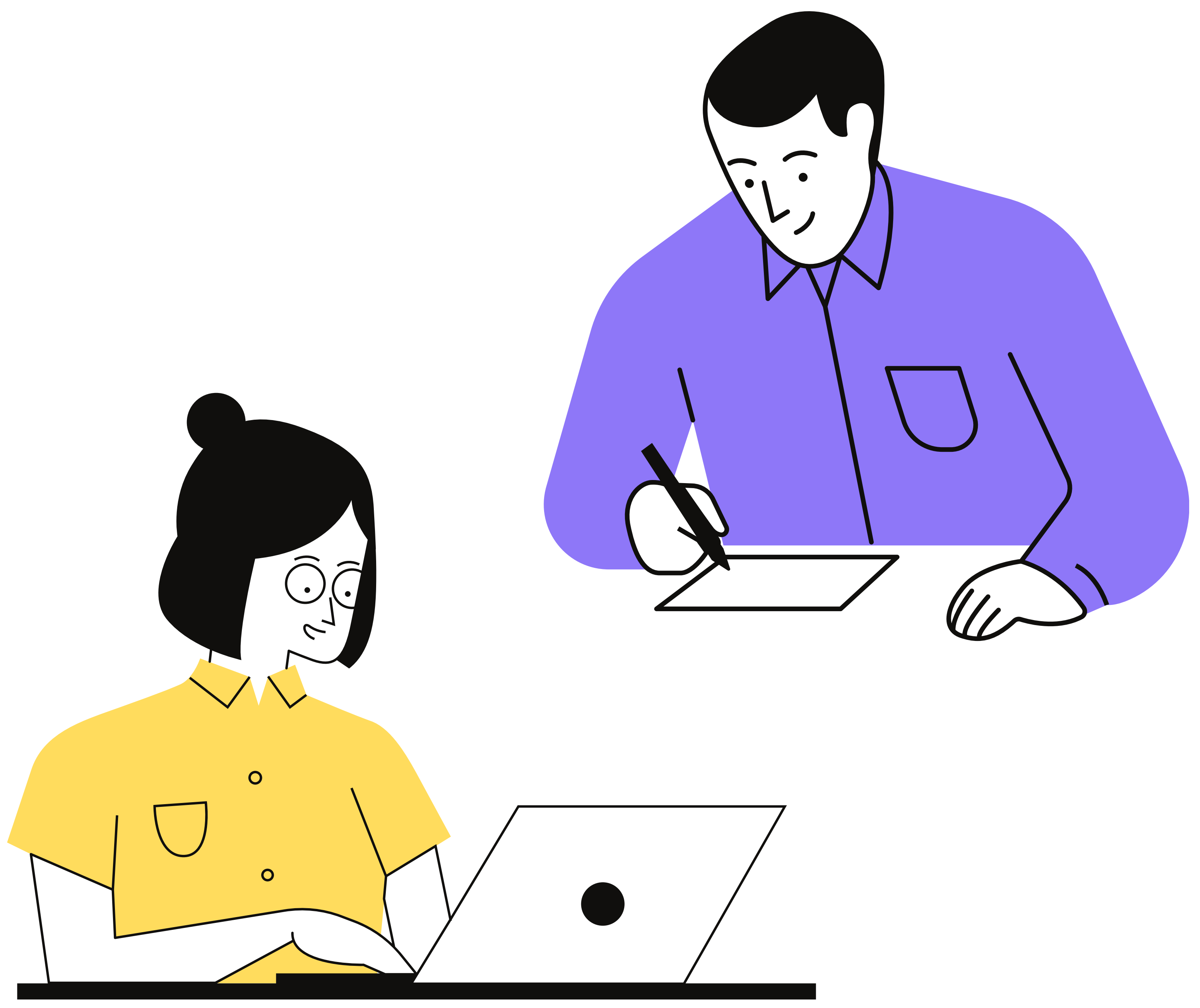
Airflow

WEEK #9



Overview

Created by:

- Muhammad Dhalhaz
- Nurul Ulum
- Ronald
- Satria Finandita
- Rifdatul Husna Ahmad

Airflow Architecture

Apache Airflow is an open-source platform used for orchestrating complex workflows and data processing pipelines. It allows users to schedule, monitor, and manage workflows as directed acyclic graphs (DAGs).

Basic Flow Architecture

Scheduler
The Scheduler is responsible for triggering the execution of tasks in a workflow based on the defined schedule.

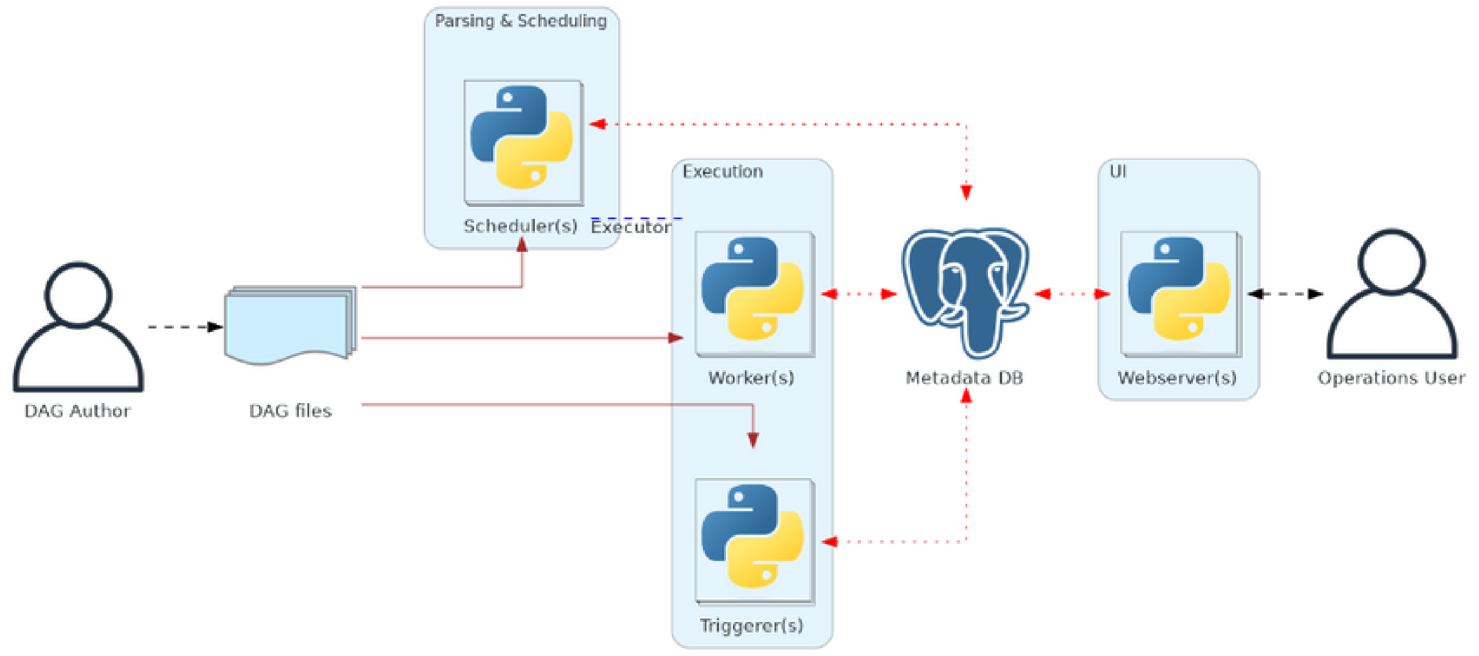
Metadata Database
The Metadata Database stores metadata related to workflows, DAGs, tasks, and their statuses. It is used by the scheduler, web server, and executor to keep track of the current state of the workflow.

