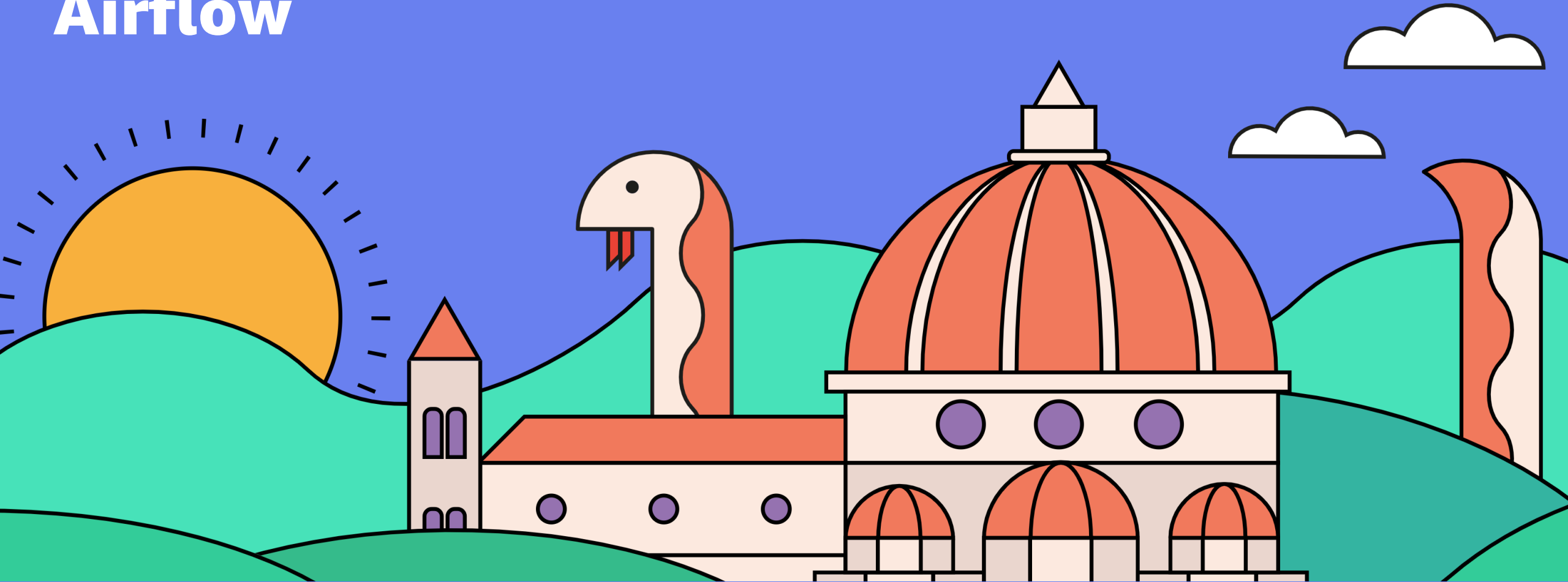


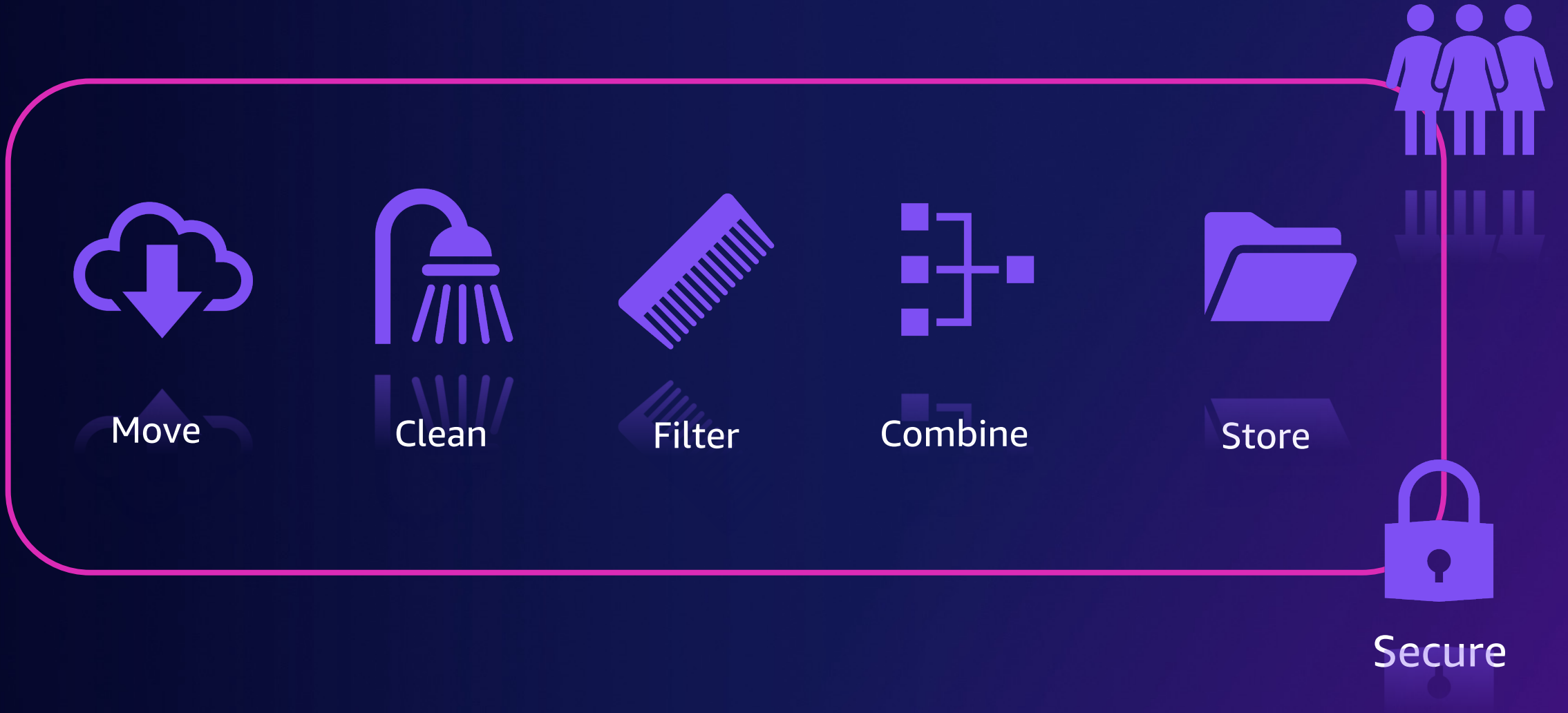
Building data pipelines with Apache Airflow



