Hands-on Lab: Create a DAG for Apache Airflow

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

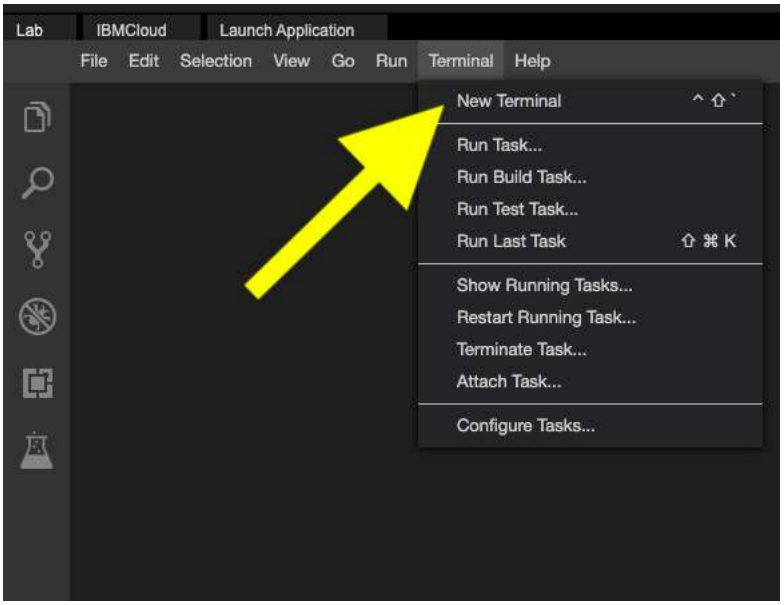
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment), that can be run on desktop or on the cloud. to complete this lab, we will be using the Cloud IDE based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

Important Notice about this lab environment

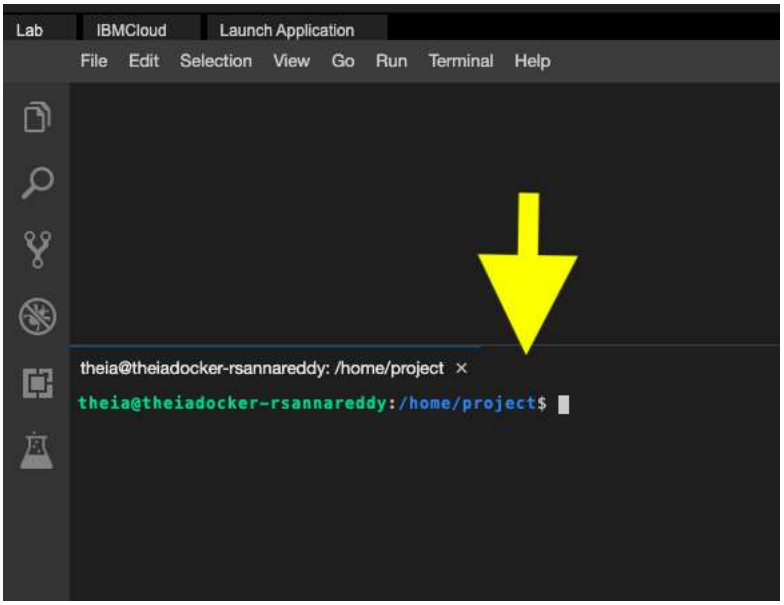
Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session will get lost. To avoid losing your data, please plan to complete these labs in a single session.

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.



