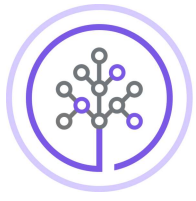


Hands-on Lab: Create a DAG for Apache Airflow



Skills
Network

Estimated time needed: **40** minutes

Objectives

After completing this lab you will be able to:

- Explore the anatomy of a DAG.
- Create a DAG.
- Submit a DAG.

About Skills Network Cloud IDE

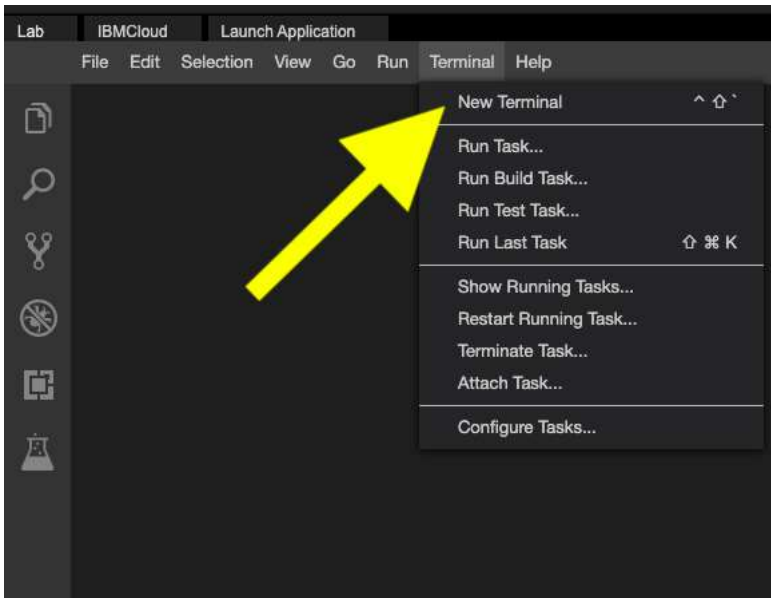
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment), that can be run on desktop or on the cloud. to complete this lab, we will be using the Cloud IDE based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

Important Notice about this lab environment

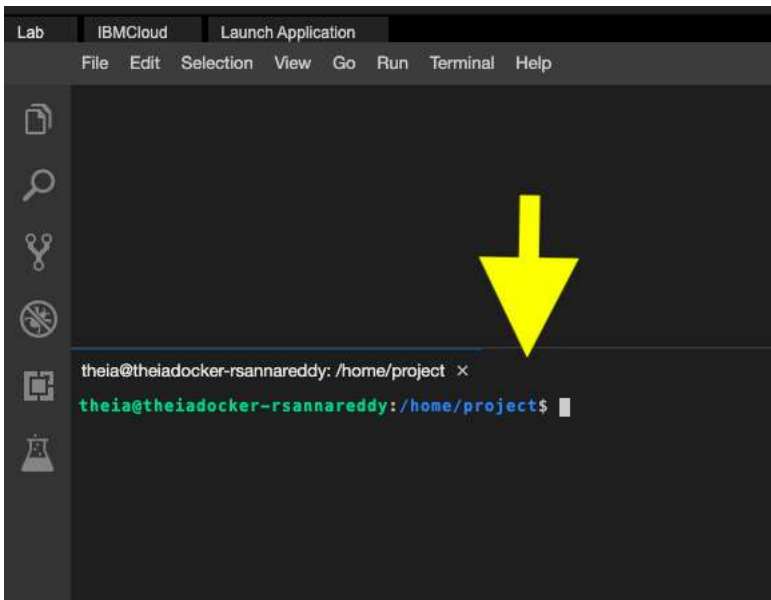
Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session will get lost. To avoid losing your data, please plan to complete these labs in a single session.

Exercise 1 - Start Apache Airflow

Open a new terminal by clicking on the menu bar and selecting **Terminal->New Terminal**, as shown in the image below.



This will open a new terminal at the bottom of the screen as in the image below.



Run the commands below on the newly opened terminal. (You can copy the code by clicking on the little copy button on the bottom right of the codeblock and then paste it wherever you wish.)

Start Apache Airflow in the lab environment.

- 1
1. start_airflow

Copied!

Please be patient, it will take a few minutes for airflow to get started.

