Airflow

What is Apache Airflow

Apache Airflow is an open-source platform to Author, Schedule and Monitor workflows.

It was created at Airbnb and currently is a part of **Apache** Software Foundation.

Airflow helps you to create workflows using Python programming language and these workflows can be scheduled and monitored easily with it.

Some use cases

Automate training, testing and deploying a machine learning model

Ingesting data from multiple REST APIs

Coordinating Extraction, Transformation and Loading (ETL) or Extraction, Loading and Transformation (ELT) Processes Across an Enterprise Data Lake

As we can see, one of the main features of **Airflow** is its flexibility: it can be used for many different data workflow scenarios.

Due to this aspect and its rich feature set, it has gained significant traction over the years.

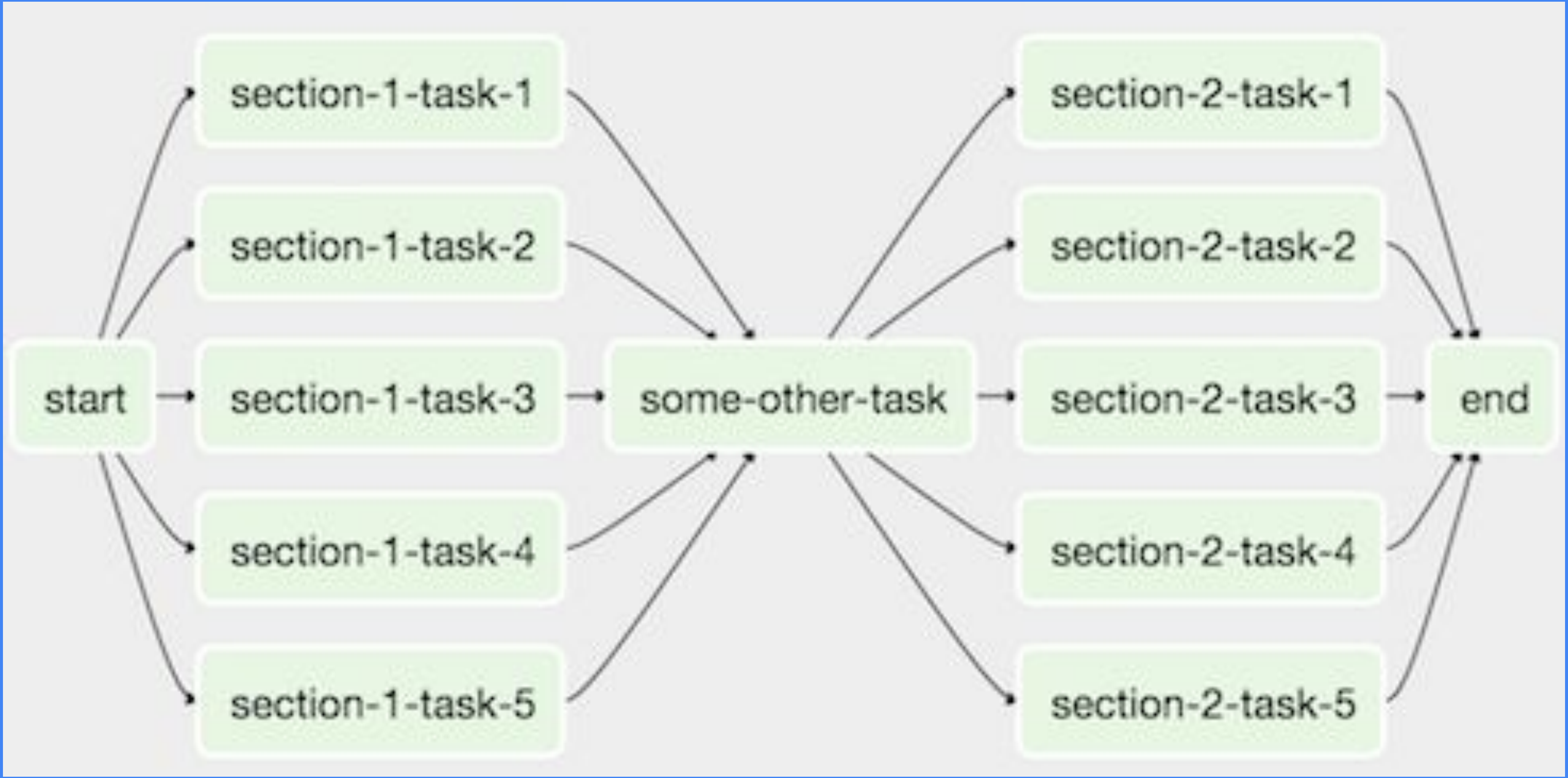
It has been battle-tested by many companies, from startups to Fortune 500 enterprises.

