Introduction to Airflow in Python

Table of Contents

Basic Concepts: ... ***1***

Airflow DAGs ... ***2***

DAG attributes ... **2**

Define a DAG in Python ... **2**

DAGs on the command line ... **3**

Run a workflow in Airflow ... 3

Command Line vs Python ... **3**

Airflow Web UI ... ***4***

DAG detail page ... **4**

DAG Tree View ... **5**

DAG Graph View ... **5**

DAG Code View ... **6**

Logs page ... **6**

Command Line vs Web UI ... **7**

Basic Concepts:

- **Data Engineering** = taking any action involving data and turning it into a reliable, repeatable, and maintainable process.
- **Workflow** =
  - A **set of steps** to accomplish a given **data engineering task**
    - Such as: downloading files, copying data, filtering information, writing to a database, etc.
  - Of varying levels of **complexity**
    - Some workflows may only have 2 or 3 steps, while others consist of hundreds of components
  - A term with various meaning depending on context – Specific meaning within specific tools
- **Airflow** is a platform to program workflows:
  - Creation, scheduling, and monitoring workflows

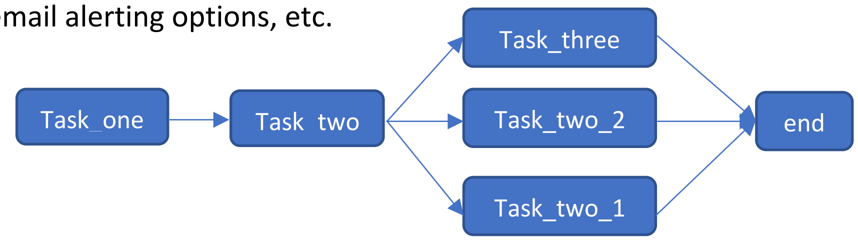
- It can use various tools and languages, but the actual workflow code is written with Python
- Implements workflows as **DAGs** (Directed Acyclic Graphs) – set of tasks and the dependencies between them
- Accessed via code, command-line, or via web interface

Airflow is not the only tool for running **data engineering workflows**. Some other options are **Spotify's Luigi**, **Microsoft's SSIS**, or even just **Bash scripting** within our Airflow usage

Airflow DAGs

Directed Acyclic Graph

- In Airflow, it represents the **set of tasks** that make up your **workflow**
- It **consists** of the **tasks** and the **dependencies** between tasks
- Created with various details about the DAG, including the name, start date, owner, email alerting options, etc.



DAG attributes

- **Directed**, there is an inherent flow representing dependencies or order between execution of components.
    - These dependencies (even the implicit ones) provide context to the tools on how to order the running of components
- **Acyclic**, does not loop/cycle/repeat. This does not imply that the entire DAG cannot be rerun, only that the **individual components** are **executed once per run**.
- **Graph**, represents the **components** and the **relationships** (dependencies) between them

Define a DAG in Python

```
etl_dag = DAG(
    dag_id = "etl_pipeline",
    default_args = {"start_date" : "2020-01-08"}
)
```

** Note that within any python code, DAG is referred to via the variable identifier, etl_dag, but within the Airflow shell command, you must use the dag_id.

```
from airflow.models import DAG #Import the DAG object from airflow.models
from datetime import datetime
```

```
#Create a default arguments dictionary consisting of attributes
#that will be applied to the components of our DAG
#These attributes are optional but provide a lot of power to define the runtime
behavior of Airflow
default_args = {
    "owner": "jdoe",
    "email": "jdoe@datacamp.com", #email address for alerting
    "start_date": datetime(2020,01,08) #The earliest datetime that a DAG could
be run
}

#Define the DAG object with the first argument using a name for the DAG, and
assign the default arguments dictionary to the default_args argument
etl_dag = DAG("etl_workflow", default_args=default_arguments)

#The variable etl_dag does not actually appear in Airflow interfaces
```

DAGs on the command line

- The **airflow command** line program contains many subcommands. Many are related to DAGs
  - `airflow -h` for descriptions
  - `airflow list_dags` to show all recognized DAGs

Run a workflow in Airflow

```
airflow run <dag_id> <task_id> <start_date>
```

For example, using a DAG named example-etl, a task named download-file and a start date of 2020-01-10, our command will look like:

```
airflow run example-etl download-file 2020-01-10
```

This task would simply download a specific file.