When airflow starts successfully, you should see an output similar to the one below.

```
theia@theiadocker-rsannareddy:/home/project$ start_airflow
Starting your airflow services....
This process can take a few minutes.

Airflow started, waiting for all services to be ready....

Your airflow server is now ready to use and available with username: airflow password: MTM40DUtcNhhbm5h

You can access your Airflow Webserver at: https://rsannareddy-8080.theiadocker-5-labs-prod-th-eiak8s-4-tor01.proxy.cognitiveclass.ai

CommandLine:
• List DAGs: airflow dags list
• List Tasks: airflow tasks list example_bash_operator
• Run an example task: airflow tasks test example_bash_operator runme_1 2015-06-01
theia@theiadocker-rsannareddy:/home/project$
```

Exercise 2 - Open the Airflow Web UI

Copy the Web-UI URL and paste it on a new browser tab. Or you can click on the URL by holding the control key (Command key in case of a Mac).

You should land at a page that looks like this.

Skills Network Airflow

| <div> <div>All 32</div> <div>Active 0</div> <div>Paused 32</div> <div>Filter DAGs by tag</div> </div> | | | | | | |
|---|---------|-------------------------------------|---------------|-------------------------------------|---|--|
| DAG | Owner | Runs | Schedule | Last Run | Recent Tasks | |
| <div>example_bash_operator</div> <div>example example2</div> | airflow | <div></div> <div></div> <div></div> | 0 0 *** | <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>example</div> | airflow | <div></div> <div></div> <div></div> | @daily | <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>example</div> | airflow | <div></div> <div></div> <div></div> | * / 1 * * * * | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |
| <div>example_branch_labels</div> | airflow | <div></div> <div></div> <div></div> | @daily | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |
| <div>example_branch_operator</div> <div>example example2</div> | airflow | <div></div> <div></div> <div></div> | @daily | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |
| <div>example_complex</div> <div>example example2 example3</div> | airflow | <div></div> <div></div> <div></div> | None | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |
| <div>example_dag_decorator</div> <div>example</div> | airflow | <div></div> <div></div> <div></div> | None | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |
| <div>example_external_task_marker_child</div> <div>example2</div> | airflow | <div></div> <div></div> <div></div> | None | <div></div> <div></div> <div></div> | <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> | |

Exercise 3 - Explore the anatomy of a DAG

An Apache Airflow DAG is a python program. It consists of these logical blocks.

- Imports
- DAG Arguments
- DAG Definition
- Task Definitions
- Task Pipeline

A typical imports block looks like this.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9

1. # import the libraries
2.
3. from datetime import timedelta
4. # The DAG object; we'll need this to instantiate a DAG
5. from airflow import DAG
6. # Operators; we need this to write tasks!
7. from airflow.operators.bash_operator import BashOperator
8. # This makes scheduling easy
9. from airflow.utils.dates import days_ago
```

Copied!

A typical DAG Arguments block looks like this.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
```

```

11. 11
12. 12

1. #defining DAG arguments
2.
3. # You can override them on a per-task basis during operator initialization
4. default_args = {
5.     'owner': 'Ramesh Sannareddy',
6.     'start_date': days_ago(0),
7.     'email': ['ramesh@somemail.com'],
8.     'email_on_failure': True,
9.     'email_on_retry': True,
10.    'retries': 1,
11.    'retry_delay': timedelta(minutes=5),
12. }

```

Copied!

DAG arguments are like settings for the DAG.

The above settings mention

- the owner name,
- when this DAG should run from: `days_ago(0)` means today,
- the email address where the alerts are sent to,
- whether alert must be sent on failure,
- whether alert must be sent on retry,
- the number of retries in case of failure, and
- the time delay between retries.

A typical DAG definition block looks like this.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7

1. # define the DAG
2. dag = DAG(
3.     dag_id='sample-etl-dag',
4.     default_args=default_args,
5.     description='Sample ETL DAG using Bash',
6.     schedule_interval=timedelta(days=1),
7. )

```

Copied!

Here we are creating a variable named `dag` by instantiating the DAG class with the following parameters.

`sample-etl-dag` is the ID of the DAG. This is what you see on the web console.

We are passing the dictionary `default_args`, in which all the defaults are defined.

`description` helps us in understanding what this DAG does.

`schedule_interval` tells us how frequently this DAG runs. In this case every day. (`days=1`).

A typical task definitions block looks like this:

```

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

1. # define the tasks
2.
3. # define the first task named extract
4. extract = BashOperator(
5.     task_id='extract',
6.     bash_command='echo "extract"',
7.     dag=dag,

```

```

8. )
9.
10. # define the second task named transform
11. transform = BashOperator(
12.     task_id='transform',
13.     bash_command='echo "transform"',
14.     dag=dag,
15. )
16.
17. # define the third task named load
18.
19. load = BashOperator(
20.     task_id='load',
21.     bash_command='echo "load"',
22.     dag=dag,
23. )

```

Copied!

A task is defined using:

- A `task_id` which is a string and helps in identifying the task.
- What bash command it represents.
- Which dag this task belongs to.

A typical task pipeline block looks like this:

```

1. 1
2. 2

1. # task pipeline
2. extract >> transform >> load

```

Copied!

Task pipeline helps us to organize the order of tasks.

Here the task `extract` must run first, followed by `transform`, followed by the task `load`.

Exercise 4 - Create a DAG

Let us create a DAG that runs daily, and extracts user information from `/etc/passwd` file, transforms it, and loads it into a file.

This DAG has two tasks `extract` that extracts fields from `/etc/passwd` file and `transform_and_load` that transforms and loads data into a file.