Ricardo Sueiras

Developer Advocate for Open Source | Amazon Web Services

What is in your data pipeline?



orchestration

/ɔːkɪ'streɪʃ(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

Introducing Apache Airflow



Apache
Airflow

workflow

/ˈwə:kfləʊ/

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

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'],
)
```

`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

Task



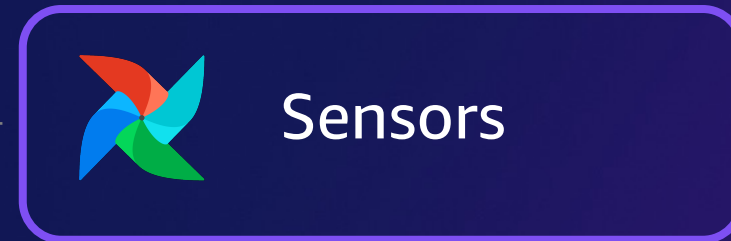
```
move_file = BashOperator(  
    task_id='move_current_file',  
    bash_command="cd {work_dir} && mv  
    {source_file} {destination_file}"  
    dag=dag  
)
```

import



Operators

BashOperator
PythonOperator
DummyOperator
...



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=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=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=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

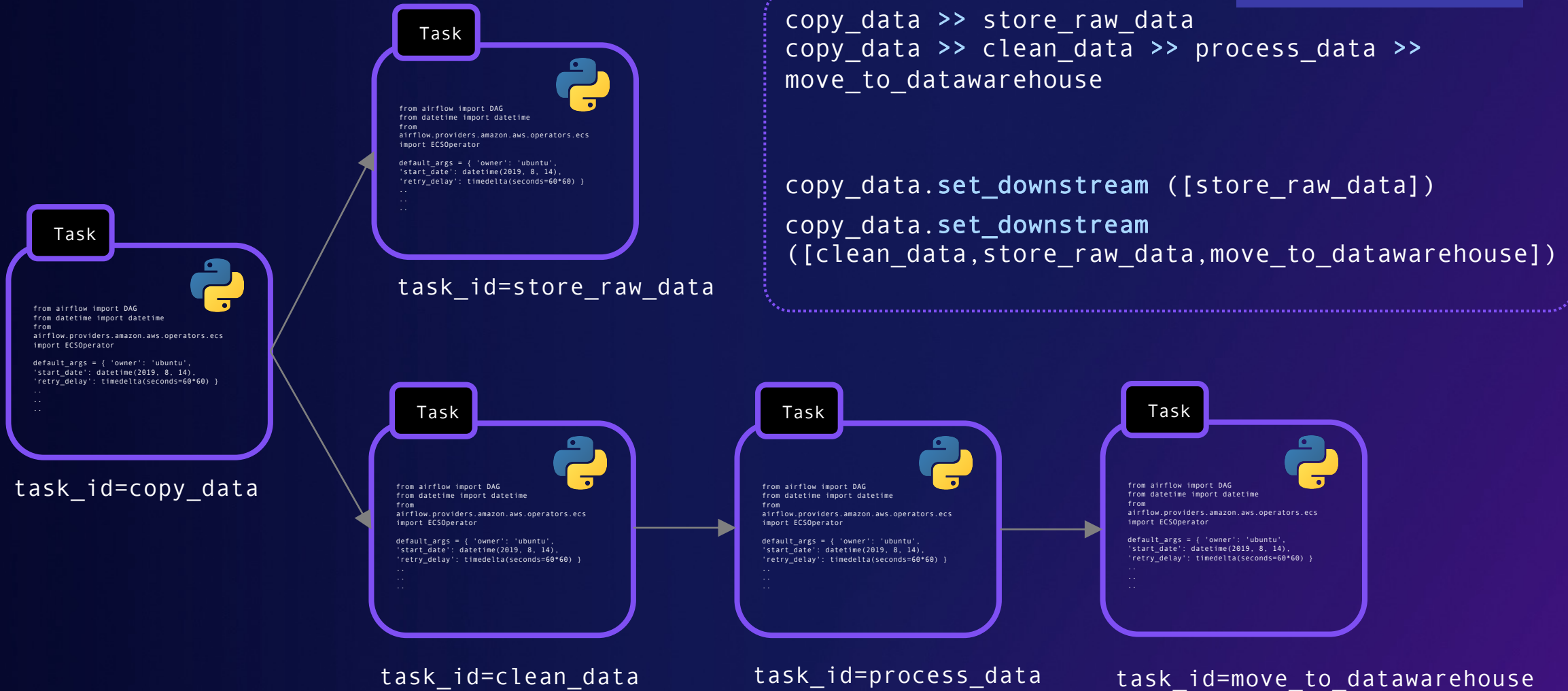
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['ricscue@amazon.com'],
    'email_on_failure': False,
    'email_on_retry': False
}

DAG_ID = 'daily_dw_ingest'

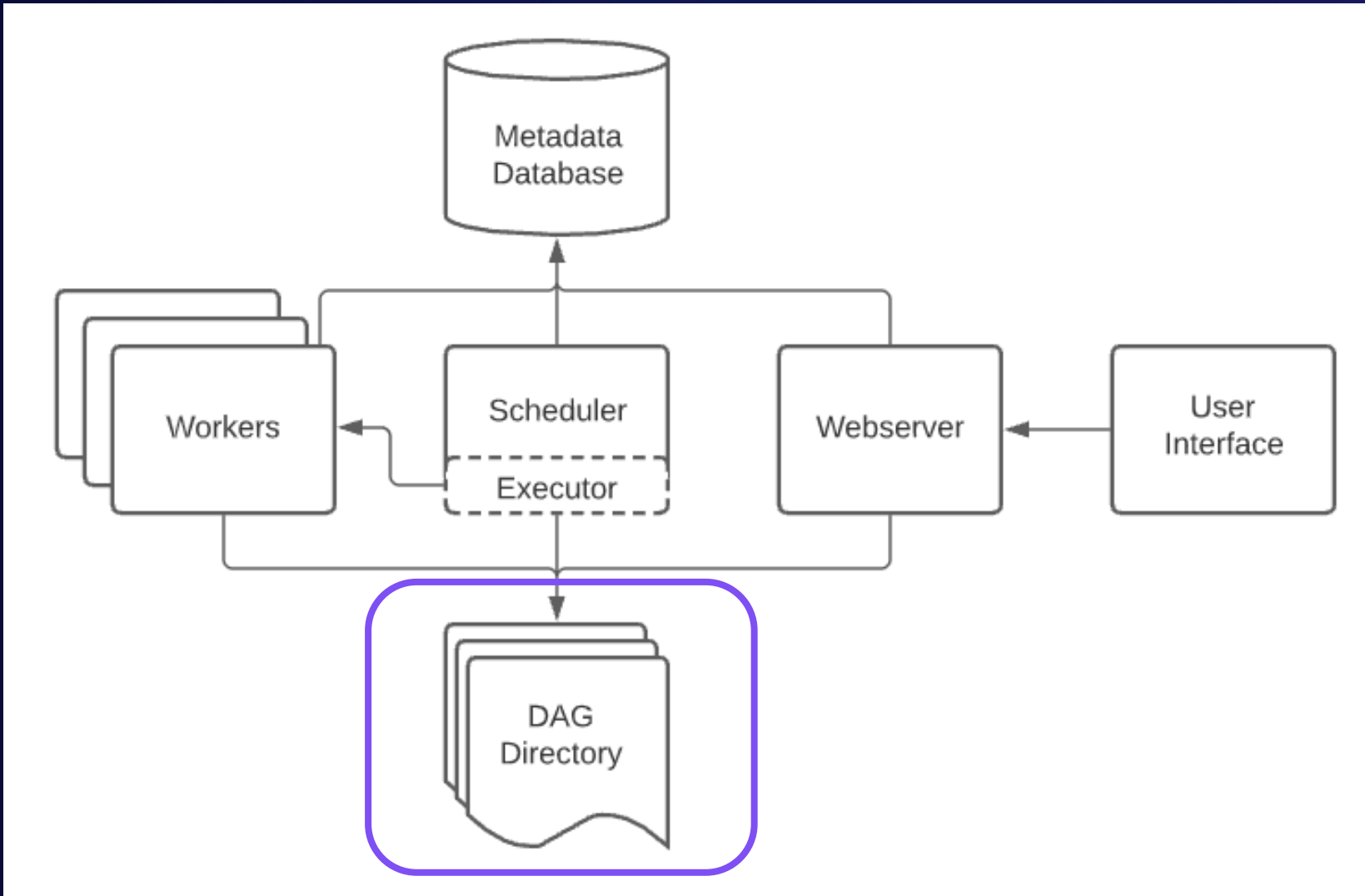
dag = DAG(
    dag_id=DAG_ID,
    default_args=default_args,
    description='First Apache Airflow DAG',
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['devcon', 'demo'],
)
```

