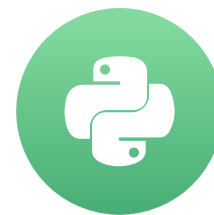


Airflow operators

INTRODUCTION TO AIRFLOW IN PYTHON



Mike Metzger
Data Engineer

Operators

- Represent a single task in a workflow.
- Run independently (usually).
- Generally do not share information.
- Various operators to perform different tasks.

```
DummyOperator(task_id='example', dag=dag)
```

BashOperator

```
BashOperator(  
    task_id='bash_example',  
    bash_command='echo "Example!"',  
    dag=m1_dag)
```

```
BashOperator(  
    task_id='bash_script_example',  
    bash_command='runcleanup.sh',  
    dag=m1_dag)
```

- Executes a given Bash command or script.
- Runs the command in a temporary directory.
- Can specify environment variables for the command.

BashOperator examples

```
from airflow.operators.bash_operator import BashOperator

example_task = BashOperator(task_id='bash_ex',
                             bash_command='echo 1',
                             dag=dag)
```

```
bash_task = BashOperator(task_id='clean_addresses',
                          bash_command='cat addresses.txt | awk "NF==10" > cleaned.txt',
                          dag=dag)
```

Operator gotchas

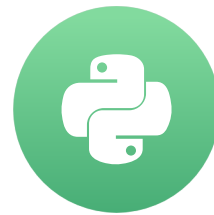
- Not guaranteed to run in the same location / environment.
- May require extensive use of Environment variables.
- Can be difficult to run tasks with elevated privileges.

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Airflow tasks

INTRODUCTION TO AIRFLOW IN PYTHON



Mike Metzger
Data Engineer

Tasks

Tasks are:

- Instances of operators
- Usually assigned to a variable in Python

```
example_task = BashOperator(task_id='bash_example',  
                             bash_command='echo "Example!"',  
                             dag=dag)
```

- Referred to by the task_id within the Airflow tools

Task dependencies

- Define a given order of task completion
- Are not required for a given workflow, but usually present in most
- Are referred to as *upstream* or *downstream* tasks
- In Airflow 1.8 and later, are defined using the *bitshift* operators
 - `>>`, or the upstream operator
 - `<<`, or the downstream operator

Upstream vs Downstream

Upstream means before

Downstream means after

Simple task dependency

```
# Define the tasks
task1 = BashOperator(task_id='first_task',
                      bash_command='echo 1',
                      dag=example_dag)

task2 = BashOperator(task_id='second_task',
                      bash_command='echo 2',
                      dag=example_dag)

# Set first_task to run before second_task
task1 >> task2    # or task2 << task1
```

Task dependencies in the Airflow UI

The screenshot displays the Airflow web interface for a DAG named `simple_dependency`. The top navigation bar includes links for DAGs, Data Profiling, Browse, Admin, Docs, and About, along with the current date and time: 2020-02-12 05:27:33 UTC.

The DAG header shows the name `DAG: simple_dependency` and its schedule: `schedule: 1 day, 0:00:00`. The toolbar provides various views and actions: Graph View (selected), Tree View, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, Trigger DAG, Refresh, and Delete.

The filter bar includes a 'None' button, a 'Base date' field set to 2020-02-12 05:26:17, a 'Number of runs' dropdown set to 25, a 'Run' dropdown, a 'Layout' dropdown set to Left->Right, and a 'Go' button. A search bar is also present.

The task status legend at the top right of the DAG canvas includes: success, running, failed, skipped, up_for_reschedule, up_for_retry, queued, and no_status.

The DAG canvas shows two tasks: `first_task` and `second_task`, connected by a dependency arrow. A refresh button is located in the top right corner of the canvas.

