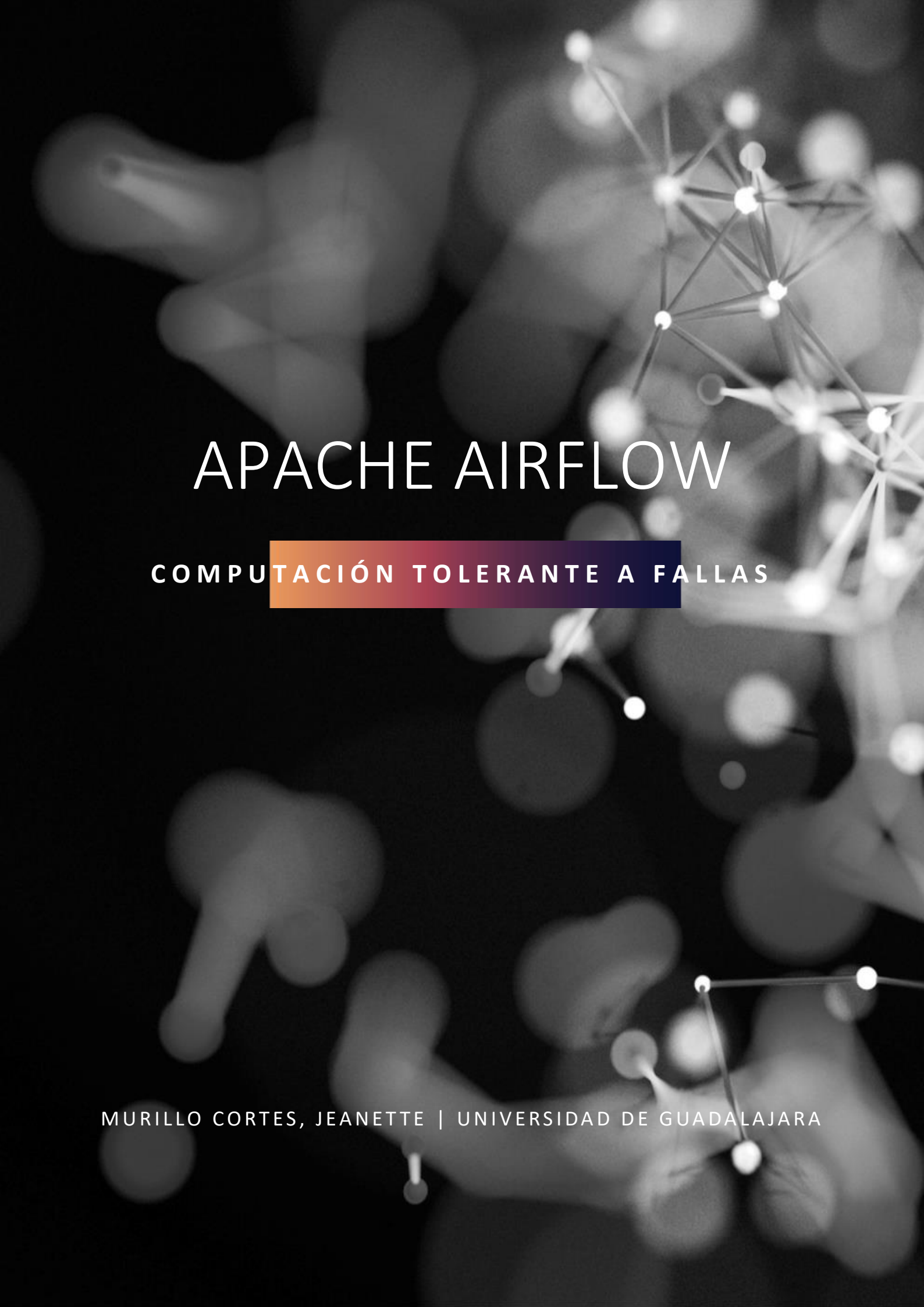