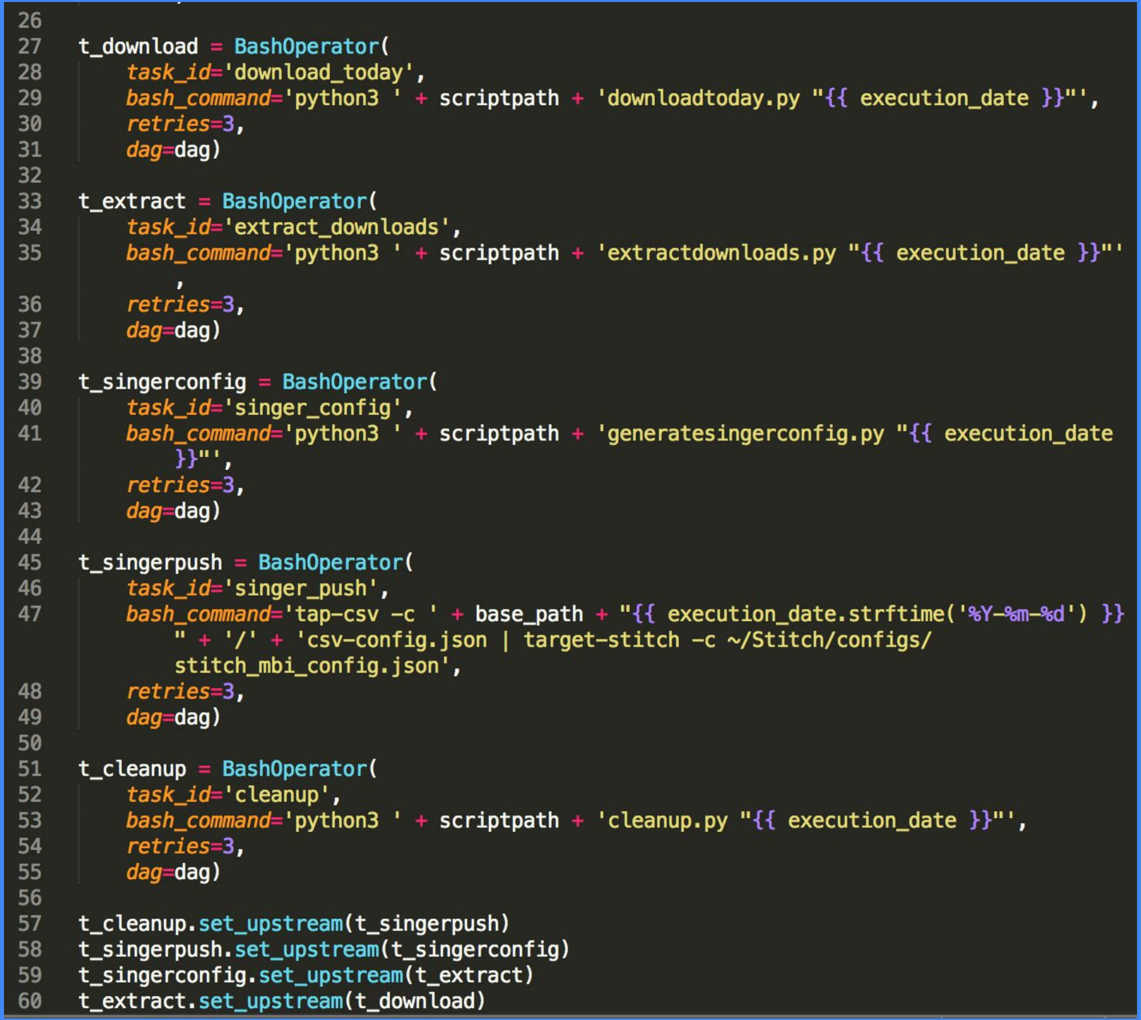
Some examples include Spotify, Twitter, Walmart, Slack, Robinhood, Reddit, PayPal, Lyft, and of course, Airbnb.

