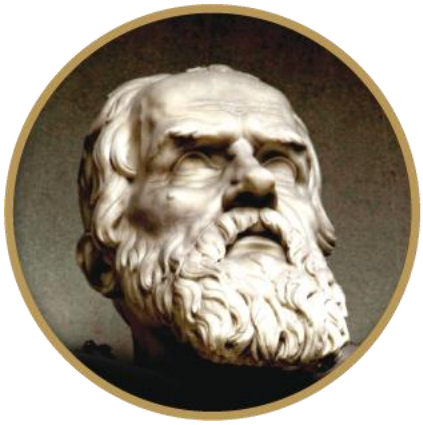
PROYECTO FINAL

Product Development



Galileo
UNIVERSIDAD
La Revolución en la Educación

docker

Streamlit

Apache
Airflow

José Alberto Ligorría Taracena - 20000604
Juan Pablo Carranza Hurtado - 20000607
Diego Fernando Valle Morales - 20003022
José Sebastián Rodríguez Velásquez - 2000307

Contenido

Primeros pasos Docker

Implementando Containers por primera vez

Creando una red

docker network create --driver bridge red_parcial1

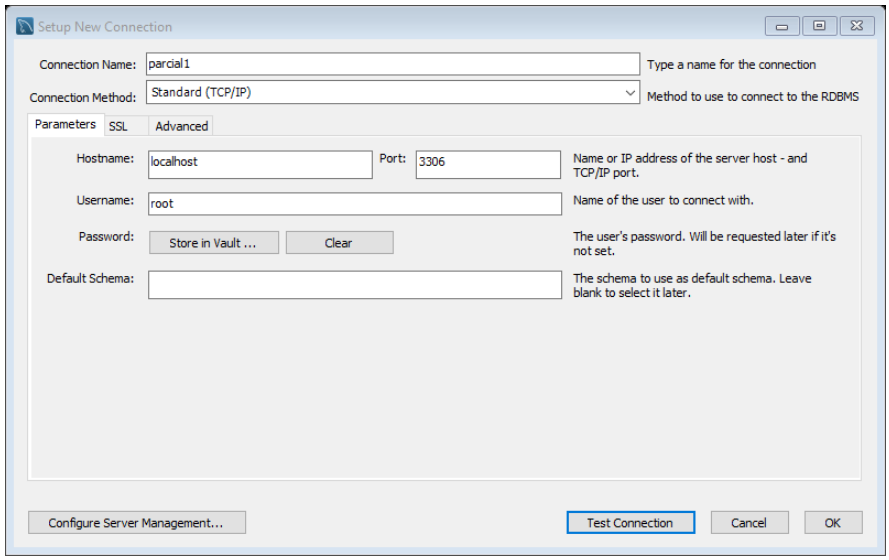
```
PS C:\WINDOWS\system32> docker network create --driver bridge red_parcial1
74a358d1758129f297b6eafafe889bab4d2a7cb9bd52dda806581252674ca90f
```

Conectando a MySQL

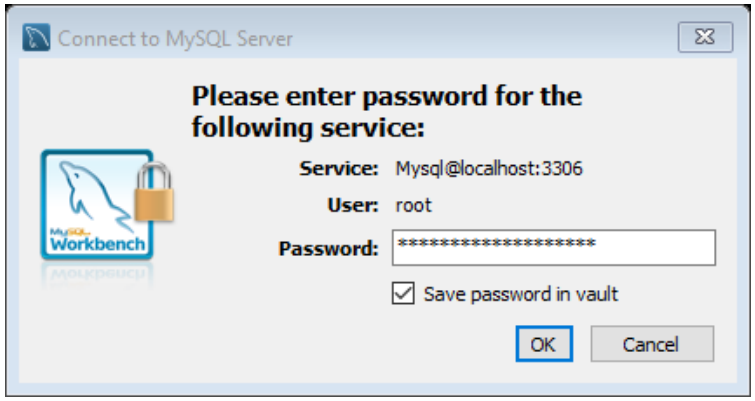
docker run --network red_parcial1 -p 3306:3306 --name docker_bd_parcial1 -e MYSQL_ROOT_PASSWORD=parcial1ACADEMATICA -d mysql:latest

```
PS C:\WINDOWS\system32> docker run --network red_parcial1 -p 3306:3306 --name docker_bd_parcial1 -e MY
SQL_ROOT_PASSWORD=parcial1ACADEMATICA -d mysql:latest
9feb83bd9e42dffa8047684fde60368b7708d72c1933e61de2fa9a1859ae6b53
```

