INFO-H-420
Management of Data Science and Business Workflows

Assignment 3
- REPORT -

Authors:
Cedric Kouamou Djampo
Salma SALMANI

Professor:
Dimitrios SACHARIDIS

Academic year 2023-2024

1 Exercise 1 : Workflows with Apache Airflow

A Directed Acyclic Graph (DAG) is a graph structure with nodes connected by edges that have a directional flow and contain no cycles, ensuring there is no way to return to an earlier node once moved forward. In computing and data processing, it's commonly used to represent and manage tasks where order and dependencies are crucial, as in workflow scheduling and data pipelines

The **process_web_log** DAG in Apache Airflow is specifically designed for daily processing of web log files, starting its operation from November 11, 2023. It is configured with a set of default arguments, including a `start_date` and a `catchup` parameter set to `False`, ensuring no back-filling of past data. This setup, aimed at data engineering tasks, simplifies and automates the routine analysis and management of web log data, reflecting a focus on efficiency and regularity in data processing workflows.

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta

default_args = {
    'start_date': datetime(2023, 11, 11),
    'catchup': False
}

with DAG(dag_id='process_web_log', default_args=default_args, schedule_interval='@daily',
         description='A simple DAG to process web log files', tags=['data_engineering']) as dag:

```

Figure 1: Create a DAG process_web_log

1.1 Create a task to scan for a log

The `scan_for_log` function is designed to check for the existence of a specific log file within a designated directory. It first imports the `os` module, essential for file path operations, and then constructs the full path to the log file by combining a predefined folder (`the_logs`) and a file path (`/home/ubuntus/airflow/dags/log.txt`). The function then checks if this file path exists using `os.path.exists`. If the file is found, it prints a confirmation message; otherwise, it notifies that the file is not found. This function is intended to be used as a task in an Airflow workflow, as shown by its integration with a `PythonOperator` in the given code.

```python
def scan_for_log():
    import os
    log_folder = 'the_logs'
    log_file = '/home/ubuntus/airflow/dags/the_logs/log.txt'
    log_file_path = os.path.join(log_folder, log_file)
    if os.path.exists(log_file_path):
        print(f"Found '{log_file_path}'")
    else:
        print(f"'{log_file_path}' not found")

scan_log_task = PythonOperator(
    task_id='scan_for_log',
    python_callable=scan_for_log)
```

Figure 2: scan for a logs

1.2 creat task to extract data

The `extract_data` function is designed to process a log file by extracting specific information from it. It opens a log file located at `/home/ubuntus/airflow/dags/the_logs/log.txt`, reads it line by line, and extracts the first element from each line, assumed to be an IP address.

This data is then written to a new file, `extracted_data.txt`. The function uses nested `with` statements for file handling, ensuring that both the input and output files are properly opened and closed. This Python function is set to be executed as a task in an Apache Airflow workflow, configured through a `PythonOperator` with the task ID `extract_data`. This setup indicates its use in data processing pipelines, particularly for extracting and saving specific pieces of data from larger datasets.

```
def extract_data():
    log_file_path = '/home/ubuntus/airflow/dags/the_logs/log.txt'   # Chemin complet vers log.txt
    output_file_path = '/home/ubuntus/airflow/dags/extracted_data.txt'  # Chemin complet pour le fichier de sorti

    with open(log_file_path, 'r') as file:
        with open(output_file_path, 'w') as output_file:
            for line in file:
                ip_address = line.split()[0]
                output_file.write(ip_address + '\n')


extract_data_task = PythonOperator(
    task_id='extract_data',
    python_callable=extract_data)
```

Figure 3: extract data

1.3 transform data task

The `transform_data` function takes the data from the "`extracted_data.txt`" file, filters out all occurrences of the IP address '`198.46.149.143`', and saves the filtered data into the "`transformed_data.txt`" file. It performs this operation by reading each line from the input file, checking if the specified IP address is present in the line, and only writing lines that do not contain the IP address to the output file. The `transform_data_task` is a `PythonOperator` that represents this function as a task within your DAG, ensuring this data transformation step is part of your workflow.

```
def transform_data():
    input_file = '/home/ubuntus/airflow/dags/extracted_data.txt'
    output_file = '/home/ubuntus/airflow/dags/transformed_data.txt'
    filter_ip = '198.46.149.143'
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        for line in infile:
            if filter_ip not in line:
                outfile.write(line)

transform_data_task = PythonOperator(
    task_id='transform_data',
    python_callable=transform_data)
```

Figure 4: transform data

1.4 creat task to load the data task

The `load_data` function creates a tar archive file named '`weblog.tar`' and adds the '`transformed_d` file to it. This task packages the transformed data for archiving or distribution. The `load_data_task` is a `PythonOperator` that integrates this function into our DAG, ensuring that the transformed data is efficiently archived, making it easier to manage and share as needed in your workflow.

```python
def load_data():
    import tarfile
    with tarfile.open('/home/ubuntus/airflow/dags/weblog.tar', 'w') as tar:
        tar.add('/home/ubuntus/airflow/dags/transformed_data.txt')

load_data_task = PythonOperator(
    task_id='load_data',
    python_callable=load_data)
```

Figure 5: load the data

1.5 Define the workflow that execute

This syntax is used to set the order of task execution in a DAG, indicating the dependencies between tasks.
`scan_log_task` ≫ `extract_data_task` ≫ `transform_data_task` ≫ `load_data_task`

2 Exercise 3: new task adding

Having tested all the tasks in our workflow thoroughly, it's reassuring to see that every component works well. From the initial task "`scan_log_task`" to the subsequent tasks "`extract_data_task`", "`transform_data_task`", and finally "`load_data_task`", each element was meticulously tested and found to work perfectly. This positive result is the fruit of careful planning and execution of the workflow design, which ensured that each task not only fulfilled its individual function effectively but also integrated seamlessly with the others.

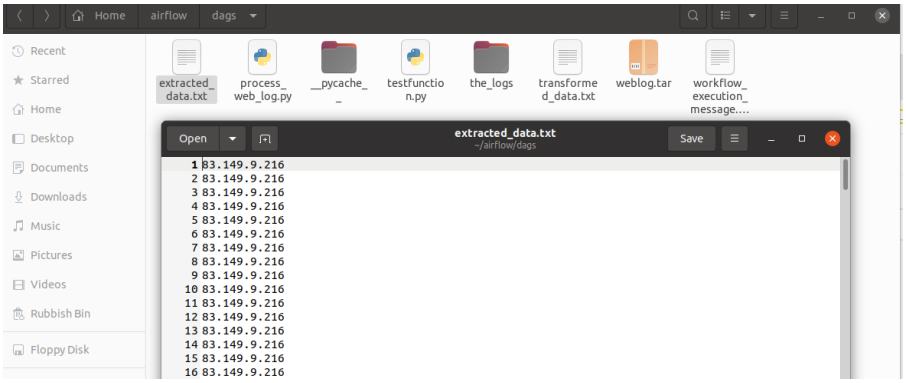

Figure 6: extract the data output

The screenshot fig 7 presents a user interface for a data processing workflow, specifically from a job scheduling or orchestration tool. It displays a Directed Acyclic Graph for a process named '`process_web_log`' timestamped for execution on the 14th of November, 2023, at 00:00:00 UTC. The visual elements include a series of our tasks aligned vertically with bar charts representing their duration. Each task is marked with green checkmarks, indicating successful completion. The interface also provides tabs for details, graph, and possibly additional information.

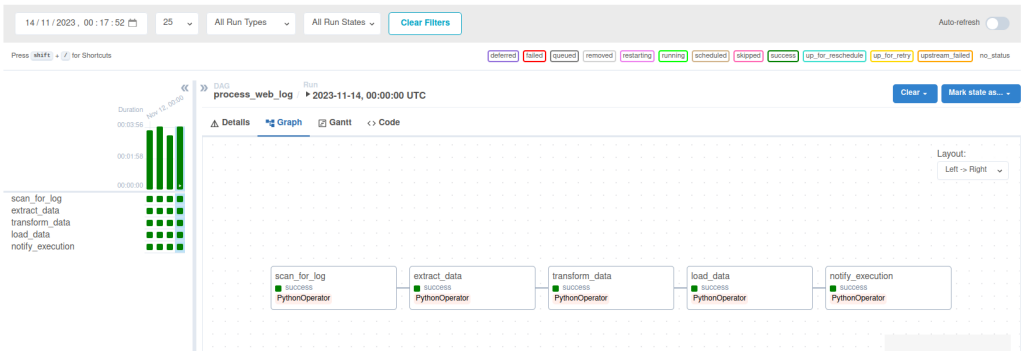

Figure 7: The screenshot of a user interface for a data processing workflow

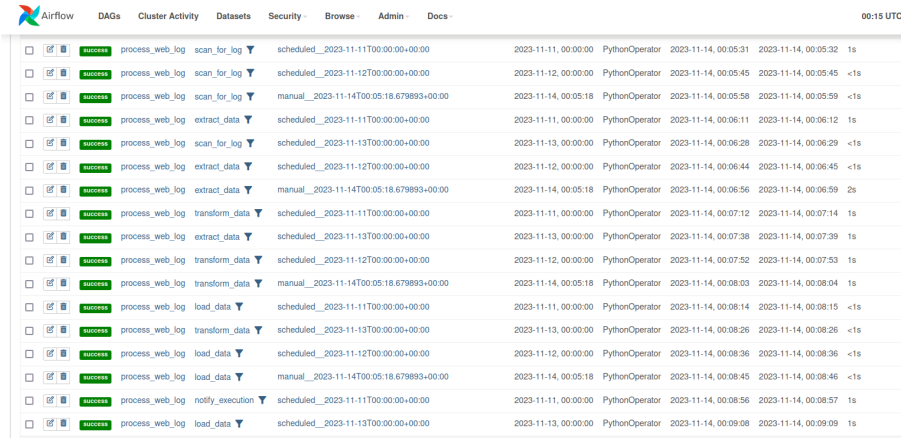

Figure 8: detail running

3 Exercise 4

For the specific notification task in our Airflow workflow, we opted for a simple but effective approach: send the execution confirmation message in a local file. This file is located directly on the machine running Airflow, at `'/home/ubuntus/airflow/dags/workflow_execution_message.t`. This method guarantees fast, direct access to the notification message, without the need for additional configuration of e-mail services or cloud storage. What's more, it simplifies the process by avoiding external dependencies, making the notification task both reliable and easily maintainable.

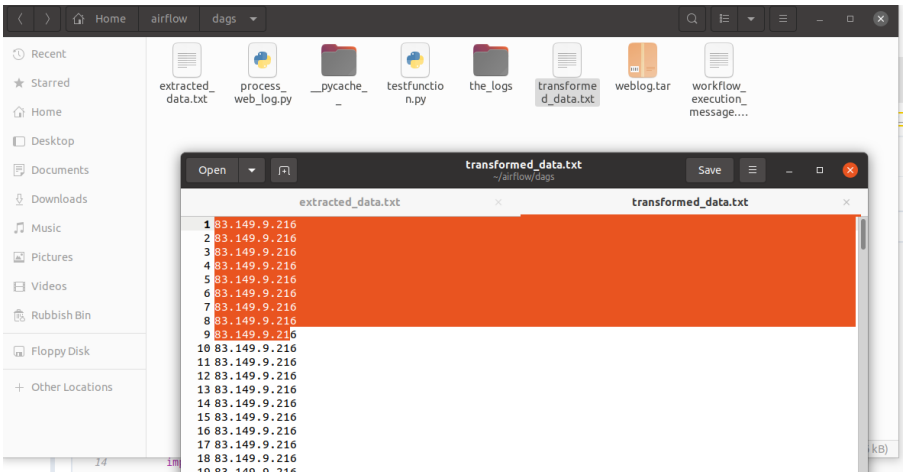

Figure 9: transform data output with all files output