```

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

1. # import the libraries
2.
3. from datetime import timedelta
4. # The DAG object; we'll need this to instantiate a DAG
5. from airflow import DAG
6. # Operators; we need this to write tasks!
7. from airflow.operators.bash_operator import BashOperator
8. # This makes scheduling easy
9. from airflow.utils.dates import days_ago
10.
11. #defining DAG arguments
12.
13. # You can override them on a per-task basis during operator initialization
14. default_args = {
15.     'owner': 'Ramesh Sannareddy',
16.     'start_date': days_ago(0),
17.     'email': ['ramesh@somemail.com'],
18.     'email_on_failure': False,
19.     'email_on_retry': False,
20.     'retries': 1,
21.     'retry_delay': timedelta(minutes=5),
22. }
23.
24. # defining the DAG
25.
26. # define the DAG
27. dag = DAG(
28.     'my-first-dag',
29.     default_args=default_args,
30.     description='My first DAG',
31.     schedule_interval=timedelta(days=1),
32. )
33.
34. # define the tasks
35.
36. # define the first task
37.
38. extract = BashOperator(
39.     task_id='extract',
40.     bash_command='cut -d":" -f1,3,6 /etc/passwd > /home/project/airflow/dags/extracted-data.txt',
41.     dag=dag,
42. )
43.
44. # define the second task
45. transform_and_load = BashOperator(
46.     task_id='transform',
47.     bash_command='tr ":" " ," < /home/project/airflow/dags/extracted-data.txt > /home/project/airflow/dags/transformed-data.csv',
48.     dag=dag,
49. )
50.
51. # task pipeline
52. extract >> transform_and_load

```

Copied!

Create a new file by choosing File->New File and name it `my_first_dag.py`. Copy the code above and paste it into `my_first_dag.py`.

Exercise 5 - Submit a DAG

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

Open a terminal and run the command below to submit the DAG that was created in the previous exercise.

Note: While submitting the dag that was created in the previous exercise, use **sudo** in the terminal before the command used to submit the dag.

```

1. 1

1. cp my_first_dag.py $AIRFLOW_HOME/dags

```

Copied!

Verify that our DAG actually got submitted.

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

```

1. 1

1. airflow dags list

```

Copied!

Verify that `my-first-dag` is a part of the output.

```

1. 1

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

```

Copied!

You should see your DAG name in the output.

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

- 1
1. airflow tasks list my-first-dag

Copied!

You should see 2 tasks in the output.

Practice exercises

1. Problem:

Write a DAG named ETL_Server_Access_Log_Processing.

Task 1: Create the imports block.

Task 2: Create the DAG Arguments block. You can use the default settings

Task 3: Create the DAG definition block. The DAG should run daily.

Task 4: Create the download task.

download task must download the server access log file which is available at the URL: <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Python/Airflow/Build%20a%20DAG%20using%20Airflow/web-server-access-log.txt>

Task 5: Create the extract task.

The server access log file contains these fields.

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

The extract task must extract the fields timestamp and visitorid.

Task 6: Create the transform task.

The transform task must capitalize the visitorid.

Task 7: Create the load task.

The load task must compress the extracted and transformed data.

Task 8: Create the task pipeline block.

The pipeline block should schedule the task in the order listed below:

1. download
2. extract
3. transform
4. load

Task 10: Submit the DAG.

Task 11. Verify if the DAG is submitted

- Click here for Hint
- Click here for Solution

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

| Date (YYYY-MM-DD) | Version | Changed By | Change Description |
|-------------------|---------|---------------------|------------------------------------|
| 2022-11-10 | 0.5 | Appalabhaktula Hema | Updated instruction |
| 2022-08-22 | 0.4 | Lakshmi Holla | updated bash command |
| 2022-07-29 | 0.3 | Lakshmi Holla | changed dag name |
| 2022-06-28 | 0.2 | Lakshmi Holla | updated DAG path |
| 2021-07-05 | 0.1 | Ramesh Sannareddy | Created initial version of the lab |

Copyright (c) 2021 IBM Corporation. All rights reserved.