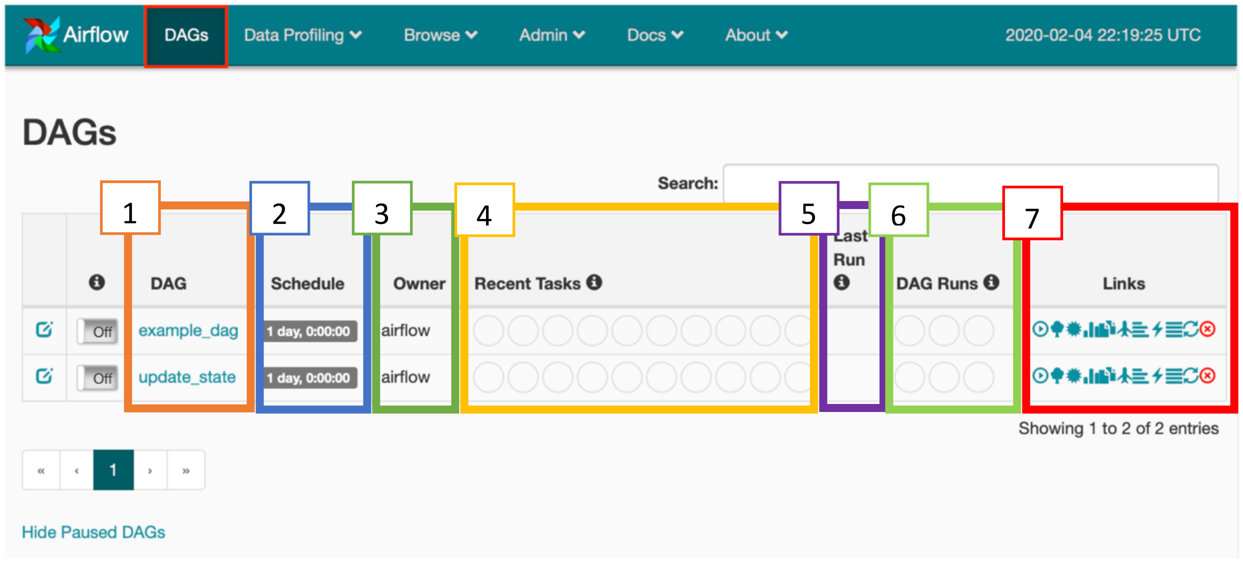
Remember you can use the **`airflow -h`** command to obtain further information about any **Airflow command.**

Command Line vs Python

(When to use the Airflow command tool vs writing Python)

Use the command line tool to:	Use Python to:
• Start Airflow processes • Manually run DAGs/Tasks • Review logging information	• Create a DAG • Edit the individual properties of a DAG

Airflow Web UI

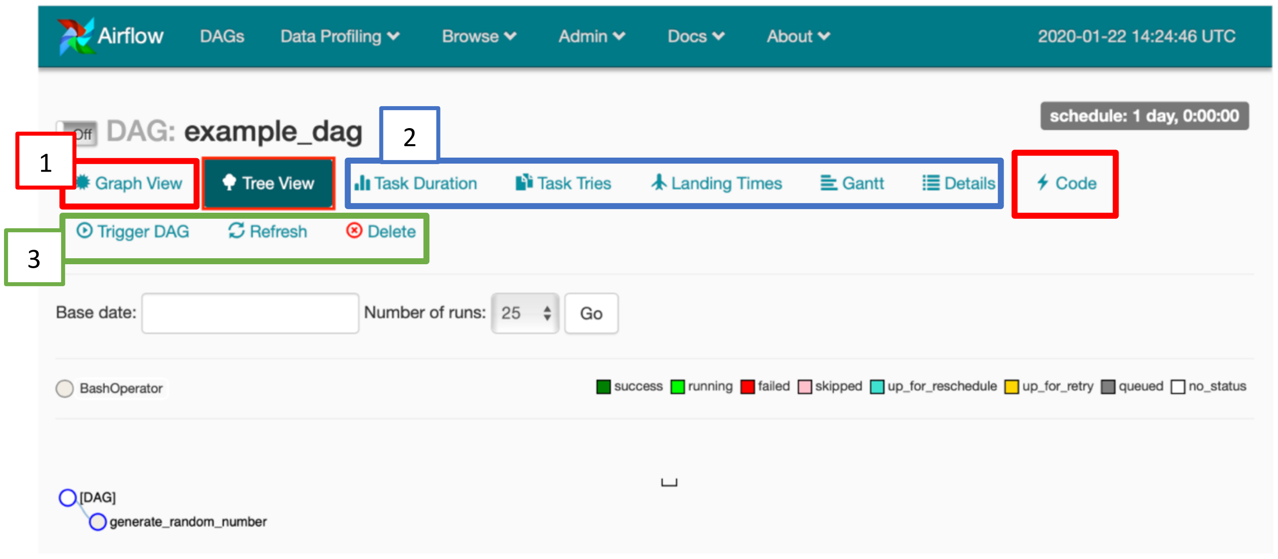


DAGs

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	Off	example_dag	1 day, 0:00:00	airflow				
	Off	update_state	1 day, 0:00:00	airflow				

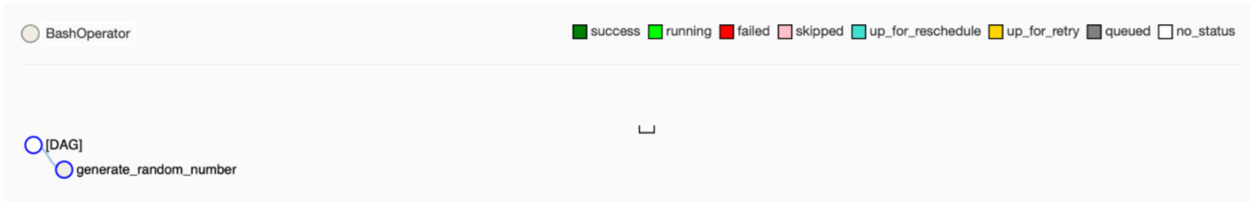
1. DAGs - It provides a quick status of the number of DAGs/workflows available
2. Schedule - It shows us the schedule for the DAG (in date or cron format)
3. Owner - The owner of the DAG
4. Recent Tasks - Which of the most recent tasks have run
5. Last Run - When the last run started
6. Last three – the last three DAG runs
7. Links – The links area on the right gives us quick access to many of the DAG specific views

DAG detail page



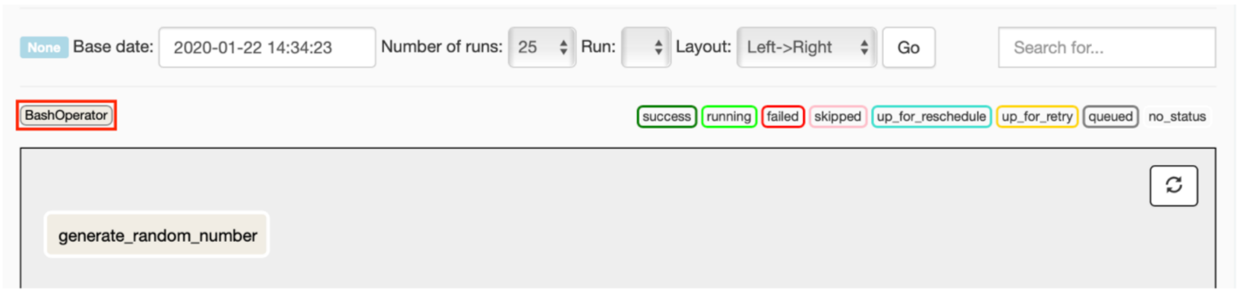
1. The **DAG detail view** gives us specific access to information about the DAG itself. Including several views of information (Graph, Tree, and Code) illustrating the tasks and dependencies in the code.
2. We also get access to the **Task duration**, **task tries**, **timings**, a **Gantt chart view**, and specific **details** about the DAG.
3. We have the ability to **trigger** the DAG (to start), **refresh** our view, and **delete** the DAG if we desire

DAG Tree View



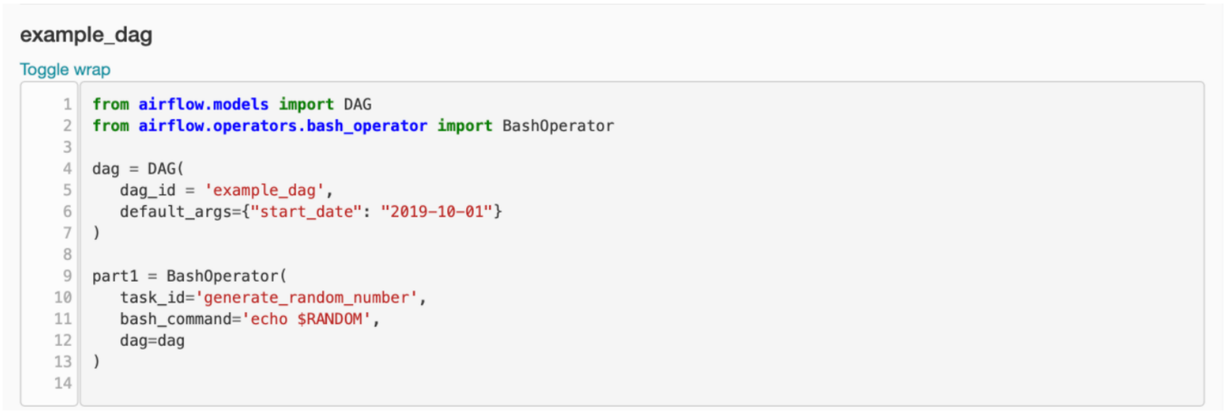
- The detail view defaults to the DAG **Tree View**. It shows the specific name tasks, which operators are in use, and any dependencies between tasks.
- The circles in front of the words represent the state of the task/DAG. For example, in the Tree view above, we have one task called **generate_random_number.**

DAG Graph View



- The DAG **Graph View** arranges the tasks and dependencies in a chart format – this provides another view into the flow of the DAG.
- You can see the operators in use and the state of the tasks at any point in time.
- From the image above, we can see a task called **generate_random_number** that is of the type **BashOperator**

DAG Code View

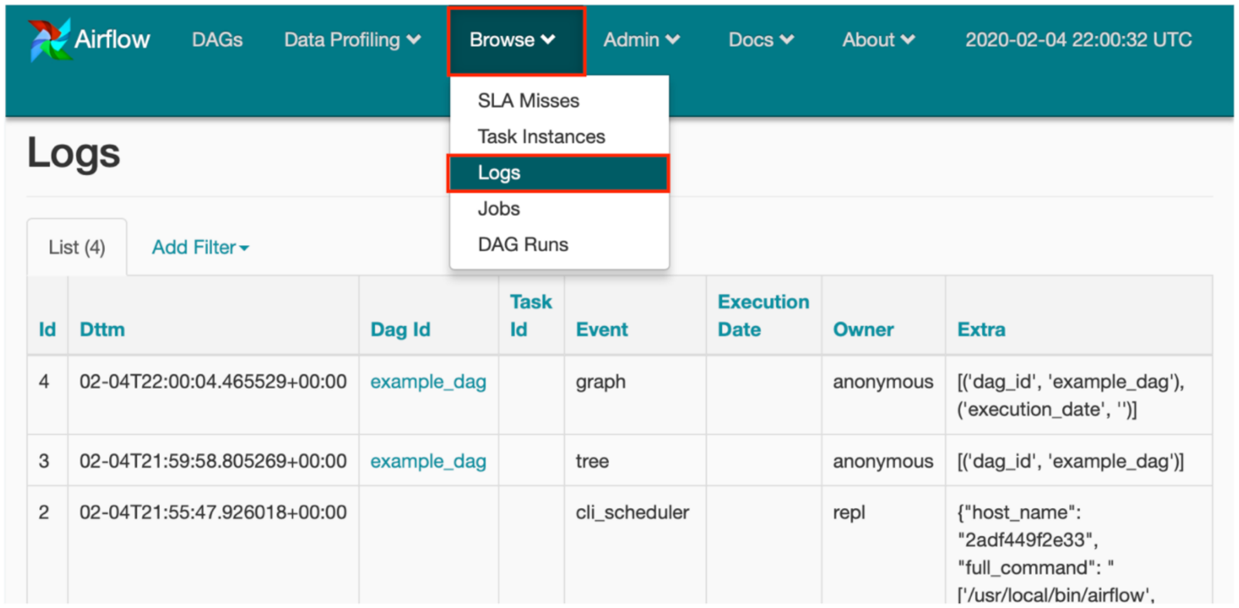

example_dag

Toggle wrap

```
from airflow.models import DAG
from airflow.operators.bash_operator import BashOperator

dag = DAG(
    dag_id = 'example_dag',
    default_args={"start_date": "2019-10-01"}
)

part1 = BashOperator(
    task_id='generate_random_number',
    bash_command='echo $RANDOM',
    dag=dag
)

```

- It provides a copy of the Python code that makes up the DAG.
- The code view provides easy access to exactly what defines the DAG without clicking in various portions of the UI
- The code view is **read-only**
- Any DAG code changes must be done via the actual DAG script

Logs page



Logs

Id	Dttm	Dag Id	Task Id	Event	Execution Date	Owner	Extra
4	02-04T22:00:04.465529+00:00	example_dag		graph		anonymous	[('dag_id', 'example_dag'), ('execution_date', '')]
3	02-04T21:59:58.805269+00:00	example_dag		tree		anonymous	[('dag_id', 'example_dag')]
2	02-04T21:55:47.926018+00:00			cli_scheduler		repl	{"host_name": "2adf449f2e33", "full_command": " ['/usr/local/bin/airflow',

- The **Logs page**, under the **Browse menu option**, provides **troubleshooting** and **audit** ability while using Airflow.
- This view includes **items** such as starting the Airflow webserver, viewing the graph or tree node, creating users, starting DAGs, etc.
-

Command Line vs Web UI

(When to use the Airflow command tool vs WEB UI) – Equally powerful depending on needs

Command line tool:	WEB UI:
• It may be easier to access depending on the settings (via SSH, etc.)	• It is easier to use