



Airflow

Nombre: Diana Fernanda Castillo Rebolledo

Materia: Computación Tolerante a Fallas

Horario: L-I 11:00 am-1:00 pm

Profesor: Dr. Michel Emanuel López Franco

Sección: D06

Mi ejemplo generado con Apache Airflow consta de un proceso ETL (Extract, transform and load) que consiste en la extracción de datos del link “<https://jsonplaceholder.cypress.io/todos>”, el cual simula información sobre varios usuarios, esto en una lista de objetos json. Antes de comenzar, instalé Airflow, para esto tuve que activar WSL (Windows Subsystem for Linux) en Windows e instalar Ubuntu para ya poder instalar Airflow desde ahí e inicializar el servidor web, ya que no puede instalarse directamente en Windows.

Una vez instalado, cree un archivo .py junto a los dags de ejemplo generados al momento de la instalación. Primero importé DAG de airflow para poder definir un dag (Grafo Dirigido Acíclico), también importé PythonOperator el cual servirá para que unas funciones de Python se puedan describir como tareas del flujo de trabajo, y también importé days_ago para poder especificarle al dag desde cuando se tuvo que haber ejecutado.

Después definí un diccionario llamado default_args en donde se pueden especificar algunos parámetros bajo los cuales se van a ejecutar las tareas, parámetros como el propietario de la tarea, un email a donde enviar notificaciones en caso de fallo o reintento, el número de veces que se debe reintentar una tarea si es que falla, que en este caso serán 5 veces y el tiempo que pasará entre cada reintento, en este caso 1 minuto.

```
default_args = {
    'owner': 'Diana',
    'depends_on_past': False,
    'email': ['diana.castillo7676@alumnos.udg.mx'],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 5,
    'retry_delay': timedelta(minutes=1)
}
```

Lo siguiente fue crear las tres funciones que van a realizar el proceso ELT. La primera que es la de extract, realiza una petición para extraer la información del link mencionado anteriormente con ayuda de la librería requests que es para realizar peticiones HTTP y retorna el resultado de esa petición, que es una lista de objetos json.

La siguiente función es transform, aquí se utilizan los datos obtenidos de la extracción, los cuales son una lista de diccionarios. Para compartir datos entre tareas se utiliza el mecanismo xcoms (cross-communications), el cual puede recuperar los datos retornados por otras tareas del dag, pues estos se almacenan en la base de datos del airflow. Para ello se tiene que crear una instancia de la tarea (ti) y llamar a xcoms_pull para extraer datos, y especificar el id de la tarea de los datos a extraer, en este caso es el dato retornado por extract.

```

def extract():
    resultado = requests.get('https://jsonplaceholder.cypress.io/todos')
    return json.loads(resultado.content)

def transform(ti=None):
    datos = ti.xcom_pull(task_ids="extract")
    usuarios = []
    for usuario in datos:
        usuarios.append({
            'UsuarioID': usuario['userId'],
            'ID': usuario['id'],
            'Titulo': usuario['title'],
            'Completado': usuario['completed']
        })
    return usuarios

```

Después en esa misma función, por cada objeto dentro de la lista, se crea un diccionario usando los valores de cada objeto, pero modificando las keys, y cada uno de estos nuevos diccionarios se agregan a una lista nueva y esta se retorna.

En la siguiente función llamada load, se utiliza la lista retornada de transform con xcoms como en la otra función, se abre un archivo.json, se guarda la lista de diccionarios con ayuda de json.dump y se cierra el archivo.

