REPLY DATA

Apache Airflow in the Cloud

Programmatically orchestrating workloads with Python

Basic instructions

- **Join the chat room:** https://tlk.io/pydata-london-airflow (All necessary links are in this chat room)
  - **Please fill the google form - (For those who haven't filled the form prior to this tutorial) - This is to add you in our Google Cloud Platform project**
  - **Download the JSON file for Google Cloud access**
  - **Clone the Github repo for this tutorial**
  - **Follow the link for airflow installation**
  - **Link to this slide deck**

- **Clone the repository**
  - **Github link:** https://github.com/DataReplyUK/airflow-workshop-pydata-london-2018

Agenda

- Who we are and why we are here ?
- Different workflow management system
- What is Apache Airflow ?
- Basic Concept and UI Walkthrough

- Tutorial 1: Basic workflow
- Tutorial 2: Dynamic workflow
- GCP introduction
- Tutorial 3: Workflow in GCP

Who we are!

Hello!

I am *Kaxil Naik*

Data Engineer at Data Reply and an Airflow contributor

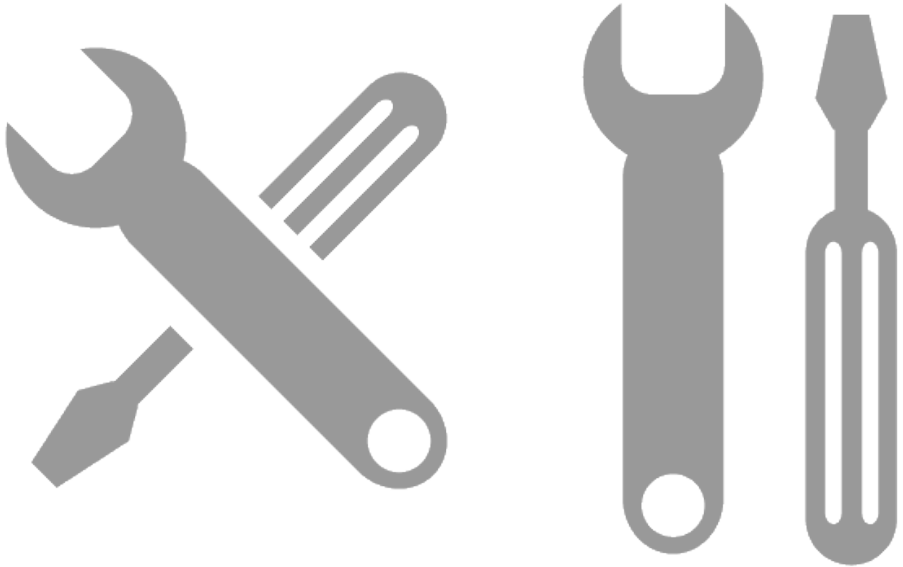
Hi!

I am *Satyasheel*

Data Engineer at Data Reply

First of all Some Truth

Data is weird and it breaks stuffs



Robust pipelines need robust workflow

- Cron (Seriously!)
- Oozie
- Luigi
- Apache Airflow

Image Source: https://www.free-vectors.com/images/Objects/263-hand-wrench-tool---spanner-vector.png

In the beginning, there was Cron.

We had one job, it ran at 1AM, and it was good.

- Pete Owlett, PyData London 2016
  from the outline of his talk:
  “Lessons from 6 months of using Luigi in production”

https://visitkokomo.files.wordpress.com/2010/10/elwood-first-car.jpg

In the beginning, there was ~~Cron.~~ CHAOS

We had ~~one~~ 100 job, it ran at ~~1AM~~ DEPENDS, and it was ~~good.~~

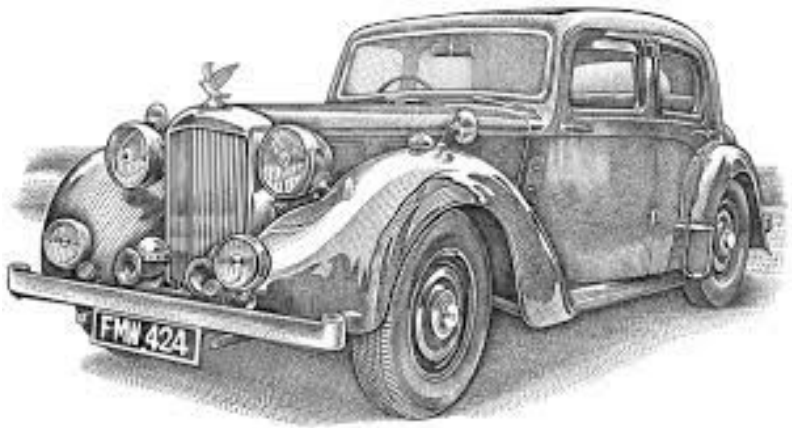
Pete Owlett, PyData London 2016
from the outline of his talk:
“Lessons from 6 months of using Luigi in production”

https://visitkokomo.files.wordpress.com/2010/10/elwood-first-car.jpg

Oozie

Pros:

- Used by thousands of companies
- Web api, Java api, cli and html support
- Oldest among all



Cons:

- XML
- Significant effort in managing
- Difficult to customize

https://i.pinimg.com/originals/31/f9/bc/31f9bc4099f1ab2c553f29b8d95274c7.jpg

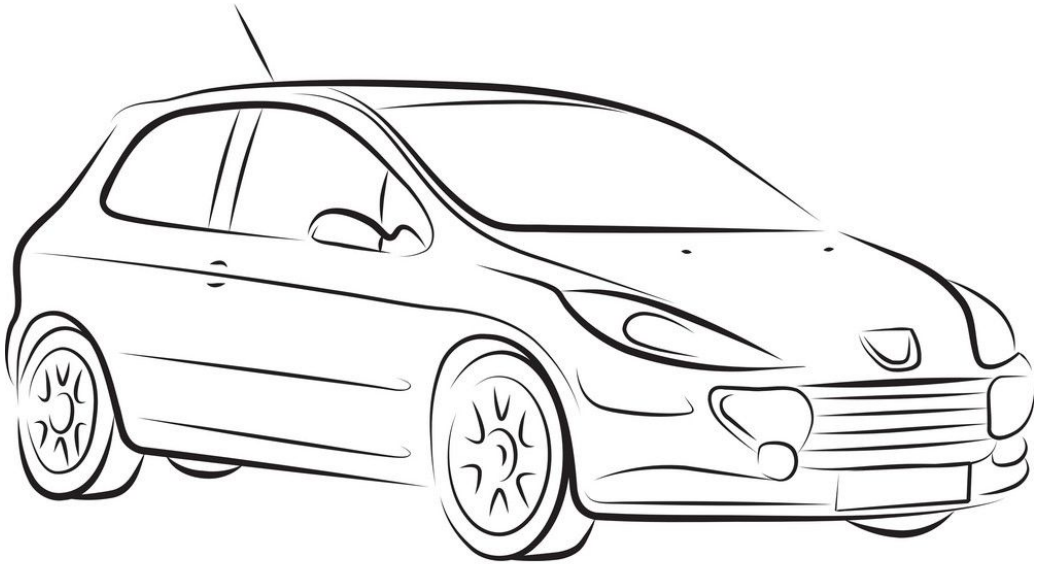
Luigi

Pros:

- Pythonic way to write a DAG
- Pretty Stable
- Huge Community
- Came from Spotify engineering team

Cons:

- Have to schedule workflows externally
- The open source Luigi UI is hard to use
- No inbuilt monitoring, alerting



https://cdn.vectorstock.com/i/1000x1000/88/00/car-sketch-vector-98800.jpg

Airflow

Pros:

- Python Code Base
- Active community
- Trigger rules
- Cool web UI and rich CLI
- Queues & Pools
- Zombie Cleanup
- Easily extendable

Cons:

- No role based access control
- Minor issues (Deleting DAGs is not straight forward)
- Umm!!!

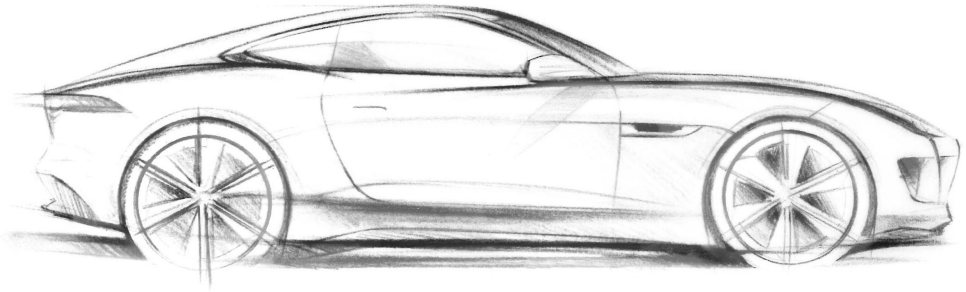


Image Source: https://s3.envato.com/files/133677334/Preview_Image_Set/GRID_06.jpg

Airflow

Pros:

- Python Code Base
- Active community
- Trigger rules
- Cool web UI and rich CLI
- Queues & Pools
- Zombie Cleanup
- Easily extendable

Cons:

- ~~No role based access control~~
- Minor issues (Deleting DAGs is not straight forward)
- Umm!!!

