**Universidad de Guadalajara**
**Sistema de Educación Media Superior**
**Centro Universitario de Ciencias Exactas e Ingenierías**


# Workflow managers

Materia: Computación Tolerante a Fallas

D06 2023 B

Alumno: Esquivel Barbosa Diego Humberto

Código: 211401635

Carrera: INCO

Fecha: 02/10/2023

## Introducción

La computación tolerante a fallos se centra en el diseño, desarrollo y despliegue de sistemas que tienen la capacidad de resistir, mitigar y recuperarse de fallos y errores, manteniendo su funcionalidad esencial en situaciones adversas. Estos sistemas no solo aspiran a evitar la interrupción total, sino que también buscan ofrecer una operación confiable en escenarios donde componentes individuales pueden experimentar problemas. Ya sea en aplicaciones críticas para la seguridad, sistemas médicos, redes de comunicación o dispositivos de consumo, la tolerancia a fallos se ha convertido en un aspecto esencial para garantizar la integridad y la confianza en la tecnología que utilizamos diariamente.

Este código Python muestra un ejemplo de un proceso ETL (Extracción, Transformación y Carga) utilizando la biblioteca Prefect para automatizar el flujo de trabajo. El objetivo de este código es extraer datos de una fuente en línea, transformarlos y cargarlos en una base de datos local SQLite.

## Código

```
import requests
import json
from collections import namedtuple
from contextlib import closing
import sqlite3
from prefect import task, Flow

# Task para extraer datos
@task
def extract_data():
    url = "https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1"
    params = {'size': 10}
    response = requests.get(url, params=params)
    response_json = json.loads(response.text)
    return response_json['hits']['hits']

# Task para transformar datos
@task
def transform_data(raw_data):
    complaints = []
    Complaint = namedtuple('Complaint', ['data_received', 'state', 'product', 'company', 'complaint_what_happened'])

    for row in raw_data:
        source = row.get('_source')
        complaint = Complaint(
            data_received=source.get('date_received'),
            state=source.get('state'),
            product=source.get('product'),
            company=source.get('company'),
            complaint_what_happened=source.get('complaint_what_happened')
```

```
            )
            complaints.append(complaint)

        return complaints

    # Task para cargar datos
    @task
    def load_data(parsed_data):
        create_table_sql = '''
        CREATE TABLE IF NOT EXISTS complaint (
            timestamp TEXT,
            state TEXT,
            product TEXT,
            company TEXT,
            complaint_what_happened TEXT
        )
        '''
        insert_data_sql = '''
        INSERT INTO complaint
        VALUES (?, ?, ?, ?, ?)
        '''

        with closing(sqlite3.connect("cfpbcomplaints.db")) as conn:
            with closing(conn.cursor()) as cursor:
                cursor.executescript(create_table_sql)
                cursor.executemany(insert_data_sql, parsed_data)
                conn.commit()

    # Creación del flujo
    with Flow("CFPB Complaints ETL") as flow:
        raw_data = extract_data()
        parsed_data = transform_data(raw_data)
        load_data(parsed_data)

    # Ejecución del flujo
    flow.run()
```

**Desarrollo del Código**

El código se divide en tres tareas principales: extracción de datos, transformación de datos y carga de datos. Cada tarea se define como una función decorada con @task y se ejecuta en el contexto de un flujo de trabajo Prefect.

Tarea de Extracción de Datos
python
Copy code

```
@task
def extract_data():
    # Supongamos que tienes un archivo CSV como fuente de datos
    data = pd.read_csv("datos_pydata_denver.csv")
    return data
```

Esta tarea lee datos desde un archivo CSV llamado "datos_pydata_denver.csv" y devuelve el contenido como un DataFrame de Pandas.

Tarea de Transformación de Datos
python
Copy code

```
@task
def transform_data(data):
    # Realiza un análisis simple de los datos (por ejemplo, calcula estadísticas descriptivas)
    summary = data.describe()
    return summary
```

Esta tarea toma el DataFrame de la tarea de extracción y realiza una transformación simple, en este caso, calcula estadísticas descriptivas como el recuento, la media, la desviación estándar, etc.

Tarea de Carga de Datos
python
Copy code

```
@task
def load_data(summary):
    print("Resumen de datos para PyData Denver:")
    print(summary)
```

La tarea de carga simplemente imprime el resumen de datos en la consola.

**Funcionamiento**

El flujo de trabajo se crea utilizando with Flow("PyData Denver ETL") as flow: y las tareas se conectan en secuencia. Primero se extraen los datos, luego se transforman y finalmente se cargan.

El flujo se ejecuta con flow.run() en el bloque if __name__ == "__main__":, lo que significa que se ejecutará cuando se ejecute el archivo directamente.

## Conclusión

este código ejemplifica la importancia del proceso ETL en la gestión de datos, desde la extracción de datos de una fuente externa, pasando por la transformación en un formato estructurado hasta la carga en una base de datos local. La biblioteca Prefect facilita la administración de flujos de trabajo ETL y garantiza que las tareas se ejecuten en el orden correcto. Este programa puede servir como un punto de partida sólido para proyectos ETL más complejos en Python.

## Bibliografía


- Biblioteca Prefect: https://docs.prefect.io/
- Documentación de Python: https://docs.python.org/3/
- Documentación de SQLite: https://sqlite.org/docs.html