Airflow

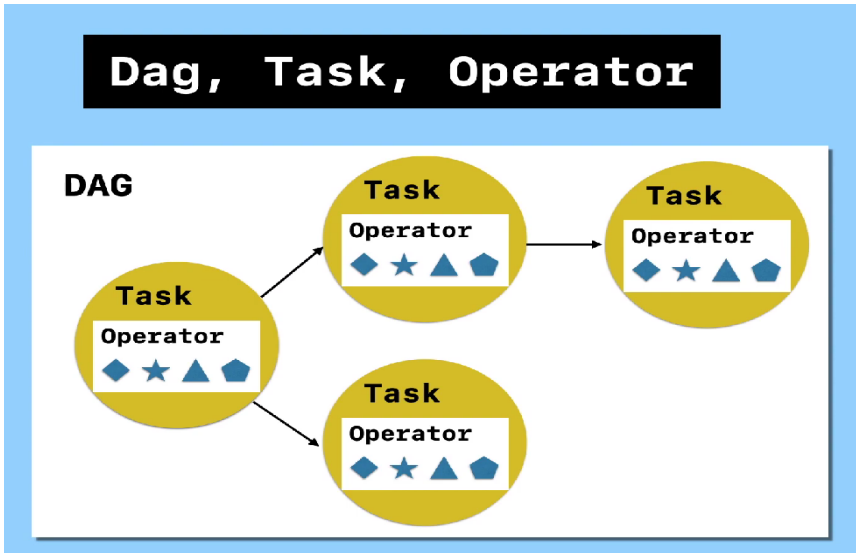
Airflow is a platform for programmatically authoring, scheduling, and monitoring workflows.

Airflow is a tool for managing data pipelines, which is an essential task for data engineering.

Airflow Journey : Airbnb - 2014 **:** Apache- Airflow - 2019

Advantages:	**Alternatives:**	**Applications:**
Dynamic Scalable UI Extensibility	Luigi Apache NiFi AWS Step Function Apache Oozie Perfect	Data Pipeline Workflow automation Machine learning ETL Monitoring

Core Components: DIRECTED ACYCLIC GRAPH



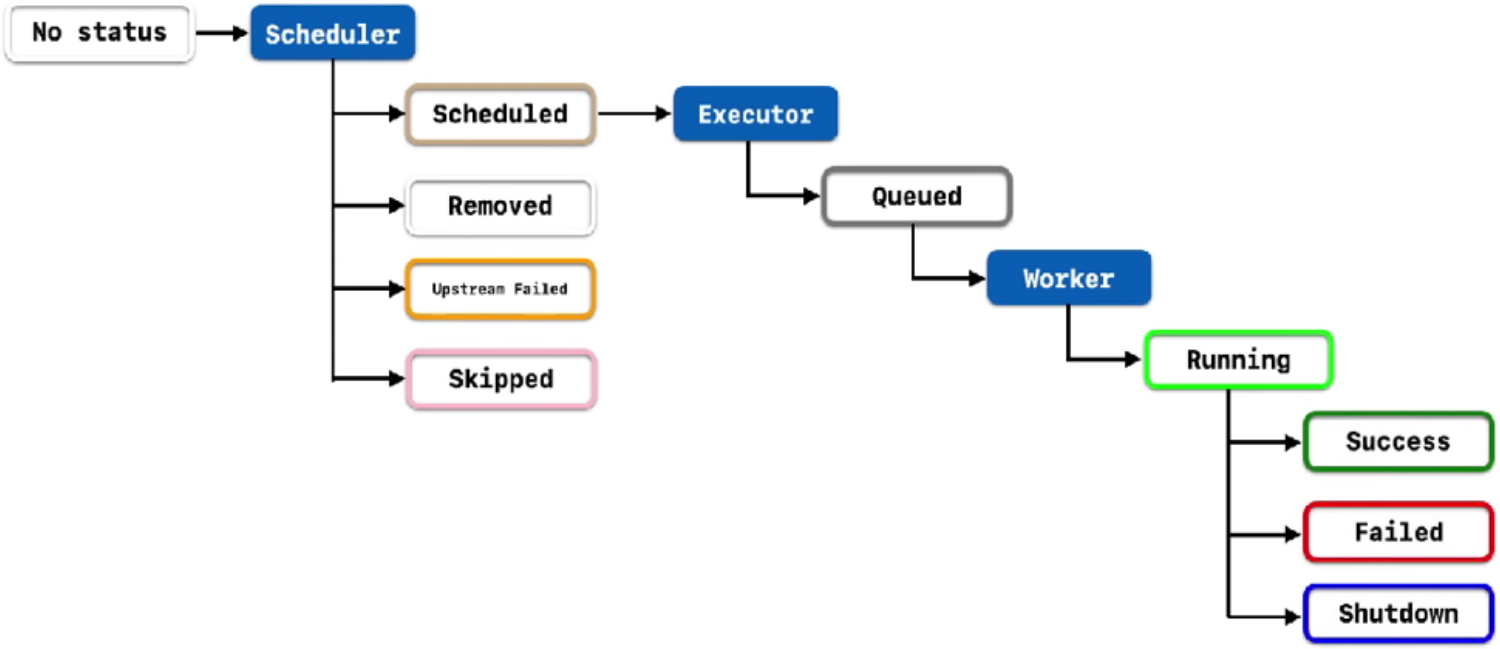
Sensors
Task Instance
Execution Date
Connections
Hooks
Schedulers
Executors
WebUI - View
XComs and Variables

