Workshop:

Contributing to Apache Airflow

Outreachy 2021

About us

Jarek Potiuk

Independent Open-Source Contributor and Advisor
Airflow Committer & PMC member
Twitter: @jarekpotiuk

Elad Kalif

Data Engineer at Wix.com
Airflow Committer & PMC member
Twitter: @eladkal

Nasser Kaze

Software Engineer at Buea
Apache Fineract Committer
Google Summer of Code 2021 intern with The Apache Software
Twitter: @xurror

Agenda

- Introduction
- Part 1: Prepare environment & choose issue
- Part 2: Coding!
- Part 3: Live code reviews

Intro

Apache Software Foundation

- The World's Largest Open Source Foundation
- Established in 1999, the ASF is a US 501(c)(3) charitable organization
- Funded by individual donations and corporate sponsors
- Our all-volunteer board oversees more than 350 leading Open Source projects,
  - including Apache HTTP Server, Apache Spark, Apache Airflow …

The Apache Way

Methodology followed to ensure collaborative environment across the projects

- Community over code - the main motto
- Community of peers - no-one is “boss”
- Merit - recognizing your work
- Independence - vendor neutrality
- Open Communication - transparency everywhere
- Decision Making - consensus & votes

Airflow is Orchestrator

APACHE Spark

docker

kubernetes

Google Cloud

amazon web services

Azure

Apache Airflow

- Pure python, workflow management tool
- Define workflows as .py files
- Processing data intervals
- Schedule jobs by cron format and datetime
- Define relations between tasks
- Works locally and in the cloud
- Debug -> Deploy -> Scale

Luigi

argo

OOZIE

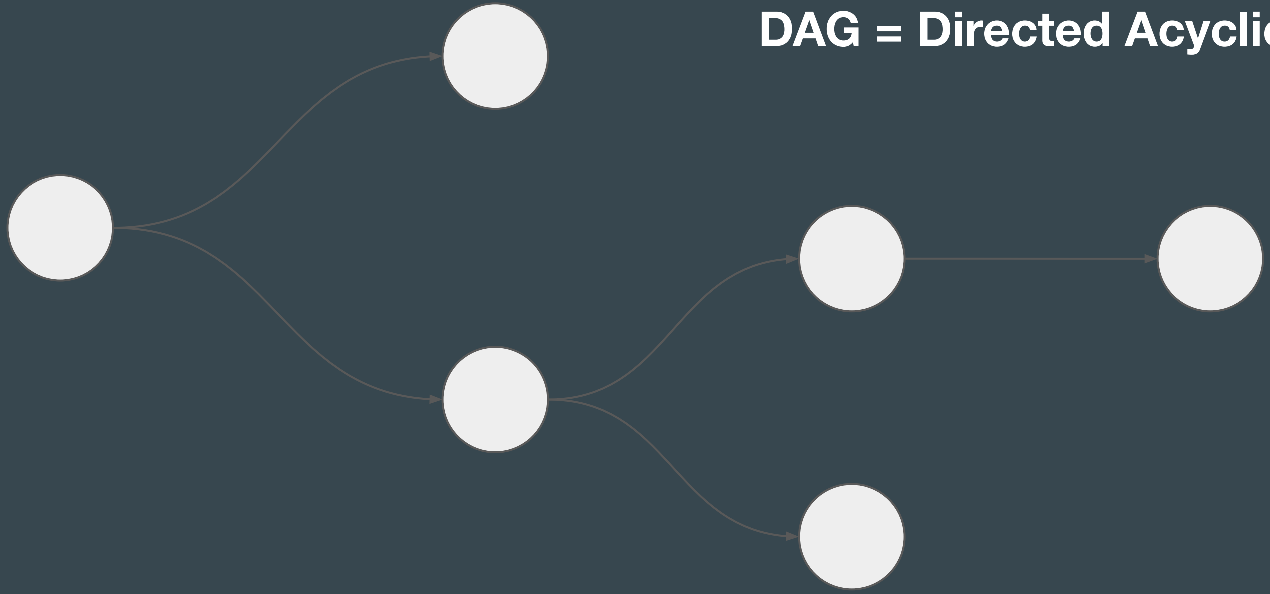
Azkaban
Open-source Workflow Manager

History

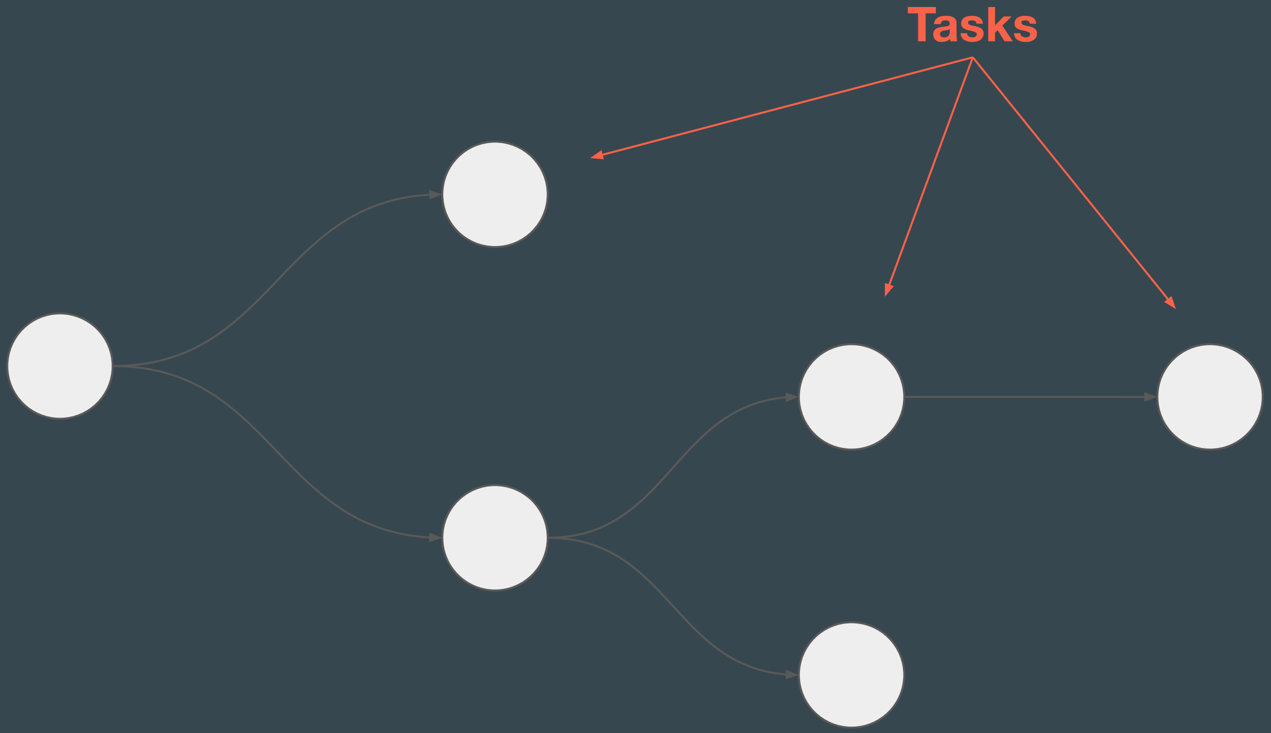
- Originally developed in AirBNB
- Incubating in Apache Software Foundation since 2017
- Official ASF project (TLP) since January 2019
- Airflow 2 - December 2020
- One of the most popular orchestrators out there
- As of September 2021 - ASF project with highest number of contributors (>1700)
- Airflow Summit 2021 - >10.000 attendees

