



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!

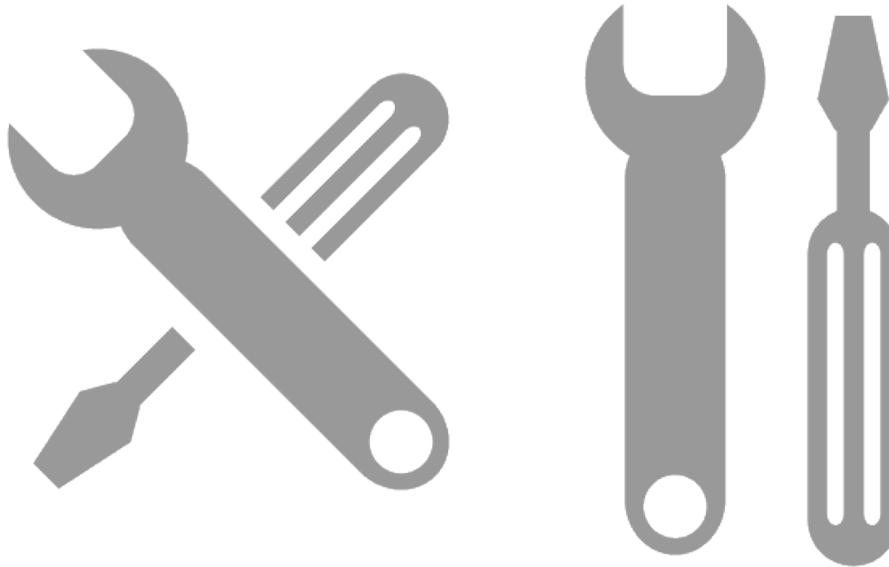
I am *Satyasheel*

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



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”



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"

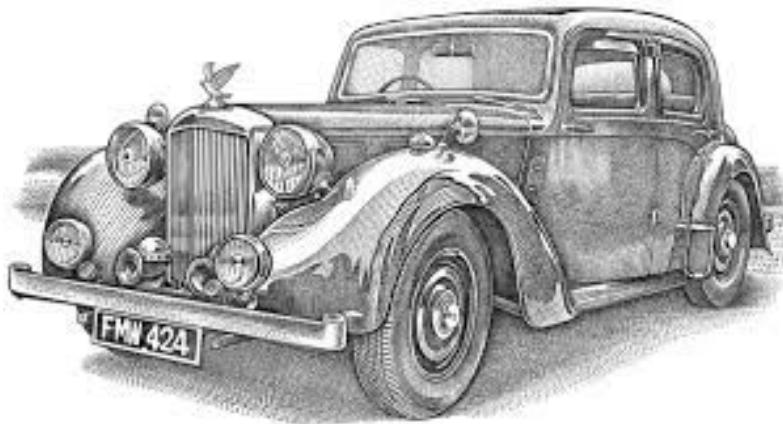
Oozie

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



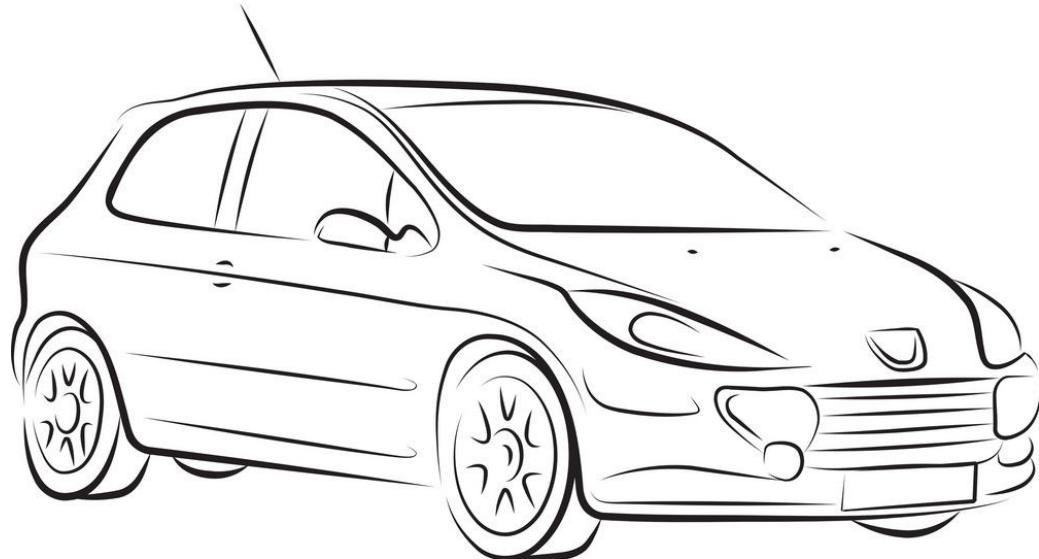
Luigi

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



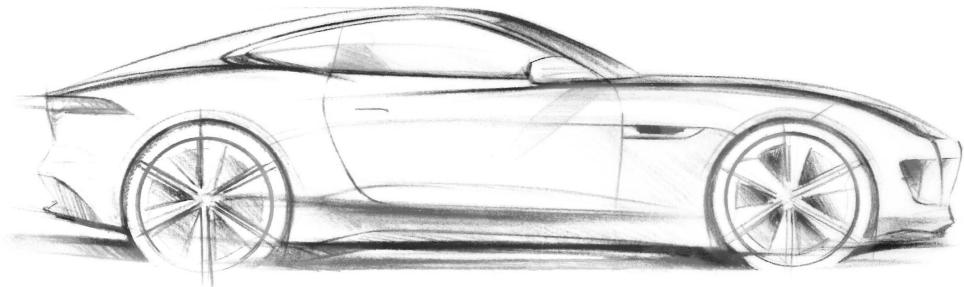
Airflow

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!!!



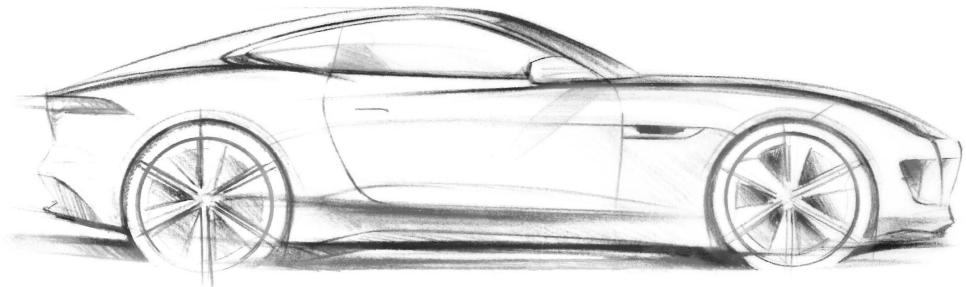
Airflow

Pros:

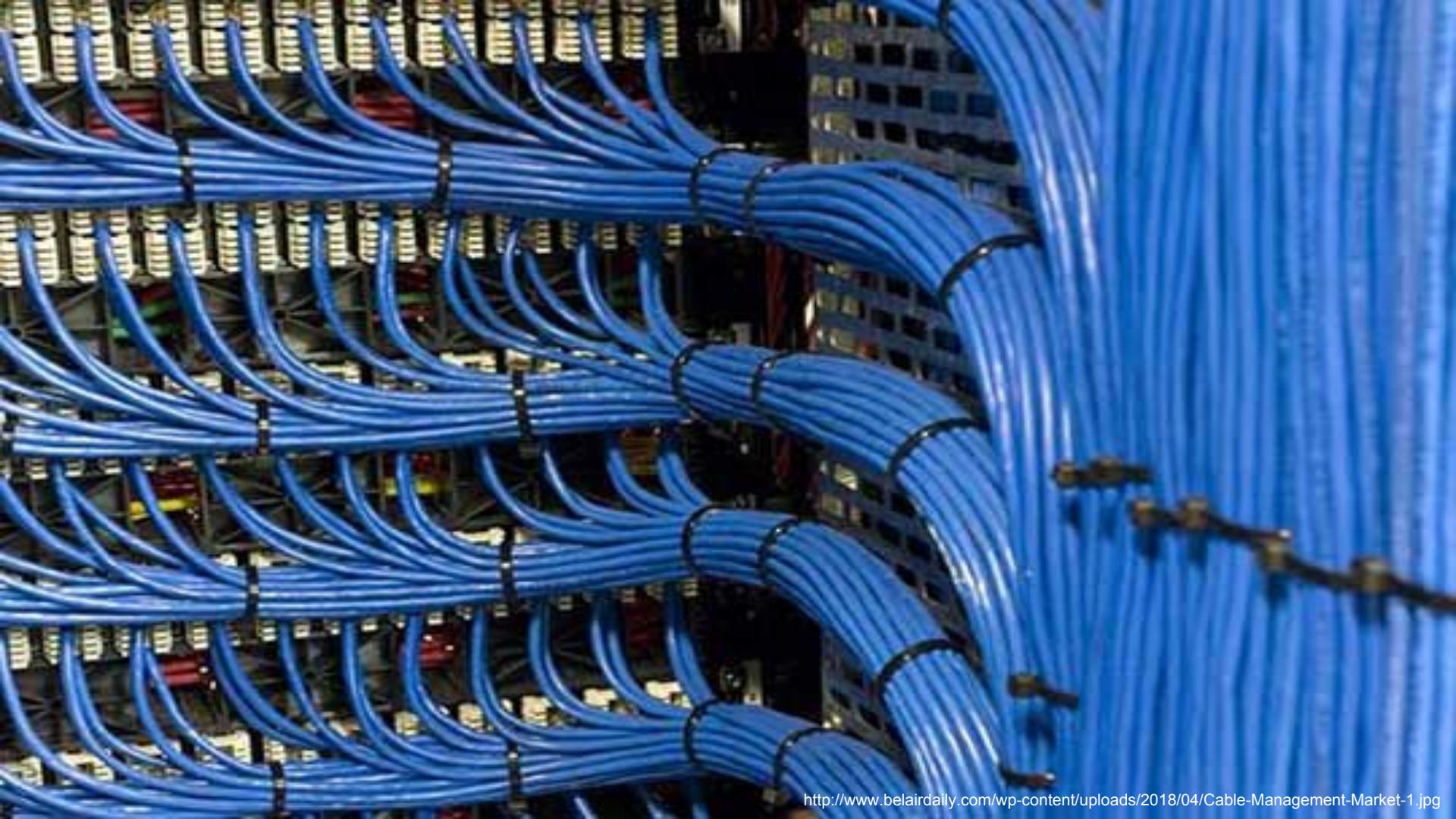
- Python Code Base
- Active community
- Trigger rules
- Cool web UI and rich CLI
- Queues & Pools
- Zombie Cleanup
- Easily extendable

Cons:

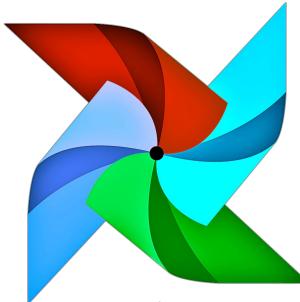
- ~~No role based access control~~
- Minor issues (Deleting DAGs is not straight forward)
- Umm!!!



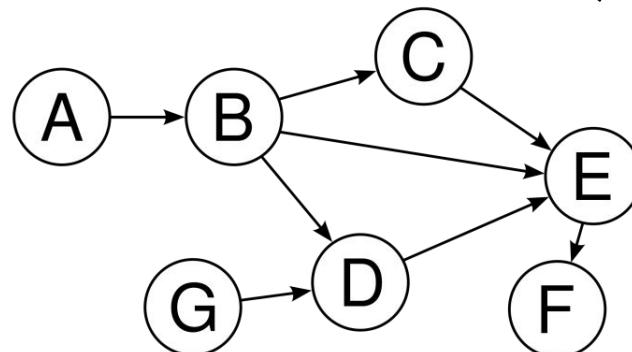




What is Airflow ?

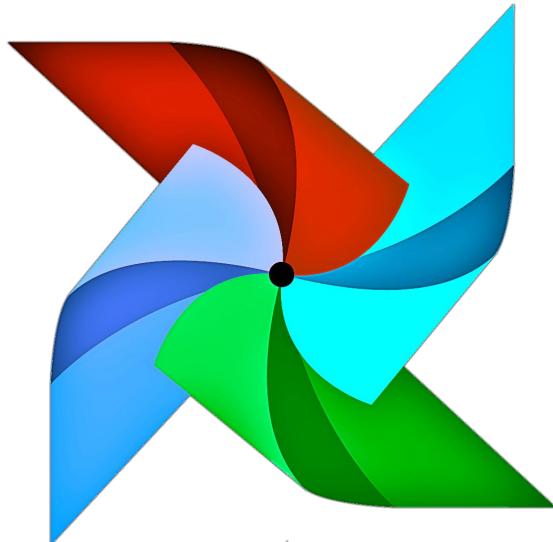


Airflow is a platform to programmatically author,
schedule and monitor workflows (**a.k.a DAGs**)



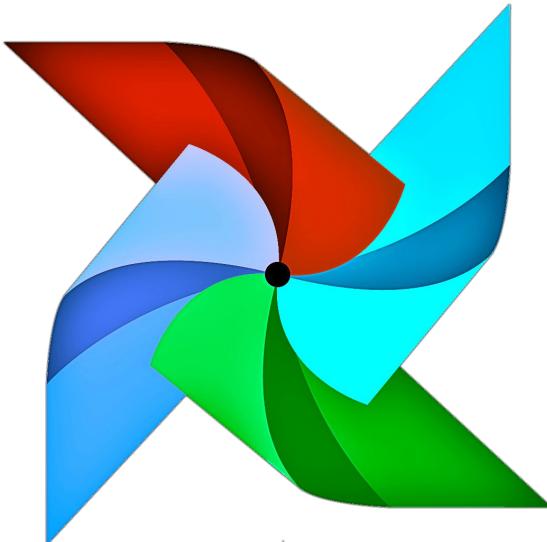
What is Airflow ?

