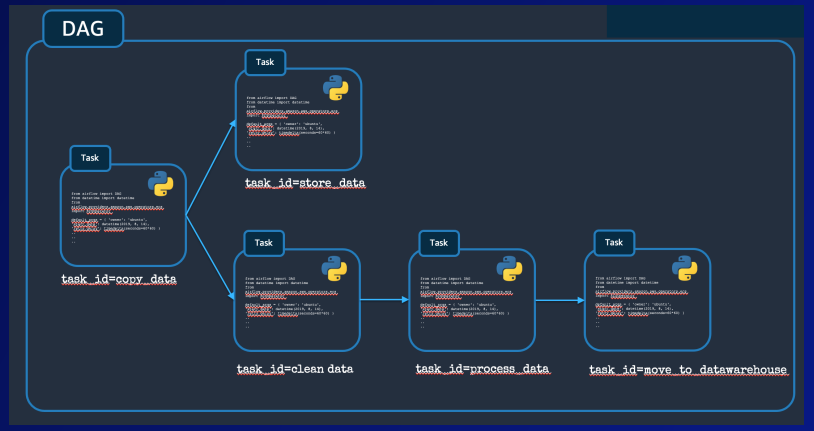
All datetime values returned by a custom timetable **MUST** be "aware", i.e. contains timezone information. Furtherm

Timetable Registration

A timetable must be a subclass of `Timetable`, and be registered as a part of a plugin. The following is a skeleton f

```
from airflow.plugins_manager import AirflowPlugin
from airflow.timetables.base import Timetable
```

Understanding how workflows (DAGs) run




```
dag = DAG(
        dag_id="daily_dw_ingest ",
        schedule_interval="0 */1 * * *",
        start_date=datetime.datetime(2022, 2,
1),
        catchup=True,
        tags=["example"],
    )
```