```python
def notify_execution():
    message_file_path = '/home/ubuntus/airflow/dags/workflow_execution_message.txt'
    message = 'The workflow was executed successfully.'
    with open(message_file_path, 'w') as file:
        file.write(message)

notify_execution_task = PythonOperator(
  task_id='notify_execution',
   python_callable=notify_execution,
)


scan_log_task >> extract_data_task >> transform_data_task >> load_data_task>> notify_execution_task
```

Figure 10: send a message task

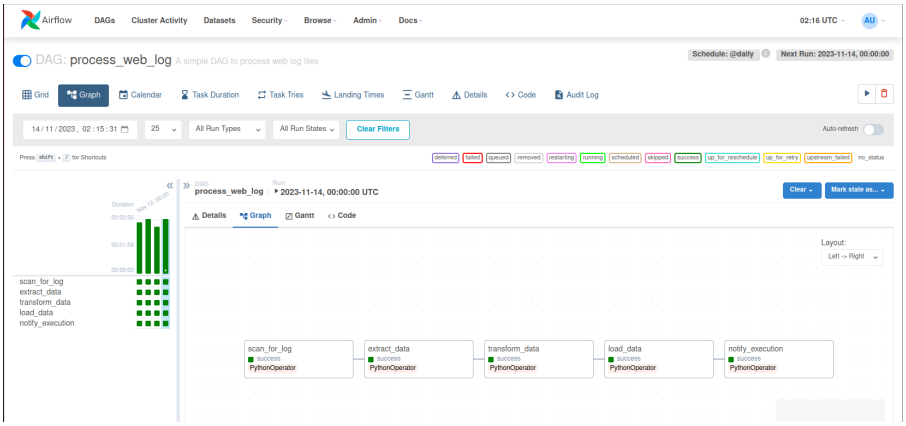

Figure 11: all task

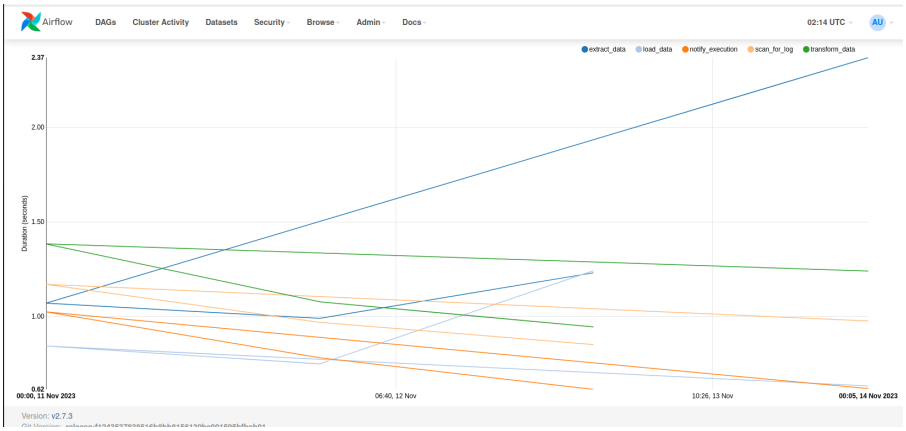

Figure 12: time duration task

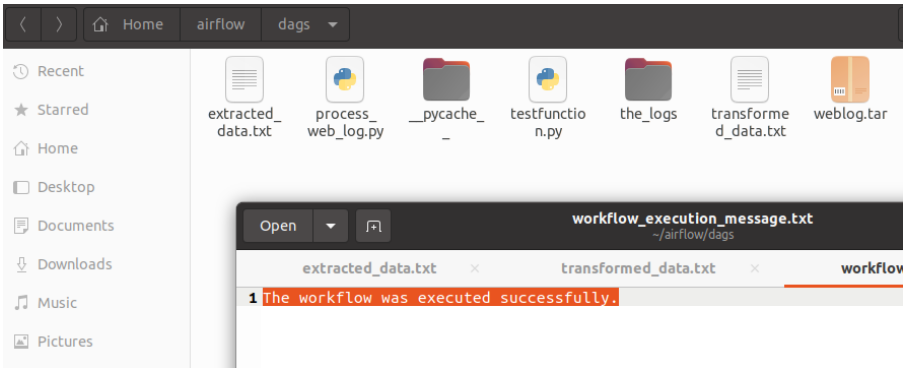

Figure 13: notify output task