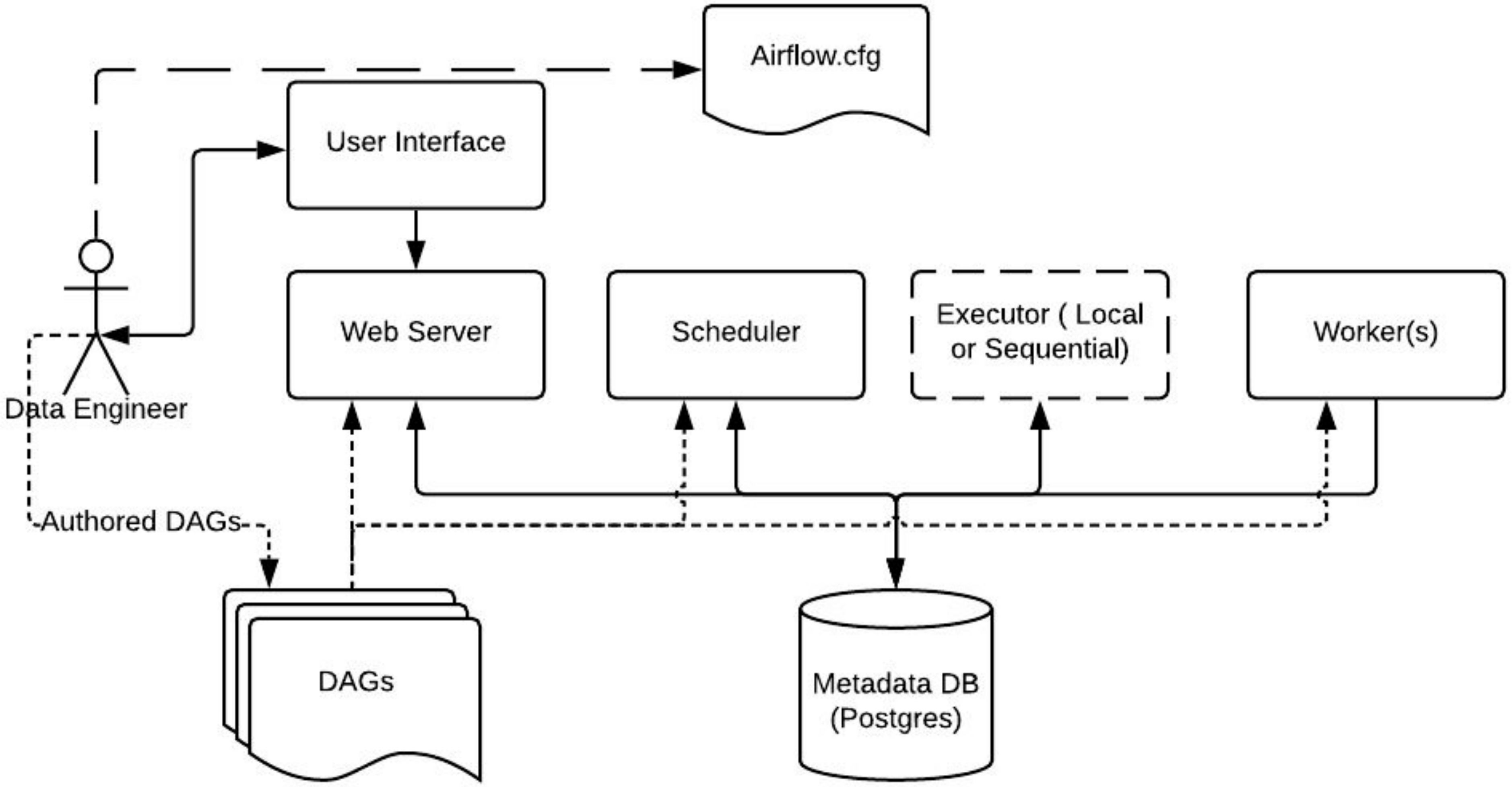
Apache Airflow

Platform to programmatically author, schedule and monitor workflows

Brief History

- Airflow was started in October 2014 by Maxime Beauchemin at Airbnb.
- It was open source from the very first commit and officially brought under the Airbnb GitHub and announced in June 2015.
- The project joined the Apache Software Foundation’s Incubator program in March 2016 and the Foundation announced Apache Airflow as a Top-Level Project in January 2019

Architecture



Airflow Components

- **Metadata Database:** Airflow uses a SQL database to store metadata about the data pipelines being run. In the diagram above, this is represented as Postgres which is extremely popular with Airflow. Alternate databases supported with Airflow include MySQL.

- **Web Server and Scheduler:** The Airflow web server and Scheduler are separate processes run (in this case) on the local machine and interact with the database mentioned above.

- **airflow.cfg** is the Airflow configuration file which is accessed by the Web Server, Scheduler, and Workers.

- **DAGs** refers to the DAG files containing Python code, representing the data pipelines to be run by Airflow. The location of these files is specified in the Airflow configuration file, but they need to be accessible by the Web Server, Scheduler, and Workers.

- The **Executor** is shown separately in architecture but runs within the Scheduler.