Task dependencies in the Airflow UI

The screenshot displays the Airflow web interface. At the top, a dark teal header contains the Airflow logo, navigation links (DAGs, Data Profiling, Browse, Admin, Docs, About), and the current time (2020-02-12 05:27:33 UTC). Below the header, the main content area shows the DAG 'simple_dependency' with a toggle switch set to 'Off' and a 'schedule: 1 day, 0:00:00' badge. A row of view options includes 'Graph View' (selected), 'Tree View', 'Task Duration', 'Task Tries', 'Landing Times', 'Gantt', 'Details', 'Code', 'Trigger DAG', 'Refresh', and 'Delete'. Below these are filters for 'Base date' (2020-02-12 05:26:17), 'Number of runs' (25), 'Run' (dropdown), 'Layout' (Left->Right), and a 'Go' button. A search bar is also present. The task list shows 'BashOperator' with status filters: success, running, failed, skipped, up_for_reschedule, up_for_retry, queued, and no_status. The task graph area shows two tasks, 'first_task' and 'second_task', connected by a dependency arrow. A red box highlights this dependency. A refresh button is in the top right of the graph area.

Airflow

DAGs Data Profiling Browse Admin Docs About

2020-02-12 05:27:33 UTC

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

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

None Base date: 2020-02-12 05:26:17 Number of runs: 25 Run: Layout: Left->Right Go Search for...

BashOperator success running failed skipped up_for_reschedule up_for_retry queued no_status

first_task

second_task

Task dependencies in the Airflow UI

The screenshot displays the Airflow web interface. At the top, the navigation bar includes the Airflow logo, links to DAGs, Data Profiling, Browse, Admin, Docs, and About, and the current time: 2020-02-12 05:38:17 UTC. Below the navigation bar, the main header shows the DAG name 'DAG: simple_dependency' with a toggle switch set to 'Off' and a schedule of '1 day, 0:00:00'. A row of action buttons follows: Graph View (selected), Tree View, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, Trigger DAG, Refresh, and Delete. Below these buttons, a filter section includes a 'None' button, a 'Base date' input field with the value '2020-02-12 05:37:25', a 'Number of runs' dropdown set to '25', a 'Run' dropdown, a 'Layout' dropdown set to 'Left->Right', and a 'Go' button. To the right of these filters is a 'Search for...' input field. Below the filter section, a 'BashOperator' button is visible on the left, and a row of status buttons (success, running, failed, skipped, up_for_reschedule, up_for_retry, queued, no_status) is on the right. The main content area shows a task dependency graph with two tasks, 'first_task' and 'second_task', connected by an arrow. A red rectangular box highlights this dependency. A refresh button is located in the top right corner of the graph area.

Airflow

DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾

2020-02-12 05:38:17 UTC

☐ Off DAG: simple_dependency schedule: 1 day, 0:00:00

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

None Base date: 2020-02-12 05:37:25 Number of runs: 25 Run: Layout: Left->Right Go Search for...

BashOperator success running failed skipped up_for_reschedule up_for_retry queued no_status

first_task → second_task

Multiple dependencies

Chained dependencies:

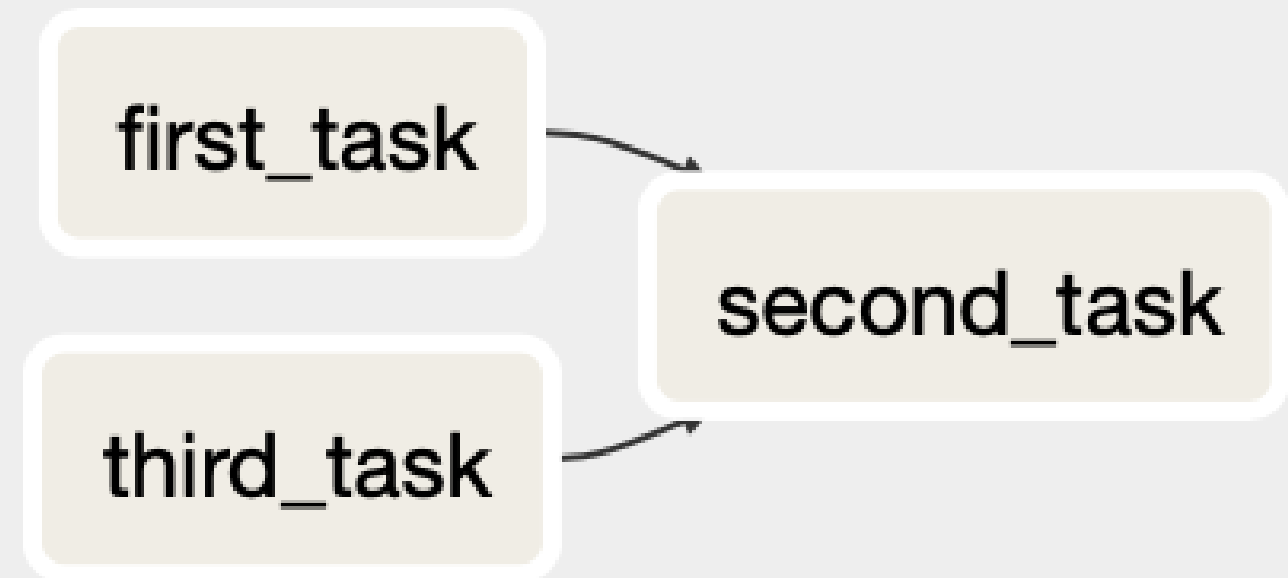
```
task1 >> task2 >> task3 >> task4
```

Mixed dependencies:

```
task1 >> task2 << task3
```

or:

```
task1 >> task2  
task3 >> task2
```



Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Additional operators

INTRODUCTION TO AIRFLOW IN PYTHON



Mike Metzger
Data Engineer

PythonOperator

- Executes a Python function / callable
- Operates similarly to the BashOperator, with more options
- Can pass in arguments to the Python code

```
from airflow.operators.python_operator import PythonOperator

def printme():
    print("This goes in the logs!")

python_task = PythonOperator(
    task_id='simple_print',
    python_callable=printme,
    dag=example_dag
)
```

Arguments

- Supports arguments to tasks
 - Positional
 - Keyword
- Use the `op_kwargs` dictionary

op_kwargs example

```
def sleep(length_of_time):  
    time.sleep(length_of_time)  
  
sleep_task = PythonOperator(  
    task_id='sleep',  
    python_callable=sleep,  
    op_kwargs={'length_of_time': 5}  
    dag=example_dag  
)
```

EmailOperator

- Found in the `airflow.operators` library
- Sends an email
- Can contain typical components
 - HTML content
 - Attachments
- Does require the Airflow system to be configured with email server details

EmailOperator example

```
from airflow.operators.email_operator import EmailOperator
```

```
email_task = EmailOperator(  
    task_id='email_sales_report',  
    to='sales_manager@example.com',  
    subject='Automated Sales Report',  
    html_content='Attached is the latest sales report',  
    files='latest_sales.xlsx',  
    dag=example_dag  
)
```

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Airflow scheduling

INTRODUCTION TO AIRFLOW IN PYTHON




















Mike Metzger
Data Engineer

DAG Runs

















- A specific instance of a workflow at a point in time
- Can be run manually or via `schedule_interval`
- Maintain state for each workflow and the tasks within
 - `running`
 - `failed`
 - `success`

¹ <https://airflow.apache.org/docs/stable/scheduler.html>

DAG Runs view

<div><div> Airflow</div><div>DAGs</div><div>Data Profiling ▾</div><div>Browse ▾</div><div>Admin ▾</div><div>Docs ▾</div><div>About ▾</div></div> <div>2020-02-26 14:31:13 UTC</div>						
Dag Runs						
<div><div>List (12)</div><div>Create</div><div>Add Filter ▾</div><div>With selected ▾</div><div>Search: dag_id, state, run_id</div></div>						
<input type="checkbox"/>		State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		failed	simple_python_operator	02-25T06:39:50.248084+00:00	manual__2020-02-25T06:39:50.248084+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T06:08:08.823092+00:00	manual__2020-02-25T06:08:08.823092+00:00	
<input type="checkbox"/>		failed	simple_python_operator	02-25T06:02:54.809486+00:00	manual__2020-02-25T06:02:54.809486+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:42:19.997484+00:00	manual__2020-02-25T05:42:19.997484+00:00	
<input type="checkbox"/>		failed	simple_python_operator	02-25T05:39:23.812012+00:00	manual__2020-02-25T05:39:23.812012+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:29:00.355729+00:00	manual__2020-02-25T05:29:00.355729+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:25:40.655538+00:00	manual__2020-02-25T05:25:40.655538+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:21:10.765962+00:00	manual__2020-02-25T05:21:10.765962+00:00	

DAG Runs state

Airflow DAGs Data Profiling Browse Admin Docs About 2020-02-26 14:31:13 UTC						
Dag Runs						
List (12) Create Add Filter With selected Search: dag_id, state, run_id						
		State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		failed	simple_python_operator	02-25T06:39:50.248084+00:00	manual__2020-02-25T06:39:50.248084+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T06:08:08.823092+00:00	manual__2020-02-25T06:08:08.823092+00:00	
<input type="checkbox"/>		failed	simple_python_operator	02-25T06:02:54.809486+00:00	manual__2020-02-25T06:02:54.809486+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:42:19.997484+00:00	manual__2020-02-25T05:42:19.997484+00:00	
<input type="checkbox"/>		failed	simple_python_operator	02-25T05:39:23.812012+00:00	manual__2020-02-25T05:39:23.812012+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:29:00.355729+00:00	manual__2020-02-25T05:29:00.355729+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:25:40.655538+00:00	manual__2020-02-25T05:25:40.655538+00:00	
<input type="checkbox"/>		success	simple_python_operator	02-25T05:21:10.765962+00:00	manual__2020-02-25T05:21:10.765962+00:00	

Schedule details

When scheduling a DAG, there are several attributes of note:

- `start_date` - The date / time to initially schedule the DAG run
- `end_date` - Optional attribute for when to stop running new DAG instances
- `max_tries` - Optional attribute for how many attempts to make
- `schedule_interval` - How often to run

Schedule interval

`schedule_interval` represents:

- How often to schedule the DAG
- Between the `start_date` and `end_date`
- Can be defined via `cron` style syntax or via built-in presets.

cron syntax

```
# |----- minute (0 - 59)
# | |----- hour (0 - 23)
# | | |----- day of the month (1 - 31)
# | | | |----- month (1 - 12)
# | | | | |----- day of the week (0 - 6) (Sunday to Saturday;
# | | | | |                                     7 is also Sunday on some systems)
# | | | | |
# | | | | |
# * * * * * command to execute
```

- Is pulled from the Unix cron format
- Consists of 5 fields separated by a space
- An asterisk `*` represents running for every interval (ie, every minute, every day, etc)
- Can be comma separated values in fields for a list of values

cron examples

```
0 12 * * *          # Run daily at noon
```

```
* * 25 2 *          # Run once per minute on February 25
```

```
0,15,30,45 * * * *  # Run every 15 minutes
```

Airflow scheduler presets

Preset:

- @hourly
- @daily
- @weekly
- @monthly
- @yearly

cron equivalent:

- 0 * * * *
- 0 0 * * *
- 0 0 * * 0
- 0 0 1 * *
- 0 0 1 1 *

¹ <https://airflow.apache.org/docs/stable/scheduler.html>

Special presets

Airflow has two special `schedule_interval` presets:

- `None` - Don't schedule ever, used for manually triggered DAGs
- `@once` - Schedule only once

schedule_interval issues

When scheduling a DAG, Airflow will:

- Use the `start_date` as the earliest possible value
- Schedule the task at `start_date` + `schedule_interval`

```
'start_date': datetime(2020, 2, 25),  
'schedule_interval': @daily
```

This means the earliest starting time to run the DAG is on February 26th, 2020

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON