

Workflow managers

Alumno: Oswaldo Gael Cervantes Castoño

Código: 219747468

Computación Tolerante a Fallas

Sección D06

Profesor: Michel Emanuel López Franco



Objetivo:

Seguir el tutorial siguiente y ejecutarlo en tu máquina.

Desarrollo:

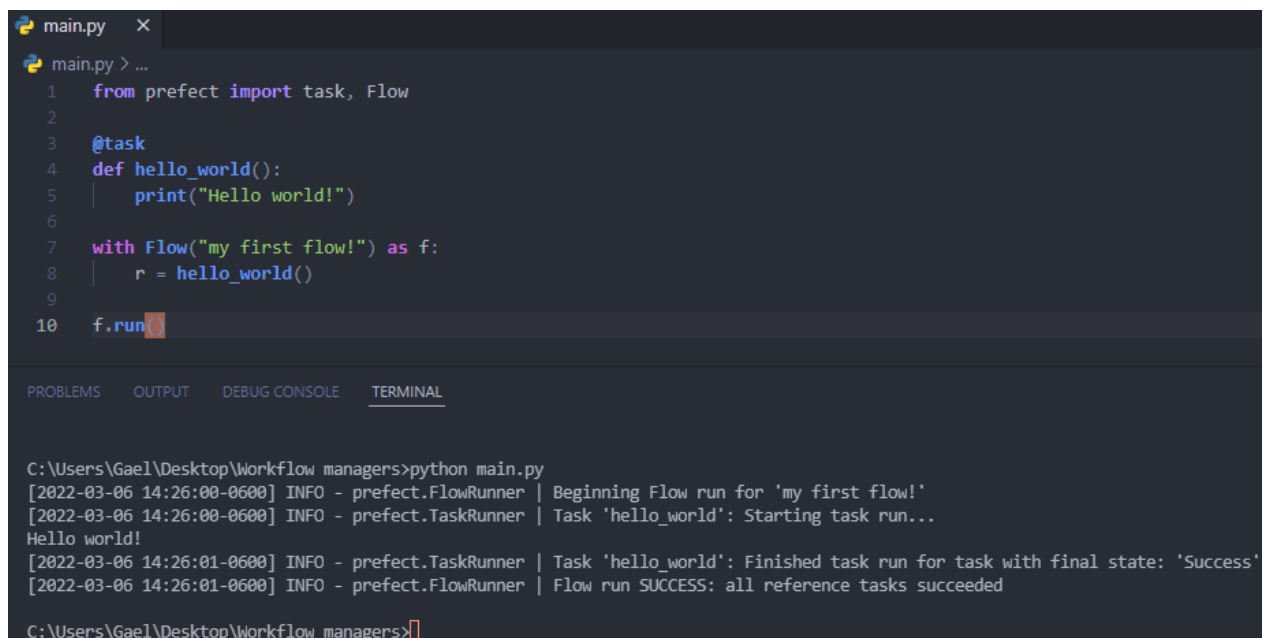
Para la realización de esta actividad se hizo uso del módulo “prefect” en Python, el cual permite generar y combinar tareas para posteriormente darle un seguimiento a su flujo de trabajo y poder determinar los puntos específicos en los que un programa funciona de forma correcta o presenta alguna anomalía que se necesite corregir.

El primer paso para poder utilizar “prefect” es instalarlo desde la línea de comandos de nuestra preferencia; en mi caso yo utilicé la línea de comandos embebida en Visual Studio Code por medio del siguiente comando:

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

C:\Users\Gael\Desktop\Workflow managers>pip install prefect
```

Importando “task” y “Flow” podemos generar un ejemplo simple el cual consiste en este caso en una función que se llama “hello_world” y simplemente realiza la impresión de un mensaje, dicha función utiliza un decorador para indicar que se trata de una tarea; con “Flow” generamos el flujo que dará seguimiento a la ejecución de las tareas:



```
main.py x
main.py > ...
1  from prefect import task, Flow
2
3  @task
4  def hello_world():
5      print("Hello world!")
6
7  with Flow("my first flow!") as f:
8      r = hello_world()
9
10 f.run()

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\Gael\Desktop\Workflow managers>python main.py
[2022-03-06 14:26:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my first flow!'
[2022-03-06 14:26:00-0600] INFO - prefect.TaskRunner | Task 'hello_world': Starting task run...
Hello world!
[2022-03-06 14:26:01-0600] INFO - prefect.TaskRunner | Task 'hello_world': Finished task run for task with final state: 'Success'
[2022-03-06 14:26:01-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded

C:\Users\Gael\Desktop\Workflow managers>
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Ahora definimos una segunda tarea que recibe un parámetro retornado desde la primera función; esto puede decirse que genera una cierta dependencia entre las funciones contempladas en el flujo:

```
main.py > hello_world
1  from prefect import task, Flow
2
3  @task
4  def hello_world():
5      print("Hello world!")
6      return "Hello Prefect!"
7
8  @task
9  def prefect_say(s: str):
10     print(s)
11
12  with Flow("my first flow!") as f:
13     r = hello_world()
14     s2 = prefect_say(r)
15
16  f.run()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Users\Gael\Desktop\Workflow managers>python main.py
[2022-03-06 14:56:22-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my first flow!'
[2022-03-06 14:56:22-0600] INFO - prefect.TaskRunner | Task 'hello_world': Starting task run...
Hello world!
[2022-03-06 14:56:22-0600] INFO - prefect.TaskRunner | Task 'hello_world': Finished task run for task with final state: 'Success'
[2022-03-06 14:56:22-0600] INFO - prefect.TaskRunner | Task 'prefect_say': Starting task run...
Hello Prefect!
[2022-03-06 14:56:22-0600] INFO - prefect.TaskRunner | Task 'prefect_say': Finished task run for task with final state: 'Success'
[2022-03-06 14:56:22-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
C:\Users\Gael\Desktop\Workflow managers>
```

Para poder visualizar de forma más clara la representación de esta dependencia generada, podemos cambiar el método de “run” por “visualize” para generar un archivo pdf que contiene un grafo con las tareas que definimos anteriormente; para lo anterior necesitamos instalar “Graphviz” tanto en la computadora como en Python con el comando “pip” y añadir al path los archivos para que se ejecute correctamente.

