Apache Airflow

Orchestration

Automation

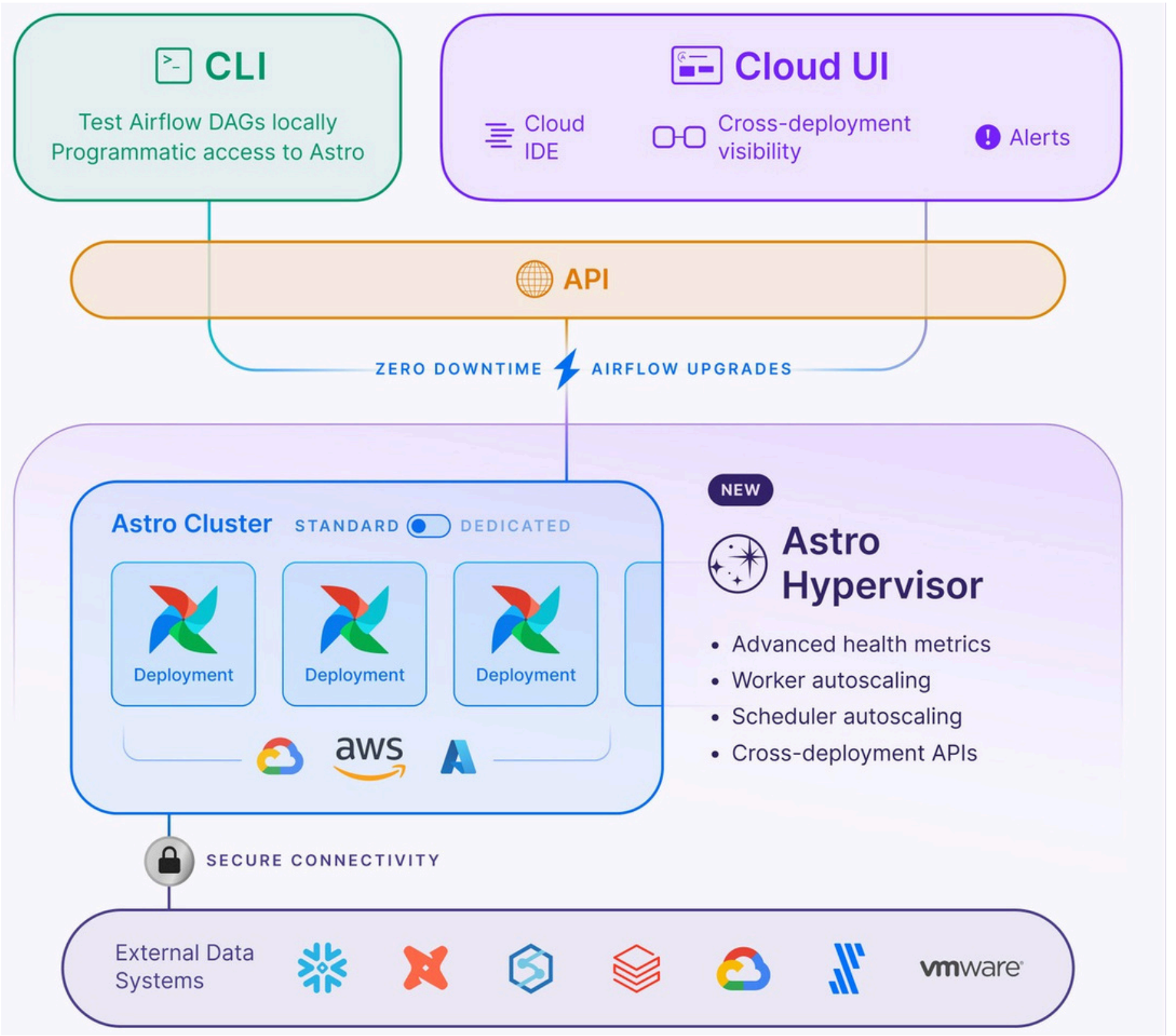
Monitoring

Scheduling

Pipeline

Python

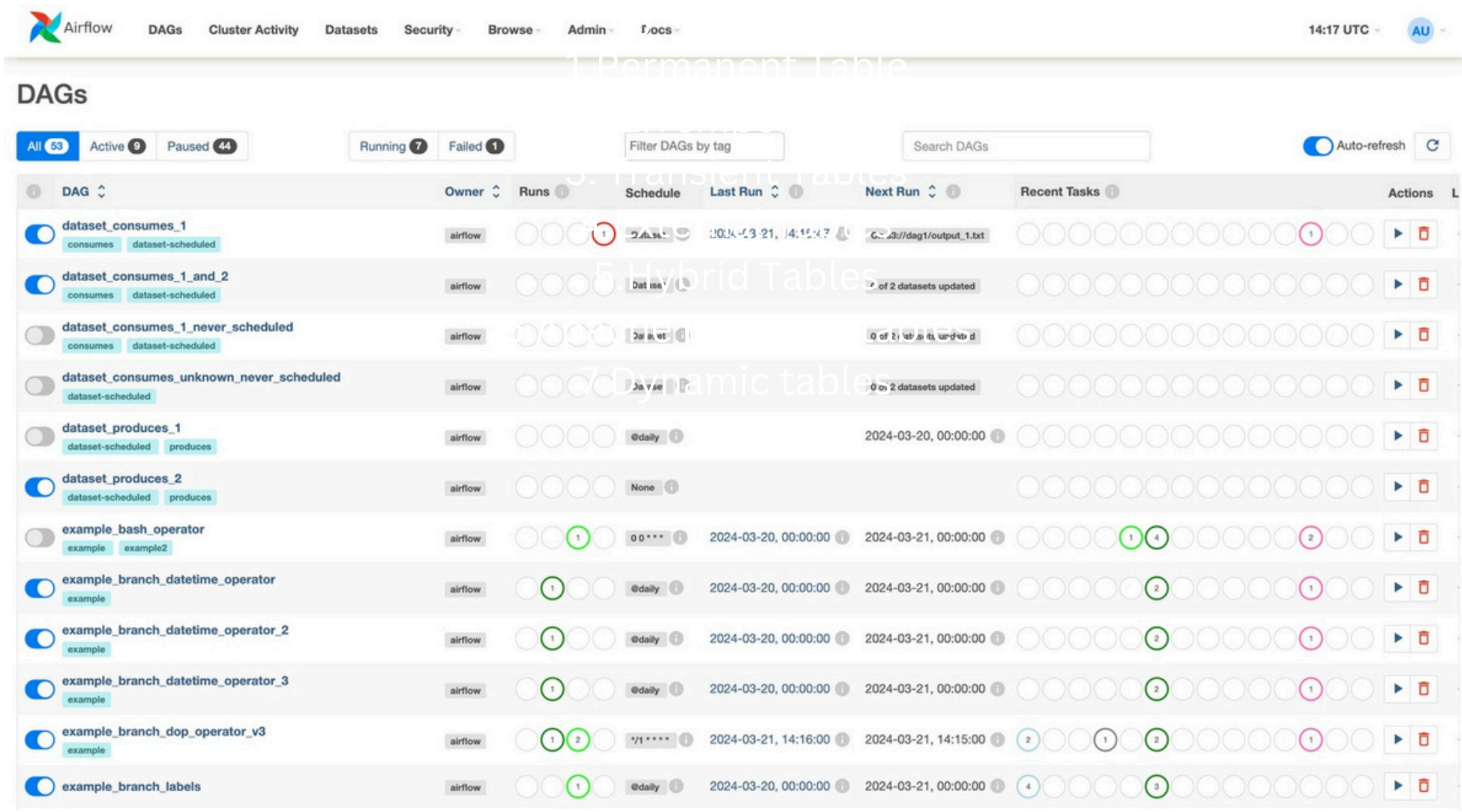
Airflow Architecture (Astro)



Apache Airflow

Apache Airflow is an open-source tool for creating, scheduling, and monitoring workflows as code, using DAGs (Directed Acyclic Graphs) to define task dependencies and manage execution, retries, and failures.

Airflow Web UI (Enable/Disable DAGS ,Monitor,Refresh)

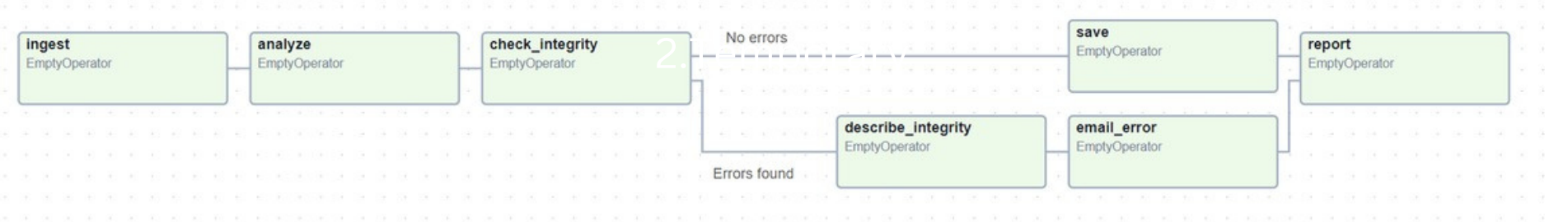


Airflow DAG

DAGs (Directed Acyclic Graphs) –

A DAG represents the overall structure of a workflow in Airflow

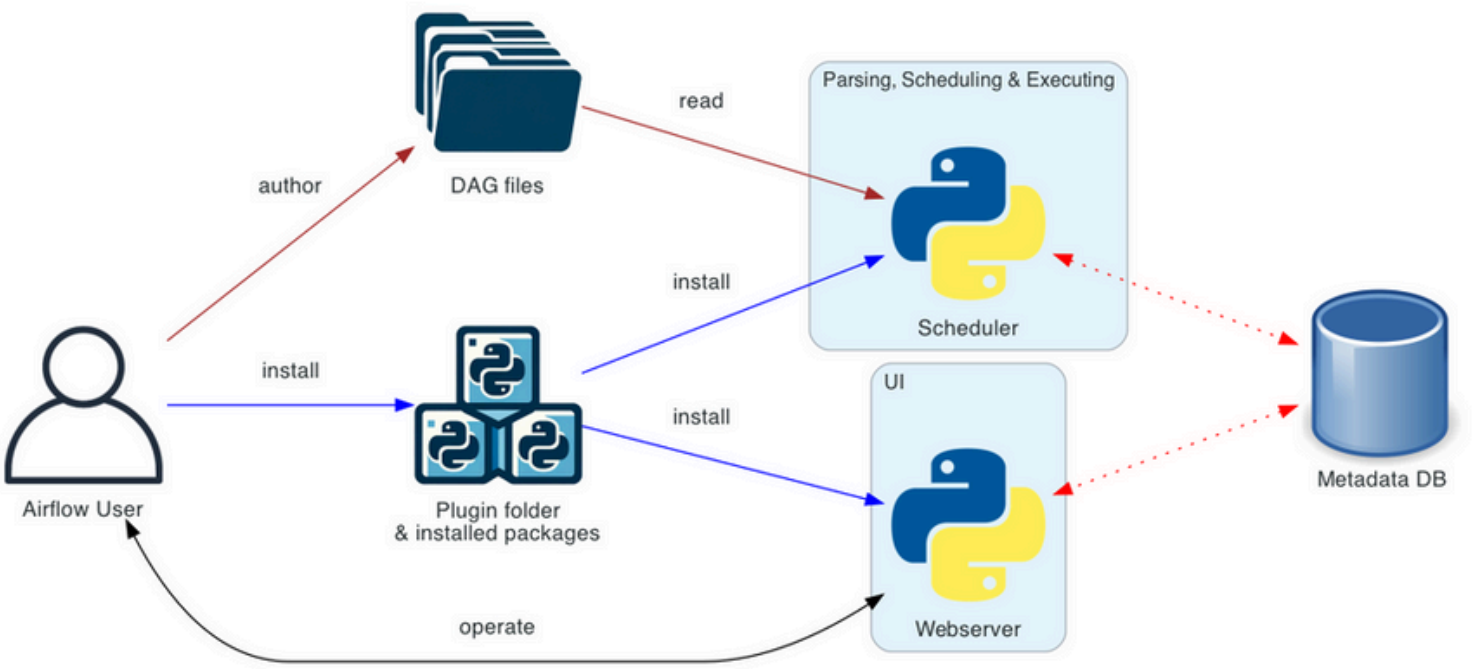
An Airflow DAG (Directed Acyclic Graph) is a collection of tasks in a workflow, with specified execution order and dependencies. Written in Python, it ensures tasks run in sequence without loops, allowing users to define complex workflows ,including scheduling and logic.



