

Workflow managers

(Código con prefect)



Gutiérrez Villalobos Dayal

Código: 216586382

Computación tolerante a fallos

Lunes y miércoles: 11:00-12:55

Sección D06

Maestro: López Franco Michel Emanuel

Lo primero que se realizó para poder empezar a crear el programa fue la instalación de la herramienta prefect. Esto se realizó con el siguiente comando

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Versión 10.0.19043.1526]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Dayal>pip install prefect
```

Así como también el mismo procedimiento para la librería requests para poder hacer operaciones http, sin embargo, como yo ya tenía instalada esa librería previamente no fue necesario poner el código.

Después de tener todas las librerías correspondientes para que se pudiera trabajar se procedió a realizar los apartados del script. En este caso el primer paso fue definir todos los apartados del ETL empezando con el apartado de extraer, después el de modificar y por último el de cargar, teniendo un flujo ya bien definido se declara de la siguiente manera

```
with Flow("my etl flow",schedule) as f:
    doc = create_document()
    raw = get_complaint_data()
    modificados = parse_complaint_data(raw)
    #Lo que se hace al pasar raw o modificados es que se ponen
    #como requerimiento
    populated_table = store_complaints(modificados)
```

Cómo se podrá observar en la imagen de abajo en el decorador de la tarea para obtener los datos se le agregaron unos parámetros, los cuales entran en función cuando la tarea pertinente falla, poniendo en espera los procesos siguientes del flujo

```
#Extraer
@task(max_retries=10, retry_delay=datetime.timedelta(seconds=30))
def get_complaint_data():
    r = requests.get("https://www.consumerfinance.gov/data-research/consumer-com
    response_json = json.loads(r.text)
    return response_json['hits']['hits']
```

```

return request('get', url, params=params, **kwargs)
File "C:\Users\Dayal\AppData\Local\Programs\Python\Python38\lib\site-packages\requests\api.py", line 61, in request
return session.request(method=method, url=url, **kwargs)
File "C:\Users\Dayal\AppData\Local\Programs\Python\Python38\lib\site-packages\requests\sessions.py", line 529, in request
resp = self.send(prepared_request, **kwargs)
File "C:\Users\Dayal\AppData\Local\Programs\Python\Python38\lib\site-packages\requests\sessions.py", line 645, in send
r = adapter.send(request, **kwargs)
File "C:\Users\Dayal\AppData\Local\Programs\Python\Python38\lib\site-packages\requests\adapters.py", line 519, in send
raise ConnectionError(e, request=request)
requests.exceptions.ConnectionError: HTTPSConnectionPool(host='www.coumerfinance.gov', port=443): Max retries exceeded with url: /data-research/consumer-complaints/search/api/v1/?size=10 (Caused by NewConnectionError('<urllib3.connection.HTTPSConnection object at 0x00000196462E49A0>: Failed to establish a new connection: [Errno 11001] getaddrinfo failed'))
[2022-03-06 23:56:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Retrying'
[2022-03-06 23:56:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-06 23:56:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Pending'
[2022-03-06 23:56:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-06 23:56:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Pending'
[2022-03-06 23:56:00-0600] INFO - prefect.FlowRunner | Flow run RUNNING: terminal tasks are incomplete.

```

Haciendo que nuestro programa funcione a diferentes niveles, este tipo de intentos se codifica con el propósito de poder evitar que nuestro programa deje de funcionar en caso de que el repositorio donde se obtienen los datos no esté disponible por alguna razón este pasó se ejecutará un máximo de 10 veces con 30 segundos entre cada intento.

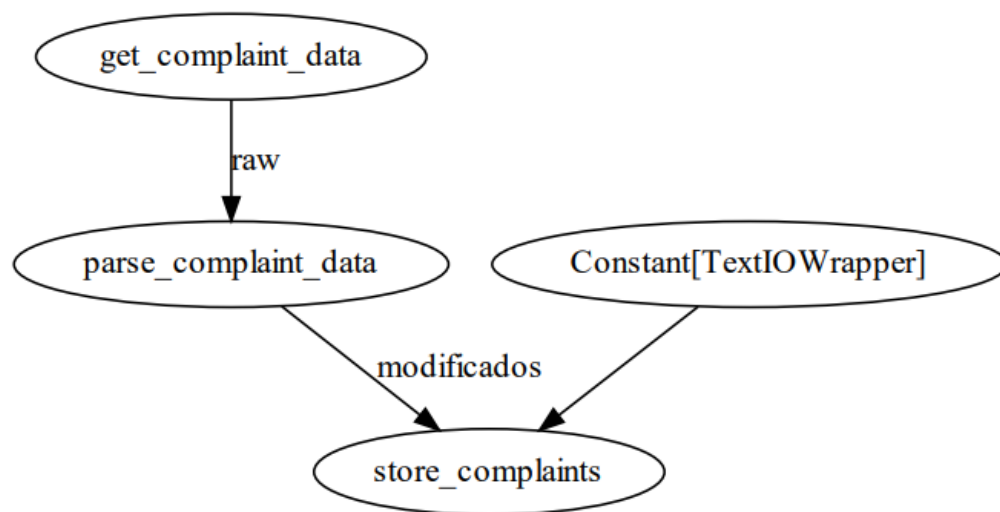
Además de tener ese tipo de protección, se declaran ciertas dependencias de diferentes maneras para hacer que nuestro programa sepa que necesita ciertas cosas para poder manejar todo de una manera correcta