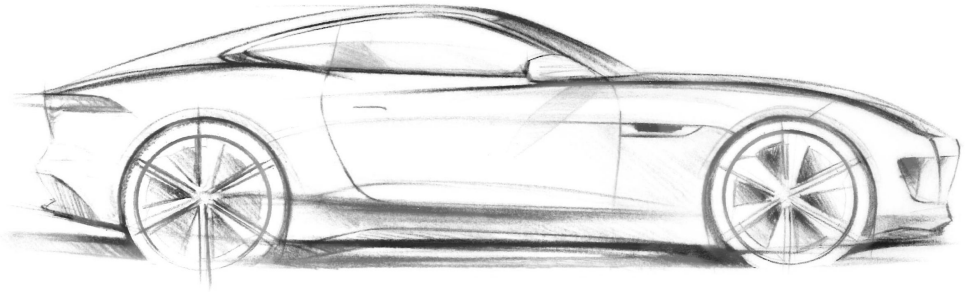
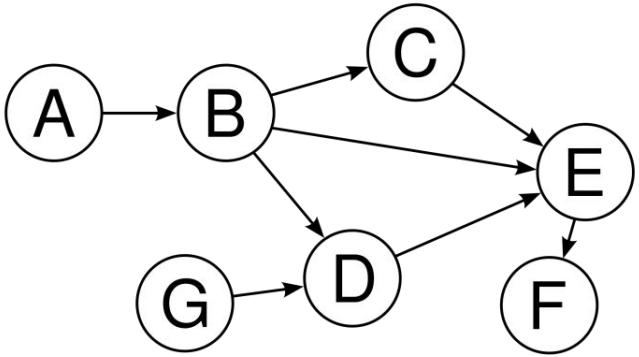


Image Source: https://s3.envato.com/files/133677334/Preview_Image_Set/GRID_06.jpg

http://www.belairdaily.com/wp-content/uploads/2018/04/Cable-Management-Market-1.jpg

What is Airflow ?

Airflow is a platform to programatically author, schedule and monitor workflows (a.k.a DAGs)



What is Airflow ?

- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures (Email, Slack)
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

- Open Source ETL workflow management tool written purely in python
- It's the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook's dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

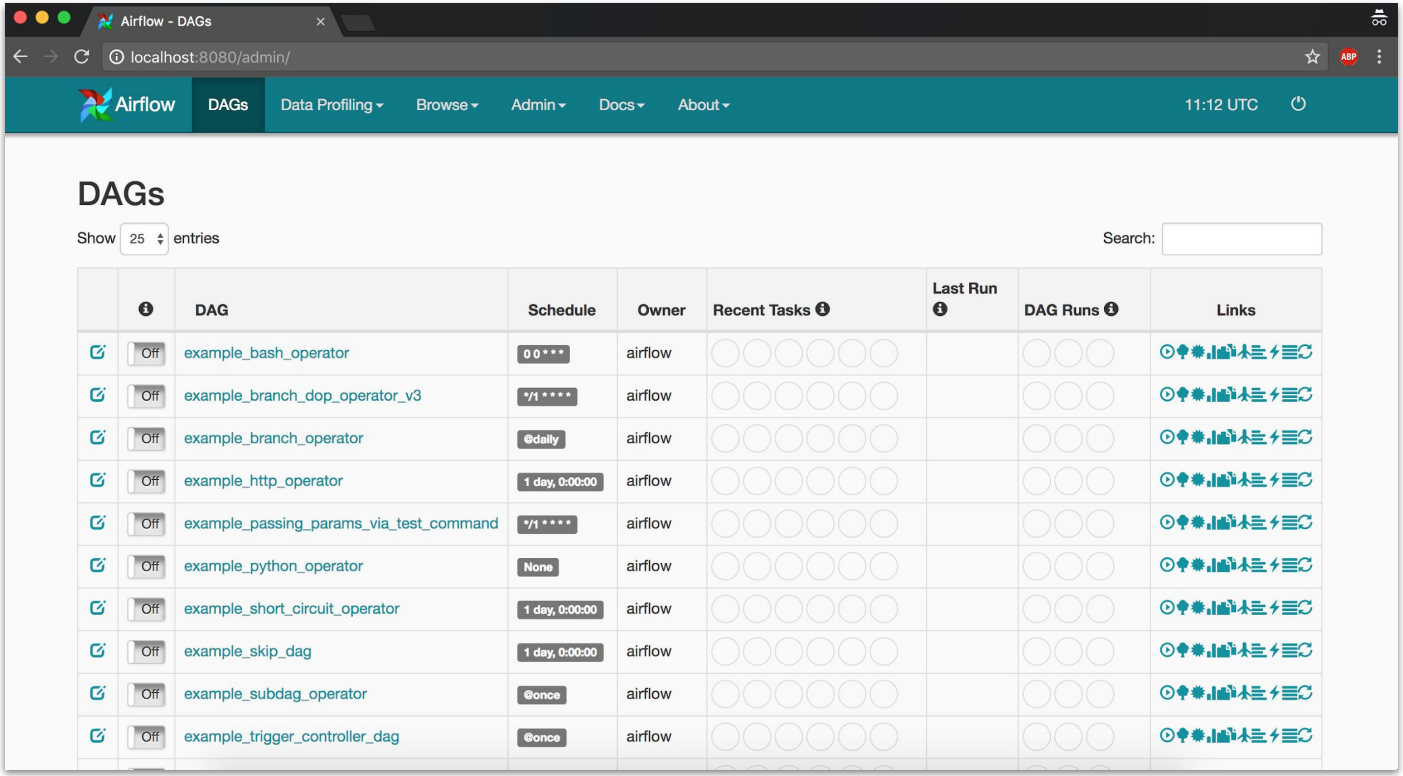
- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is a production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

What is Airflow ?

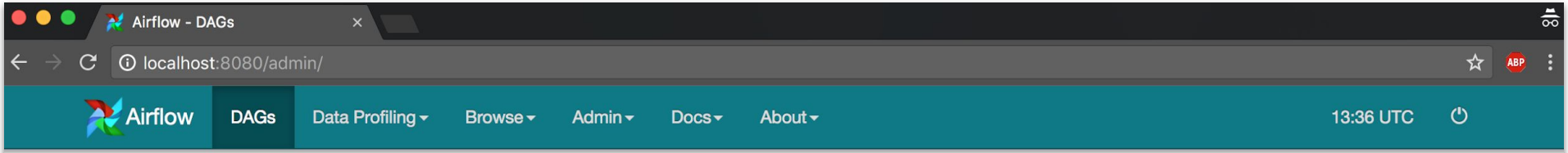
- Open Source ETL workflow management tool written purely in python
- It’s the glue that binds your data ecosystem together
- It can handle failures
- Alert on failures
- Monitor performance of tasks over time
- Scale!
- Developed by Airbnb
- Inspired by Facebook’s dataswarm
- It is production ready
- It Ships with:
  - DAG scheduler
  - Web Application UI
  - Powerful CLI

Airflow Web UI

Airflow Web UI



First look of Airflow WebUI right after installation



DAGs

Similar to Cron

Show 25 entries

Search:

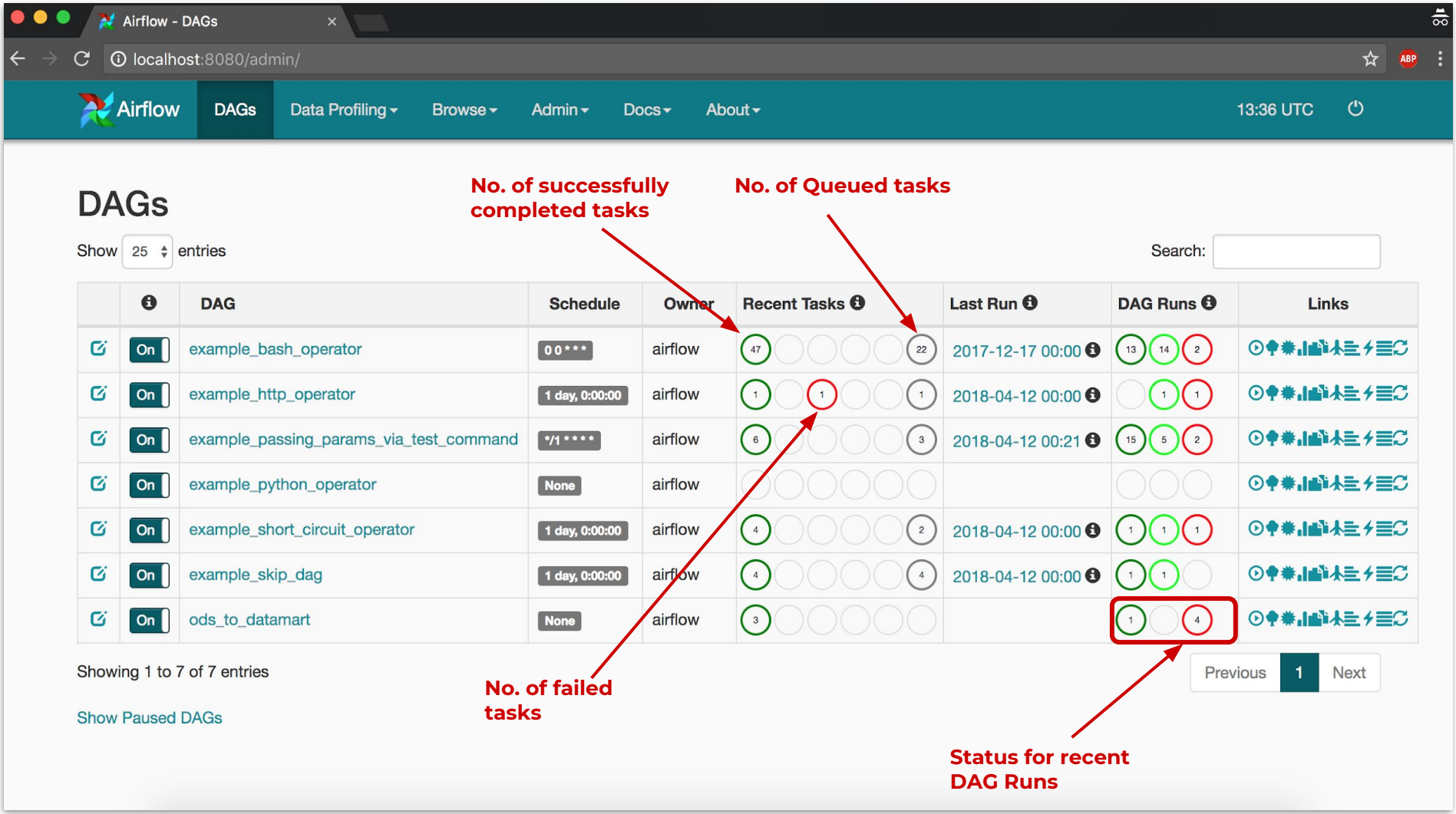
		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	example_bash_operator	0 0 * * *	airflow	47 22	2017-12-17 00:00	13 14 2	
	On	example_http_operator	1 day, 0:00:00	airflow	1 1 1	2018-04-12 00:00	1 1	
	On	example_passing_params_via_test_command	*/1 * * * *	airflow	6 3	2018-04-12 00:21	15 5 2	
	On	example_python_operator	None	airflow				
	On	example_short_circuit_operator	1 day, 0:00:00	airflow	4 2	2018-04-12 00:00	1 1 1	
	On	example_skip_dag	1 day, 0:00:00	airflow	4 4	2018-04-12 00:00	1 1	
	On	ods_to_datamart	None	airflow	3		1 4	

