

# 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=ml_dag)
```

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

 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

 Airflow

DAGsData 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

 Airflow

DAGsData 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:

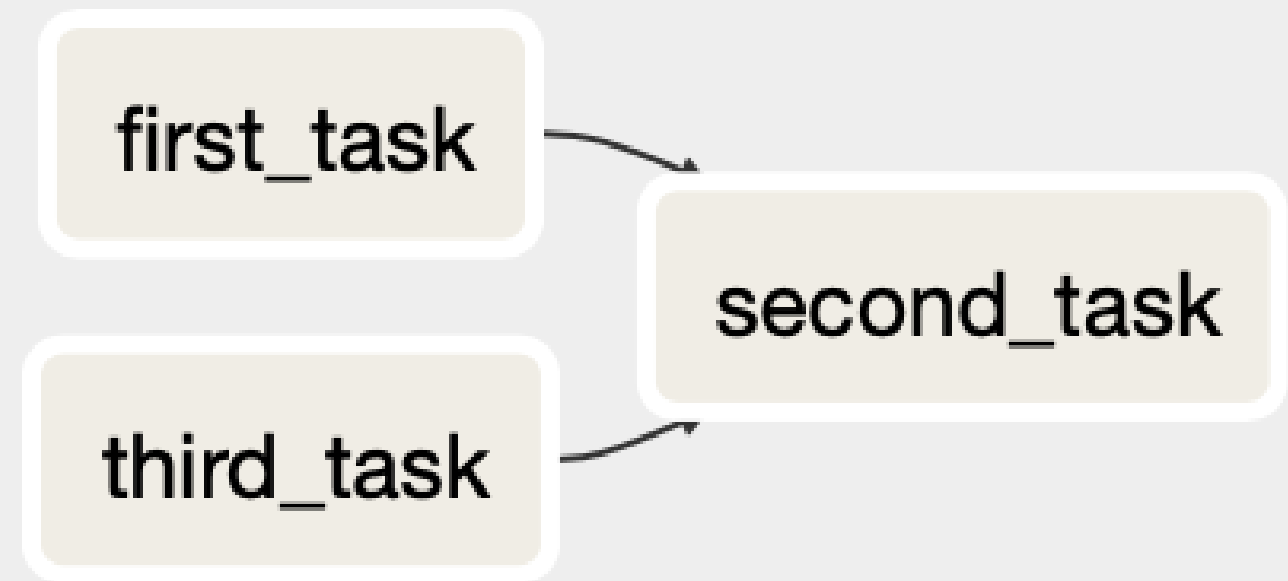
```
task1 >> task2 >> task3 >> task4
```

Mixed dependencies:

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

or:

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



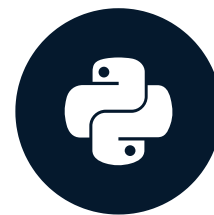
# Let's practice!

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

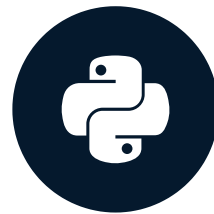
email_task = EmailOperator(
    task_id='email_sales_report',
    to='sales_manager@example.com',
    subject='Automated Sales Report',
    html_content='Attached is the latest sales report',
    files='latest_sales.xlsx',
    dag=example_dag
)
```

# 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`

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

# DAG Runs view

 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

<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 \*

<sup>1</sup> <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