Airflow 101

Agenda

- ❏ Who I am
- ❏ What is Airflow?
- ❏ Why Airflow?
- ❏ Airflow at its Core
- ❏ Concepts
- ❏ Real Time Example
- ❏ Airflow at Jampp
- ❏ Tips for a better experience

Who I am

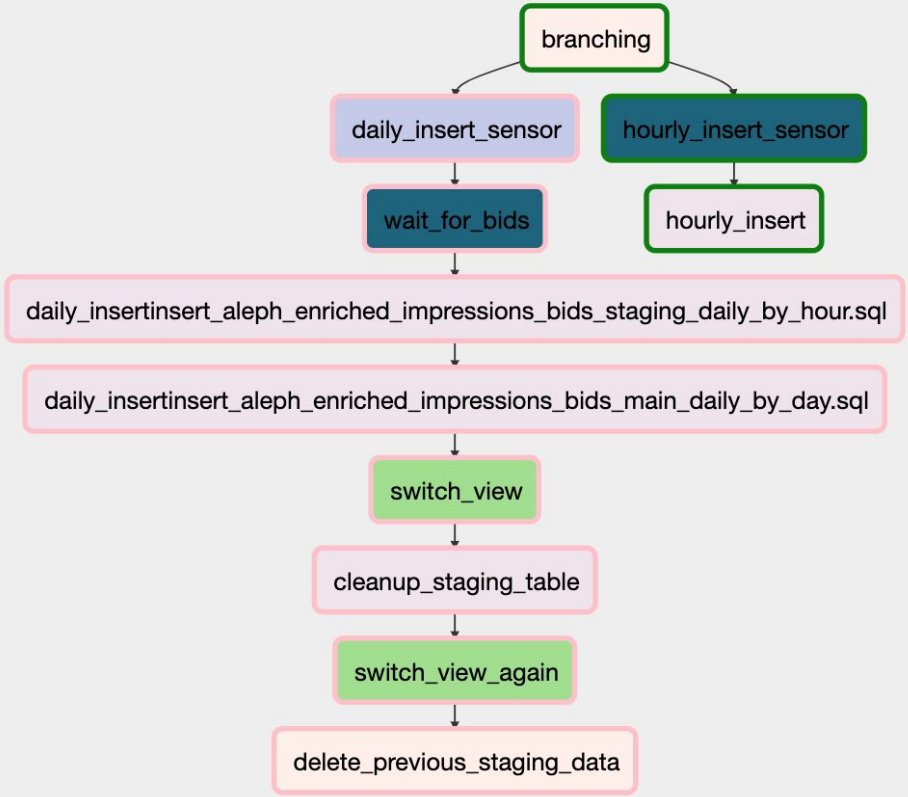
- Data Analytics Developer at Jampp for the last year and a half.
    - Development of reporting tools.
    - Automated periodical reports for clients.
    - Automated pipelines for A / B tests.
- (Aspiring) Physicist
- Decent guitar player and mid - defender on saturdays and sundays.

What is Airflow?

“Airflow is a platform to programmatically author, schedule and monitor workflows.”

- Through **DAGs** (Directed Acyclic Graphs), define the stream of dependencies between tasks to execute a workflow.
- Built for **scaling**.
- **Opensource**, end - to - end Python.

What is Airflow?



- Pipelines are defined by a **set of tasks**, that should have **defined dependencies**.
- **Each node is a task**, connected by the way they depend from each other.
- One restriction: Can’t have cycles.

Why Airflow?

- **Open Source**:
  - ~~Incubation~~ **Top-Level Project at Apache**, used by top companies in the world.
- **Web Interface**:
  - Very **comfortable and intuitive UI**. Facilitates the monitoring of every process.
- **End - to - end Python**:
  - The whole of Airflow is built on python, the same code use to define the workflow executes them, what makes it **easy to debug and read**.

Airflow at its Core

Scheduler

- **Brains** behind the whole thing.
- Regularly **scans the DAG FOLDER** looking for the code needed to generate the DAGs and **loads them** into the db.
- **Orders workers** if and when they should run each task.
- Practically **no user interaction**.

Airflow at its Core

Webserver

- Airflows **face**.
- Provides interface for:
    - **Pause** and delete(>=1.10.1) DAGs
    - Read **logs**
    - **Analytics** on execution times and resources consumption.
- Easily **monitor** multiple tasks status.
- **Facilitates** practically all of the **user interaction.**

Airflow at its Core

Executor

- In charge of **task execution**.