- 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



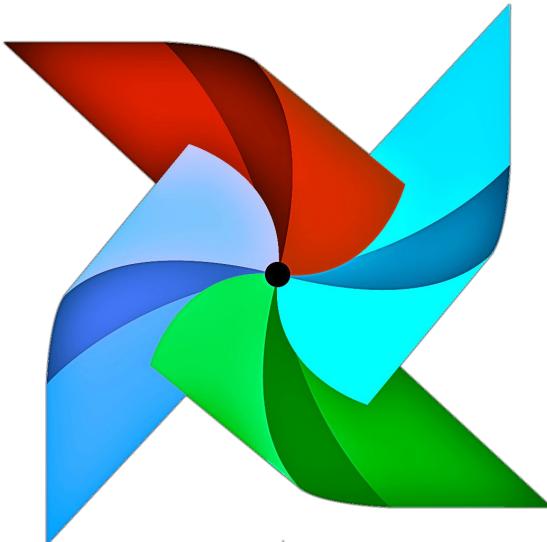
What is Airflow ?

- 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



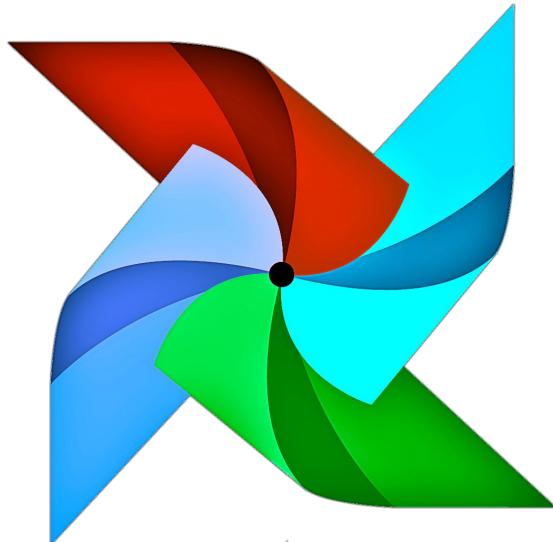
What is Airflow ?

- 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



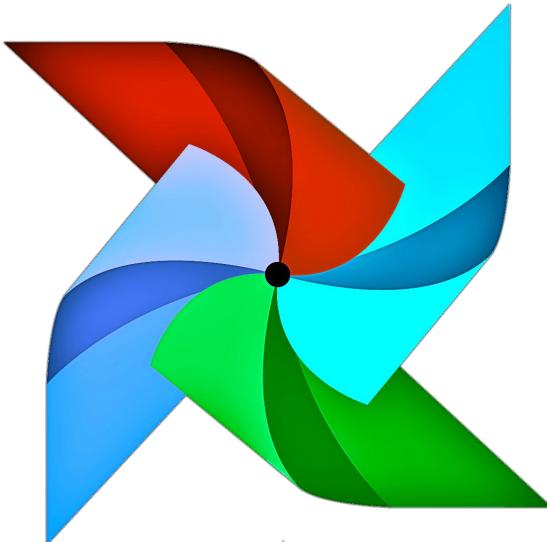
What is Airflow ?

- 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



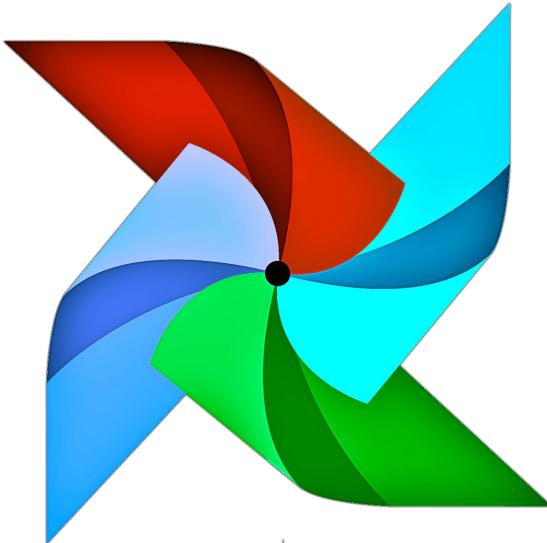
What is Airflow ?

- 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



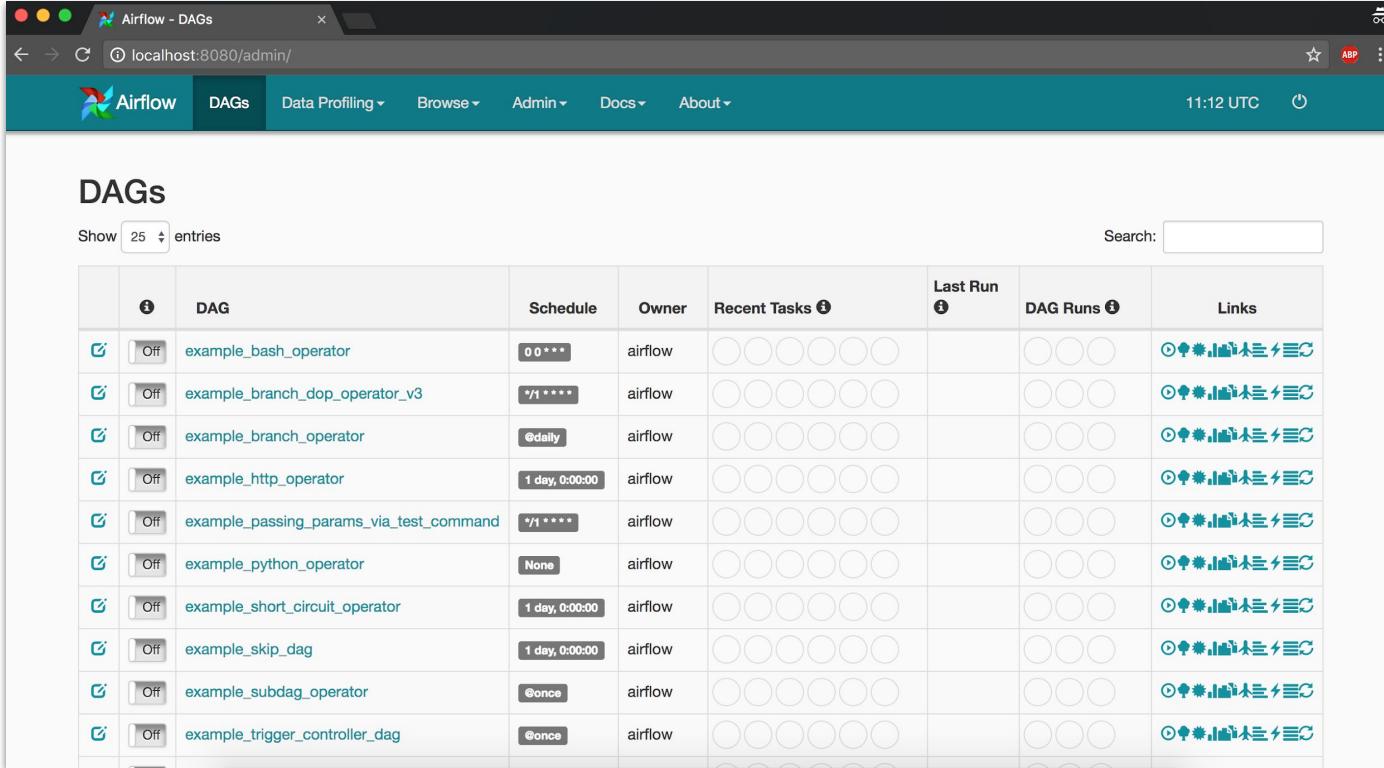
What is Airflow ?