Execution Date: 1st Feb, 2022, 01:00

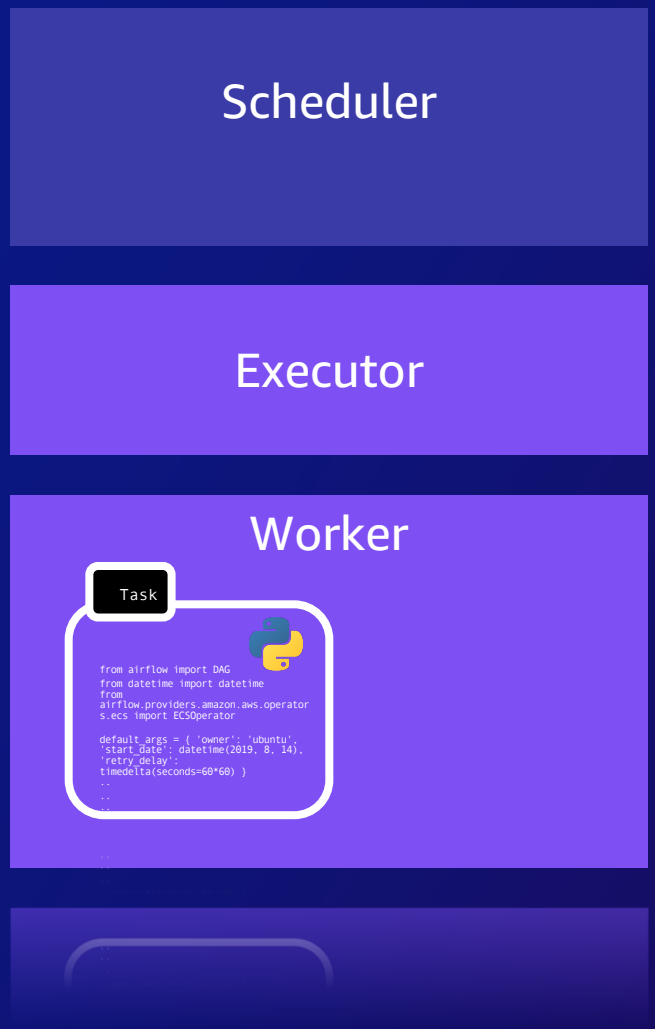
Execution Date: 1st Feb, 2022, 02:00

Execution Date: 1st Feb, 2022, 03:00 …

Running tasks – the Executor

Local Executors

Remote Executors



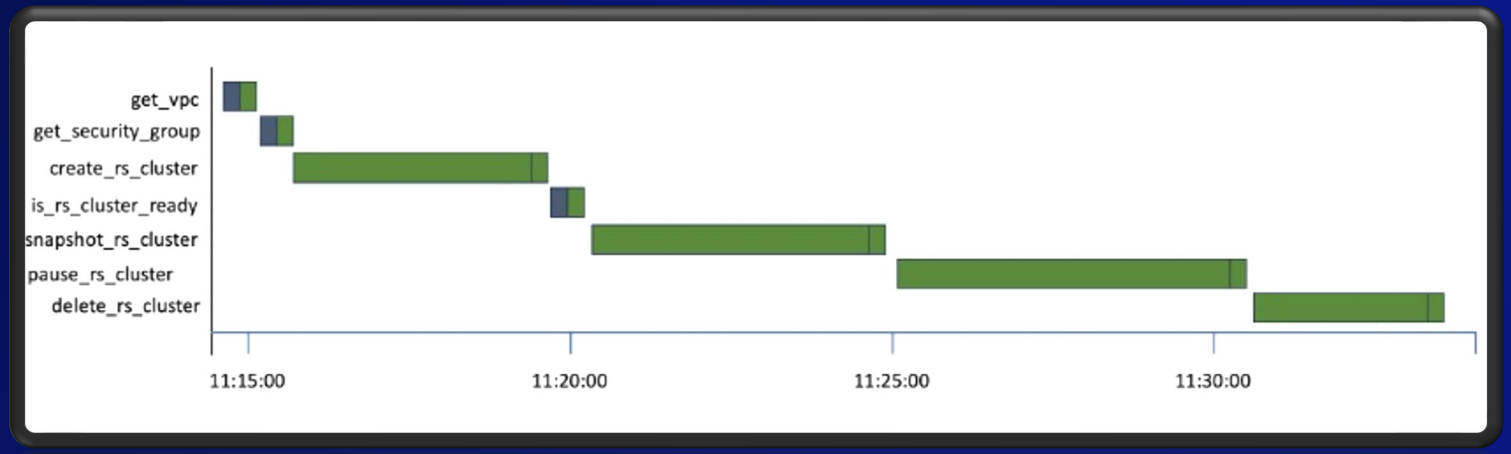
idempotency

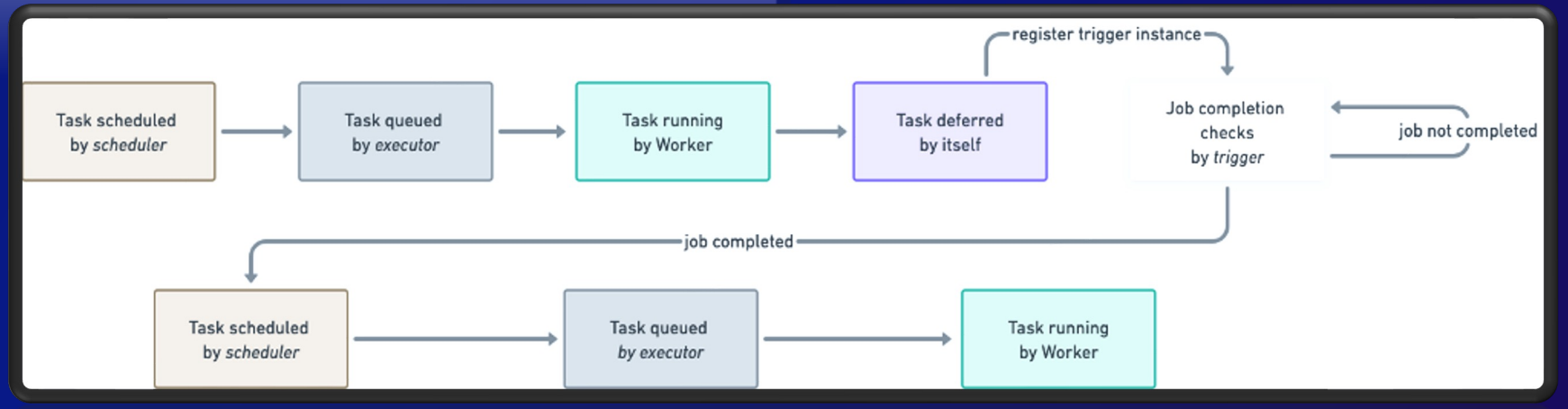
//ˌɪd.əmˈpəʊ.tənt//

adjective

1. An idempotent element of a set does not change in value when multiplied by itself.

Deferrable Operators and Triggers





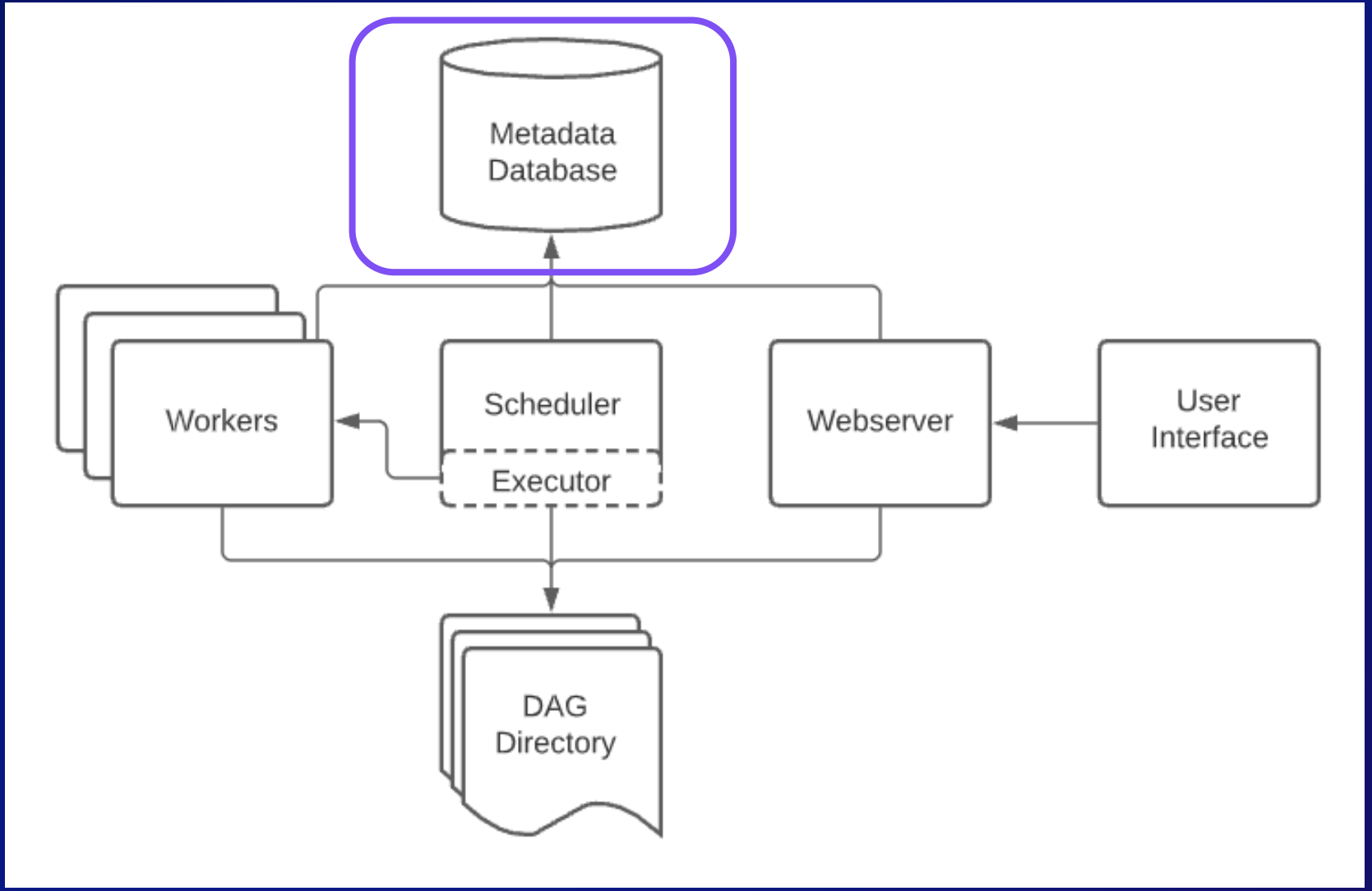
https://aws-oss.beachgeek.co.uk/3e8

Backfill

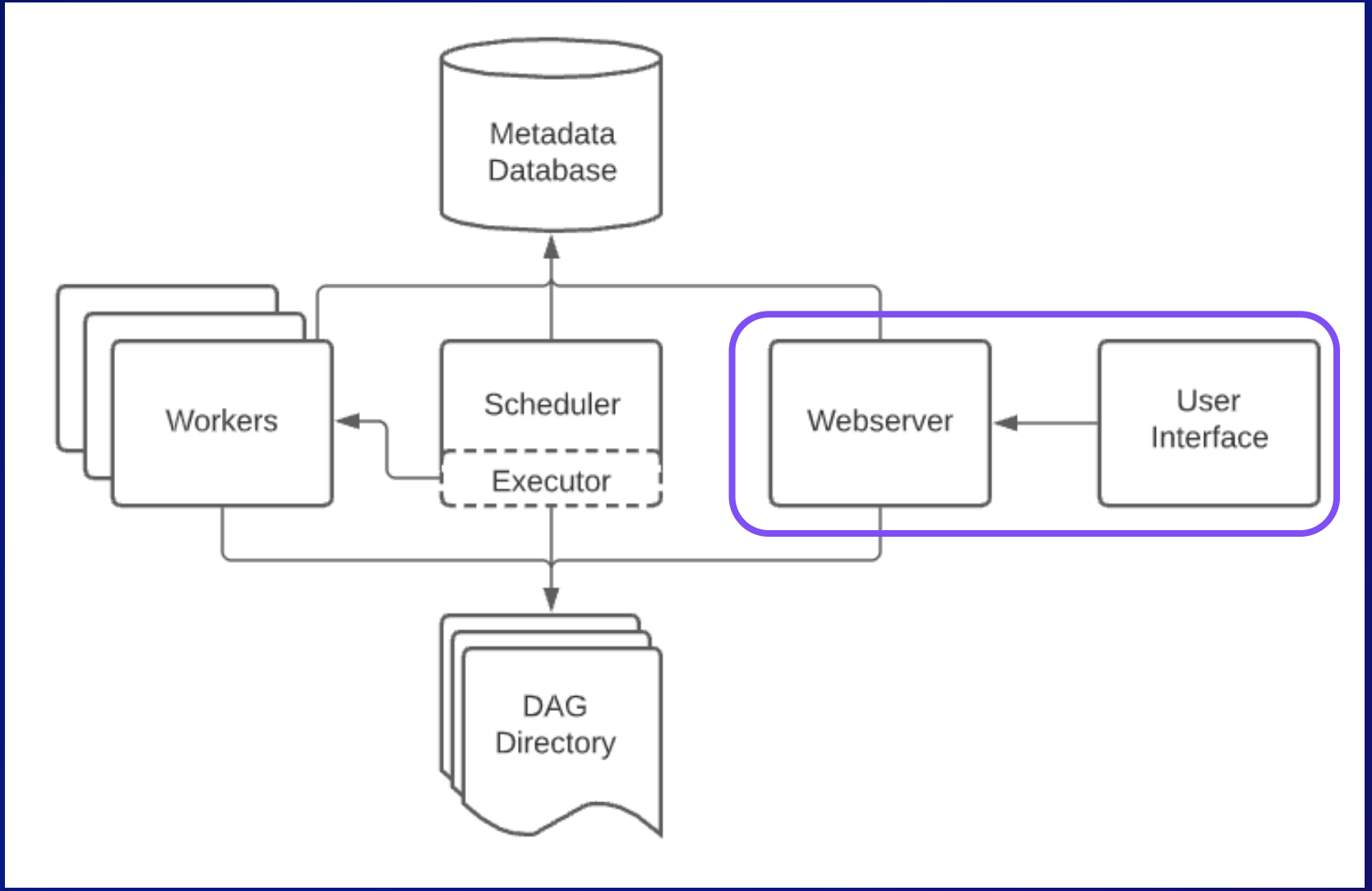
```
def ts_query(**kwargs):
    start = str(kwargs['execution_date']).replace("T", " ")