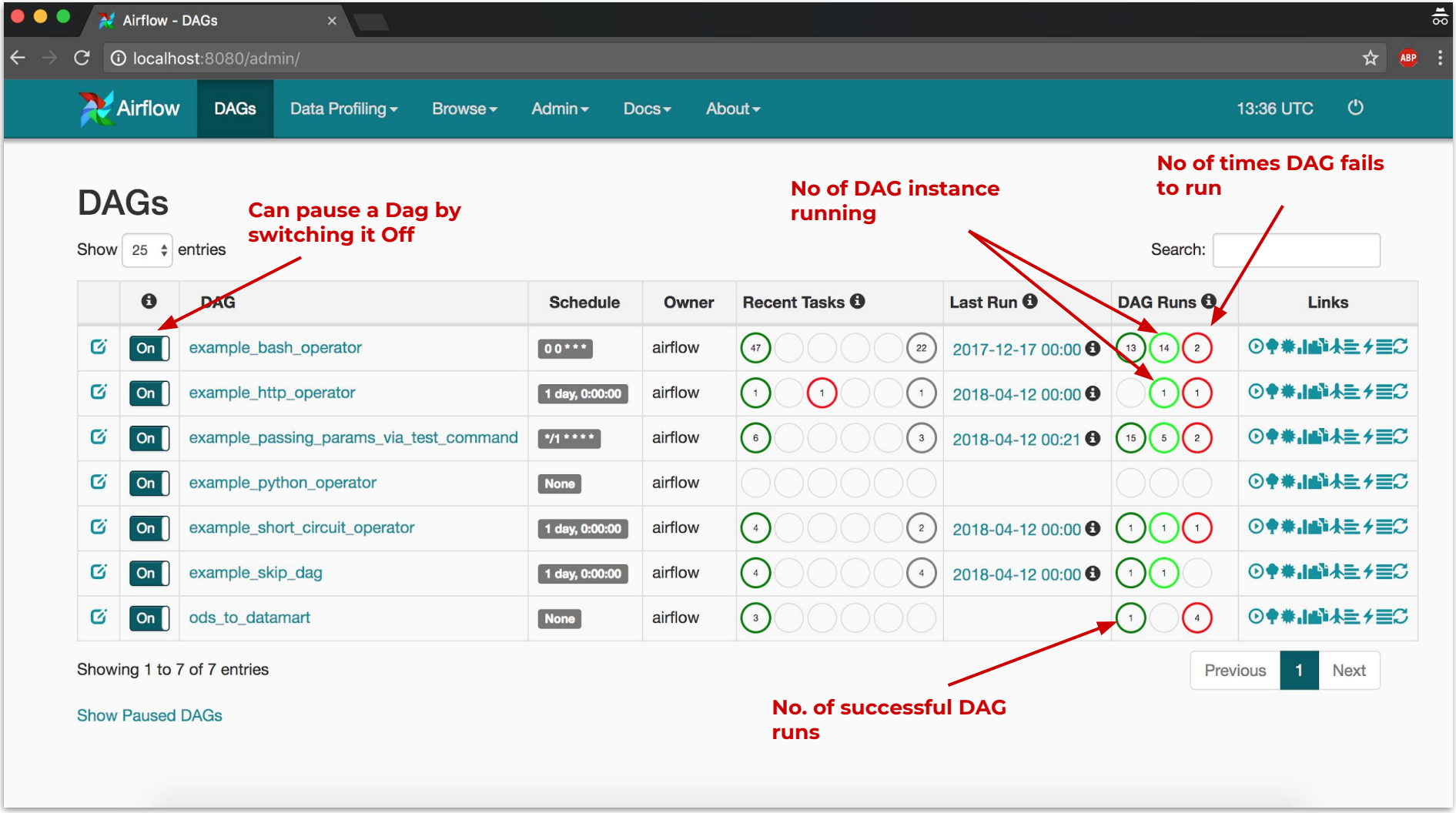
Showing 1 to 7 of 7 entries

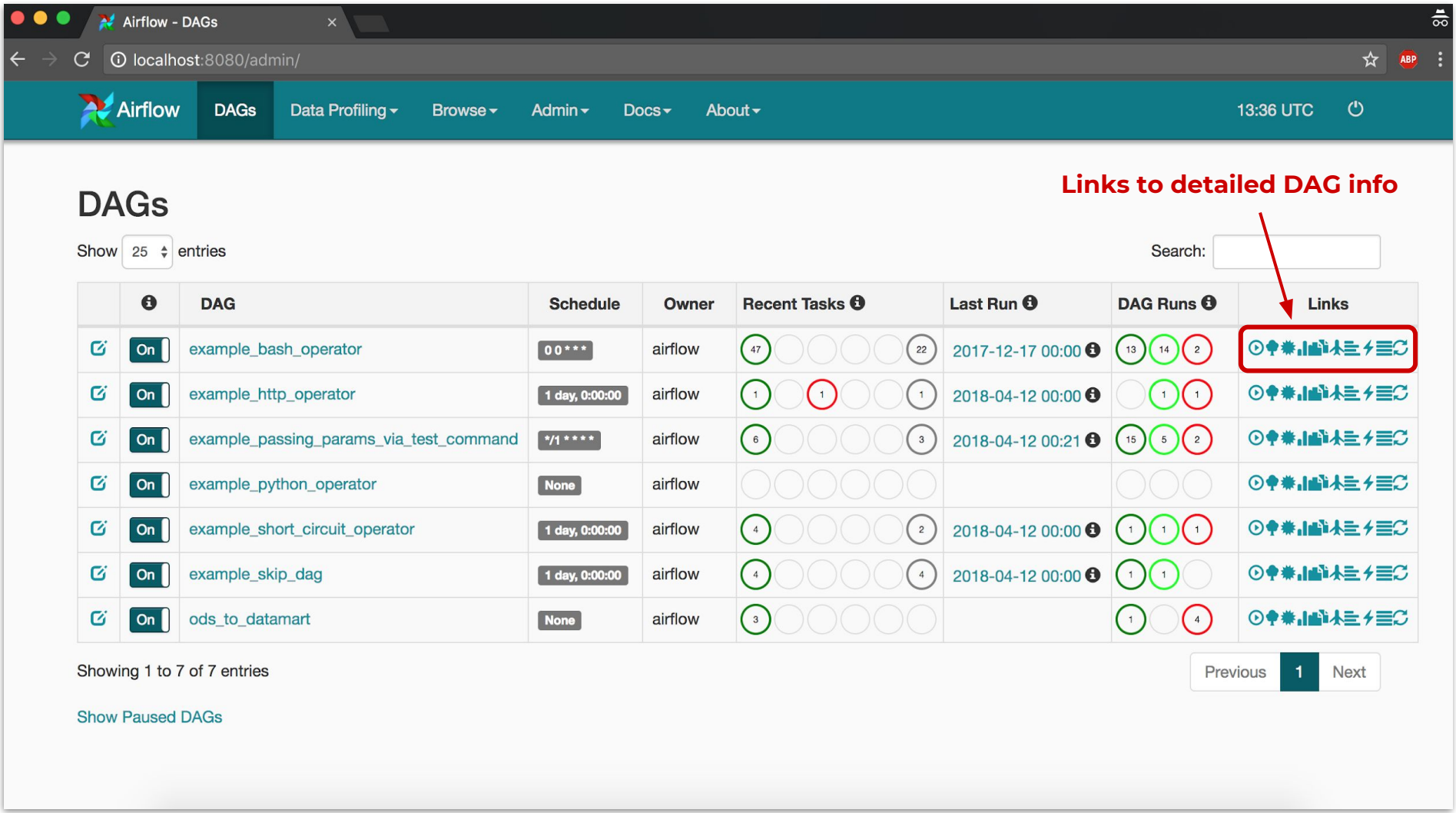
Previous 1 Next

Show Paused DAGs

This is how it looks once you start running your DAGs ....

