

Hands-on Lab: Create a DAG for Apache Airflow with PythonOperator



Estimated time needed: **40** minutes

Introduction

In this lab, you will explore the Apache Airflow web user interface (UI). You will then create a Direct Acyclic Graph (DAG) using PythonOperator and finally run it through the Airflow web UI.

Objectives

After completing this lab, you will be able to:

- Explore the Airflow Web UI
- Create a DAG with PythonOperator
- Submit a DAG and run it through the Web UI

Prerequisite

Please ensure that you have completed the reading on the [Airflow DAG Operators](#) before proceeding with this lab. You should be familiar with Python input and output (I/O) operations and request packages to complete this lab.

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs for course and project-related labs. Theia is an open-source IDE (Integrated Development Environment) that can be run on a desktop or on the cloud. To complete this lab, you will be using the Cloud IDE based on Theia, running in a Docker container.

Important notice about this lab environment

Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session will get lost. To avoid losing your data, please plan to complete these labs in a single session.

Exercise 1: Start Apache Airflow

1. Click on **Skills Network Toolbox**.
2. From the **BIG DATA** section, click **Apache Airflow**.
3. Click **Start** to start the Apache Airflow.

The screenshot shows the Skills Network Terminal interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar contains a list of categories: DATABASES, BIG DATA, CLOUD, EMBEDDABLE AI, and OTHER. Under BIG DATA, 'Apache Airflow' is listed as 'INACTIVE'. A red box highlights the 'Launch Application' button at the bottom of the sidebar. The main panel displays the 'Apache Airflow' status as 'INACTIVE' with a 'Start' button highlighted. Below the status bar, there are tabs for 'Summary' and 'Connection Information'. The 'Summary' tab is active, showing the text 'Get started with Apache Airflow in a faster, e'.

Note: Please be patient, it will take a few minutes for Airflow to start. If there is an error starting Airflow, please restart it.

Exercise 2: Open the Airflow Web UI

1. When Airflow starts successfully, you should see an output similar to the one below. Once **Apache Airflow** has started, click on the highlighted icon to open **Apache Airflow Web UI** in the new window.

Apache Airflow

ACTIVE

2.9.1 | 2.9.1 | 2.9.1

Connect to Apache Airflow directly in your Skills Network Labs environment.

Start

Stop

Summary Connection Information Details

Your Apache Airflow Services are now ready to use and available with the following login credentials. For more details on how to navigate Apache Airflow, please check out the Details section.


Username: airflow

Password: MzE3NjUtOGF2YW55

You can manage Apache Airflow via:

Airflow Webserver

You should land on a page that looks like this.



DAGs

Cluster Activity

Datasets

Security

Browse

Admin

Docs

03:53 UTC

Log In

<div><div></div><div>example_bash_operator</div></div> <div><div>exampleexample2</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>00:00:00</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_datetime_operator</div></div> <div><div>example</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_datetime_operator_2</div></div> <div><div>example</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_datetime_operator_3</div></div> <div><div>example</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_dop_operator_v3</div></div> <div><div>example</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>*1:00:00</div><div></div></div> <div><div>2024-05-30, 03:51:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_labels</div></div> <div><div></div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_operator</div></div> <div><div>exampleexample2</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>
<div><div></div><div>example_branch_python_operator_decorator</div></div> <div><div>exampleexample2</div><div>airflow</div><div><div></div><div></div><div></div><div></div></div><div>@daily</div><div></div></div> <div><div>2024-05-29, 00:00:00</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>

Exercise 3: Create a DAG with PythonOperator

Next, you will create a DAG, which will define a pipeline of tasks, such as extract, transform, load, and check with PythonOperator.