Airflow Basics

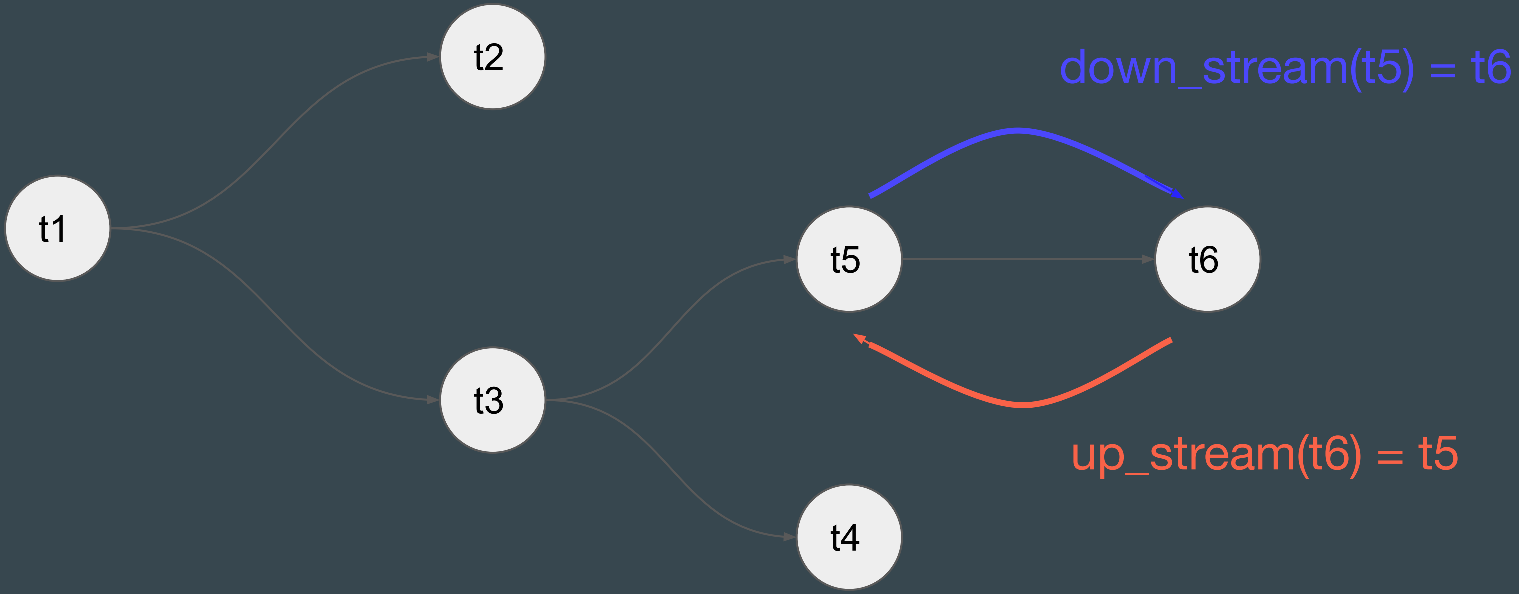
DAG = Directed Acyclic Graph



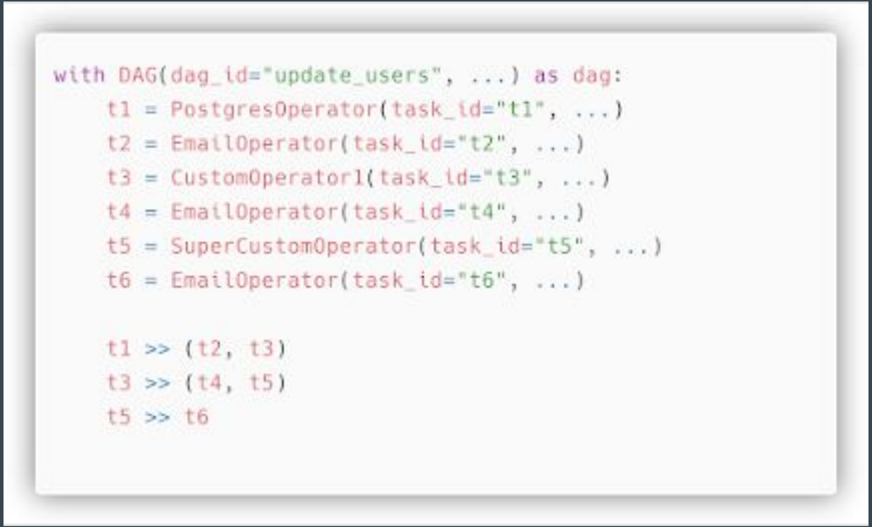
DAG: Tasks



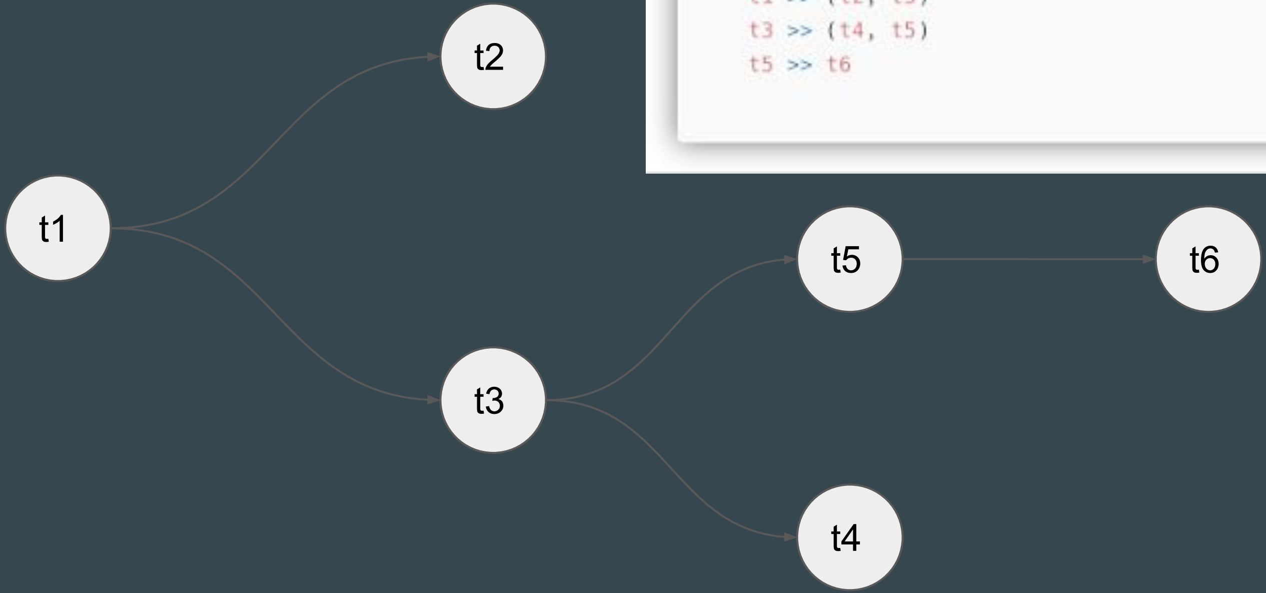
DAG: relations



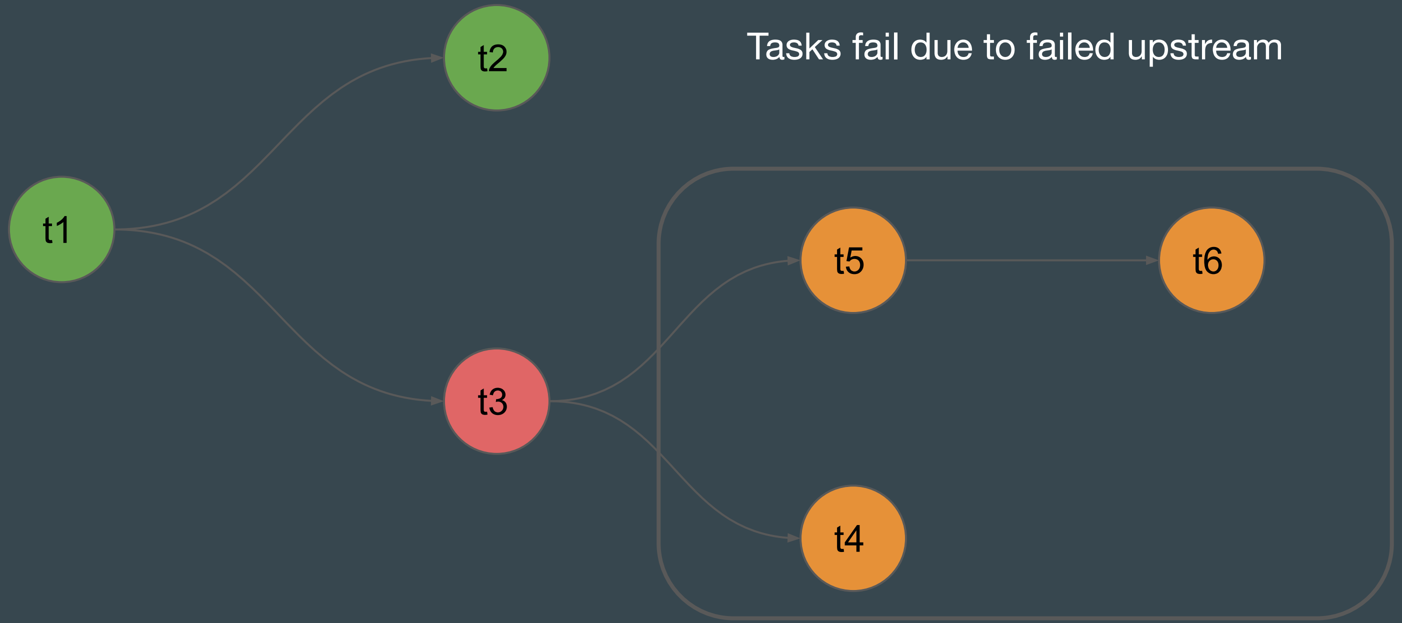
DAG: relations



```
with DAG(dag_id="update_users", ...) as dag:
    t1 = PostgresOperator(task_id="t1", ...)
    t2 = EmailOperator(task_id="t2", ...)
    t3 = CustomOperator1(task_id="t3", ...)
    t4 = EmailOperator(task_id="t4", ...)
    t5 = SuperCustomOperator(task_id="t5", ...)
    t6 = EmailOperator(task_id="t6", ...)

    t1 >> (t2, t3)
    t3 >> (t4, t5)
    t5 >> t6
```