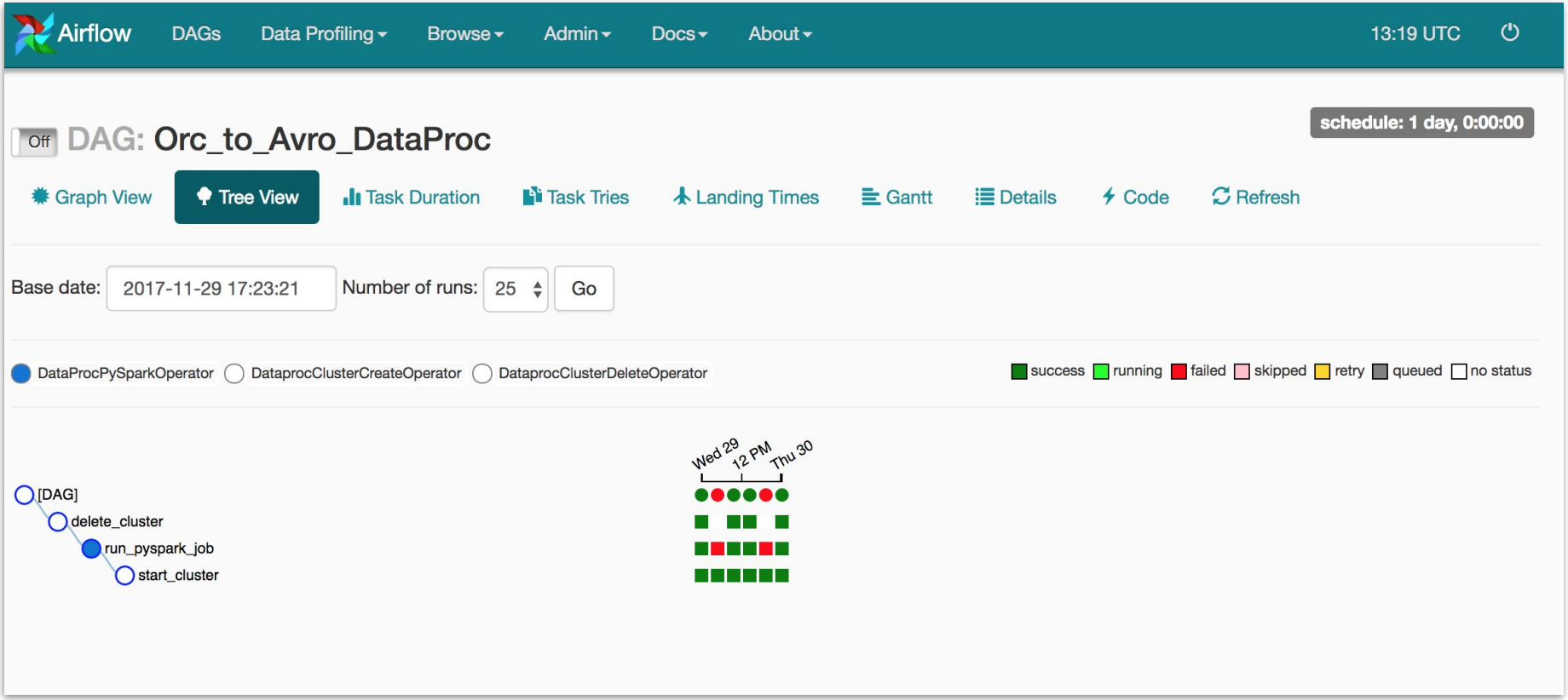

Airflow - DAGs
localhost:8080/admin/
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
13:36 UTC
DAGs
Show 25 entries
Search:
DAG
Schedule
Owner
Recent Tasks
Last Run
DAG Runs
Links
example_bash_operator
0 0 * * *
airflow
2017-12-17 00:00
example_http_operator
1 day, 0:00:00
2018-04-12 00:00
example_passing_params_via_test_command
*/1 * * * *
2018-04-12 00:21
example_python_operator
None
example_short_circuit_operator
2018-04-12 00:00
example_skip_dag
2018-04-12 00:00
ods_to_datamart
Showing 1 to 7 of 7 entries
Previous
1
Next
Show Paused DAGs
No. of successfully completed tasks
No. of Queued tasks
No. of failed tasks
Status for recent DAG Runs


Airflow - DAGs
localhost:8080/admin/
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
13:36 UTC
DAGs
Can pause a Dag by switching it Off
No of DAG instance running
No of times DAG fails to run
Show 25 entries
Search:
DAG
Schedule
Owner
Recent Tasks
Last Run
DAG Runs
Links
On example_bash_operator 0 0 * * * airflow 2017-12-17 00:00
On example_http_operator 1 day, 0:00:00 airflow 2018-04-12 00:00
On example_passing_params_via_test_command */1 * * * * airflow 2018-04-12 00:21
On example_python_operator None airflow
On example_short_circuit_operator 1 day, 0:00:00 airflow 2018-04-12 00:00
On example_skip_dag 1 day, 0:00:00 airflow 2018-04-12 00:00
On ods_to_datamart None airflow
Showing 1 to 7 of 7 entries
Previous 1 Next
Show Paused DAGs
No. of successful DAG runs


Airflow - DAGs
localhost:8080/admin/
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
13:36 UTC
Links to detailed DAG info
DAGs
Show 25 entries
Search:
DAG
Schedule
Owner
Recent Tasks
Last Run
DAG Runs
Links
On
example_bash_operator
0 0 * * *
airflow
2017-12-17 00:00
example_http_operator
1 day, 0:00:00
2018-04-12 00:00
example_passing_params_via_test_command
*/1 * * * *
2018-04-12 00:21
example_python_operator
None
example_short_circuit_operator
example_skip_dag
ods_to_datamart
Showing 1 to 7 of 7 entries
Previous
1
Next
Show Paused DAGs

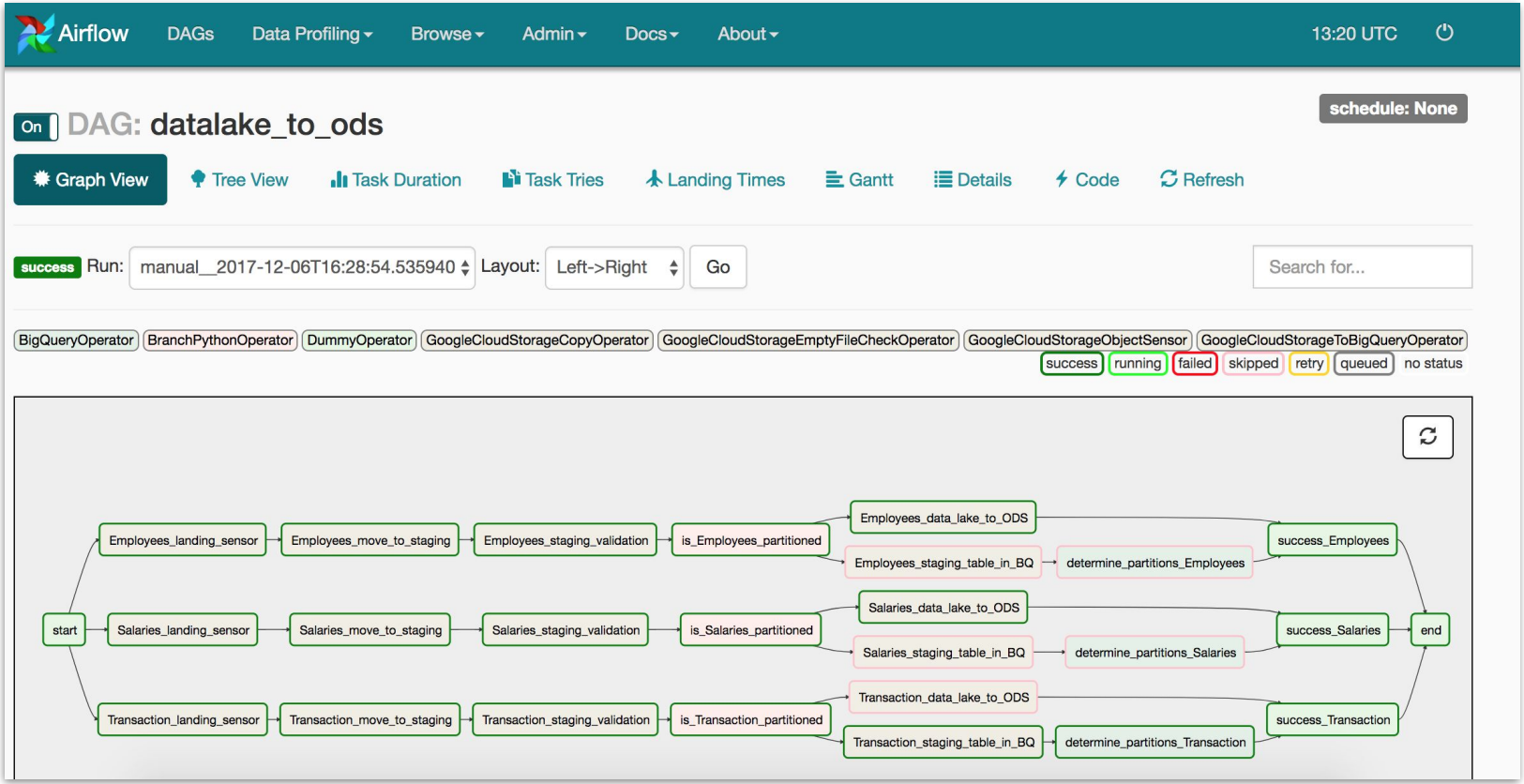
Web UI: Tree View

A tree representation of the DAG that spans across time(task run history)



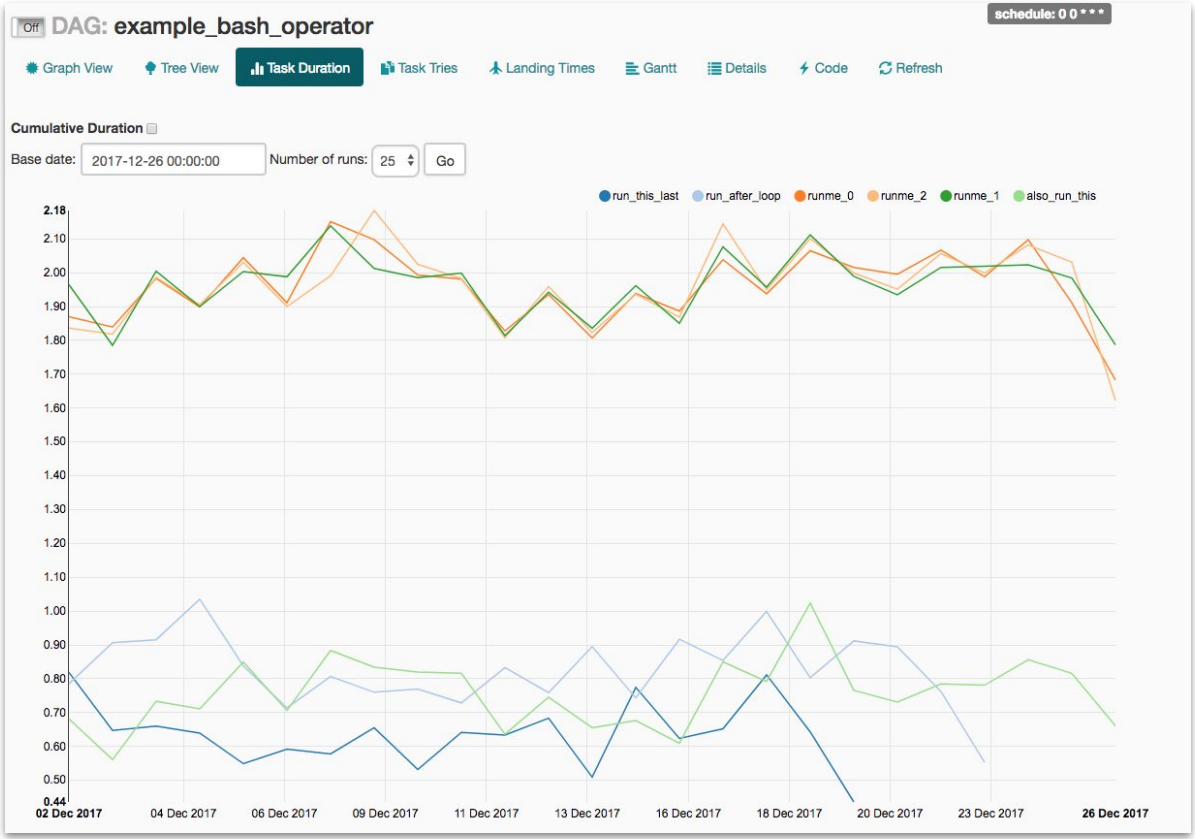
Web UI: Graph View

Visualize task dependencies & current status for a specific run



Web UI: Task Duration

The duration of your different tasks over the past N runs.



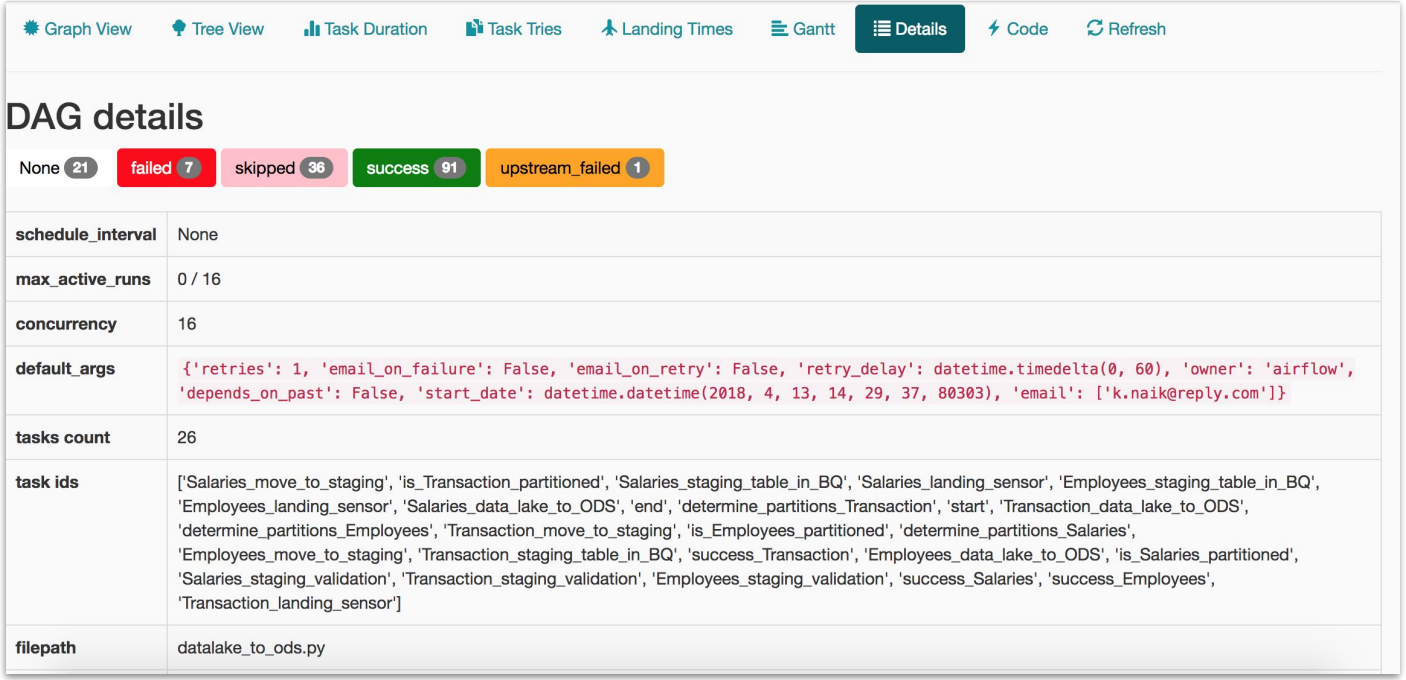
Web UI: Gantt

Which task is a blocker?

https://speakerdeck.com/artwr/apache-airflow-at-airbnb-introduction-and-lessons-learned

Web UI: DAG details

See task metadata, rendered template, execution logs etc… for debugging

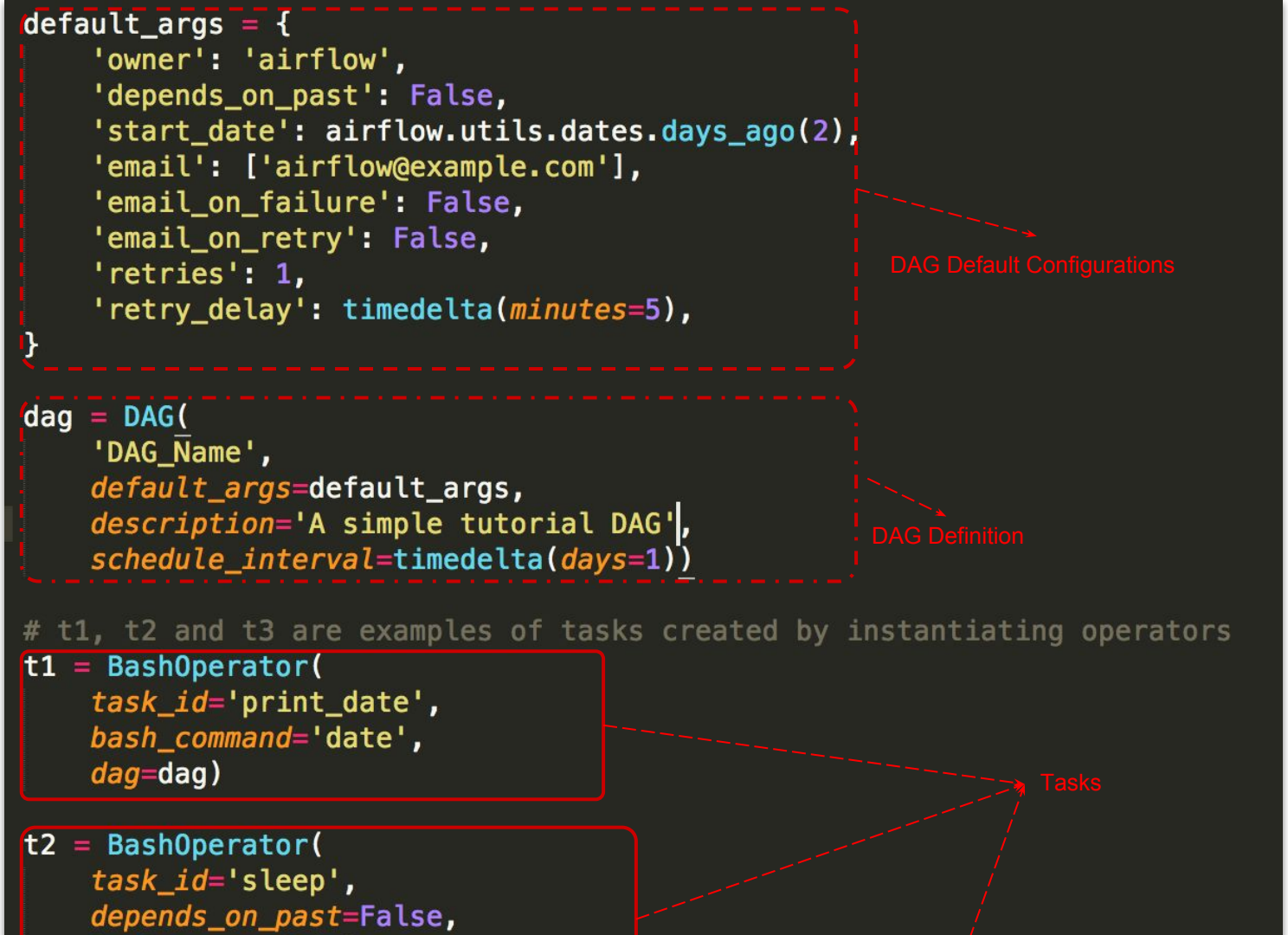


DAG Anatomy

Workflow as code

DAG Anatomy

Python Code


default_args = {
'owner': 'airflow',
'depends_on_past': False,
'start_date': airflow.utils.dates.days_ago(2),
'email': ['airflow@example.com'],
'email_on_failure': False,
'email_on_retry': False,
'retries': 1,
'retry_delay': timedelta(minutes=5),
}
DAG Default Configurations
dag = DAG(
'DAG_Name',
default_args=default_args,
description='A simple tutorial DAG',
schedule_interval=timedelta(days=1))
DAG Definition
# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
task_id='print_date',
bash_command='date',
dag=dag)
t2 = BashOperator(
task_id='sleep',
depends_on_past=False,
Tasks

```
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    dag=dag)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7)}}"
    echo "{{ params.my_param }}"
{% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)
```

Tasks

Task Dependencies

Workflow as code

DAG Anatomy

```python
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': airflow.utils.dates.days_ago(2),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'DAG_Name',
    default_args=default_args,
    description='A simple tutorial DAG',
    schedule_interval=timedelta(days=1))

# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    dag=dag)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7)}}"
    echo "{{ params.my_param }}"
{% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)
```

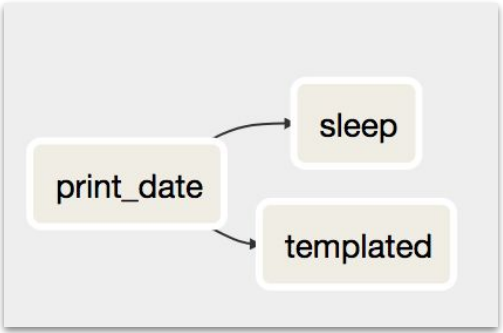
DAG Default Configurations

DAG Definition

Tasks

Task Dependencies

Python Code

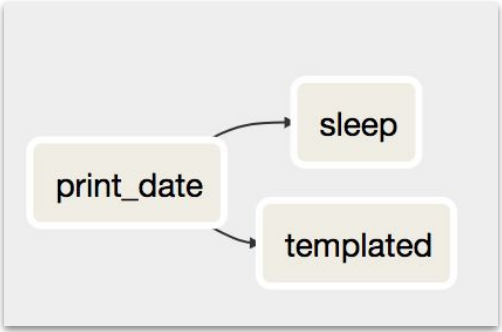


DAG (Workflow)

Concepts: Core Ideas

Concepts: DAG

- **DAG** - Directed Acyclic Graph



- Define workflow logic as shape of the graph

- It is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.