Apache DAGs

Apache Airflow works with the concept of Directed Acyclic Graphs (DAGs), which are a powerful way of defining dependencies across different types of tasks.

In **Apache Airflow**, DAGs are developed in **Python**, which unlocks many interesting features from software engineering: modularity, reusability, readability, among others.


start
section-1-task-1
section-1-task-2
section-1-task-3
section-1-task-4
section-1-task-5
some-other-task
section-2-task-1
section-2-task-2
section-2-task-3
section-2-task-4
section-2-task-5
end



```
t_download = BashOperator(
    task_id='download_today',
    bash_command='python3 ' + scriptpath + 'downloadtoday.py "{{ execution_date }}"',
    retries=3,
    dag=dag)

t_extract = BashOperator(
    task_id='extract_downloads',
    bash_command='python3 ' + scriptpath + 'extractdownloads.py "{{ execution_date }}"'
        ,
    retries=3,
    dag=dag)

t_singerconfig = BashOperator(
    task_id='singer_config',
    bash_command='python3 ' + scriptpath + 'generatesingerconfig.py "{{ execution_date
        }}"',
    retries=3,
    dag=dag)

t_singerpush = BashOperator(
    task_id='singer_push',
    bash_command='tap-csv -c ' + base_path + "{{ execution_date.strftime('%Y-%m-%d') }}
        " + '/' + 'csv-config.json | target-stitch -c ~/Stitch/configs/
        stitch_mbi_config.json',
    retries=3,
    dag=dag)

t_cleanup = BashOperator(
    task_id='cleanup',
    bash_command='python3 ' + scriptpath + 'cleanup.py "{{ execution_date }}"',
    retries=3,
    dag=dag)

t_cleanup.set_upstream(t_singerpush)
t_singerpush.set_upstream(t_singerconfig)
t_singerconfig.set_upstream(t_extract)
t_extract.set_upstream(t_download)
```