Una vez realizadas las tres funciones, definí el dag. Primero definí su nombre 'airflow_dag', después le pasé el diccionario de argumentos definido al principio, una breve descripción del dag, cada cuándo se va ejecutar (en este caso una vez al día), cuándo es que el dag tuvo que haber comenzado (en este caso dos días antes), y etiquetas para marcar el trabajo si así se requiere.

```


with DAG(
    'airflow_dag',
    default_args=default_args,
    description='Airflow dag con proceso etl',
    schedule_interval=timedelta(days=1),
    start_date=days_ago(2),
    tags=['airflow dag']
) as dag:
    extract_task = PythonOperator(task_id="extract", python_callable=extract)
    transform_task = PythonOperator(task_id="transform", python_callable=transform)
    load_task = PythonOperator(task_id="load", python_callable=load)

    extract_task >> transform_task >> load_task

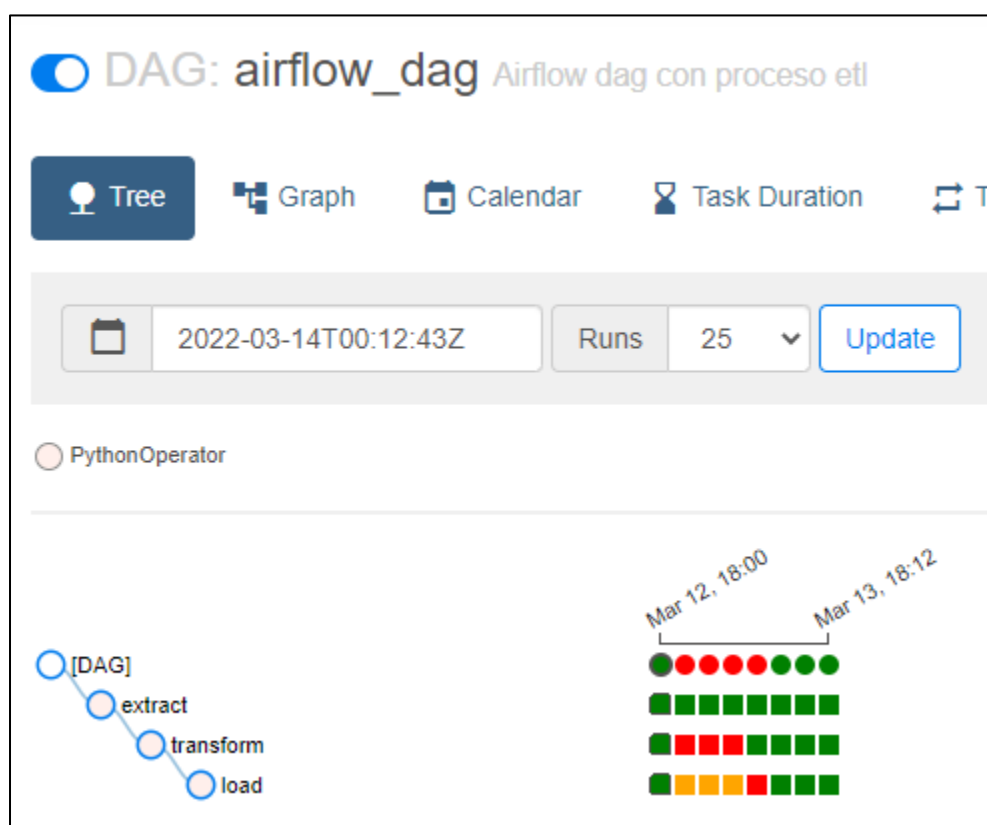
```

Dentro del dag definí cada una de las tareas con ayuda del PythonOperator, ahí se le coloca un identificador único a cada tarea y se indica cuál es la función que se debe llamar. Por último, identifiqué las dependencias entre las tareas con el operador ">>".

