UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Apache Airflow

Oscar Evanilson Gutiérrez Pérez

Carrera: Ingeniería en Computación

Código de estudiante: 219748308

13 de marzo de 2022

En este programa, utilicé el lenguaje de Python para seguir un tutorial de internet que encontré en el que pude utilizar la librería ariflow de apache, es un ejemplo sencillo donde se hace un ejemplo del uso de DAGs al momento de crear tareas como las del ejercicio anterior.

```python
from datetime import timedelta
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
import pandas as pd


def first_function_execute(**context):
    print("first_function_execute   ")
    context['ti'].xcom_push(key='mykey', value="first_function_execute says Hello ")


def second_function_execute(**context):
    instance = context.get("ti").xcom_pull(key="mykey")
    data = [{"name":"Soumil","title":"Full Stack Software Engineer"}, { "name":"Nitin","title":"Full Stack
    df = pd.DataFrame(data=data)
    print('@'*66)
    print(df.head())
    print('@'*66)

    print("I am in second_function_execute got value :{} from Function 1  ".format(instance))

```

En esta primera parte del ejemplo vemos las librerías que se importan para que funcione de manera correcta, primero tenemos la datetime que con esa va a poder marcar la hora de la información en vivo que va recabando, después tenemos la de airflow y unas extensiones de la misma con las que se tienen modulos como el task y DAG.

Además, vemos que se definen dos funciones donde se va a imprimir información y se les asigna unos valores que funcionan para la parte del código donde se pone en funcionamiento del DAG.

```python
with DAG(
        dag_id="first_dag",
        schedule_interval="@daily",
        default_args={
            "owner": "airflow",
            "retries": 1,
            "retry_delay": timedelta(minutes=5),
            "start_date": datetime(2021, 1, 1),
        },
        catchup=False) as f:

    first_function_execute = PythonOperator(
        task_id="first_function_execute",
        python_callable=first_function_execute,
        provide_context=True,
        op_kwargs={"name":"Soumil Shah"}
    )

    second_function_execute = PythonOperator(
        task_id="second_function_execute",
        python_callable=second_function_execute,
        provide_context=True,
    )

first_function_execute >> second_function_execute
```

En esta parte del programa podemos ver como se implementan estos módulos mencionados a continuación, podemos ver la declaración de un DAG y todos los parámetros que este necesita, además de dos ejecuciones de funciones que van de la mano con el módulo PythonOperator.

Link de supositorio en GitHub para revisar el código:

https://github.com/oscarevanilson/Computacion-Tolerante-a-Fallas

Conclusiones

Este programa me resultó demasiado complicado, aunque solamente haya tenido que seguir un tutorial de internet, la verdad es que se me hace muy complicado y confuso el funcionamiento de esta librería de airflow, además de que en este ejemplo no entendí muy bien que es lo que el programa realizaba al momento de ejecutar las funciones mostradas en el código y el papel de DAG en este mismo.

Creo que esta librería es muy reciente y no hay demasiada información sobre ella en internet y por esto me costó trabajo entender bien el funcionamiento de esta. Por

esta misma razón creo que esta herramienta es muy poco usada por lo que pude ver en los ejemplos que busqué en internet, espero que en un futuro pueda obtener más conocimiento de esta y que pueda utilizarla de mejor manera en tareas o programas más complejos.