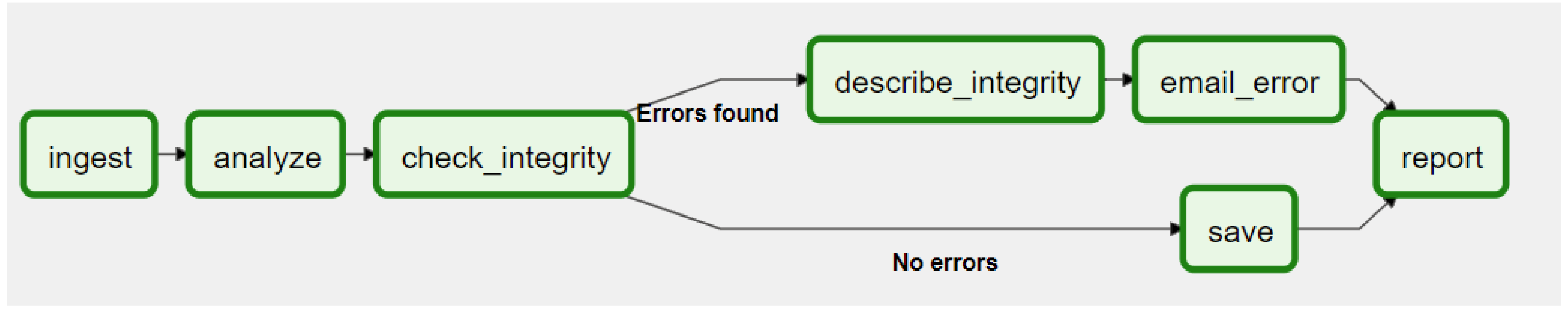
Web Server
The Web Server provides a web-based user interface for users to interact with Airflow

Executor
The Executor is responsible for actually running the tasks. Airflow supports multiple executors

DAGs (Directed Acyclic Graphs):
A DAG is a collection of tasks with defined dependencies and a schedule. It represents the workflow that needs to be executed.

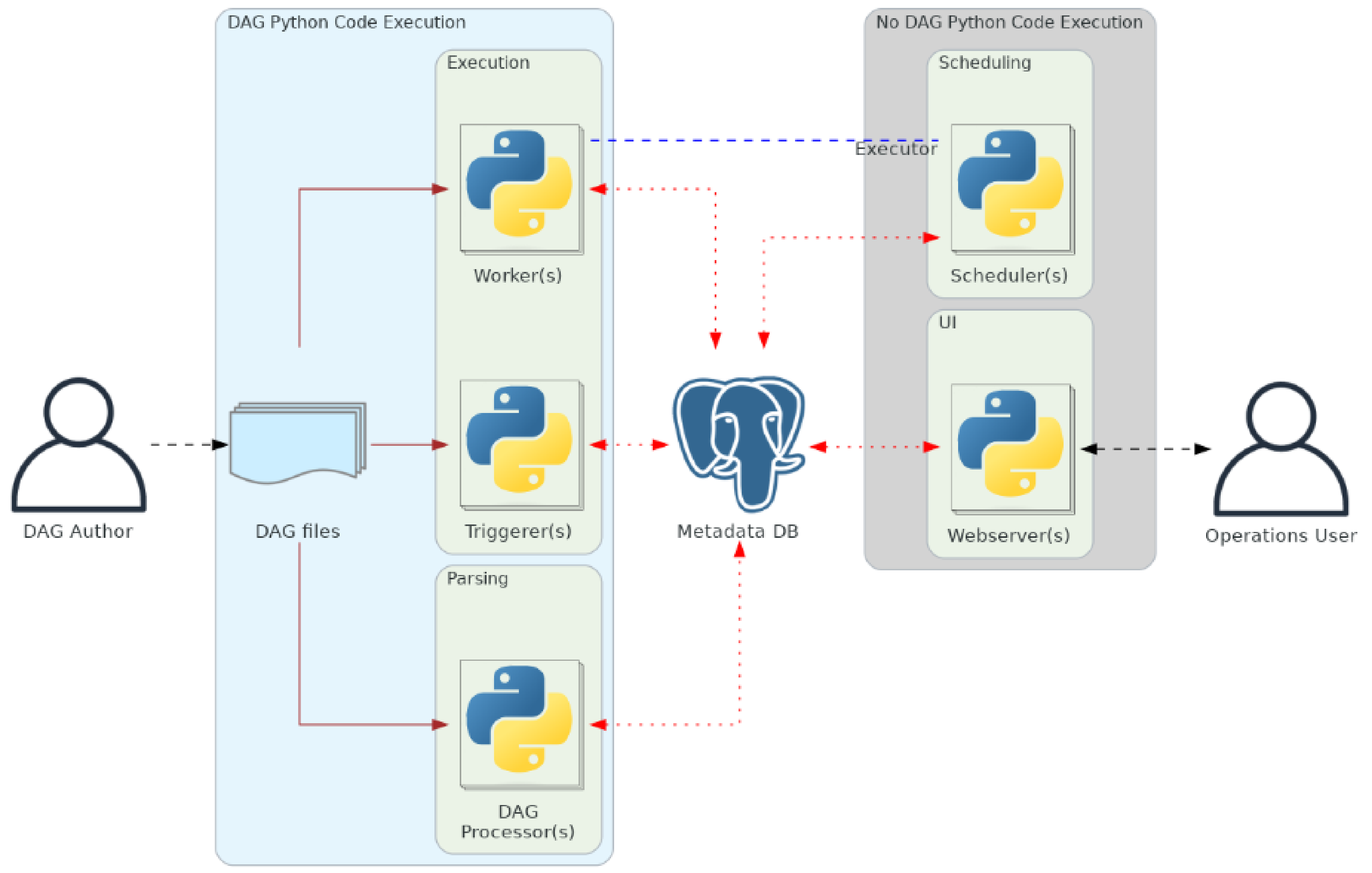
Operators
Operators define the individual steps or tasks in a workflow.