- **DAGs (Directed Acyclic Graphs)**: These represent the workflows or data pipelines that you want to run.
- **Operators**: These are the building blocks of DAGs, which perform individual tasks or actions.
- **Sensors**: These are special types of operators that wait for a certain condition to be met before proceeding.
- **Scheduler**: This is the component that schedules and executes DAGs based on their dependencies and specified schedule.
- **Executors**: These are responsible for actually running the tasks defined in the DAG.
- **Web UI:** This is the interface that users interact with to manage and monitor DAGs.

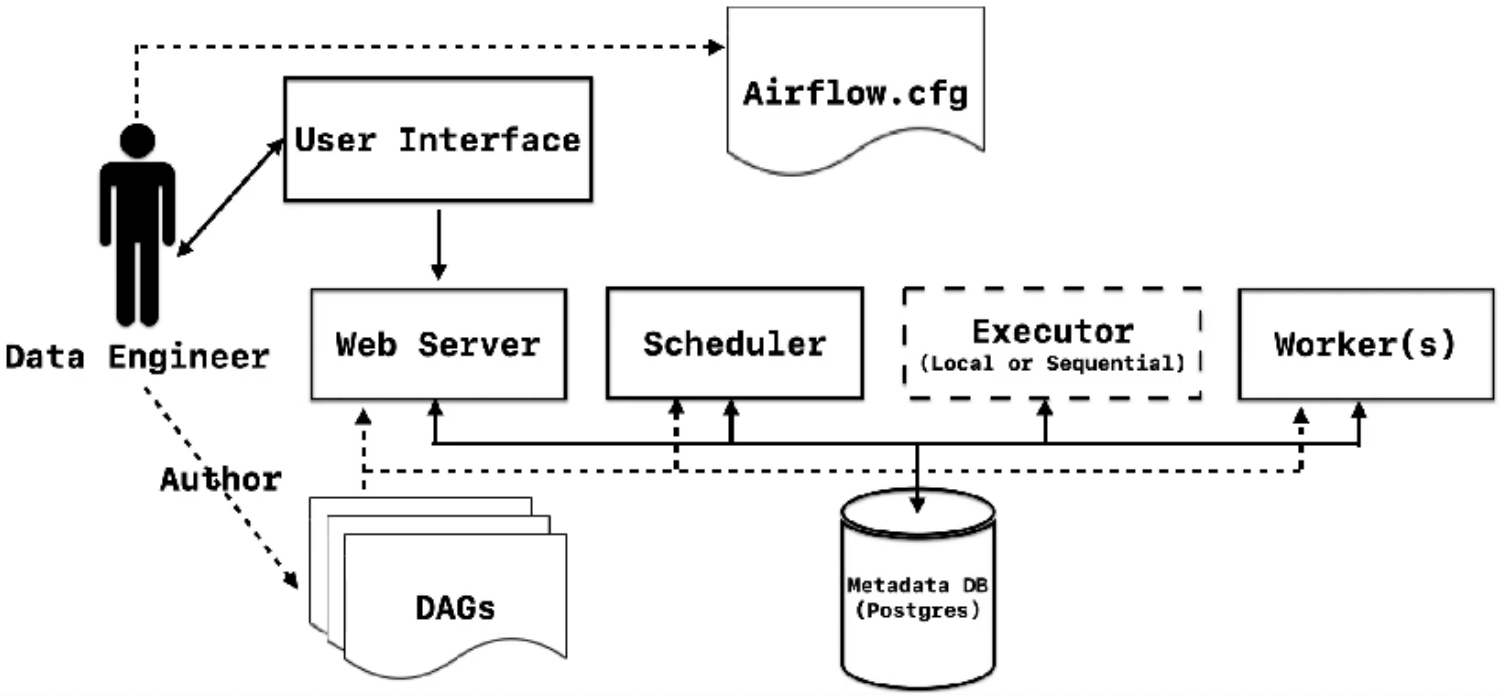
Advance Concepts of Airflow
Data Profiling
SubDAGs
PLUGINS- Creating Custom Components
Trigger Rules, airflow ignore file, Zombies & Undeads

Components Flow:

Webserver | Scheduler | Metastore | Triggerer
Executer
Queue | Worker



Airflow Architecture:



Directory Flow:

1. **dags**/: This is where DAGs (Directed Acyclic Graphs) are defined.
2. **plugins**/: Airflow can be extended with plugins that provide additional functionality.
3. **logs**/: Airflow logs are stored here.
4. **airflow.cfg**: Airflow's configuration file is stored here.
5. **venv**/: Airflow requires a Python virtual environment to run.
6. **tests**/: Directory for Airflow unit tests.
7. **docs**/: Directory for Airflow documentation.
8. **scripts**/: Contains scripts to perform various actions on Airflow.