Control flow

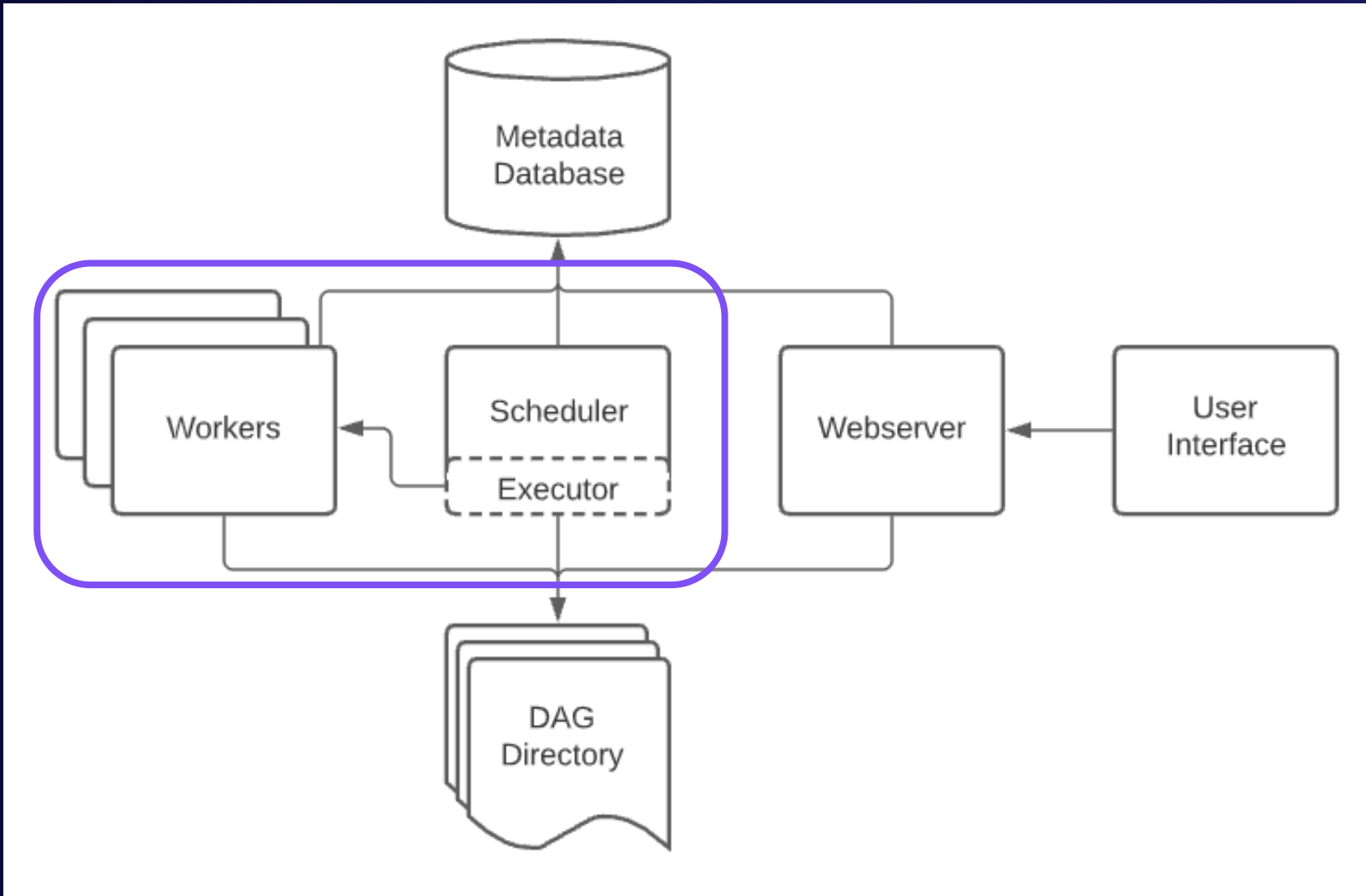
Defining task dependencies



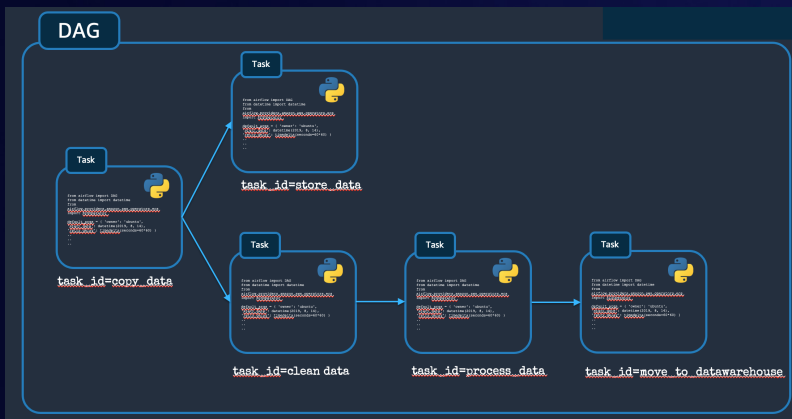
Deploying your DAGs



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

****New****

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 week. We can use the `schedule_interval` parameter to define this, but this means data collected on Friday will not be processed until the next Monday. What we want is:

- Schedule a run for each Monday, Tuesday, Wednesday, Thursday, and Friday. The run's data interval would cover the period from 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 Tuesday. No runs happen on midnights Sunday and Monday.

