Assignment 4: Workflows with Apache Airflow

Team members: Bogdana Živković, Nikola Ivanović

Exercise 1

We installed Docker Engine for 64-bit Ubuntu 22.04: set up the repository and installed Docker Engine. Then we insstalled Docker Desktop with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install ./docker-desktop-<version>-<arch>.deb
```

We created an airflow directory named "airflow-docker" and inside of it we create logs, plugins and dags directories. We acquired docker-compose.yaml from:

https://airflow.apache.org/docs/apache-airflow/stable/docker-compose.yaml

Finally we had to set up the environment variables:

AIRFLOW_UID = 1000

AIRFLOW_GID = 0

Additionally, we had to create "the_logs" directory and add following line in docker-compose.yaml so that the directory could be shared between the host and the docker container:

```
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
  - ./the_logs:/the_logs
```

Exercise 2

We used the following imports:

```
from airflow import DAG
from airflow.operators.python import PythonOperator, BranchPythonOperator

from datetime import datetime
import os
import tarfile
```

The DAG is defined in the following code snippet:

```
with DAG('process_web_log', start_date = datetime(2023, 1, 1),
    schedule_interval = '@daily', catchup = False) as dag:

    scan_for_log = BranchPythonOperator (
            task_id='scan_for_log',
            python_callable= log_exists
        )

    extract_data = PythonOperator (
            task_id='extract_data',
            python_callable= extract_ip_address
        )

    transform_data = PythonOperator (
        task_id = 'transform_data',
        python_callable = transform_extracted
    )

    load_data = PythonOperator (
        task_id = 'load_data',
        python_callable = export_tar
    )

    scan_for_log >> [extract_data] >> transform_data >> load_data
```

The DAG is named "process_web_log" and it is set to run daily by assigning '@daily' to schedule_interval. The dag includes 4 tasks that are performed sequentially. The first task scan_for_log has type BranchPythonOperator and it determines whether the following tasks will be performed. All following tasks have type PythonOperator.

The task "scan_for_log" calls the function "log_exists()". If the log.txt file is present in "the_logs directory" the next task will be executed. Otherwise all following tasks will be skipped.

```
def log_exists():
    if 'log.txt' in os.listdir('/the_logs'):
        return 'extract_data'
    else:
        return None
```

The task "extract_data" calls the function "extract_ip_address()".

```
def extract_ip_address():
    ips = []
    with open('/the_logs/log.txt', 'r') as log:
        for line in log.readlines():
            ip = line.split(' ')[0]
            ips.append(ip)

    with open('/the_logs/extracted_data.txt', 'w+') as res:
        for ip in ips:
            res.write(ip + '\n')
```

The task "transform_data" calls the function "transform_extracted()".

```
def transform_extracted():
    ips = []
    with open('/the_logs/extracted_data.txt', 'r') as extracted:
        for ip in extracted.readlines():
            if ip != '198.46.149.143\n':
                ips.append(ip)

    with open('/the_logs/transformed_data.txt', 'w+') as res:
        for ip in ips:
           res.write(ip)
```

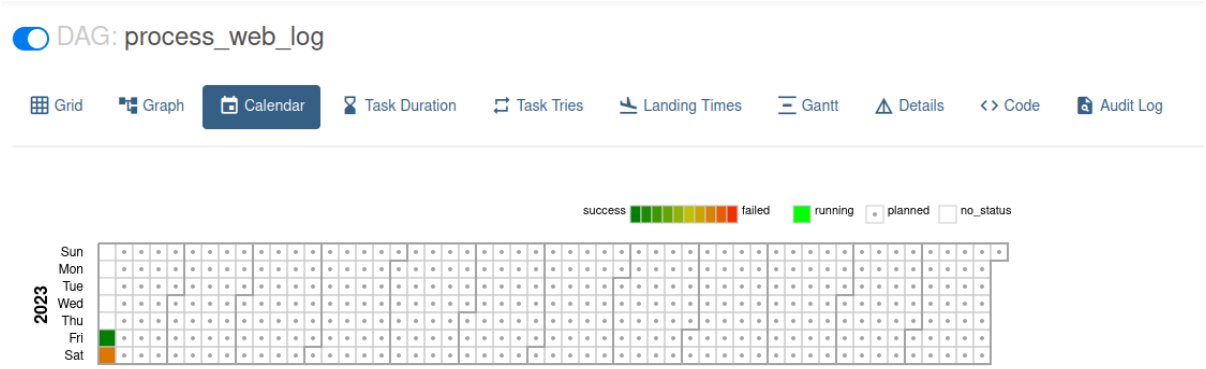
The task "load_data " calls the function "load_data()".