```

with Flow("etl flow",schedule) as f:
    doc = create_document()
    raw = get_complaint_data()
    modificados = parse_complaint_data(raw)
    #Lo que se hace al pasar raw o modificados es que se ponen
    #como requerimiento
    populated_table = store_complaints(modificados)
    populated_table.set_upstream(doc)
    #La instrucción de arriba nos indica que
    #primero debe haber ocurrido la creación de un archivo
    #para poder llenarlo con la información

```

Haciendo que todo el programa siga un recorrido con ciertas especificaciones y ciertos previos procesos para poder seguir con los siguientes, es como se muestra en la siguiente imagen



Por último, para poder hacer que nuestro programa sea ejecutado más de una vez a través de programaciones de tiempo, se volvió a utilizar la herramienta de IntervalSchedule para hacer que nuestro flujo corra cada minuto

```
schedule = IntervalSchedule(interval=datetime.timedelta(minutes=1))
```

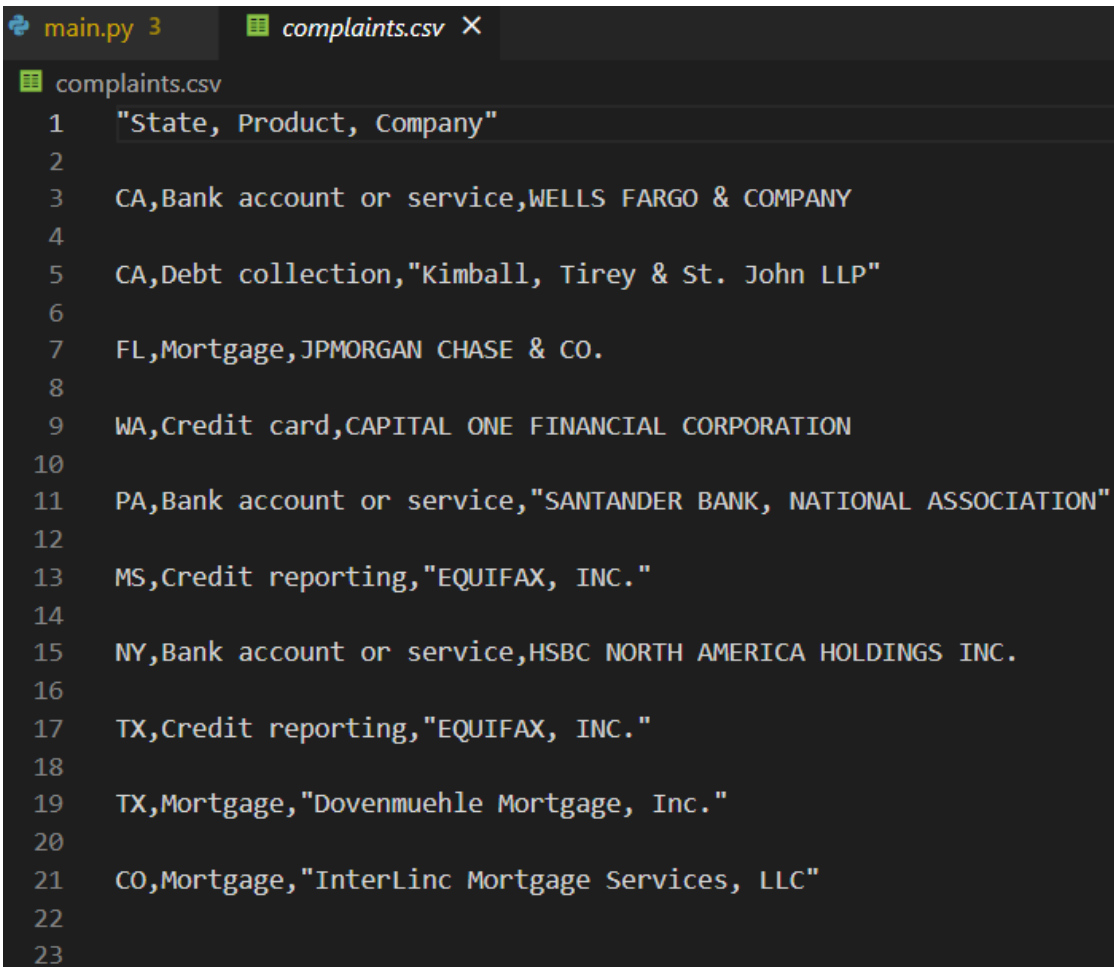
Que como se observa en la imagen de abajo, se recorre todo el flujo cada minuto