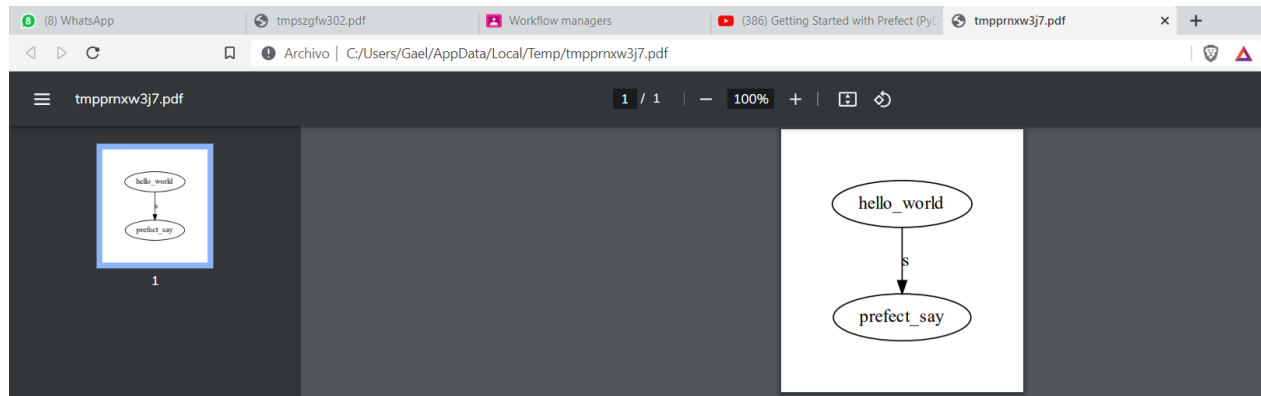
```
16  f.visualize()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
D:\Computacion tolerante a fallas\Workflow managers>
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

El archivo pdf se genera de forma automática y se ve de la siguiente manera:



Para trabajar en un ejemplo un poco más complejo relacionado con la definición, asociación y seguimiento de tareas, se propone desarrollar el siguiente ejemplo, el cual consiste en realizar una petición a una API. Con el código que se muestra abajo, podemos imprimir el texto devuelto como respuesta del método “get”.

```
File Edit Selection View Go Run Terminal Help
main.py - Workflow managers - Visual Studio Code

main.py x
main.py > ...
1 import requests
2
3 r = requests.get("https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/", params={'size':10})
4
5 print(r.text)
```

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
{"took":1055,"timed out":false,"shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":{"value":2514963,"relation":"eq"},"max score":null,"hits":[{"_index":"complaint-public-v1","type":"doc","id":"99999","score":1.0,"source":{"product":"Bank account or service","complaint_what_happened":"","date sent to company":"2013-11-18T12:00:00-05:00","issue":"Deposits and withdrawals","sub_product":"Checking account","zip code":"94605","tags":["Older American","has narrative":false,"complaint_id":"99999","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with explanation","submitted via":"Phone","company":"WELLS FARGO & COMPANY","date received":"2012-06-12T12:00:00-05:00","state":"CA","consumer_disputed":"No","company_public_response":null,"sub_issue":null},"sort":[1.0,"99999"]},{"_index":"complaint-public-v1","type":"doc","id":"99998","score":1.0,"source":{"product":"Debt collection","complaint_what_happened":"","date sent to company":"2014-09-04T12:00:00-05:00","issue":"Communication tactics","sub_product":"I do not know","zip code":"93305","tags":null,"has narrative":false,"complaint_id":"99998","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed","submitted via":"Web","company":"Kimball, Tiley & St. John LLP","date received":"2014-08-25T12:00:00-05:00","state":"CA","consumer_disputed":"No","company_public_response":null,"sub_issue":"Used obscene/profane/abusive language"},"sort":[1.0,"99998"]},{"_index":"complaint-public-v1","type":"doc","id":"99996","score":1.0,"source":{"product":"Mortgage","complaint_what_happened":"","date sent to company":"2014-08-25T12:00:00-05:00","issue":"Application, originator, mortgage broker","sub_product":"Conventional fixed mortgage","zip code":"22181","tags":null,"has narrative":false,"complaint_id":"99996","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with explanation","submitted via":"Phone","company":"JP Morgan Chase & Co.","date received":"2014-08-25T12:00:00-05:00","state":"FL","consumer_disputed":"Yes","company_public_response":null,"sub_issue":null},"sort":[1.0,"99996"]},{"_index":"complaint-public-v1","type":"doc","id":"99998","score":1.0,"source":{"product":"Credit card","complaint_what_happened":"","date sent to company":"2012-06-15T12:00:00-05:00","issue":"Advertising and marketing","sub_product":null,"zip code":"98221","tags":["Older American","has narrative":false,"complaint_id":"99998","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with explanation","submitted via":"Web","company":"CAPITAL ONE FINANCIAL CORPORATION","date received":"2012-06-12T12:00:00-05:00","state":"VA","consumer_disputed":"No","company_public_response":null,"sub_issue":null},"sort":[1.0,"99998"]},{"_index":"complaint-public-v1","type":"doc","id":"99997","score":1.0,"source":{"product":"Bank account or service","complaint_what_happened":"","date sent to company":"2012-06-21T12:00:00-05:00","issue":"Problems caused by my funds being low","sub_product":"Checking account","zip code":"19140","tags":["Older American","Servicemember","has narrative":false,"complaint_id":"99997","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed","submitted via":"Postal mail","company":"SANTANDER BANK, NATIONAL ASSOCIATION","date received":"2012-06-12T12:00:00-05:00","state":"PA","consumer_disputed":"No","company_public_response":null,"sub_issue":null},"sort":[1.0,"99997"]},{"_index":"complaint-public-v1","type":"doc","id":"99996","score":1.0,"source":{"product":"Credit reporting","complaint_what_happened":"","date sent to company":"2014-08-25T12:00:00-05:00","issue":"Unable to get credit report/credit score","sub_product":null,"zip code":"99503","tags":null,"has narrative":false,"complaint_id":"99996","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with non-monetary relief","submitted via":"Web","company":"EQUIFAX, INC.","date received":"2014-08-25T12:00:00-05:00","state":"WV","consumer_disputed":"No","company_public_response":null,"sub_issue":"Problem getting my free annual report"},"sort":[1.0,"99996"]},{"_index":"complaint-public-v1","type":"doc","id":"99996","score":1.0,"source":{"product":"Bank account or service","complaint_what_happened":"","date sent to company":"2012-06-18T12:00:00-05:00","issue":"Account opening, closing, or management","sub_product":"Savings account","zip code":"11375","tags":["Older American","has narrative":false,"complaint_id":"99996","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with monetary relief","submitted via":"Phone","company":"HESC NORTH AMERICA HOLDINGS INC.","date received":"2012-06-12T12:00:00-05:00","state":"NY","consumer_disputed":"No","company_public_response":null,"sub_issue":null},"sort":[1.0,"99996"]},{"_index":"complaint-public-v1","type":"doc","id":"99998","score":1.0,"source":{"product":"Credit reporting","complaint_what_happened":"","date sent to company":"2014-09-23T12:00:00-05:00","issue":"Incorrect information on credit report","sub_product":null,"zip code":"76124","tags":null,"has narrative":false,"complaint_id":"99998","timely":"Yes","consumer consent_provided":"N/A","company_response":"Closed with explanation","submitted via":"Postal mail","company":"EQUIFAX, INC.","date received":"2014-08-25T12:00:00-05:00","state":"TX","consumer_disputed":"No","company_public_response":null,"sub_issue":"Information is not mine"},"sort":[1.0,"99998"]},{"_index":"complaint-public-v1","type":"doc","id":"99995","score":1.0,"source":{"product":"Mortgage","complaint_what_happened":"","date sent to company":"2014-08-28
```