- Most commonly, one of these:
    - **SequentialExecutor**: No parallelism.
    - **LocalExecutor**: Parallelized execution, all on local.
    - **CeleryExecutor**: Distributed execution on worker nodes.

Airflow at its Core

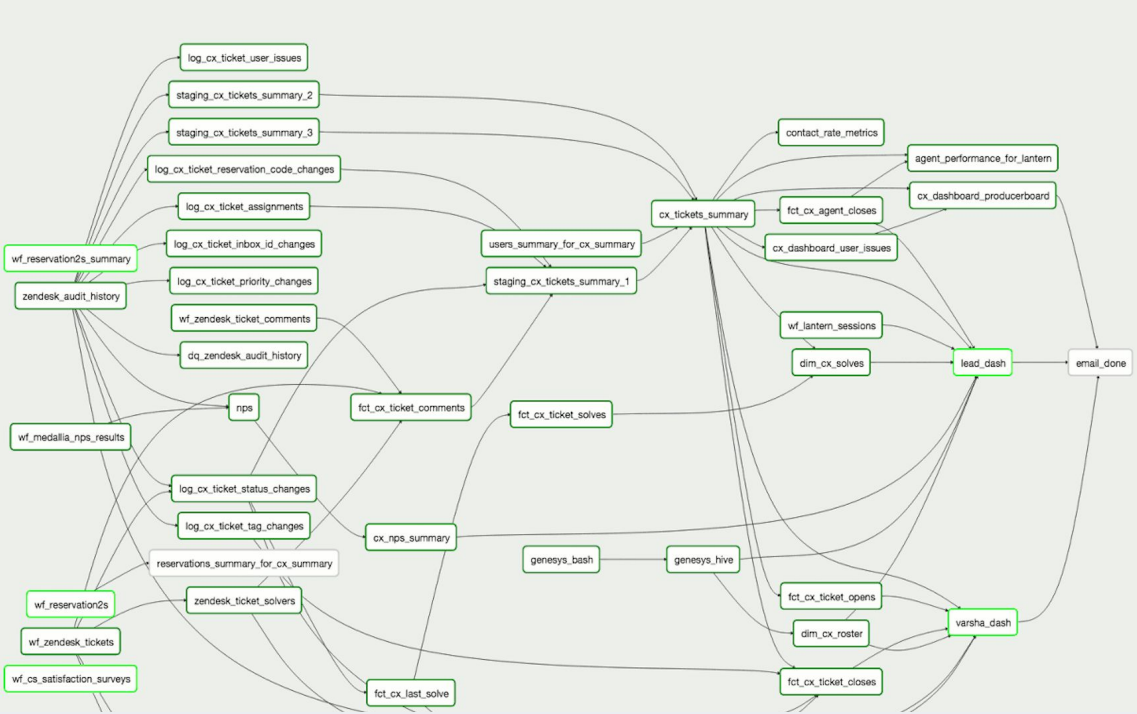
Configuration

- **Highly** customizable.
- Two ways of defining configuration:
    - **Configuration file:**
        - Ideal for portability
        - A airflow.cfg file on the AIRFLOW_HOME will be loaded upon initialization.
    - **Environment Variables**
        - Ideal for testing cases
        - All values can be set as AIRFLOW__{SECTION}__{VARIABLE}

Concepts

DAG

- **Directed Acyclic Graph**
- Each **node is a task**
- Each line is a dependency between tasks



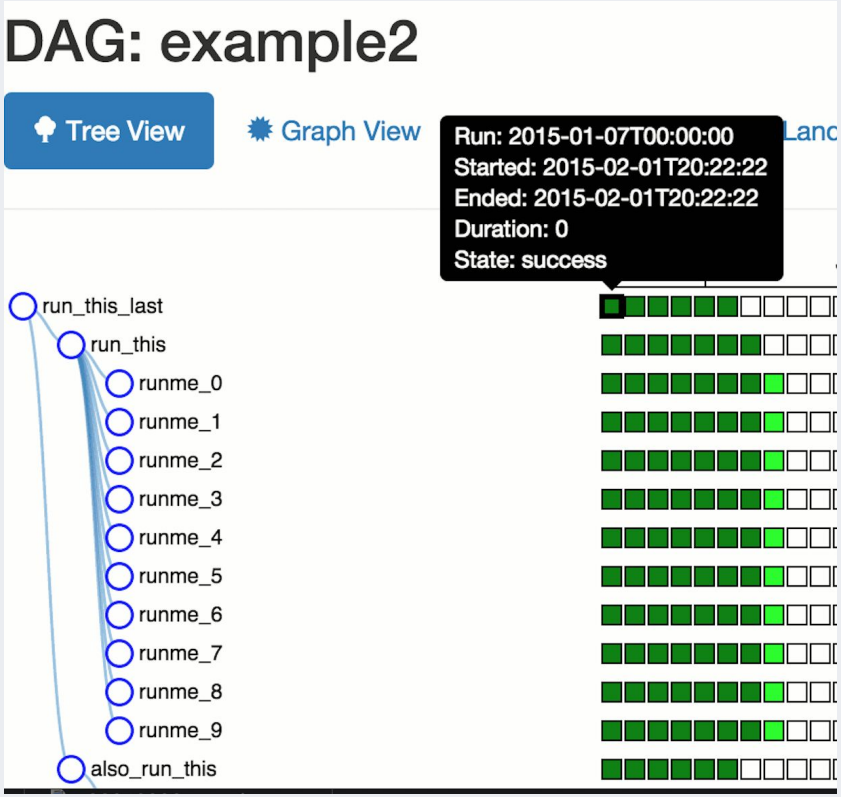
"Information flows downstream like a river[...] It has a (or several) source(s) and a (or several) tributary(ies)"

Michal Karzynski, europython 2017

Concepts

DAGRun

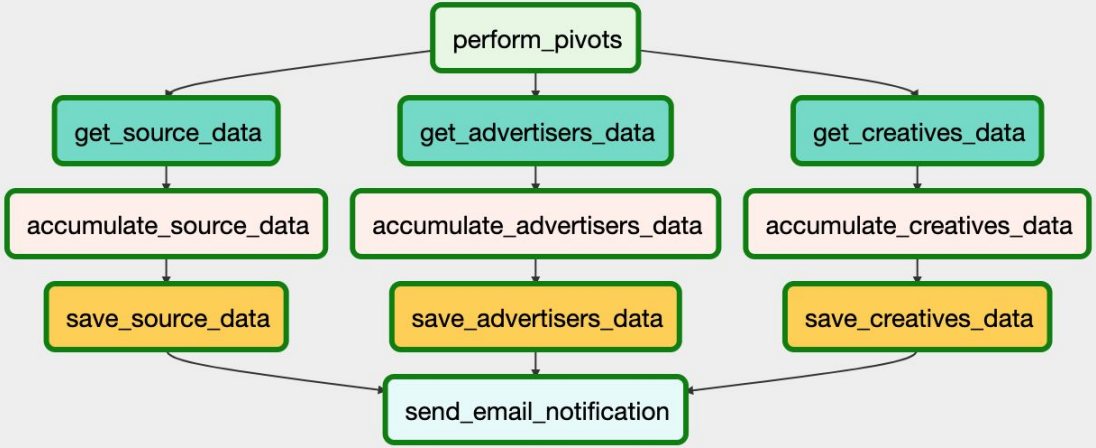
- Dag with context
- Dags are made up of tasks, DagRuns from TaskInstances
- “Down to earth” version of the DAGs



Concepts

Tasks

- Principles:
  - **Idempotency**
  - **Atomicity**
- Can be defined over:
  - **Operators**
  - **Sensors**



Operators

- Operators **execute**.
- The execute method has only one argument, that is the context

```python
import datetime as dt
import logging

from airflow.models import BaseOperator


class TodayOperator(BaseOperator):

    def execute(self, context):
        """Log the difference between now and execution_date."""
        logging.info(f'Today is {dt.datetime.today()}')
        logging.info(f'Execution date is {context["execution_date"]}')
```