```

[2022-03-07 08:12:00-0600] INFO - prefect.etl flow | Waiting for next scheduled run at 2022-03-07T14:13:00+00:00
[2022-03-07 08:13:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'etl flow'
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 08:13:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 08:13:00-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 08:13:00-0600] INFO - prefect.etl flow | Waiting for next scheduled run at 2022-03-07T14:14:00+00:00
[2022-03-07 08:14:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'etl flow'
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 08:14:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 08:14:00-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 08:14:00-0600] INFO - prefect.etl flow | Waiting for next scheduled run at 2022-03-07T14:15:00+00:00
[2022-03-07 08:15:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'etl flow'
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 08:15:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 08:15:00-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
  
```

Para poder terminar guardando todos los datos ya modificados en un archivo .csv que se actualiza cada vez sobrescribiendo los datos

```
#Cargar
@task
def store_complaints(modificados):
    file = open('complaints.csv','w')
    csv_writer = csv.writer(file)
    title = ('State, Product, Company',)
    csv_writer.writerow(title)
    for complain in modificados:
        csv_writer.writerow(complain[1:-1])
    file.close()
    return
```



```
main.py 3  complaints.csv X
complaints.csv
1  "State, Product, Company"
2
3  CA,Bank account or service,WELLS FARGO & COMPANY
4
5  CA,Debt collection,"Kimball, Tirey & St. John LLP"
6
7  FL,Mortgage,JPMORGAN CHASE & CO.
8
9  WA,Credit card,CAPITAL ONE FINANCIAL CORPORATION
10
11 PA,Bank account or service,"SANTANDER BANK, NATIONAL ASSOCIATION"
12
13 MS,Credit reporting,"EQUIFAX, INC."
14
15 NY,Bank account or service,HSBC NORTH AMERICA HOLDINGS INC.
16
17 TX,Credit reporting,"EQUIFAX, INC."
18
19 TX,Mortgage,"Dovenmuehle Mortgage, Inc."
20
21 CO,Mortgage,"InterLinc Mortgage Services, LLC"
22
23
```

Repositorio

<https://github.com/DayalGutierrez/Workflow-manager-Prefect-.git>

Conclusión

Utilizar una herramienta como prefect me dio otra perspectiva sobre cómo es que se pueden llegar a programar los flujos teniendo un mayor control sobre todos los procesos que se llevan a cabo en un programa, incluso creo que ayuda a poder dividir de una mejor manera los programas en tareas que cumplan con propósitos dividiendo de una manera efectiva para que se puedan complementar entre todas las tareas con el propósito de tener un programa que funcione correcto. Sin embargo, creo que es muy útil cuando hay proyectos grandes sobre los que se deben de tener mucho cuidado en cada parte del programa debido a que tienen dependencias las tareas, pero para proyectos muy pequeños siento que puede llegar a no ser tan eficiente. Aunque todas las herramientas que están incluidas en la librería pueden llegar a ser muy útiles dependiendo del proyecto.

Bibliografía

Radečić, D. (2022, 6 enero). Prefect: How to Write and Schedule Your First ETL Pipeline with Python. Medium. <https://towardsdatascience.com/prefect-how-to-write-and-schedule-your-first-etl-pipeline-with-python-54005a34f10b>