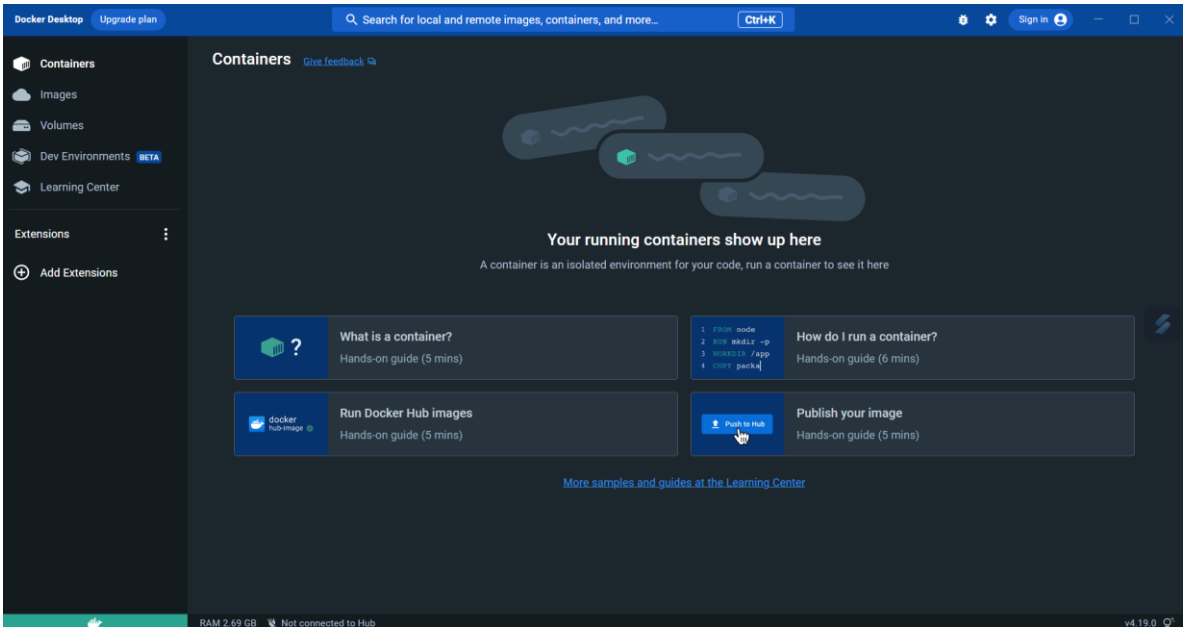
Solución desafio técnico: construir un pipeline con Docker y Apache Ariflow

Objetivo: Crear un pipeline de datos simple usando Apache Airflow que extraiga datos de un archivo CSV, los transforme y los exporte a otro archivo CSV, todo ejecutándose en un contenedor Docker. Este desafío te ayudará a demostrar tus habilidades en Python, procesos ETL, trabajo con DAG's en Apache Airflow y contenerización con Docker. Cualquier duda enviar un correo ageracollante95@gmail.com.

1. Configurar Apache Airflow con Docker
    a. Instalar Docker en tu máquina local.

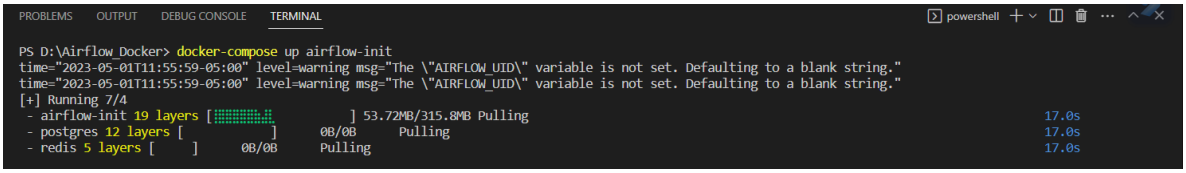
R/ Se descarga docker y se instala en una maquina local.

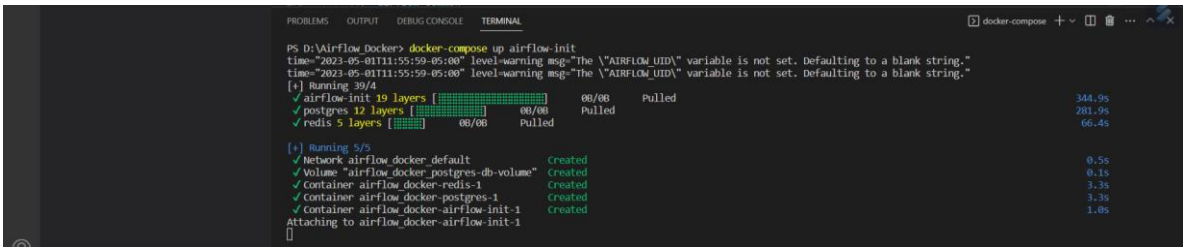


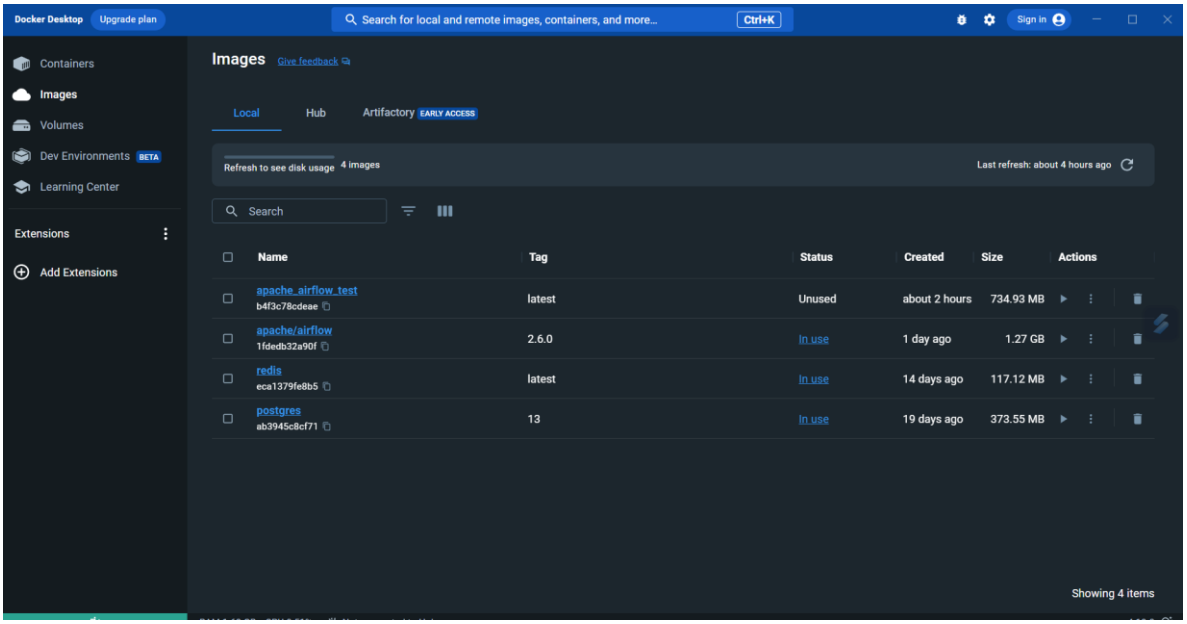
Nota: tener en cuenta la actualización para ejecutar subsitemas Linux (WSL2).