9. **airflow_home**/: The home directory for Airflow.
10. **airflow_db**/: Directory for Airflow's metadata database.

Installation

Airflow Direct Installation

Basic Pre-Req: Python | VS Code

Install Python and Add to path
Install VS Code and Open a New Folder
Check Version : Python3 —version [till 3.10]

Initialize VS Code:

Create a Virtual Environment
python3 -m venv myenv

Activate Virtual Environment
source myenv/bin/activate

Install Airflow
Open: https://github.com/apache/airflow

Navigate to Install Via pypi and Copy The Pip Command:

pip install 'apache-airflow==2.5.2' --constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.5.2/constraints-3.10.txt"

Export airflow home directory into current directory
export AIRFLOW_HOME=.

Initialise db : airflow db init

Start web server: airflow webserver -p 8080

[Sqlite error :
Go to airflow config file - Change sql_alchemy_conn
sql_alchemy_conn = sqlite:////tmp/airflow.db]

Open Web UI:
Go to Browser and Open **localhost:8080**

To Create User:
Go to VS Code and Stop web server [**^+C**]

Create user:
airflow users create --username admin --firstname Peter --lastname Parker --role Admin --email spiderman@superhero.org —password admin

Run Server and login using username and password

Run scheduler:
Open Another Terminal in VS Code : (activated env)
Export airflow home : export AIRFLOW_HOME=.
airflow scheduler