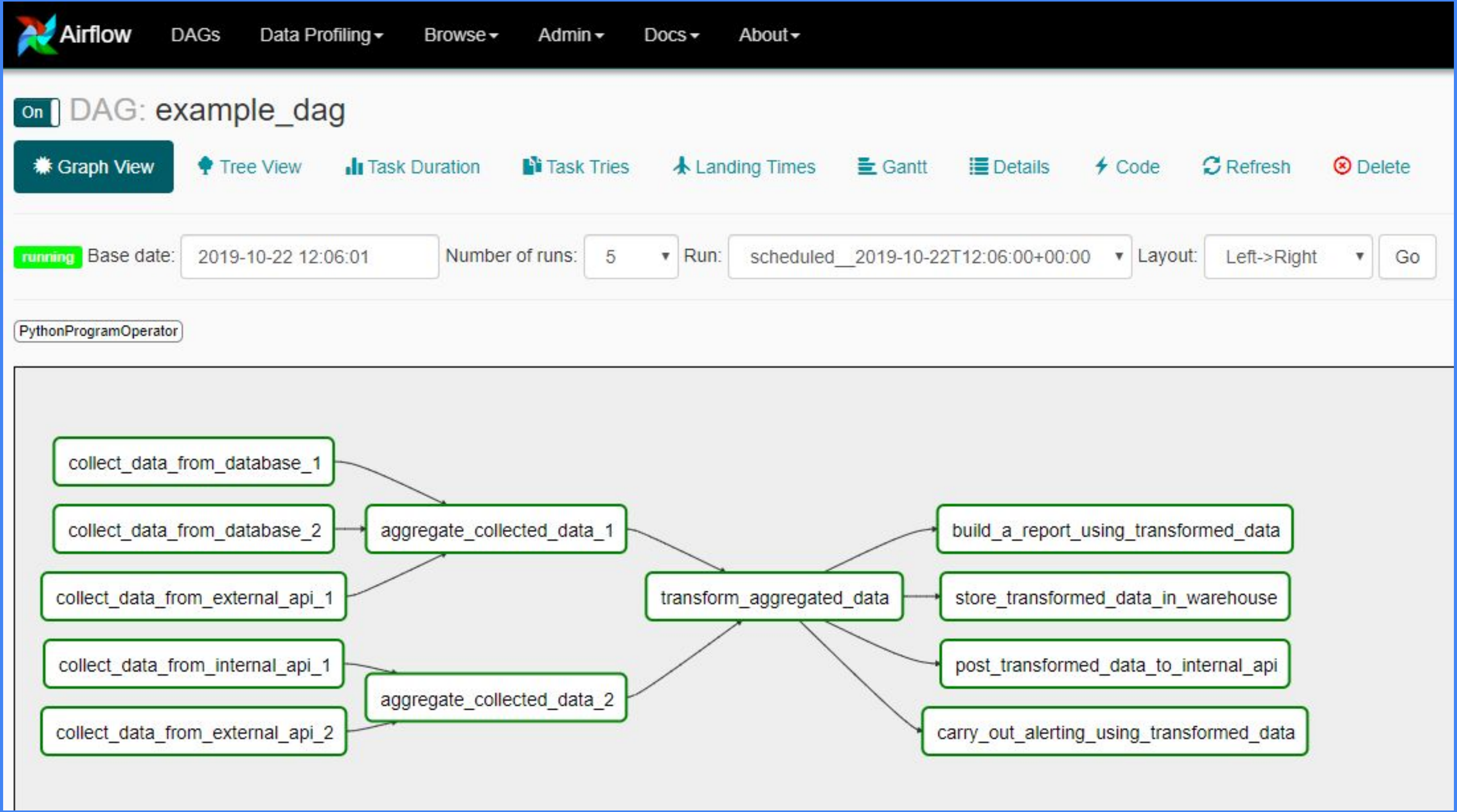
Apache Sensors, Hooks, Operator

Sensors, Hooks, and Operators are the main building blocks of Apache Airflow. They provide
an easy and flexible way of connecting to multiple external resources. Want to integrate with Amazon S3? Maybe you also want to add Azure Container Instances to the picture in order to run some short-lived Docker containers? Perhaps running a batch workload in an Apache Spark or Databricks cluster? Or maybe just executing some basic Python code to connect with REST APIs? Airflow ships with multiple operators, hooks, and sensors out of the box, which allow for easy integration with these resources, and many more, such as DockerOperator, BashOperator, HiveOperator, JDBCOperator — the list goes on. You can also build upon one of the standard operators and create your own. Or you can simply write your own operators, hooks, and sensors from scratch.

Airflow UI

The UI allows for quick and easy monitoring and management of your Airflow instance.