    finish = str(kwargs['next_execution_date']).replace("T", " ")
    execution_time = str(kwargs['execution_date'])
    s3folder = execution_time[0:16]
    query = """
                WITH interpolated_timeseries AS (
                SELECT sensor_id,
                        INTERPOLATE_LINEAR(
                        CREATE_TIME_SERIES(time, measure_value::double),
                        SEQUENCE(min(time), max(time), 1s)) AS interpolated_temperature,
                        INTERPOLATE_LOCF(
                        CREATE_TIME_SERIES(time, status),
                        SEQUENCE(min(time), max(time), 1s)) AS locf_status
                    FROM "{db}"."{tbl}"
                WHERE measure_name = 'temperature' AND time BETWEEN '{start}' AND '{finish}'
                GROUP BY sensor_id
                )
                SELECT int.sensor_id, t.time, min(s.status) AS status, avg(t.temp) AS temperature
                FROM interpolated_timeseries AS int
                CROSS JOIN UNNEST(interpolated_temperature) AS t (time, temp)
                CROSS JOIN UNNEST(locf_status) AS s (time, status)
                WHERE t.time = s.time
                GROUP BY int.sensor_id, t.time
                """.format(start=start,finish=finish,db=tsdb,tbl=tstbl)
```

```
airflow dags backfill --start-date START_DATE --end-date END_DATE dag_id
```

