Arellano Granados Angel Mariano

218123444

Computación Tolerante a Fallas

D06 2023B

Workflow managers

Introducción

Prefect es una librería para el lenguaje de programación de Python la cual nos ayuda a evitar la programación negativa dejándole todo el control de fallos a una herramienta externa que traqueara, registrara y analizara los fallos que se presenten en nuestros programas, presentándonoslos en una interfaz simple y atractiva.

Desarrollo

Para esta actividad se realizo varios ejemplos de programas pequeños para probar la librería de Prefect y familiarizarnos con su estructura y funciones.

Este primer código es de un video de una clase muestran un ejemplo de Prefect en su primera versión, si se intenta ejecutar este código con Prefect2 este no funciona, sin embargo, haciendo 2 cambios uno en la línea donde importa la librería cambiando el “Flow” por “flow” y el otro es cambiar toda la declaración e iniciación del Flow, pues este cambio bastante con el paso de las versiones el código pasa de:

```
with Flow("my etl flow") as f:
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    store_complaints(parsed)

f.run()
```

A una versión compatible del mismo código, pero para Prefect2:

```
@flow(name="my etl flow")
def my_etl_flow():
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    store_complaints(parsed)

my_etl_flow._run()
```

Terminando con la siguiente ejecución:

```
PS C:\Users\Servidor\Documents\Trabajos 6 CUCEI\2. COMPUTACION TOLERANTE A FALLAS\Prefect> & C:/Users/Servidor/AppData/Local/Progra
. COMPUTACION TOLERANTE A FALLAS/Prefect/test2.py"
19:34:21.748 | INFO    | prefect.engine - Created flow run 'mighty-quokka' for flow 'my etl flow'
19:34:21.899 | INFO    | Flow run 'mighty-quokka' - Created task run 'get_complaint_data-0' for task 'get_complaint_data'
19:34:21.901 | INFO    | Flow run 'mighty-quokka' - Executing 'get_complaint_data-0' immediately...
19:34:27.881 | INFO    | Task run 'get_complaint_data-0' - Finished in state Completed()
19:34:27.943 | INFO    | Flow run 'mighty-quokka' - Created task run 'parse_complaint_data-0' for task 'parse_complaint_data'
19:34:27.945 | INFO    | Flow run 'mighty-quokka' - Executing 'parse_complaint_data-0' immediately...
19:34:28.147 | INFO    | Task run 'parse_complaint_data-0' - Finished in state Completed()
19:34:28.199 | INFO    | Flow run 'mighty-quokka' - Created task run 'store_complaints-0' for task 'store_complaints'
19:34:28.201 | INFO    | Flow run 'mighty-quokka' - Executing 'store_complaints-0' immediately...
19:34:28.386 | INFO    | Task run 'store_complaints-0' - Finished in state Completed()
19:34:28.453 | INFO    | Flow run 'mighty-quokka' - Finished in state Completed('All states completed.')
```

Código1:

```python
import requests
import json
from collections import namedtuple
from contextlib import closing
import sqlite3

from prefect import task, flow

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

## transform
@task
def parse_complaint_data(raw):
    complaints = []
    Complaint = namedtuple('Complaint', ['data_received', 'state',
'product', 'company', 'complaint_what_happened'])
    for row in raw:
        source = row.get('_source')
        this_complaint = Complaint(
            data_received=source.get('date_recieved'),
            state=source.get('state'),
            product=source.get('product'),
            company=source.get('company'),
            complaint_what_happened=source.get('complaint_what_happened')
        )
        complaints.append(this_complaint)
    return complaints
```

```
## load
@task
def store_complaints(parsed):
    create_script = 'CREATE TABLE IF NOT EXISTS complaint (timestamp TEXT,
state TEXT, product TEXT, company TEXT, complaint_what_happened TEXT)'
    insert_cmd = "INSERT INTO complaint VALUES (?, ?, ?, ?, ?)"

    with closing(sqlite3.connect("cfpbcomplaints.db")) as conn:
        with closing(conn.cursor()) as cursor:
            cursor.executescript(create_script)
            cursor.executemany(insert_cmd, parsed)
            conn.commit()

@flow(name="my etl flow")
def my_etl_flow():
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    store_complaints(parsed)

my_etl_flow._run()
```

El segundo programa es proporcionado en la misma pagina oficial de Prefect en github el cual hace una consulta a una pagina web y recolecta un dato que en este caso son estrellas, cabe resaltar que este programa funciona con la versión más nueva de Prefect 2

```
C:\Users\Servidor\AppData\Local\Programs\Python\Python311\Lib\contextlib.py:144: SAWarning: Skipped unsupported
  next(self.gen)
17:49:50.171 | INFO    | prefect.engine - Created flow run 'prudent-coua' for flow 'GitHub Stars'
17:49:50.318 | INFO    | Flow run 'prudent-coua' - Created task run 'get_stars-0' for task 'get_stars'
PrefectHQ/Prefect has 12930 stars!
17:49:51.445 | INFO    | Task run 'get_stars-0' - Finished in state Completed()
17:49:51.517 | INFO    | Flow run 'prudent-coua' - Finished in state Completed('All states completed.')
```

Codigo2:

```
from prefect import flow, task
from typing import List
import httpx


@task(retries=3)
def get_stars(repo: str):
    url = f"https://api.github.com/repos/{repo}"
    count = httpx.get(url).json()["stargazers_count"]
```

```
    print(f"{repo} has {count} stars!")


@flow(name="GitHub Stars")
def github_stars(repos: List[str]):
    for repo in repos:
        get_stars(repo)


# run the flow!
github_stars(["PrefectHQ/Prefect"])
```

Conclusión

Prefect es una de tantas herramientas que tenemos a nuestro alcance para gestionar el flujo de trabajo de nuestros programas, en nuestro ambiente como estudiantes tal vez este tipo de herramientas no sean tan atractivas para nosotros ya que nuestros programas son cortos con vidas de ejecución cortas, sin embargo, en el futuro trabajaremos con programas enormes que necesiten estar el ejecución por horas o días para que el usuario lo use cuando sea necesario, en situaciones como esta Prefect nos ofrece una simple solución para que nuestros programas no sufran de errores fatales que o que al menos no deje de estar en ejecución, para nosotros como programadores logremos encontrar el fallo y solucionarlo lo antes posible.

Referencias

- https://github.com/PrefectHQ/prefect
- https://www.prefect.io