Concepts: OPERATORS

- Workflows are composed of **Operators**

- While DAGs describe how to run a workflow, Operators determine **what actually gets done**

```
t1 = BashOperator(
    task_id='print_date_time',
    bash_command='date',
    dag=dag)
```

Concepts: OPERATORS

- 3 main types of operators:
    - **Sensors** are a certain type of operator that will keep running until a certain criterion is met
    - Operators that perform an **action**, or tell another system to perform an action
    - **Transfer** Operators move data from one system to another

Concepts: TASKS

- A **parameterized** instance of an operator

  Once an operator is instantiated, it is referred as a “task”. The instantiation defines specific values when calling the abstract operator

- The parameterized task becomes a node in a DAG

Concepts: Tasks

```
t1 = BashOperator(
    task_id='print_date_time',
    bash_command='date',
    dag=dag)
```

Parameterised Operator

Setting Dependencies

t1.set_downstream(t2)

OR

t2.set_upstream(t1)

OR

t1 >> t2

Concepts:
TASK INSTANCE

- Represents a specific run of a task
- Characterized as the combination of a dag, a task, and a point in time.
- Task instances also have an indicative state, which could be “running”, “success”, “failed”, “skipped”, “up for retry”, etc.

Architecture

Architecture

Airflow comes with 5 main types of built in execution modes:

- Sequential
- Local

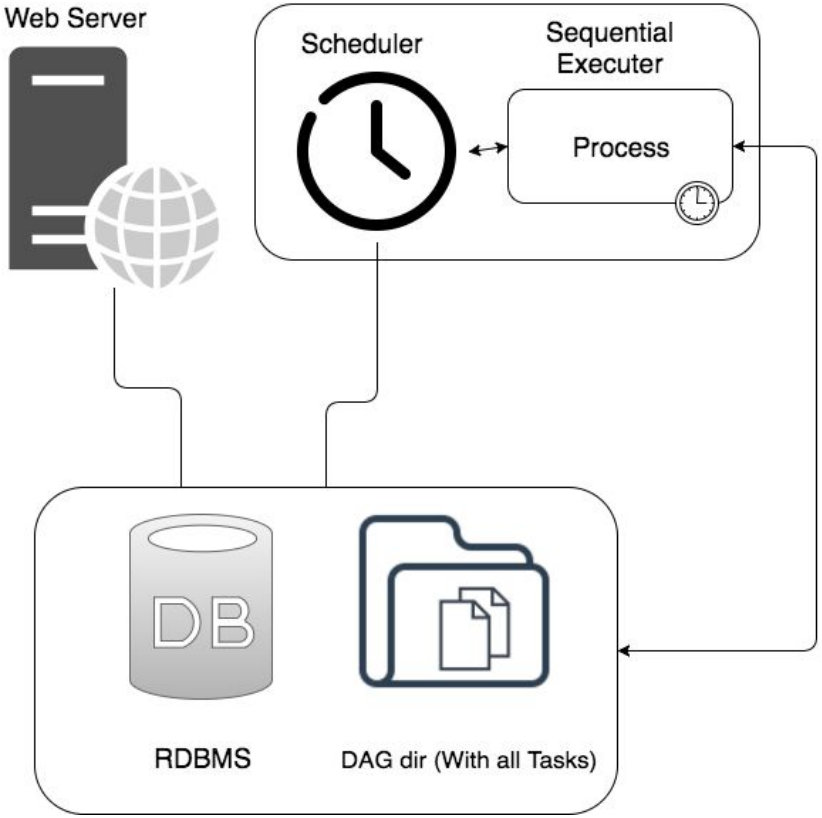
Runs on a single machine

- Celery (Out of the box)
- Mesos (Community driven)
- Kubernetes (Community driven)

Runs on distributed system

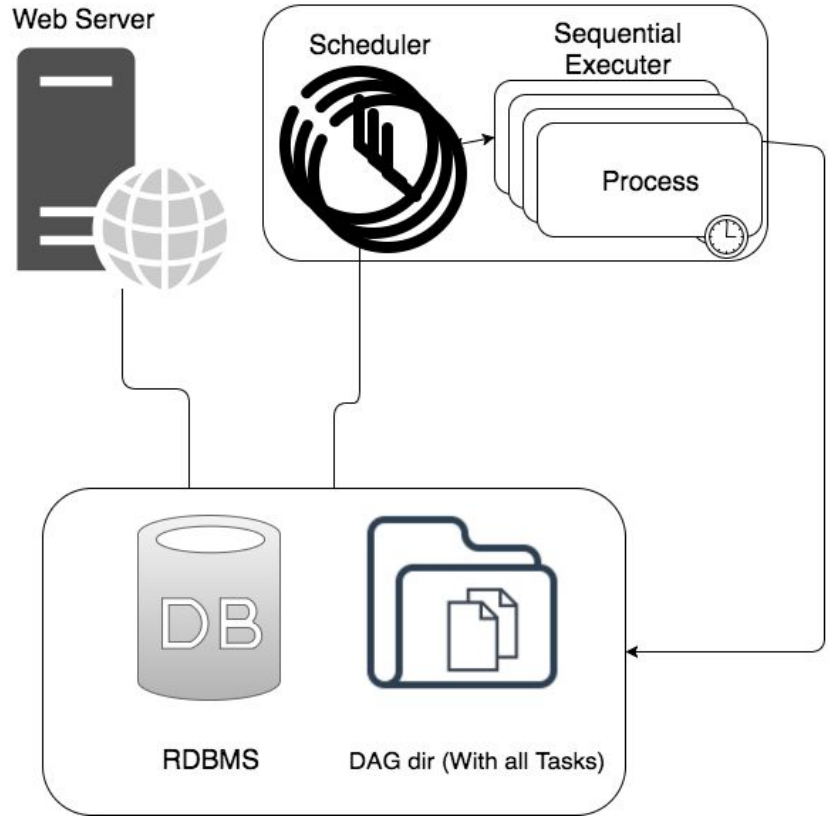
Architecture: Sequential Executor

- Default mode
- Minimum setup - work with sqlite as well
- Process 1 task at a time
- Good for demo purpose



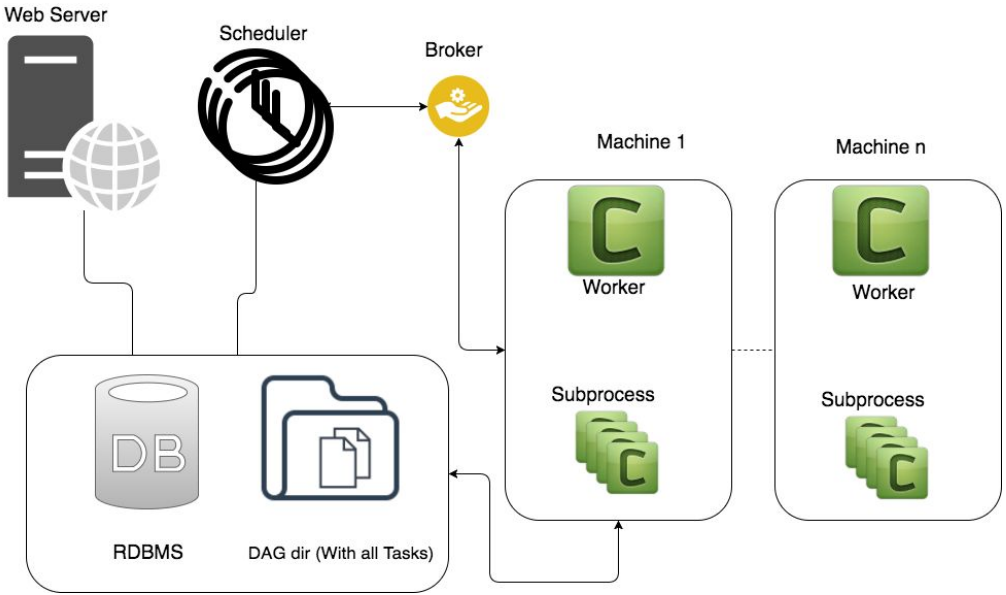
Architecture: Local Executor

- Spawned by scheduler process
- Vertical scaling
- Production grade
- Does not need broker or any other negotiator