Apache Airflow Metadata



Apache Airflow UI



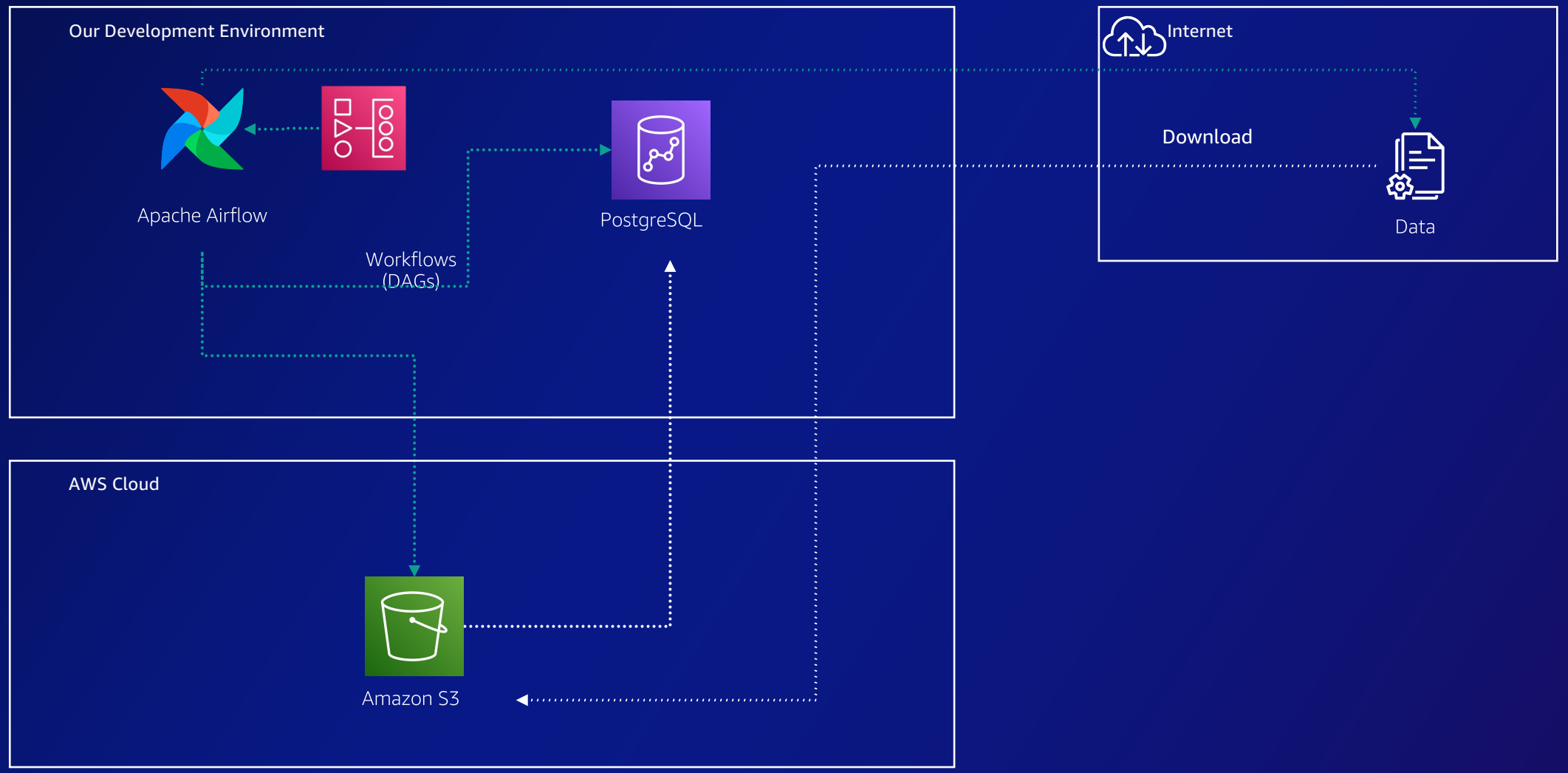
Task Flow

```
create_joke_table = """
CREATE TABLE IF NOT EXISTS bad_jokes (
category TEXT NOT NULL,
joke TEXT NOT NULL,
punchline TEXT NOT NULL
);

"""

create_mysql_table = MySqlOperator(
    task_id="create_mysql_table",
    sql=create_joke_table,
    dag=dag
    )

create_mysql_table
```