Para validar la conexión se puede usar herramientas como MySQL Workbench



Y allí colocar Test Connection



Ejecución del primer Docker con red creada

Para crear un Docker de rstudio se deberá de colocar el siguiente comando:

docker run -d --network red_parcial1 --name docker_rstudio_parcial1 -e PASSWORD=parcial1ACADEMATICA -p 8787:8787 rocker/tidyverse

```
PS C:\WINDOWS\system32> docker run -d --network red_parcial1 --name docker_rstudio_parcial1 -e PASSWOR
D=parcial1ACADEMATICA -p 8787:8787 rocker/tidyverse
Unable to find image 'rocker/tidyverse:latest' locally
latest: Pulling from rocker/tidyverse
a4a2a29f9ba4: Pull complete
127c9761dcba: Pull complete
d13bf203e905: Pull complete
4039240d2e0b: Pull complete
3c152e313525: Pull complete
742ac31d6184: Pull complete
d0724624a7d0: Pull complete
baeef85a2aa1: Pull complete
ee1f9b8d6cdc: Pull complete
Digest: sha256:6c228f305c6e1322e7259cd22d0bcfffb26b56fde08b4d6eb854405f7943d9da
Status: Downloaded newer image for rocker/tidyverse:latest
884ec4b54ef7091be332dc8d3d0bc7fe4640b0fd07784e4dba9f86a9a8cd8aaa
```

Validando ejecución de contenedores

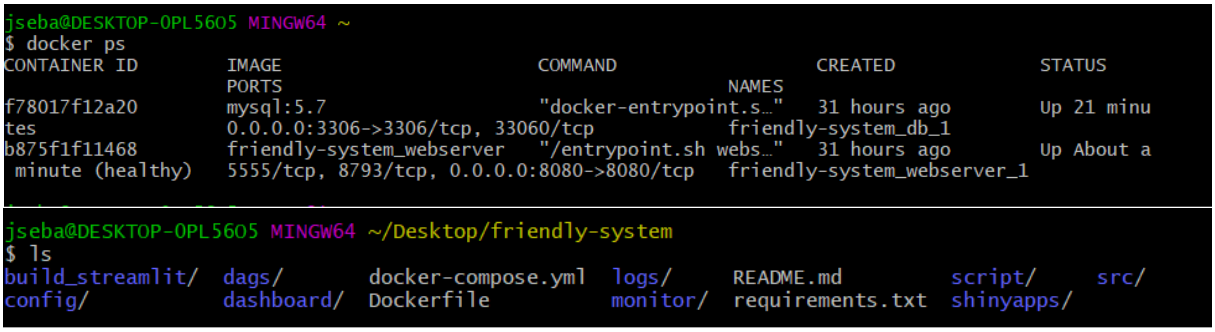
Para validar la ejecución de contenedores se debe usar el comando Docker ps y devolverá la siguiente información:

```
PS C:\WINDOWS\system32> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
   PORTS                               NAMES
884ec4b54ef7        rocker/tidyverse    "/init"                  8 minutes ago       Up 8 minutes
   0.0.0.0:8787->8787/tcp              docker_rstudio_parcial1
9feb83bd9e42        mysql:latest        "docker-entrypoint.s…"   52 minutes ago      Up 52 minutes
   0.0.0.0:3306->3306/tcp, 33060/tcp   docker_bd_parcial1
```

Implementación de Docker

Para la creación del Docker en este proyecto se realizó la copia del repositorio inicial con airflow dado en el curso y se realizó un pull de este para trabajarlo como base. El Docker puede encontrarse en la siguiente dirección: https://github.com/obedaeg/airflow

Para esto creamos una carpeta nombrada “friendly-system” la cual contiene todos los archivos e imagen que vamos a utilizar a lo largo del proyecto.