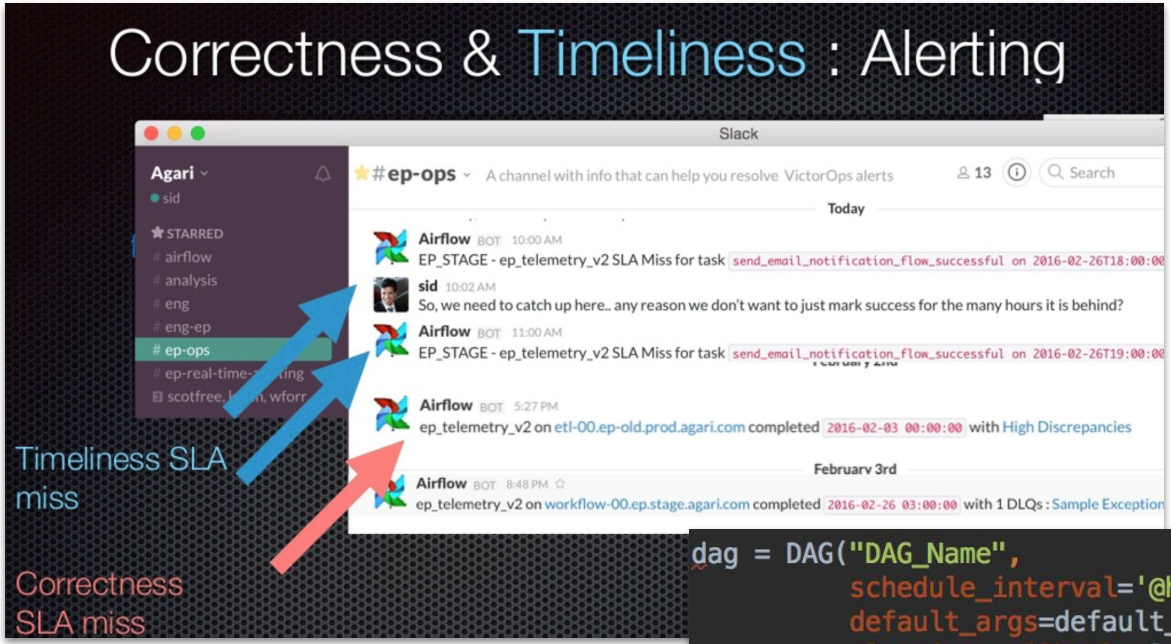
Architecture: Celery Executor

- Vertical & Horizontal scaling
- Production grade
- Can be monitored (Via Flower)
- Supports pool and queues



Useful Feature

Task callbacks for success / failure / SLA miss




```
dag = DAG("DAG_Name",
          schedule_interval='@hourly',
          default_args=default_args,
          sla_miss_callback=sla_alert_func)
```

https://speakerdeck.com/takus/building-data-pipelines-with-apache-airflow

Starting Airflow

Instructions to start Airflow

- **SetUp Airflow installation directory**
  ```
  $ export AIRFLOW_HOME=~/airflow
  ```

- **Initiating Airflow Database**
  ```
  $ source airflow_workshop/bin/activate
  $ airflow initdb
  ```

- **Start the web server, default port is 8080**
  ```
  $ airflow webserver -p 8080
  ```

- **Start the scheduler (In another terminal)**
  ```
  $ source airflow_workshop/bin/activate
  $ airflow scheduler
  ```

- **Visit localhost:8080 in the browser to see Airflow Web UI**

Copy DAGs

- **Clone the Git Repo**
  ```
  $ git clone https://github.com/DataReplyUK/airflow-workshop-pydata-london-2018.git
  ```

- **Copy the dags folder in AIRFLOW_HOME**
  ```
  $ cp -r airflow-workshop-pydata-london-2018/dags $AIRFLOW_HOME/dags
  ```

Tutorial 1: Basic Workflow

Tutorial 2: Dynamic Workflow

Advanced Concepts

- ➔ **XCom**
- ➔ **Trigger Rules**
- ➔ **Variables**
- ➔ **Branching**
- ➔ **SubDAGs**

Concepts: XCOM

- Abbreviation of “cross-communication”
- Means of communication between task instances
- Saved in database as a pickled object
- Best suited for small pieces of data (ids, etc.)

Concepts: Trigger Rule

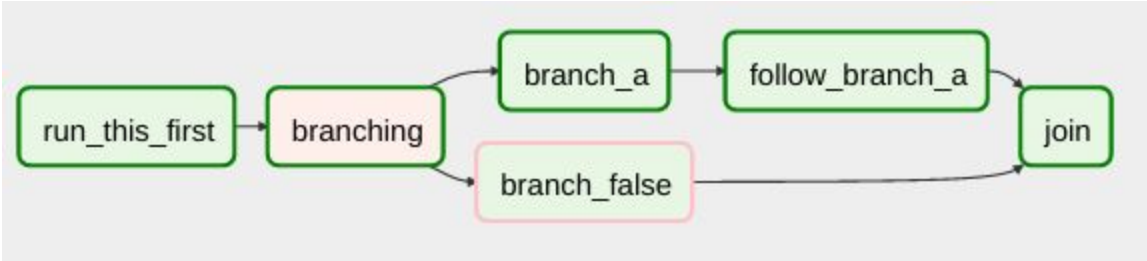
- Trigger condition for next upstream task
- Each operator has ‘trigger_rule’ argument
- Following are the different trigger rules:
  - **all_success**: (default) all **parents** have succeeded
  - **all_failed**: all **parents** are in a failed or **upstream_failed** state
  - **all_done**: all **parents** are done with their execution
  - **one_failed**: fires as soon as at least one parent has failed, it does not wait for all parents to be done
  - **one_success**: fires as soon as at least one parent succeeds, it does not wait for all parents to be done

Concepts: Variables

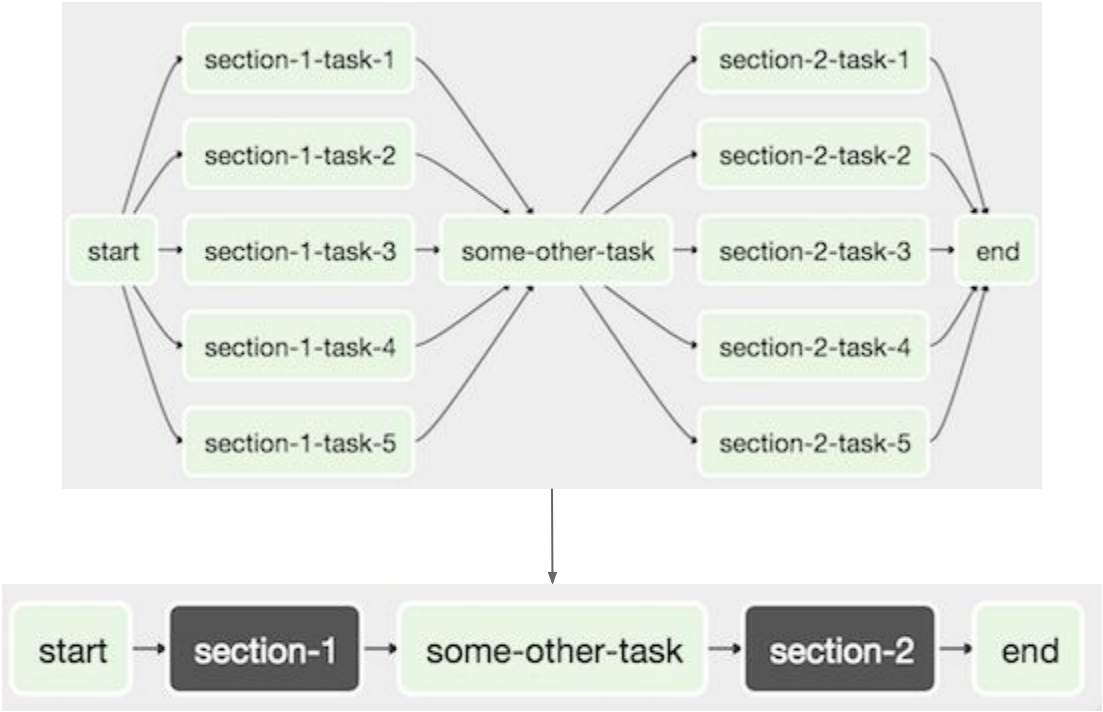
- Generic way to store and retrieve arbitrary content
- Can be used for storing settings as a simple key value store
- Variables can be created, updated, deleted and exported into json file form the UI

Concepts: Branching

- Branches the workflow based on condition.
- Condition can be defined using BranchPythonOperator



Concepts: SubDAGs



Perfect for repeating patterns

Tutorial 3: Workflow in GCP

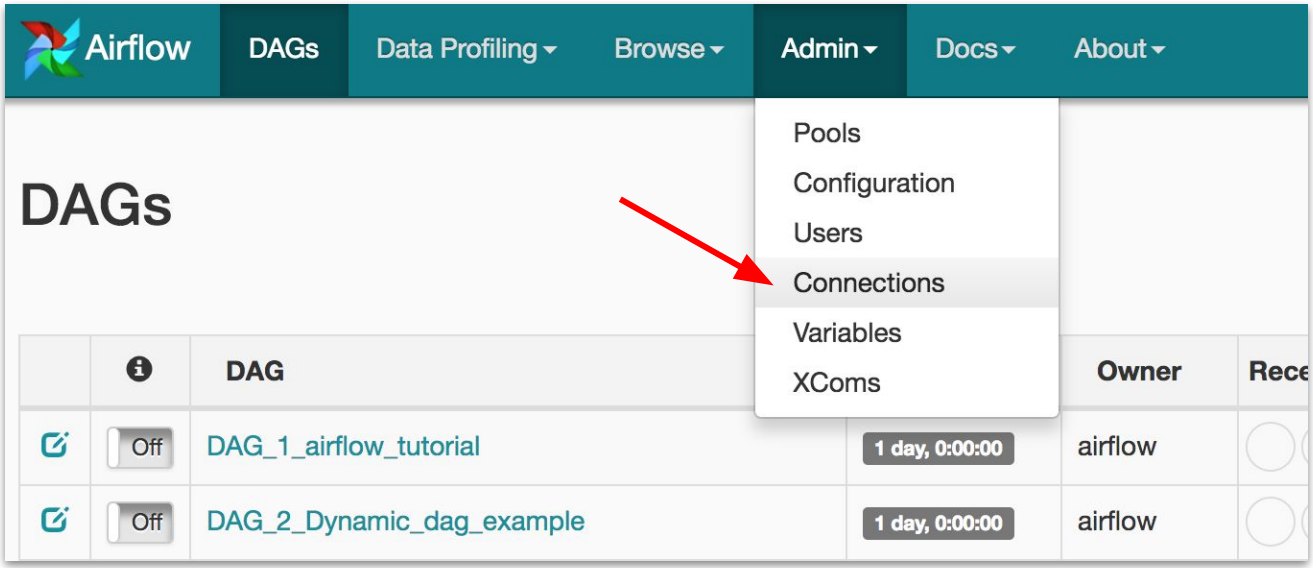
GCP: Introduction

- A cloud computing service offered by Google
- Popular products that we are going to use today:
  - Google Cloud storage: File and object storage
  - BigQuery: Large scale analytics data warehouse
  - And many more..

