Université libre de Bruxelles

École Polytechnique de Bruxelles

Assignment 3

Management of Data Science and Business Workflows INFO-H420

Fall 2023

Authors:
Bouzaher, Mohamed Louai
Dilbar Isakova

Professor:
Dimitri Sacharidis

Contents

Exercise 1

1.1 Overview

For the first exercise, we are asked to create the `"process_web_log"` DAG that automates daily web server log processing. The DAG involves scanning for a log file, extracting IP addresses, filtering specific occurrences, archiving the transformed data, and executing tasks sequentially using Apache Airflow.

1.2 Solution

- In this workflow, a DAG named `"process_web_log"` has been defined to automate the daily processing of web server logs. The DAG is scheduled to run on a daily basis.

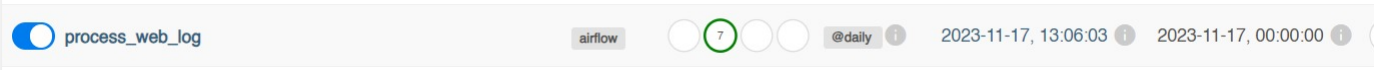

Figure 1.1: DAG

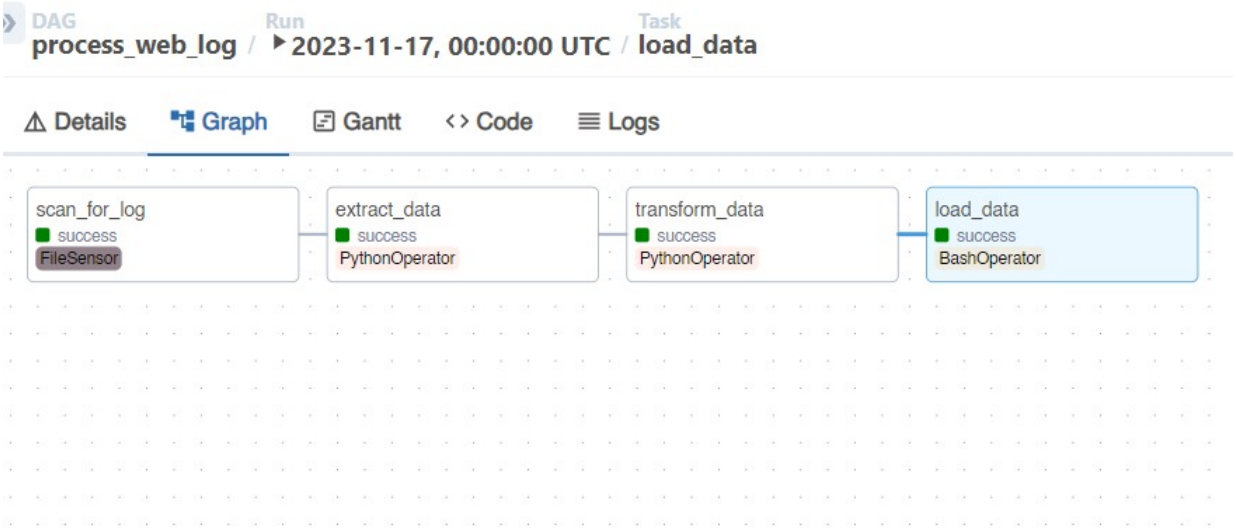

Figure 1.2: Workflow

Default Arguments:

```
default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 11, 16),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

Figure 1.3: Arguments

File Paths:

```
data_folder_path = '/home/louaibouzaher/airflow/data/'
log_file = os.path.join(data_folder_path, 'the_logs/log.txt')
extracted_file = os.path.join(data_folder_path, 'extracted_data.txt')
transformed_file = output_file = os.path.join(data_folder_path, 'transformed_data.txt')
archived_file = output_file = os.path.join(data_folder_path, 'weblog.tar')
```

Figure 1.4: Paths

Data Processing Functions: `extract_data` and `transform_data`, are defined to perform specific data processing tasks. `extract_data` reads an input log file, extracts IP addresses, and writes them to another file. `transform_data` reads the extracted data, filters out specific IP addresses, and writes the result to another file.

```
def extract_data():
    ip_pattern = r"\b(?:\d{1,3}\.){3}\d{1,3}\b"
    with open(log_file, 'r') as file:
        log_data = file.read()

    ip_addresses = re.findall(ip_pattern, log_data)
    with open(extracted_file, 'w') as file:
        for ip in ip_addresses:
            file.write(ip + '\n')



def transform_data():
    eliminated_values = ['198.46.149.143']

    with open(extracted_file, 'r') as infile:
        lines = infile.readlines()

    filtered_lines = [line.strip() for line in lines
                      if not any(value in line for value in eliminated_values)]

    with open(transformed_file, 'w') as outfile:
        outfile.write('\n'.join(filtered_lines))
```

Figure 1.5: Functions

`scan_for_log_task`

- This task checks for the existence of a log file named "log.txt" in the "the_logs" folder. If the file is found, it triggers the subsequent tasks in the workflow. This task is implemented as a `FileSensor` operator.

`extract_data_task`

- The `extract_data` task is responsible for extracting the IP address field from the web server log file. The extracted data is then saved into a file named "extracted_data.txt". This task is implemented as a `PythonOperator`, and the extracted data is passed to the next task using XCom.

`transform_data_task`

- In the `transform_data` task, the workflow filters out all occurrences of the IP address "198.46.149.143" from the "extracted_data.txt" file. The filtered data is then saved into a new file named "transformed_data.txt". Like the previous task, this one is also implemented as a `PythonOperator`.

`load_data_task`

- The final task, `load_data_task`, archives the "transformed_data.txt" file into a tar file named "weblog.tar." This task is implemented as a `BashOperator`.