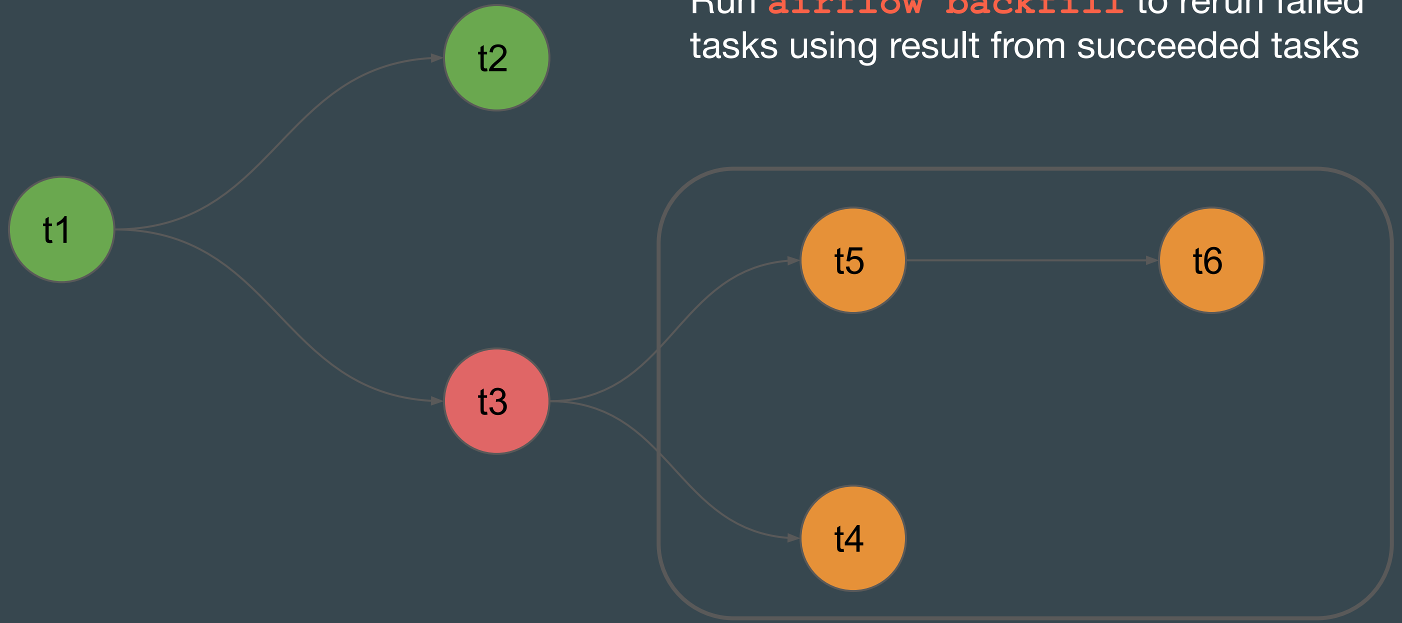
DAG: relations



Tasks fail due to failed upstream

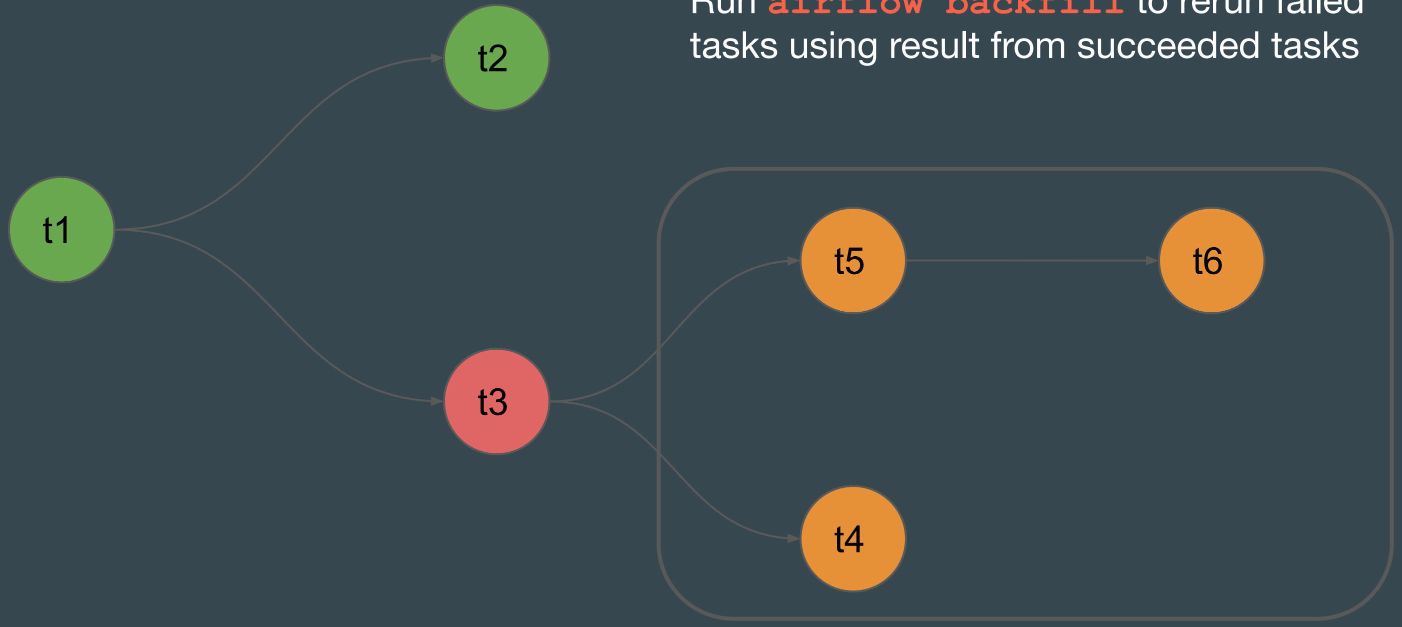
DAG: backfill

Run `airflow backfill` to rerun failed tasks using result from succeeded tasks



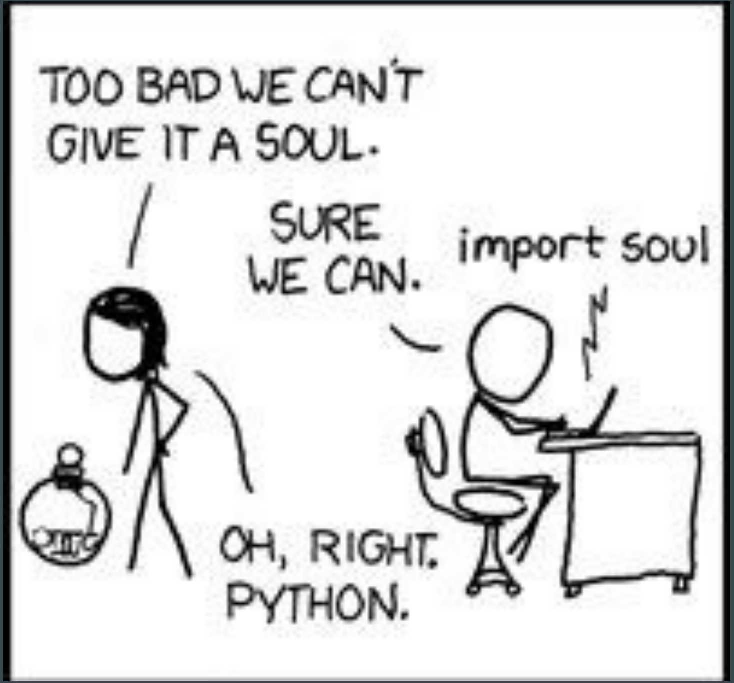
DAG: backfill

Tasks have to be `idempotent`



Run `airflow backfill` to rerun failed tasks using result from succeeded tasks

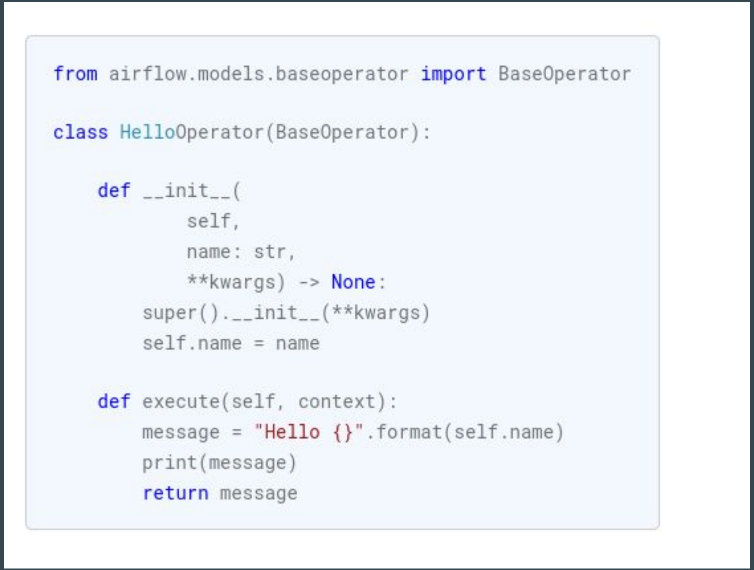
Tasks


TOO BAD WE CAN'T GIVE IT A SOUL.
SURE WE CAN.
import soul
OH, RIGHT. PYTHON.

Tasks

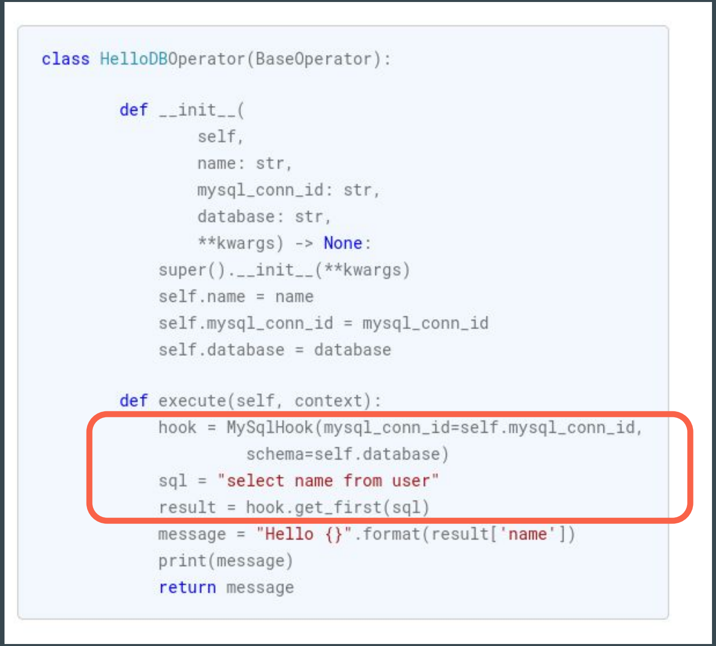
- Tasks types