1. Create a DAG file, `my_first_dag.py`, which will run daily. The `my_first_dag.py` file defines tasks `execute_extract`, `execute_transform`, `execute_load`, and `execute_check` to call the respective Python functions.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
```

25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
70. 70
71. 71
72. 72
73. 73
74. 74
75. 75
76. 76
77. 77
78. 78
79. 79
80. 80
81. 81
82. 82
83. 83
84. 84
85. 85
86. 86
87. 87
88. 88
89. 89
90. 90
91. 91
92. 92
93. 93
94. 94
95. 95
96. 96
97. 97
98. 98
99. 99
100. 100
101. 101
102. 102
103. 103
104. 104
105. 105
106. 106
107. 107
108. 108

```
1. # Import the libraries
2. from datetime import timedelta
3. # The DAG object; we'll need this to instantiate a DAG
4. from airflow.models import DAG
5. # Operators; you need this to write tasks!
6. from airflow.operators.python import PythonOperator
7.
8. # This makes scheduling easy
9. from airflow.utils.dates import days_ago
10.
11. # Define the path for the input and output files
12. input_file = '/etc/passwd'
13. extracted_file = 'extracted-data.txt'
14. transformed_file = 'transformed.txt'
15. output_file = 'data_for_analytics.csv'
16.
17.
18. def extract():
19.     global input_file
```

```

20.     print("Inside Extract")
21.     # Read the contents of the file into a string
22.     with open(input_file, 'r') as infile, \
23.         open(extracted_file, 'w') as outfile:
24.         for line in infile:
25.             fields = line.split(':')
26.             if len(fields) >= 6:
27.                 field_1 = fields[0]
28.                 field_3 = fields[2]
29.                 field_6 = fields[5]
30.                 outfile.write(field_1 + ":" + field_3 + ":" + field_6 + "\n")
31.
32.
33. def transform():
34.     global extracted_file, transformed_file
35.     print("Inside Transform")
36.     with open(extracted_file, 'r') as infile, \
37.         open(transformed_file, 'w') as outfile:
38.         for line in infile:
39.             processed_line = line.replace(':', ',')
40.             outfile.write(processed_line + '\n')
41.
42.
43. def load():
44.     global transformed_file, output_file
45.     print("Inside Load")
46.     # Save the array to a CSV file
47.     with open(transformed_file, 'r') as infile, \
48.         open(output_file, 'w') as outfile:
49.         for line in infile:
50.             outfile.write(line + '\n')
51.
52.
53. def check():
54.     global output_file
55.     print("Inside Check")
56.     # Save the array to a CSV file
57.     with open(output_file, 'r') as infile:
58.         for line in infile:
59.             print(line)
60.
61.
62. # You can override them on a per-task basis during operator initialization
63. default_args = {
64.     'owner': 'Your name',
65.     'start_date': days_ago(0),
66.     'email': ['your email'],
67.     'retries': 1,
68.     'retry_delay': timedelta(minutes=5),
69. }
70.
71. # Define the DAG
72. dag = DAG(
73.     'my-first-python-etl-dag',
74.     default_args=default_args,
75.     description='My first DAG',
76.     schedule_interval=timedelta(days=1),
77. )
78.
79. # Define the task named execute_extract to call the `extract` function
80. execute_extract = PythonOperator(
81.     task_id='extract',
82.     python_callable=extract,
83.     dag=dag,
84. )
85.
86. # Define the task named execute_transform to call the `transform` function
87. execute_transform = PythonOperator(
88.     task_id='transform',
89.     python_callable=transform,
90.     dag=dag,
91. )
92.
93. # Define the task named execute_load to call the `load` function
94. execute_load = PythonOperator(
95.     task_id='load',
96.     python_callable=load,
97.     dag=dag,
98. )
99.
100. # Define the task named execute_check to call the `check` function
101. execute_check = PythonOperator(
102.     task_id='check',
103.     python_callable=check,
104.     dag=dag,
105. )
106.
107. # Task pipeline
108. execute_extract >> execute_transform >> execute_load >> execute_check

```

Copied!

Exercise 4: Submit a DAG

Submitting a DAG is as simple as copying the DAG Python file into the `dags` folder in the `AIRFLOW_HOME` directory.

1. Open a terminal and run the command below to set the `AIRFLOW_HOME`.

```

1. 1
2. 2
1. export AIRFLOW_HOME=/home/project/airflow

```

```
2. echo $AIRFLOW_HOME
```

Copied!