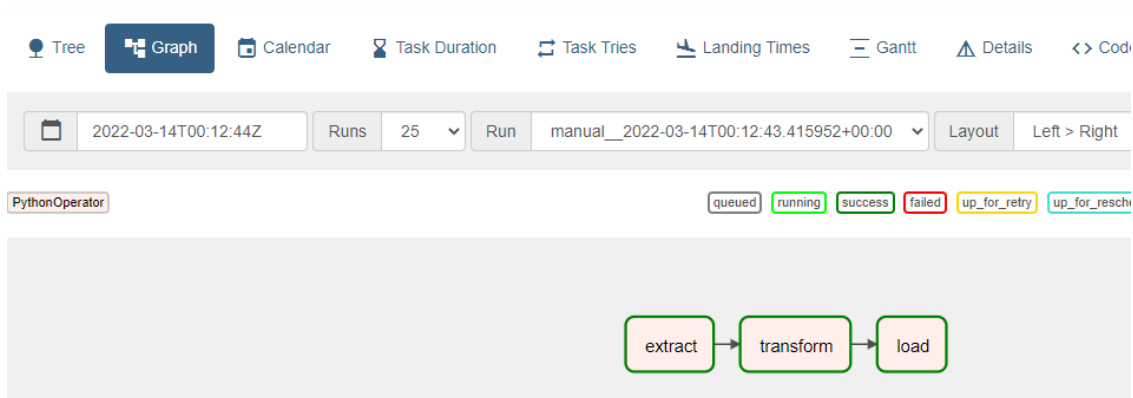
Al guardar el archivo este apareció en la interfaz de airflow:

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
<div><div></div><div>airflow_dag</div><div>airflow dag</div></div>	Diana	<div><div></div><div>4</div><div>4</div></div>	1 day, 0:00:00		2022-03-14, 00:00:00	<div><div></div><div></div><div></div><div></div><div></div><div>3</div></div>

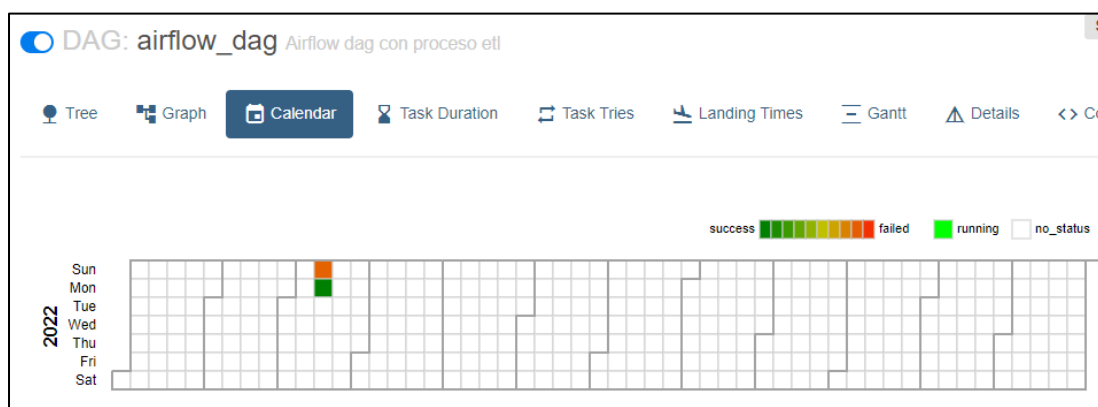
Al entrar a ese dag aparecen las veces que se ha ejecutado y sus resultados. Cada una de sus tareas tiene un color. El color verde indica que se ejecutó con éxito, el amarillo indica que ocurrió un error y se reintentó la tarea, y el rojo indica que aun así falló.



En vista de gráficos se puede ver un grafo con las tareas y sus dependencias, así como el estado en el que se encuentran, en este caso se encuentran en verde ya que la última ejecución fue exitosa:



En calendario se muestra cada cuándo se ejecutó el dag y el estatus de su ejecución:



También hay opciones para mostrar la duración de las tareas, los reintentos de las mismas, detalles del dag que se especificaron en el código, y el código en sí.