A DAG defines task dependencies, determining their execution order, while tasks specify the actions, such as fetching data, running analyses, or triggering other systems.

Airflow Architecture

The diagrams below show different ways to deploy Airflow - gradually from the simple “one machine” and single person deployment, to a more complex deployment with separate components, separate user roles and finally with more isolated security perimeters.



Create a DAG

Design a Basic DAG -

It defines four Tasks - A, B, C, and D - and dictates the order in which they have to run, and which tasks depend on what others. It will also say how often to run the DAG - maybe “every 5 minutes starting tomorrow”, or “every day since January 1st, 2020”.

Python program DAG

Part 1//2

```python
# Import the necessary modules from the Airflow library
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta
import time

# Define a Python function to print the current date
def print_current_date():
    print(f"Current date and time is: {datetime.now()}")

# Define a Python function to simulate sleeping for 5 seconds
def sleep_for_seconds():
    print("Sleeping for 5 seconds...")
    time.sleep(5)
    print("Woke up after 5 seconds")

# Default arguments that will be passed to each task in the DAG
# These settings are applied to the entire DAG and can be overridden for individual tasks
default_args = {
    'owner': 'airflow',  # Owner of the DAG
    'depends_on_past': False,  # Don't depend on previous runs
    'start_date': datetime(2024, 10, 18),  # Start date of the DAG
    'email_on_failure': False,  # Do not send an email when a task fails
    'email_on_retry': False,  # Do not send an email when a task retries
    'retries': 1,  # Number of retry attempts in case of failure
    'retry_delay': timedelta(minutes=5),  # Wait 5 minutes between retries
}
```

Python program DAG

Part 2//2

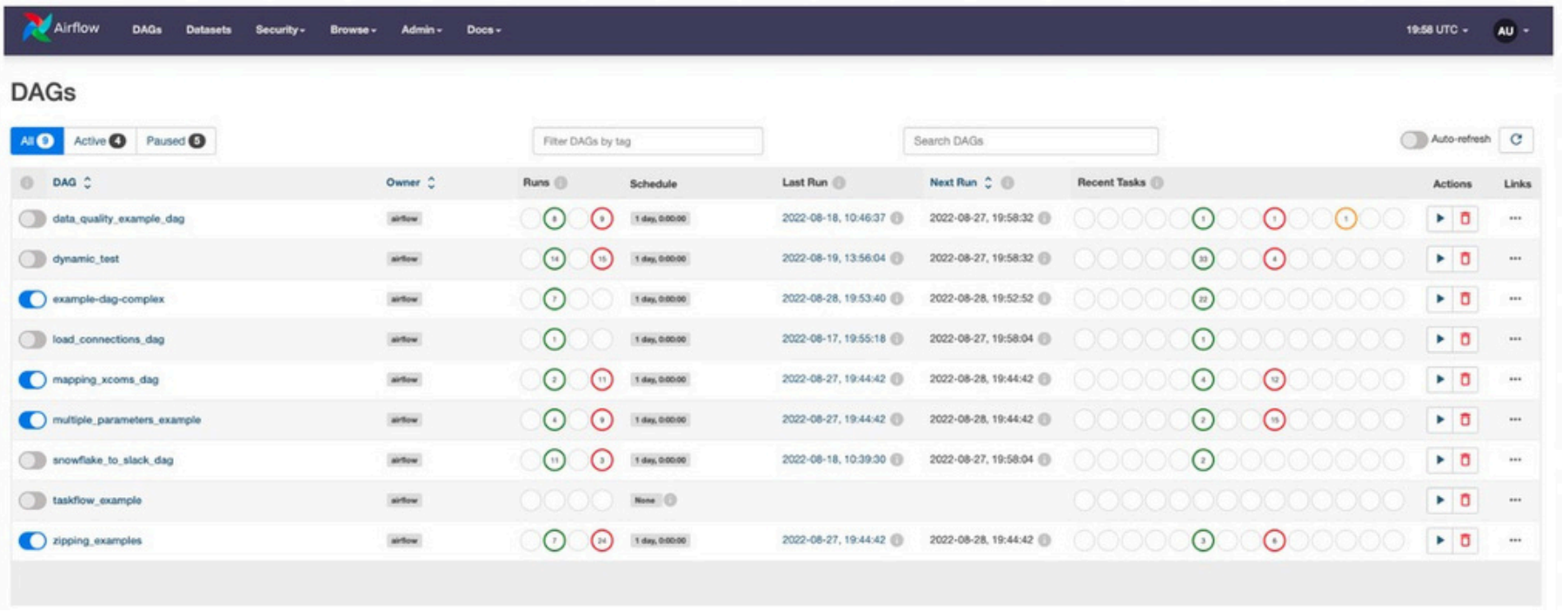
```
# Initialize the DAG (Directed Acyclic Graph)
# The DAG object is created with a unique name, default arguments, and a schedule interval
with DAG(
    'simple_dag',  # Name of the DAG
    default_args=default_args,  # Default arguments for the DAG
    description='A simple tutorial DAG',  # Description of the DAG
    schedule_interval=timedelta(days=1),  # The DAG will run once every day
) as dag:

    # Define the first task, which prints the current date
    # We use PythonOperator to run a Python function as a task
    task1 = PythonOperator(
        task_id='print_date',  # Unique task identifier
        python_callable=print_current_date,  # The function to be executed
    )

# Define the second task, which will sleep for 5 seconds
    task2 = PythonOperator(
        task_id='sleep_task',  # Unique task identifier
        python_callable=sleep_for_seconds,  # The function to be executed
    )

    # Set task dependencies (task1 will run first, then task2)
    task1 >> task2  # This means task1 must complete before task2 starts
```

Then Deploy python code in GIT/AWS S3 based on repo (it’s very simple)