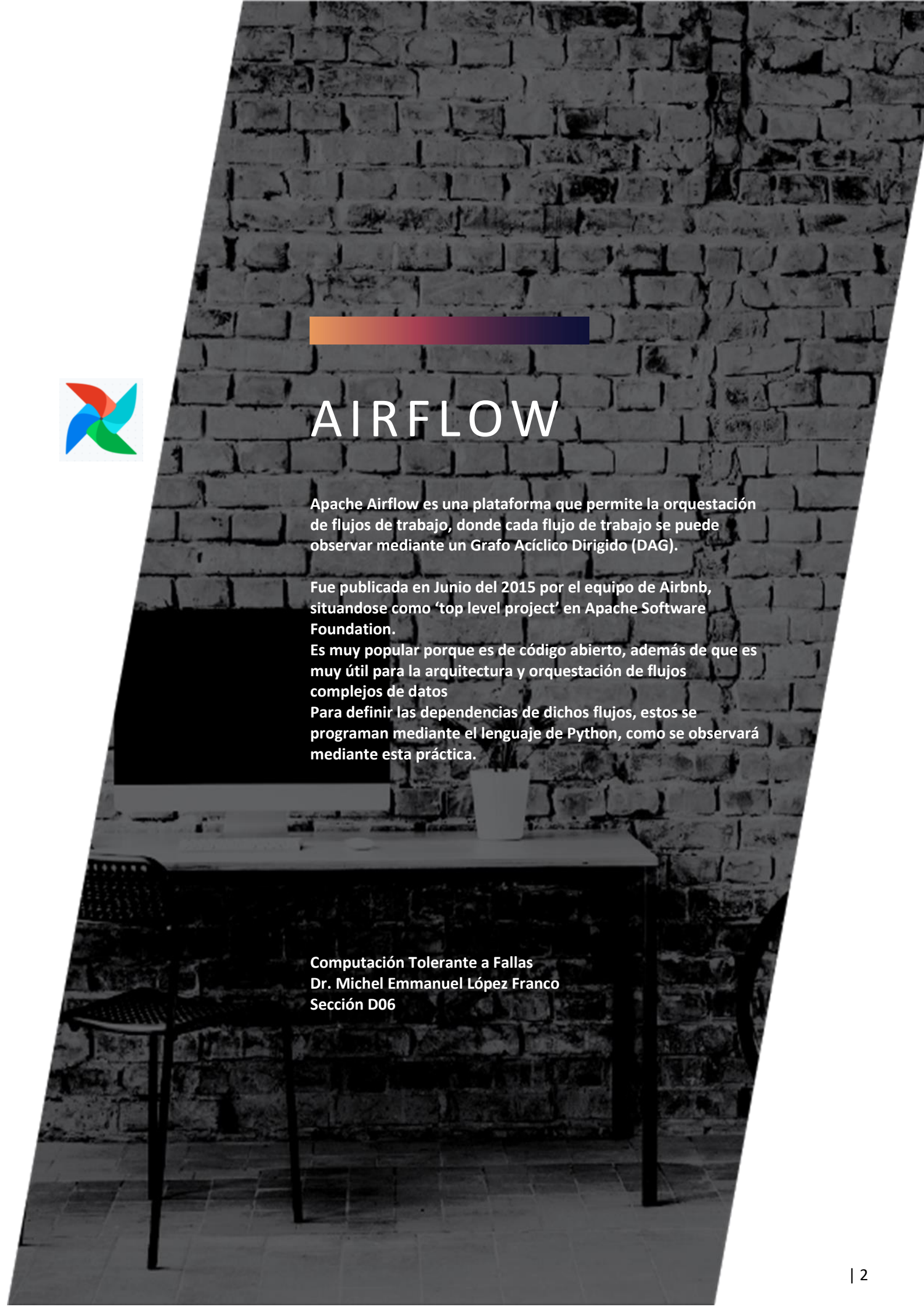