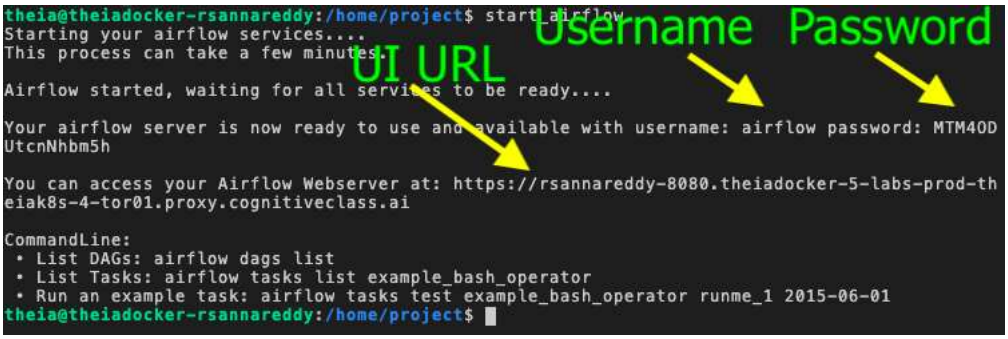
Run the commands below on the newly opened terminal. (You can copy the code by clicking on the little copy button on the bottom right of the codeblock and then paste it wherever you wish.)

Start Apache Airflow in the lab environment.

```
start_airflow
```

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

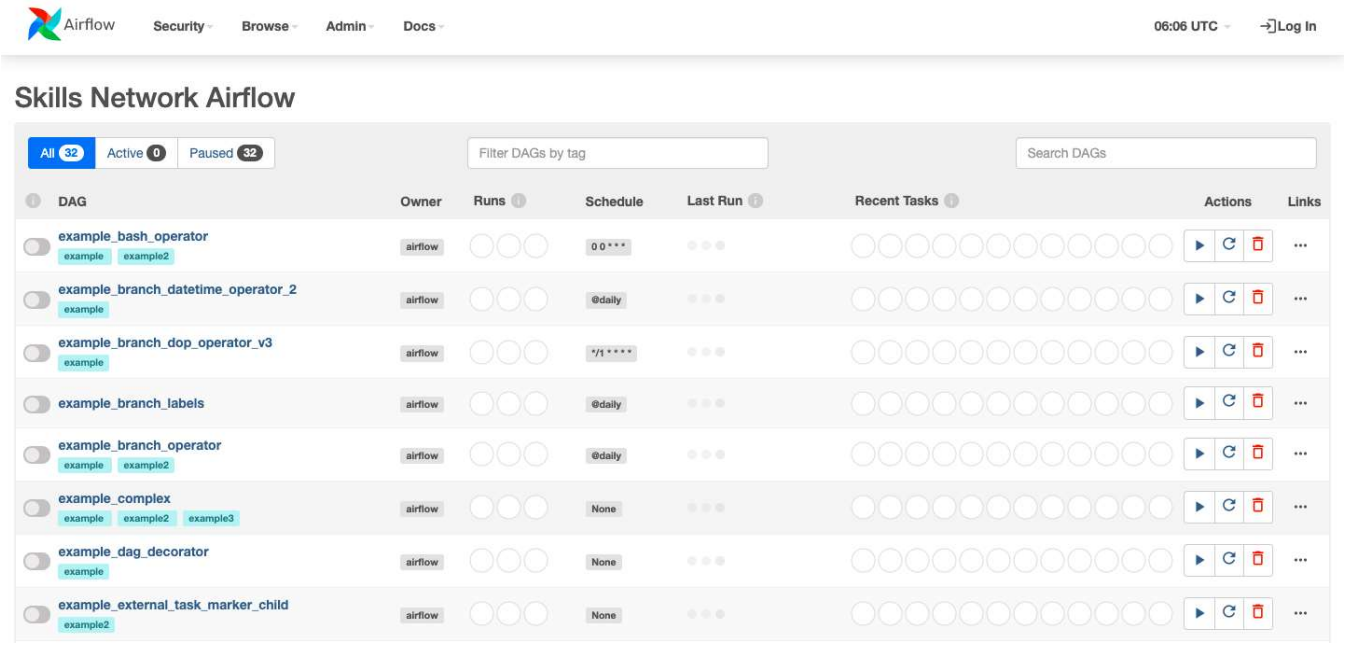
When airflow starts successfully, you should see an output similar to the one below.



