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

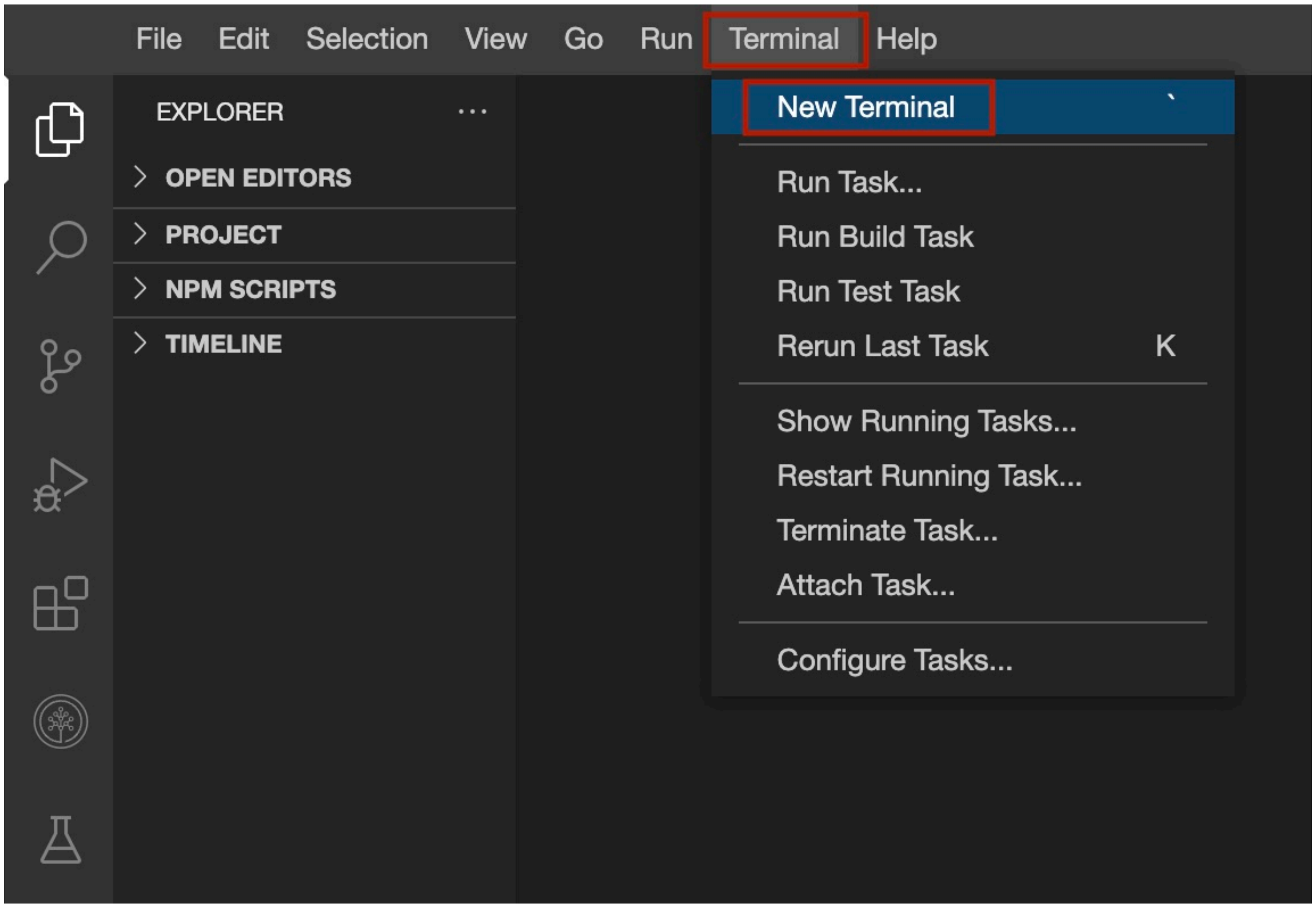
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment), that can be run on desktop or on the cloud. to complete this lab, you will be using the Cloud IDE based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

Important Notice about this lab environment

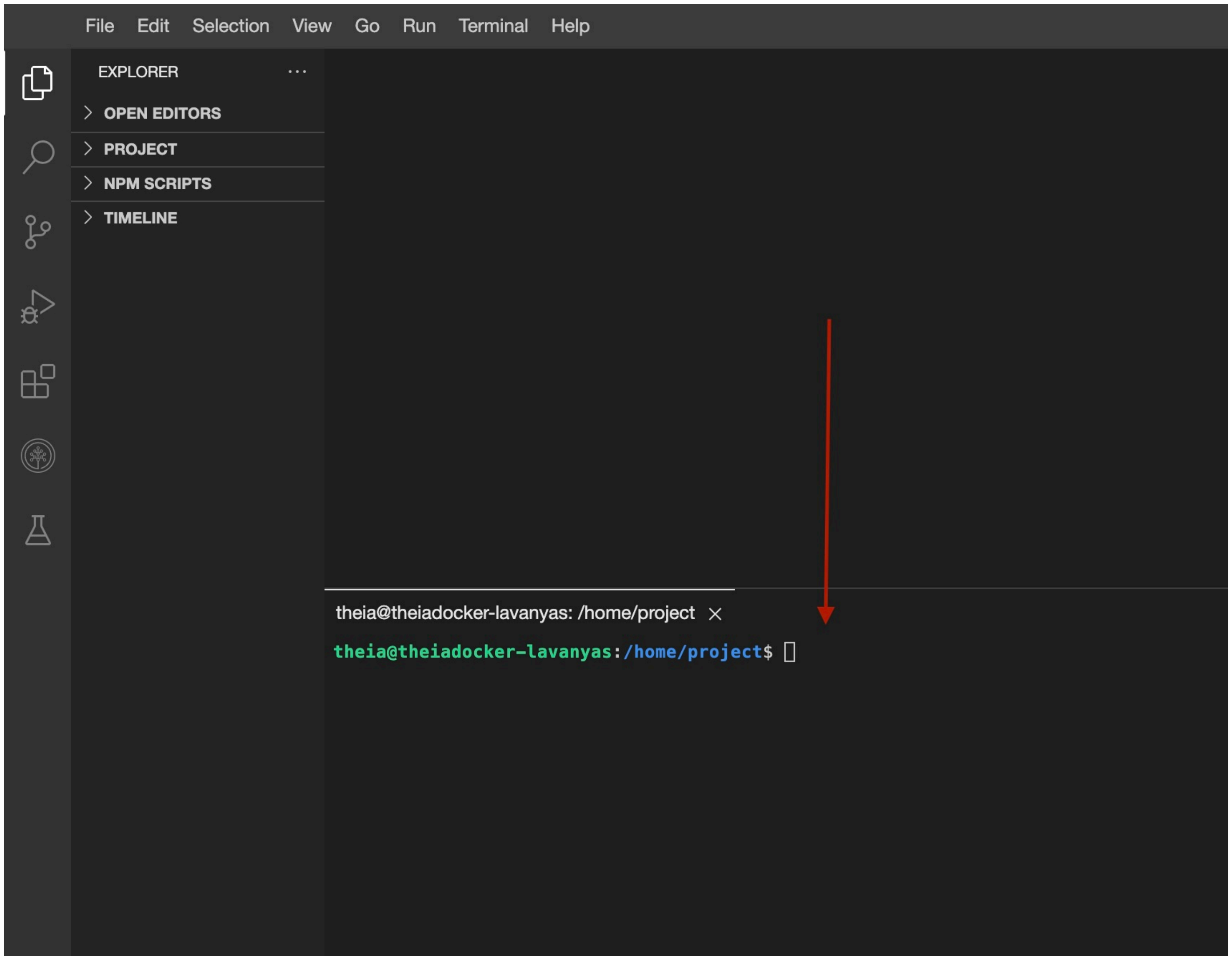
Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session will get lost. To avoid losing your data, please plan to complete these labs in a single session.

Exercise 1 - Start Apache Airflow

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.



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

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 in the terminal or use the >_ button to execute on the terminal.)

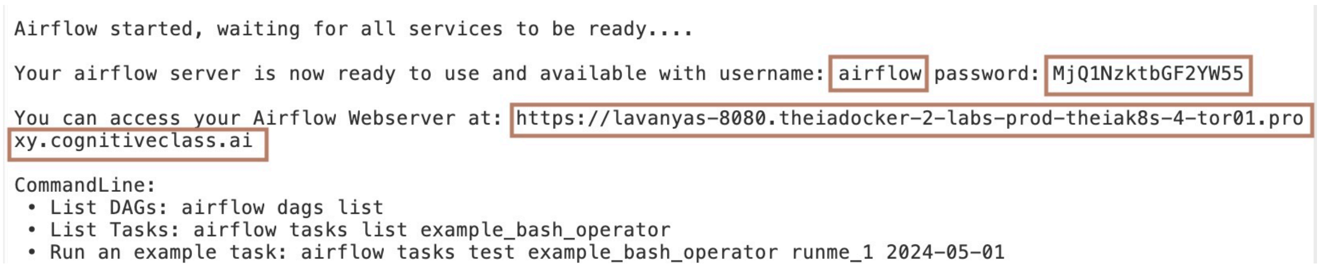
Start Apache Airflow in the lab environment.

```
1. 1
1. start_airflow
```

Copied! Executed!

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.

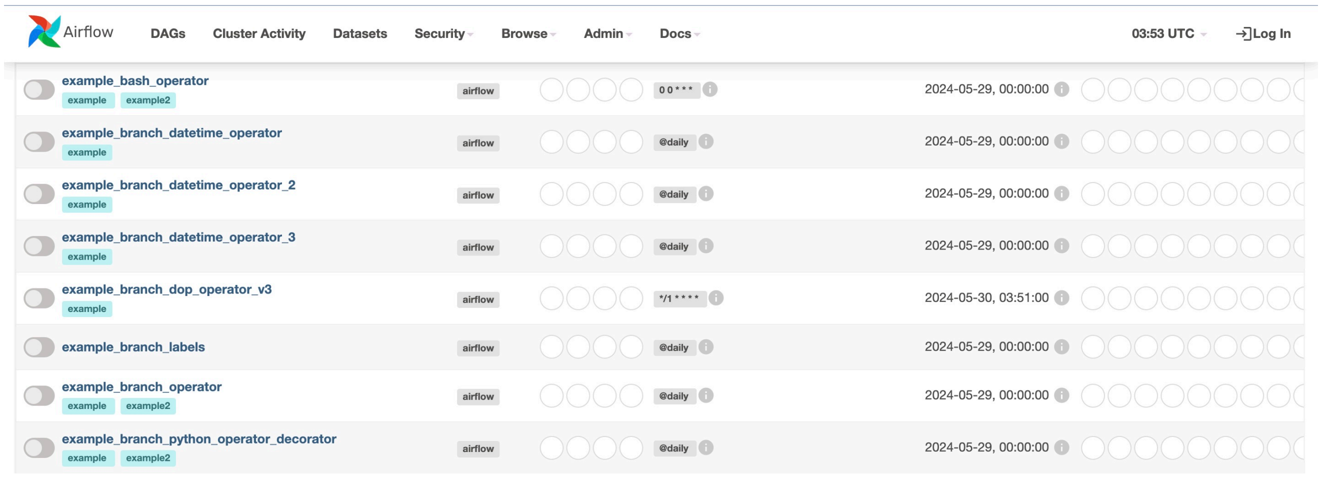


```
Airflow started, waiting for all services to be ready....

Your airflow server is now ready to use and available with username: airflow password: MjQ1NzktbGF2YW55

You can access your Airflow Webserver at: https://lavanyas-8080.theiadocker-2-labs-prod-theiak8s-4-tor01.pro
xy.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 2024-05-01
```

Exercise 2 - Open the Airflow Web UI

Click the button below or follow the steps given to open the airflow console on the browser.
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.



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 - 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.models import DAG
# Operators; you need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago
```

Copied!

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': 'Lavanya',
    'start_date': days_ago(0),
    'email': ['lavanya@somemail.com'],
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

Copied!

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,
- the number of retries in case of failure, and
- the time delay between retries.

The other options that you can include are:

- ‘queue’: The name of the queue the task should be a part of.
- ‘pool’: The pool that this task should use.
- ‘email_on_failure’: Whether an email should be sent to the owner on failure
- ‘email_on_retry’: Whether an email should be sent to the owner on retry
- ‘priority_weight’: Priority weight of this task against other tasks.
- ‘end_date’: End date for the task
- ‘wait_for_downstream’: Boolean value indicating whether it should wait for downtime
- ‘sla’: Time by which the task should have succeeded. This can be a timedelta object.
- ‘execution_timeout’: Time limit for running the task. This can be a timedelta object.
- ‘on_failure_callback’: Some function, or list of functions to call on failure
- ‘on_success_callback’: Some function, or list of functions to call on success
- ‘on_retry_callback’: Another function, or list of functions to call on retry
- ‘sla_miss_callback’: Yet another function, or list of functions when sla is missed
- ‘on_skipped_callback’: Some function to call when the task is skipped
- ‘trigger_rule’: Defines the rule by which the generated task gets triggered

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),
)
```

Copied!

Here you 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.

you 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,
)
```

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:

```
# task pipeline
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.

```python
# 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.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': 'your_name_here',
    'start_date': days_ago(0),
    'email': ['your_email_here'],
    '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 > /home/project/airflow/dags/extracted-data.txt',
    dag=dag,
)

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

# task pipeline
extract >> transform_and_load
```

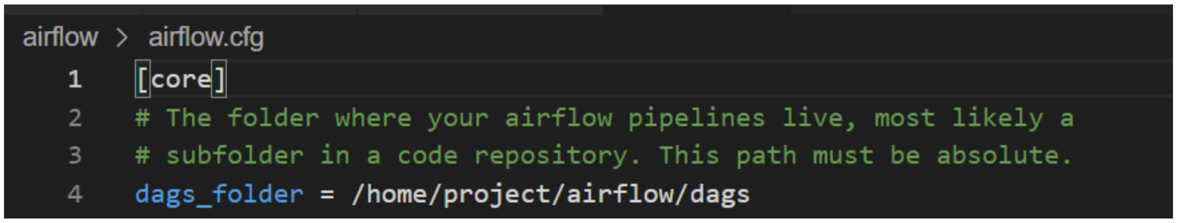
Copied!

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.

Airflow searches for Python source files within the specified `DAGS_FOLDER`. The location of `DAGS_FOLDER` can be located in the airflow.cfg file, where it has been configured as `/home/project/airflow/dags`.

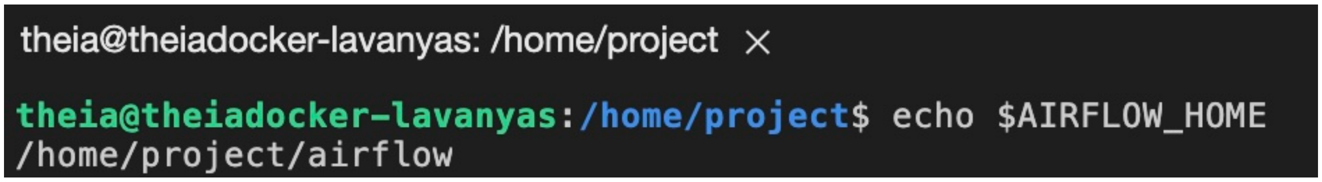


airflow > airflow.cfg

```
[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository. This path must be absolute.
dags_folder = /home/project/airflow/dags
```

Airflow will load the Python source files from this designated location. It will process each file, execute its contents, and subsequently load any DAG objects present in the file.

Therefore, when submitting a `DAG`, it is essential to position it within this directory structure. Alternatively, the `AIRFLOW_HOME` directory, representing the structure `/home/project/airflow`, can also be utilized for DAG submission.



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

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

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
```

Copied!

Verify that your DAG actually got submitted.

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

```
airflow dags list
```

Copied!

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

```
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`.

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

Copied!

You should see 2 tasks in the output.

Practice exercises

Write a DAG named `ETL_Server_Access_Log_Processing.py`.

1. Create the imports block.
2. Create the DAG Arguments block. You can use the default settings
3. Create the DAG definition block. The DAG should run daily.
4. Create the download task. The download task must download the server access log file which is available at the URL:

```
curl -o https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Apache%20Airflow/Build%20a%20DAG%20using%2
```

Copied!

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`.

6. Create the transform task. The `transform` task must capitalize the `visitorid`.

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

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

9. Submit the DAG.

10. Verify if the DAG is submitted

- ▶ Click here for Hint
- ▶ Click here for Solution

Authors

Lavanya T S
Ramesh Sannareddy