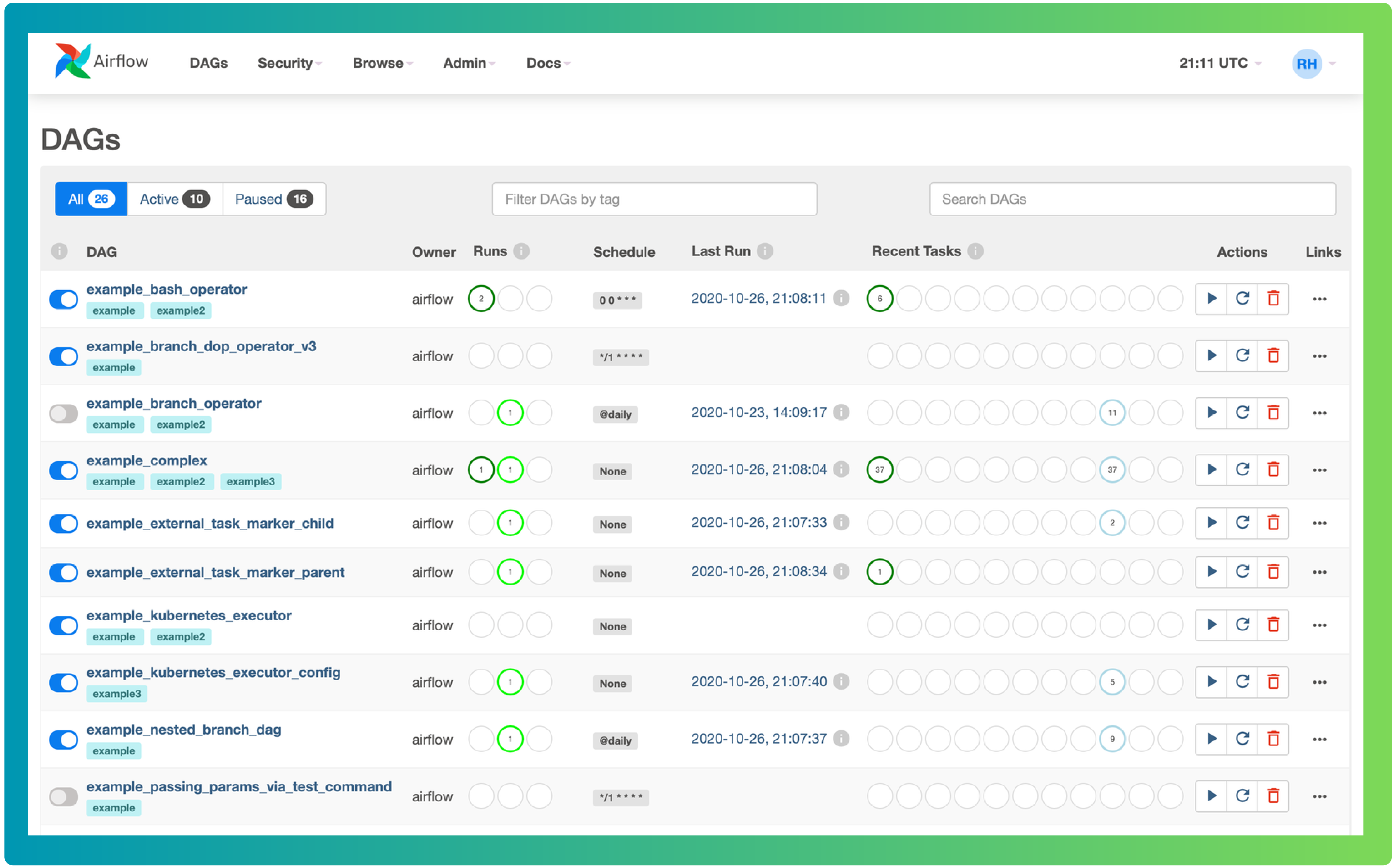
SEPARATE DAG PROCESSING ARCHITECTURE

This involves distributing the workload across multiple Airflow instances, each responsible for executing a subset of the DAGs. There are a few ways to achieve this, and one common approach is to use a combination of a CeleryExecutor and multiple Airflow Scheduler and Web Server instances.



Airflow UI: DAGs View

Directed Acyclic Graph (DAG) is a collection of tasks with defined dependencies that can be executed in a specific order.



DAGs

All 26 | Active 10 | Paused 16

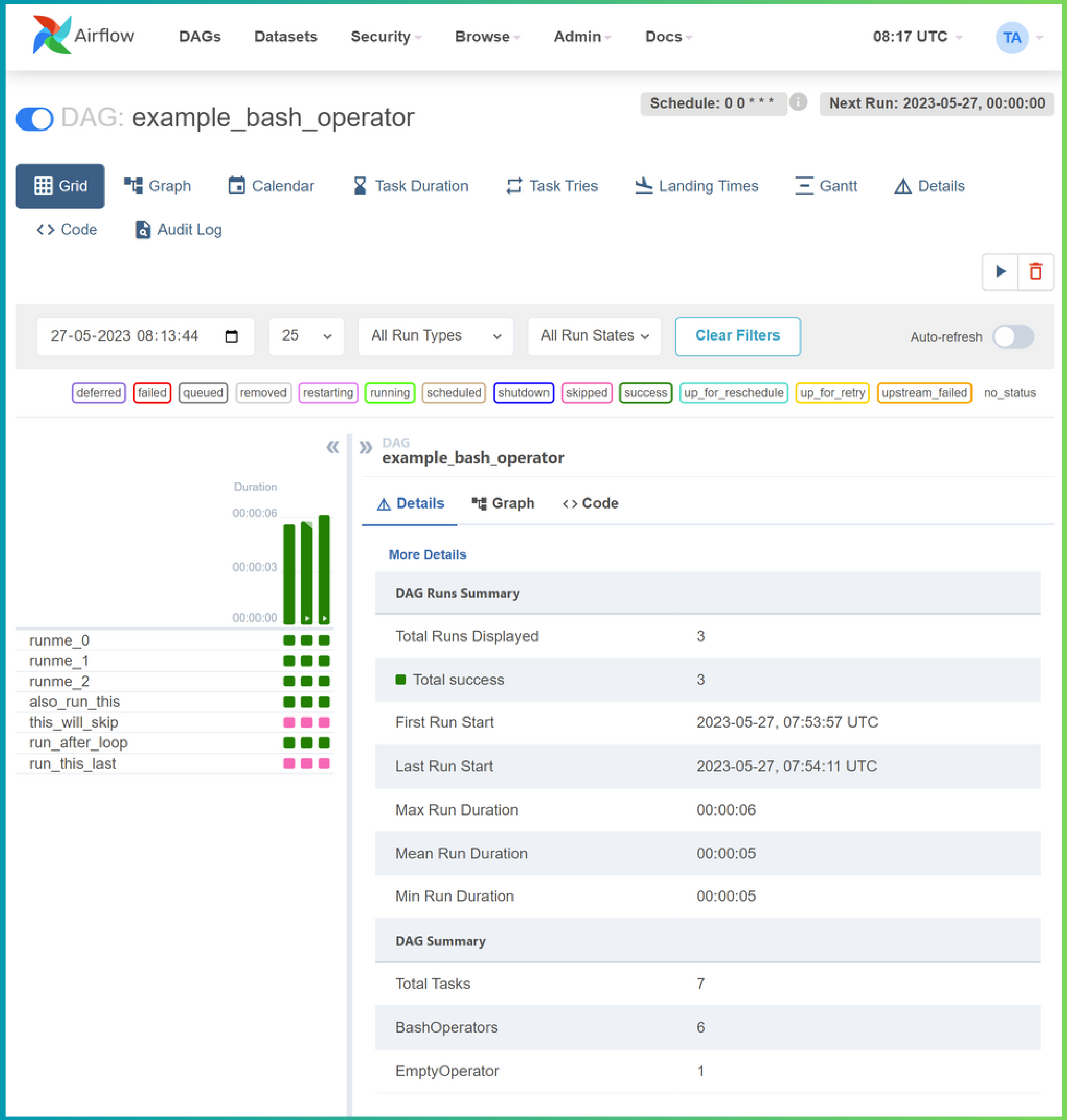
Filter DAGs by tag

Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator (example, example2)	airflow	2	0 0 * * *	2020-10-26, 21:08:11	6		
example_branch_dop_operator_v3 (example)	airflow		*/1 * * * *				
example_branch_operator (example, example2)	airflow	1	@daily	2020-10-23, 14:09:17	11		
example_complex (example, example2, example3)	airflow	1 1	None	2020-10-26, 21:08:04	37 37		
example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2		
example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1		
example_kubernetes_executor (example, example2)	airflow		None				
example_kubernetes_executor_config (example3)	airflow	1	None	2020-10-26, 21:07:40	5		
example_nested_branch_dag (example)	airflow	1	@daily	2020-10-26, 21:07:37	9		
example_passing_params_via_test_command (example)	airflow		*/1 * * * *				

Airflow UI: Grid View

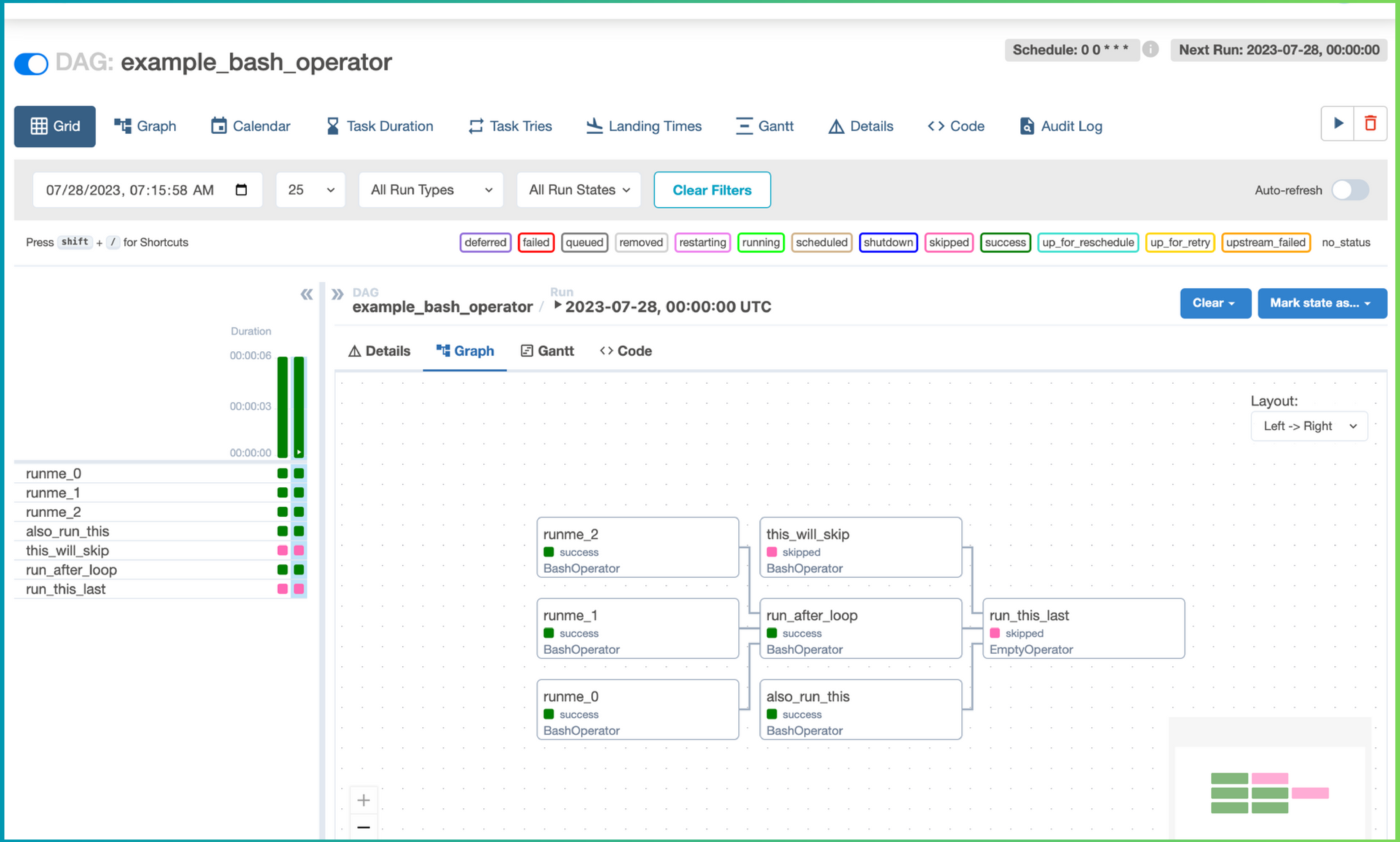
A bar chart and grid representation of the DAG that spans across time. The top row is a chart of DAG Runs by duration, and below, task instances.



DAG Runs Summary	
Total Runs Displayed	3
🟩 Total success	3
First Run Start	2023-05-27, 07:53:57 UTC
Last Run Start	2023-05-27, 07:54:11 UTC
Max Run Duration	00:00:06
Mean Run Duration	00:00:05
Min Run Duration	00:00:05
DAG Summary	
Total Tasks	7
BashOperators	6
EmptyOperator	1

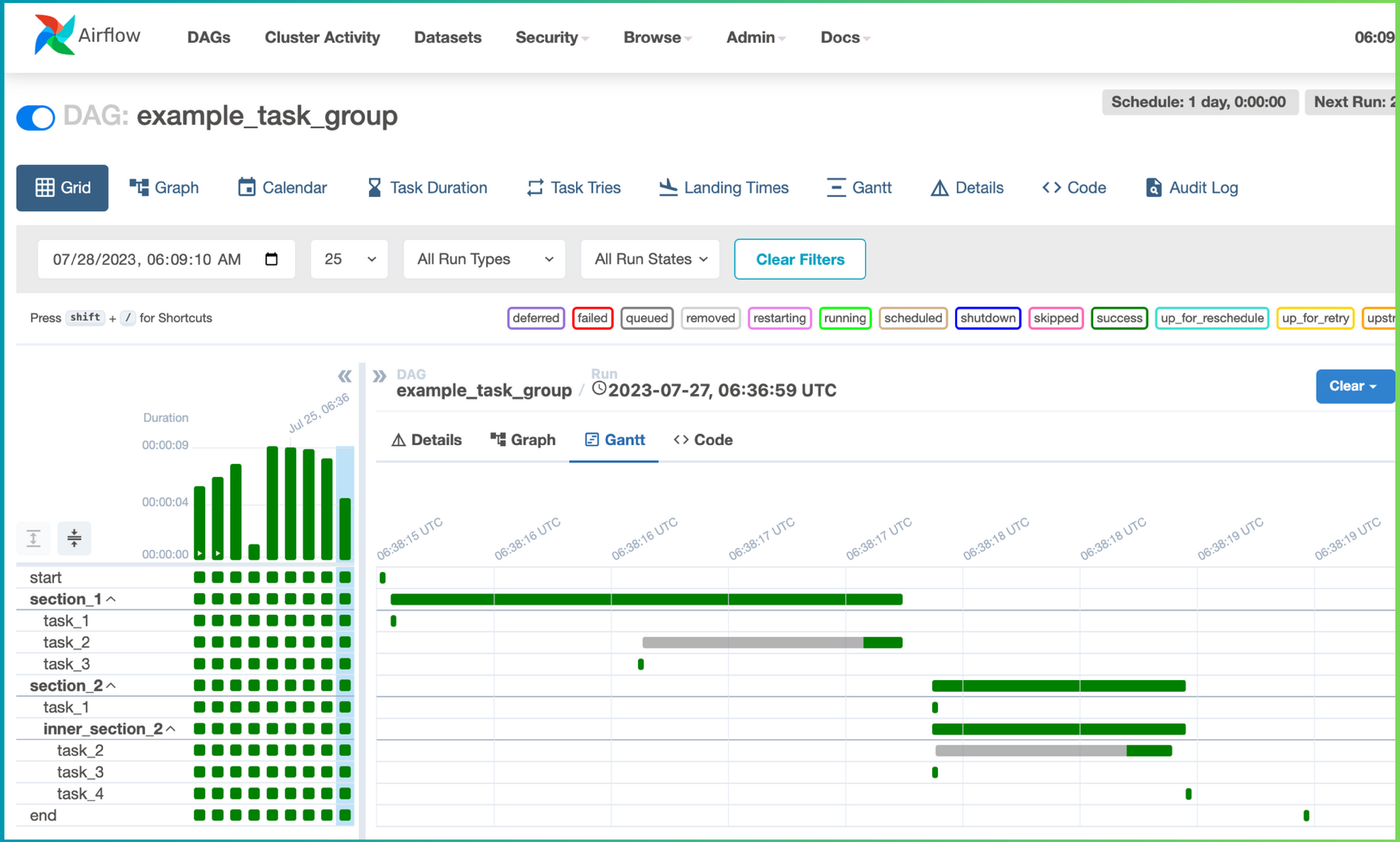
Airflow UI: Graph View

The graph view is perhaps the most comprehensive. Visualize your DAG's dependencies and their current status for a specific run.



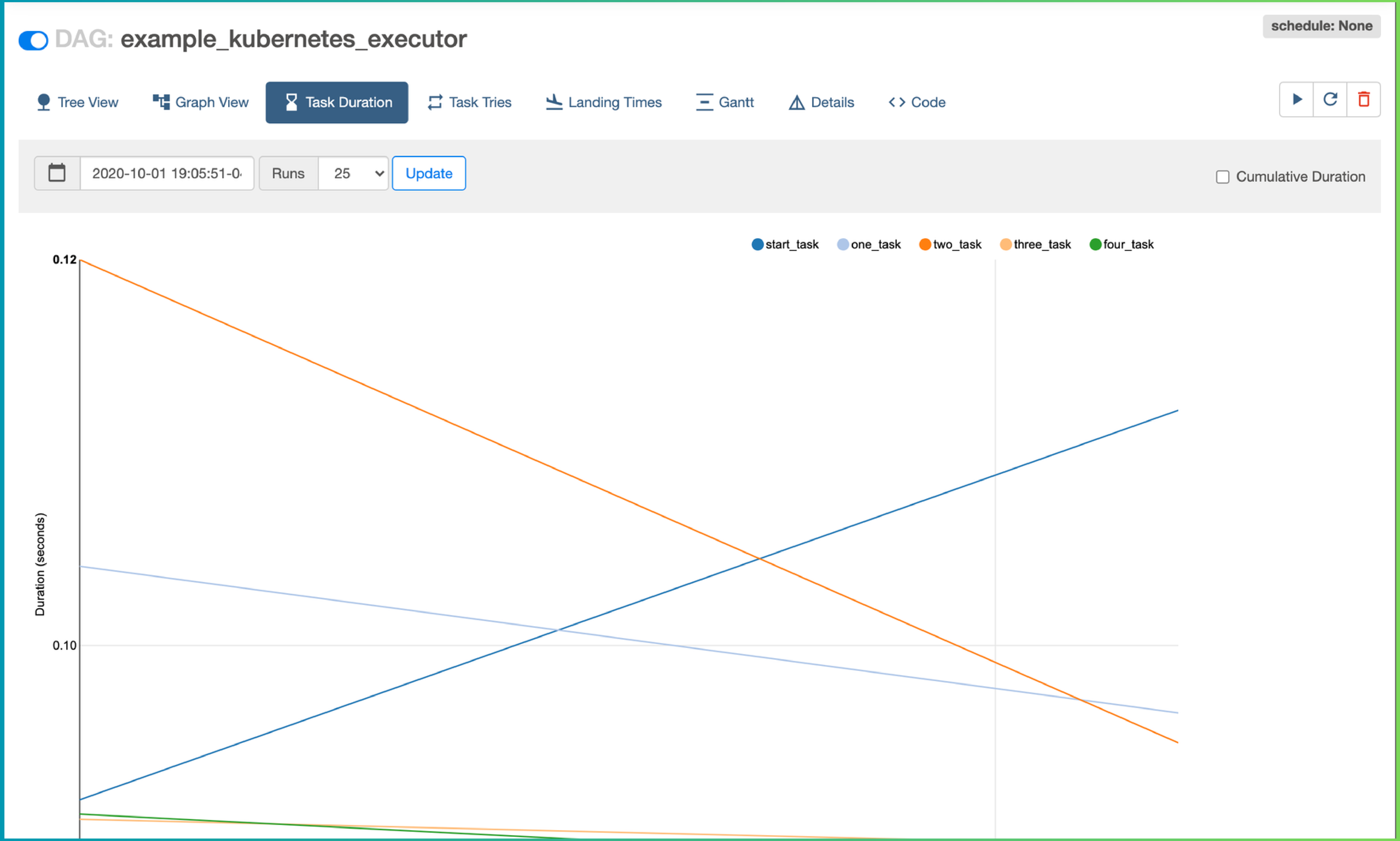
Airflow UI: Gantt Chart

The Gantt chart lets you analyse task duration and overlap. You can quickly identify bottlenecks and where the bulk of the time is spent for specific DAG runs.



Airflow UI: Task Duration

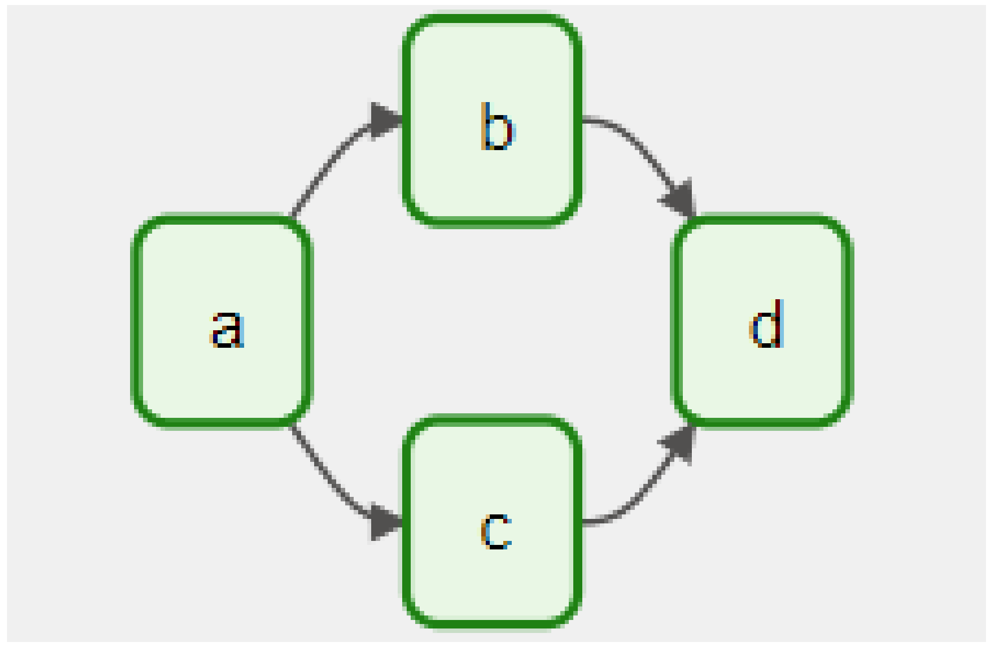
The duration of your different tasks over the past N runs. This view lets you find outliers and quickly understand where the time is spent in your DAG over many runs.



DAG (Directed Acyclic Graph) Elemets

1. DAG (Directed Acyclic Graph)
2. DAG Runs
3. DAG Task
3. DAG Task Instances

DAG (Directed Acyclic Graph)



You can declare a DAG in three ways:

- using a context manager,
- a standard constructor, or
- the @dag decorator.

In **Apache Airflow**, a **Directed Acyclic Graph** (DAG) is the core concept. It is a collection of tasks with defined execution dependencies. Each node in the graph represents a task, and the edges define the order of execution.

You can also declare task dependencies using the >> and << operators, or the set_upstream and set_downstream methods. For more complex dependencies, you can use cross_downstream and chain.

DAG Runs

DAG Run is an object representing an instantiation of the DAG in time. Here are the key points about DAG Runs:

- Execution: Any time the DAG is executed, a DAG Run is created and all tasks inside it are executed.
- Status: The status of the DAG Run depends on the tasks states. Each DAG Run is run separately from one another, meaning that you can have many runs of a DAG at the same time.
- DAG Run Status: A DAG Run status is determined when the execution of the DAG is finishe1. The execution of the DAG depends on its containing tasks and their dependencies. The status is assigned to the DAG Run when all of the tasks are in one of the terminal states (i.e., if there is no possible transition to another state) like success, failed, or skipped.
- Terminal States: There are two possible terminal states for the DAG Run: success if all of the leaf nodes states are either success or skipped, failed if any of the leaf nodes state is either failed or upstream_failed.
- Trigger Rule: Be careful if some of your tasks have defined some specific trigger rule. These can lead to some unexpected behavior.
- UI Dashboard: DAGs that have a currently running DAG run can be shown on the UI dashboard in the “Running” tab. Similarly, DAGs whose latest DAG run is marked as failed can be found on the “Failed” tab.
- Data Interval: Each DAG run in Airflow has an assigned “data interval” that represents the time range it operates in.

DAG Task

In Apache Airflow, a Task is the basic unit of execution. Tasks are arranged into Directed Acyclic Graphs (DAGs), and then have upstream and downstream dependencies set between them to express the order they should run. Here are the key points about Tasks:

- Types of Tasks: There are three basic kinds of Task: Operators, Sensors, and TaskFlow.
- Task Dependencies: The key part of using Tasks is defining how they relate to each other – their dependencies, or as we say in Airflow, their upstream and downstream tasks.
- Task Instances: Much in the same way that a DAG is instantiated into a DAG Run each time it runs, the tasks under a DAG are instantiated into Task Instances. An instance of a Task is a specific run of that task for a given DAG (and thus for a given data interval). They are also the representation of a Task that has state, representing what stage of the lifecycle it is in.
- Task States: The possible states for a Task Instance are: none (The Task has not yet been queued for execution), scheduled (The scheduler has determined the Task’s dependencies are met and it should run), and queued (The task has been assigned to an Executor and is awaiting a worker).