  - Operators, Sensors, Transfers
- Specialized operators in Providers (70+)
- General Purpose Operators
  - Bash
  - Python
  - Python Virtualenv
  - Docker
  - Kubernetes Pod
- “Functional” dags/tasks via decorators

Operators

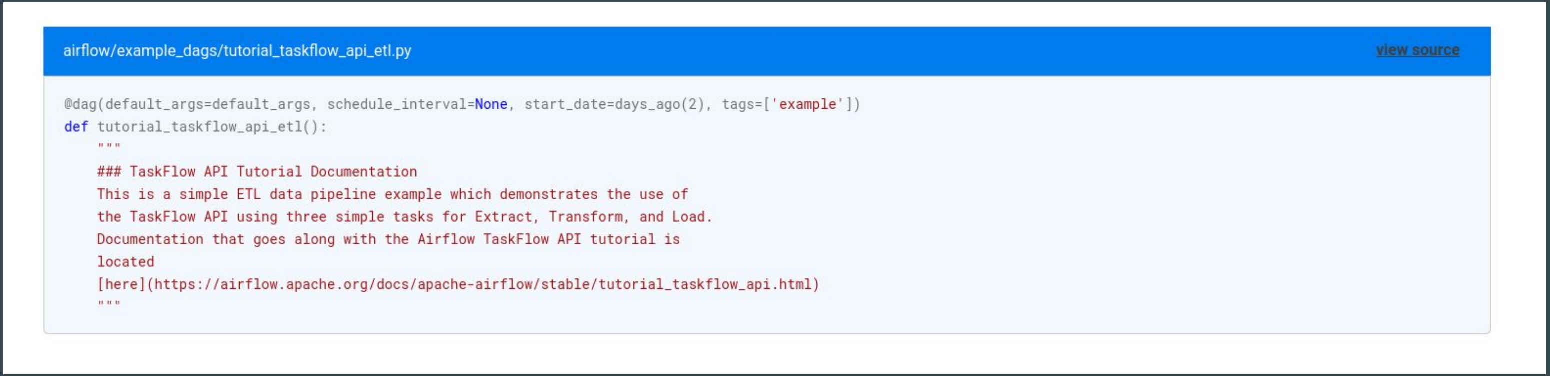


```python
from airflow.models.baseoperator import BaseOperator

class HelloOperator(BaseOperator):

    def __init__(
            self,
            name: str,
            **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = "Hello {}".format(self.name)
        print(message)
        return message
```