DAG will display in Airflow UI once you place DAG code

Ways to View and Manage DAGs in Airflow

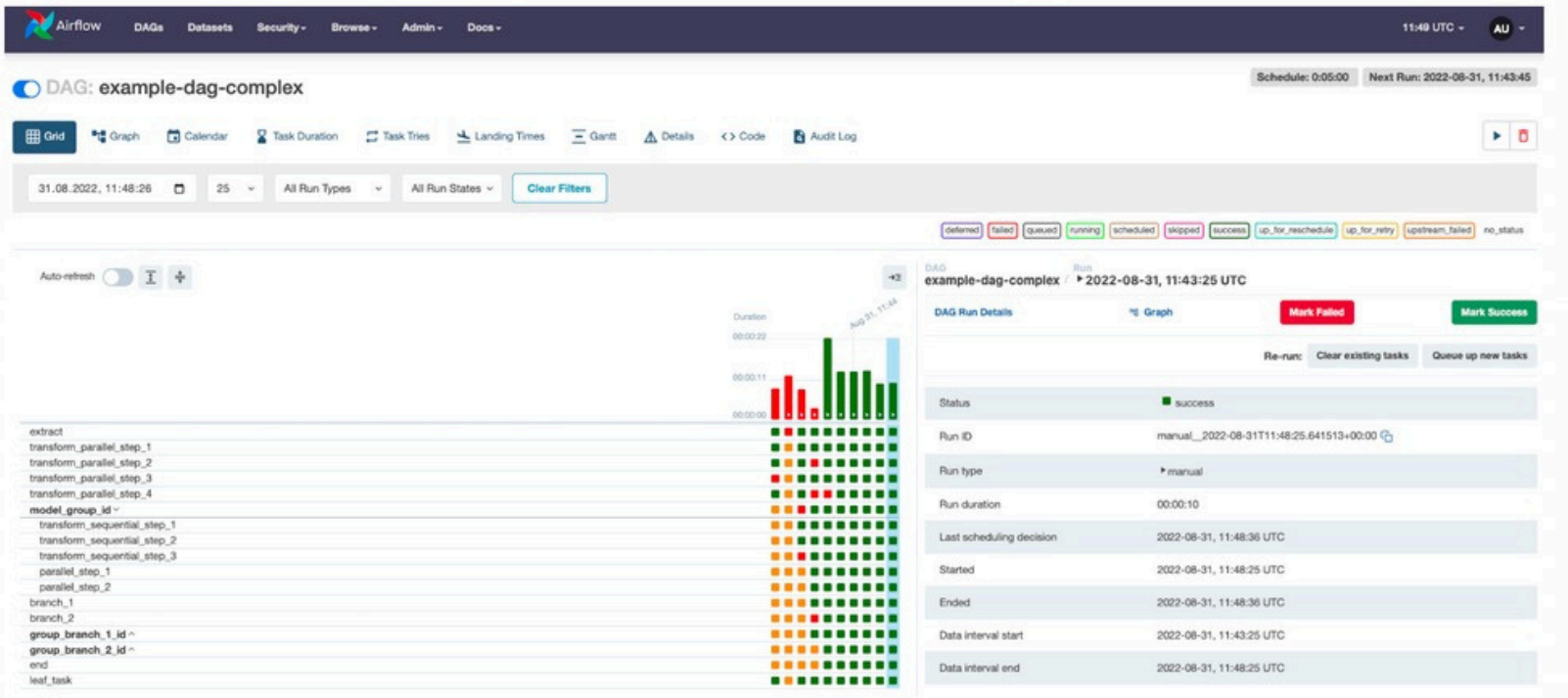
Airflow DAGs View



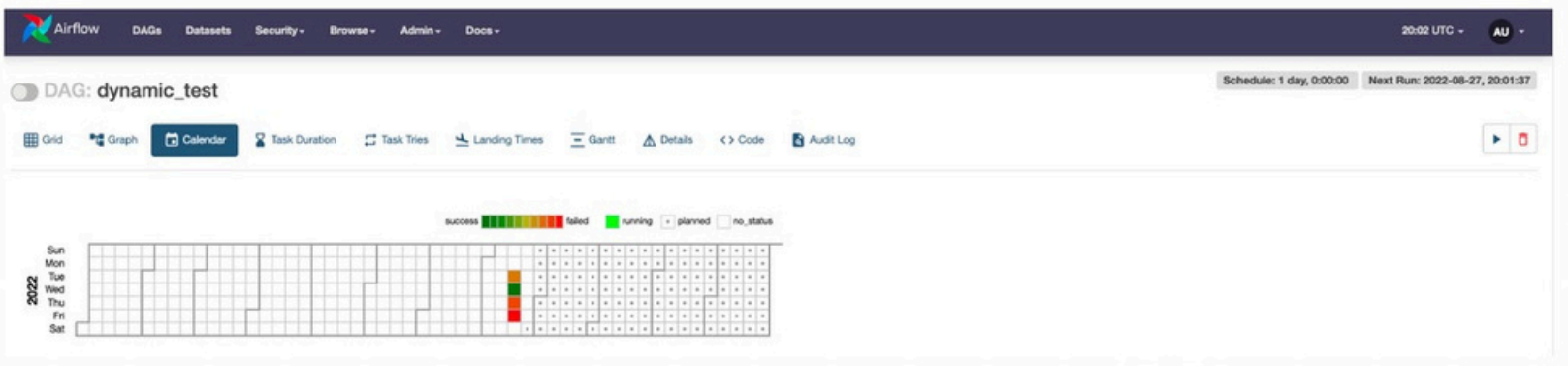
Airflow Graph View

Ways to View and Manage DAGs in Airflow

Airflow Grid View



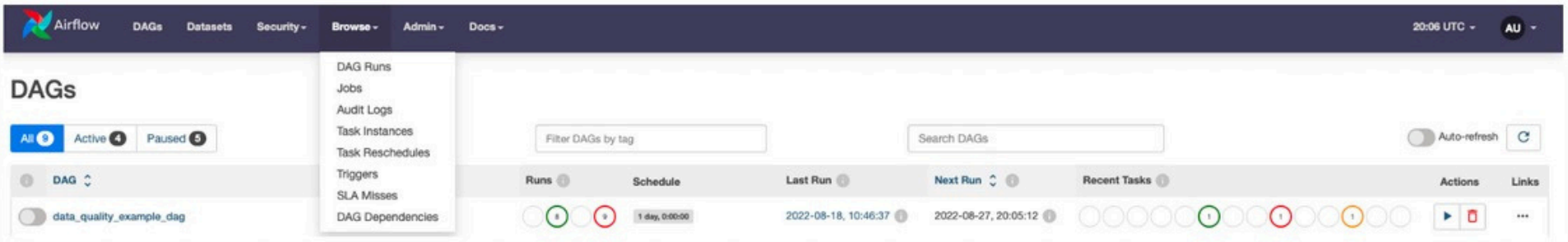
Airflow Calendar View




Anil Patel
@aniltppatel

Ways to View and Manage DAGs in Airflow

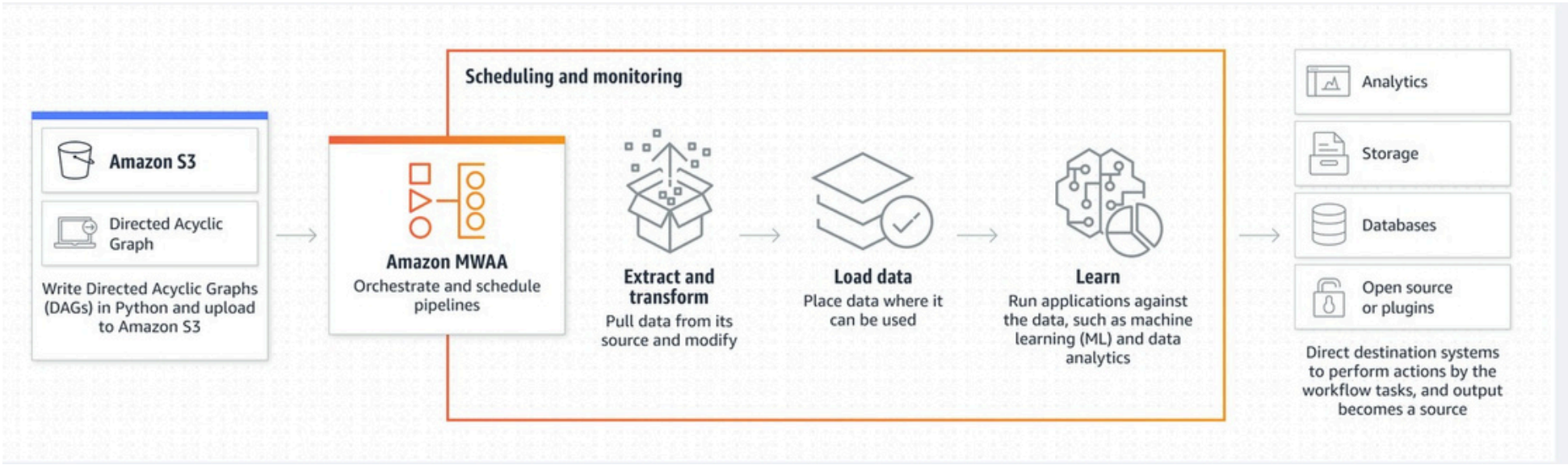
Airflow Browse Tab



cloud providers that offer Airflow services

Amazon Web Services (AWS) -

Amazon Managed Workflows for Apache Airflow (MWAA) is a fully managed Airflow service on AWS



cloud providers that offer Airflow services

Microsoft Azure –

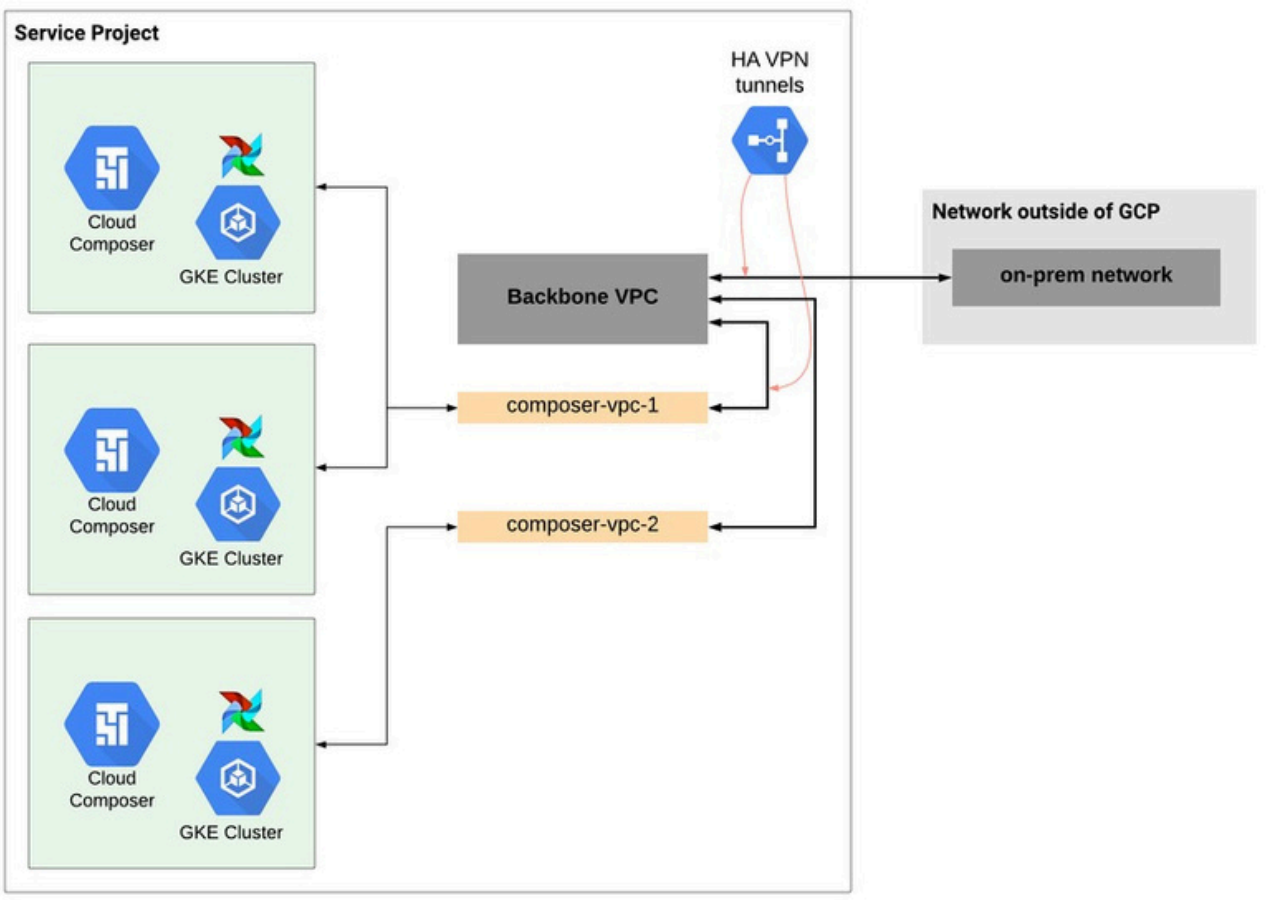
Apache Airflow integrates with Microsoft Azure services through Microsoft. azure provider. You can install any provider package by editing the airflow environment from the Azure Data Factory UI

Apache Airflow™ on Astro – An Azure Native ISV Service

cloud providers that offer Airflow services

Google Cloud Platform (GCP)-

Cloud Composer is a fully managed workflow orchestration service that runs on GCP. It's built on the Apache Airflow open source project



cloud providers that offer Airflow services

Astronomer -

A managed Airflow service that allows data teams to build, run, and manage data pipelines as code. Astronomer can run Airflow on AWS, GCP, Azure, or on-premise

Introduction to Apache Airflow

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

What is data engineering?

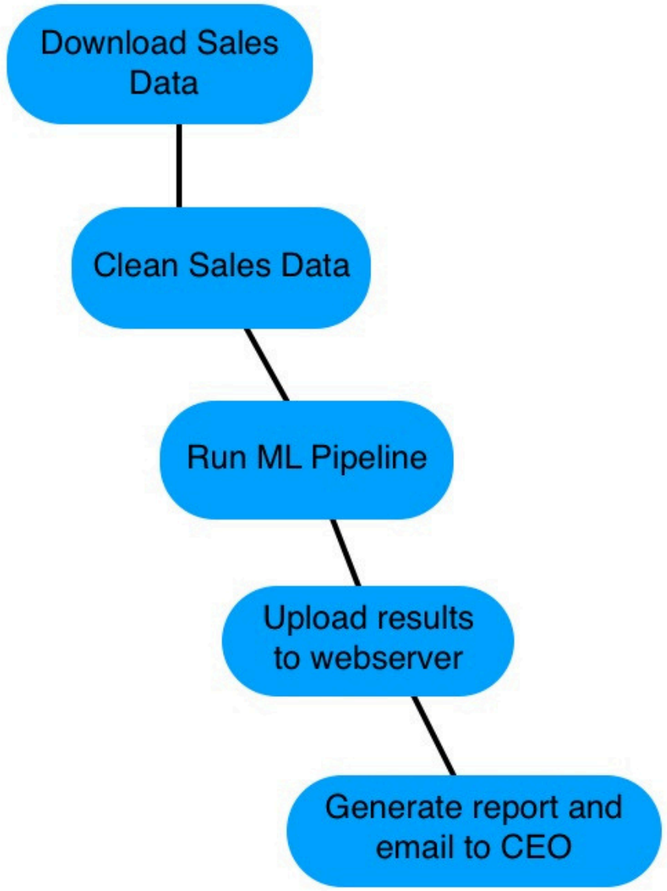
Data engineering is:

- Taking any action involving data and turning it into a reliable, repeatable, and maintainable process.

