FAULT TORERANCE

CALENDARIO 2022

MURILLO CORTES, JEANETTE

COMPUTACIÓN TOLERANTE A FALLAS

DR. MICHEL EMANUEL LÓPEZ FRANCO

SECCIÓN D06

LUNES – MIÉRCOLES
11:00 – 1:00PM

LUNES ---- DE MARZO, 2022

GUADALAJARA, JALISCO

WORKFLOW MANAGERS

ARCHIVO DEL TUTORIAL DEL USO DE WORKFLOW MANAGERS

En este documento se describe el Tutorial de:

1. Desarrollo del ejemplo propuesto en clase para el uso del ejemplo de Workflow Managers
2. Cómo se implementó el ejemplo para la entrega de la práctica.

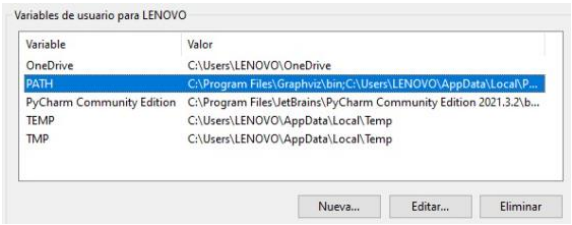
Un flujo de trabajo es un conjunto de pasos repetibles. Workflow Managers es una interfaz que administra flujos de trabajos para que se puedan realizar de manera automatizada. Sirve para que tareas que se tienen que realizar diariamente puedan realizarse automáticamente mediante un software.

Para ello, se realiza la representación gráfica de las tareas que se realizarán con las tareas que se han determinado programar.

WORKFLOW

PREINSTALACIONES

Para el desarrollo de esta práctica como prerrequisito se necesita tener instalado Python.



INSTALACIONES

- Graphviz. Link de descarga: https://graphviz.org/download/. Para su correcto funcionamiento es necesario editar las variables de usuario.
- **Prefect.** Para su instalación, en la Consola de Python escribir los siguientes comandos para la instalación de prefect:

```
pip install prefect
pip install "prefect[all_extras]"
```

DESARROLLO DE LA PRÁCTICA

Para el desarrollo de un Workflow mediante el lenguaje Python, se utiliza Prefect para la gestión del flujo de trabajo. programan prefect con decoradores:

```
from prefect import task, Flow #Import


@task #Decorator
def hello_world():
      print("Hello world!")


with Flow("my first flow!") as f:
      r = hello_world()


f.run() # Call run method
```

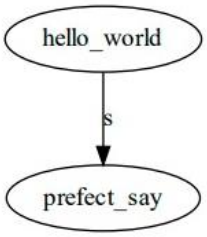
3) Después de escribir el código anterior en el archivo, lo pegamos en la consola de Python y damos *Enter* para continuar, de manera que debe de mandar el siguiente mensaje:

```
Have to go to <Sucess "all references tasks succeded.">
```

La manera de observar el código es poniéndolo en la consola de comandos consola de Python, de manera que se obtengan los siguientes mensajes:

```
| Beginning Flow run for 'my first flow!'
| Task 'hello_world': Starting task run...
| Task 'hello_world': Finished task run for task with final state: 'Success'
| Task 'prefect_say': Starting task run...
Hello Prefect!
| Task 'prefect_say': Finished task run for task with final state: 'Success'
| Flow run SUCCESS: all reference tasks succeeded
<Success: "All reference tasks succeeded.">
```

Nota: si por ejemplo queremos observar el grafo del Workflow, podemos llamar al método '`f.visualize()`', mediante la importación y descarga de la librería graphviz, que permite generar grafos a partir de código. De esta manera se generó el grafo que se muestra en la imagen de la derecha.



En el uso de un procesamiento de datos ETL, primeramente, importamos librerías para los diccionarios, JSON, solicitudes que se solicitarán, además de las librerías de Prefect que serán requeridas para la práctica. Por su parte, en la configuración de la tabla que se crea para el ETL, se nombra a la base de datos y se pasan los atributos que tendrán default. Cada decorador del código es donde se encuentran las funciones y handelers que se mencionan a continuación.

En la parte de transform se crea una gran lista con sus parámetros en donde se almacenan los datos que son analizados y que se van a retornar para cargarlos a la base de datos. Esta función, al igual que también la anterior y la que sigue, tienen un **`handler`** que checa el estado del manejador para observar si no existe algún fallo.

Para la carga de los datos, los datos se añaden en una tabla que se genera pasando los datos obtenidos a la base de datos en SQL Lite, e insertándolos mediante `insert_cmd` utilizando la conexión sqlite3 y ejecutando el script de esta función que genera la tabla, para así tener cargados los nuevos datos que han sido analizados.

Por último, escribimos y llamamos las funciones que contienen las tareas que serán realizadas por el flujo de trabajo, de manera que se pueda realizar la extracción (‘`get_complaint_data`’), transformación (‘`parse_complaint_data`’) y carga de los datos (‘`store_complaints`’).

```
from venv import create
import requests
import json
from collections import namedtuple
from contextlib import closing
import sqlite3
import datetime

from prefect import task, Flow
from prefect.tasks.database.sqlite import SQLiteScript
from prefect.schedules import IntervalSchedule
from prefect.engine import signals
from prefect.engine.result_handlers import LocalResultHandler

def alert_failed(obj, old_state, new_state):
    if new_state.is_failed():
        print("Failed!")


## Setup
create_table = SQLiteScript(
    db = 'cfpbcomplaints.db',
    script = 'CREATE TABLE IF NOT EXISTS complaint (timestamp TEXT, state TEXT, company TEXT,
complaint_what_happended TEXT)'
)

## Extract
@task(cache_for=datetime.timedelta(days=1), state_handlers=[alert_failed],
result_handler=LocalResultHandler())
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(state_handlers=[alert_failed])
```

```
def parse_complaint_data(raw):
    raise signals.SUCCESS
    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(state_handlers=[alert_failed])
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()


with Flow("my etl Flow", state_handlers=[alert_failed]) as f:
    db_table = create_table()
    raw = get_complaint_data()
    parse_complaint_data(raw)
    populated_table = store_complaints(parsed)
    populated_table.set_upstream(db_table)

f.run()
```

Para su ejecución cuando vamos a la terminal, ejecutamos el comando **`prefect backend server`** y el comando **`prefect server star --postgres -port 8080`** para comenzar a trabajar con Prefect y ejecutar el programa, de manera que al ejecutarse el código manda los mensajes donde se pueden observar los estados del flujo de trabajos. Toda la información acerca del estado actual de las tareas se muestra en la consola.

Si el estado es 'Retraying' es que el estado tiene error, lo cuál es señalado como rojo. Si el estado es 'Pending' es que se esta pendiente la tarea de ejecutarse y el estado se pone de color amarillo. Por ultimo, si es estado es verde, entonces la tarea se ha completado con éxito y la marcará como 'Success' como se muestra en la siguiente ejecución del programa de la práctica.

```
prefect.FlowRunner | Beginning Flow run for 'etl_flow'
prefect.TaskRunner | Task 'parametro_url': Starting task run...
prefect.TaskRunner | Task 'parametro_url': Finished task run for task with final state:
'Success'
prefect.TaskRunner | Task 'extract': Starting task run...
prefect.TaskRunner | Task 'extract': Finished task run for task with final state: 'Success'
prefect.TaskRunner | Task 'transform': Starting task run...
```

```
prefect.TaskRunner | Task 'transform': Finished task run for task with final state: 'Success'
prefect.TaskRunner | Task 'load': Starting task run...
prefect.TaskRunner | Task 'load': Finished task run for task with final state: 'Success'
prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
      <Success: "All reference tasks succeeded.">
prefect.etl_flow | Waiting for next scheduled run at 2022-04-20-07T02:36:00+00:00
prefect.FlowRunner | Beginning Flow run for 'etl_flow'
prefect.TaskRunner | Task 'parametro_url': Starting task run...
prefect.TaskRunner | Task 'parametro_url': Finished task run for task with final state:
'Success'
prefect.TaskRunner | Task 'extract': Starting task run...
prefect.TaskRunner | Task 'extract': Finished task run for task with final state: 'Success'
prefect.TaskRunner | Task 'transform': Starting task run...
prefect.TaskRunner | Task 'transform': Finished task run for task with final state: 'Success'
prefect.TaskRunner | Task 'load': Starting task run...
prefect.TaskRunner | Task 'load': Finished task run for task with final state: 'Success'
prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
```

Observando que los estados de todas las tareas programadas dentro del día fueron cumplidos, según como nos muestran sus estados.

CONCLUSIÓN

El uso de un Workflow para la ejecución de distintas tareas lo hace una herramienta bastante útil para la programación de grandes cantidades de trabajo, por eso la implementación de esta práctica con el uso de Prefect nos permite observar cómo administrar interna y automáticamente estas tareas de una manera sencilla y por periodos de tiempo. De hecho, como el mismo programa marca el estado de la ejecución de una tarea, esto nos permite saber en qué momento hay un error, lo cuál nos ayuda a observar dónde se pueden, en un futuro, implementar soluciones que sean tolerantes a estas fallas y de esta manera completar todas las tareas que se tengan pensadas programar en el flujo, haciendo un programa más robusto.

BIBLIOGRAFÍA

- Laura L. (Abril, 2020). *"Getting Started with Prefect (PyData Denver)"*. Recuperado el 05/03/22. Obtenido del URL: https://www.youtube.com/watch?v=FETN0iivZps&t=2545s
- *"¿Qué es un workflow y para qué le sirve a tu empresa?"* - INTEGRADOC BPM. Recuperado el 28/03/22. Obtenido del URL: https://www.integradoc.com/que-es-un-workflow/
- *"Graphviz"*. Recuperado el 20/03/22. Obtenido del URL: https://graphviz.org/download/