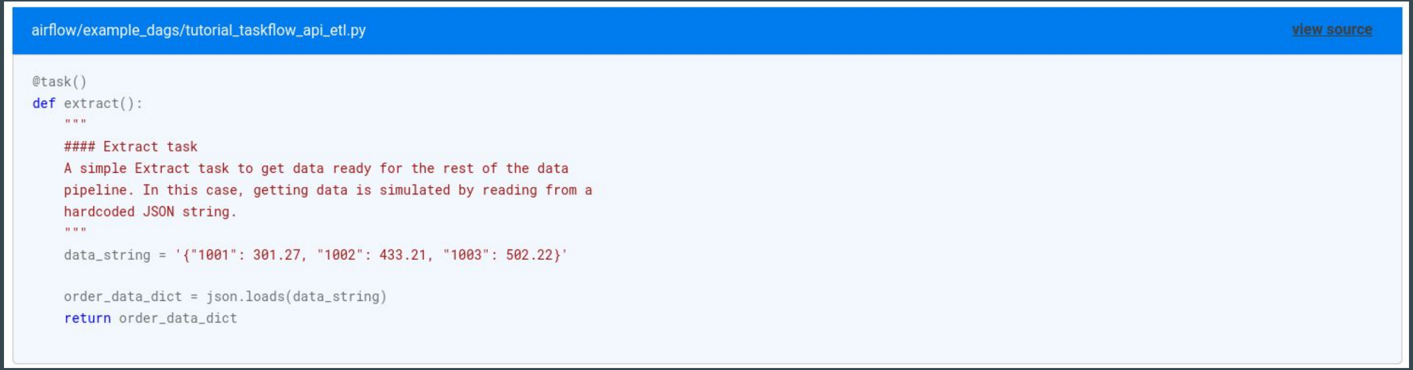
Hooks


```
class HelloDBOperator(BaseOperator):

        def __init__(
                self,
                name: str,
                mysql_conn_id: str,
                database: str,
                **kwargs) -> None:
            super().__init__(**kwargs)
            self.name = name
            self.mysql_conn_id = mysql_conn_id
            self.database = database

        def execute(self, context):
            hook = MySqlHook(mysql_conn_id=self.mysql_conn_id,
                     schema=self.database)
            sql = "select name from user"
            result = hook.get_first(sql)
            message = "Hello {}".format(result['name'])
            print(message)
            return message
```

TaskFlow - “functional” DAG/Task definition


airflow/example_dags/tutorial_taskflow_api_etl.py view source

```
@dag(default_args=default_args, schedule_interval=None, start_date=days_ago(2), tags=['example'])
def tutorial_taskflow_api_etl():
    """
    ### TaskFlow API Tutorial Documentation
    This is a simple ETL data pipeline example which demonstrates the use of
    the TaskFlow API using three simple tasks for Extract, Transform, and Load.
    Documentation that goes along with the Airflow TaskFlow API tutorial is
    located
    [here](https://airflow.apache.org/docs/apache-airflow/stable/tutorial_taskflow_api.html)
    """
```

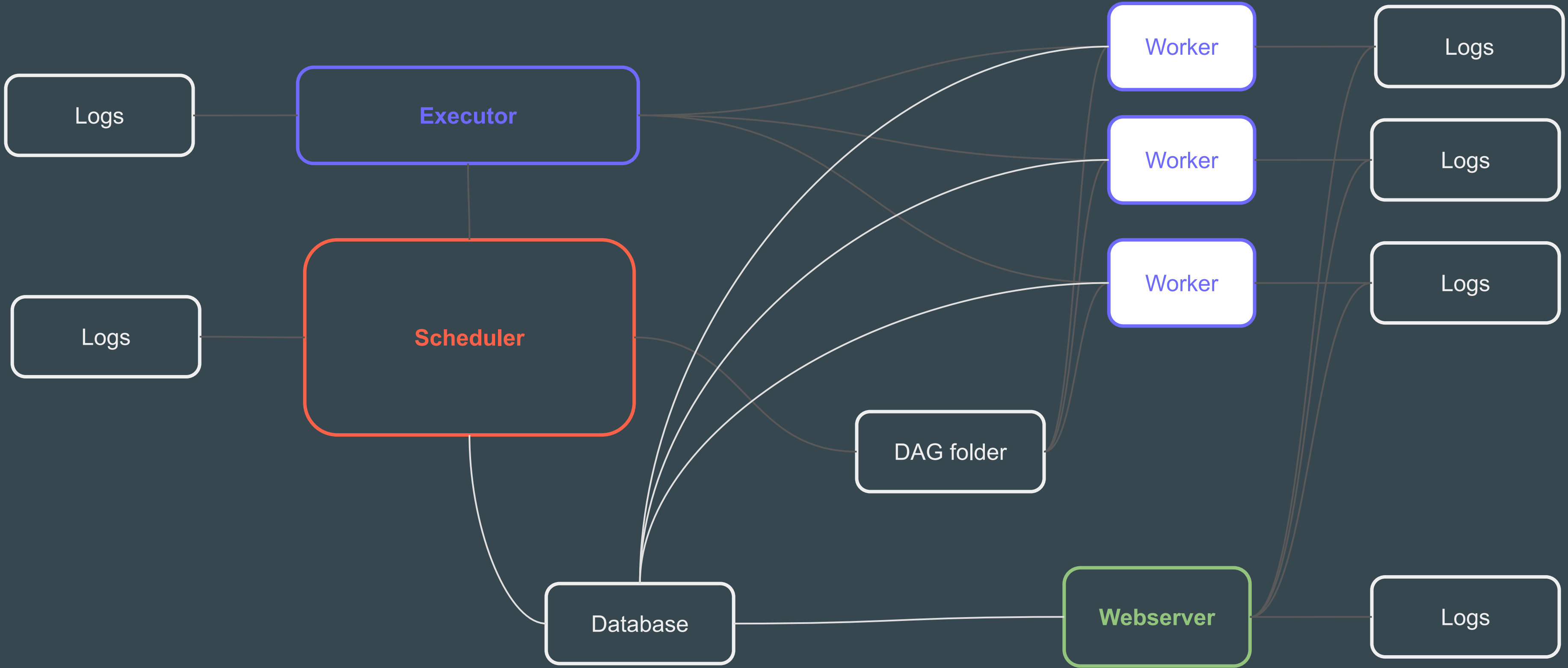

airflow/example_dags/tutorial_taskflow_api_etl.py view source

```
@task()
def extract():
    """
    #### Extract task
    A simple Extract task to get data ready for the rest of the data
    pipeline. In this case, getting data is simulated by reading from a
    hardcoded JSON string.
    """
    data_string = '{"1001": 301.27, "1002": 433.21, "1003": 502.22}'

    order_data_dict = json.loads(data_string)
    return order_data_dict
```