```
def export_tar():
    file = tarfile.open('/the_logs/weblog.tar', 'w')
    file.add('/the_logs/transformed_data.txt')
    file.close()
```

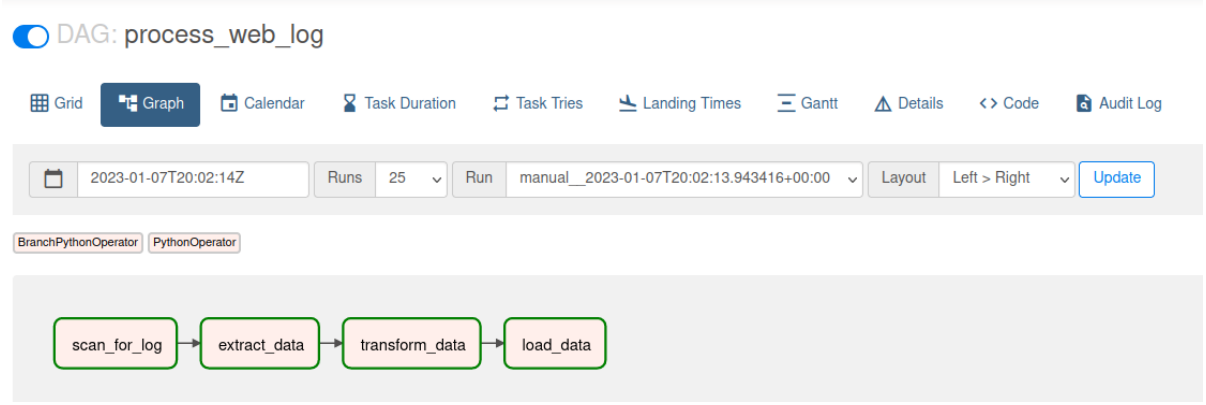
For the entire implementation visit: https://github.com/ivanovicnikola/airflow-assignment

Exercise 3

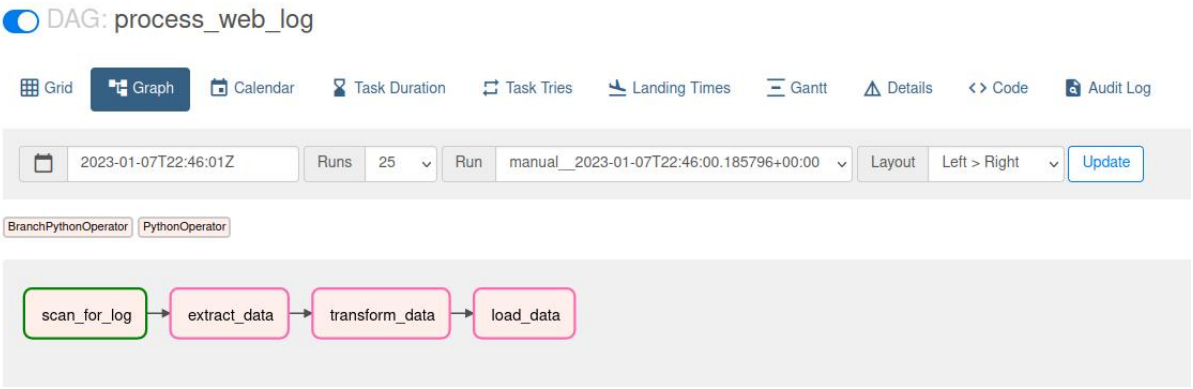
In the first photo you can see that the workflow "process_web_log" is scheduled to be executed every day:



Here you can see all the tasks in the defined dag, that have been executed in sequence:



In the case that log.txt file is not present in "the_logs" directory all the tasks after "scan_for_log" will be skipped (pink color in the graph):



The following picture shows the process of testing the workflow cumulatively after implementing each task: