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

Skills Network

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

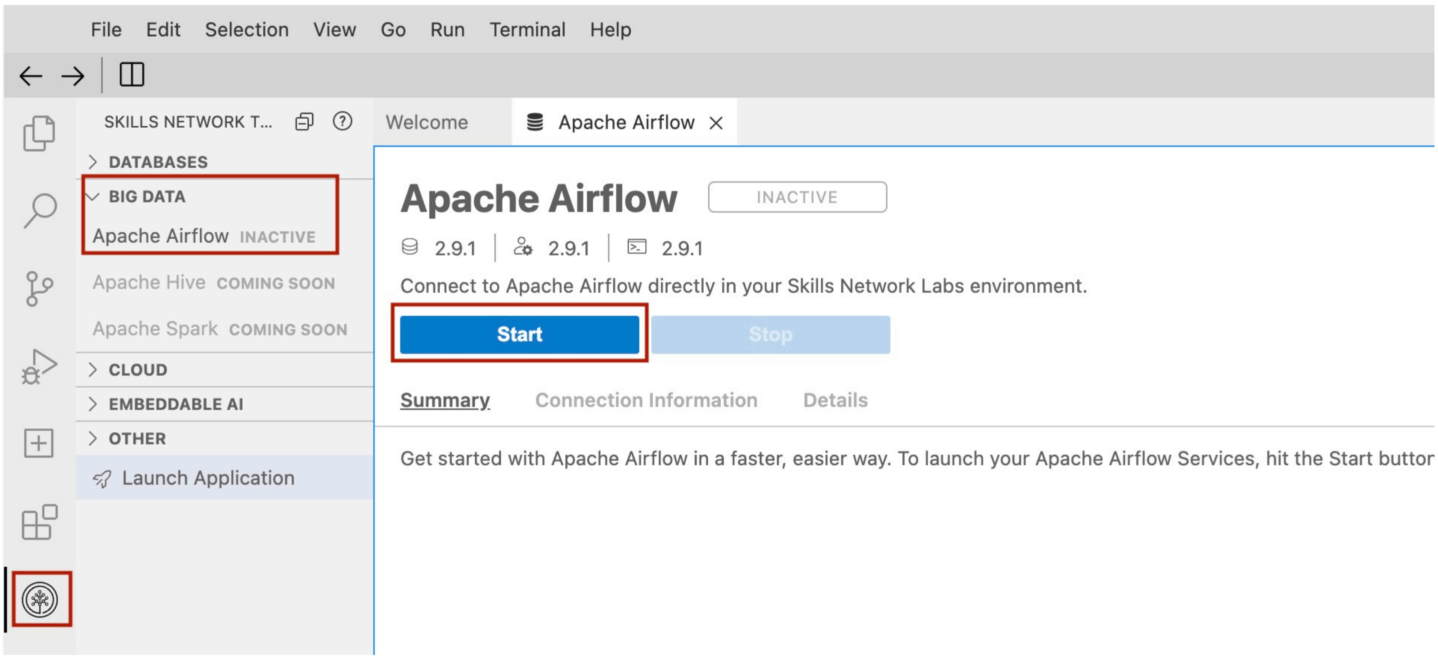
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs for course and project-related labs. Theia is an open-source IDE (Integrated Development Environment) that can be run on a desktop or on the cloud. To complete this lab, you will be using the Cloud IDE based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



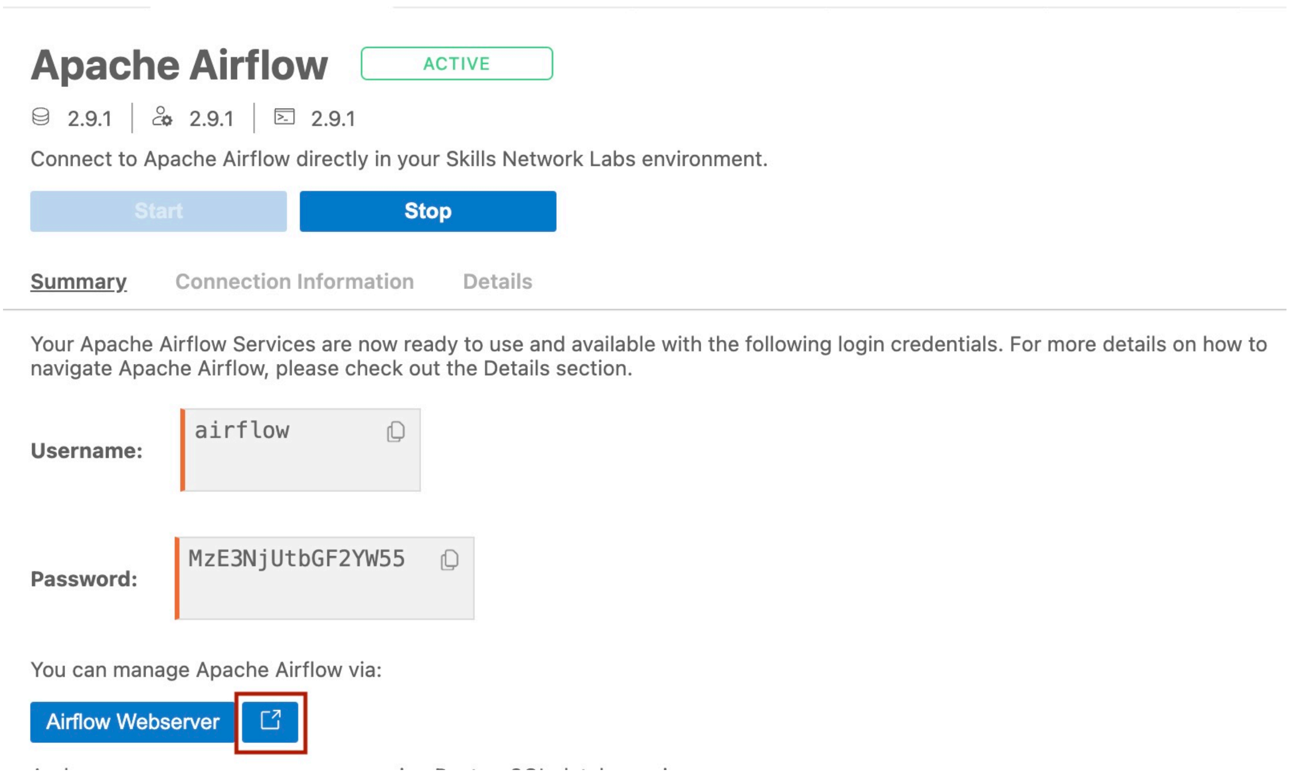
Connect to Apache Airflow directly in your Skills Network Labs environment.

Get started with Apache Airflow in a faster, easier way. To launch your Apache Airflow Services, hit the Start buttor

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.



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

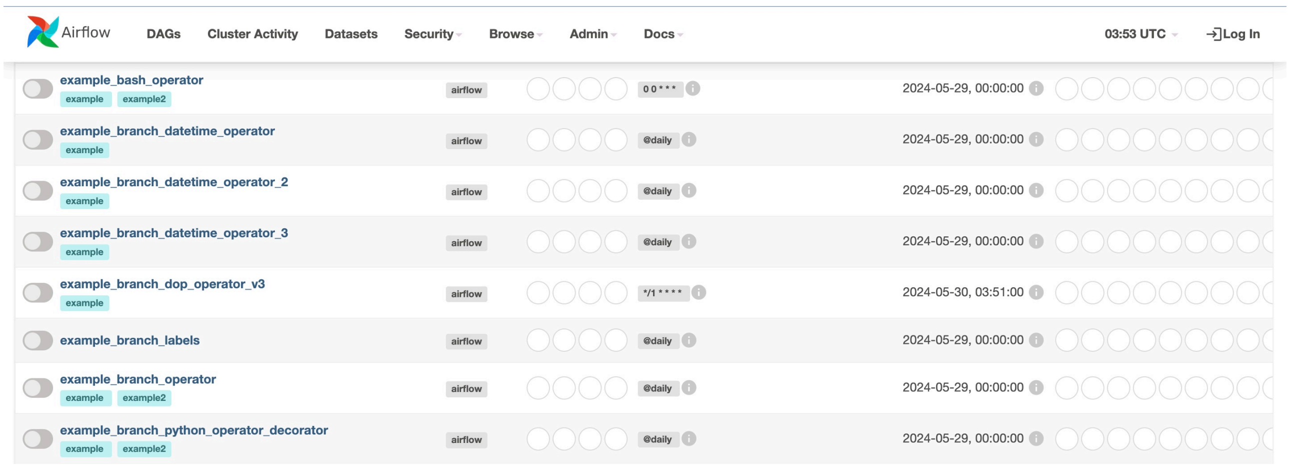
Your Apache Airflow Services are now ready to use and available with the following login credentials. For more details on how to navigate Apache Airflow, please check out the Details section.

Username: airflow

Password: MzE3NjUtbGF2YW55

You can manage Apache Airflow via:

You should land on a page that looks like this.



example_bash_operator example example2	airflow	0 0 * * *	2024-05-29, 00:00:00
example_branch_datetime_operator example	airflow	@daily	2024-05-29, 00:00:00
example_branch_datetime_operator_2 example	airflow	@daily	2024-05-29, 00:00:00
example_branch_datetime_operator_3 example	airflow	@daily	2024-05-29, 00:00:00
example_branch_dop_operator_v3 example	airflow	*/1 * * * *	2024-05-30, 03:51:00
example_branch_labels	airflow	@daily	2024-05-29, 00:00:00
example_branch_operator example example2	airflow	@daily	2024-05-29, 00:00:00
example_branch_python_operator_decorator example example2	airflow	@daily	2024-05-29, 00:00:00

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. To Create a new file choose `File->New File` and name it as `my_first_dag.py`.
   The `my_first_dag.py` file defines tasks `execute_extract`, `execute_transform`, `execute_load`, and `execute_check` to call the respective Python functions.

```
# Import the libraries
from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow.models import DAG
# Operators; you need this to write tasks!
from airflow.operators.python import PythonOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
# Define the path for the input and output files
input_file = '/etc/passwd'
extracted_file = 'extracted-data.txt'
transformed_file = 'transformed.txt'
output_file = 'data_for_analytics.csv'
def extract():
    global input_file
    print("Inside Extract")
    # Read the contents of the file into a string
    with open(input_file, 'r') as infile, \
            open(extracted_file, 'w') as outfile:
        for line in infile:
            fields = line.split(':')
            if len(fields) >= 6:
                field_1 = fields[0]
                field_3 = fields[2]
                field_6 = fields[5]
                outfile.write(field_1 + ":" + field_3 + ":" + field_6 + "\n")
def transform():
    global extracted_file, transformed_file
    print("Inside Transform")
    with open(extracted_file, 'r') as infile, \
            open(transformed_file, 'w') as outfile:
        for line in infile:
            processed_line = line.replace(':', ',')
            outfile.write(processed_line + '\n')
def load():
    global transformed_file, output_file
    print("Inside Load")
    # Save the array to a CSV file
    with open(transformed_file, 'r') as infile, \
            open(output_file, 'w') as outfile:
        for line in infile:
            outfile.write(line + '\n')
def check():
    global output_file
    print("Inside Check")
    # Save the array to a CSV file
    with open(output_file, 'r') as infile:
        for line in infile:
            print(line)
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Your name',
    'start_date': days_ago(0),
    'email': ['your email'],
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
# Define the DAG
```

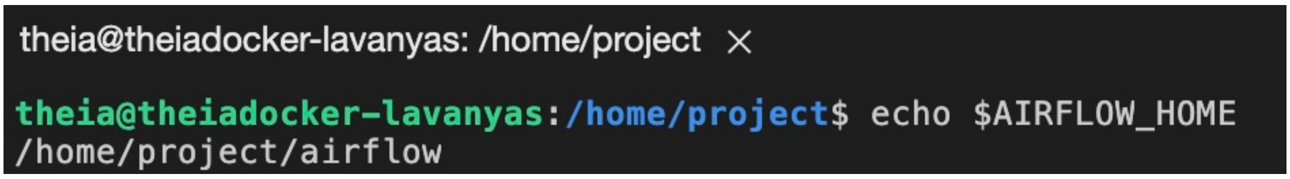
```
dag = DAG(
    'my-first-python-etl-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)
# Define the task named execute_extract to call the `extract` function
execute_extract = PythonOperator(
    task_id='extract',
    python_callable=extract,
    dag=dag,
)
# Define the task named execute_transform to call the `transform` function
execute_transform = PythonOperator(
    task_id='transform',
    python_callable=transform,
    dag=dag,
)
# Define the task named execute_load to call the `load` function
execute_load = PythonOperator(
    task_id='load',
    python_callable=load,
    dag=dag,
)
# Define the task named execute_load to call the `load` function
execute_check = PythonOperator(
    task_id='check',
    python_callable=check,
    dag=dag,
)
# Task pipeline
execute_extract >> execute_transform >> execute_load >> execute_check
```

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`.

```
export AIRFLOW_HOME=/home/project/airflow
echo $AIRFLOW_HOME
```



```
theia@theiadocker-lavanyas: /home/project ×

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.

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

3. Verify that your DAG actually got submitted.
4. Run the command below to list out all the existing DAGs.

```
airflow dags list
```

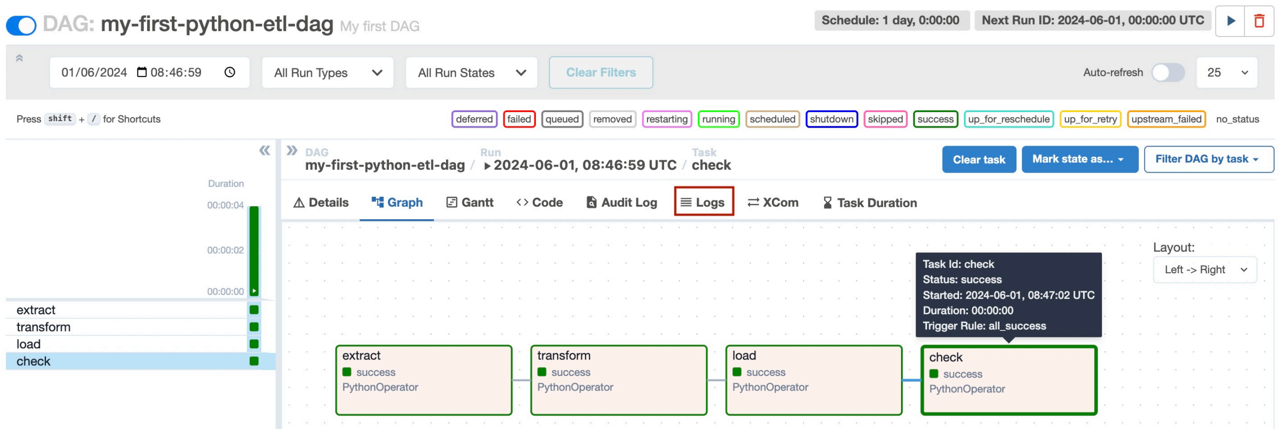
5. Verify that `my-first-python-etl-dag` is a part of the output.

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

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`.

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

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.



DAG: my-first-python-etl-dag My first DAG

Schedule: 1 day, 0:00:00   Next Run ID: 2024-06-01, 00:00:00 UTC

01/06/2024 08:46:59   All Run Types   All Run States   Clear Filters   Auto-refresh   25

DAG my-first-python-etl-dag / Run 2024-06-01, 08:46:59 UTC / Task check

Task Id: check
Status: success
Started: 2024-06-01, 08:47:02 UTC
Duration: 00:00:00
Trigger Rule: all_success

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.

a. `timestamp` - TIMESTAMP
b. `latitude` - float
c. `longitude` - float
d. `visitorid` - char(37)
e. `accessed_from_mobile` - boolean
f. `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**.

Follow the example code given in the lab and make necessary changes to create the new DAG.

▼ Click here for the **solution**.

Create a new file by going to **File -> New File** from the menu and name it as `ETL_Server_Access_Log_Processing.py`.
Copy the code below in the python file. This will contain your DAG with five tasks:

- download
- execute_extract
- execute_transform
- execute_load
- execute_check

```
# Import the libraries
from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow.models import DAG
# Operators; you need this to write tasks!
from airflow.operators.python import PythonOperator
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
import requests
# Define the path for the input and output files
input_file = 'web-server-access-log.txt'
extracted_file = 'extracted-data.txt'
transformed_file = 'transformed.txt'
output_file = 'capitalized.txt'
def download_file():
    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"
    # Send a GET request to the URL
    with requests.get(url, stream=True) as response:
        # Raise an exception for HTTP errors
        response.raise_for_status()
        # Open a local file in binary write mode
        with open(input_file, 'wb') as file:
            # Write the content to the local file in chunks
            for chunk in response.iter_content(chunk_size=8192):
                file.write(chunk)
    print(f"File downloaded successfully: {input_file}")
def extract():
    global input_file
    print("Inside Extract")
    # Read the contents of the file into a string
    with open(input_file, 'r') as infile, \
            open(extracted_file, 'w') as outfile:
        for line in infile:
            fields = line.split('#')
            if len(fields) >= 4:
                field_1 = fields[0]
                field_4 = fields[3]
                outfile.write(field_1 + "#" + field_4 + "\n")
def transform():
    global extracted_file, transformed_file
    print("Inside Transform")
    with open(extracted_file, 'r') as infile, \
            open(transformed_file, 'w') as outfile:
        for line in infile:
            processed_line = line.upper()
            outfile.write(processed_line + '\n')
def load():
    global transformed_file, output_file
    print("Inside Load")
    # Save the array to a CSV file
    with open(transformed_file, 'r') as infile, \
            open(output_file, 'w') as outfile:
        for line in infile:
            outfile.write(line + '\n')
def check():
    global output_file
    print("Inside Check")
    # Save the array to a CSV file
    with open(output_file, 'r') as infile:
        for line in infile:
            print(line)
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Your name',
    'start_date': days_ago(0),
    'email': ['your email'],
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
# Define the DAG
dag = DAG(
    'my-first-python-etl-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)
# Define the task named download to call the `download_file` function
download = PythonOperator(
    task_id='download',
    python_callable=download_file,
    dag=dag,
)
# Define the task named execute_extract to call the `extract` function
execute_extract = PythonOperator(
    task_id='extract',
    python_callable=extract,
    dag=dag,
)
# Define the task named execute_transform to call the `transform` function
execute_transform = PythonOperator(
    task_id='transform',
    python_callable=transform,
    dag=dag,
)
# Define the task named execute_load to call the `load` function
execute_load = PythonOperator(
    task_id='load',
    python_callable=load,
    dag=dag,
)
# Define the task named execute_load to call the `load` function
execute_check = PythonOperator(
    task_id='check',
    python_callable=check,
    dag=dag,
)
# Task pipeline
download >> execute_extract >> execute_transform >> execute_load >> execute_check
```

Copy the DAG file into the `dags` directory.

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

Verify if the DAG is submitted by running the following command.

```
airflow dags list | grep etl-server-logs-dag
```

If the DAG didn't get imported properly, you can check the error using the following command.

```
airflow dags list-import-errors
```

Authors

Lavanya T S

Other Contributors

Rav Ahuja