```
theia@theiadocker-lavanyas: /home/project x
theia@theiadocker-lavanyas: /home/project$ echo $AIRFLOW_HOME
/home/project/airflow
```

2. Run the command below to submit the DAG that was created in the previous exercise.

```
1. 1
1. cp my_first_dag.py $AIRFLOW_HOME/dags
```

Copied!

3. Verify that your DAG actually got submitted.

4. Run the command below to list out all the existing DAGs.

```
1. 1
1. airflow dags list
```

Copied!

5. Verify that my-first-python-etl-dag is a part of the output.

```
1. 1
1. airflow dags list|grep "my-first-python-etl-dag"
```

Copied!

6. You should see your DAG name in the output.

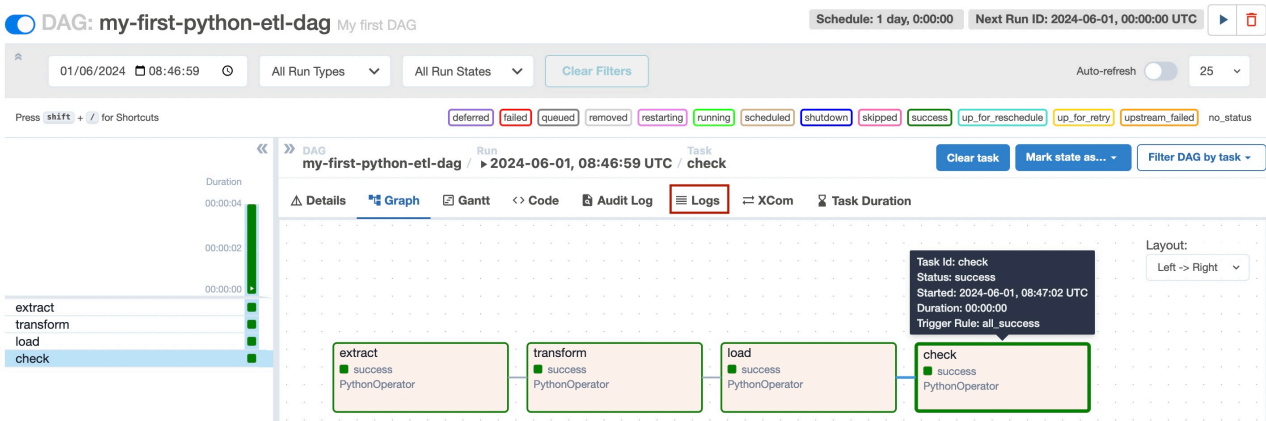
7. Run the command below to list out all the tasks in my-first-python-etl-dag.

```
1. 1
1. airflow tasks list my-first-python-etl-dag
```

Copied!

8. You should see all the four tasks in the output.

9. You can run the task from the Web UI. You can check the logs of the tasks by clicking the individual task in the Graph view.



Practice exercise

Write a DAG named `ETL_Server_Access_Log_Processing` that will extract a file from a remote server and then transform the content and load it into a file.

The file URL is given below:

<https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Apache%20Airflow/Build%20a%20DAG%20using%20Airflow/web-server-access-log.txt>

The server access log file contains these fields.

- timestamp - TIMESTAMP
- latitude - float
- longitude - float
- visitorid - char(37)
- accessed_from_mobile - boolean
- browser_code - int

Tasks

1. Add tasks in the DAG file to download the file, read the file, and extract the fields `timestamp` and `visitorid` from the `web-server-access-log.txt`.

2. Capitalize the `visitorid` for all the records and store it in a local variable.
3. Load the data into a new file `capitalized.txt`.
4. Create the imports block.
5. Create the DAG Arguments block. You can use the default settings.
6. Create the DAG definition block. The DAG should run daily.
7. Create the tasks `extract`, `transform`, and `load` to call the Python script.
8. Create the task pipeline block.
9. Submit the DAG.
10. Verify if the DAG is submitted.

- ▶ [Click here for hint.](#)
- ▶ [Click here for the solution.](#)

Authors

[Lavanya T S](#)

Other Contributors

Rav Ahuja

© IBM Corporation. All rights reserved.