Hands-on Lab: Create a DAG for Apache Airflow

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

```
# 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': 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.

DAG Definition

```
# 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
```

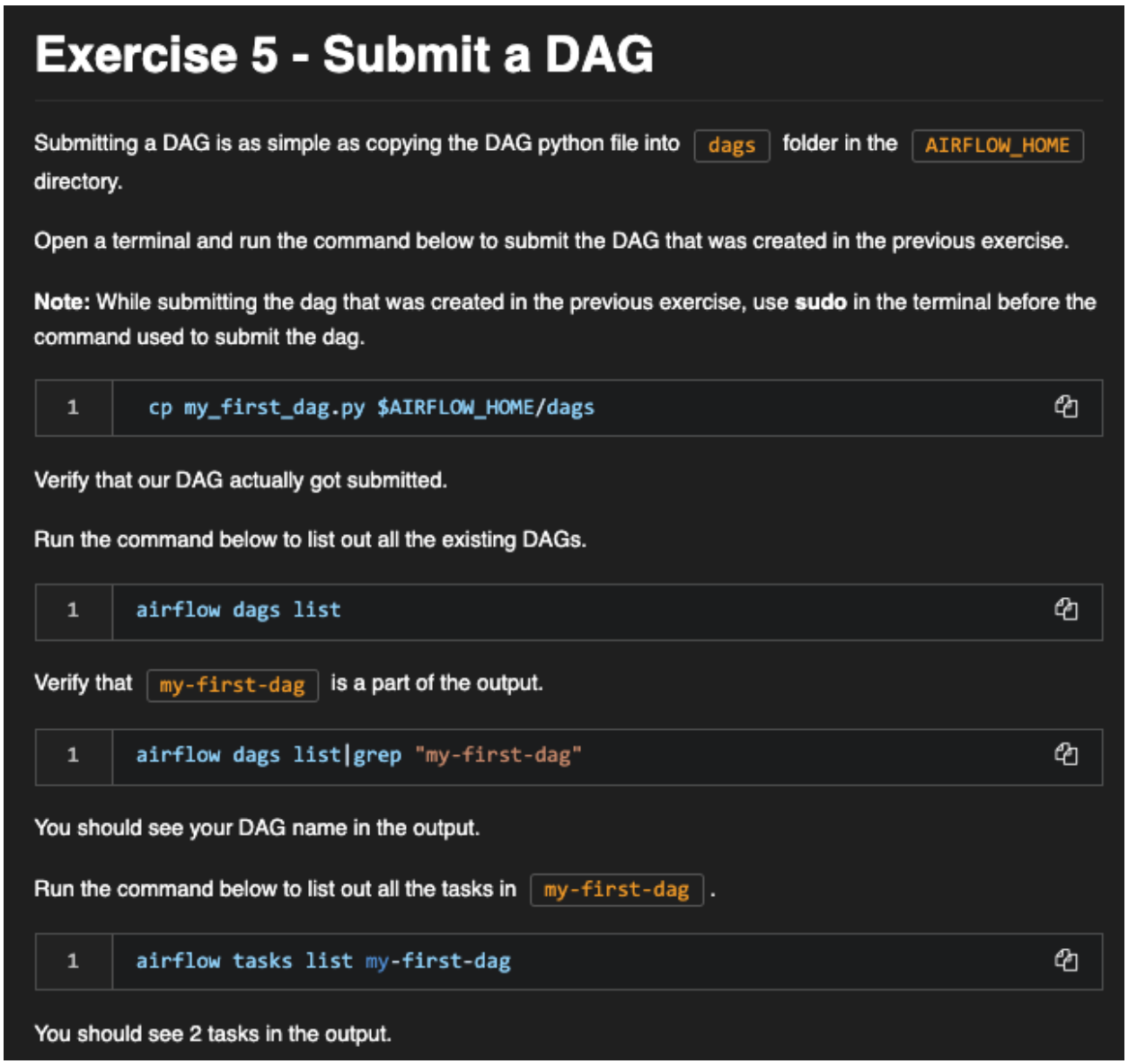
DAG Python Script

```
# 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 >
/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
```

Exercise 5 - Submit a DAG



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  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.

```
1  airflow dags list
```

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

```
1  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` .

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

You should see 2 tasks in the output.

Practice Exercise

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*

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
```