Workflow Managers

Computación Tolerante a Fallas

Alumno:

Código:

Profesor(a): Dr. Michel Emanuel López Franco

Sección: D06

Fecha de Entrega: 03 de octubre de 2022

Workflow Managers

Un sistema de gestión de flujo de trabajo es una herramienta que permite organizar la ejecución de una serie de tareas, de forma que estas pueden ser monitoreadas y supervisadas. Dentro de Python, existe un módulo denominado *prefect* con el que podemos crear flujos mediante los decoradores de funciones. Implemente una aplicación que es capaz de obtener imágenes de la API de Unsplash, descargarlas, y mostrarlas en pantalla al usuario, todo esto utilizando un Workflow Manager.

Lo primero que hice fue definir los temas de las imágenes que voy a descargar dentro de una lista, luego está la clave de la API necesaria para hacer las consultas, y la agenda de intervalos que nos permite ejecutar el flujo cada cierto tiempo (en este caso, un minuto). Para definir el tiempo, se utiliza la función *timedelta* de *datetime*.

```
# Temas de las imagenes.
unsplashQueries = ["history", "life", "wisdom", "art", "office", "nature"]

# Obtener la clave de la API.
dotenv.load_dotenv ()
apiKey = os.environ["UNSPLASH_API_KEY"]

# Creamos una agenda de intervalos, que se encargara de llamar al flujo cada cierto tiempo.
scheduleData = IntervalSchedule (interval=timedelta (minutes=1))
```

El objetivo es crear un flujo de tipo *ETL,* que significa extraer, transformar, y cargar. Para cada etapa, utilizamos una función distinta. La primera función, denominada *obtenerDatos*, se encarga de realizar la consulta a la API. Usando *random*, obtenemos un tema aleatorio y formateamos el URL para agregar el tema y la clave. Los datos deben ser convertidos a JSON y finalmente son retornados. En el decorador de función, se indica que es una tarea del flujo, y además que, en caso de error, deberá reintentar ejecutarse un total de 10 veces con intervalos de 15 segundos antes de terminar el programa.

```
# Extraer
@task(max_retries=10, retry_delay=timedelta (seconds=15))
def obtenerDatos ():
    # Solicitamos los datos de la imagen a la API.
    tema = random.choice (unsplashQueries)
    url = f"https://api.unsplash.com/photos/random?query={tema}&client_id={apiKey}"

    peticion = requests.get (url)

    datos = peticion.json()

    return datos
```

Después tenemos la función que transforma el objeto JSON obtenido de la consulta en solo una URL, de forma que la imagen puede ser descargada. Para hacerlo, simplemente nos basamos en la estructura de respuesta incluida en la documentación y accedemos a la propiedad “urls” en su campo “regular”. También se usa un decorador para indicar que es una tarea del flujo.

```python
# Transformar
@task
def transformarDatos (datos):
    # Obtenemos el url de la imagen.
    url = datos["urls"]["regular"]

    return url
```

Finalmente, tenemos la parte de cargar los datos a algún lugar. Dentro de la función *almacenarDatos* realizamos una nueva petición, pero esta vez, usando el URL de la imagen. Activamos la bandera *stream* para recibir los datos como sucesiones de bits que pueden ser escritos en un archivo, y de esa forma completamos la descarga. Verificamos si el código de estatus de la petición es distinto a 200 para lanzar una excepción o continuar con el programa. Finalmente, escribimos la sucesiones de bits en el archivo *imagen.jpg* y usamos *system* para ejecutar el archivo y que se muestre en pantalla.

```python
# Almacenar
@task
def almacenarDatos (imagen):
    # Para evitar tener muchas imagenes descargadas, sobreescribimos el mismo archivo.
    nombreArchivo = "imagen.jpg"
    peticionImagen = requests.get (imagen, stream=True)

    if (peticionImagen.status_code != 200):
        raise Exception ("Ocurrio un error al solicitar la imagen a la API.")

    with open (nombreArchivo, "wb") as imagenDescarga:
        for chunk in peticionImagen:
            imagenDescarga.write (chunk)

    os.system ("start imagen.jpg")
```

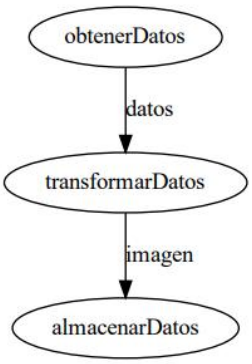
Lo único que resta es crear el flujo de trabajo, para lo que usamos el constructor de la clase *Flow* y le asignamos un nombre y pasamos como parámetro la agenda de intervalos. Dentro del flujo, llamamos las funciones en el orden correcto y después retornamos el flujo.

```python
def crearFlow (schedule):
    # Agregamos la agenda como parametro a Flow.
    with Flow ("ETL Flow", schedule) as flow:
        datos = obtenerDatos ()
        imagen = transformarDatos (datos)
        almacenarDatos (imagen)

    return flow
```

Ejecutamos el flujo y utilizamos la función *visualize* para crear un pequeño diagrama que muestra el orden de ejecución y los valores retornados.

```python
flow = crearFlow (scheduleData)
flow.visualize ()
flow.run ()
```



Al correr el programa, se muestra la siguiente información en la terminal, además de la imagen.

```
[2022-09-26 19:01:27-0500] INFO - prefect.ETL Flow | Waiting for next scheduled run at 2022-09-27T00:02:00+00:00
[2022-09-26 19:02:00-0500] INFO - prefect.FlowRunner | Beginning Flow run for 'ETL Flow'
[2022-09-26 19:02:00-0500] INFO - prefect.TaskRunner | Task 'obtenerDatos': Starting task run...
[2022-09-26 19:02:00-0500] INFO - prefect.TaskRunner | Task 'obtenerDatos': Finished task run for task with final state: 'Success'
[2022-09-26 19:02:00-0500] INFO - prefect.TaskRunner | Task 'transformarDatos': Starting task run...
[2022-09-26 19:02:00-0500] INFO - prefect.TaskRunner | Task 'transformarDatos': Finished task run for task with final state: 'Success'
[2022-09-26 19:02:00-0500] INFO - prefect.TaskRunner | Task 'almacenarDatos': Starting task run...
[2022-09-26 19:02:01-0500] INFO - prefect.TaskRunner | Task 'almacenarDatos': Finished task run for task with final state: 'Success'
[2022-09-26 19:02:01-0500] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-09-26 19:02:01-0500] INFO - prefect.ETL Flow | Waiting for next scheduled run at 2022-09-27T00:03:00+00:00
```

Conclusiones

Puedo decir que esta actividad ha sido mi favorita en lo que va del curso. Es bastante entretenido utilizar los workflow managers para crear aplicaciones que hagan una determinada secuencia de acciones en un momento especifico. Además, cuentan con una gran cantidad de herramientas para implementar la tolerancia a fallas, como la cantidad máxima de reintentos o que el programa pueda continuar ejecutándose después de que una de las ejecuciones del flujo falle.

La actividad fue realizada sin problemas gracias a la documentación proporcionada por el profesor y un poco más de investigación.