- 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



The screenshot shows the Airflow Web UI interface. At the top, there's a navigation bar with tabs for 'DAGs' (which is selected), 'Data Profiling', 'Browse', 'Admin', 'Docs', and 'About'. The time '11:12 UTC' and a power icon are also visible. Below the navigation is a search bar labeled 'Search:' with a placeholder '(empty)'.

The main content area is titled 'DAGs' and displays a table of ten Airflow DAGs. The columns are: Action (checkbox), Schedule (button), DAG Name, Owner, Recent Tasks (button), Last Run (button), DAG Runs (button), and Links (button). The 'Recent Tasks' and 'Links' buttons are represented by a grid of nine circles.

Action	Schedule	DAG	Owner	Recent Tasks	Last Run	DAG Runs	Links
Off	0 * * * *	example_bash_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	* * * * *	example_branch_dop_operator_v3	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	@daily	example_branch_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	1 day, 0:00:00	example_http_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	* * * * *	example_passing_params_via_test_command	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	None	example_python_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	1 day, 0:00:00	example_short_circuit_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	1 day, 0:00:00	example_skip_dag	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	@once	example_subdag_operator	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️
Off	@once	example_trigger_controller_dag	airflow	○○○○○○○○○		○○○	🕒✖️✖️✖️✖️✖️✖️✖️✖️✖️

First look of Airflow WebUI right after installation



DAGs

Data Profiling ▾

Browse ▾

Admin ▾

Docs ▾

About ▾

13:36 UTC



DAGs

Similar to Cron

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	airflow		2018-04-12 00:00		
	example_passing_params_via_test_command	*/1 * * * *	airflow		2018-04-12 00:21		
	example_python_operator	None	airflow				
	example_short_circuit_operator	1 day, 0:00:00	airflow		2018-04-12 00:00		
	example_skip_dag	1 day, 0:00:00	airflow		2018-04-12 00:00		
	ods_to_datamart	None	airflow				

Showing 1 to 7 of 7 entries

Previous **1** Next[Show Paused DAGs](#)**This is how it looks once you start running your DAGs**

DAGs

Show 25 entries

Search:

	i DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
	example_bash_operator	0 0 ***	airflow		2017-12-17 00:00 i		
	example_http_operator	1 day, 0:00:0	airflow		2018-04-12 00:00 i		
	example_passing_params_via_test_command	*/* ***	airflow		2018-04-12 00:21 i		
	example_python_operator	None	airflow				
	example_short_circuit_operator	1 day, 0:00:0	airflow		2018-04-12 00:00 i		
	example_skip_dag	1 day, 0:00:0	airflow		2018-04-12 00:00 i		
	ods_to_datamart	None	airflow				

Showing 1 to 7 of 7 entries

Show Paused DAGs

No. of failed tasks

Status for recent DAG Runs



DAGs

Show 25 entries

Can pause a Dag by switching it Off

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	example_bash_operator	0 0 * * *	airflow	(47) (22)	2017-12-17 00:00	(13) (14) (2)	
<input checked="" type="checkbox"/>	example_http_operator	1 day, 0:00:00	airflow	(1) (1) (1)	2018-04-12 00:00	(1) (1)	
<input checked="" type="checkbox"/>	example_passing_params_via_test_command	*/* * * *	airflow	(6) (3)	2018-04-12 00:21	(15) (5) (2)	
<input checked="" type="checkbox"/>	example_python_operator	None	airflow	(0) (0) (0)		(0) (0) (0)	
<input checked="" type="checkbox"/>	example_short_circuit_operator	1 day, 0:00:00	airflow	(4) (2)	2018-04-12 00:00	(1) (1) (1)	
<input checked="" type="checkbox"/>	example_skip_dag	1 day, 0:00:00	airflow	(4) (4)	2018-04-12 00:00	(1) (1) (1)	
<input checked="" type="checkbox"/>	ods_to_datamart	None	airflow	(3) (0) (0)		(1) (0) (4)	

Showing 1 to 7 of 7 entries

Show Paused DAGs

No of DAG instance running

No of times DAG fails to run

No. of successful DAG runs

Previous 1 Next



DAGs

Data Profiling ▾

Browse ▾

Admin ▾

Docs ▾

About ▾

13:36 UTC



DAGs

Show 25 entries

Links to detailed DAG info

Search:

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	example_bash_operator	0 0 * * *	airflow		2017-12-17 00:00		
<input checked="" type="checkbox"/>	example_http_operator	1 day, 0:00:00	airflow		2018-04-12 00:00		
<input checked="" type="checkbox"/>	example_passing_params_via_test_command	*/* * * *	airflow		2018-04-12 00:21		
<input checked="" type="checkbox"/>	example_python_operator	None	airflow				
<input checked="" type="checkbox"/>	example_short_circuit_operator	1 day, 0:00:00	airflow		2018-04-12 00:00		
<input checked="" type="checkbox"/>	example_skip_dag	1 day, 0:00:00	airflow		2018-04-12 00:00		
<input checked="" type="checkbox"/>	ods_to_datamart	None	airflow				

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)

 Airflow DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾ 13:19 UTC ⚡

Off DAG: Orc_to_Avro_DataProc schedule: 1 day, 0:00:00

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Refresh

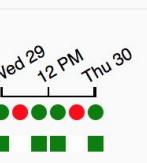
Base date: 2017-11-29 17:23:21 Number of runs: 25 Go