Basta con utilizar los comandos para construir y levantar el docker:

- Docker-compose build

- Docker-compose up

```
jseba@DESKTOP-OPL56O5 MINGW64 ~/Desktop/friendly-system
$ docker-compose up
Starting friendly-system_postgres_1  ... done
Starting friendly-system_streamlit_1 ...
friendly-system_webserver_1 is up-to-date
Starting friendly-system_streamlit_1 ... done
Starting friendly-system_shiny_app_1 ... done
Attaching to friendly-system_postgres_1, friendly-system_webserver_1, friendly-system_db_1, friendly-sy
stem_streamlit_1, friendly-system_shiny_app_1
postgres_1   |
postgres_1   | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres_1   |
db_1         | 2020-12-09 18:39:58+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.32
-1debian10 started.
db_1         | 2020-12-09 18:40:05+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
db_1         | 2020-12-09 18:40:06+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.32
-1debian10 started.
db_1         | 2020-12-09 18:40:06+00:00 [Note] [Entrypoint]: Initializing database files
db_1         | 2020-12-09T18:40:06.263854Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is depreca
ted. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
db_1         | 2020-12-09T18:40:07.687440Z 0 [Warning] InnoDB: New log files created, LSN=45790
db_1         | 2020-12-09T18:40:08.182523Z 0 [Warning] InnoDB: Creating foreign key constraint system t
ables.
db_1         | 2020-12-09T18:40:08.725602Z 0 [Warning] No existing UUID has been found, so we assume th
at this is the first time that this server has been started. Generating a new UUID: f67daaaa-3a4d-11eb-
bc0b-0242ac120005.
```

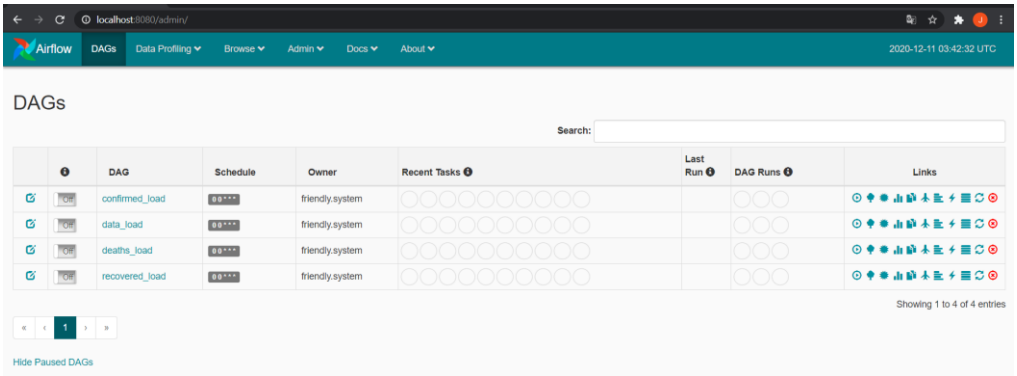
Con esto podemos acceder a nuestro localhost en el puerto 8080 desde nuestro navegador.

Dirección: http://localhost:8080/admin/

AirFlow

Conexión a la AirFlow

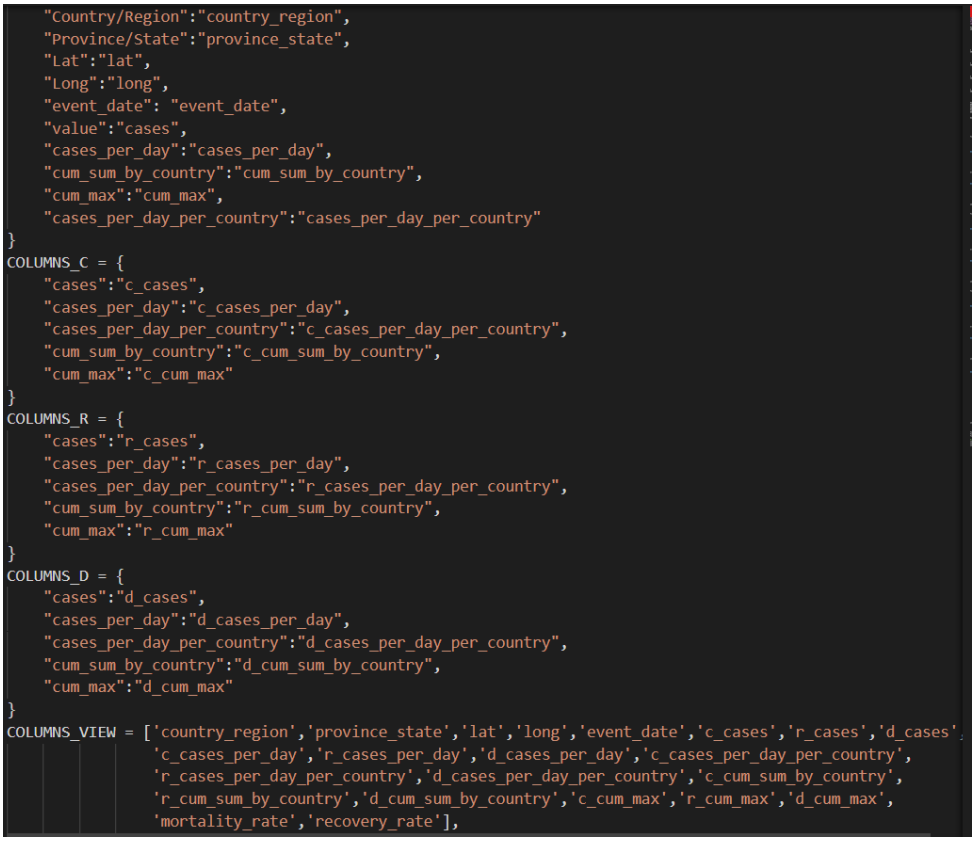
Una vez levantado nuestro Docker nos mostrará la página inicial de airflow apache en donde estarán 4 Dags creados.