What is a workflow?

A *workflow* is:

- A set of steps to accomplish a given data engineering task
  - Such as: downloading files, copying data, filtering information, writing to a database, etc
- Of varying levels of complexity
- A term with various meaning depending on context



What is Airflow?

Airflow is a platform to program workflows, including:

- Creation
- Scheduling
- Monitoring

Airflow continued...

- Can implement programs from any language, but workflows are written in Python
- Implements workflows as DAGs: Directed Acyclic Graphs
- Accessed via code, command-line, or via web interface / REST API

[1] https://airflow.apache.org/docs/stable/

Other workflow tools

Other tools:

- **ADF**
- **SSIS**
- **Dagster**

Prefect

Mage

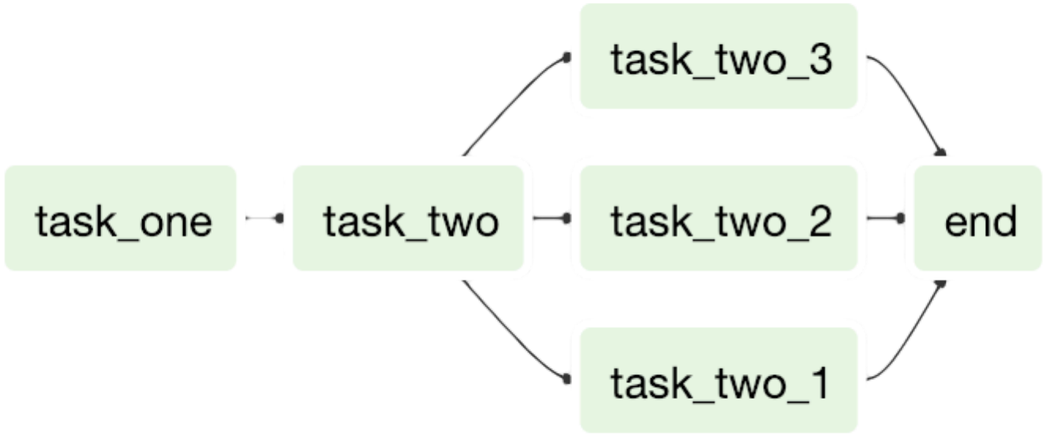
Apache Oozie

Informatica

Quick introduction to DAGs

A *DAG* stands for *Directed Acyclic Graph*

- In Airflow, this represents the set of tasks that make up your workflow.
- Consists of the tasks and the dependencies between tasks.
- Created with various details about the DAG, including the name, start date, owner, etc.
- Further depth in the next lesson.



DAG code example

Simple DAG definition:

```
etl_dag = DAG(
    dag_id='etl_pipeline',
    default_args={"start_date": "2024-01-08"}
)
```

Running a workflow in Airflow

Running a simple Airflow task

```
airflow tasks test <dag_id> <task_id> [execution_date]
```

Using a DAG named *example-etl* , a task named *download-file* on 2024-01-10:

```
airflow tasks test example-etl download-file 2024-01-10
```

Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

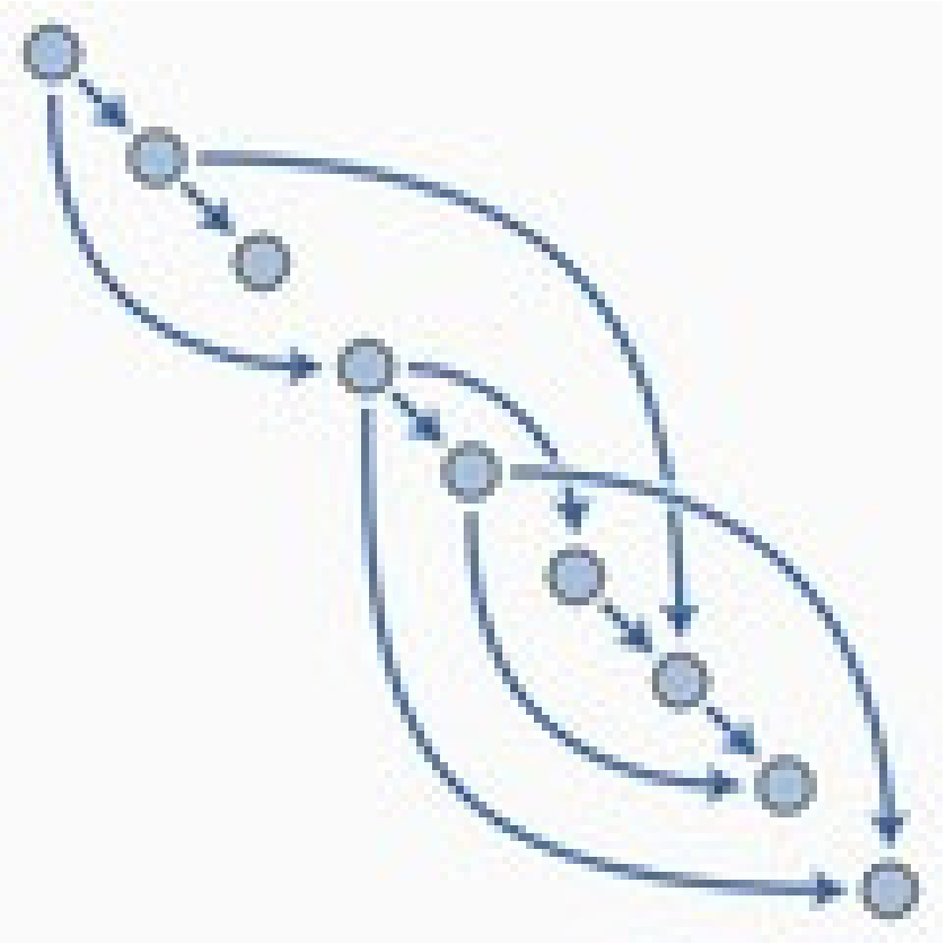
Airflow DAGs

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

What is a DAG?

DAG, or *Directed Acyclic Graph*:

- *Directed*, there is an inherent flow representing dependencies between components.
- *Acyclic*, does not loop / cycle / repeat.
- *Graph*, the actual set of components.
- Seen in Airflow, Apache Spark, dbt



DAG in Airflow

Within Airflow, DAGs:

- Are written in Python (but can use components written in other languages).
- Are made up of components (typically *tasks*) to be executed, such as operators, sensors, etc.
- Contain dependencies defined explicitly or implicitly.
  - ie, Copy the file to the server before trying to import it to the database service.

Define a DAG

Example DAG:

```
from airflow import DAG

from datetime import datetime
default_arguments = {
  'owner': 'jdoe',
  'email': 'jdoe@datacamp.com',
  'start_date': datetime(2020, 1, 20)
}

with DAG('etl_workflow', default_args=default_arguments ) as etl_dag:
```

Define a DAG (before Airflow 2.x)

Example DAG:

```
from airflow import DAG

from datetime import datetime
default_arguments = {
  'owner': 'jdoe',
  'email': 'jdoe@datacamp.com',
  'start_date': datetime(2020, 1, 20)
}

etl_dag = DAG('etl_workflow', default_args=default_arguments )
```

DAGs on the command line

Using `airflow` :

- The `airflow` command line program contains many subcommands.
- `airflow -h` for descriptions.
- Many are related to DAGs.
- `airflow dags list` to show all recognized DAGs.

Command line vs Python

Use the command line tool to:

- Start Airflow processes
- Manually run DAGs / Tasks
- Get logging information from Airflow

Use Python to:

- Create a DAG
- Edit the individual properties of a DAG

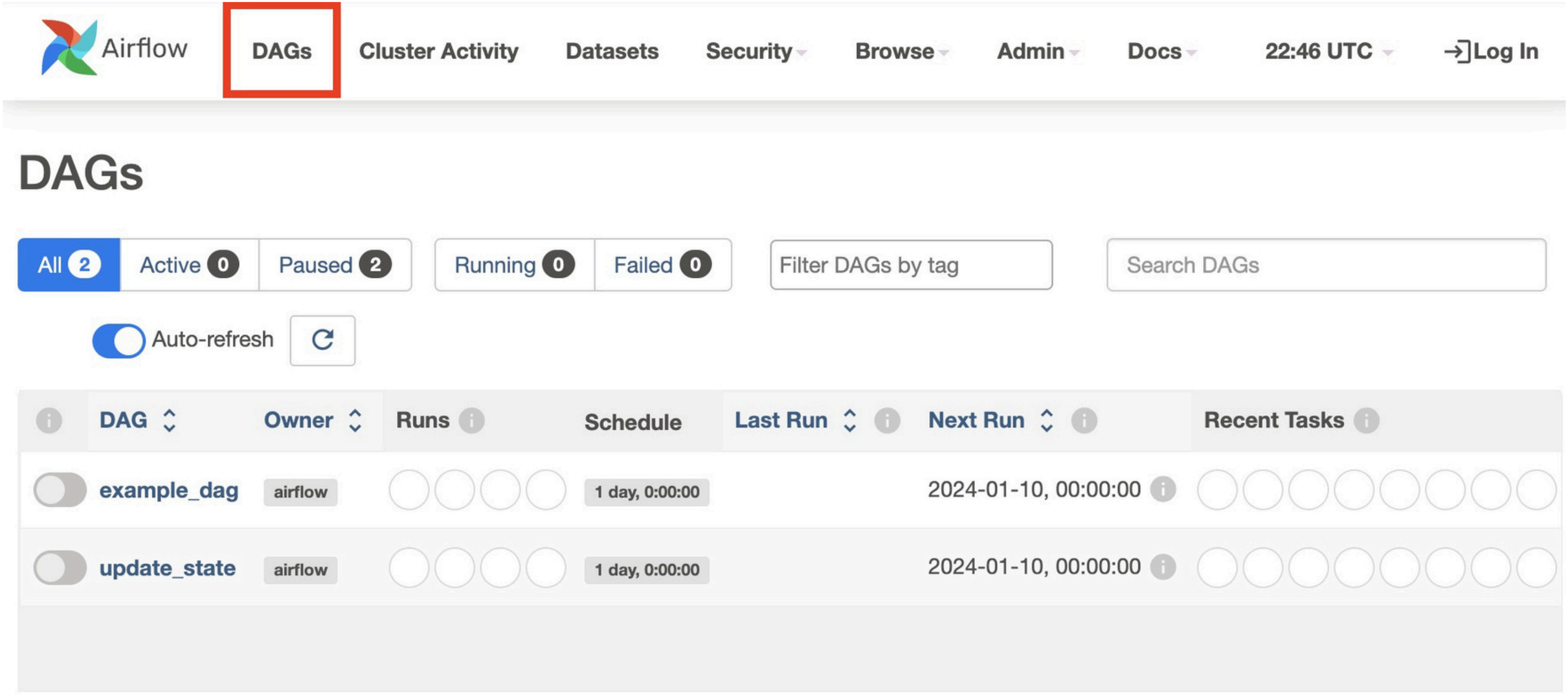
Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