```
49  with DAG('process_web_log',
50           default_args=default_args,
51           schedule="@daily"
52          ) as dag:
53
54      scan_for_log_task = FileSensor(
55          task_id="scan_for_log",
56          filepath=log_file,
57      )
58
59      extract_data_task = PythonOperator(
60          task_id='extract_data',
61          python_callable=extract_data
62      )
63
64      transform_data_task = PythonOperator(
65          task_id='transform_data',
66          python_callable=transform_data
67      )
68
69      load_data_task =  BashOperator(
70      task_id='load_data',
71      bash_command='tar -cf {} {}'.format(archived_file, transformed_file)
72      )
```

Figure 1.6: Tasks

- The tasks are set up with dependencies, where each task depends on the completion of the previous one. The arrow (`>>`) signifies the direction of dependency.

```
73
74      scan_for_log_task >> extract_data_task >> transform_data_task >> load_data_task
75
```

Figure 1.7: Task Dependencies

Exercise 2

2.1 Overview

In this exercise, the assigned task involved conducting individual test runs for each defined task. Subsequently, upon confirming their successful execution, the next step was to perform a test run for the entire workflow. Following this, the workflow was triggered and executed, and several runs were monitored. The subsequent report should detail the test runs conducted, along with any findings or observations gathered from these runs.

2.2 Solution

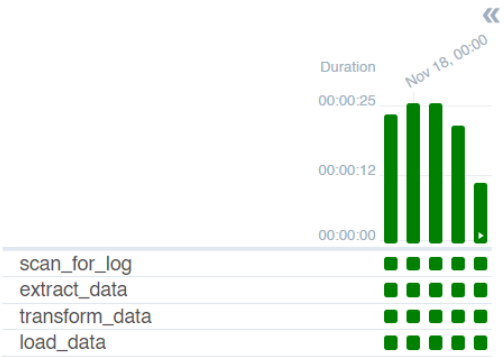

Figure 2.1: Process Concurrent Executions

- Max Run Duration: 25s
- Min Run Duration: 10s
- Average Run Duration: 21s

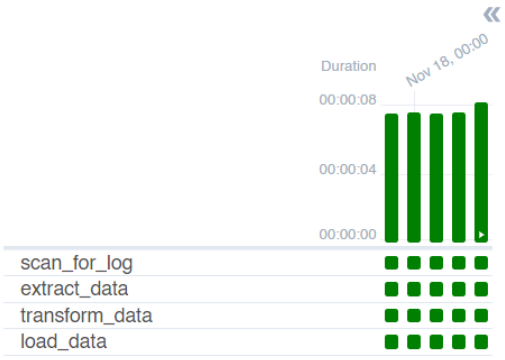

Figure 2.2: Process Sequential Executions

- Max Run Duration: 8s
- Min Run Duration: 8s
- Average Run Duration: 8s

Exercise 3

3.1 Overview

In last exercise, we have to add to the workflow an additional task after the `load_data_task`. So, the new `inform_user_task` should send an email about successful execution of the workflow.

3.2 Solution

```python
def inform_user():
    email = EmailMessage()
    email['from'] = 'Airflow - Group 3.40'
    email['to'] = 'dilbar.isakova@ulb.be'
    email['subject'] = 'Airflow Notification'
    current_time = datetime.now()
    email.set_content(f"Process finished successfully at {current_time}")

    sender_email = 'processweblog.info420@gmail.com'
    password = [redacted]

    with smtplib.SMTP(host='smtp.gmail.com', port = 587) as smtp:
        smtp.ehlo()
        smtp.starttls()
        smtp.login(sender_email, password)
        smtp.send_message(email)
        print('Email sent successfully ✅')
```

Figure 3.1: Function for the New Task

```python
inform_user_task =  PythonOperator(
    task_id='inform_user',
    python_callable=inform_user
)

scan_for_log_task >> extract_data_task >> transform_data_task >>
load_data_task >> inform_user_task
```

Figure 3.2: New Task and Task Dependencies

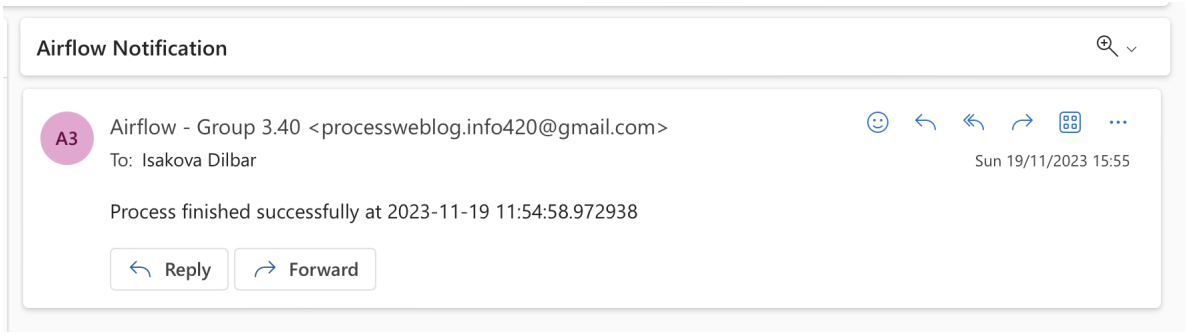

Figure 3.3: Email Notification

This is a link to the actual code for your reference:

https://louaibouzaher.notion.site/louaibouzaher/Process-Web-Log-214806cf3c2b4713b7cac4e60a469b8a