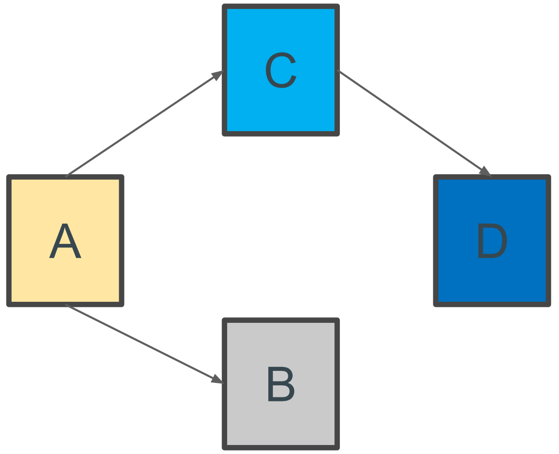
- The **Worker(s)** are separate processes which also interact with the other components of the Airflow architecture and the metadata repository.

Airflow Components

- **Task** defines a unit of work within a DAG; it is represented as a node in the DAG graph, and it is written in Python.
- **Directed Acyclic Graph** (DAG) – is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.
- **Operators** determine what actually gets done by a task. E.g. Bash Operator, Python Operator
- **Executors** are the mechanism by which task instances get run. E.g. SequentialExecutor, Celery Executor, Kubernetes Executor
- **Scheduler** monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete.

DAG Example

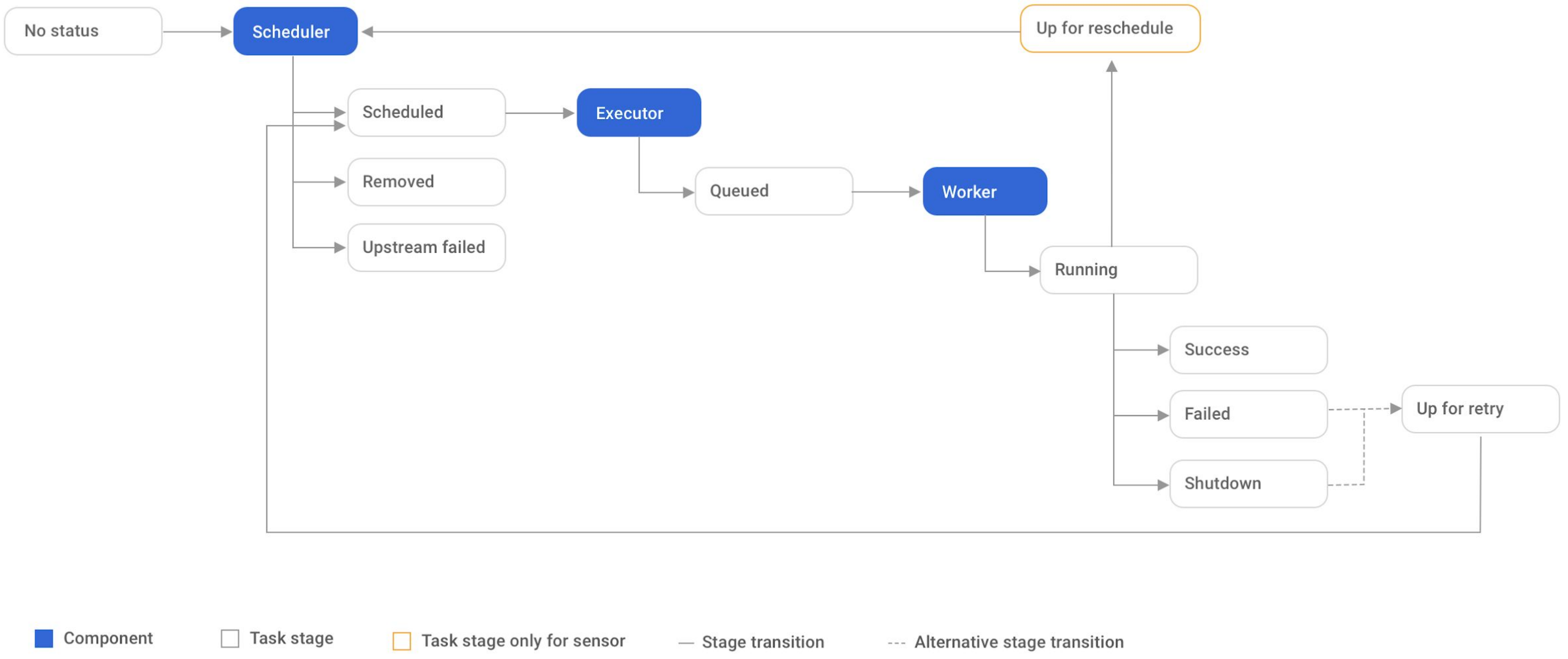
```
from airflow.models import DAG
from airflow.utils.dates import days_ago
from airflow.operators.dummy_operator import DummyOperator


with DAG(
  "etl_sales_daily",
  start_date=days_ago(1),
  schedule_interval=None,
) as dag:
   task_a = DummyOperator(task_id="task_a")
   task_b = DummyOperator(task_id="task_b")
   task_c = DummyOperator(task_id="task_c")
   task_d = DummyOperator(task_id="task_d")


   task_a >> [task_b, task_c]
   task_c >> task_d
```



