How Airflow work?

Why you should use it?

What is async operator/sensor?

Airflow Workflow

High Level Architecture

Async Operator

Why Airflow

Use Cases

Hi,

I’m Pankaj Singh

- Software Engineer at Astronomer
- Maintainer of astronomer-provider repository
- Apache Airflow contributor

What is Airflow?

Apache Airflow

Airflow is a platform created by the community to programmatically author, schedule and monitor workflows.

Install

What is an Airflow DAG?

Airflow DAG

1. It is a Python file containing DAG (Directed Acyclic Graph) object
2. DAG contains collection of tasks (operators/sensors)
3. A task usually perform a single unit of work
4. A task may have relationships with other tasks
5. Airflow executes a task according to the task's relationships

How does an Airflow DAG look?

Airflow Workflow

Airflow DAG

```python
def get_history():
    histories = Brave().fetch_history().histories
    today = datetime.today()
    histories = [
        { "date": x[0], "query": x[1]}
        for x in histories if x[0].strftime('%Y-%m-%d') == today.strftime('%Y-%m-%d')
    ]
    with open('brave_history.csv', 'w') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=['date', 'query'])
        writer.writeheader()
        writer.writerows(histories)


with DAG(dag_id="record_brave_history", schedule_interval="@daily", start_date=datetime(2022, 1, 1)) as
dag:
    read_history = PythonOperator(task_id="read_history", python_callable=get_history)

    push_to_gcs = LocalFilesystemToGCSOperator(
        task_id="push_to_gcs",
        src="brave_history.csv",
        dst="tmp/brave_history.csv",
        bucket="project",
    )

    load_to_bigquery = GCSToBigQueryOperator(
        task_id='load_to_bigquery',
        bucket=GCS_BUCKET,
        source_objects=[GCS_OBJ_PATH],
        destination_project_dataset_table=f"{DATASET_NAME}.{TABLE_NAME}",
    )

    read_history >> push_to_gcs >> load_to_bigquery
```

Concept

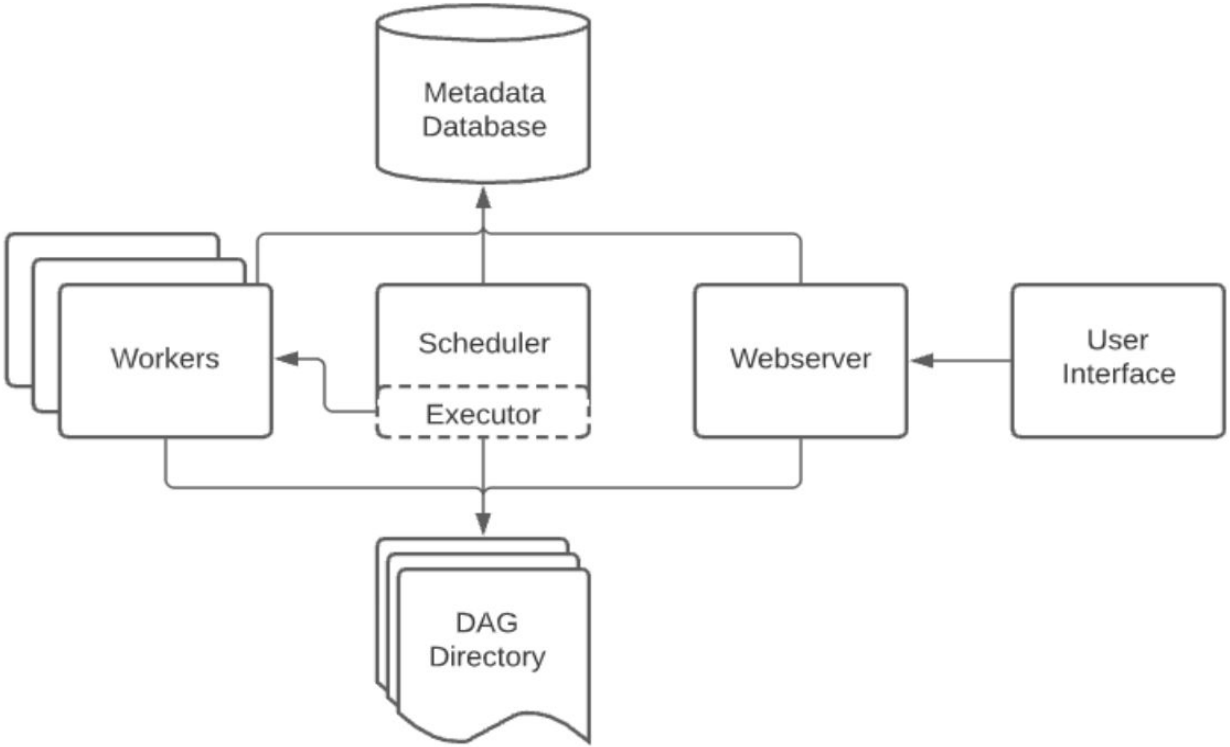
DAG → DagRun (one per scheduled run, as run starts)

Operator → Task (When you call an Operator in a DAG)

Task → TaskInstance (When task need to run as part of a DagRun)

How does Airflow work?

Airflow Architecture



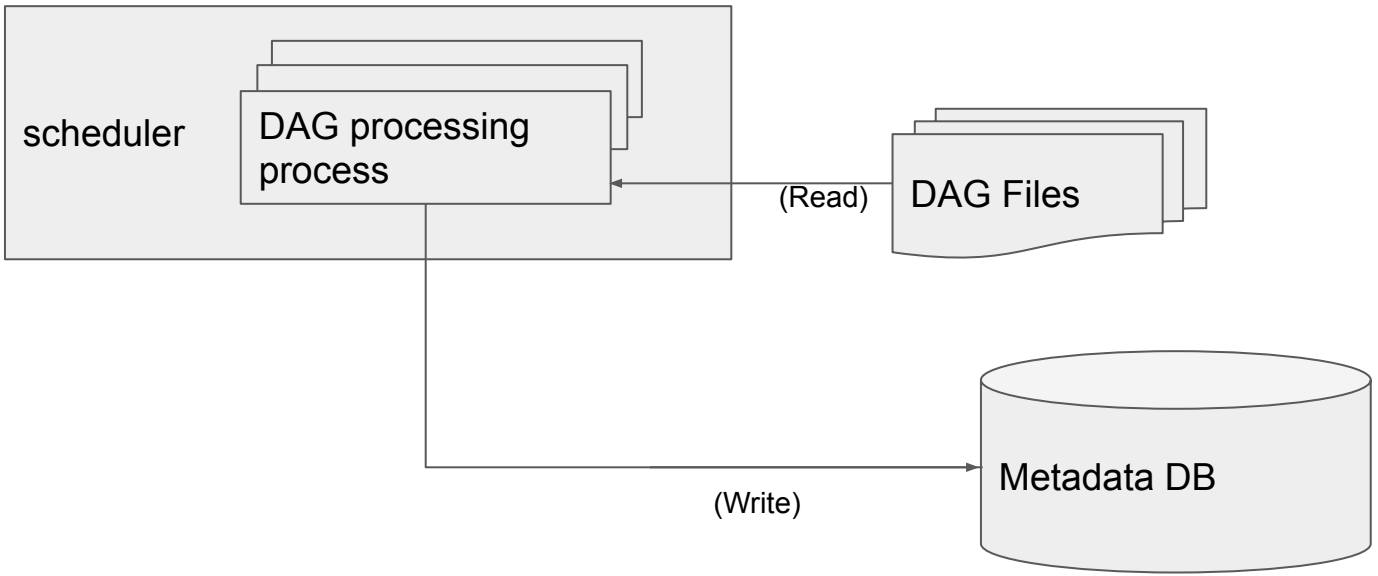
Let’s break it!

DAG Parsing

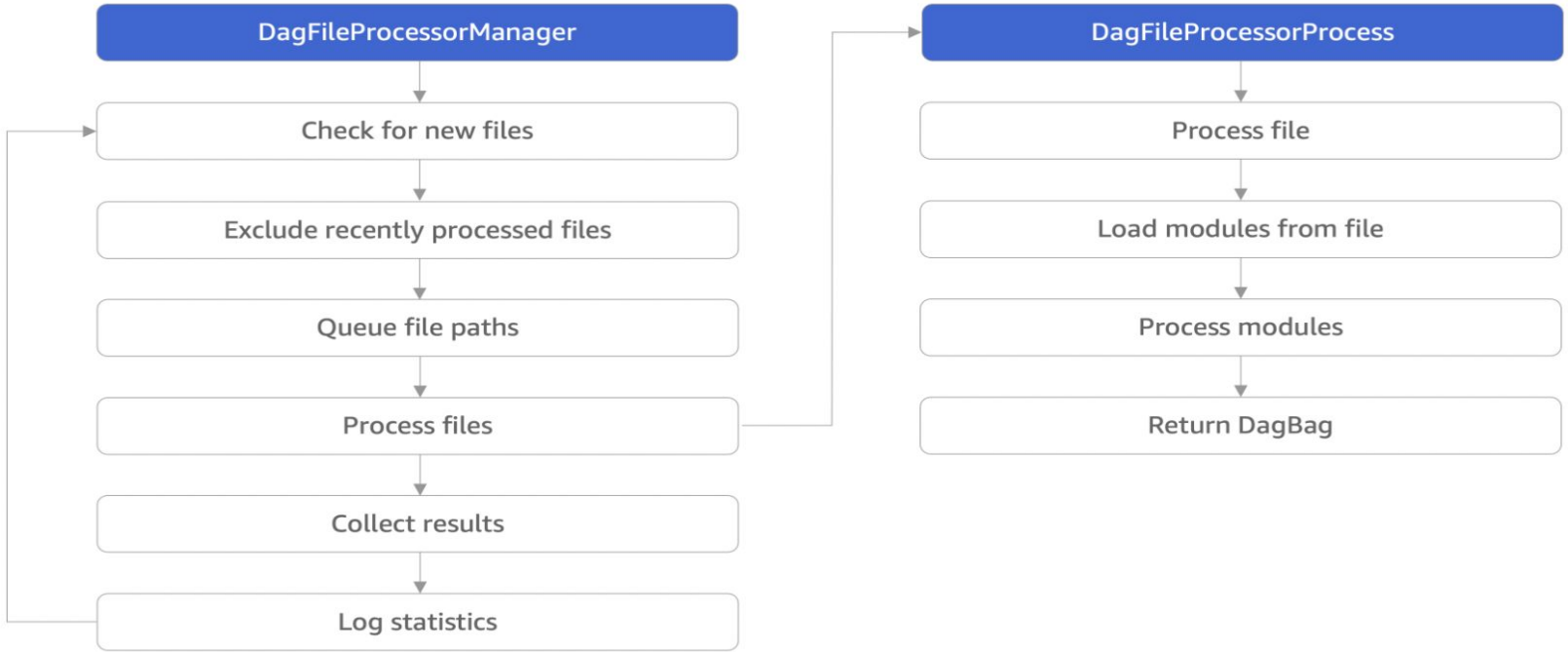
DAG File Processing

- dag_processing module is responsible for turning Python file into DAG object
- It has two main component
  - DagFileProcessorManager
  - DagFileProcessorProcess
- You can run it as standalone process by configuring AIRFLOW__SCHEDULER__STANDALONE_DAG_PROCESSOR=TRUE and run the airflow dag-processor CLI command
- By default starting scheduler process start DagFileProcessorManager