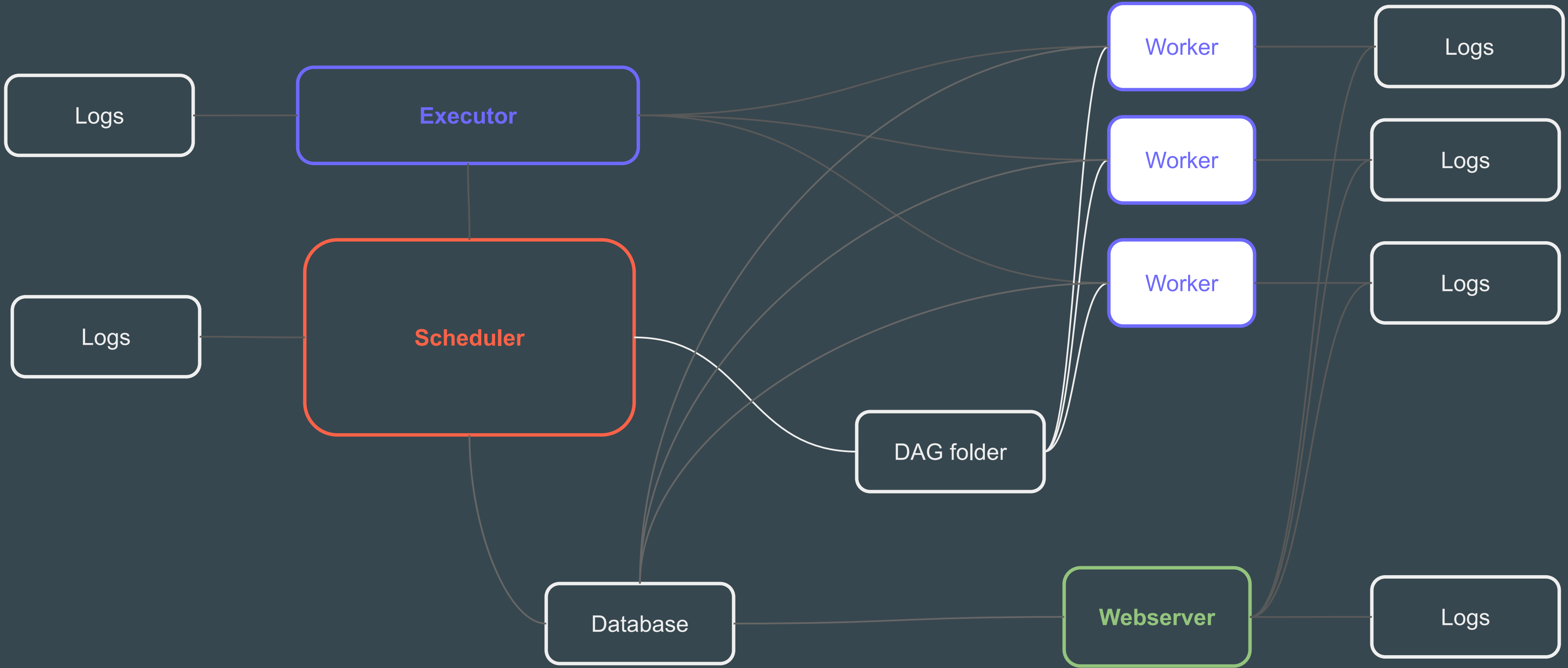
Other components

- XCom - cross task communication
- Production-level Executors: Local, Celery, Kubernetes, CeleryKubernetes
- Development-level Executors: Sequential, Debug
- Scheduler:
    - Continuous DAG parsing
    - Scheduling DAGRuns, and sending tasks to executors

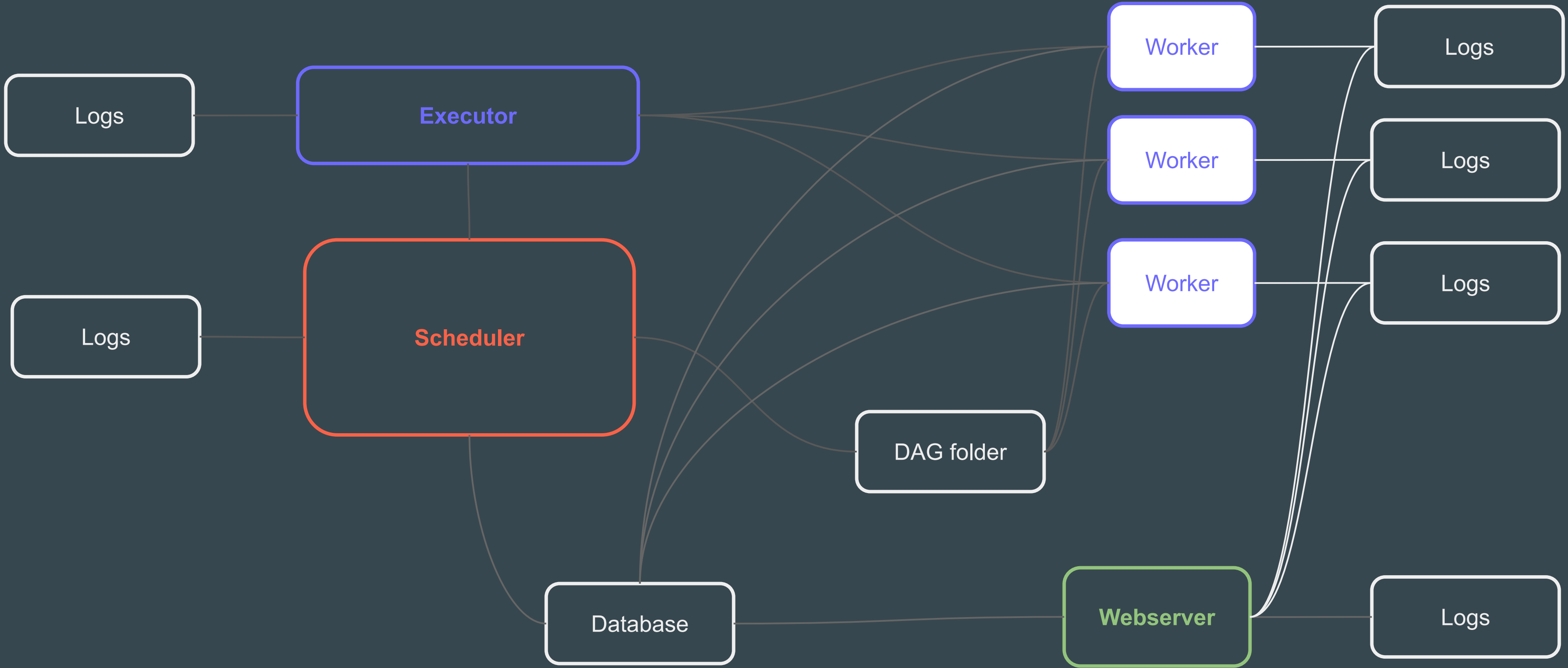
Airflow's distributed Architecture: Database