[If no changes, Go to web server terminal and restart db + web server, Go to Web Server and Refresh]
Select a DAG and Explore.
Stop Webserver and Scheduler - ^+C

Apache Airflow - Docker Version

Python | VS Code | Docker

Install Python and Add to path
Install VS Code and Open a New Folder
Install Docker and Docker-compose

Go to VS Code & Create New Folder
Check the versions of Docker and Docker-Compose

docker –version
docker-compose –version

Download Official Docker yaml File of Airflow:
curl -LfO
'https://airflow.apache.org/docs/apache-airflow/2.5.2/docker-compose.yaml'

Edit yaml File:
1 : Executer - LocalExecuter
2 : Remove Celery related params
3 : Remove Redis
4 : Remove Flower

Create Folders for Dogs, Logs and Plugins
mkdir -p ./dags ./logs ./plugins

Initialise db: docker compose up airflow-init
[All Necessary Docker images with Username and Password]

Run Airflow in Detached mode:
Docker compose up -d

To View Docker:
docker ps

We See airflow Web server, Scheduler and database

Access Web UI Airflow:
0:0:0:0:8080 or localhost:8080
Login Via airflow , airflow

Open Docker Desktop and Check Containers
Turn on and Trigger A DAG

!! Your AIRFLOW SETUP WITH DOCKER IS READY !!!

To Stop the Execution of Docker:
Docker compose down -v

Create a DAG in Airflow:

- DAGs are written in Python code, using the Airflow DAG API.
- DAG defines the structure of the workflow, including the tasks, dependencies, and schedule.

Define tasks in a DAG:

- Tasks are defined using operators, which are Python classes that perform specific actions.
- Examples of operators, such as BashOperator, PythonOperator, and MySqlOperator.
- Operators can be chained together to create complex workflows.

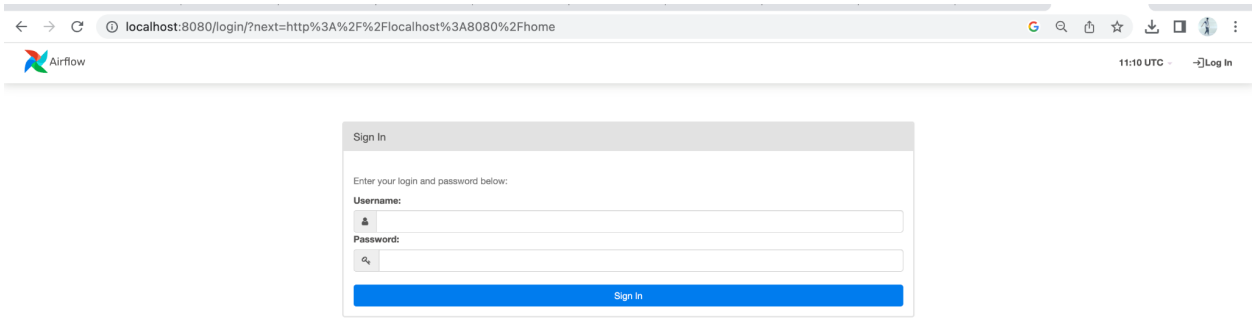
Schedule and monitor DAGs in Airflow:

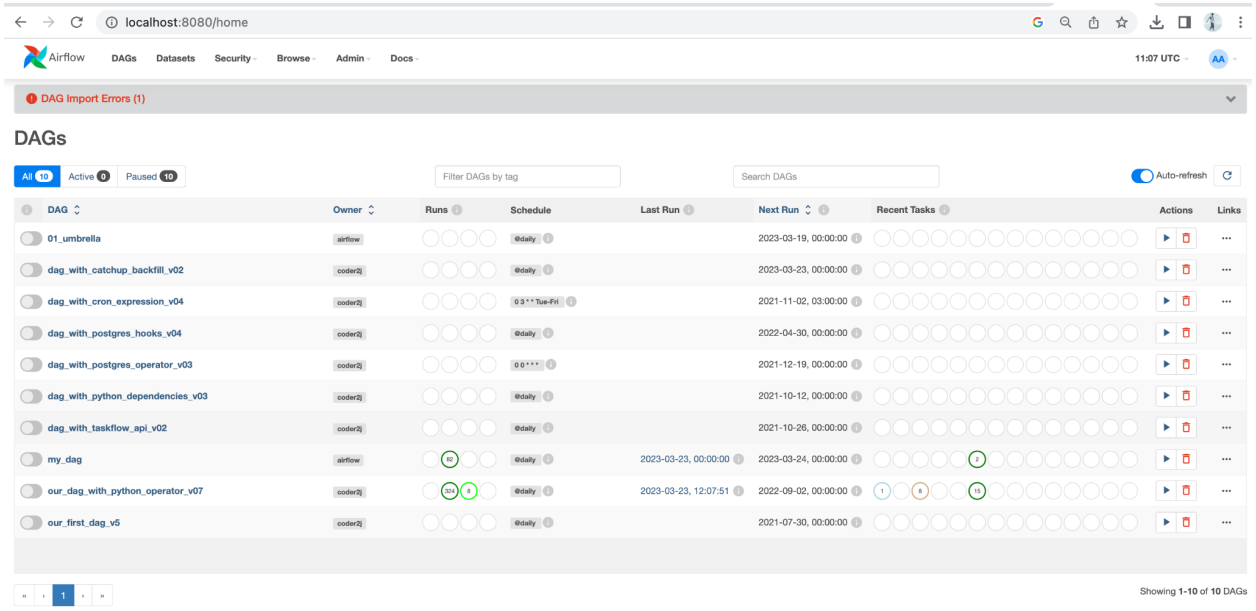
- DAGs can be scheduled using a variety of options, such as a fixed interval.
- Airflow Web UI to monitor the progress of DAGs and view logs.
- Importance of testing and debugging DAGs to ensure they are running correctly.

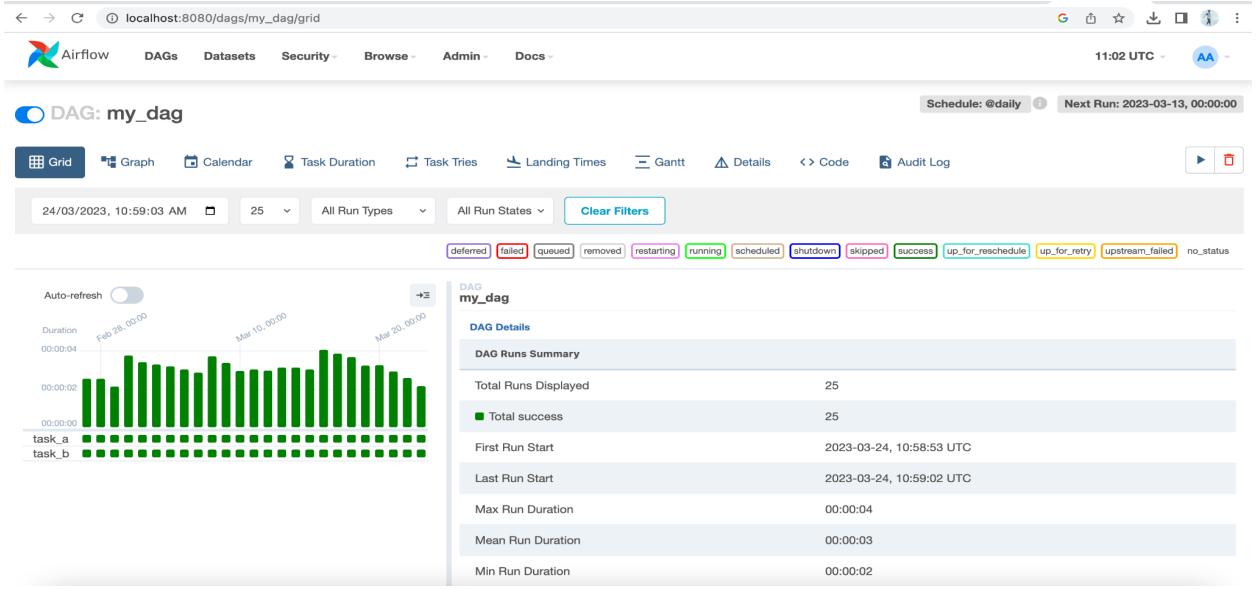
First Dag :

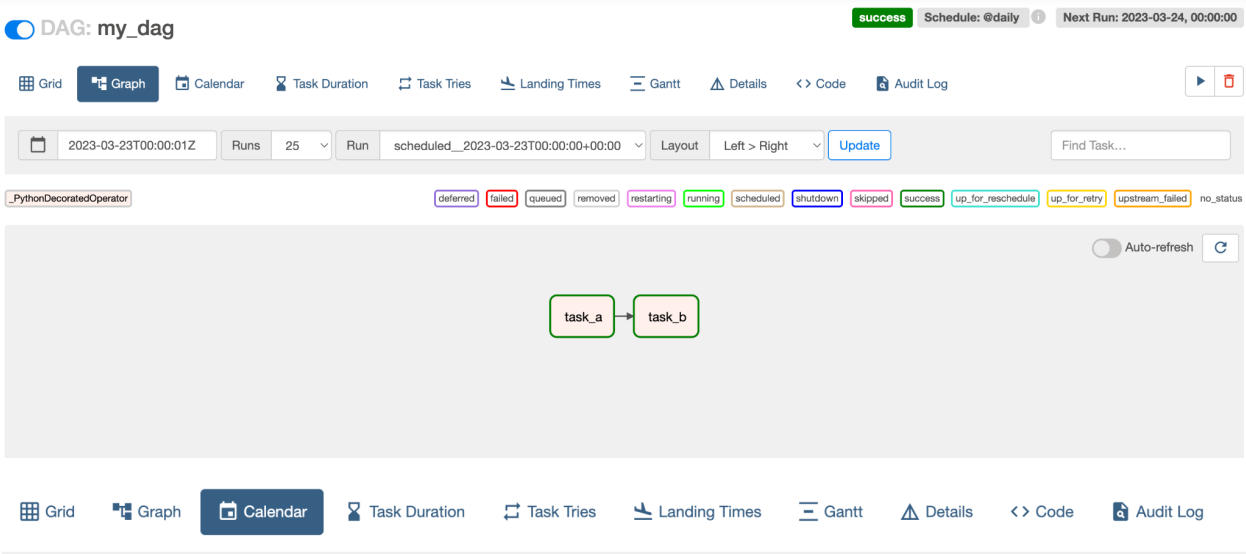
```
from airflow.decorators import dag, task
from datetime import datetime

@dag(start_date=datetime(2023, 1,1),
schedule='@daily')
def my_dag():

    @task
    def task_a():
        return val + 42

    @task
    def task_b(val):
        print(val)

    task_b(task_a(42))

my_dag()
```

Web UI:

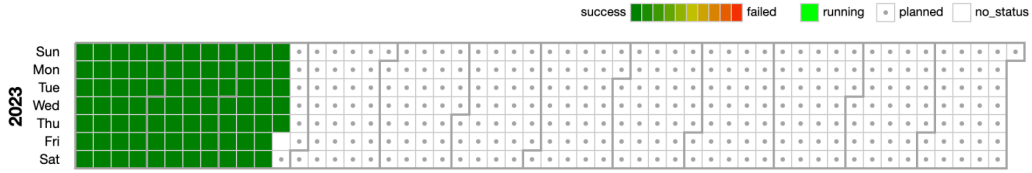

localhost:8080/login/?next=http%3A%2F%2Flocalhost%3A8080%2Fhome
Airflow
11:10 UTC
Log In
Sign In
Enter your login and password below:
Username:
Password:
Sign In