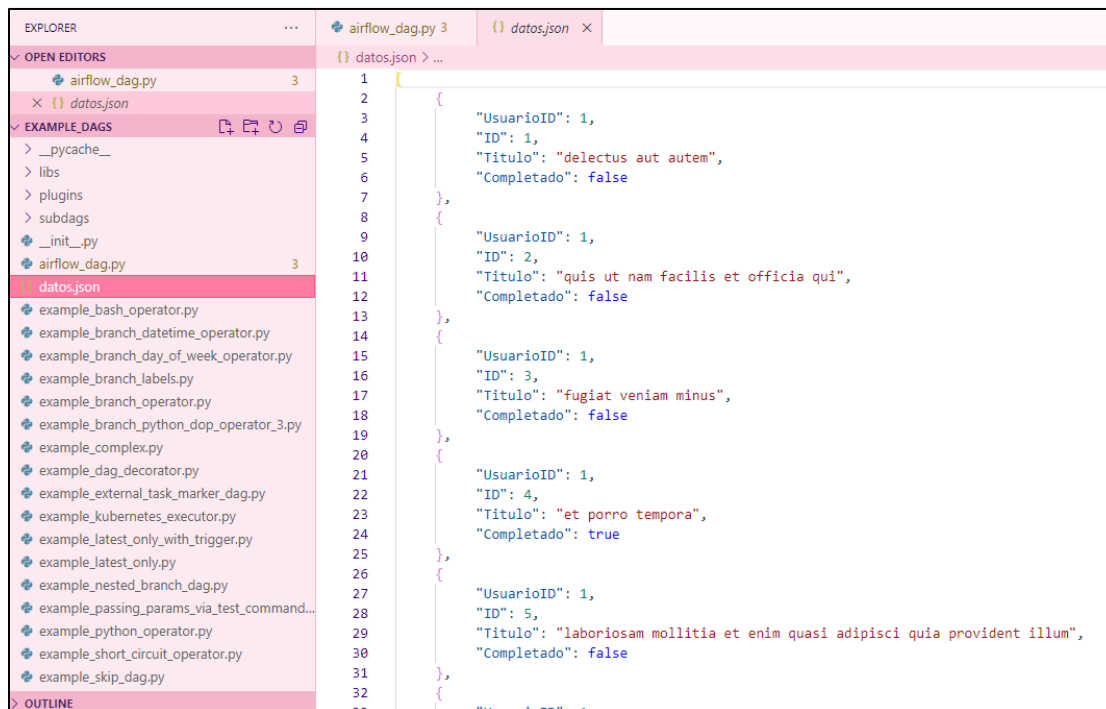
También se pueden ver los logs de cada una de las tareas para ver qué es lo que está pasando o pasó en cada una de ellas, por ejemplo la de transform:

```

*** Reading local file: /home/diana/airflow/logs/airflow_dag/transform/2022-03-14T03:24:14.592458+00:00/1.log
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: airflow_dag.transform manual__2022-03-14T03:24:14.592458+00:00 [queued]>
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1037} INFO - Dependencies all met for <TaskInstance: airflow_dag.transform manual__2022-03-14T03:24:14.592458+00:00 [queued]>
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1243} INFO -
-----
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1244} INFO - Starting attempt 1 of 6
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1245} INFO -
-----
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1264} INFO - Executing <Task(PythonOperator): transform> on 2022-03-14 03:24:14.592458+00:00
[2022-03-13, 21:28:38 UTC] {standard_task_runner.py:52} INFO - Started process 26485 to run task
[2022-03-13, 21:28:38 UTC] {standard_task_runner.py:76} INFO - Running: ['airflow', 'tasks', 'run', 'airflow_dag', 'transform', 'manual__2022-03-14T03:24:14.592458+00:00', '--job-id',
[2022-03-13, 21:28:38 UTC] {standard_task_runner.py:77} INFO - Job 49: Subtask transform
[2022-03-13, 21:28:38 UTC] {logging_mixin.py:109} INFO - Running <TaskInstance: airflow_dag.transform manual__2022-03-14T03:24:14.592458+00:00 [running]> on host user-PC.localdomain
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1429} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_EMAIL=diana.castillo7676@alumnos.udg.mx
AIRFLOW_CTX_DAG_OWNER=Diana
AIRFLOW_CTX_DAG_ID=airflow_dag
AIRFLOW_CTX_TASK_ID=transform
AIRFLOW_CTX_EXECUTION_DATE=2022-03-14T03:24:14.592458+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2022-03-14T03:24:14.592458+00:00
[2022-03-13, 21:28:38 UTC] {python.py:175} INFO - Done. Returned value was: [{'UsuarioID': 1, 'ID': 1, 'Titulo': 'delectat aut autem', 'Completado': False}, {'UsuarioID': 1, 'ID': 2,
[2022-03-13, 21:28:38 UTC] {taskinstance.py:1272} INFO - Marking task as SUCCESS. dag_id=airflow_dag, task_id=transform, execution_date=20220314T032414, start_date=20220314T032838, end
[2022-03-13, 21:28:38 UTC] {local_task_job.py:154} INFO - Task exited with return code 0
[2022-03-13, 21:28:38 UTC] {local_task_job.py:264} INFO - 1 downstream tasks scheduled from follow-on schedule check