Las líneas de código siguientes ejemplifican algunas de las maneras en las que podemos acceder a la información contenida en la respuesta obtenida por medio de las claves y valores que manejan los archivos con formato json:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
main.py x
main.py > ...
1 import requests
2 import json
3
4 r = requests.get("https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/", params={'size':10})
5
6 response_json = json.loads(r.text)
7 print(response_json.keys())
8 print(response_json["hits"].keys())
9 print(response_json["hits"]["hits"])

dict keys(['took', 'timed_out', 'shards', 'hits', 'aggregations', 'meta'])
dict keys(['total', 'max_score', 'hits'])
[{'_index': 'complaint-public-v1', '_type': 'doc', '_id': '99999', '_score': 1.0, '_source': {'product': 'Bank account or service', 'complaint_what_happened': '', 'date_sent_to_company': '2013-11-18T12:00:00-05:00', 'issue': 'Deposits and withdrawals', 'sub_product': 'Checking account', 'zip_code': '94605', 'tags': 'Older American', 'has_narrative': False, 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': None, 'sort': [1.0, '99999']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '99998', '_score': 1.0, '_source': {'product': 'Debt collection', 'complaint_what_happened': '', 'date_sent_to_company': '2014-09-04T12:00:00-05:00', 'issue': 'Communication tactics', 'sub_product': 'I do not know', 'zip_code': '93305', 'tags': None, 'has_narrative': False, 'complaint_id': '999988', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed', 'submitted_via': 'Web', 'company': 'Kimball, Tiney & St. John LLP', 'date_received': '2014-08-25T12:00:00-05:00', 'state': 'CA', 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': 'Used obscene/profane/abusive language', 'sort': [1.0, '99998']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '999986', '_score': 1.0, '_source': {'product': 'Mortgage', 'complaint_what_happened': '', 'date_sent_to_company': '2014-08-28T12:00:00-05:00', 'issue': 'Application, origination, mortgage broker', 'sub_product': 'Conventional fixed mortgage', 'zip_code': '32818', 'tags': None, 'has_narrative': False, 'complaint_id': '999986', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed with explanation', 'submitted_via': 'Phone', 'company': 'JP MORGAN CHASE & CO.', 'date_received': '2014-08-25T12:00:00-05:00', 'state': 'FL', 'consumer_disputed': 'Yes', 'company_public_response': None, 'sub_issue': None, 'sort': [1.0, '999986']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '99998', '_score': 1.0, '_source': {'product': 'Credit card', 'complaint_what_happened': '', 'date_sent_to_company': '2012-08-15T12:00:00-05:00', 'issue': 'Advertising and marketing', 'sub_product': None, 'zip_code': '98221', 'tags': 'Older American', 'has_narrative': False, 'complaint_id': '99998', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed with explanation', 'submitted_via': 'Web', 'company': 'CAPITAL ONE FINANCIAL CORPORATION', 'date_received': '2012-06-12T12:00:00-05:00', 'state': 'WA', 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': None, 'sort': [1.0, '99998']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '99997', '_score': 1.0, '_source': {'product': 'Bank account or service', 'complaint_what_happened': '', 'date_sent_to_company': '2012-06-21T12:00:00-05:00', 'issue': 'Problems caused by my funds being low', 'sub_product': 'Checking account', 'zip_code': '19140', 'tags': 'Older American, Servicemember', 'has_narrative': False, 'complaint_id': '99997', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed', 'submitted_via': 'Postal mail', 'company': 'SANTANDER BANK, NATIONAL ASSOCIATION', 'date_received': '2012-06-12T12:00:00-05:00', 'state': 'PA', 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': None, 'sort': [1.0, '99997']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '999966', '_score': 1.0, '_source': {'product': 'Credit reporting', 'complaint_what_happened': '', 'date_sent_to_company': '2014-08-25T12:00:00-05:00', 'issue': 'Unable to get credit report/credit score', 'sub_product': None, 'zip_code': '39503', 'tags': None, 'has_narrative': False, 'complaint_id': '999966', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed with non-monetary relief', 'submitted_via': 'Web', 'company': 'EQUIFAX, INC.', 'date_received': '2014-08-25T12:00:00-05:00', 'state': 'MS', 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': 'Problem getting my free annual report', 'sort': [1.0, '999966']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '99996', '_score': 1.0, '_source': {'product': 'Bank account or service', 'complaint_what_happened': '', 'date_sent_to_company': '2012-06-18T12:00:00-05:00', 'issue': 'Account opening, closing, or management', 'sub_product': 'Savings account', 'zip_code': '11375', 'tags': 'Older American', 'has_narrative': False, 'complaint_id': '99996', 'timely': 'Yes', 'consumer_consent_provided': 'N/A', 'company_public_response': 'Closed with monetary relief', 'submitted_via': 'Phone', 'company': 'HSBC NORTH AMERICA HOLDINGS INC.', 'date_received': '2012-06-12T12:00:00-05:00', 'state': 'NY', 'consumer_disputed': 'No', 'company_public_response': None, 'sub_issue': None, 'sort': [1.0, '99996']}], {'_index': 'complaint-public-v1', '_type': 'doc', '_id': '999958', '_score': 1.0, '_source': {'pr
```

Ahora el código anterior se define en una tarea por medio del decorador “@task”:

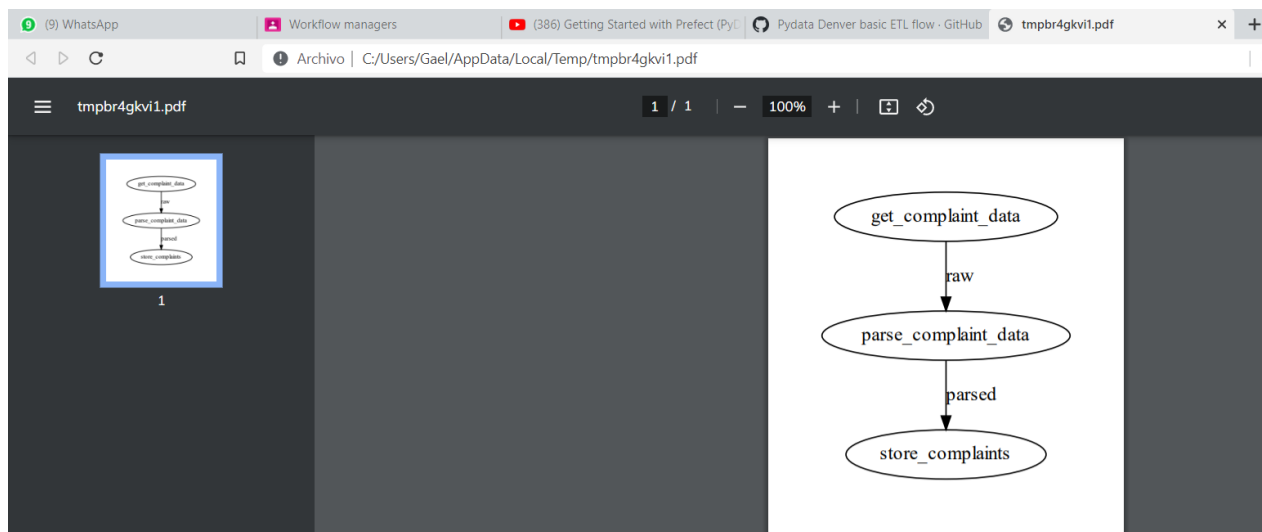
```
main.py x
main.py > ...
1 from prefect import task, Flow
2 import requests
3 import json
4
5 #extract
6 @task
7 def get_complaint_data():
8     r = requests.get("https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/", params={'size':10})
9     response_json = json.loads(r.text)
10     return response_json["hits"]["hits"]
11
12 #transform
13 @task
14 def parse_complaint_data(raw):
15     pass
16
17 #Load
18 @task
19 def store_complaints(parsed):
20     pass
21
22 with Flow("my etl flow") as f:
23     raw = get_complaint_data()
24     parsed = parse_complaint_data(raw)
25     store_complaints(parsed)
26
27 f.visualize()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
D:\Computacion tolerante a fallas\Workflow managers>python main.py
D:\Computacion tolerante a fallas\Workflow managers>
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

En la imagen anterior también se define la estructura de otras dos tareas (“parse_complaint_data” y “store_complaints”); así mismo se define un flujo que almacena en una variable la información que se obtiene de la API, misma variable se pasa a la tarea “complaint_data” que se almacena en una segunda variable que finalmente se pasa como parámetro a “store_complaints”.

Ejecutamos el código con “visualize” para observar las dependencias generadas y los datos involucrados obteniendo así el siguiente resultado:



La función “parse_complaint_data” itera sobre la información obtenida desde la petición a la API para posteriormente retornar un valor correspondiente a una lista después de tratar con la información:

```
#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
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

La función “store_complaints” obtiene la información devuelta por la función descrita en la imagen anterior para posteriormente crear una base de datos en la cual se almacenará la información que se recibe; para ello es necesario importar SQLite en Python y finalmente generar un commit como se hace con cualquier SGBD cuando se utiliza desde un lenguaje de programación:

```
#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()
```

Ejecutamos nuestro código con “run” y podemos observar que las 3 tareas que tienen una dependencia en cadena y además involucran transferencia de datos para su ejecución terminan con “Success” del mismo modo que la ejecución del flujo:

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
[2022-03-07 13:39:38-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 13:39:41-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 13:39:41-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 13:39:41-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 13:39:41-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 13:39:41-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 13:39:41-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
D:\Computacion tolerante a fallas\Workflow managers>
```