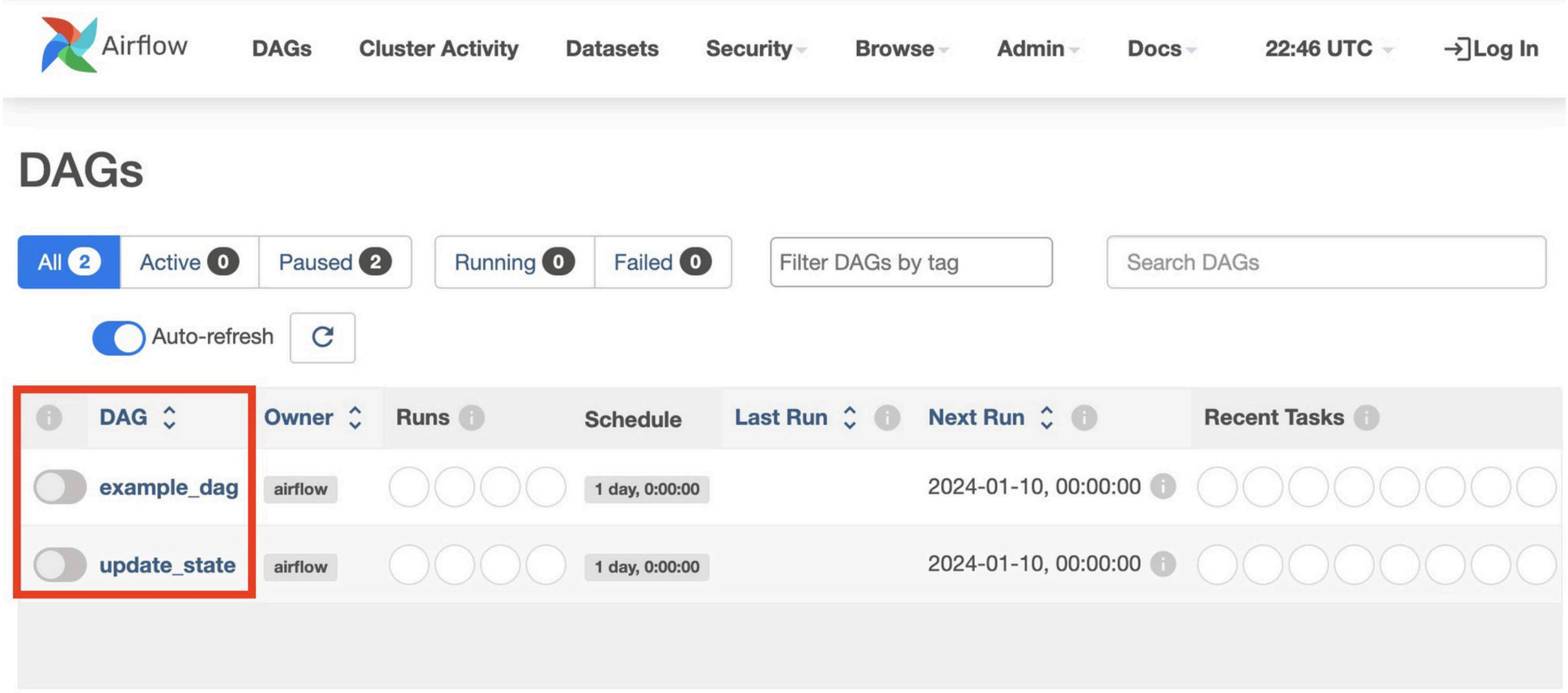
Airflow web interface

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

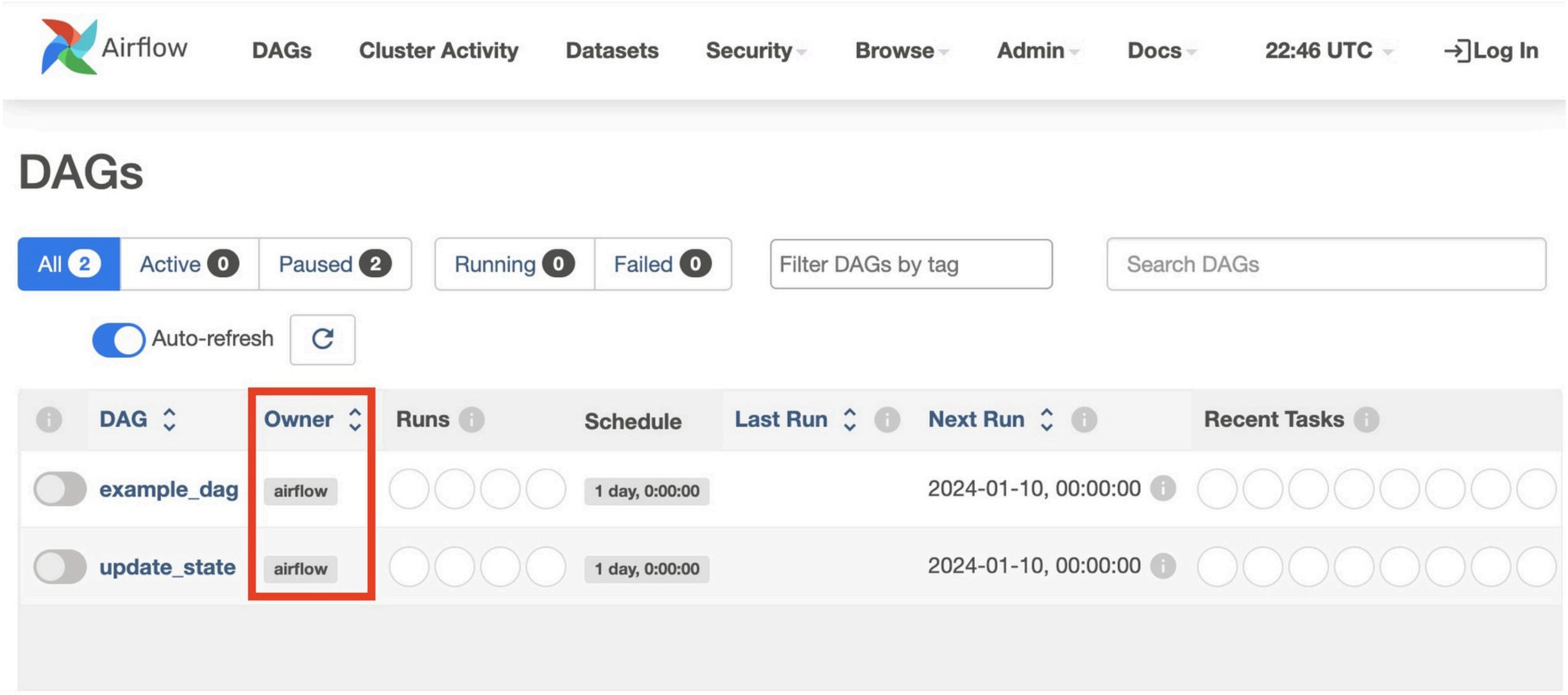
DAGs view


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
22:46 UTC
Log In
DAGs
All 2
Active 0
Paused 2
Running 0
Failed 0
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
example_dag
airflow
1 day, 0:00:00
2024-01-10, 00:00:00
update_state
airflow
1 day, 0:00:00
2024-01-10, 00:00:00