APACHE AIRFLOW

COMPUTACIÓN TOLERANTE A FALLAS

MURILLO CORTES, JEANETTE | UNIVERSIDAD DE GUADALAJARA



AIRFLOW

Apache Airflow es una plataforma que permite la orquestación de flujos de trabajo, donde cada flujo de trabajo se puede observar mediante un Grafo Acíclico Dirigido (DAG).

Fue publicada en Junio del 2015 por el equipo de Airbnb, situandose como 'top level project' en Apache Software Foundation.
Es muy popular porque es de código abierto, además de que es muy útil para la arquitectura y orquestación de flujos complejos de datos
Para definir las dependencias de dichos flujos, estos se programan mediante el lenguaje de Python, como se observará mediante esta práctica.

Computación Tolerante a Fallas
Dr. Michel Emmanuel López Franco
Sección D06

AIRFLOW

Ventajas de Apache Airflow

- Es dinámico, extensible y escalable para ejecuciones de flujos de trabajo o data pipelines.

COMANDOS PARA UBUNTU LTS

Los siguientes comandos fueron utilizados para moverse en el ambiente de Ubuntu 20.04:

- ls – Listar archivos
- cd / - Acceder a archivos del sistema
- cd .. – Regresar al directorio anterior
- mkdir – Crear directorio

PREINSTALACIONES

Para el desarrollo de la práctica se hizo uso de:

- Ubuntu 20.04 LTS y WSL
- Apache Airflow

INSTALACIÓN DE UBUNTU

Entrar a la Microsoft Store y descargar Ubuntu. Luego reiniciar la computadora. Abrir la consola de comandos y escribir "`wsl`".

El siguiente paso se hace mediante el uso de sudo, por lo que para su instalación, se descargó un archivo sudo.exe (obtenido del url: comando sudo) que se añadió a la ruta C:Windows. En este paso, solo se escribe en el cmd "`sudo cmd`" y abre la línea de comando de sudo. De esta manera, la línea de comandos normal, ya nos aceptará la posibilidad de hacer uso de sudo.

INSTALACIÓN DE AIRFLOW

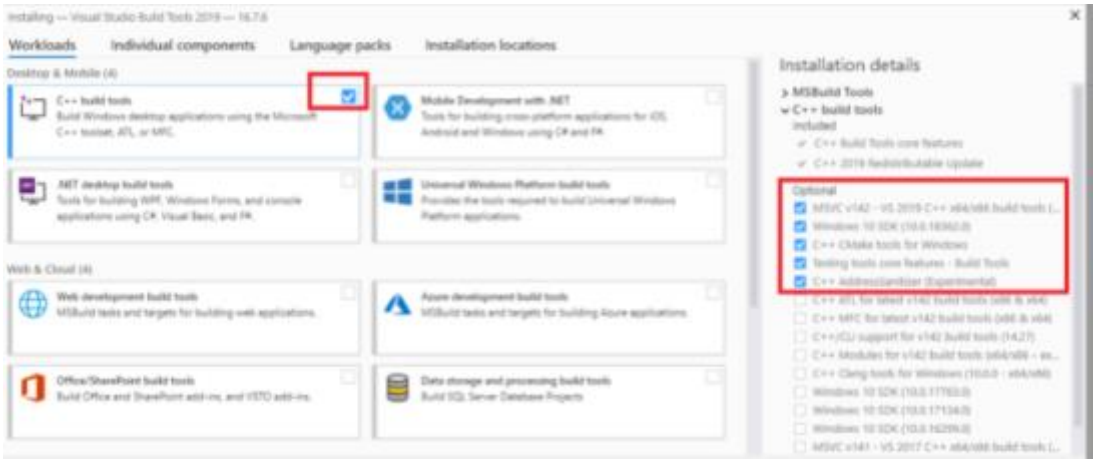
Para continuar con la instalación de Airflow, se tiene que ejecutar el comando para actualización:

```
sudo apt update && sudo apt upgrade
```

Instalar pip3 de por medio de los siguientes comandos:

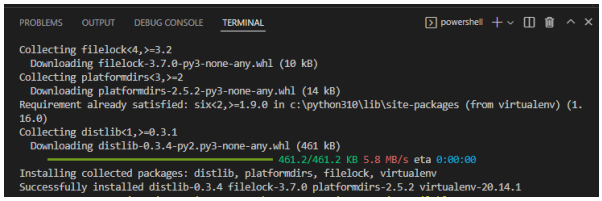
```
sudo apt-get install software-properties-common
sudo apt-add-respository universe
sudo apt-get update
sudo apt-get install python3-pip
```

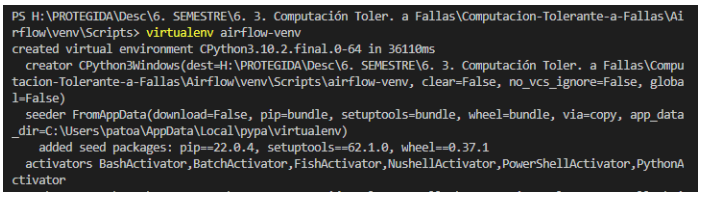
Para el siguiente paso, hay que instalar unas herramientas de Microsoft C++ Build en el link estas herramientas para que Airflow no de error:



DESARROLLO DE LA PRÁCTICA

Una vez instalados los requisitos necesarios para utilizar Airflow, utilizamos el comando **`pip3 install apache-airflow`**. En la carpeta que vayamos a utilizar de airflow, donde se pueda programar en un entorno virtual de Python, escribimos el comando python –m venv venv para la generación del proyecto. Para el desarrollo de la práctica se creó un entorno virtual escribiendo el comando **`pip install virtualenv`** y **`virtualenv airflow-venv`**.





Escribimos también el comando **`pip freeze`** para el uso de las librerías que proporciona Python.
Además, configuramos la variable de inicio de Airflow en una subcarpeta de la carpeta principal **`export AIRFLOW_HOME /=~/airflow`**.

```
scipy==1.8.0
seaborn==0.11.2
Send2Trash==1.8.0
six==1.16.0
sklearn==0.0
soupsieve==2.3.1
stack-data==0.2.0
statsmodels==0.13.2
terminado==0.13.3
testpath==0.6.0
threadpoolctl==3.1.0
tornado==6.1
traitlets==5.1.1
virtualenv==20.14.1
wcwidth==0.2.5
webencodings==0.5.1
```

Dentro del archivo de la configuración, importamos varias librerías para la configuración de fechas y horas, además de PythonOperator que permite manejar las tareas como un flujo de trabajo, junto con el export del archivo del cuál se hace la extracción del procesamiento ETL.
Luego definimos los argumentos que serán establecidos como default, los cuáles permitirán ser pasados a la configuración del DAG de Airflow; en dichos argumentos se establecen del DAG la fecha de inicio, el email y nombre del propietario, los números de intentos que se permitirán por si el DAG falla, además de que regrese cuál es el tipo de error por si el DAG falla.

```
from datetime import timedelta
from datetime import datetime
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
from base_etl import run_base_etl
```

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2022, 4, 19),
    'email': ['janemurciz@gmail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(minutes=1)
}
```

Para la configuración del DAG, también definimos un nombre, argumentos, una descripción y el intervalo en el que queramos que se realice dicho flujo, el cuál para este caso se especificó que sería diariamente.
De acuerdo con las tareas del flujo, se definió un operador que especifica cómo y el orden en que se ejecutarán las tareas del archivo **`base_etl`** el cuál contiene el procesamiento ETL.

```
dag = DAG(
    'spotify_dag',
    default_args=default_args,
    description='Our first DAG with ETL process!',
    schedule_interval=timedelta(days=1),
)
```

```
# Operator
run_etl = PythonOperator(
    task_id='whole_spotify_etl',
    python_callable=run_spotify_etl,
    dag=dag,
)

run_etl
```

Para la base del export del ETL, generamos un conector con Spotify mediante la librería spotipy, validando que la información que se está dando como argumentos el id del cliente, el url y permisos del usuario que interactúa con la lista de las ultimas canciones escuchadas por el usuario, son válidos.

```
import sqlalchemy
import pandas as pd
from sqlalchemy.orm import sessionmaker
import requests
import json
from datetime import datetime
import datetime
import sqlite3
```

```
def check_if_valid_data(df: pd.DataFrame) -> bool:
    if df.empty:
        print("No songs downloaded. Finishing execution")
        return False
    if pd.Series(df['played_at']).is_unique:
        pass
```

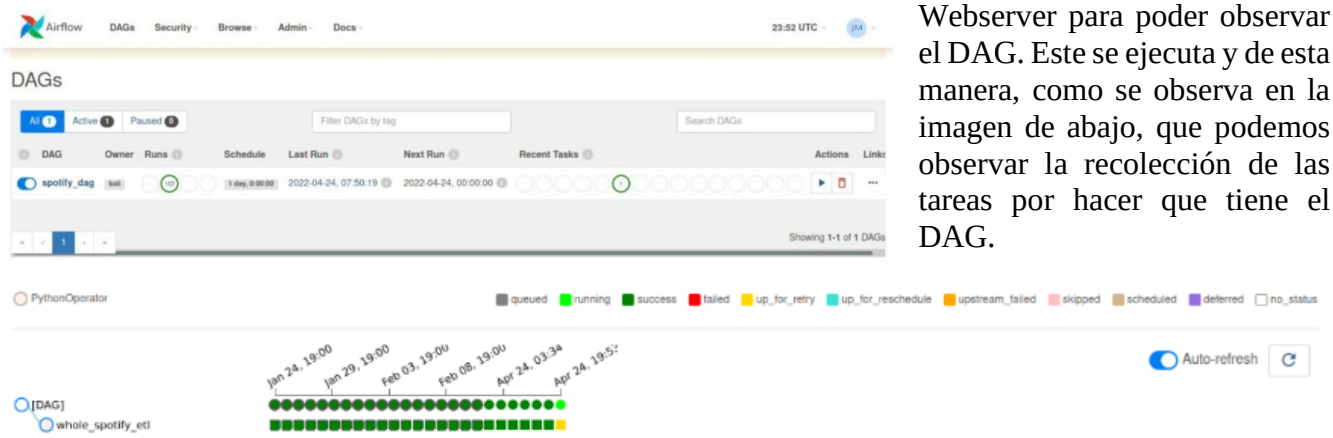
De esta manera, podemos acceder al nombre de las canciones con sus atributos y extraer dicha información con el Token y Id obtenido de Spotify que pertenece a nuestro usuario, estableciendo la lista de canciones que se escucharon en aproximadamente un día anterior; esta información se limpia mediante su almacenamiento en un diccionario llamado **`song_dict`** para así poder cargarse en una base de datos.

```
def run_spotify_etl():
    DATABASE_LOCATION = "sqlite:///my_played_tracks.sqlite"
    USER_ID = '14293487156'
    TOKEN = 'BRWiMr4nv9YZsJSVJbjjrtjFiA1IuGFgFKcE3afgNYO9C6TTbaMCV57Nd7YNvkleHnoD3KNjjOn2vKtIW2yZAyvxDo9r2G6A4iRZb6Wh4wGIjzELbrP

    headers = {
        "Accept" : "application/json",
        "Content-Type" : "application/json",
        "Authorization" : "Bearer {token}".format(token=TOKEN)
    }

    today = datetime.datetime.now()
    yesterday = today - datetime.timedelta(days=1)
    yesterday_unix_timestamp = int(yesterday.timestamp()) * 1000
```

```
    for song in data["items"]:
        song_names.append(song["track"]["name"])
        artist_names.append(song["track"]["album"]["artists"][0]["name"])
        played_at_list.append(song["played_at"])
        timestamps.append(song["played_at"][0:10])

    song_dict = {
        "song_name" : song_names,
        "artist_name": artist_names,
        "played_at" : played_at_list,
        "timestamp" : timestamps
    }
```

```
if check_if_valid_data(song_df):
    print("Data valid, proceed to Load stage")

engine = sqlalchemy.create_engine(DATABASE_LOCATION)
conn = sqlite3.connect('my_played_tracks.sqlite')
cursor = conn.cursor()

sql_query = """
CREATE TABLE IF NOT EXISTS my_played_tracks(
    song_name VARCHAR(200),
    artist_name VARCHAR(200),
    played_at VARCHAR(200),
    timestamp VARCHAR(200),
    CONSTRAINT primary_key_constraint PRIMARY KEY (played_at)
)
"""
cursor.execute(sql_query)
print("Opened database successfully")
```

Una vez teniendo la configuración del DAG y las tareas a realizar, ejecutamos Airflow y desplegamos el Webserver para poder observar el DAG. Este se ejecuta y de esta manera, como se observa en la imagen de abajo, que podemos observar la recolección de las tareas por hacer que tiene el DAG.



CONCLUSIÓN

Apache Airflow es una herramienta bastante util para el manejo de los flujos de trabajo, por lo que su implementación durante la actividad y su interfaz ayudaron bastante a su comprención y desarrollo en la programación. Por otro lado, para esta actividad, me dio una introducción más cercana acerca del manejo de Ubuntu, dado que al tuve bastantes conflictos para su uso con Airflow y tuve que investigar bastantes soluciones para su funcionamiento, aunque al final, me ayudó a comprender qué necesecitaba Airflow para su funcionamiento, mostrando este a su vez que su uso puede darnos información que es útil al momento de ejecutar varias tareas.

B I B L I O G R A F Í A

- Apache Airflow, el nuevo director de orquesta de Google Cloud - Paradigma (paradigmadigital.com) Recuperado el 04/12/22. Obtenido del URL: https://www.paradigmadigital.com/dev/apache-airflow/
- Liher (21 Abril, 2014). *"Como moverse por los directorios en la Terminal de Ubuntu | El blog de Liher"* Recuperado el 04/19/22. Obtenido del URL: https://elblogdeliher.com/como-moverse-por-los-directorios-en-la-terminal-de-ubuntu/
- Karolina Sowinska (Noviembre, 2020). *"Airflow for Beginners - Run Spotify ETL Job in 15 minutes!"*. Recuperado el 05/10/22. Obtenido del URL: https://www.youtube.com/watch?v=i25ttd32-eo