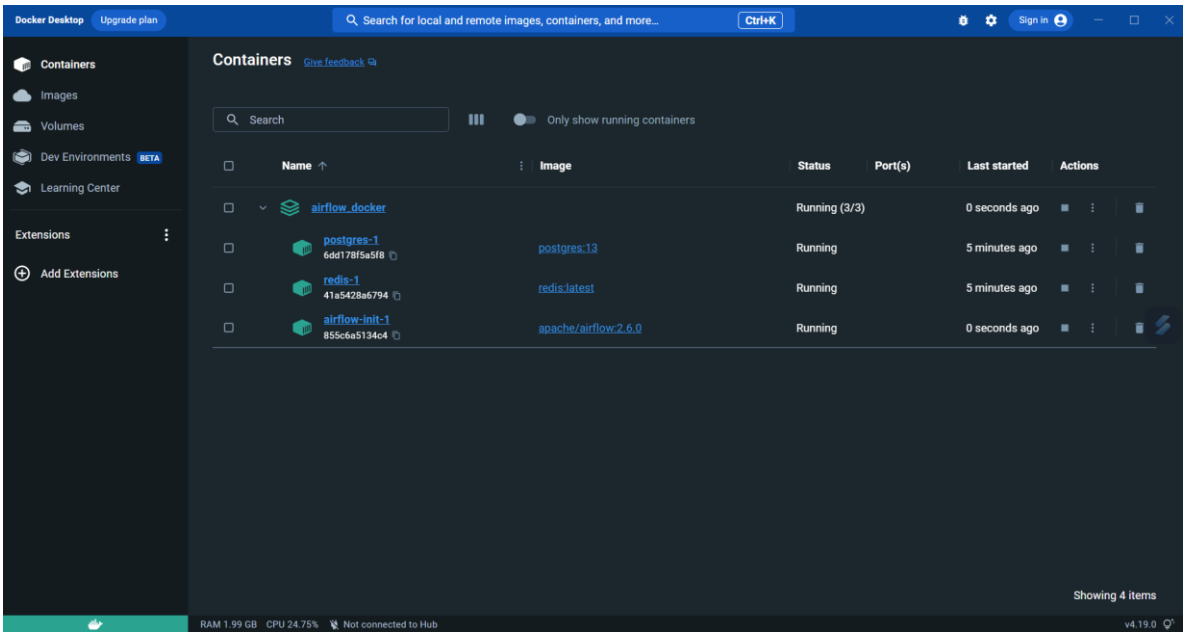
    b. Crear un Dockerfile que incluya las dependencias requeridas y la configuración para Apache Airflow

R/ La instalación de airflow se realiza por un archivo ".yaml", ya que la instalación por este medio es más manejable que por una serie de instrucciones armadas por medio de un dockerfile, el ".yaml" fue descargado desde la página de Apache Airflow (LINK), tener en cuenta que se debe instalar docker-compose, el cual también se explica AQUI









c. Construir y ejecutar el contenedor de Airflow utilizando el Dockerfile

R/ Dado que se usó el “.yaml” el proceso se ejecuta automáticamente una vez se inicia docker.

2. Extracción de Datos:

a. Descargar el archivo CSV de ejemplo que contiene datos abiertos de accidentes de tráfico en la ciudad de Seattle.

R/ Solución explicada al finalizar las preguntas en general

b. Crear una tarea PythonOperator de Airflow para leer el archivo CSV.

R/ Solución explicada al finalizar las preguntas en general

3. Transformación de Datos:

a. Crear una tarea PythonOperator de Airflow para limpiar y transformar los datos eliminando cualquier fila con valores faltantes.

R/ Se identificó que el archivo NO tiene información nula o vacía, sin embargo, se incorporó al proceso una validación para suplir este requerimiento explicada al finalizar las preguntas en general

b. Calcular el número total de accidentes por tipo de clima.

R/ No se encontró información del clima, en su lugar se realizó un conteo de accidentalidad por año

4. Exportación de Datos:

a. Crear una tarea PythonOperator de Airflow para exportar los datos transformado sa un nuevo archivo CSV.

R/ Solución explicada al finalizar las preguntas en general

5. Crear el DAG:

a. Definir un DAG que incluya las tareas de extracción, transformación y exportación de datos creadas en los pasos 2-4.