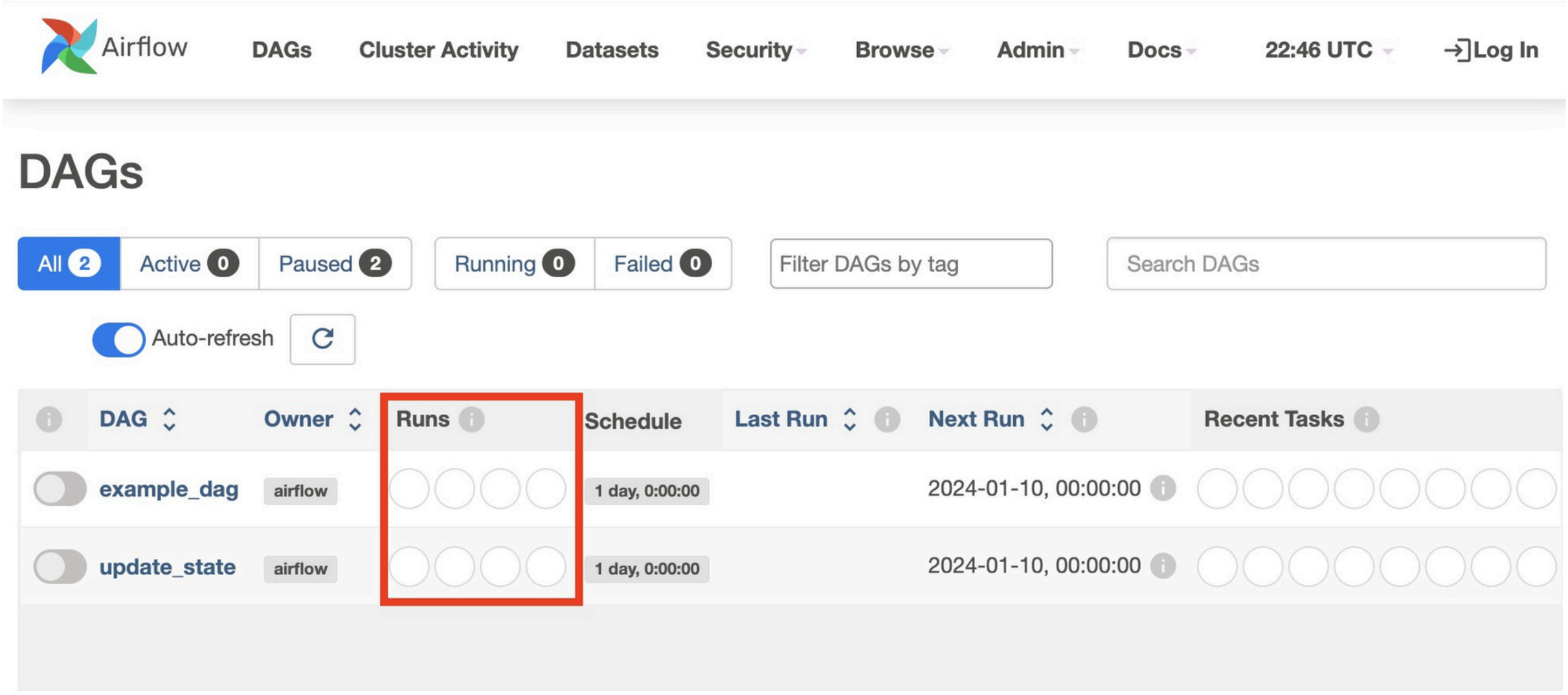
DAGs view DAGs



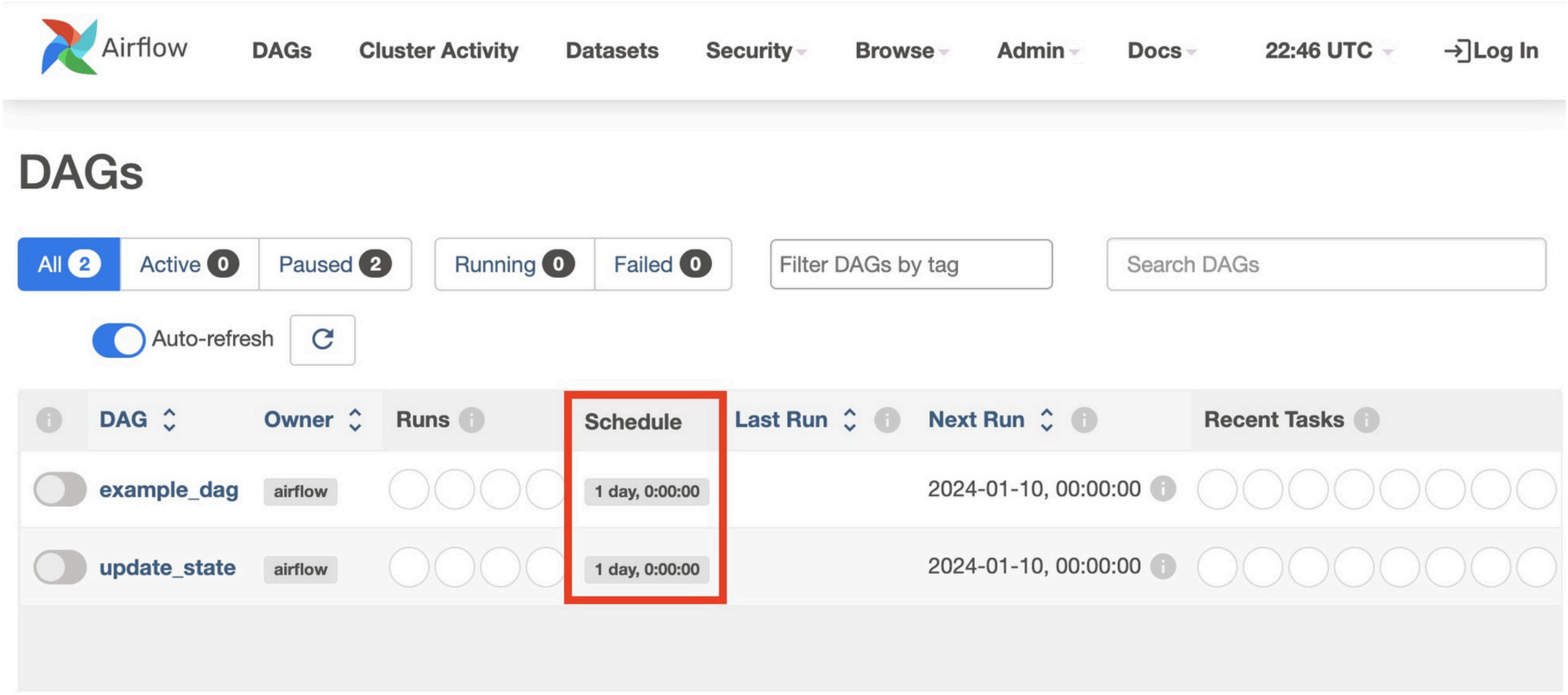
DAGs view owner



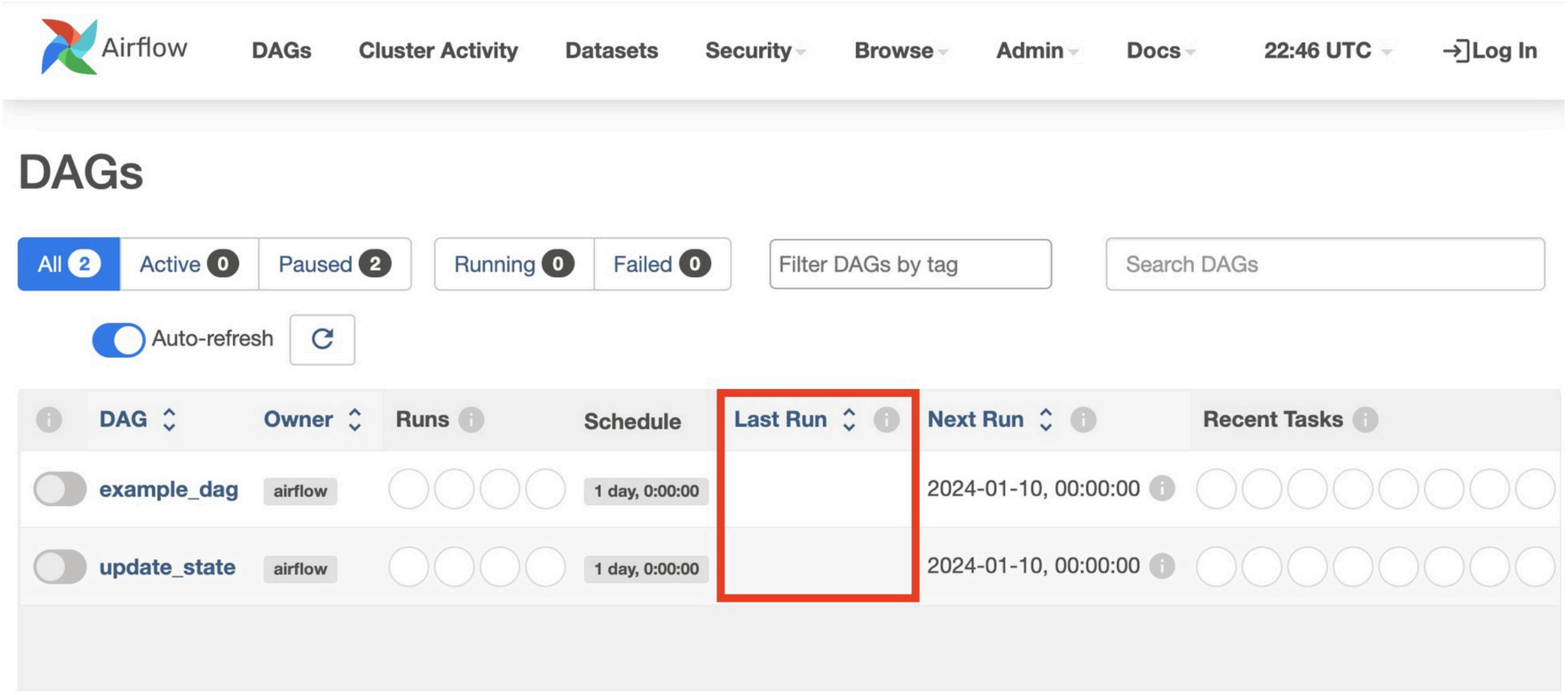
DAGs view runs



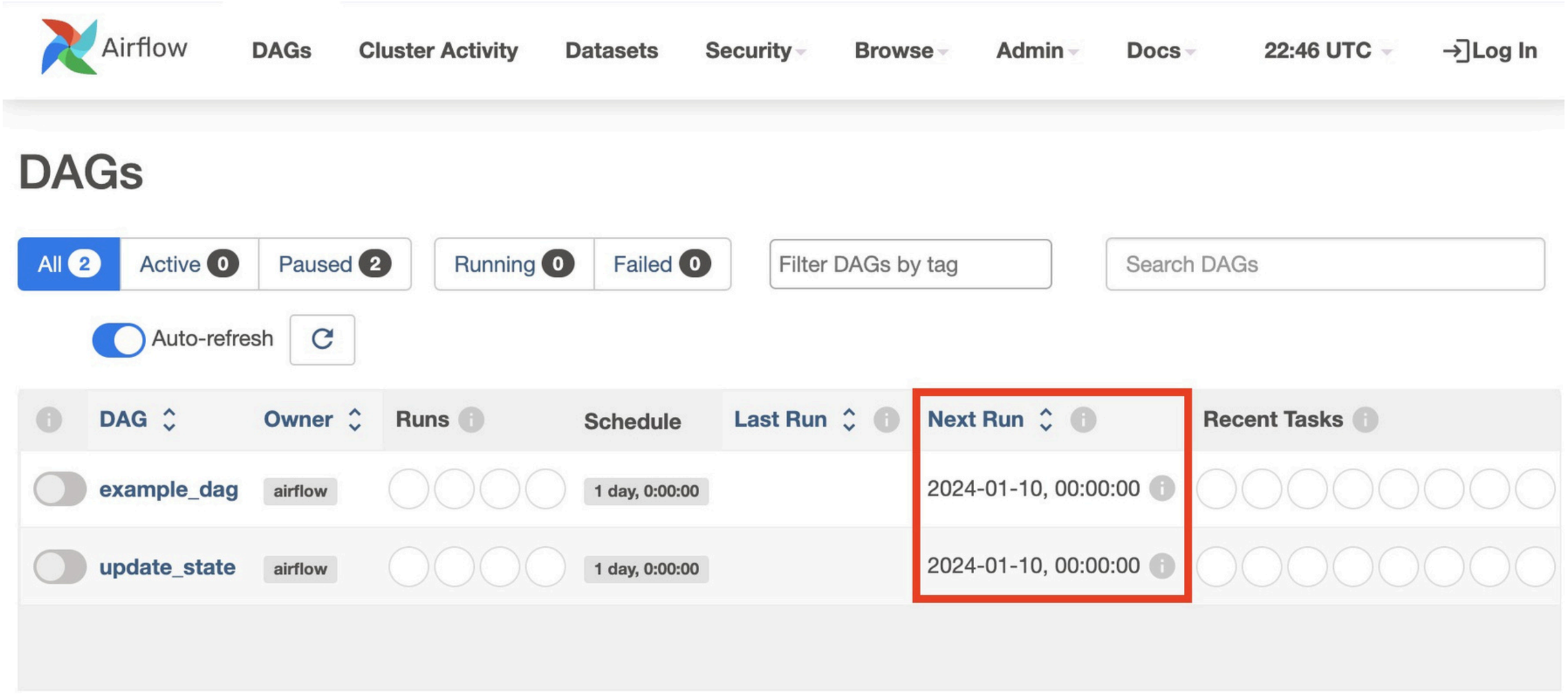
DAGs view schedule



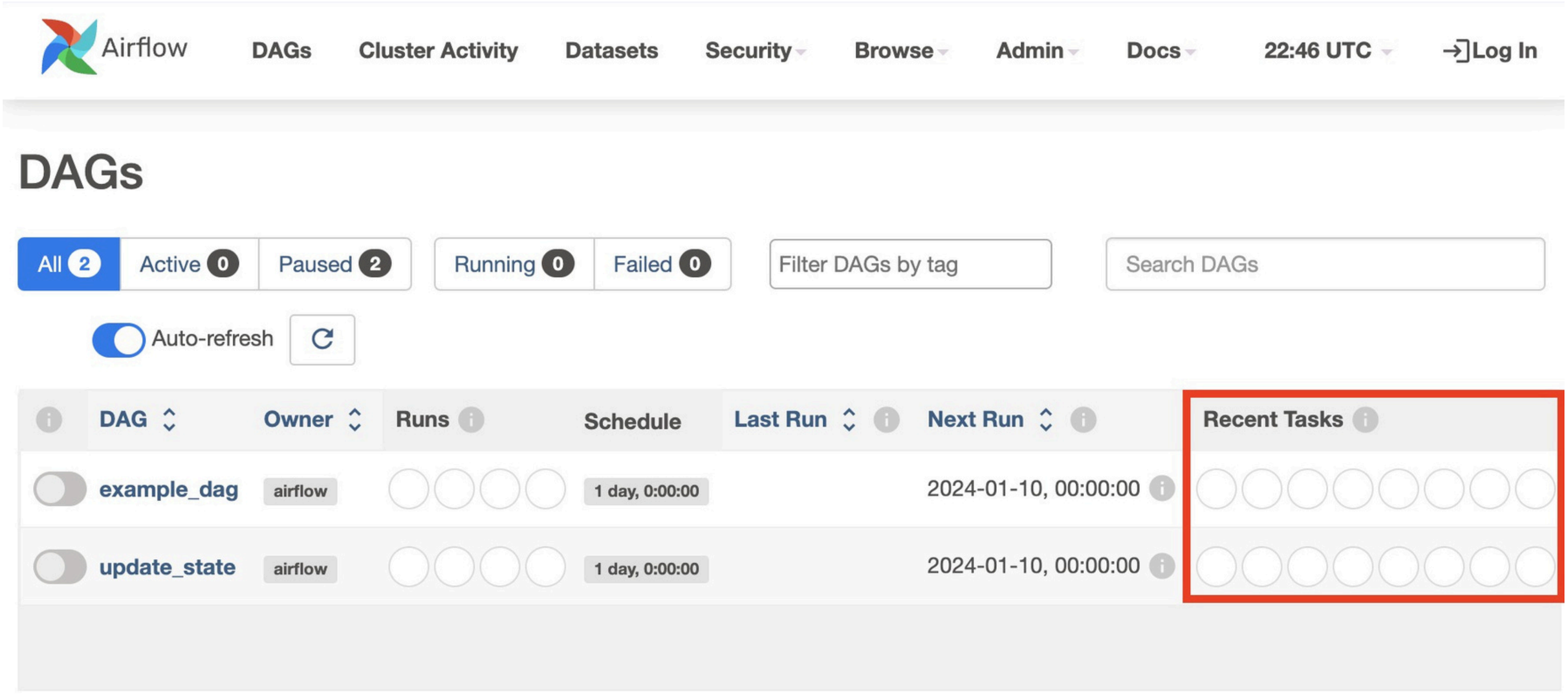
DAGs view last run



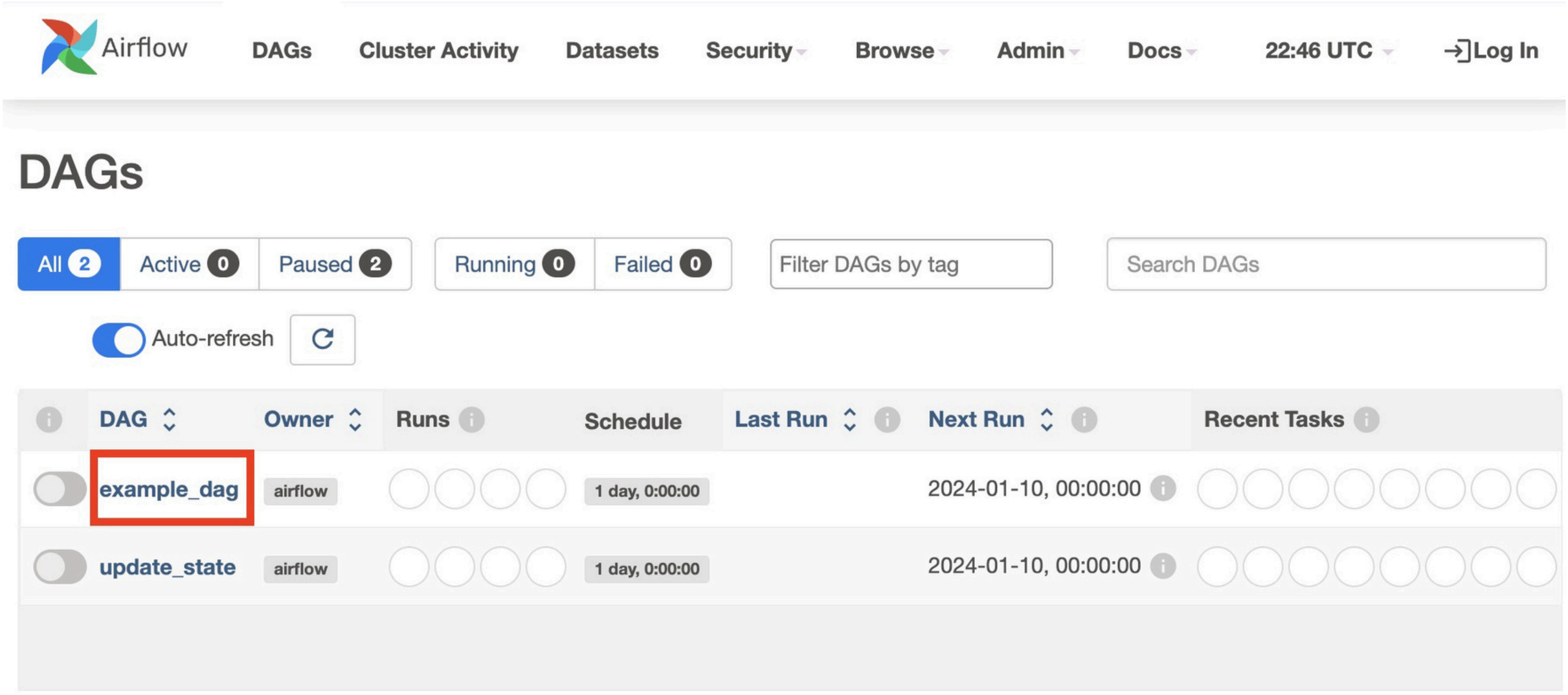
DAGs view next run



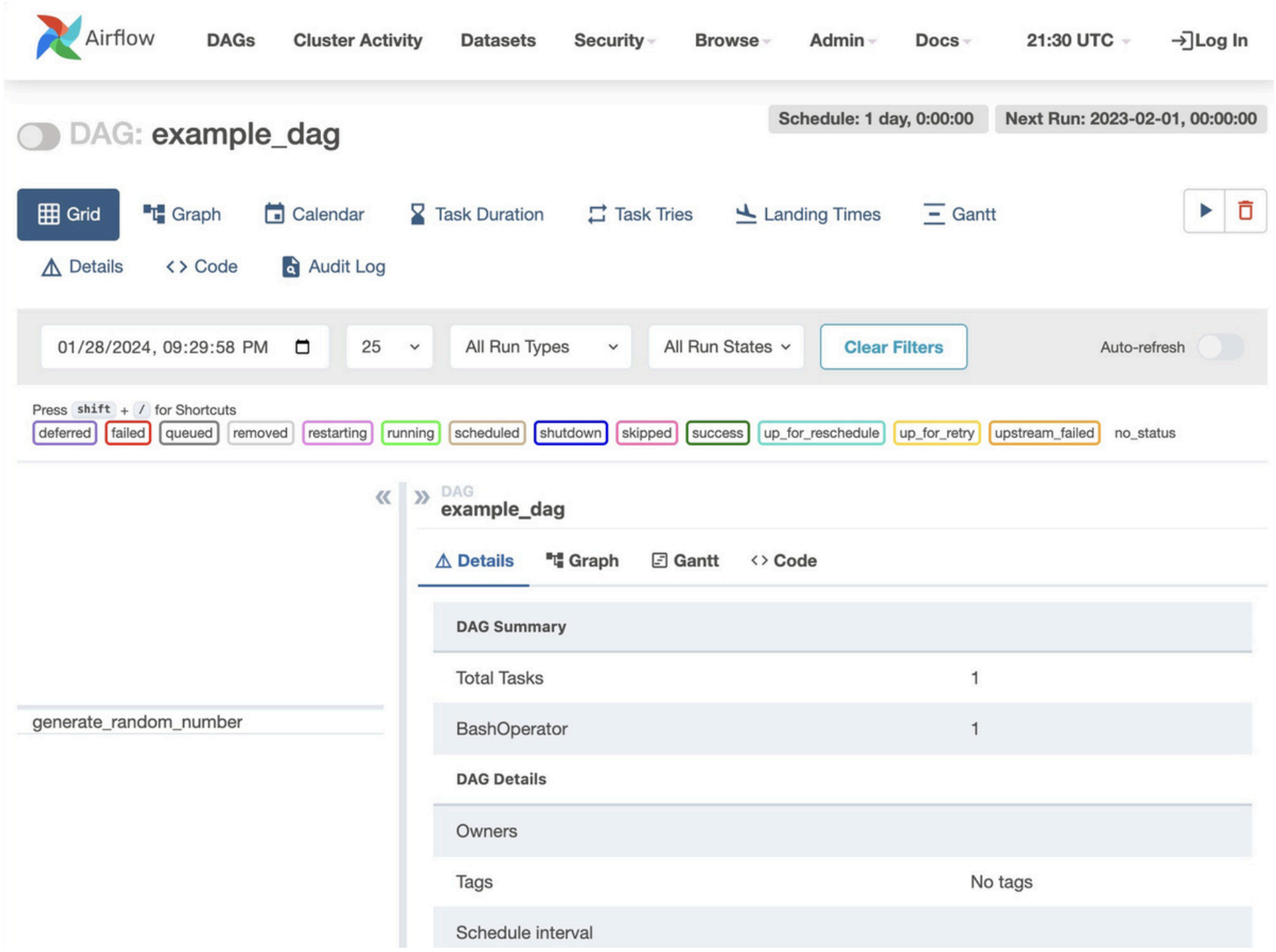
DAGs view recent tasks



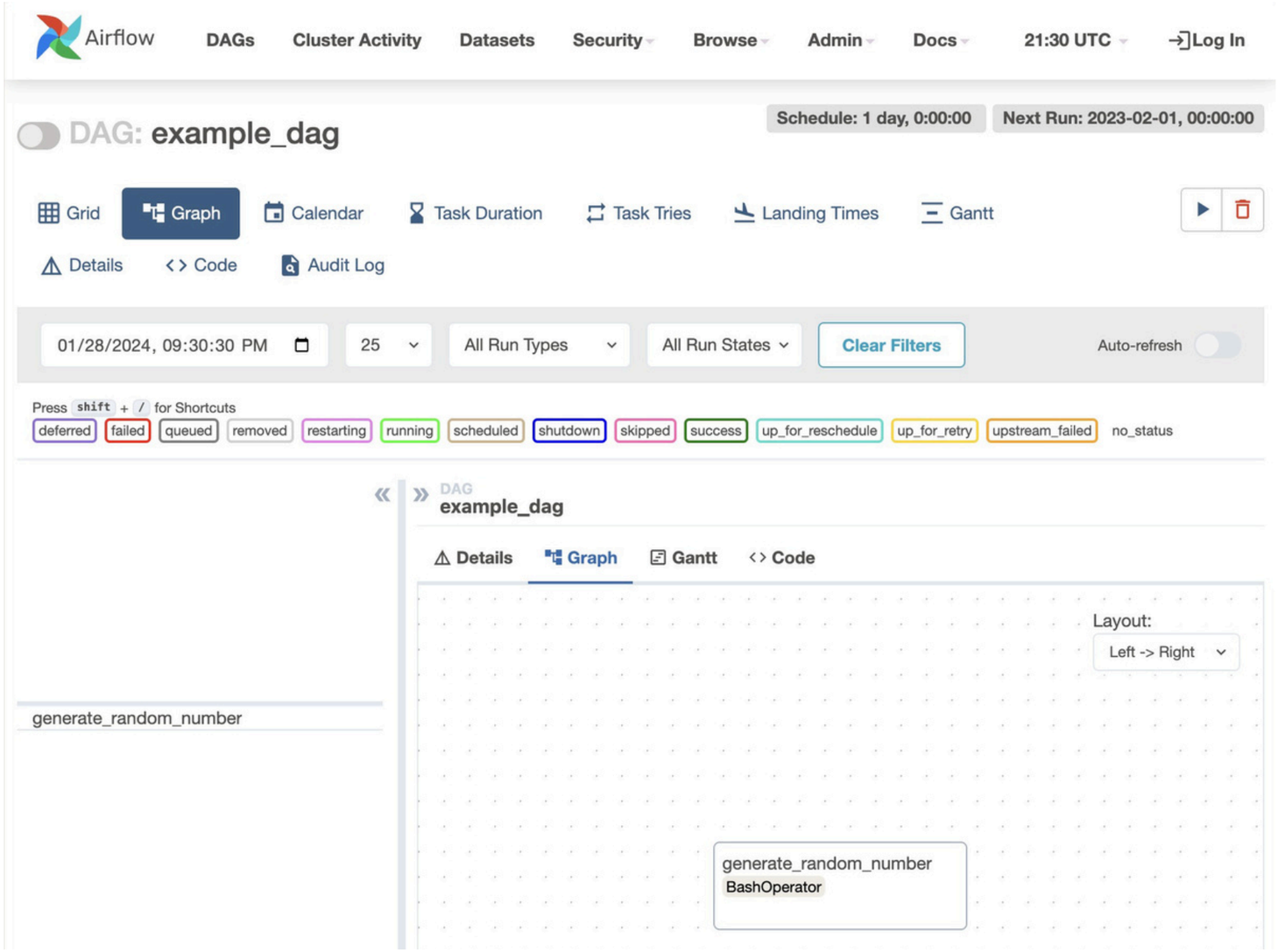
DAGs view example_dag



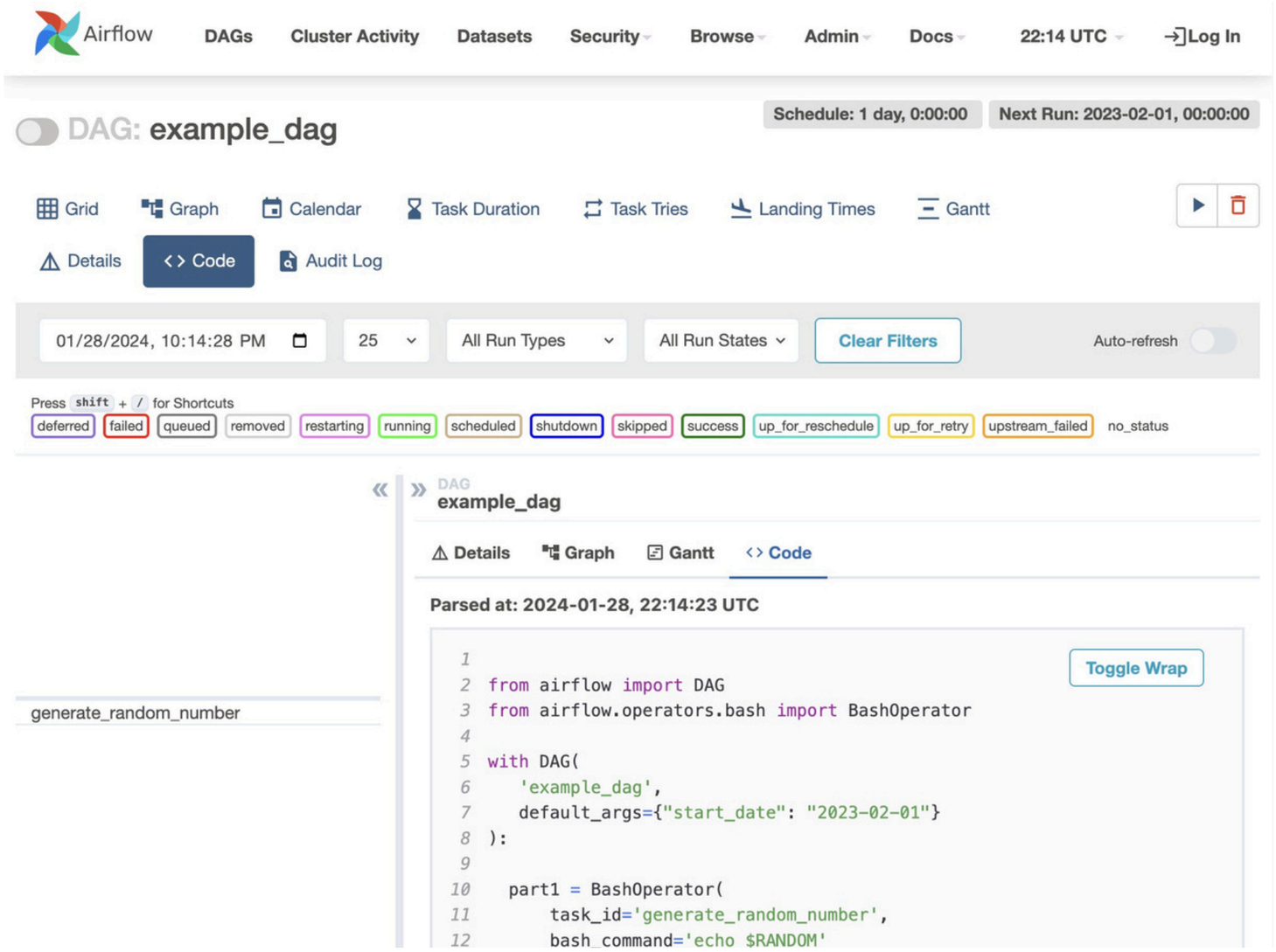