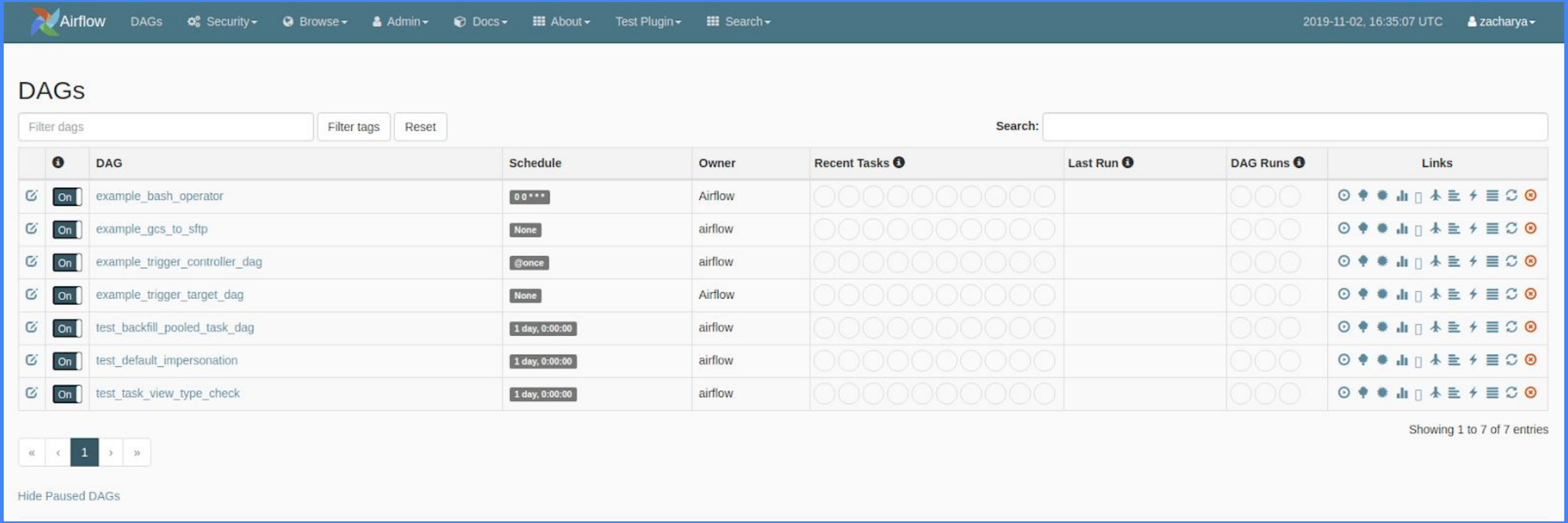
Detailed logs also make it easier to debug your DAGs.



DAG: example_dag

Base date: 2019-10-22 12:06:01 Number of runs: 5 Run: scheduled__2019-10-22T12:06:00+00:00 Layout: Left->Right

$$\begin{tikzcd}
\text{collect\_data\_from\_database\_1} \arrow[dr] & & & \text{build\_a\_report\_using\_transformed\_data} \\
\text{collect\_data\_from\_database\_2} \arrow[r] & \text{aggregate\_collected\_data\_1} \arrow[dr] & & \text{store\_transformed\_data\_in\_warehouse} \\
\text{collect\_data\_from\_external\_api\_1} \arrow[ur] & & \text{transform\_aggregated\_data} \arrow[uur] \arrow[ur] \arrow[r] \arrow[dr] & \text{post\_transformed\_data\_to\_internal\_api} \\
\text{collect\_data\_from\_internal\_api\_1} \arrow[r] & \text{aggregate\_collected\_data\_2} \arrow[ur] & & \text{carry\_out\_alerting\_using\_transformed\_data} \\
\text{collect\_data\_from\_external\_api\_2} \arrow[ur] & & &
\end{tikzcd}$$