DataProcPySparkOperator DataprocClusterCreateOperator DataprocClusterDeleteOperator

success running failed skipped retry queued no status

[DAG] → delete_cluster → run_pyspark_job → start_cluster

Wed 29 12 PM Thu 30



Web UI: Graph View

Visualize task dependencies & current status for a specific run

Airflow DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾ 13:20 UTC ⚡

On DAG: **datalake_to_ods** schedule: None

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Refresh

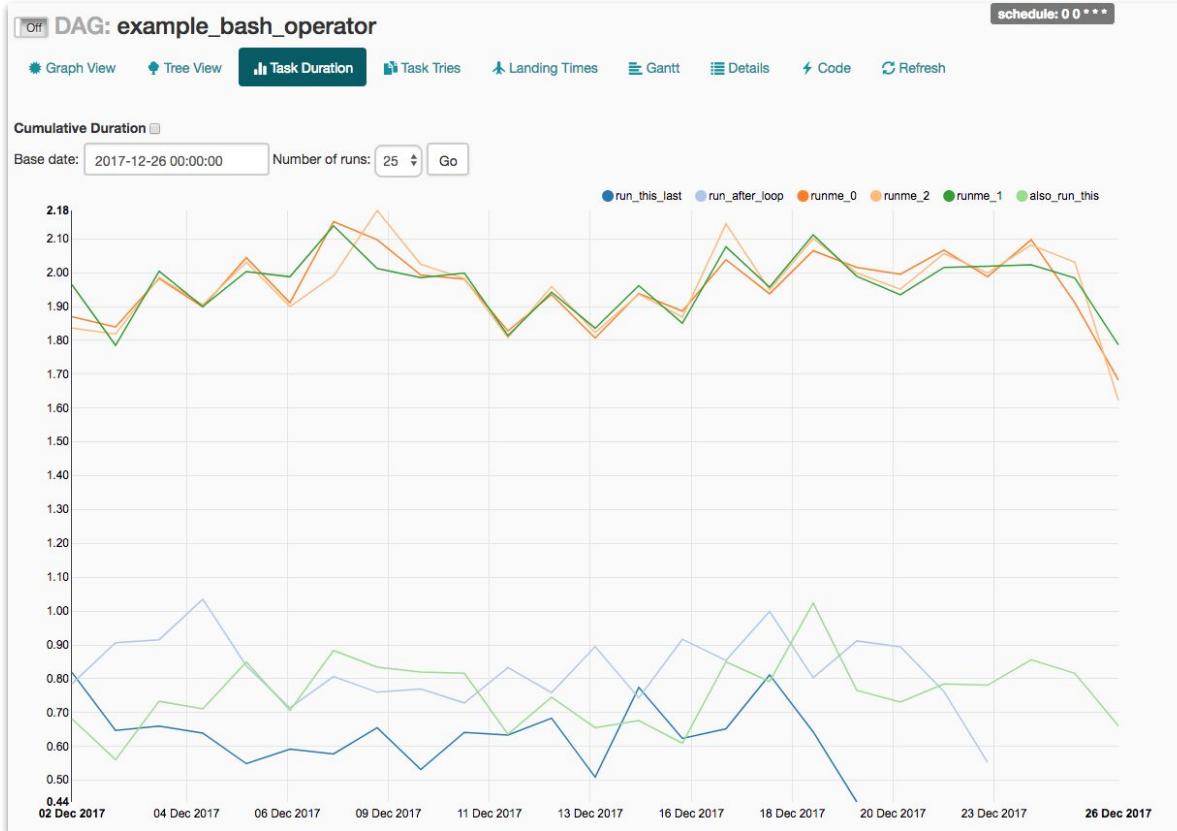
Run: manual_2017-12-06T16:28:54.535940 Layout: Left->Right Go Search for...

BigQueryOperator BranchPythonOperator DummyOperator GoogleCloudStorageCopyOperator GoogleCloudStorageEmptyFileCheckOperator GoogleCloudStorageObjectSensor GoogleCloudStorageToBigQueryOperator
success running failed skipped retry queued no status

```
graph TD; start((start)) --> Employees_landing_sensor[Employees_landing_sensor]; start --> Salaries_landing_sensor[Salaries_landing_sensor]; start --> Transaction_landing_sensor[Transaction_landing_sensor]; Employees_landing_sensor --> Employees_move_to_staging[Employees_move_to_staging]; Salaries_landing_sensor --> Salaries_move_to_staging[Salaries_move_to_staging]; Transaction_landing_sensor --> Transaction_move_to_staging[Transaction_move_to_staging]; Employees_move_to_staging --> Employees_staging_validation[Employees_staging_validation]; Salaries_move_to_staging --> Salaries_staging_validation[Salaries_staging_validation]; Transaction_move_to_staging --> Transaction_staging_validation[Transaction_staging_validation]; Employees_staging_validation --> is_Employees_partitioned[is_Employees_partitioned]; Salaries_staging_validation --> is_Salaries_partitioned[is_Salaries_partitioned]; Transaction_staging_validation --> is_Transaction_partitioned[is_Transaction_partitioned]; is_Employees_partitioned --> Employees_data_lake_to_ODS[Employees_data_lake_to_ODS]; is_Salaries_partitioned --> Salaries_data_lake_to_ODS[Salaries_data_lake_to_ODS]; is_Transaction_partitioned --> Transaction_data_lake_to_ODS[Transaction_data_lake_to_ODS]; Employees_data_lake_to_ODS --> Employees_staging_table_in_BQ[Employees_staging_table_in_BQ]; Salaries_data_lake_to_ODS --> Salaries_staging_table_in_BQ[Salaries_staging_table_in_BQ]; Transaction_data_lake_to_ODS --> Transaction_staging_table_in_BQ[Transaction_staging_table_in_BQ]; Employees_staging_table_in_BQ --> determine_partitions_Employees[determine_partitions_Employees]; Salaries_staging_table_in_BQ --> determine_partitions_Salaries[determine_partitions_Salaries]; Transaction_staging_table_in_BQ --> determine_partitions_Transaction[determine_partitions_Transaction]; determine_partitions_Employees --> success_Employees[success_Employees]; determine_partitions_Salaries --> success_Salaries[success_Salaries]; determine_partitions_Transaction --> success_Transaction[success_Transaction]; success_Employees --> end((end)); success_Salaries --> end; success_Transaction --> end;
```