Exercise 2 - Open the Airflow Web UI

Copy the Web-UI URL and paste it on a new browser tab. Or your 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.



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.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
```

A typical `DAG Arguments` block looks like this.

```
#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': True,
    'email_on_retry': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

DAG arguments are like settings for the DAG.

The above settings mention

- the owner name,
- when this DAG should run from: days_age(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.

```
# define the DAG
dag = DAG(
    dag_id='sample-etl-dag',
    default_args=default_args,
    description='Sample ETL DAG using Bash',
    schedule_interval=timedelta(days=1),
)
```

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:

```
# define the tasks

# define the first task named extract
extract = BashOperator(
    task_id='extract',
    bash_command='echo "extract"',
    dag=dag,
)


# define the second task named transform
transform = BashOperator(
    task_id='transform',
    bash_command='echo "transform"',
    dag=dag,
)

# define the third task named load

load = BashOperator(
    task_id='load',
    bash_command='echo "load"',
    dag=dag,
)
```

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:

```
# task pipeline
extract >> transform >> load
```

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.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago

#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG

# define the DAG
dag = DAG(
    'my-first-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)

# define the tasks

# define the first task

extract = BashOperator(
    task_id='extract',
    bash_command='cut -d":" -f1,3,6 /etc/passwd > extracted-data.txt',
    dag=dag,
)


# define the second task
transform_and_load = BashOperator(
    task_id='transform',
    bash_command='tr ":" "," < extracted-data.txt > transformed-data.csv',
    dag=dag,
)


# task pipeline
extract >> transform_and_load
```

Copy the code above and save it into a file named `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.

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

Verify that our DAG actually got submitted.

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

```
airflow dags list
```

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

```
airflow dags list|grep "my-first-dag"
```

You should see your DAG name in the output.

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

```
airflow tasks list my-first-dag
```

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/Apache%20Airflow/Build%20a%20DAG%20using%20Airflow/web-server-access-log.txt

***Task 5**: Create the extract task.*
The server access log file contains these fields.

a. `timestamp` - TIMESTAMP
b. `latitude` - float
c. `longitude` - float
d. `visitorid` - char(37)
e. `accessed_from_mobile` - boolean
f. `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
Select File -> New File from the menu and name it as `ETL_Server_Access_Log_Processing.py`.

Add to the file the following parts of code to complete the tasks given in the problem.

Task 1: Create the imports block.

```
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
```

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

```
#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

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

```
# defining the DAG

# define the DAG
dag = DAG(
    'etl-log-processsing-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)
```

Task 4: Create the download task.

```
# define the tasks

# define the task 'download'

download = BashOperator(
    task_id='download',
    bash_command='wget "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"',
    dag=dag,
)
```

Task 5: Create the extract task.

The extract task must extract the fields `timestamp` and `visitorid`.

```
# define the task 'extract'

extract = BashOperator(
    task_id='extract',
    bash_command='cut -f1,4 -d"#" web-server-access-log.txt > extracted.txt',
    dag=dag,
)
```

Task 6: Create the transform task.

The transform task must capitalize the `visitorid`.

```
# define the task 'transform'

transform = BashOperator(
    task_id='transform',
    bash_command='tr "[a-z]" "[A-Z]" < extracted.txt > capitalized.txt',
    dag=dag,
)
```

Task 7: Create the load task.

The `load` task must compress the extracted and transformed data.

```
# define the task 'load'

load = BashOperator(
    task_id='load',
    bash_command='zip log.zip capitalized.txt' ,
    dag=dag,
)
```

Task 8: Create the task pipeline block.

```
# task pipeline

download >> extract >> transform >> load
```

Task 9: Submit the DAG.

```
cp  ETL_Server_Access_Log_Processing.py $AIRFLOW_HOME/dags
```

Task 10: Verify if the DAG is submitted.

```
airflow dags list
```

Verify that the DAG `etl-log-processsing-dag` is listed.

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-07-05	0.1	Ramesh Sannareddy	Created initial version of the lab