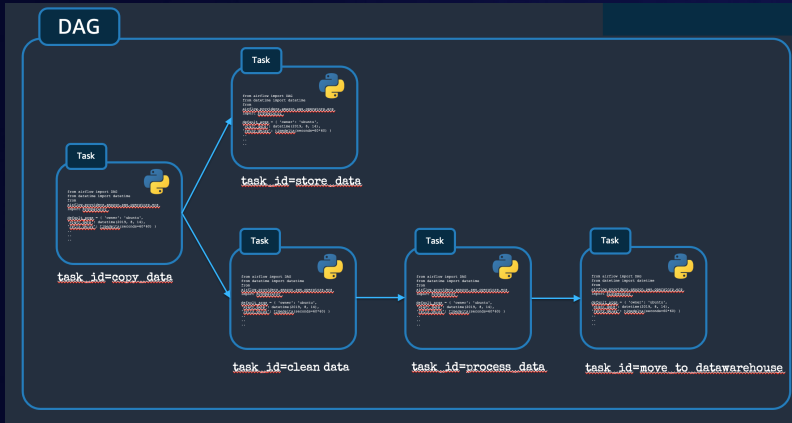
For simplicity, we will only deal with UTC datetimes in this example.

Timetable Registration

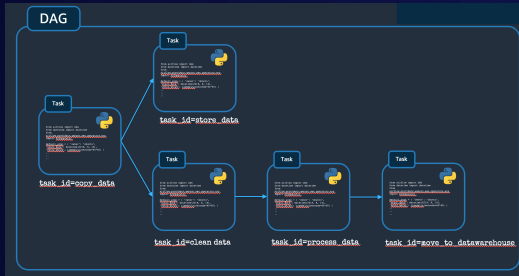
A timetable must be a subclass of `Timetable`, and be registered as a part of a plugin. The following is a skeleton for a plugin:

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

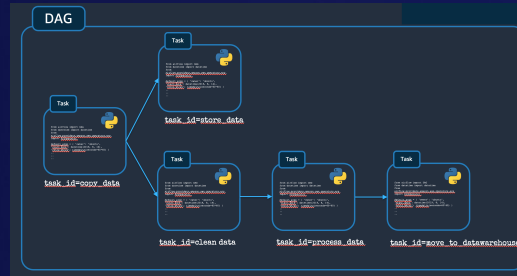

Understanding how workflows (DAGs) run



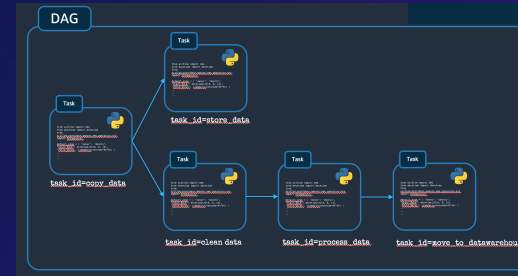
```
dag = DAG(  
    dag_id="daily_dw_ingest ",  
    schedule_interval="0 */1 * * *",  
    start_date=datetime.datetime(2022, 2, 1),  
    catchup=True,  
    tags=["example"],  
)
```