DAG File Processing



DAG File Processing

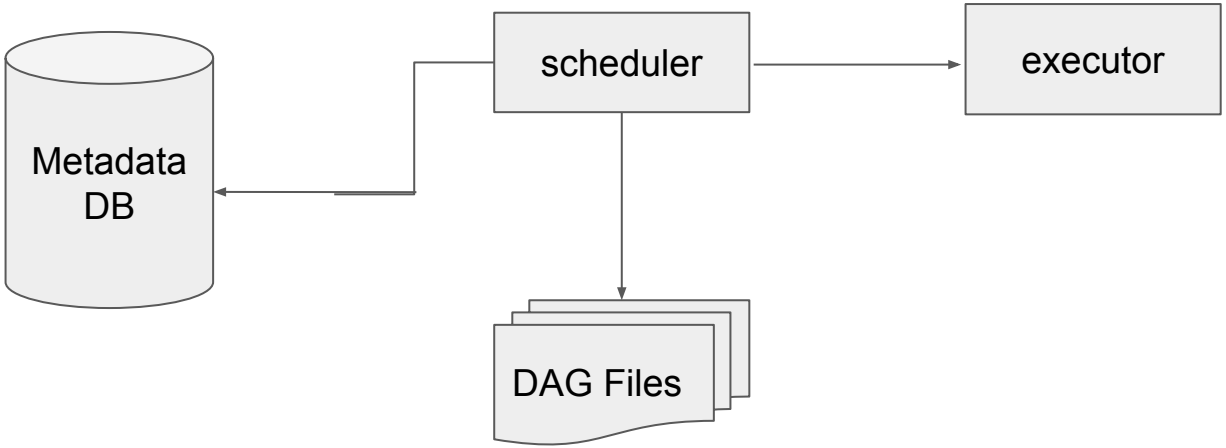


The Scheduler

Airflow Scheduler

- Python subprocess
- Monitor tasks and DAGs
- Creating DagRuns and taskInstance once their dependencies are complete
- It highly available