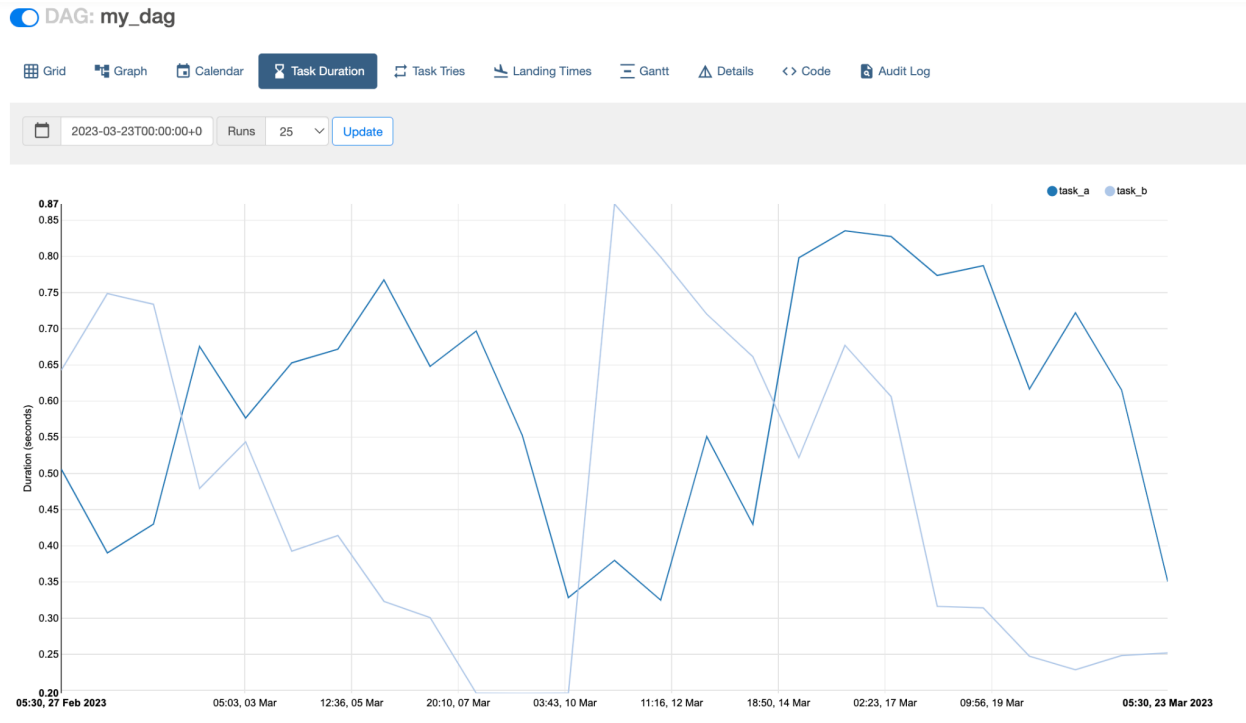

localhost:8080/home
Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
11:07 UTC
DAG Import Errors (1)
DAGs
All 10
Active 0
Paused 10
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
Actions
Links
01_umbrella
dag_with_catchup_backfill_v02
dag_with_cron_expression_v04
dag_with_postgres_hooks_v04
dag_with_postgres_operator_v03
dag_with_python_dependencies_v03
dag_with_taskflow_api_v02
my_dag
our_dag_with_python_operator_v07
our_first_dag_v5
Showing 1-10 of 10 DAGs


localhost:8080/dags/my_dag/grid
Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
11:02 UTC
DAG: my_dag
Schedule: @daily
Next Run: 2023-03-13, 00:00:00
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
24/03/2023, 10:59:03 AM
25
All Run Types
All Run States
Clear Filters
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
Duration
Feb 28, 00:00
Mar 10, 00:00
Mar 20, 00:00
00:00:04
00:00:02
00:00:00
task_a
task_b
DAG
my_dag
DAG Details
DAG Runs Summary
Total Runs Displayed 25
Total success 25
First Run Start 2023-03-24, 10:58:53 UTC
Last Run Start 2023-03-24, 10:59:02 UTC
Max Run Duration 00:00:04
Mean Run Duration 00:00:03
Min Run Duration 00:00:02


DAG: my_dag
success
Schedule: @daily
Next Run: 2023-03-24, 00:00:00
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2023-03-23T00:00:01Z
Runs
25
Run
scheduled__2023-03-23T00:00:00+00:00
Layout
Left > Right
Update
Find Task...
_PythonDecoratedOperator
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
task_a
task_b

Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log


success
failed
running
planned
no_status
2023
Sun
Mon
Tue
Wed
Thu
Fri
Sat


DAG: my_dag
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2023-03-23T00:00:00+0
Runs
25
Update
task_a
task_b
Duration (seconds)
0.87
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
05:30, 27 Feb 2023
05:03, 03 Mar
12:36, 05 Mar
20:10, 07 Mar
03:43, 10 Mar
11:16, 12 Mar
18:50, 14 Mar
02:23, 17 Mar
09:56, 19 Mar
05:30, 23 Mar 2023

Second Dag:

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.bash import BashOperator

default_args = {
    'owner': admin,
    'retries': 5,
    'retry_delay': timedelta(minutes=2)
}

with DAG(
    dag_id=’first_dag',
    default_args=default_args,
    description='This is our first dag that we write',
    start_date=datetime(2021, 7, 29, 2),
    schedule_interval='@daily'
) as dag:
    task1 = BashOperator(
        task_id='first_task',
        bash_command="echo hello world, this is the
first task!"
    )

    task2 = BashOperator(
        task_id='second_task',
```

```
        bash_command="echo hey, I am task2 and will be
running after task1!"
    )

    task3 = BashOperator(
        task_id='thrid_task',
        bash_command="echo hey, I am task3 and will be
running after task1 at the same time as task2!"
    )

    # Task dependency method 1
    # task1.set_downstream(task2)
    # task1.set_downstream(task3)

    # Task dependency method 2
    # task1 >> task2
    # task1 >> task3

    # Task dependency method 3
    task1 >> [task2, task3]
```

Third Dag:

```
"""DAG demonstrating the umbrella use case with dummy
operators."""

import airflow.utils.dates
from airflow import DAG
from airflow.operators.dummy import DummyOperator

dag = DAG(
    dag_id="01_umbrella",
    description="Umbrella example with
DummyOperators.",
    start_date=airflow.utils.dates.days_ago(5),
    schedule_interval="@daily",
)

fetch_weather_forecast =
DummyOperator(task_id="fetch_weather_forecast",
dag=dag)
fetch_sales_data =
DummyOperator(task_id="fetch_sales_data", dag=dag)
clean_forecast_data =
DummyOperator(task_id="clean_forecast_data", dag=dag)
clean_sales_data =
DummyOperator(task_id="clean_sales_data", dag=dag)
join_datasets = DummyOperator(task_id="join_datasets",
dag=dag)
```

```
train_ml_model =
DummyOperator(task_id="train_ml_model", dag=dag)
deploy_ml_model =
DummyOperator(task_id="deploy_ml_model", dag=dag)

# Set dependencies between all tasks
fetch_weather_forecast >> clean_forecast_data
fetch_sales_data >> clean_sales_data
[clean_forecast_data, clean_sales_data] >>
join_datasets
join_datasets >> train_ml_model >> deploy_ml_model
```