- Click here to access our GCP Project
  - Google Cloud Storage - link
  - BigQuery - link

Tutorial 3:
Create Connection

- Click on **Admin** ↠ **Connections**
- Or Visit http://localhost:8080/admin/connection/



Tutorial 3:
Create Connection

- Click on **Create** and enter following details:
  - **Conn Id**: airflow-service-account
  - **Conn Type**: Google Cloud Platform
  - **Project ID**: pydata2018-airflow
  - **Keyfile Path**: PATH_TO_YOUR_JSON_FILE
    - E.g. “/Users/kaxil/Desktop/Service_Account_Keys/sb01-service-account.json”
  - **Keyfile JSON**:
  - **Scopes**: https://www.googleapis.com/auth/cloud-platform
- Click on **Save**

Connection [create]

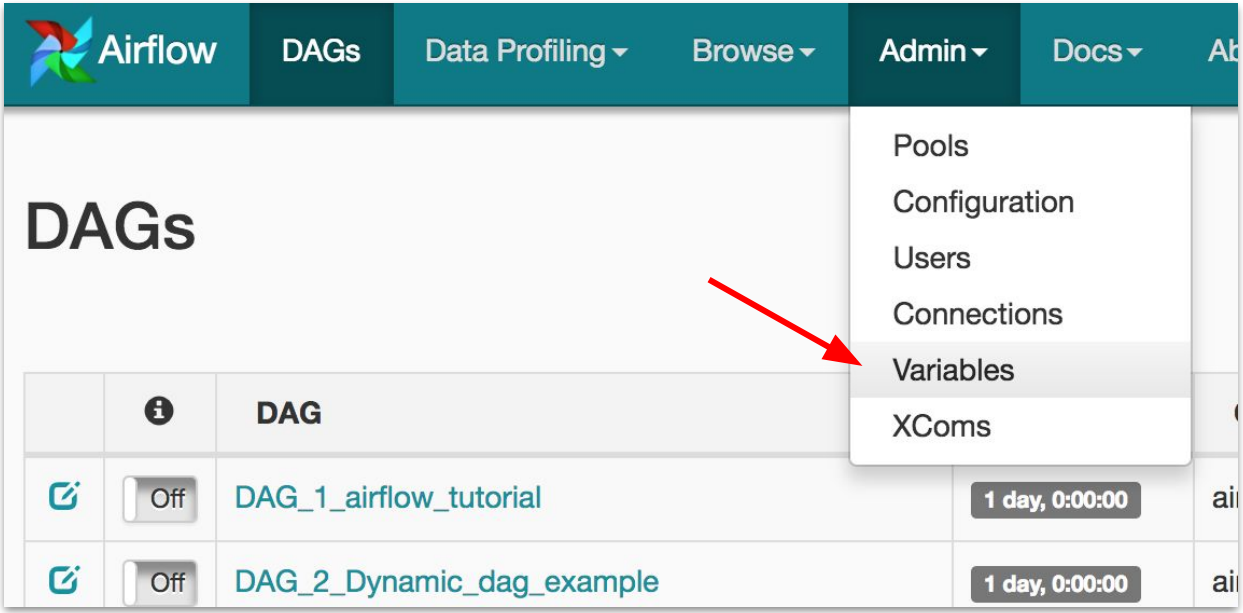
List | Create

Conn Id	airflow-service-account
Conn Type	Google Cloud Platform
Project Id	[redacted]
Keyfile Path	/Users/kaxil/Desktop/Service_Account_Keys/sb01-service-account.json
Keyfile JSON	
Scopes (comma seperated)	https://www.googleapis.com/auth/cloud-platform

Save | Save and Add Another | Save and Continue Editing | Cancel

Tutorial 3:
Import Variables

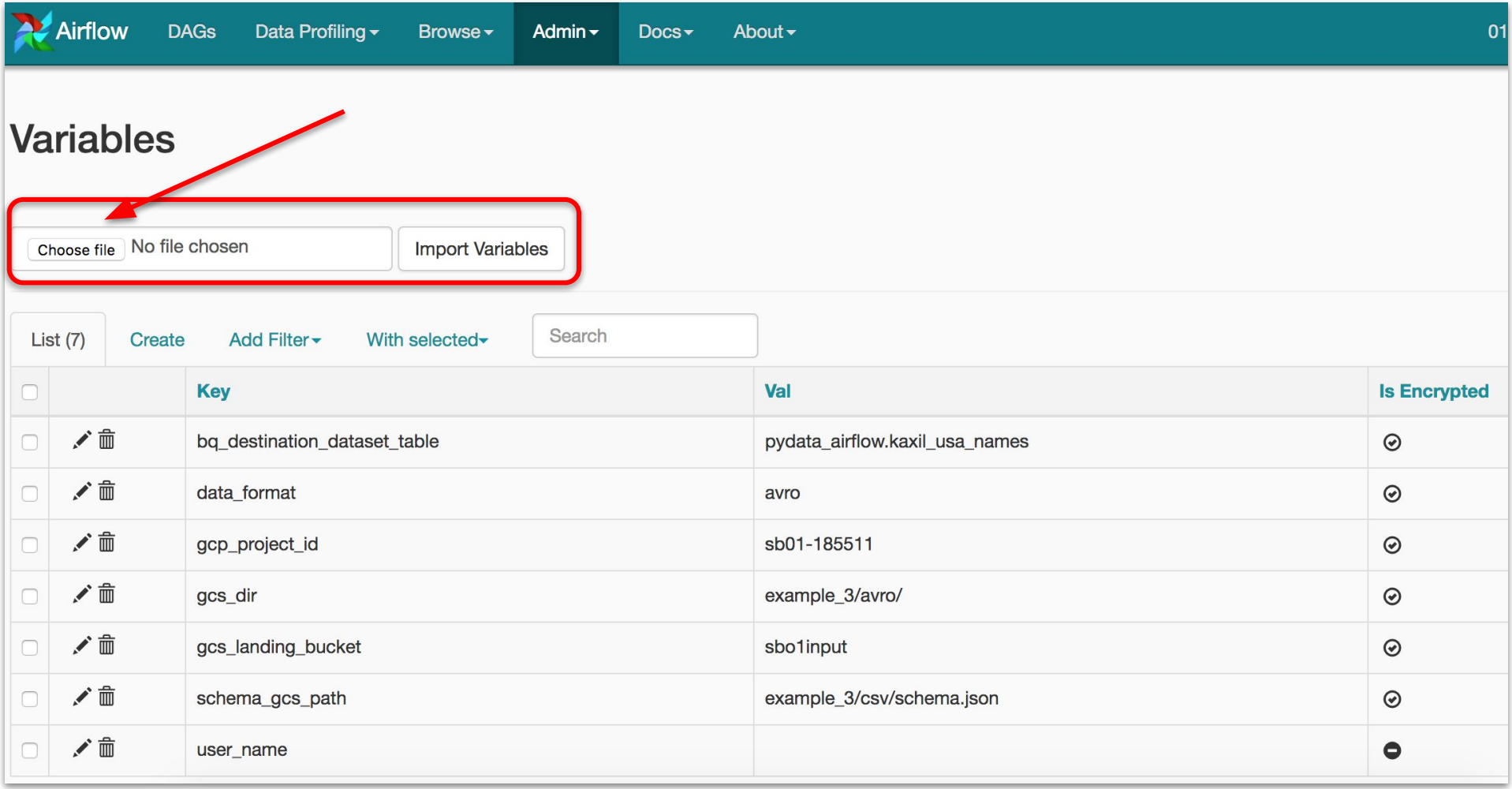
- Click on **Admin** ↠ **Variables**
- Or Visit http://localhost:8080/admin/variable/



Tutorial 3:
Import Variables

- Click on **Choose file** and select **variables.json** (file in the directory where you have cloned the Git repo)
- Click on **Import Variables** button.
- Edit **bq_destination_dataset_table** variable to enter:

  “pydata_airflow.kaxil_usa_names” after replacing **kaxil** with your **firstname_lastname**


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
01
Variables
Choose file
No file chosen
Import Variables
List (7)
Create
Add Filter
With selected
Search
Key
Val
Is Encrypted
bq_destination_dataset_table
pydata_airflow.kaxil_usa_names
data_format
avro
gcp_project_id
sb01-185511
gcs_dir
example_3/avro/
gcs_landing_bucket
sbo1input
schema_gcs_path
example_3/csv/schema.json
user_name

Tutorial 3

- **Objective**:
  - Waits for a file to be uploaded in Google Cloud Storage
  - Once the files are uploaded, a BigQuery table is created and the data from GCS is imported to it

- **Visit**:
  - Folder where files would be uploaded: **click here**
  - Dataset where the table would be created: **click here**

Tutorial 3

- Trigger **DAG_3_GCS_To_BigQuery** dag and check **Graph View** to see the current running task.

Summary

- Airflow = workflow as a code
- Integrates seamlessly into “pythonic” data science stack
- Easily extensible
- Clean management of workflow metadata
- Different alerting system (email, Slack)
- Huge community and under active development
- Proven real-world projects

Thank You