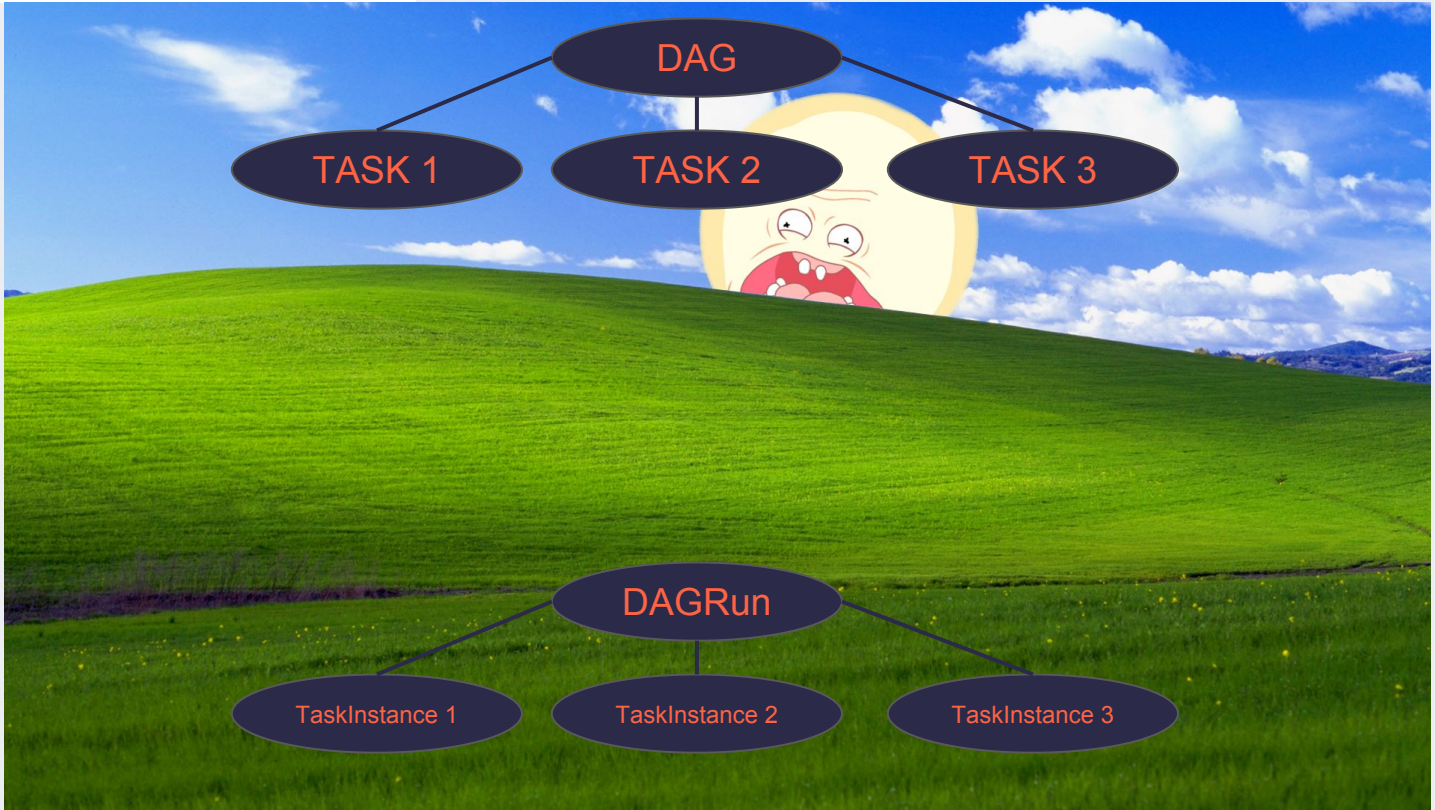
Sensors

- Sensors **poke** at something to see if it's alive.
- The poke method should **return True** when the **conditions have been met.**

```python
import datetime as dt
import logging

from airflow.sensors.base_sensor_operator import BaseSensorOperator


class PrimeMinuteSensor(BaseSensorOperator):

    PRIME_MINUTES = [
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59]

    def poke(self, context):
        minute = dt.datetime.today().minute
        logging.info(f'Minute sensed: {minute}')
        return dt.datetime.today().minute in self.PRIME_MINUTES
```

Concepts

TaskInstance



Concepts

Context

- Context turns a **Task** into a **TaskInstance**.
- Contains specific information about the DAGRun.
- Argument of both the execute and poke method.
- Virtually anything that can be added to Airflow relies on the context.

```
{
        'dag': task.dag,
        'ds': ds,
        'next_ds': next_ds,
        'next_ds_nodash': next_ds_nodash,
        'prev_ds': prev_ds,
        'prev_ds_nodash': prev_ds_nodash,
        'ds_nodash': ds_nodash,
        'ts': ts,
        'ts_nodash': ts_nodash,
        'ts_nodash_with_tz': ts_nodash_with_tz,
        'yesterday_ds': yesterday_ds,
        'yesterday_ds_nodash': yesterday_ds_nodash,
        'tomorrow_ds': tomorrow_ds,
        'tomorrow_ds_nodash': tomorrow_ds_nodash,
        'END_DATE': ds,
        'end_date': ds,
        'dag_run': dag_run,
        'run_id': run_id,
        'execution_date': self.execution_date,
        'prev_execution_date': prev_execution_date,
        'next_execution_date': next_execution_date,
        'latest_date': ds,
        'macros': macros,
        'params': params,
        'tables': tables,
        'task': task,
        'task_instance': self,
        'ti': self,
        'task_instance_key_str': ti_key_str,
        'conf': configuration,
        'test_mode': self.test_mode,
        'var': {
            'value': VariableAccessor(),
            'json': VariableJsonAccessor()
        },
        'inlets': task.inlets,
        'outlets': task.outlets,
}
```