DAG Task Instances

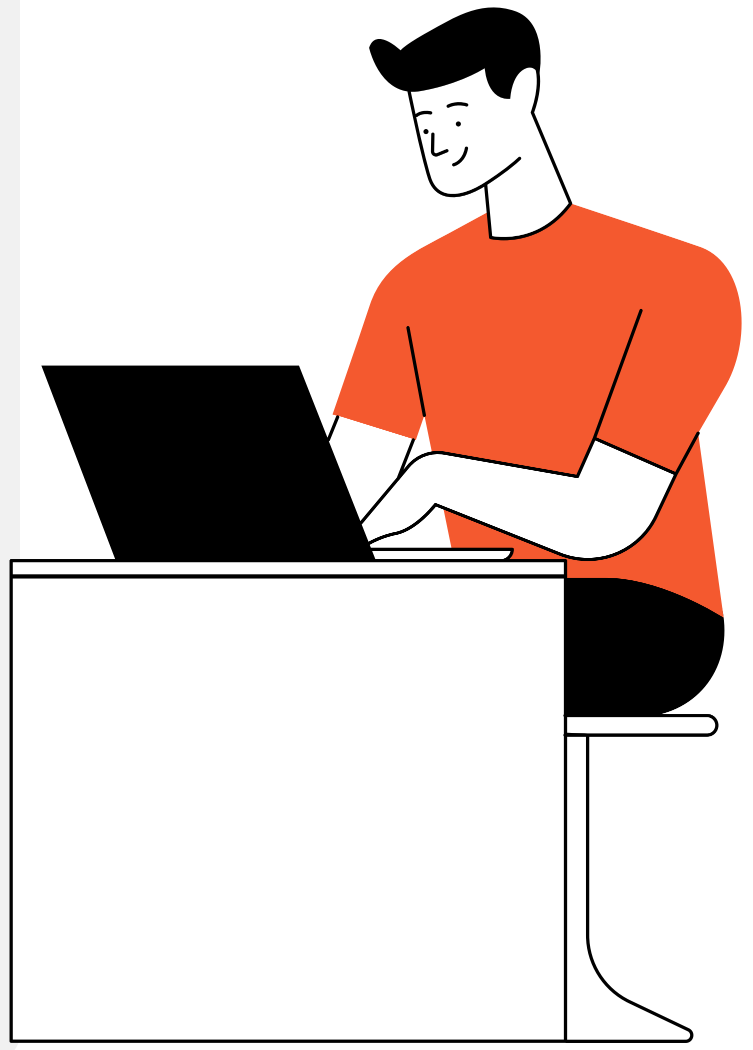
In Apache Airflow, a Task Instance represents an individual execution of a task within a DAG for a specific run. Here are the key points about Task Instances:

- Definition: A Task Instance is defined as a representation for a specific run of a Task and a categorization with a collection of 'a DAG, a task, and a point in time’.
- Unique Key: Each Task Instance is associated with a unique combination of a DAG ID, task ID, execution date, and try number, forming a Task Instance key.
- State: Task Instances have a follow-up loop that indicates which state the Task Instance falls upon. They are also the representation of a Task that has state, representing what stage of the lifecycle it is in.
- Possible States: The possible states for a Task Instance are: none (The Task has not yet been queued for execution, its dependencies are not yet met), scheduled (The scheduler has determined the Task’s dependencies are met and it should run), and queued (The task has been assigned to an Executor and is awaiting a worker).

Operators

Operators are a building block of the Airflow Directed Acyclic Graph (DAG). They contain the logic to perform a unit of work within a pipeline. Each task in a DAG is defined by instantiating an operator.

An operator is a **Python class** that encapsulates the logic to perform a specific unit of work. They can be seen as wrappers around each unit of work, specifying the actions to be completed and abstracting away much of the code that would typically need to be written. When we instantiate an operator within a DAG and provide it with the necessary parameters, it becomes a task.



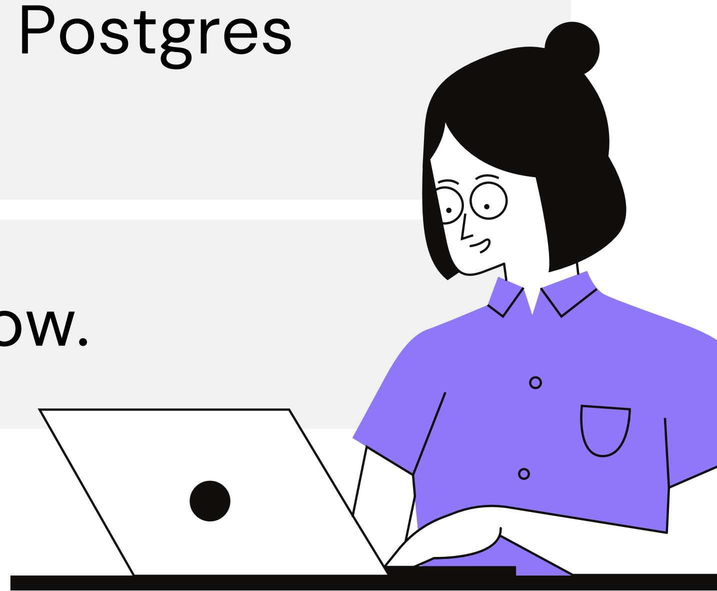
Airflow Operators Type

Here are some of the most commonly used Airflow operators:

1. **PythonOperator**: Executes a Python function.
2. **BashOperator**: Runs a bash script.
3. **KubernetesPodOperator**: Executes a task defined as a Docker image within a Kubernetes Pod.
4. **SnowflakeOperator**: Executes queries against the Snowflake database.

Use of Operators

EmptyOperator	Organizes the task flow within the DAG. Included in Airflow.
PythonDecoratedOperator	Executes a Python function. Its functionality is similar to PythonOperator, but it is instantiated using the @task decorator, as discussed earlier. Included in Airflow.
LocalFilesystemToS3Operator	Uploads files from the local file system to Amazon S3. Included in the AWS provider package.
S3ToRedshiftOperator	Transfers data from Amazon S3 to Redshift. Included in the AWS provider package.
PostgresOperator	Executes queries against a Postgres database. Included in the Postgres provider package.
SQLCheckOperator	Checks against a database using SQL queries. Included in Airflow.

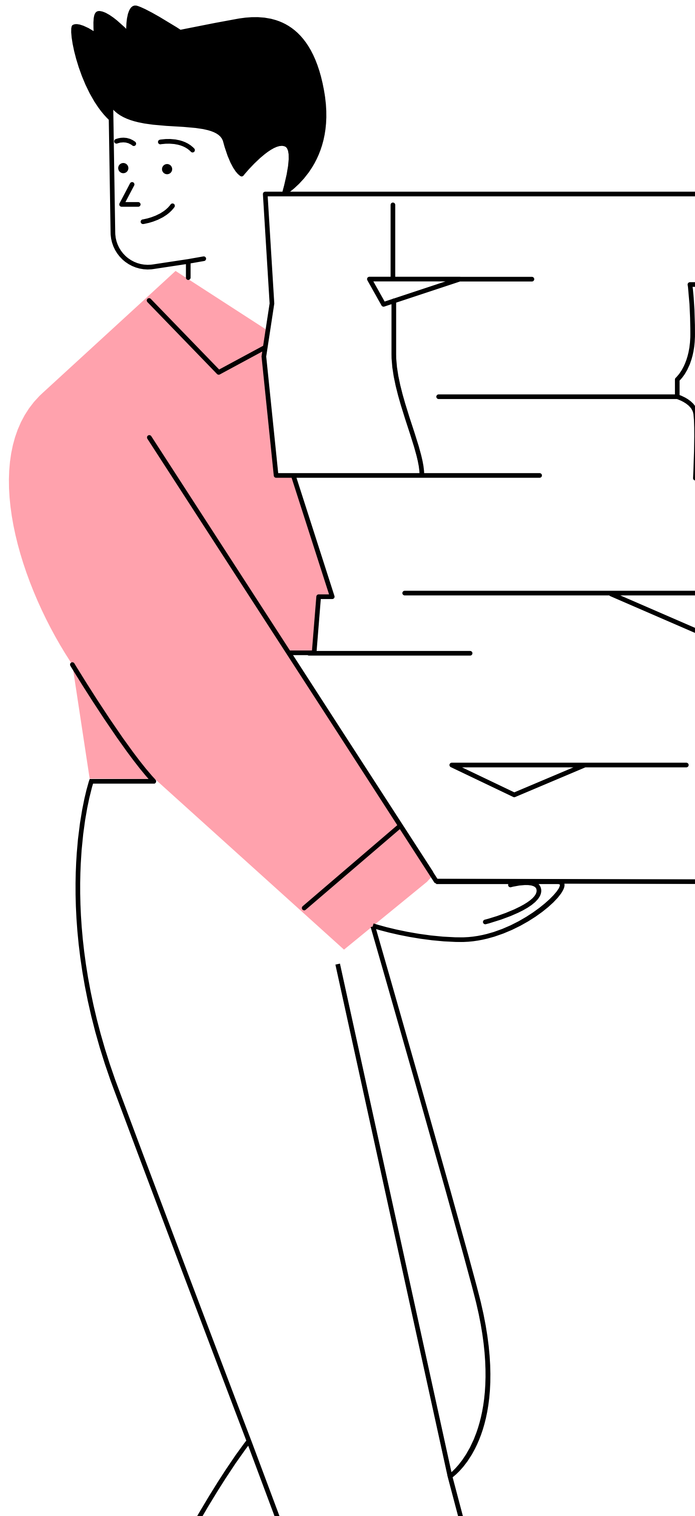


Sensors

A **Sensor** is a specialized type of Operator designed to do one thing – wait for something to happen. This could be time-based, waiting for a file, or an external event to occur. What they do is simply wait until something happens according to the specified condition, and then succeed so that their downstream tasks can proceed.

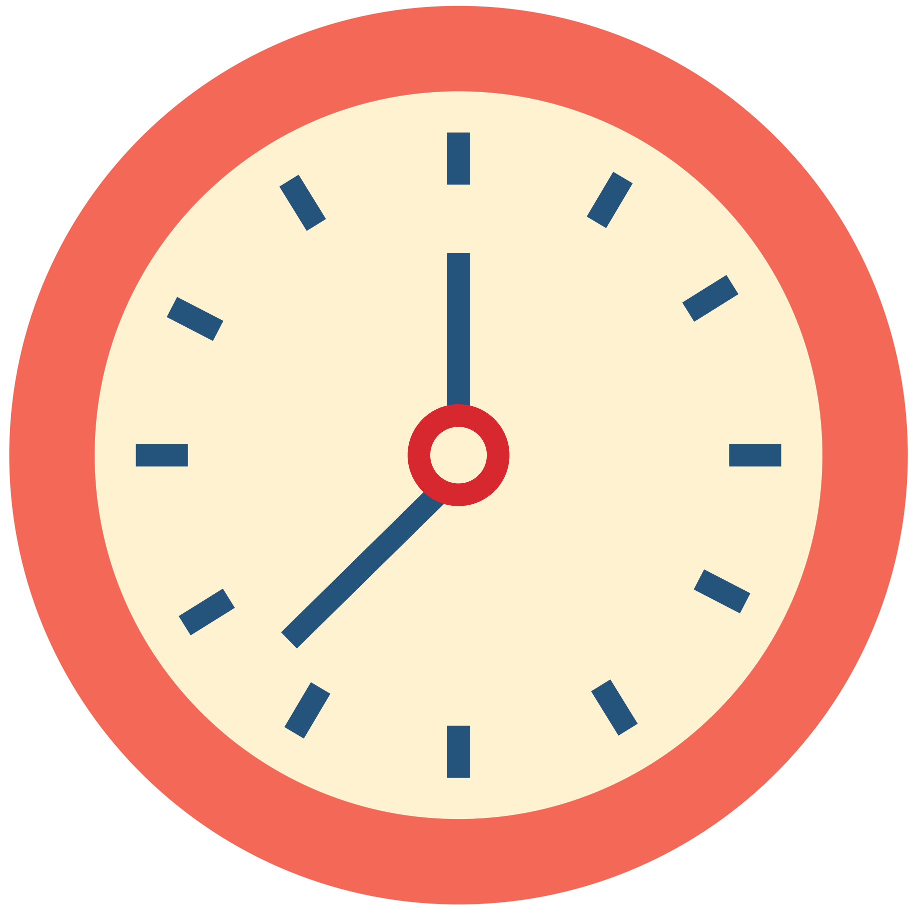
Sensors have **two different modes** of executing their tasks, allowing us to be a bit more efficient in their use:

1. **poke (default):** The Sensor uses one worker slot for the entire duration of its execution.
2. **reschedule:** The Sensor uses one worker slot only when checking, and will sleep for the specified duration between checks.



Scheduler

Airflow Scheduler is used to schedule all tasks and **DAGS**. Scheduler run behind the scene to trigger the task one after the another based on their dependencies.



Scheduling Concepts

Scheduling Concepts you need to know:

- **Data Interval:** a property of each **DAG run** that represents the period of data that each task should operate on.
- **Catchup:** refers to the behavior of Airflow where it catches up on all the runs it missed during a period of inactivity or when the DAG is paused. When a DAG is set to catch up, it will backfill and execute all the task instances that should have run according to the schedule interval since the last DAG run.
- **Backfill:** is the process of scheduling and running historical task instances that should have run at an earlier point in time. This is useful when you have a DAG (Directed Acyclic Graph) with a schedule interval, and you want to execute tasks for time periods that were missed due to system downtime, maintenance, or other reasons.

DAGs run

DAGS run is an object representing an instantiation of the DAG in time

To set DAG run schedule there are several options:

- **Cron expression:** You can pass any cron expression as a string to the schedule parameter in your DAG. For example, if you want to schedule your DAG at 4:05 AM every day, you would use schedule='5 4 * * *'.

- **Time Delta:** If you want to schedule your DAG on a particular cadence, rather than at a specific time, you can use Time Delta. For example, if you want to run your DAG every 30 minutes you would use: schedule=timedelta(minutes=30)

- **Cron Presets:** Airflow provides us with presets schedule like:
    a. **@once:** Schedule once and only once.
    b. **@hourly:** Run once an hour.
    c. **@daily:** Run once a day.

Thank you...