Con el código que acontece puede ejemplificarse que pueden generarse dependencias que no involucran transferencia de datos, simplemente se establece que algunas funciones ocurran antes que otras, en este caso por medio del uso de la librería “task”:

```
from prefect.tasks.database.sqlite import SQLiteScript

#setup
create_table = SQLiteScript(
    db = "cfpbcomplaints.db",
    script = "CREATE TABLE IF NOT EXISTS complaint (timestamp TEXT, state TEXT, product TEXT, company TEXT, complaint_what_happened TEXT)"
)
```

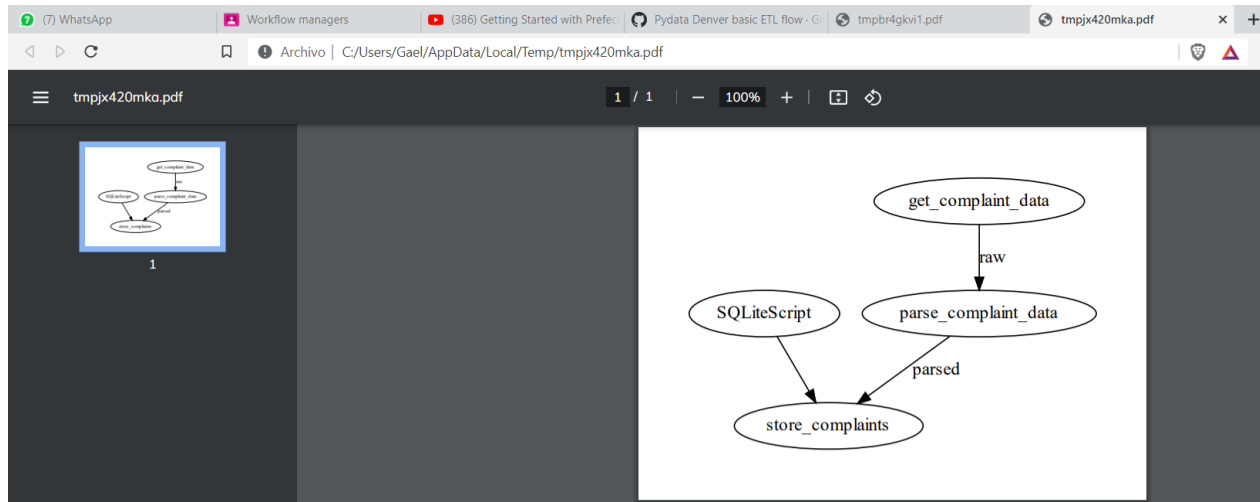
```
with Flow("my etl flow") as f:
    db_table = create_table()
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    populated_table = store_complaints(parsed)
    populated_table.set_upstream(db_table)

f.visualize()
```


Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Con las imágenes anteriores observamos que se genera un script de SQLite para generar la base de datos en primer lugar antes de que se ejecuten el resto de las funciones, posteriormente se procede a modificar el flujo definido al final del código y generamos la dependencia de funciones apropiada en cascada para que los eventos ocurran en el orden esperado.

Ejecutamos el código con “visualize” y observamos el siguiente resultado:



El siguiente paso propuesto en el tutorial es generar un “calendario” para programar cada cierto tiempo la ejecución de alguna tarea, para lo cual se necesitan importar las siguientes librerías:

```
import datetime
from prefect.schedules import IntervalSchedule
```

Antes de nuestro flujo se define el objeto “IntervalSchedule”:

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

with Flow("my etl flow", schedule) as f:
    db_table = create_table()
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    populated_table = store_complaints(parsed)
    populated_table.set_upstream(db_table)

f.run()
```


Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Podemos observar que en la parte de parámetros se define su temporizador para cada minuto y también observamos que se le pasa como parámetro a nuestro objeto “Flow”.

Ejecutamos y podemos observar la obtención de los siguientes resultados:

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
[2022-03-07 14:21:50-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:22:00+00:00
[2022-03-07 14:22:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 14:22:02-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:02-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 14:22:02-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:23:00+00:00
```

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
[2022-03-07 14:21:50-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:22:00+00:00
[2022-03-07 14:22:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 14:22:02-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 14:22:02-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 14:22:02-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:23:00+00:00
[2022-03-07 14:23:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Starting task run...
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Finished task run for task with final state: 'Success'
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:23:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 14:23:02-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 14:23:02-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 14:23:02-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:24:00+00:00
```

Podemos observar en las imágenes anteriores que el “calendario” vuelve a programar una nueva ejecución dentro de un minuto cada vez.

A continuación, se ejemplifica como se podría ajustar un “calendario” de forma interna por medio de “cache_for” el cual se define en el decorador “@task” de la función deseada. En este caso definimos el intervalo de tiempo para cada día:

```
#extract
@task(cache_for=datetime.timedelta(days=1))
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)
    print("I actually requested this time!")
    return response_json["hits"]["hits"]
```

Podemos observar que la tarea que definimos con “cache_for” se muestra en estado “cached” y además las tareas del flujo que requieren de la tarea “get_complaint_data”

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

toman la información que se obtuvo en la primera llamada, no en la más actual, puesto que volverá a retornar información hasta el día siguiente:

```
D:\Computacion tolerante a fallas\Workflow managers>python main.py
[2022-03-07 14:25:54-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:26:00+00:00
[2022-03-07 14:26:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Starting task run...
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Finished task run for task with final state: 'Success'
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 14:26:00-0600] WARNING - prefect.TaskRunner | Task 'get_complaint_data': Can't use cache because it is now invalid
I actually requested this time!
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Cached'
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:26:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 14:26:01-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 14:26:01-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 14:26:01-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:27:00+00:00
[2022-03-07 14:27:00-0600] INFO - prefect.FlowRunner | Beginning Flow run for 'my etl flow'
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Starting task run...
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'SQLiteScript': Finished task run for task with final state: 'Success'
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Starting task run...
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'get_complaint_data': Finished task run for task with final state: 'Cached'
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Finished task run for task with final state: 'Success'
[2022-03-07 14:27:00-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2022-03-07 14:27:02-0600] INFO - prefect.TaskRunner | Task 'store_complaints': Finished task run for task with final state: 'Success'
[2022-03-07 14:27:02-0600] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-07 14:27:02-0600] INFO - prefect.my etl flow | Waiting for next scheduled run at 2022-03-07T20:28:00+00:00
```

Lo siguiente que se propone en el tutorial son los “state handlers” o manejadores/administradores de estado, para lo cual tuve complicaciones con la definición de algunos parámetros en un archivo de configuración para el reconocimiento de los parámetros involucrados en la función y no pude lograr esta parte de forma satisfactoria, aunque lo pude documentar:

```
#state handlers
def alert_failed(obj, old_state, new_state):
    if new_state.isfailed():
        print("Failed!")
```

La novedad es que se define en el decorador de “@task” un parámetro llamado “state_handlers” con el cual se pasa como parámetro la función que definimos anteriormente para detectar fallos o anomalías en la ejecución de alguna de las tareas comprendidas en nuestro código:

```
#extract
@task(cache_for=datetime.timedelta(days=1), state_handlers = [alert_failed])
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)
    print("I actually requested this time!")
    return response_json["hits"]["hits"]
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Para lo anterior fue necesario provocar un fallo en el código, en este caso por medio de generar una excepción:

```
#transform
@task(state_handlers = [alert_failed])
def parse_complaint_data(raw):
    raise Exception
    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
```

Se define en todas y cada una de las tareas el parámetro “state_handlers” en el decorador de la tarea:

```
#load
@task(state_handlers = [alert_failed])
def store_complaints(parsed):
    insert_cmd = "INSERT INTO complaint VALUES (?, ?, ?, ?, ?)"

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

Como mencioné, no pude ejecutar esta parte de forma satisfactoria, pero el resultado tiene que verse como lo siguiente:

```
[2020-04-16 00:04:00,070] INFO - prefect.TaskRunner | Task 'get_complaint_data': finished task run for task with final state: 'Cached'
[2020-04-16 00:04:00,113] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...
[2020-04-16 00:04:00,127] INFO - prefect.TaskRunner | Task 'parse_complaint_data': finished task run for task with final state: 'Success'
[2020-04-16 00:04:00,161] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...
[2020-04-16 00:04:00,173] INFO - prefect.TaskRunner | Task 'store_complaints': finished task run for task with final state: 'Success'
[2020-04-16 00:04:00,174] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2020-04-16 00:04:00,174] INFO - prefect.Flow: my etl flow | Waiting for next scheduled run at 2020-04-16T00:05:00+00:00
Traceback (most recent call last):
  File "/Users/laura/Library/Preferences/PyCharm2019.3/scratches/pydata_denver_demo.py", line 63, in <module>
    f.run()
  File "/Users/laura/Development/prefect/src/prefect/core/flow.py", line 1025, in run
    state = self._run_on_schedule()
  File "/Users/laura/Development/prefect/src/prefect/core/flow.py", line 871, in _run_on_schedule
    time.sleep(naptime)
KeyboardInterrupt

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Donde se genera la excepción podemos ver que se ejecuta el bloque de código definido en la función “alert_failed()”:

```
I actually requested this time!  
[2020-04-16 00:11:18,000] INFO - prefect.TaskRunner | Task 'get_complaint_data': finished task run for task with final state: 'Cached'  
[2020-04-16 00:11:18,019] INFO - prefect.TaskRunner | Task 'parse_complaint_data': Starting task run...  
[2020-04-16 00:11:18,019] ERROR - prefect.TaskRunner | Unexpected error: Exception()  
Traceback (most recent call last):  
  File "/Users/laura/Development/prefect/src/prefect/engine/runner.py", line 48, in inner  
    new_state = method(self, state, *args, **kwargs)  
  File "/Users/laura/Development/prefect/src/prefect/engine/task_runner.py", line 883, in get_task_run_state  
    result = timeout_handler(  
  File "/Users/laura/Development/prefect/src/prefect/utilities/executors.py", line 182, in timeout_handler  
    return fn(*args, **kwargs)  
  File "/Users/laura/Library/Preferences/PyCharm2019.3/scratches/pydata_denver_demo.py", line 34, in parse_complaint_data  
    raise Exception  
Exception  
Failed!  
[2020-04-16 00:11:18,032] INFO - prefect.TaskRunner | Task 'parse_complaint_data': finished task run for task with final state: 'Failed'  
[2020-04-16 00:11:18,058] INFO - prefect.TaskRunner | Task 'store_complaints': Starting task run...  
Failed!
```

Observamos que también debe definirse el parámetro de administradores de estado en nuestro flujo (objeto “Flow”):