DAG detail view



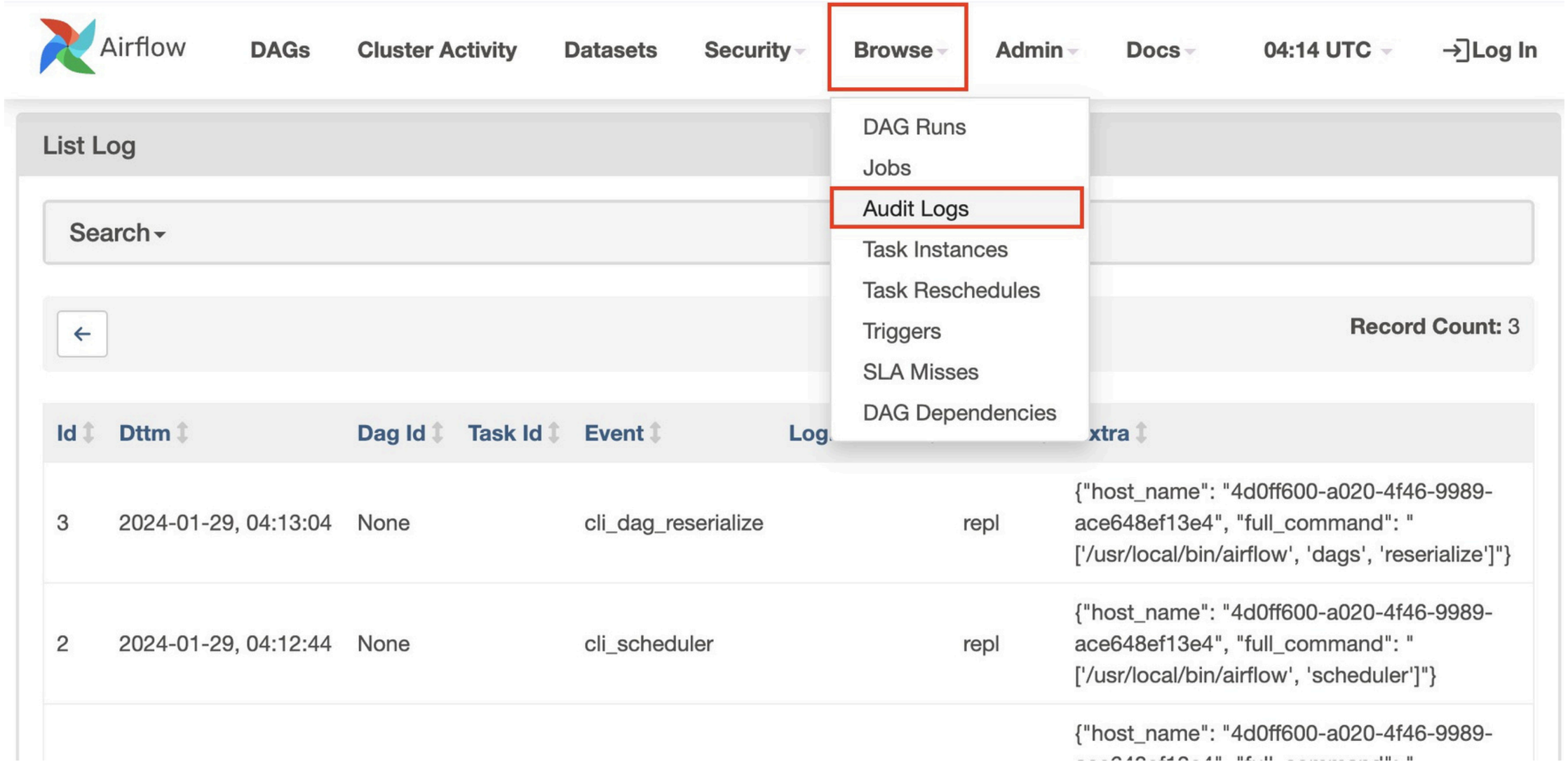
DAG graph view



DAG code view



Audit logs


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
04:14 UTC
Log In
DAG Runs
Jobs
Audit Logs
Task Instances
Task Reschedules
Triggers
SLA Misses
DAG Dependencies
List Log
Search
Record Count: 3
Id
Dttm
Dag Id
Task Id
Event
Log
xtra
3
2024-01-29, 04:13:04
None
cli_dag_reserialize
repl
{"host_name": "4d0ff600-a020-4f46-9989-ace648ef13e4", "full_command": "['/usr/local/bin/airflow', 'dags', 'reserialize']"}
2
2024-01-29, 04:12:44
None
cli_scheduler
repl
{"host_name": "4d0ff600-a020-4f46-9989-ace648ef13e4", "full_command": "['/usr/local/bin/airflow', 'scheduler']"}
{"host_name": "4d0ff600-a020-4f46-9989-

Web UI vs command line

In most cases:

- Equally powerful depending on needs
- Web UI is easier
- Command line tool may be easier to access depending on settings

Let's practice!

INTRODUCTION TO APACHE AIRFLOW IN PYTHON

Thanks for reading !

Follow my profile for more coding related contents

Like Comment Share

@rganesh0203