Airflow Scheduler

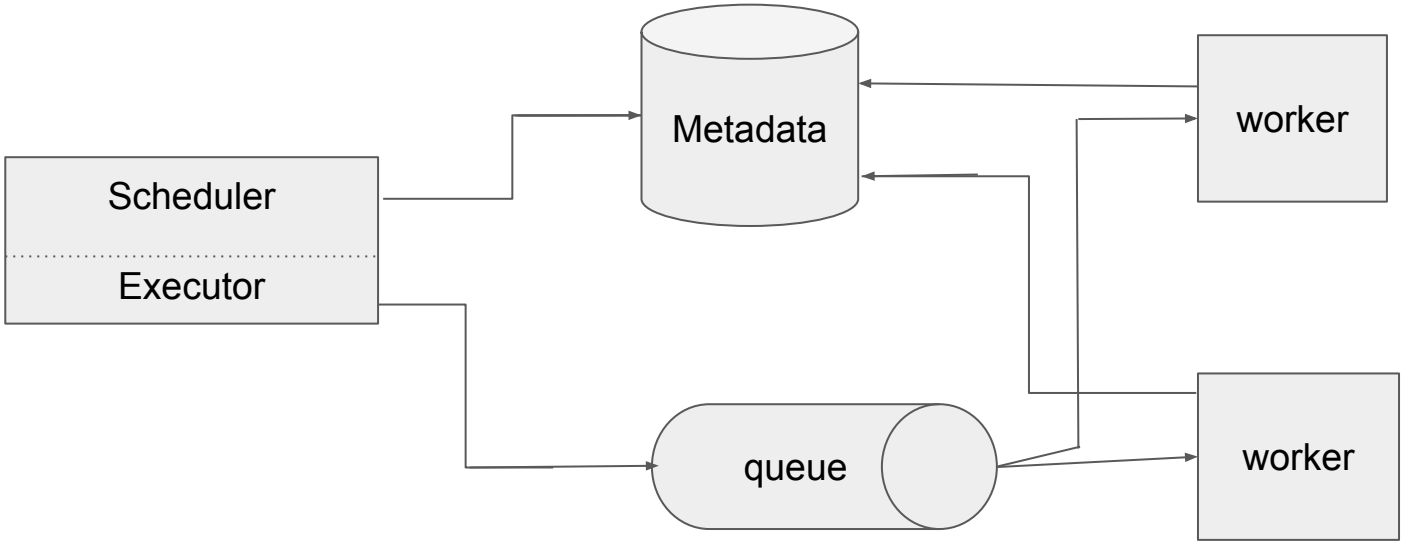


Executor

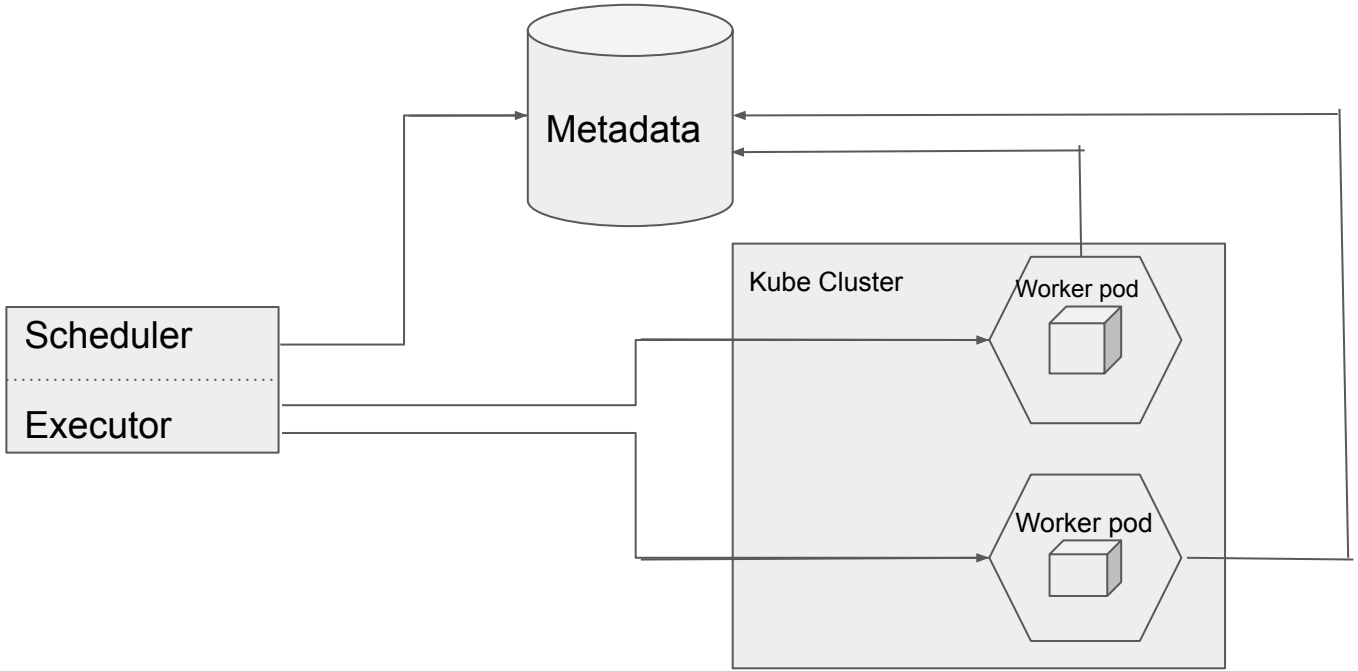
Airflow Executor

- It take a charge of running the task instance
- Executor types
    - Local Executor: run task inside scheduler process
        - Local executor
        - Sequential executor
    - Remote Executor: run tasks remotely usually via worker pool
        - Celery executor
        - Kubernetes executor
    - Hybrid Executor: run tasks based on task queue
        - CeleryKubernetes executor