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

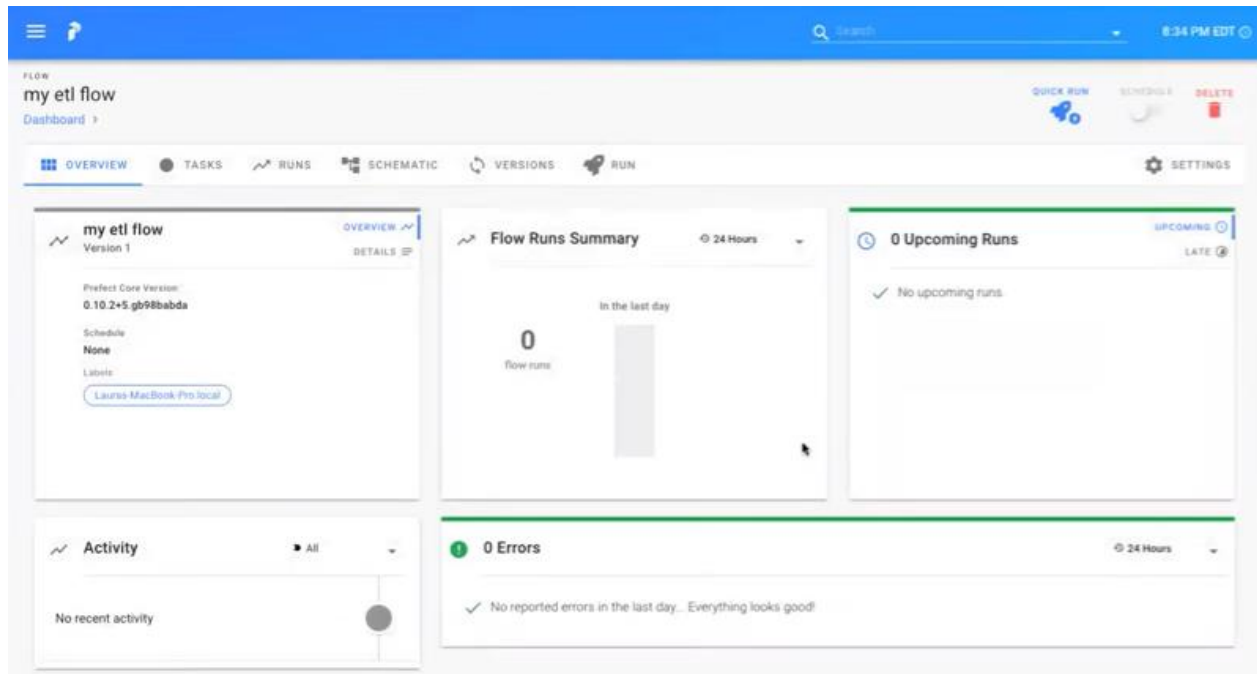
Otra forma sugerida de hacer esto en el tutorial es importando “signals” desde “prefect.engine” y definir el tipo de señal a nuestro gusto; en este caso es “SUCCESS”:

```
from prefect.engine import signals
```