R/ Se agrupa respuesta con las preguntas anteriores, el dag se encuentra en el GITHUB con todo lo relacionado

b. Establecer las dependencias de las tareas para garantizar que se ejecuten en el orden correcto.

R/ igual que el punto anterior

c. Programar el DAG para que se ejecute en un intervalo específico (por ejemplo, todos los días a la media noche).

R/ igual que el punto anterior

Como fue mencionado en las respuestas relacionadas, se creó un DAG, donde hay 3 funciones:

- ✓ Función input_file:

La función descarga el archivo https://data.seattle.gov/api/views/38vd-gytv/rows.csv?accessType=DOWNLOAD y lo almacena en la carpeta input

```
# -------- Function definition -----------------------
def input_file(url):
    """Descarga archivo a local de acuerdo a url determinada."""
    response = requests.get(url)
    inpunt_file_name = os.path.join(in_path, file_name)
    print(f'\n\n {response.status_code} \n\n')
    print(f'\n\n {inpunt_file_name} \n\n')
    if response.status_code == 200:
        with open(inpunt_file_name, "wb") as file:
            file.write(response.content)
        return '\n\n Done input file \n\n'
    else:
        return '\n\n Don't input file \n\n'
```

se crea una variable en airflow para setear la url mencionada y así dar protestad a modificar el link de obtención de la información

- ✓ Función output_file:

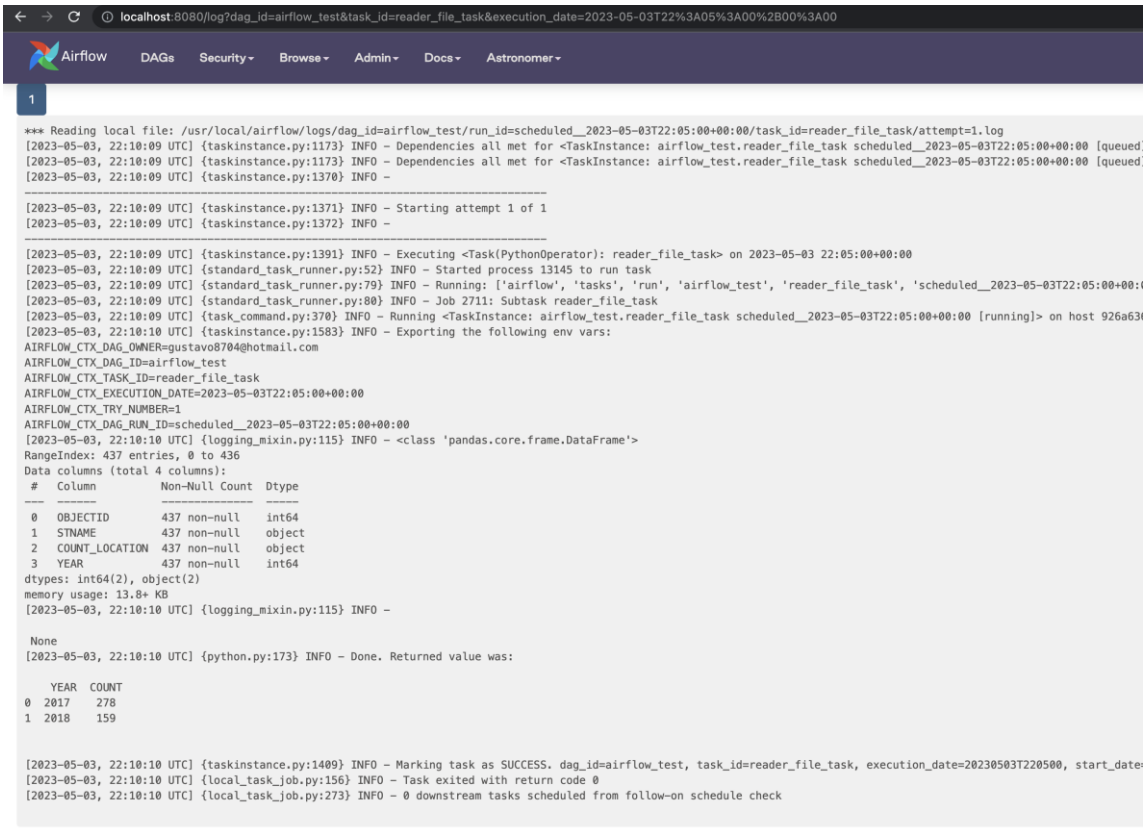
La función captura el archivo descargado en la carpeta input, realiza borrado de filas si encuentra alguna nula y lo deja con las 4 primeras columnas, el archivo se deja en la carpeta output

```
def output_file(in_p, out_p):
    """Trandforma el archivo importado."""
    path_file = os.path.join(in_p, file_name)
    out_path_file = os.path.join(out_p, output_file_name)
    file_df = pd.read_csv(path_file, sep=',', header=0)
    print(f'\n\n {file_df.head(10)} \n\n')
    file_df = file_df.dropna(how='any').reset_index(drop=True)
    file_df = file_df.iloc[:, [0, 1, 2, 3]]
    print(f'\n\n {file_df.head(10)} \n\n')
    file_df.to_csv(out_path_file, index=False, sep=';')
    return '\n\n Done output file \n\n'
```