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

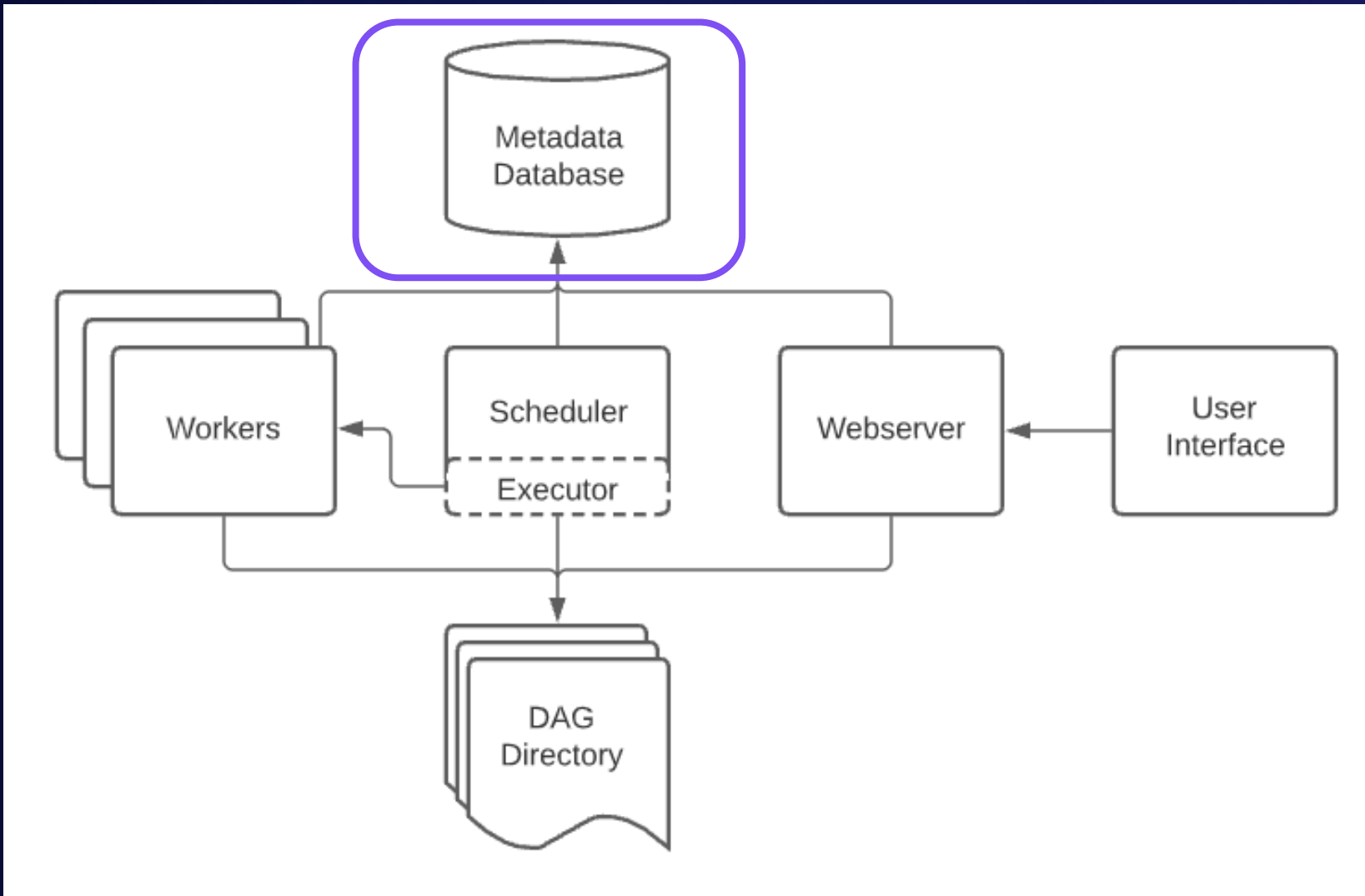


Execution Date: 1st Feb, 2022, 02:00

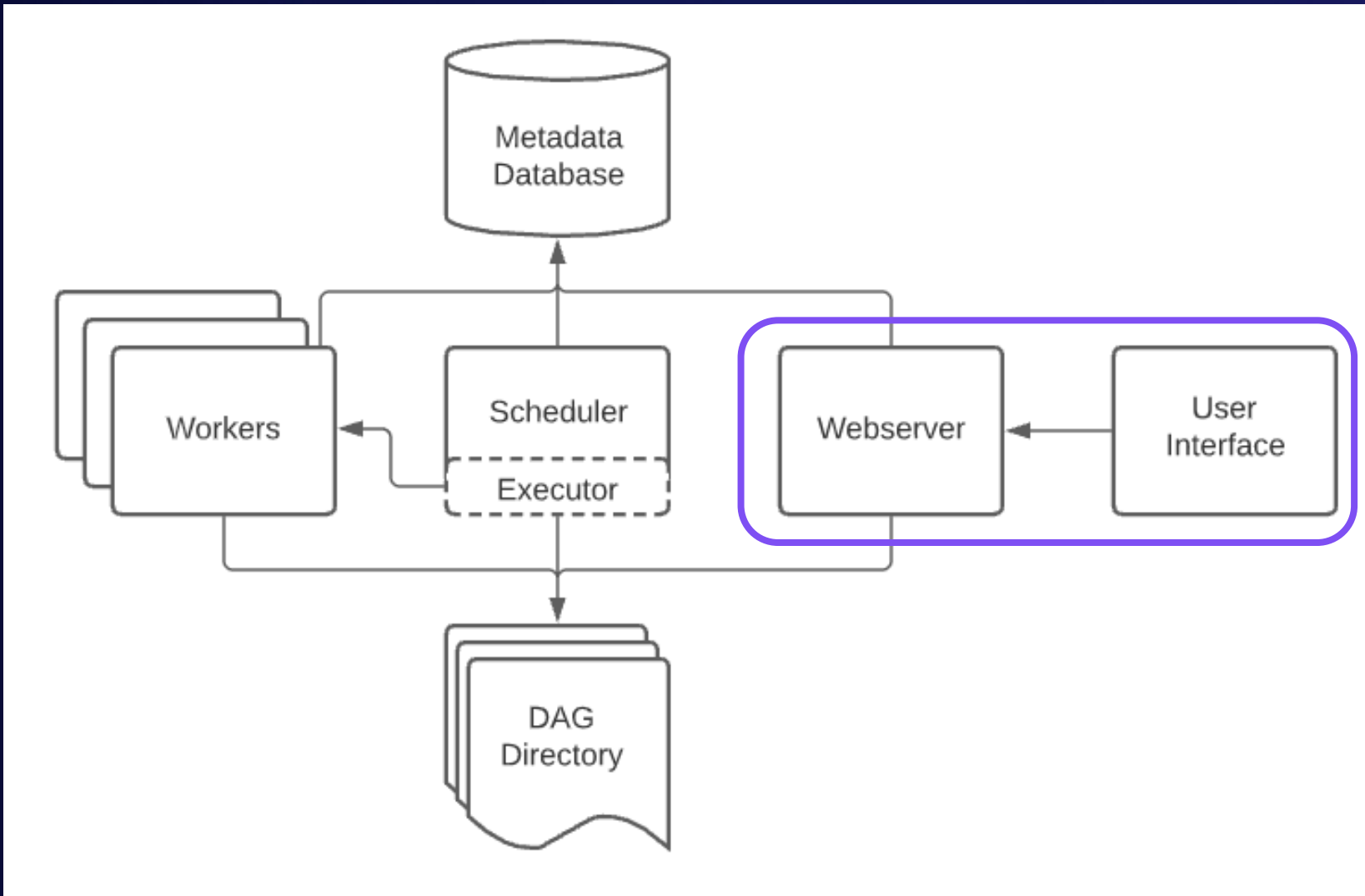