Task Lifecycle

Task Status:

🟩 success 🟩 running 🟥 failed ▢ skipped 🟦 rescheduled 🟨 retry ⬛ queued ⬜ no status



Applications

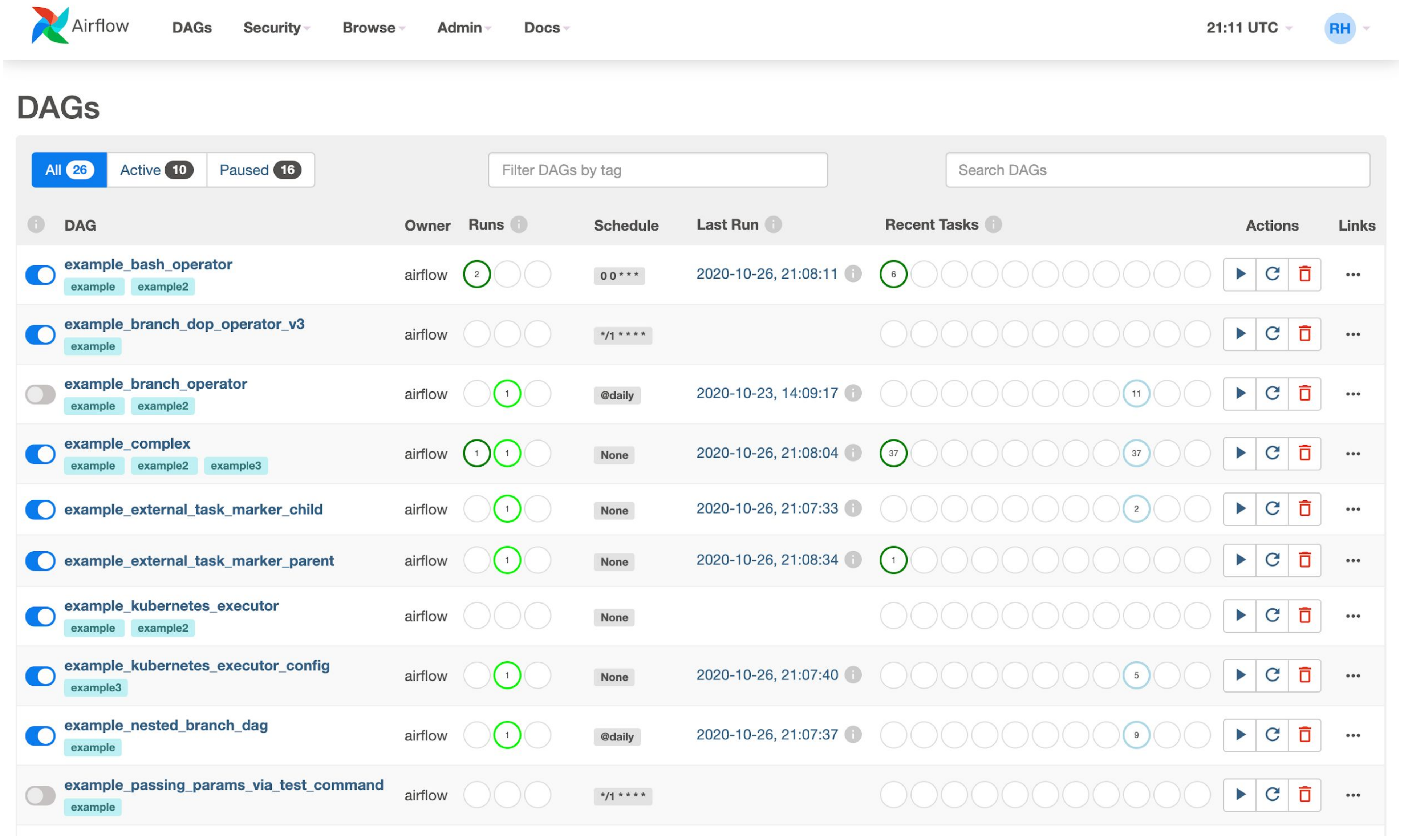
Apache Airflow can be used to schedule:

- ETL pipelines that extract data from multiple sources and run Spark jobs or any other data transformations
- Training machine learning models
- Report generation
- Backups and similar DevOps operations

Airflow Setup

- https://airflow.apache.org/docs/apache-airflow/stable/start/docker.html
- https://github.com/krulletc/airflow-dag-etl-demo/

Demo - Interface



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 * * * *				...

Celery

Distributed Task Queue

Celery

- Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system.

- It’s a task queue with focus on real-time processing, while also supporting task scheduling.

- Celery instance / *Celery application* or just celery *app* used as the entry-point for everything you want to do in Celery, like creating tasks and managing workers, it must be possible for other modules to import it.

Ref: https://docs.celeryproject.org/en/stable/getting-started/introduction.html#celery-is