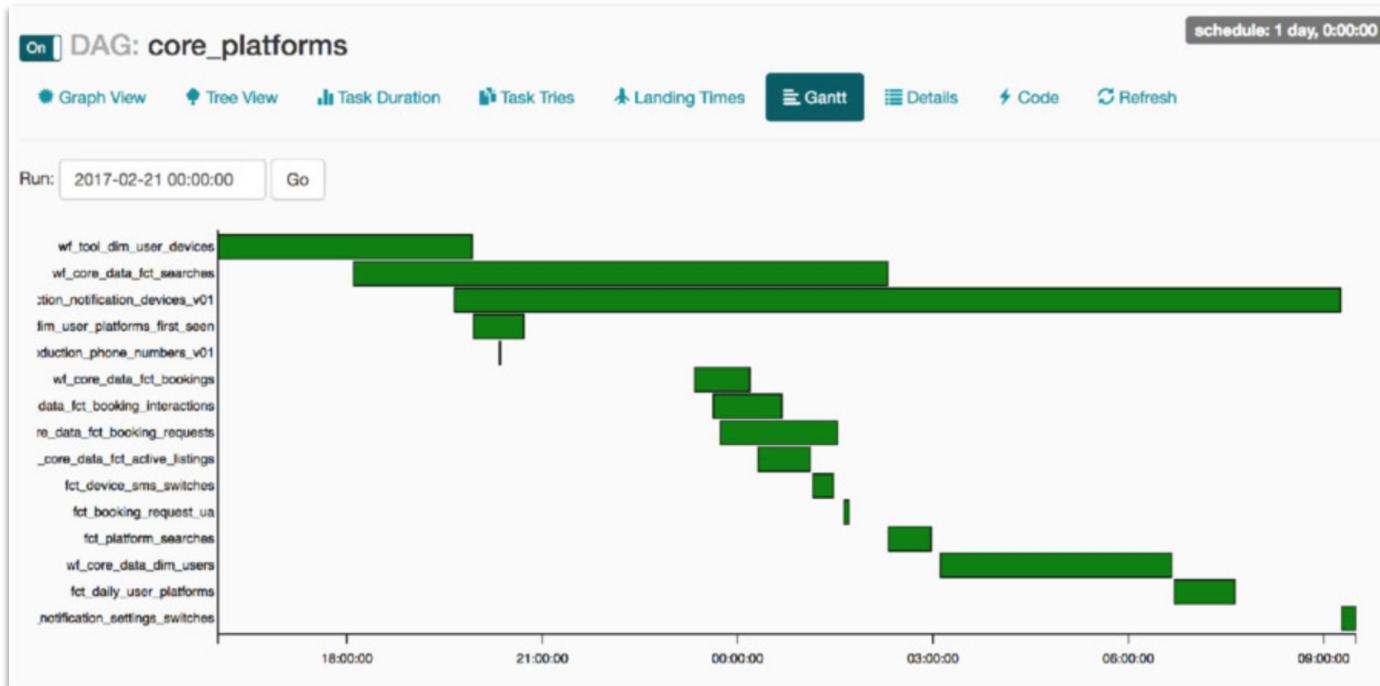
Web UI: Task Duration

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



Web UI: Gantt

Which task is a blocker?



Web UI: DAG details

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

DAG details	
None	21
failed	7
skipped	36
success	91
upstream_failed	1
schedule_interval	None
max_active_runs	0 / 16
concurrency	16
default_args	<pre>{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'airflow', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 13, 14, 29, 37, 80303), 'email': ['k.naik@reply.com']}</pre>
tasks count	26
task ids	<pre>['Salaries_move_to_staging', 'is_Transaction_partitioned', 'Salaries_staging_table_in_BQ', 'Salaries_landing_sensor', 'Employees_staging_table_in_BQ', 'Employees_landing_sensor', 'Salaries_data_lake_to_ODS', 'end', 'determine_partitions_Transaction', 'start', 'Transaction_data_lake_to_ODS', 'determine_partitions_Employees', 'Transaction_move_to_staging', 'is_Employees_partitioned', 'determine_partitions_Salaries', 'Employees_move_to_staging', 'Transaction_staging_table_in_BQ', 'success_Transaction', 'Employees_data_lake_to_ODS', 'is_Salaries_partitioned', 'Salaries_staging_validation', 'Transaction_staging_validation', 'Employees_staging_validation', 'success_Salaries', 'success_Employees', 'Transaction_landing_sensor']</pre>
filepath	datalake_to_ods.py

DAG Anatomy

Workflow as code

DAG Anatomy

```
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)
```

Python Code

DAG Default Configurations

DAG Definition

Tasks

Task Dependencies

```
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)
```

Tasks

```
t2 = BashOperator(  
    task_id='sleep',  
    depends_on_past=False,
```

```
    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

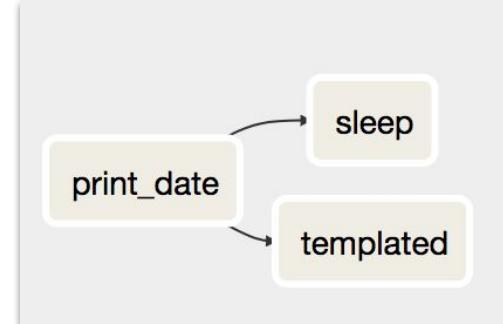
```
default_args = {  
    'owner': 'airflow',  
    'depends_on_past': False,  
    'start_date': airflow.utils.dates.days_ago(2),  
    'email': ['airflow@example.com'],  
    'email_on_failure': False,  
    'email_on_retry': False,  
    'retries': 1,  
    'retry_delay': timedelta(minutes=5),  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

Python Code

DAG Definition

Tasks

Task Dependencies

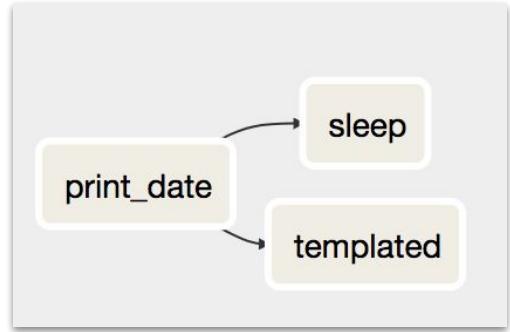


DAG
(Workflow)

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

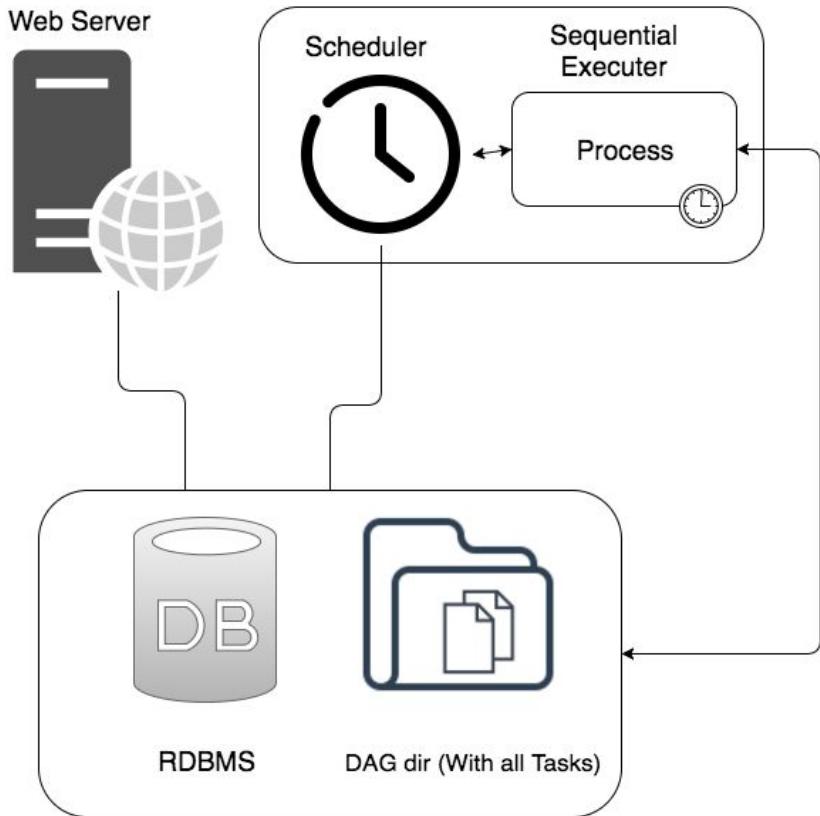
Airflow comes with 5 main types of built in execution modes:

- Sequential
 - Local
- 
- Runs on a single machine

-
- Celery (Out of the box)
 - Mesos (Community driven)
 - Kubernetes (Community driven)
- 
- Runs on distributed system

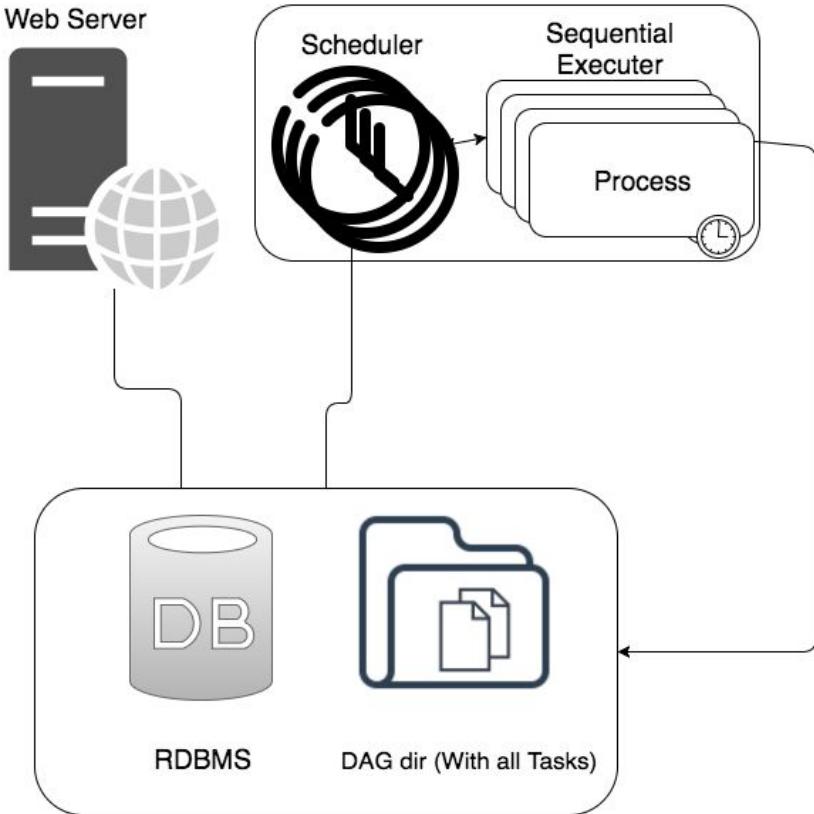
Architecture: Sequential Executor

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



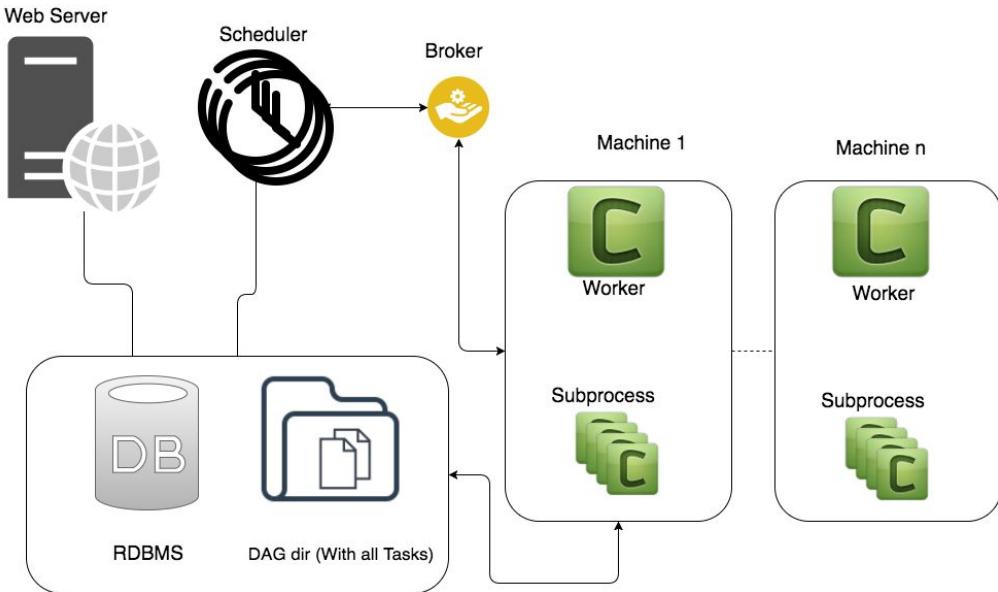
Architecture: Local Executor

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



Architecture: Celery Executor

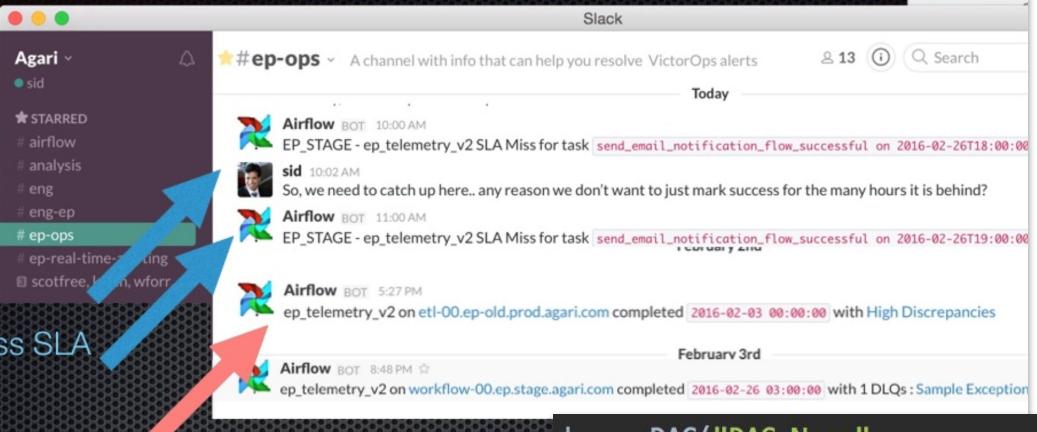
- Vertical & Horizontal scaling
- Production grade
- Can be monitored (Via Flower)
- Supports pool and queues



Useful Feature

Task callbacks for success / failure / SLA miss

Correctness & Timeliness : Alerting



Timeliness SLA miss

Correctness SLA miss

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

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

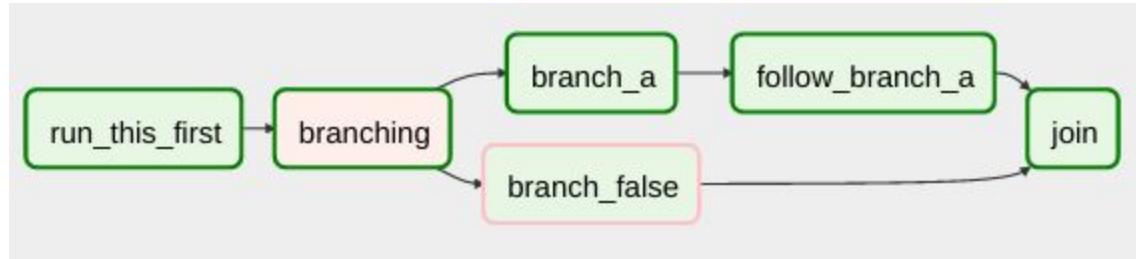
- Trigger condition for next upstream task
- Each operator has '**trigger_rule**' argument
- Following are the different trigger rules:
 - **all_success**: (default) all **parents** have succeeded
 - **all_failed**: all **parents** are in a failed or **upstream_failed** state
 - **all_done**: all **parents** 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

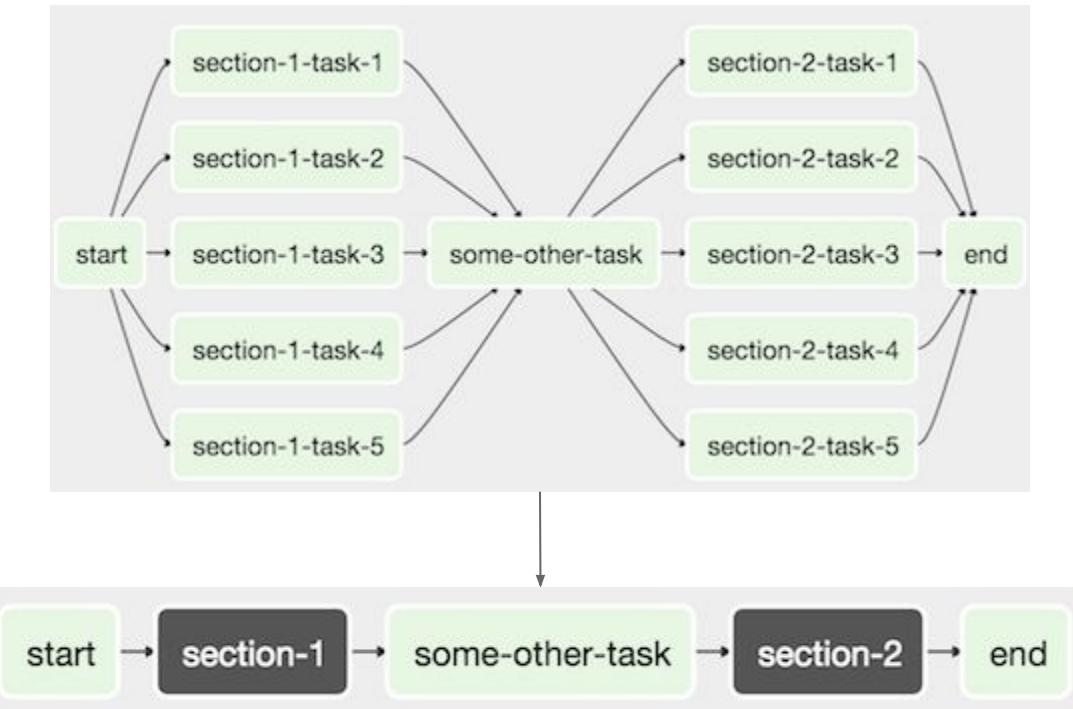
- Generic way to store and retrieve arbitrary content
- Can be used for storing settings as a simple key value store
- Variables can be created, updated, deleted and exported into json file form the UI

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/>

The screenshot shows the Airflow web interface. At the top, there is a navigation bar with the Airflow logo, followed by links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The Admin link is currently selected, and a dropdown menu appears, listing Pools, Configuration, Users, Connections (which is highlighted with a grey background), Variables, and XComs. Below the navigation bar, the main content area is titled "DAGs". It displays a table with two rows of DAG information. The columns include a checkbox, an info icon, the DAG name, its current state (Off), and its execution duration (1 day, 0:00:00). The table also includes columns for Owner (airflow) and Rec (represented by empty circles).

	i	DAG	State	Duration	Owner	Rec
<input checked="" type="checkbox"/>	Off	DAG_1_airflow_tutorial	Off	1 day, 0:00:00	airflow	<input type="radio"/>
<input checked="" type="checkbox"/>	Off	DAG_2_Dynamic_dag_example	Off	1 day, 0:00:00	airflow	<input type="radio"/>

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



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/>

The screenshot shows the Airflow web interface with a teal header bar. On the left, there's a logo and the word "Airflow". To the right are several navigation links: "DAGs", "Data Profiling", "Browse", "Admin", "Docs", and "About". A red arrow points from the text below to the "Admin" dropdown menu, which is currently open, revealing options like "Pools", "Configuration", "Users", "Connections", "Variables", and "XComs". Below the header, the main content area has a title "DAGs" and a table listing two DAGs: "DAG_1_airflow_tutorial" and "DAG_2_Dynamic_dag_example". Each row in the table includes a status icon, a switch button labeled "Off", the DAG name, and a duration of "1 day, 0:00:00".

		DAG	
		DAG_1_airflow_tutorial	1 day, 0:00:00
		DAG_2_Dynamic_dag_example	1 day, 0:00:00

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**

Variables

 No file chosen

List (7)

Create

Add Filter ▾

With selected ▾

Search

	Key	Val	Is Encrypted
<input type="checkbox"/>	bq_destination_dataset_table	pydata_airflow.kaxil_usa_names	<input checked="" type="checkbox"/>
<input type="checkbox"/>	data_format	avro	<input checked="" type="checkbox"/>
<input type="checkbox"/>	gcp_project_id	sb01-185511	<input checked="" type="checkbox"/>
<input type="checkbox"/>	gcs_dir	example_3/avro/	<input checked="" type="checkbox"/>
<input type="checkbox"/>	gcs_landing_bucket	sbo1input	<input checked="" type="checkbox"/>
<input type="checkbox"/>	schema_gcs_path	example_3/csv/schema.json	<input checked="" type="checkbox"/>
<input type="checkbox"/>	user_name		<input type="checkbox"/>

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