Execution Date: 1st Feb, 2022, 03:00 ...

Apache Airflow Metadata



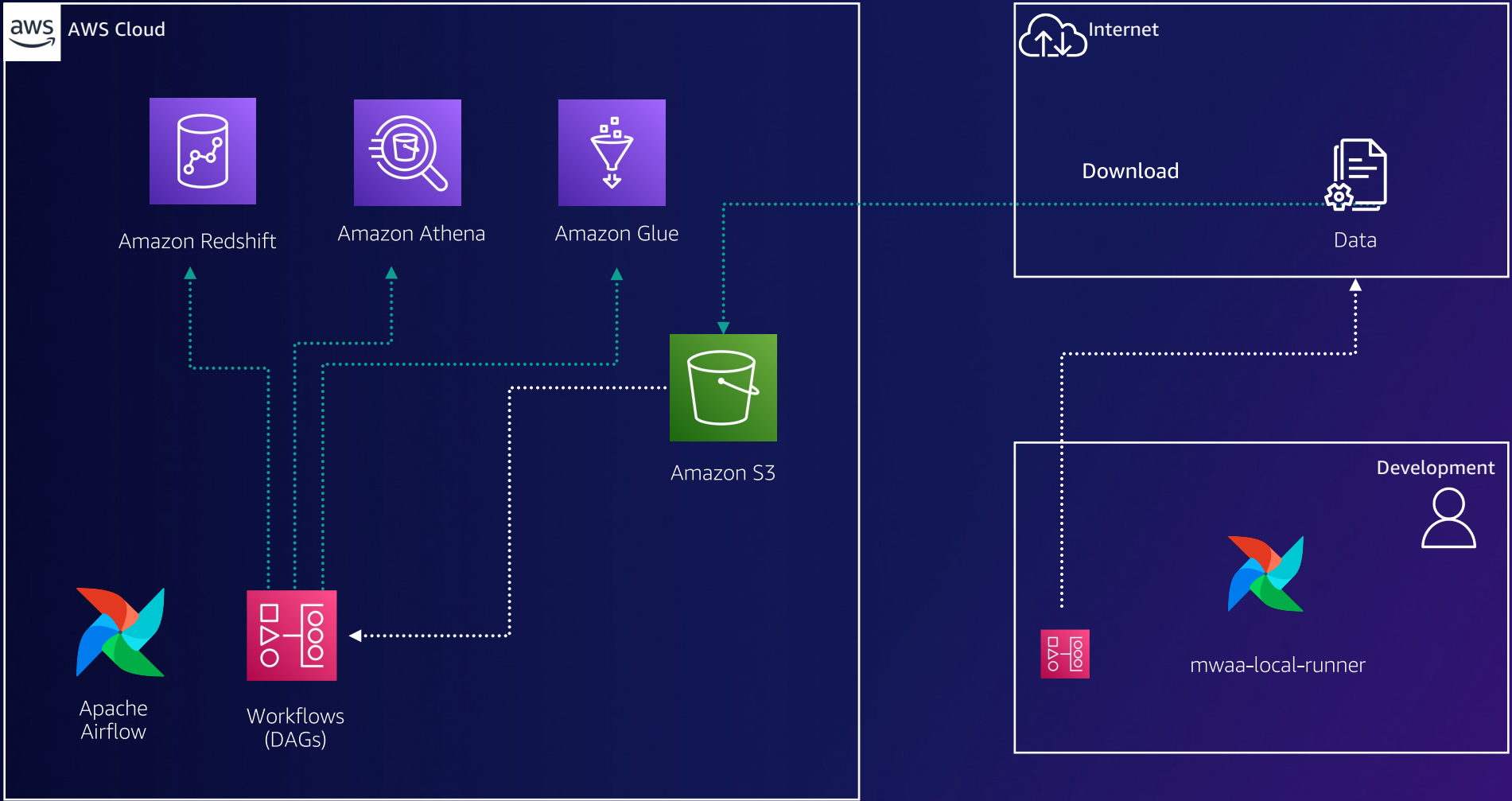
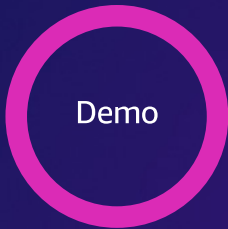
Apache Airflow UI



Demo



Building our data pipeline



Building our data pipeline

Demo

Source and copy
data

Ingest into data
lake

Merge and
consolidate data

Clean data

Copy into our
data warehouse

Building our data pipeline

Demo

Source and copy
data

Task

PythonOperator

```
files_to_s3 = PythonOperator(  
    task_id="files_to_s3",  
    python_callable=download_zip)
```

Task

S3SensorKey

```
check_s3_for_key = S3KeySensor(  
    task_id='check_s3_for_key',  
    bucket_key=check_s3,  
    wildcard_match=True,  
    bucket_name=s3_bucket_name,  
    aws_conn_id='aws_default',  
    timeout=20,  
    poke_interval=5)
```

Building our data pipeline

Demo

Ingest into data
lake

Task

AWSAthenaOperator

```
create_athena_movie_table = AWSAthenaOperator(  
    task_id="create_athena_movie_table",  
    query=create_athena_movie_table_query,  
    database=athena_db,  
    output_location='s3:///'+s3_bucket_name+"/"+a  
thena_results+'create_athena_movie_table')
```

Building our data pipeline

Demo

Ingest into data
lake

Task

AWSAthenaOperator

```
create_athena_movie_table_query="""
CREATE EXTERNAL TABLE IF NOT EXISTS
{database}.ML_Latest_Small_Movies (
`movieId` int,`title` string,`genres` string )
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('serialization.format' =
',','field.delim' = ',')
LOCATION 's3://{s3_bucket_name}/{s3_key}ml-latest-
small/movies.csv/ml-latest-small/'
TBLPROPERTIES (
'has_encrypted_data'='false',
'skip.header.line.count'='1'
);
""".format(database=athena_db, s3_bucket_name=s3_bucket_name,
s3_key=s3_key)
```


Building our data pipeline

Demo

Merge and
consolidate data

Task

AWSAthenaOperator

```
join_athena_tables = AWSAthenaOperator(  
    task_id="join_athena_tables",  
    query=join_tables_athena_query,  
    database=athena_db,  
    output_location='s3://' + s3_bucket_name + "/" + a  
thena_results + 'join_athena_tables'  
)
```

Building our data pipeline

Demo

Clean data

Task

PythonOperator

```
clean_up_csv = PythonOperator(  
    task_id="clean_up_csv",  
    python_callable=clean_up_csv_fn,  
    provide_context=True )
```

Building our data pipeline

Demo

Copy into our
data warehouse

Task

S3ToRedshiftOperator

```
task_transfer_s3_to_redshift = S3ToRedshiftOperator(  
    s3_bucket=s3_bucket_name,  
    s3_key="athena-results/join_athena_tables/{{  
task_instance.xcom_pull(task_ids='join_athena_tables',  
key='return_value') }}_clean.csv",  
    schema='PUBLIC',  
    redshift_conn_id='redshift_default',  
    table=redshift_table_name,  
    copy_options=['csv IGNOREHEADER 1'],  
    task_id='transfer_s3_to_redshift',)
```

Demo



Apache Airflow evolution

Optimisations around the scheduler utilisation and performance

Optimisations around the scheduler utilisation and performance

More options for asynchronous workflows

Writing cleaner DAGs (closer to Python)

Improvements to the UI



<https://airflowsummit.org/>



Contributing to Apache Airflow

The Apache Airflow community makes contributing to this project both a welcoming and straight forward experience.

<https://dev.to/aws>






Ricardo Sueiras for AWS

Posted on Mar 11

EditManageStats

Contributing to the Apache Airflow project - part two



Ricardo Sueiras for AWS

Posted on Feb 10 • Originally published at blog.beachgeek.co.uk

EditManageStats

Contributing to the Apache Airflow project

#opensource

Contributing to Apache Airflow

Introduction

In this series of posts, I am going to share what I learn as embark on my first upstream contribution to the Apache Airflow project. The purpose is to show you how typical open source projects like Apache Airflow work, how you engage with the community to orchestrate change and hopefully inspire more people to contribute to this open source project. I will post regular updates as a series of posts, as the journey unfolds.

But as always, we need to set the stage and start with our reason for doing so.

The problem

In a previous post ([Creating a multi architecture CI/CD solution with Amazon ECS and ECS Anywhere](#)) I set up a typical environment that you might come across with customers that are looking at a Hybrid approach to managing and deploying their workloads. This allows you to run container images anywhere

of the
ect. You can

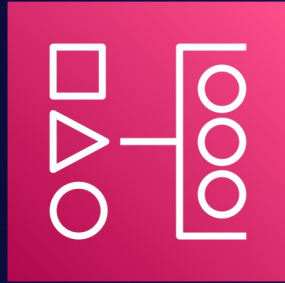
to the Apache
, our first
exploring how

works in my
r on, as I have
("), and then
the

the
)' and then
on' as I have
orks in my

deploying their workloads. This allows you to run container images anywhere across with customers that are looking at a Hybrid approach to managing and

Resources



Demo code

<https://bit.ly/3sY19fP>



Apache Airflow

<https://airflow.apache.org>

Thank you!

Ricardo Sueiras

Developer Advocate for Open Source
Amazon Web Services.



<https://www.linkedin.com/in/ricardosueiras>



@094459



<https://eventbox.dev/survey/QT0NYO6>

For a copy of these slides
please complete this very
short survey