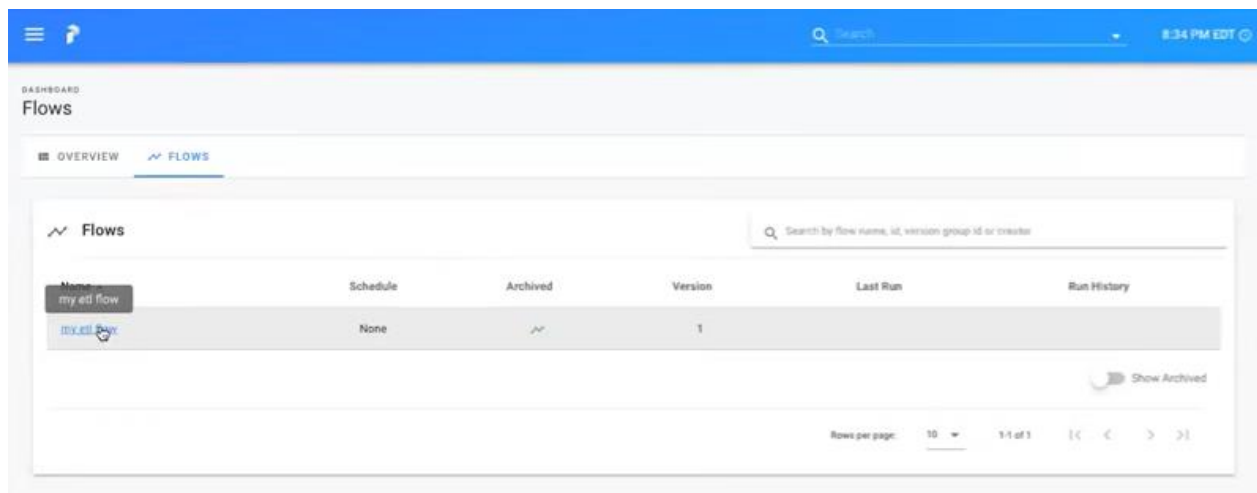
```
#transform  
@task(state_handlers = [alert_failed])  
def parse_complaint_data(raw):  
    raise signals.SUCCESS  
    complaints = []  
    complaint = namedtuple('Complaint', ['data_received', 'state', 'product', 'company', 'complaint_what_happened'])  
    for row in raw:  
        source = row.get('_source')  
        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
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

En la parte final del tutorial se propone únicamente utilizar una interfaz generada en el servidor local, con la cual podemos gestionar nuestros flujos de trabajo de una forma más amigable:

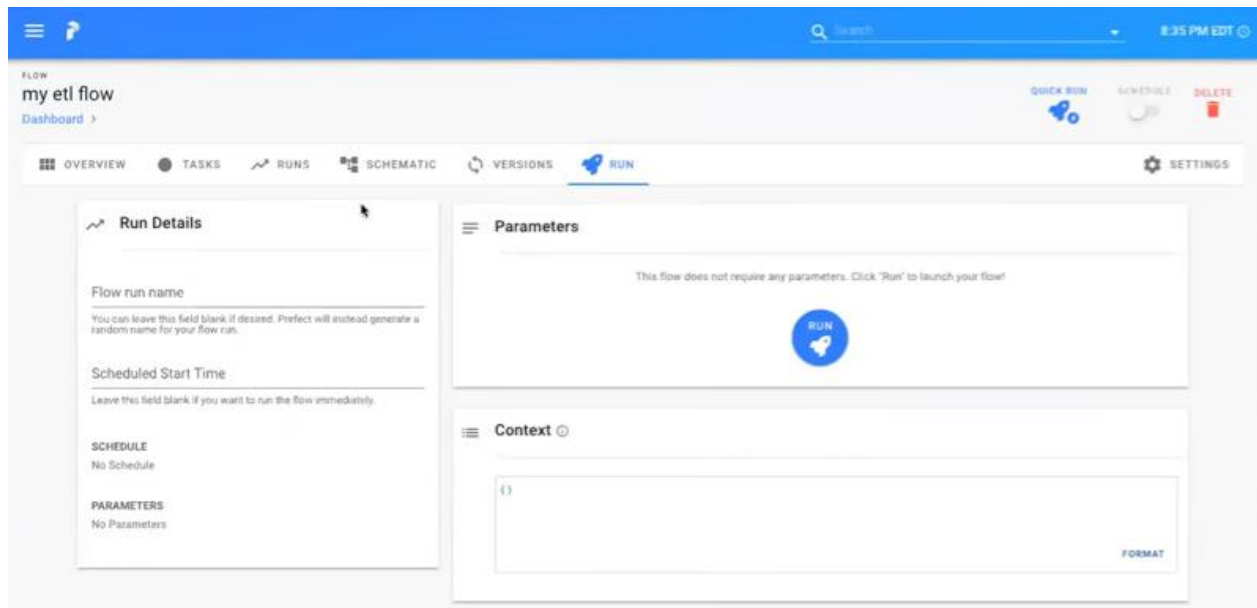


Podemos observar los flujos de trabajo que hemos generado en el apartado de “Flows” que se muestra a continuación:

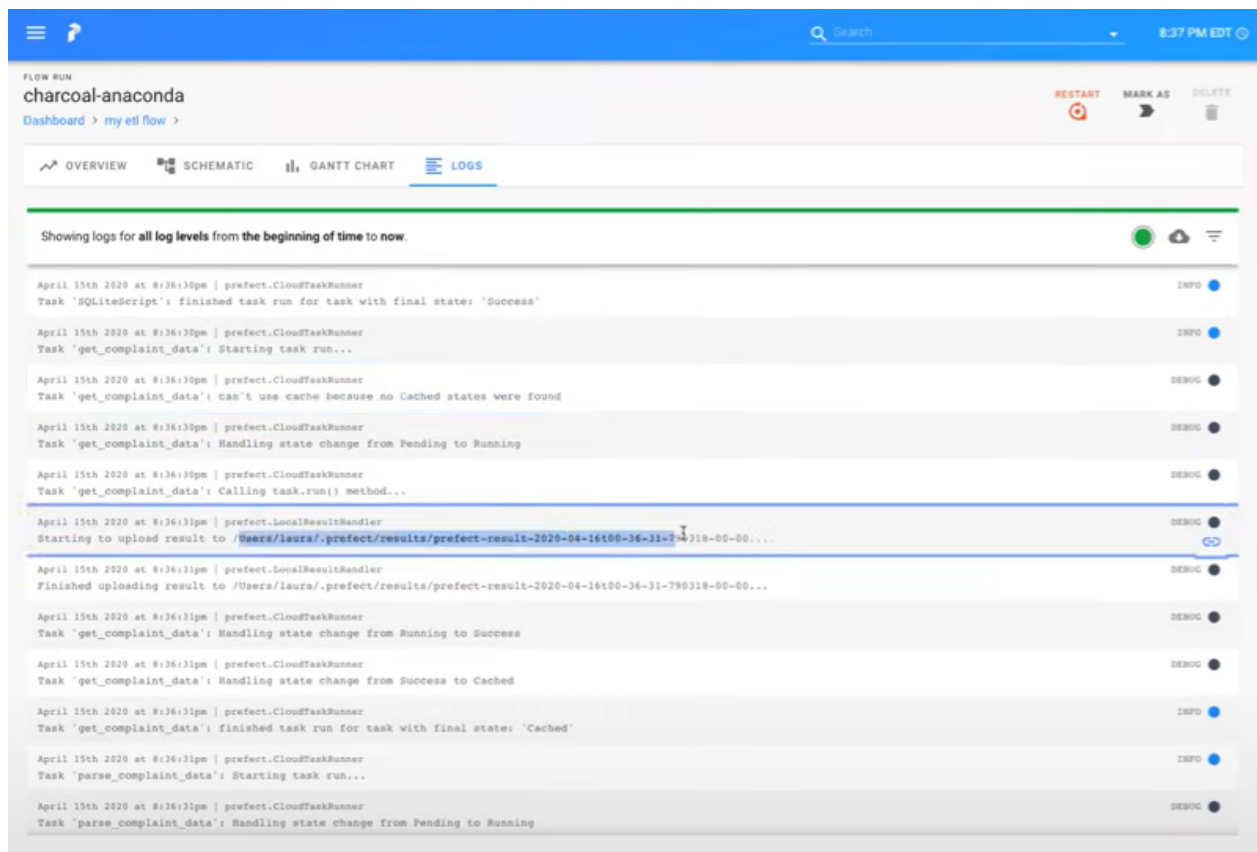


Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Podemos ejecutar nuestros flujos de trabajo en la siguiente ventana con solo hacer clic sobre un botón:



Finalmente podemos observar el estado de las tareas dentro de un flujo:



Codificación

Link al repositorio con el código fuente: <https://github.com/gaeltcervantes65/Workflow-managers.git>

Conclusiones