Extras

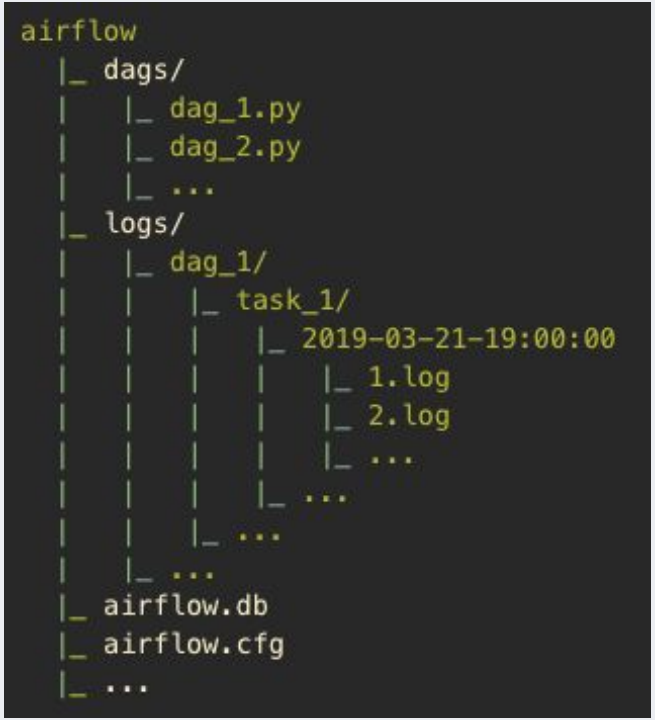
- **Variables**: Airflow provides an API to **store values on its db**, that can be pushed and pulled constantly.
- **Connections**: One can also store **sensitive data** like passwords and connections to different services, that are **safely encrypted**.
- **Hooks**: It also provides hooks that **use this connections**, that can be easily instantiated just by providing the connection id.
- **Pools**: To **limit the parallelism** on heavy tasks, one can define a pool that limits the amount of tasks of that kind to execute.

Real Time Example

Real Time Example

- We will **build a DAG** with the custom operator and sensor defined, installing Airflow from scratch and running it.

- All the code used here (and a bit more as well) is available at: https://www.github.com/gonzalezzfelipe/airflow101



Real Time Example

Dag structure and layout


```python
import datetime as dt

from airflow import DAG

from operators.today_operator import TodayOperator
from sensors.even_minute_sensor import EvenMinuteSensor

dag = DAG(
    dag_id='test_custom_sensor_and_operator',
    start_date=dt.datetime(2019, 3, 19, 13),
    schedule_interval='2 * * * *',
    catchup=True)

with dag:
    sense = EvenMinuteSensor(
        task_id='check_minute_for_even_number',
        poke_interval=60,
        timeout=60 * 60)
    execute = TodayOperator(task_id='print_now_and_execution_date')

    sense >> execute
```

Airflow at Jampp

- **Custom Automated Reporting**
  - Special clients are treated with special care, so special reports are prepared and send to them periodically through Airflow.
- **Database Population**
  - Airflow is used from the start to the end to fill the different databases used by all of the teams.
- **User segmentation ETFLs**
  - Several machine learning models are applied to target users intelligently, all through Airflow.

Tips for a better experience

- **Always use connections**. This allows for a better portability and smoother transition when deploying new DAGs.
- Don’t be afraid to **write custom operators and sensors**. Allows for a better readability and monitoring.
- **Be aware of what you store in Variables**. Using the Variables is easy and tempting, but it shouldn’t be treated as a database.
- **Don’t use SubDAGs!** This is a feature that has brought tons of problems to uncountable companies (including us).

Thank you!

Links

- Building better data pipelines with Airflow, Sid Anand.
- Airflow: Lesser known tips, tricks and best practices, Kaxil Naik.
- Get started developing workflows with Apache-Airflow, Michal Karzynski.
- Developing elegant workflows in Python code with Apache Airflow, Michal Karzynksi.
- Airflow 101: Start automating your data pipelines with Airflow, Sriram Baskaran.
- Code from examples