Airflow's distributed Architecture: DAG folder



Airflow's distributed Architecture: Logs



Workshop Communication & Rules

- Slack channel

  #outreachy
- We can break-out to sub-group as needed
- Anyone can share their screen if needed
- Ask questions any time

Workflow to follow

1. **Fork airflow/main**

   Make your own fork of Apache Airflow main repo

2. **Configure environment**

   Create virtualenv
   Initialize Breeze
   Install pre-commit

3. **Connect with people**

   Join devlist
   Setup slack account

4. **Prepare PR**

   PR from your fork
   Follow PR guidelines in CONTRIBUTING.rst

5. **PR review**

   Ping @ #development slack
   Comment @people
   Be annoying
   Be considerate

Part 1:

- Setup Dev environment
- Choose issue to work on

Setting up

- Setup development environment
  - git
  - Breeze, local virtualenv
  - pre-commit (!)
  - IDE setup
- Choose Issue to work on (small)
  - https://github.com/apache/airflow/contribute
  - https://github.com/apache/airflow/labels/contributors-workshop
- (With our help) locate where to make changes

Part 2: Coding!

Project structure

```
airflow/
 |- airflow/
 |- - . . .
 |- - executors/
 |- - hooks/
 |- - operators/
 |- - providers/
 |- - www/
 |- docs/
 |- tests
```

```
airflow/
 |- tests/
 |- - . . .
 |- - executors/
 |- - hooks/
 |- - operators/
 |- - providers/
 |- - www/
```

Writing code

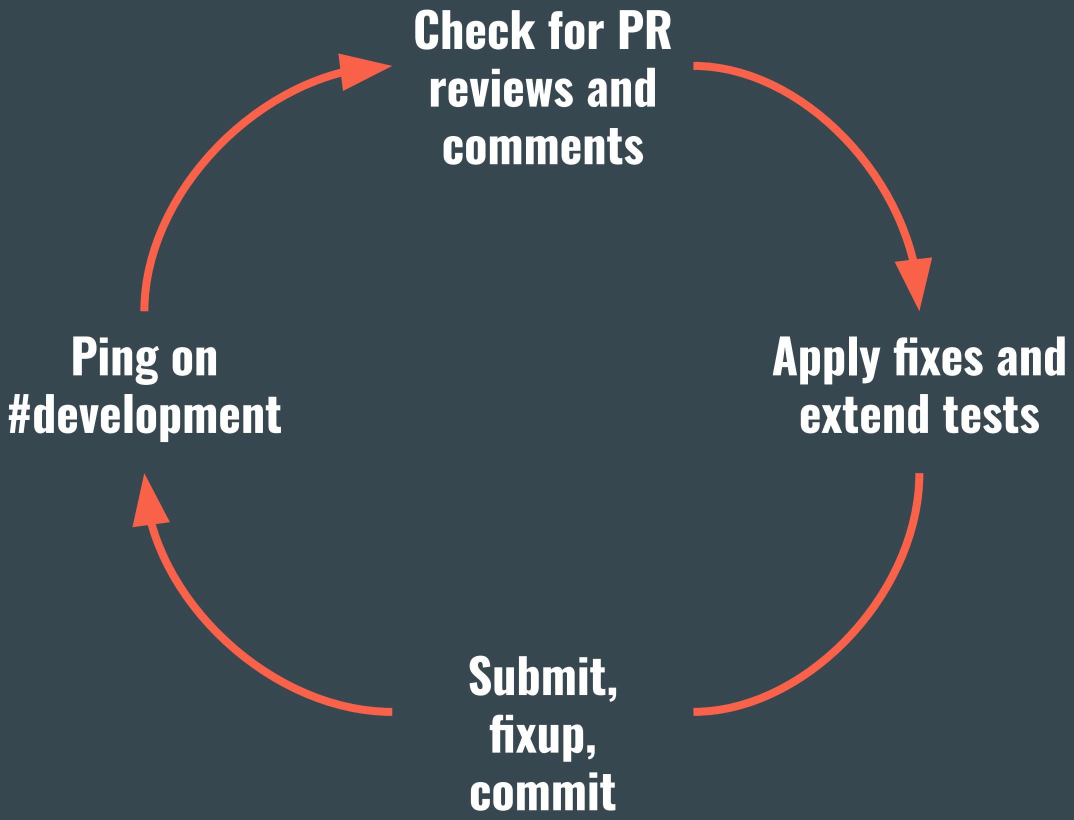
- Write code / docs
- Install pre-commit
  - `pre-commit install`
- Run tests build docs
  - `pytest -s tests/models/test_dagrun.py`
  - `./breeze build-docs`
  - `./breeze build-docs -- --package-filter apache-airflow`

Part 3: Review

Review etiquette

- Remember about diversity & inclusion
- Be empathetic
- Do not be afraid to ask, argue (with code not people) or suggest - **we all learn** from each other!
- Rebase when you are asked to do it

Review process


Check for PR reviews and comments
Apply fixes and extend tests
Submit, fixup, commit
Ping on #development