Como conclusiones para la realización de esta actividad puedo mencionar que considero muy valioso el haber aprendido sobre el uso de este módulo para la administración de los flujos de trabajo de tareas que podemos definir como funciones que utilizamos todo el tiempo cuando codificamos cualquier cosa; desconocía que existieran este tipo de herramientas y los alcances que pueden tener para facilitar el trabajo de un programador en el sentido de que se invierte menos tiempo en tratar de encontrar errores de cualquier tipo puesto que los flujos dan información detallada de cada función que se le asocia por medio del decorador “@task”. Tuve algunas dificultades en el camino y afortunadamente supe gestionar cada una de ellas y creo que me llevo cosas muy valiosas de lo que realicé en esta ocasión.

Considero que aprendí muchas cosas por medio de la realización de esta actividad y espero que este conocimiento me sirva para la realización de actividades futuras en lo que respecta a este curso.

Fuentes de información

- «*RuntimeError: Make sure the Graphviz executables are on your system's path*» after installing Graphviz 2.38. (2016, 28 enero). Stack Overflow. Recuperado 7 de marzo de 2022, de <https://stackoverflow.com/questions/35064304/runtimeerror-make-sure-the-graphviz-executables-are-on-your-systems-path-aft>
- P. (2020, 17 abril). Getting Started with Prefect (PyData Denver) [Vídeo]. YouTube. <https://www.youtube.com/watch?v=FETN0iivZps&feature=youtu.be>