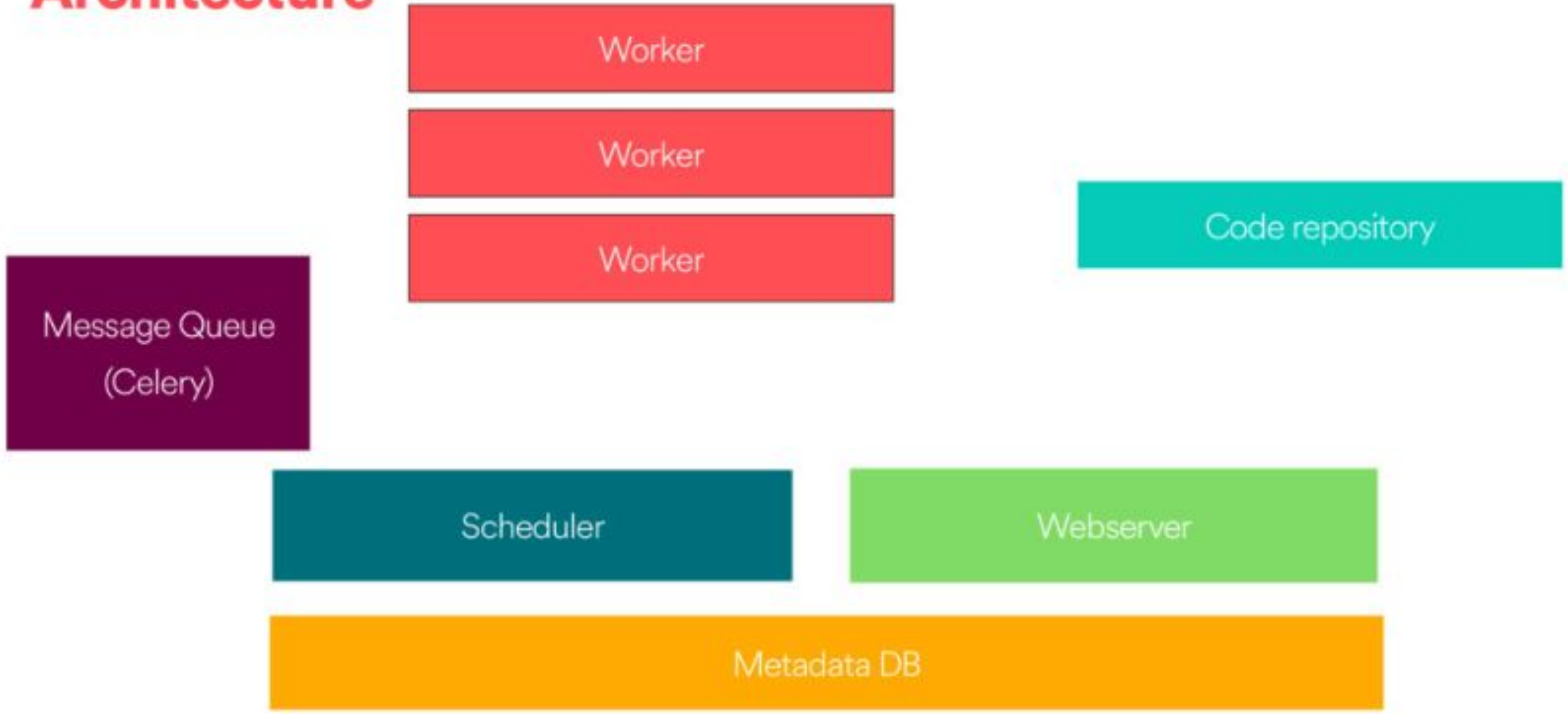
DAGs

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	example_bash_operator	0 0 * * *	Airflow				
	On	example_gcs_to_sftp	None	airflow				
	On	example_trigger_controller_dag	@once	airflow				
	On	example_trigger_target_dag	None	Airflow				
	On	test_backfill_pooled_task_dag	1 day, 0:00:00	airflow				
	On	test_default_impersonation	1 day, 0:00:00	airflow				
	On	test_task_view_type_check	1 day, 0:00:00	airflow				

Showing 1 to 7 of 7 entries

Airflow Components

Architecture



Web Server

The GUI. This is under the hood a Flask app where you can track the status of your jobs and read logs from a remote file store

Scheduler

This component is responsible for scheduling jobs. This is a multithreaded Python process that uses the DAGb object to decide what tasks need to be run, when and where.

The task state is retrieved and updated from the database accordingly. The web server then uses these saved states to display job information.

Executor

The mechanism that gets the tasks done.

Metadata Database

- Powers how the other components interact
- Stores the Airflow states
- All processes read and write from here

Questions?