```

Como las últimas ejecuciones fueron exitosas, se puede ver que se creó el archivo json en el mismo directorio que el dag y se guardaron los datos:



The screenshot shows an IDE interface with a file explorer on the left and a code editor on the right. The file explorer lists various Airflow DAG files under the 'EXAMPLE DAGS' section, including `airflow_dag.py`, `datos.json`, `example_bash_operator.py`, `example_branch_datetime_operator.py`, `example_branch_day_of_week_operator.py`, `example_branch_labels.py`, `example_branch_operator.py`, `example_branch_python_dop_operator_3.py`, `example_complex.py`, `example_dag_decorator.py`, `example_external_task_marker_dag.py`, `example_kubernetes_executor.py`, `example_latest_only_with_trigger.py`, `example_latest_only.py`, `example_nested_branch_dag.py`, `example_passing_params_via_test_command...`, `example_python_operator.py`, `example_short_circuit_operator.py`, and `example_skip_dag.py`. The code editor displays the content of `datos.json`, which is a JSON array of objects. Each object contains fields: `UsuarioID`, `ID`, `Título`, and `Completado`. The objects represent different tasks or data points, with varying titles and completion statuses.

```
1 [
2   {
3     "UsuarioID": 1,
4     "ID": 1,
5     "Título": "delectus aut autem",
6     "Completado": false
7   },
8   {
9     "UsuarioID": 1,
10    "ID": 2,
11    "Título": "quis ut nam facilis et officia qui",
12    "Completado": false
13  },
14  {
15    "UsuarioID": 1,
16    "ID": 3,
17    "Título": "fugiat veniam minus",
18    "Completado": false
19  },
20  {
21    "UsuarioID": 1,
22    "ID": 4,
23    "Título": "et porro tempora",
24    "Completado": true
25  },
26  {
27    "UsuarioID": 1,
28    "ID": 5,
29    "Título": "laboriosam mollitia et enim quasi adipisci quia provident illum",
30    "Completado": false
31  },
32  {
33    "UsuarioID": 1,
```

Código en GitHub: <https://github.com/diana-castillo/Airflow.git>

Conclusiones:

Esta actividad me pareció bastante interesante y un tanto diferente ya que tuve que trabajar en Ubuntu la instalación de airflow, así como el conceder permisos a algunas carpetas, lo cual fue nuevo para mí, fue un poco complicado entender algunos conceptos sobre lo mismo y sobre la manera de trabajar de airflow. Al principio me parecía semejante a prefect ya que ambos trabajan con flujos, pero a mi parecer airflow tiene más funcionalidades a explotar, además de que la interfaz de airflow tiene mucha información sobre los dags, lo cual ayuda a revisar el estatus de cada ejecución y si algo no funcionó como se esperaba, aparte de que se pueden ejecutar las tareas y observar lo que ocurre en el proceso.

Referencias:

Cardona, J. [Jorge Cardona]. (2021, 16 octubre). *Instalación, Configuración Apache Airflow - Windows sin Docker - Ubuntu, Creación y Ejecución DAGs* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=YPIX-qT6d8M>

Running Airflow locally. (s. f.). Apache Airflow. Recuperado 11 de marzo de 2022, de <https://airflow.apache.org/docs/apache-airflow/stable/start/local.html>

Feregrino – That C# guy. (2021, 11 julio). *Creando DAGs con AIRFLOW* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=i-JRuKaxL-4&t>