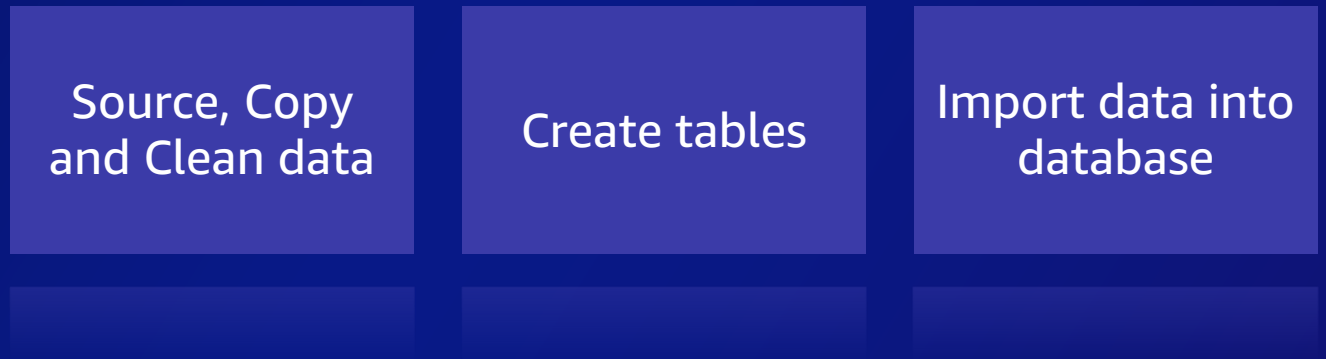
```
@task
def create_mysql_table():
    create_joke_table = """
    CREATE TABLE IF NOT EXISTS bad_jokes (
    category TEXT NOT NULL,
    joke TEXT NOT NULL,
    punchline TEXT NOT NULL
    );
    """
    return create_joke_table

create_joke_table_task = create_mysql_table()
create_joke_table_task
```

Workshop

Building the ultimate joke repository



Building our data pipeline



Which Operators to use?

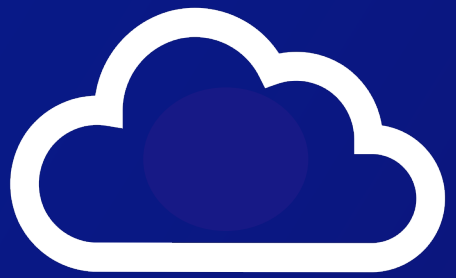
Source, Copy and Clean data	`PythonOperator`
Create tables	`PostgresOperator`
Import data into database	`S3ToSQLOperator`

Source, Copy and Clean data

py-jokes.py

```
import requests
import json
from datetime import datetime
import csv
import boto3

s3_bucket="094459-jokes"
time = datetime.now().strftime("%m/%d/%Y").replace('/', '-')

def pull_jokes():

# pull jokes with the api
url = r"https://official-joke-api.appspot.com/random_ten"
response = requests.get(url)
text = json.loads(response.text)
csv_filename = f"jokes-{time}.csv"
s3_csv_file= f"{time}/{csv_filename}"

# export to csv
..
..
```

1



https://official-joke-api.appspot.com/random_ten

2

```
-rw-r--r-- 1 ricsue 1896053708 1084 Mar 23 08:08 jokes-02-23-2023.csv
-rw-r--r-- 1 ricsue 1896053708  720 Mar 24 11:10 jokes-03-23-2023.csv
-rw-r--r-- 1 ricsue 1896053708  484 Mar 24 11:16 jokes-04-23-2023.csv
-rw-r--r-- 1 ricsue 1896053708    0 Mar 24 14:20 jokes-05-23-2023.csv
-rw-r--r-- 1 ricsue 1896053708  841 Mar 24 14:23 jokes-06-24-2023.csv
..
```

My secret stash of brilliant jokes

Source, Copy and Clean data

jokes-dag.py

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator

from datetime import datetime, timedelta
import requests
import json
from datetime import datetime
import csv
import boto3


def pull_jokes():

   # pull jokes with the api
   url = r"https://official-joke-api.appspot.com/random_ten"
   response = requests.get(url)
   text = json.loads(response.text)
"""
"""

dag = DAG(
   'my_funny_joke_archive',
   description='The essential collection of bad jokes to keep me amused',
   start_date=datetime(2023, 6, 1),
   schedule_interval=timedelta(days=1),
)

csv_generate_task = PythonOperator(
     task_id='grab_jokes',
     python_callable=pull_jokes,
     dag=dag
   )

csv_generate_task
```

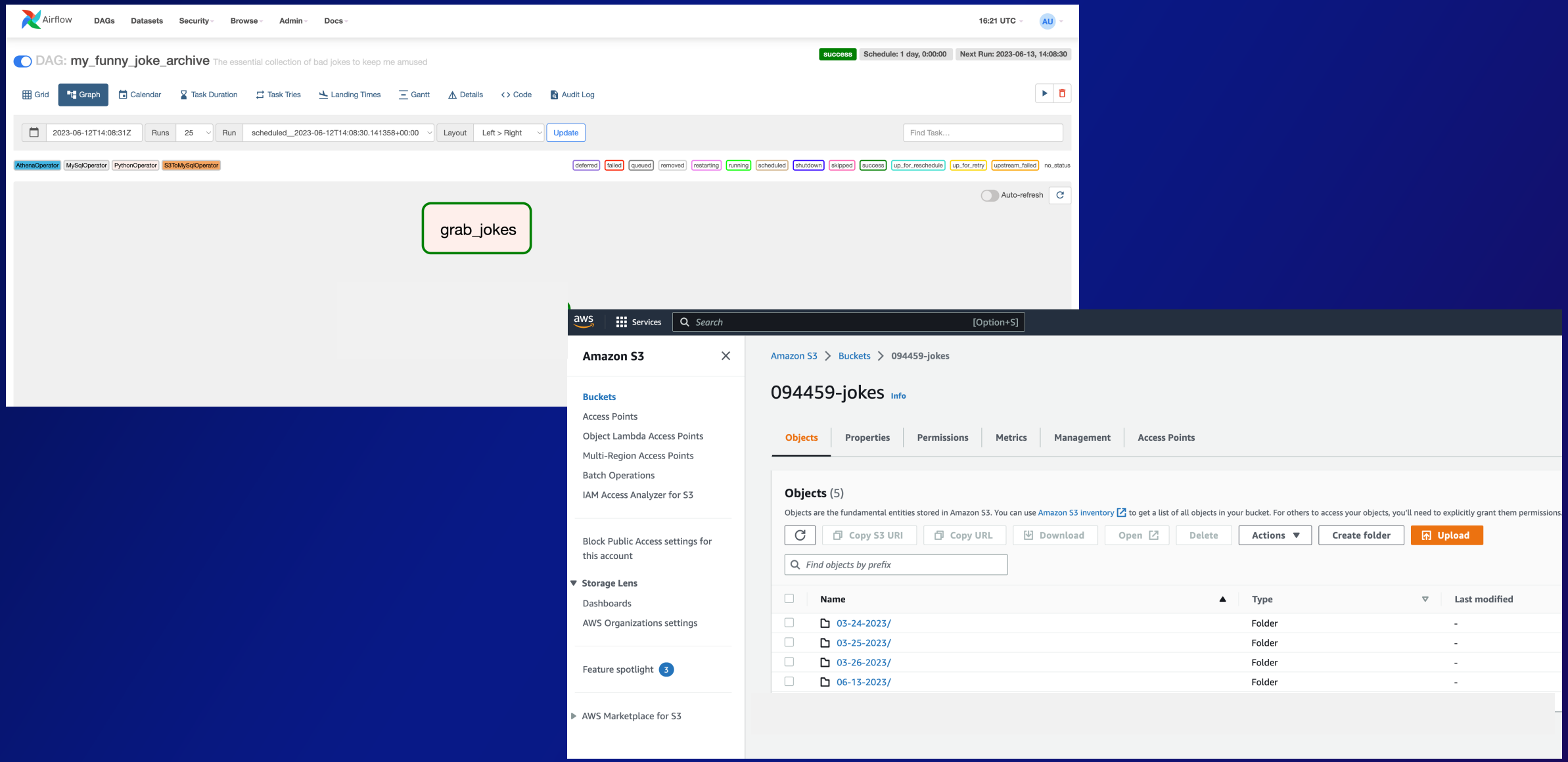
Import Airflow libraries and Operators

Our Python code to grab the jokes

Define our DAG characteristics

Create a task which uses the PythonOperator

Define our DAG graph



Create tables

jokes-dag.py

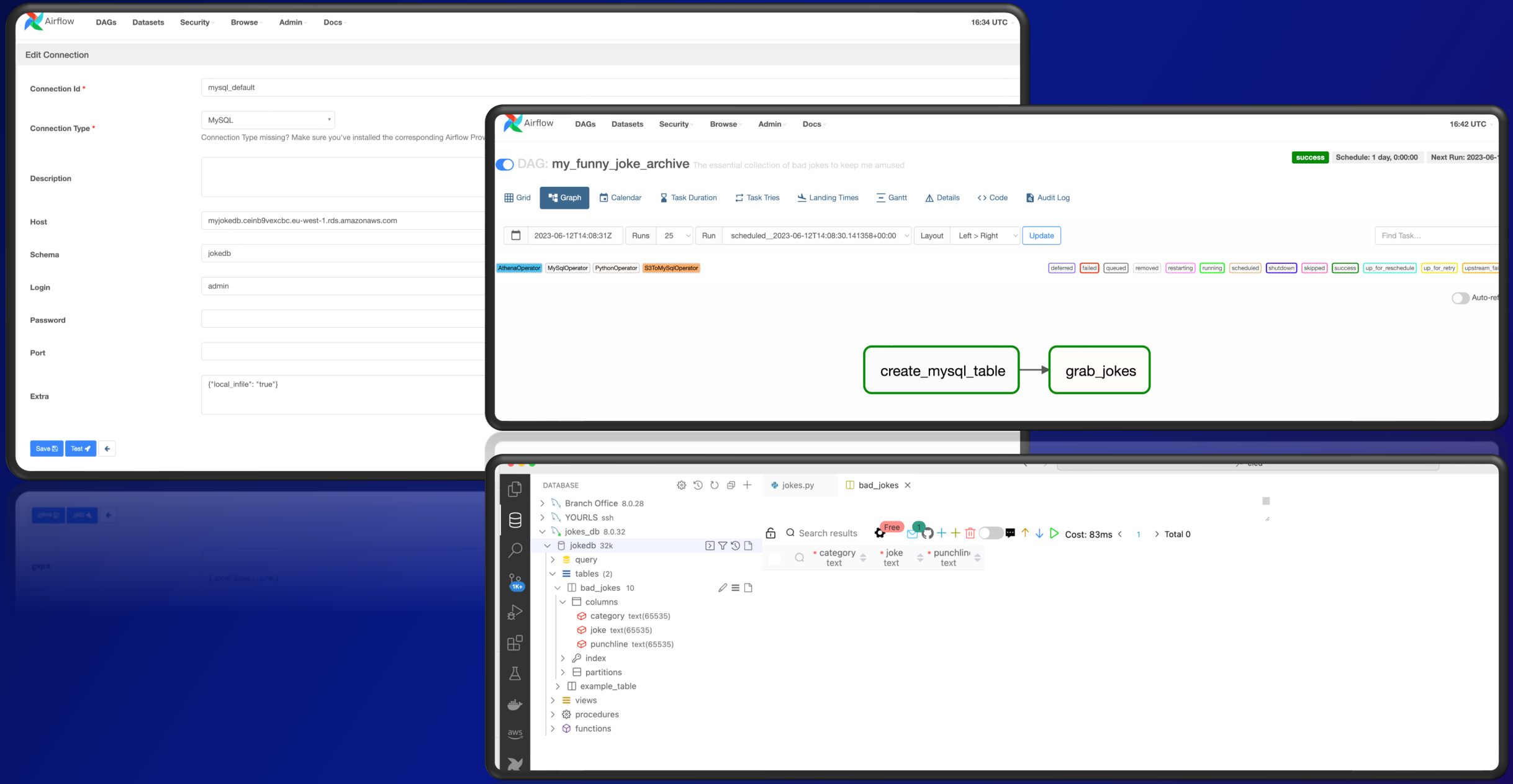
```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.operators.postgres import PostgresOperator

from datetime import datetime, timedelta
import requests
import json
from datetime import datetime
import csv
import boto3


create_joke_table = """
CREATE TABLE IF NOT EXISTS bad_jokes (
 category TEXT NOT NULL,
 joke TEXT NOT NULL,
 punchline TEXT NOT NULL
);
"""

dag = DAG(
   'my_funny_joke_archive',
   description='The essential collection of bad jokes to keep me amused',
   start_date=datetime(2023, 6, 1),
   schedule_interval=timedelta(days=1),
)

create_postgres_table = PostgresOperator(
   task_id="create_postgres_table",
   sql=create_joke_table,
   dag=dag
   )


create_mysql_table >> csv_generate_task
```

Import PostgresOperator

Create table SQL

Create a task which uses the PostgresOperator to create our tables

Define our DAG graph



Import data into database

jokes-dag.py

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.amazon.aws.transfers.s3_to_sql import
S3ToSqlOperator

from datetime import datetime, timedelta
import requests
import json
from datetime import datetime
import csv
import boto3


dag = DAG(
   'my_funny_joke_archive',
   description='The essential collection of bad jokes to keep me amused',
   start_date=datetime(2023, 6, 1),
   schedule_interval=timedelta(days=1),
)

export_csv_to_postgres = S3ToSqlOperator(
     task_id="export_csv_to_postgres_task",
     s3_bucket=f"{s3_bucket}",
     s3_key=f"{s3_csv_file}",
     table=SQL_TABLE_NAME,
     column_list=SQL_COLUMN_LIST,
     parser=parse_csv_to_list,
     sql_conn_id="postgres-jokes",
   )
create_mysql_table >> csv_generate_task >> export_csv_to_postgres
```

Import S3ToSQLOperator

Create task to ingest files from the S3 bucket into our PostgreSQL tables using the S3ToSQLOperator

Define our DAG graph