- ✓ Función reader_file:

La función captura el archivo alojado en la carpeta output y hace una operación simple, realiza un conteo de accidentes por año y los muestra en el log de la tarea en airflow.

```python
def reader_file(out_p):
    """Realiza operaciones sobre el archivo transformado."""
    out_path_file = os.path.join(out_p, output_file_name)
    file_df = pd.read_csv(out_path_file, sep=';', header=0)
    conteo = file_df.groupby(['YEAR']).size().reset_index(name='COUNT')
    print(f'\n\n {file_df.info()} \n\n')
    return f'\n\n {conteo} \n\n'
```

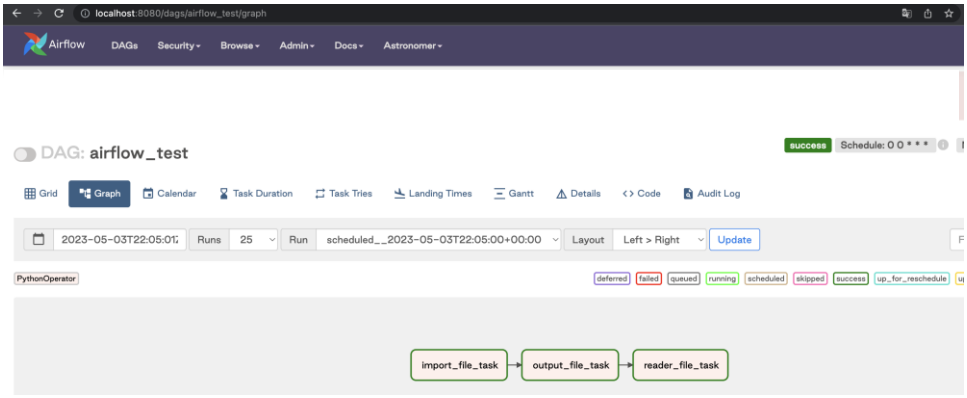


Cada función se ejecuta por medio de PythonOperator y se determinó el orden de ejecución como se refleja a continuación:

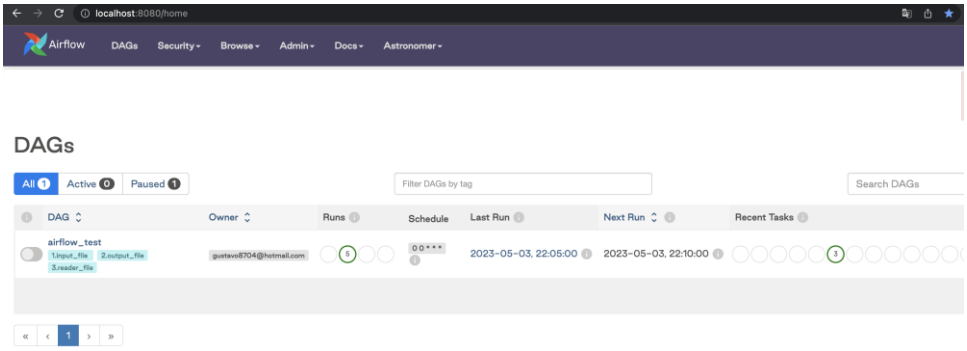
```python
# -------- Dag definition ------------------------
default_args = {
    'owner': 'gustavo8704@hotmail.com',
    'depends_on_past': False,
    'retries': 0
}

dag = DAG('airflow_test',
          max_active_runs=1,
          schedule_interval='0 0 * * *',
          start_date=datetime(2022, 1, 13),
          tags=['1.input_file', '2.output_file', '3.reader_file'],
          default_args=default_args,
          catchup=False
          )
```

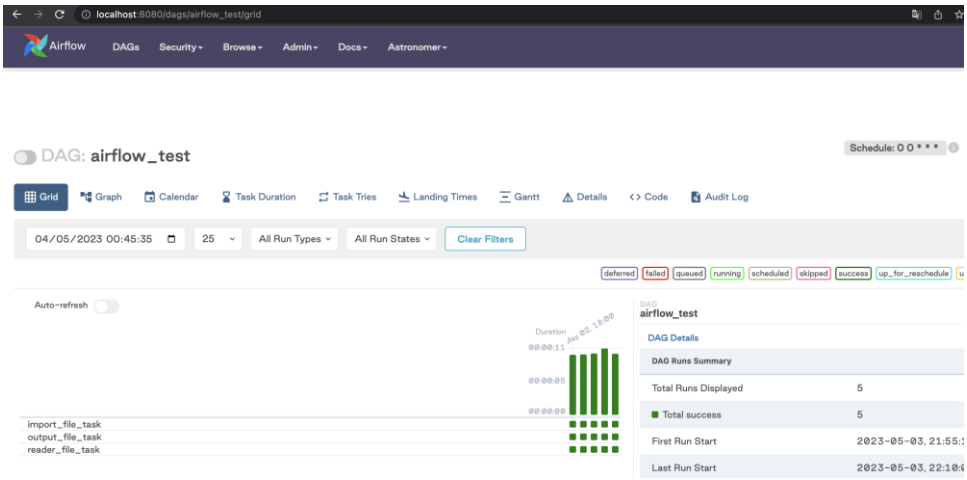
```python
with dag:
    import_file_task = PythonOperator(
        task_id='import_file_task',
        python_callable=input_file,
        op_args=[url]
        )
    output_file_task = PythonOperator(
        task_id='output_file_task',
        python_callable=output_file,
        op_args=[in_path, out_path]
        )
    reader_file_task = PythonOperator(
        task_id='reader_file_task',
        python_callable=reader_file,
        op_args=[out_path]
        )
    import_file_task
    output_file_task.set_upstream(import_file_task)
    reader_file_task.set_upstream(output_file_task)
```

Se ilustra la dependencia de las tareas en la siguiente imagen



También se evidencia como se ve el dag desde airflow





Entregables:

● Dockerfile para el contenedor de Airflow.

●Scripts de Python para las tareas de Airflow, incluidos extracción, transformación y exportación de datos.

●El archivo de definición del DAG de Airflow.

●Un archivo README que contenga instrucciones sobre cómo configurar y ejecutar el proyecto localmente utilizando Docker, así como cualquier dependencia requerida.

Porfavor, crea un repositorio en GitHub que contenga todos los archivos requeridos y comparte el enlace del repositorio para enviar tu solución. ¡Buenasuerte!

Los entrregables se encuentran en el siguiente link de GITHUB
https://github.com/gagarb/airflow_tesst/tree/main/airflow_test