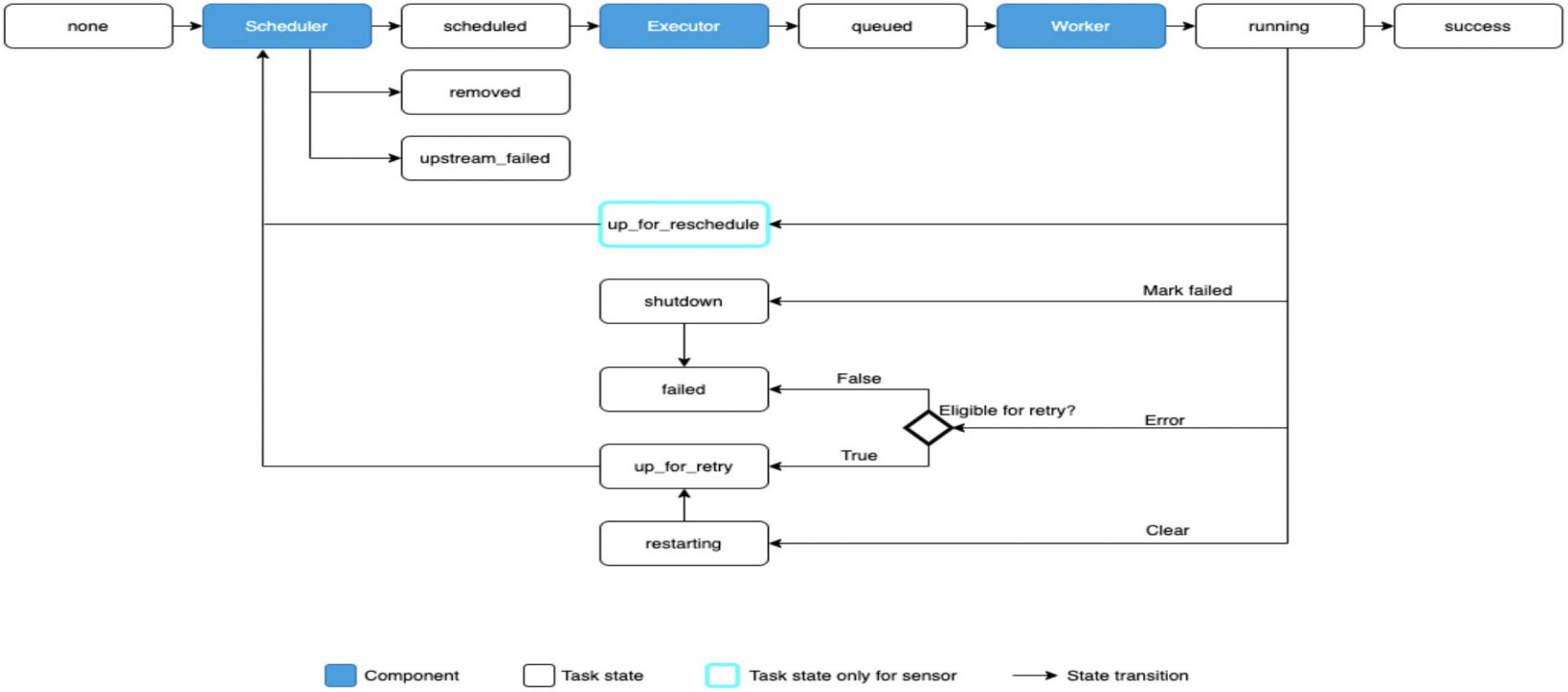
Celery Executor



Kubernetes Executor



Task Life Cycle

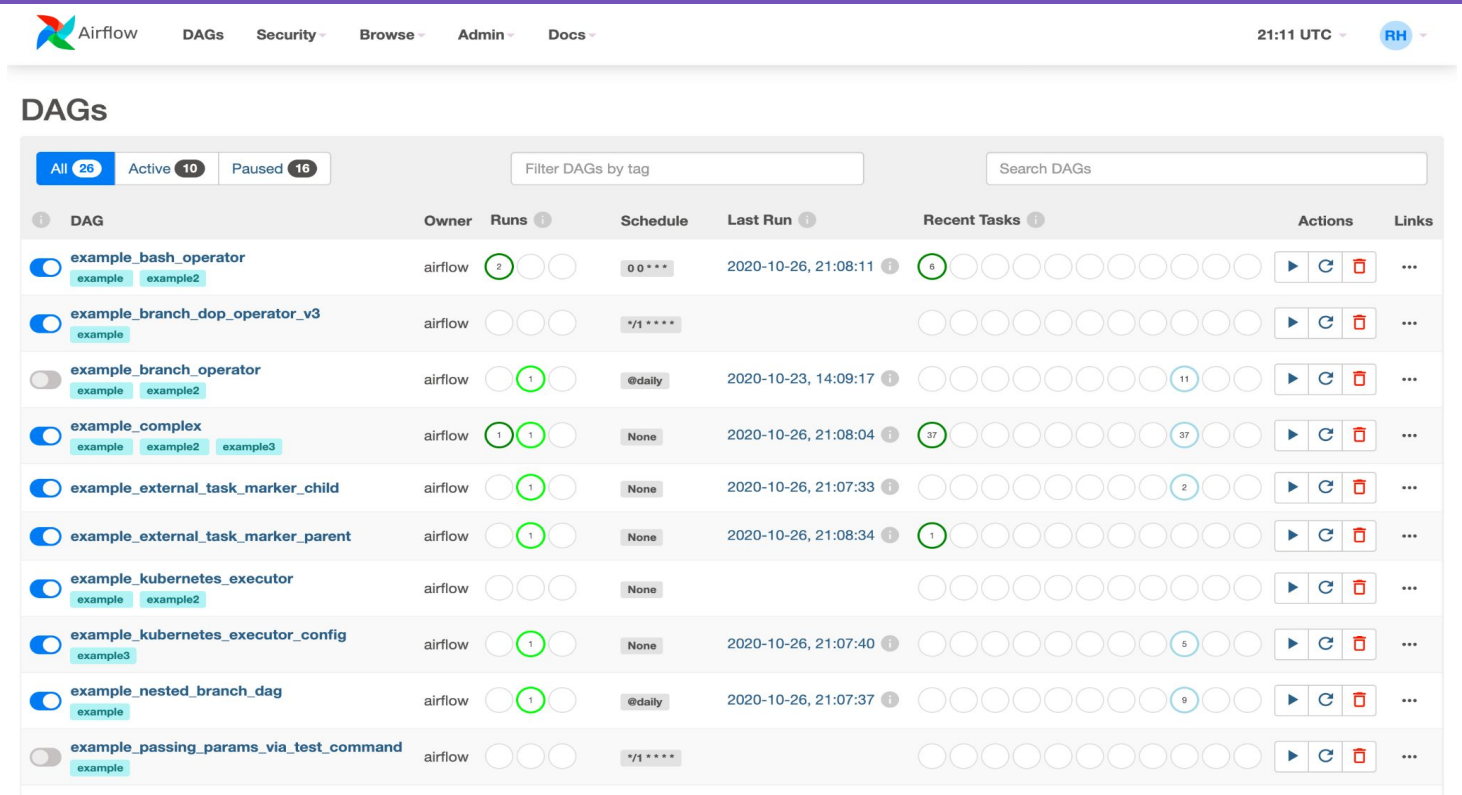


Webserver

Airflow Webserver

- A Flask server that serves the Airflow UI

Airflow Webserver

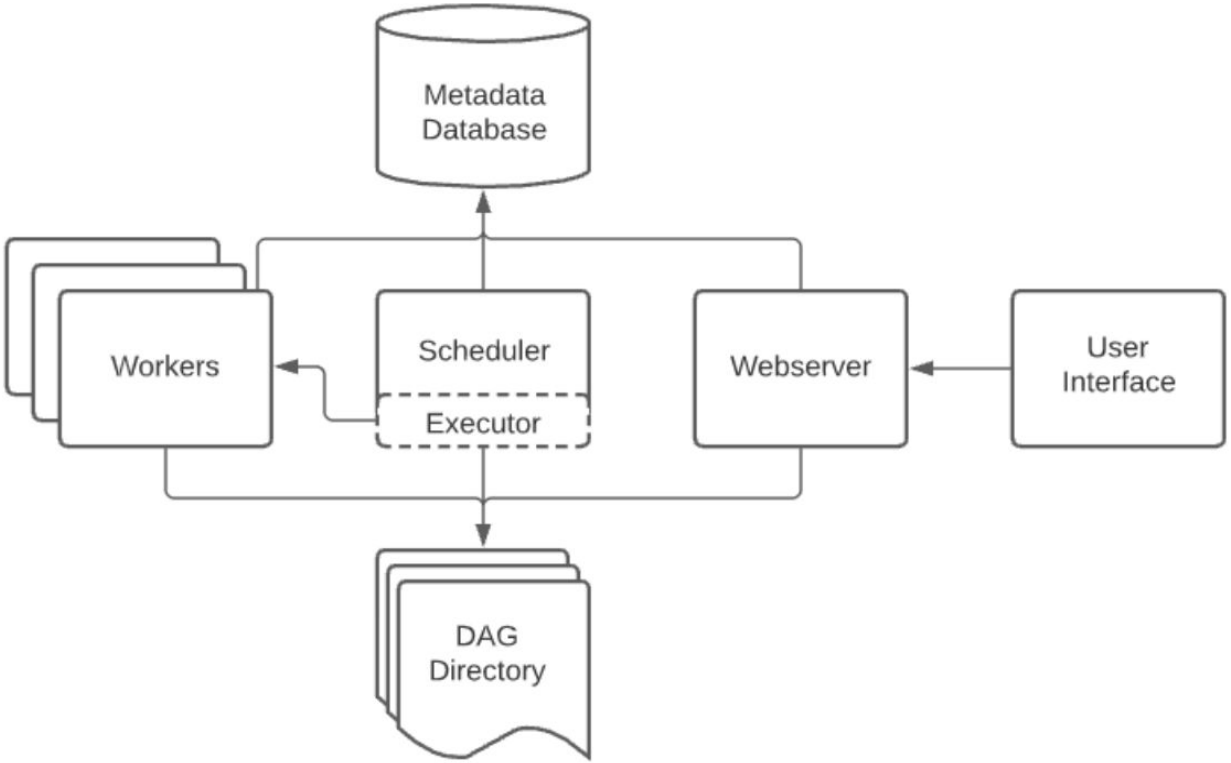


Metadata DB

Airflow Metadata Database

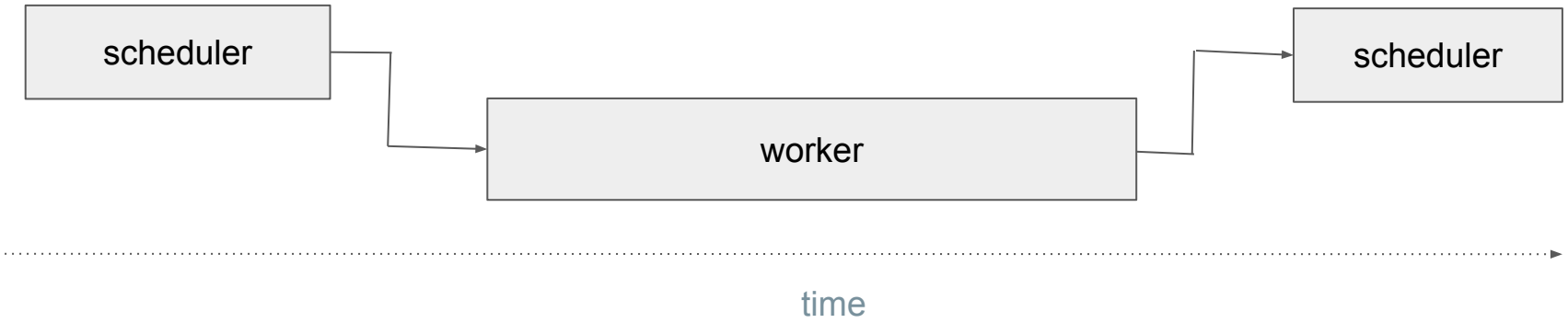
- Brain of Airflow
- It stores information like the connection of your Airflow variables roles and permissions, as well as all metadata for past and present DAG and task runs
- Airflow uses SQLAlchemy as Object Relational Mapping (ORM) in Python

Airflow Architecture



What is async operators/sensors?

Task Execution



What is Airflow triggerer?

Airflow Triggerer

Triggers: These are small, asynchronous pieces of Python code

Triggerer: A Airflow service that runs an asyncio event loop in your Airflow environment

Deferred: A task state to indicate that a task has paused its execution, released the worker slot, and submitted a trigger to be picked up by the triggerer process.

Triggerer Example

```python
import asyncio

from airflow.triggers.base import BaseTrigger, TriggerEvent
from airflow.utils import timezone


class DateTimeTrigger(BaseTrigger):

    def __init__(self, moment):
        super().__init__()
        self.moment = moment

    def serialize(self):
        return ("airflow.triggers.temporal.DateTimeTrigger", {"moment": self.moment})

    async def run(self):
        while self.moment > timezone.utcnow():
            await asyncio.sleep(1)
        yield TriggerEvent(self.moment)
```

Async Sensor

```python
from datetime import timedelta

from airflow.sensors.base import BaseSensorOperator
from airflow.triggers.temporal import TimeDeltaTrigger


class WaitOneHourSensor(BaseSensorOperator):
    def execute(self, context):
        self.defer(trigger=TimeDeltaTrigger(timedelta(hours=1)), method_name="execute_complete")

    def execute_complete(self, context, event=None):
        # We have no more work to do here. Mark as complete.
        return
```

Benchmark

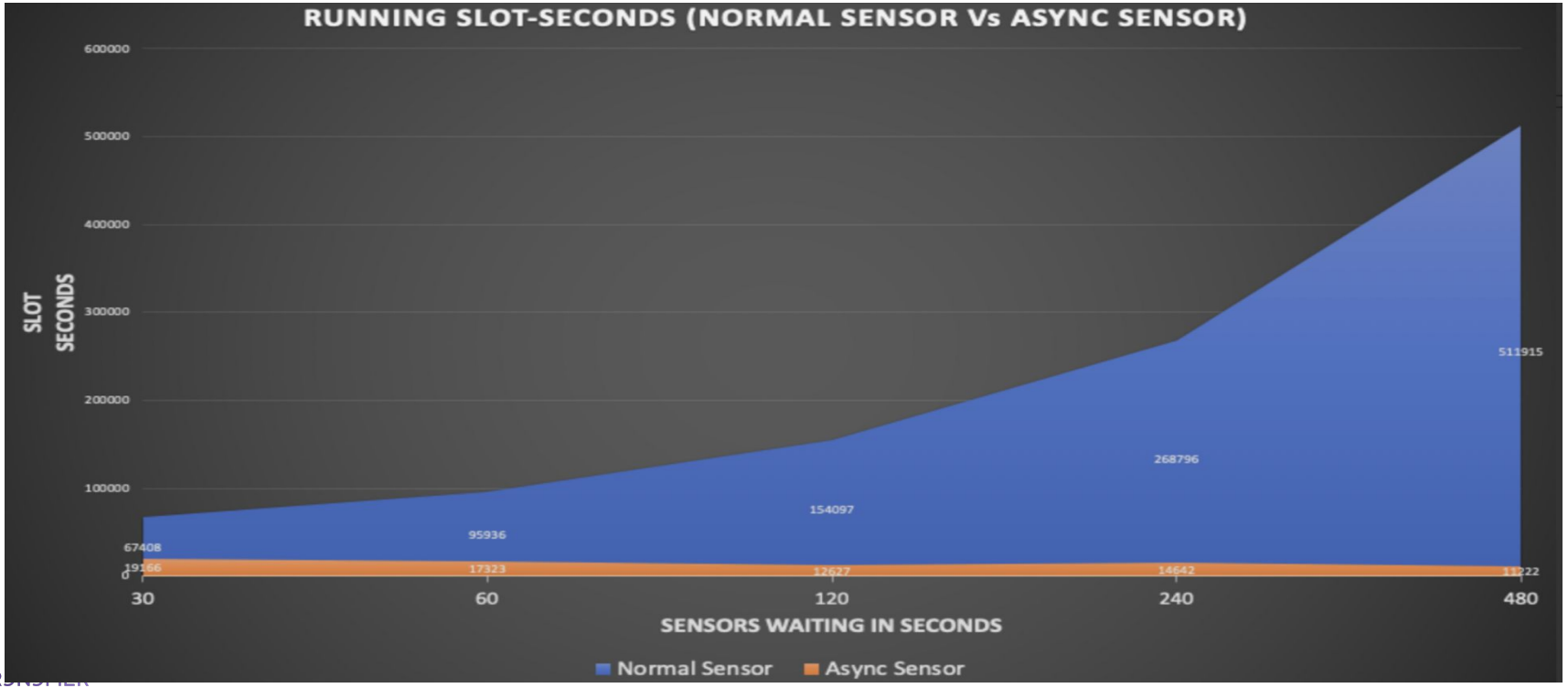
- Conducted an experiment to measure the impact of using deferred/async sensors via a trigger compared to regular sensors that occupy a slot while running. Our hypothesis was that we will observe lower slot consumption on deferred/async sensors.
- Metrics used: The total number of minutes a task spends running on a worker, summed over all tasks (slot usage seconds)

Bench Mark

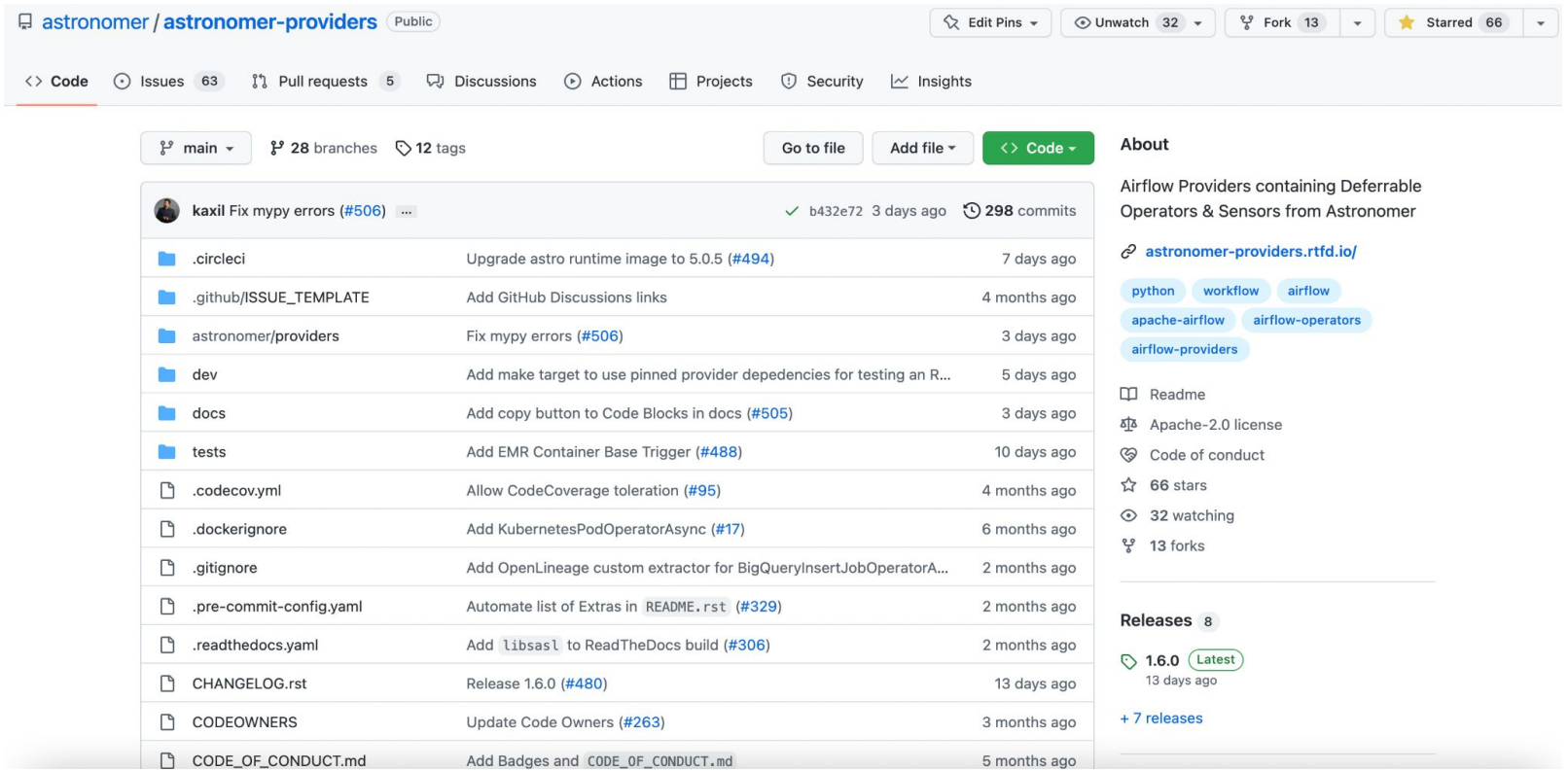
Slots-seconds for a 1000-task DAG run

sleep before create	running	running (async)	improvement	occupied	occupied (async)	improvement
30s	67408	19166	71.6%	92500	61798	33.2%
1m	95936	17323	81.9%	124752	55741	55.3%
2m	154097	12627	91.8%	186224	41135	77.9%
4m	268796	14642	94.6%	300157	45061	85.0%
8m	511915	11222	97.8%	542793	35890	93.4%

Benchmark

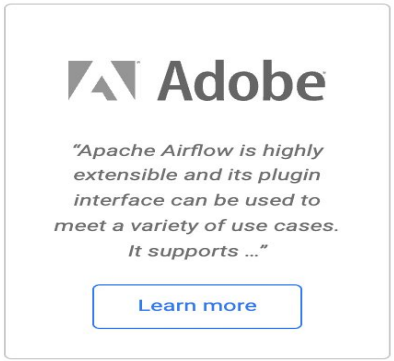

RUNNING SLOT-SECONDS (NORMAL SENSOR Vs ASYNC SENSOR)
SLOT SECONDS
600000
500000
400000
300000
200000
100000
0
67408
95936
154097
268796
511915
19166
17323
12627
14642
11222
30
60
120
240
480
SENSORS WAITING IN SECONDS
Normal Sensor
Async Sensor

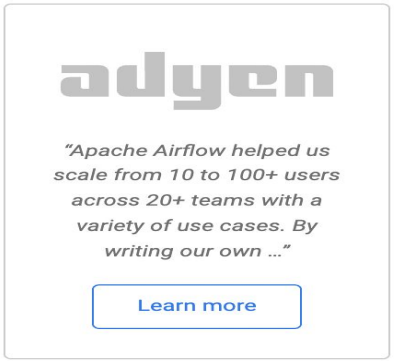
Astronomer Providers

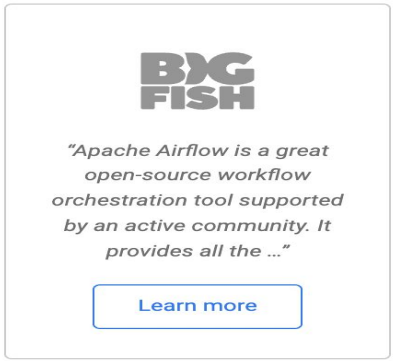


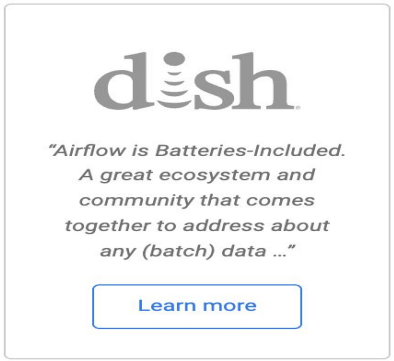
Use Cases

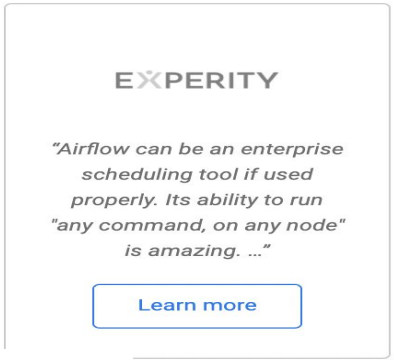
Use Cases

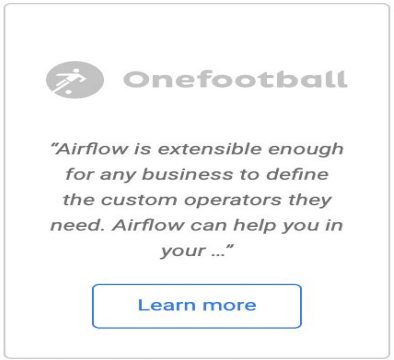

Adobe
"Apache Airflow is highly extensible and its plugin interface can be used to meet a variety of use cases. It supports ..."
Learn more

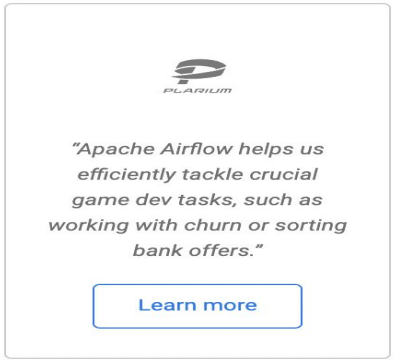

adyen
"Apache Airflow helped us scale from 10 to 100+ users across 20+ teams with a variety of use cases. By writing our own ..."
Learn more

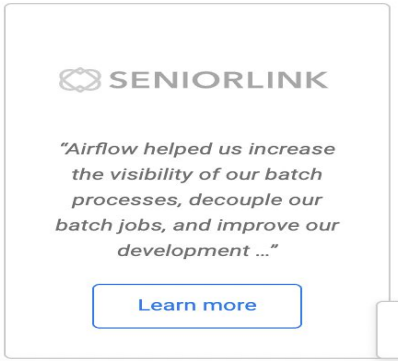

BIG FISH
"Apache Airflow is a great open-source workflow orchestration tool supported by an active community. It provides all the ..."
Learn more


dish
"Airflow is Batteries-Included. A great ecosystem and community that comes together to address about any (batch) data ..."
Learn more


EXPERITY
"Airflow can be an enterprise scheduling tool if used properly. Its ability to run "any command, on any node" is amazing. ..."
Learn more


Onefootball
"Airflow is extensible enough for any business to define the custom operators they need. Airflow can help you in your ..."
Learn more


PLARIUM
"Apache Airflow helps us efficiently tackle crucial game dev tasks, such as working with churn or sorting bank offers."
Learn more


SENIORLINK
"Airflow helped us increase the visibility of our batch processes, decouple our batch jobs, and improve our development ..."
Learn more

DISHA

DISHA

- National data platform for world largest democracy where MPs and MLAs can monitor the progress of national level schemes

“DISHA is a crucial step towards good governance through which we will be able to monitor everything centrally. It will enable us to effectively monitor every village of the country”

Why you should use it?

Why Airflow?

1. It is open source and part of prestigious Apache Software Foundation that has project like Spark etc
    a. 26K+ star, 10K+ fork, 2k+ contributor
2. Great community (Slack, Github, Airflow Summit)
3. Highly extensible
4. Scalable (Webserver, Scheduler, Worker, Triggerer)
5. Rich feature
    a. Beautiful UI
    b. 75+ client integration available
    c. Awesome CLI
    d. Docker image and Helm Chart
    e. . . .

Airflow Managed Service

ASTRONOMER

Google Cloud

Thank you