Explicación de Dags

Transform

Este DAG se encarga de realizar la transformación en nuestros dataframes de tal forma que los agrupa y relaciona para crear diferentes columnas con variables para mostrar en nuestro StreamLit. Abajo encontrarán las diferentes columnas que se crearon para dar interpretabilidad por medio de streamlit en forma gráfica.



```
    "Country/Region":"country_region",
    "Province/State":"province_state",
    "Lat":"lat",
    "Long":"long",
    "event_date": "event_date",
    "value":"cases",
    "cases_per_day":"cases_per_day",
    "cum_sum_by_country":"cum_sum_by_country",
    "cum_max":"cum_max",
    "cases_per_day_per_country":"cases_per_day_per_country"
}
COLUMNS_C = {
    "cases":"c_cases",
    "cases_per_day":"c_cases_per_day",
    "cases_per_day_per_country":"c_cases_per_day_per_country",
    "cum_sum_by_country":"c_cum_sum_by_country",
    "cum_max":"c_cum_max"
}
COLUMNS_R = {
    "cases":"r_cases",
    "cases_per_day":"r_cases_per_day",
    "cases_per_day_per_country":"r_cases_per_day_per_country",
    "cum_sum_by_country":"r_cum_sum_by_country",
    "cum_max":"r_cum_max"
}
COLUMNS_D = {
    "cases":"d_cases",
    "cases_per_day":"d_cases_per_day",
    "cases_per_day_per_country":"d_cases_per_day_per_country",
    "cum_sum_by_country":"d_cum_sum_by_country",
    "cum_max":"d_cum_max"
}
COLUMNS_VIEW = ['country_region','province_state','lat','long','event_date','c_cases','r_cases','d_cases',
                'c_cases_per_day','r_cases_per_day','d_cases_per_day','c_cases_per_day_per_country',
                'r_cases_per_day_per_country','d_cases_per_day_per_country','c_cum_sum_by_country',
                'r_cum_sum_by_country','d_cum_sum_by_country','c_cum_max','r_cum_max','d_cum_max',
                'mortality_rate','recovery_rate'],
```

Data_Load

Monitor

Este DAG realiza la detección de archivos por medio de una variable denominada file_sensor la cual monitorea los archivos que colocamos en la carpeta de “monitor”, esta se encuentra en la carpeta mencionada anteriormente de “friendly-system”.

Transformación

Se realiza el proceso de transformación el cual obtiene el path de nuestra data y asigna el destino con el output del dag Transform. Ahí mismo se renombra las columnas en el archivo destino con las del generado en el dag de transform.

```python
    def transform_func_C(**kwargs):
        folder_path = FSHook(conn_id=FILE_CONNECTION_ID).get_path()
        file_path = f"{folder_path}/{FILE_NAME_C}"
        destination_file = f"{folder_path}/{OUTPUT_TRANSFORM_FILE_C}"
        df = pd.read_csv(file_path,encoding="ISO-8859-1")
        df_final = transform.transformm_df(df)
        df_final = df_final.rename(columns=COLUMNS_BASE)
        df_final.to_csv(destination_file, index=False)
        os.remove(file_path)
        return destination_file

    transform_process_C = PythonOperator(dag=dag,
                                         task_id="transform_process_C",
                                         python_callable=transform_func_C,
                                         provide_context=True
                                         )

def transform_func_D(**kwargs):
    folder_path = FSHook(conn_id=FILE_CONNECTION_ID).get_path()
    file_path = f"{folder_path}/{FILE_NAME_D}"
    destination_file = f"{folder_path}/{OUTPUT_TRANSFORM_FILE_D}"
    df = pd.read_csv(file_path,encoding="ISO-8859-1")
    df_final = transform.transformm_df(df)
    df_final = df_final.rename(columns=COLUMNS_BASE)
    df_final.to_csv(destination_file, index=False)
    os.remove(file_path)
    return destination_file
```

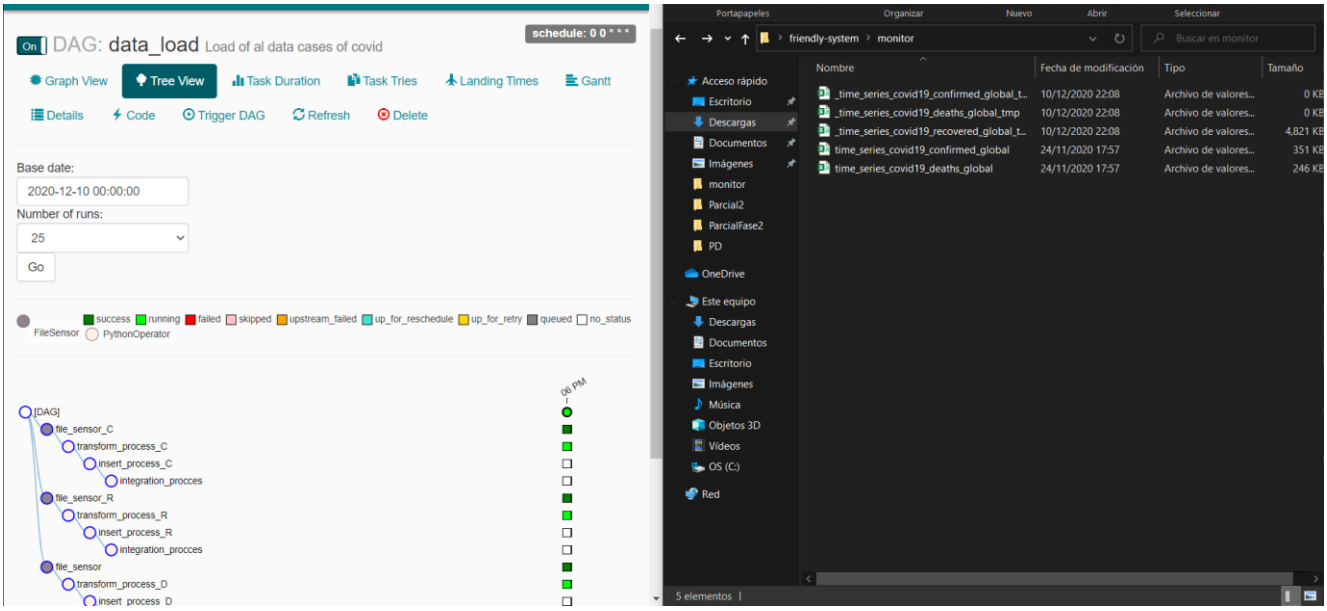
Insert

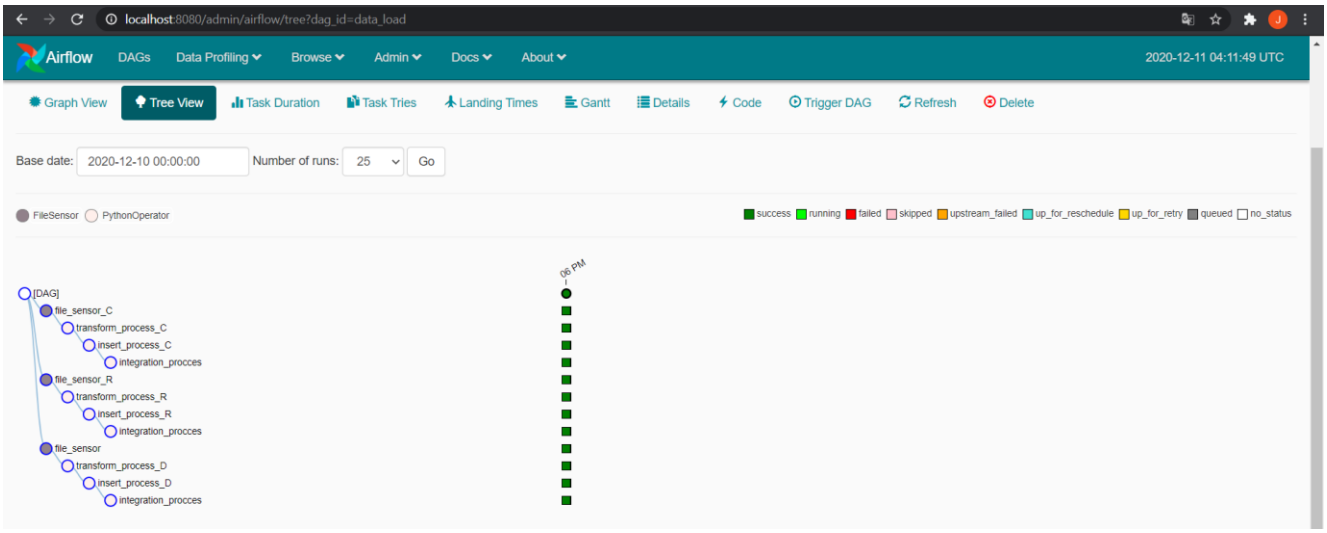
Se realiza la inserción de la data transformada para la base de datos que consideramos para este proyecto que es MySQL. En este mismo archivo tenemos la escritura automática con la base de datos para el MySQL por lo que no es necesario realizar el proceso manual aunque de igual forma será explicado más adelante.

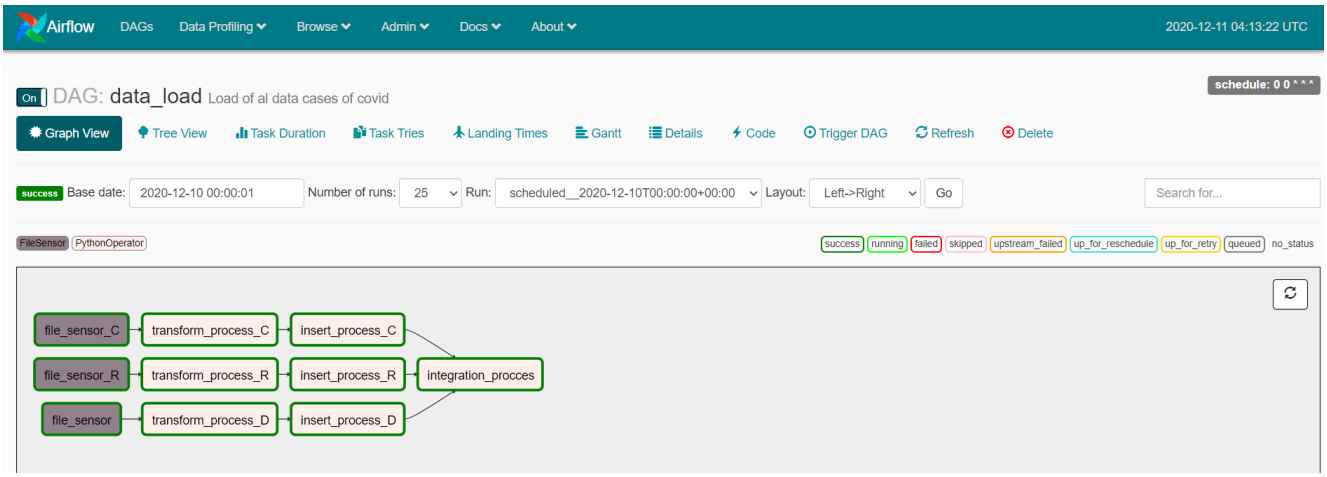
```python
def insert_process_R(**kwargs):
    ti = kwargs['ti']
    source_file = ti.xcom_pull(task_ids='transform_process_R')
    db_connection = MySqlHook('airflow_db').get_sqlalchemy_engine()
    df = pd.read_csv(source_file)
    with db_connection.begin() as transaction:
        transaction.execute("DELETE FROM covid.recovered WHERE 1=1")
        df.to_sql("recovered", con=transaction, schema="covid", if_exists="append",
                  index=False)
    os.remove(source_file)
```

Carga de data

Con ese paso realizado se procede a realizar el ETL automáticamente y realizar la conexión automática con la base datos de MySQL que será explicada en un apartado diferente.







Una vez esto se encuentra ejecutado podemos encontrar la data procesada en nuestra base datos lista para utilizarse en la aplicación de Streamlit.

La forma en que funciona es la siguiente:

El file sensor: Realiza la detección de los archivos en la carpeta “monitor” que se encuentra en la carpeta inicial.

Se realiza la transformación de data por medio del Dag Transform el cual es importado en el Dag de Data_Load.

Se realiza el insert de cada archivo en la base de datos de MySQL y por último se realiza la integración de toda esa información en una única tabla la cual podemos observar en la base de datos de MySQL.

Transformación ETL

Entrando más a detalle con las transformación ETL. Los pasos a realizar son:

1. La fusión en el dataset seteando por posición las variables de interés.
2. Realizar limpieza de la data y darle un formato/nombre más amigable para nosotros.
3. Realizar los primeros

```python
def transformm_df(df):
    print('Entramos 1')
    df = pd.melt(df, df.iloc[:,0:4])
    df['Province/State'] = df['Province/State'].fillna('')
    df['variable'] = pd.to_datetime(df['variable'])
    df.rename(columns = {'variable':'event_date'}, inplace = True)
    print('Entramos 2')
    df_sorted = df.sort_values(by = ['Country/Region', 'Province/State', 'event_date'])
    df_sorted['ID'] = df_sorted['Country/Region'] + df_sorted['Province/State']
    df_sorted['cases_per_day'] = df_sorted.value - df_sorted.groupby(['ID'])['value'].shift(1)
    df_sorted['cases_per_day'] = df_sorted['cases_per_day'].fillna(0)
    df_sorted = df_sorted.sort_values(by = ['Country/Region', 'event_date'])
    print('Entramos 3')
    print(pd.__version__)
    #df_sorted['cases_per_day_per_country'] = df_sorted.value - df_sorted.groupby(['Country/Region'])['value'].shift(1)
    #df_sorted['cases_per_day_per_country'] = df_sorted['cases_per_day_per_country'].fillna(0)
    #df_final = df_sorted.drop(columns = ['ID'])
    df_sorted['cum_sum_by_country'] = df_sorted.groupby(['Country/Region', 'event_date'])['value'].cumsum()
    print('Entramos 3 1')
    df_sorted.cum_sum_by_country[df_sorted['Province/State'] == ''] = 0
    print('Entramos 3 2')
    df_sorted = df_sorted.sort_values(['Country/Region', 'event_date','cum_sum_by_country'], ascending=[True, True, False])
    print('Entramos 3 3')
    print(df_sorted.columns)
    print(df_sorted.groupby(['Country/Region', 'event_date']))
    df_maxes = df_sorted.groupby(['Country/Region', 'event_date'])['Country/Region','event_date','cum_sum_by_country'].max()
    print(df_maxes)
    print('Entramos 4')
    print(df_maxes.columns)
    print(df_maxes.info())
    df_maxes.reset_index(inplace=True)
    print(df_maxes.columns)
    print(df_maxes.iloc[:,0])
    keys = np.array(df_maxes.iloc[:,0] + df_maxes['event_date'].astype('str'))
    print(keys)
    print('Entramos 4 1')
    print(df_maxes['cum_sum_by_country'])
    dictionary = dict()
    for i in range(len(df_maxes.cum_sum_by_country)):
        dictionary[keys[i]] = df_maxes.cum_sum_by_country[i]

    print('Entramos 5')
    max_list = []
    temp_country = ''
    temp_date = '12-10-1492'
    temp_max = -1
    key = ''

    df_sorted.event_date = df_sorted.event_date.astype('str')
    print('Entramos 6')

    print(df_sorted.iloc[:,0])
    print(df_sorted.iloc[:,1])
    print(df_sorted.iloc[:,5])
```

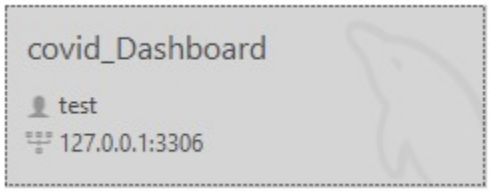
Conexión automática a Base de datos MySQL

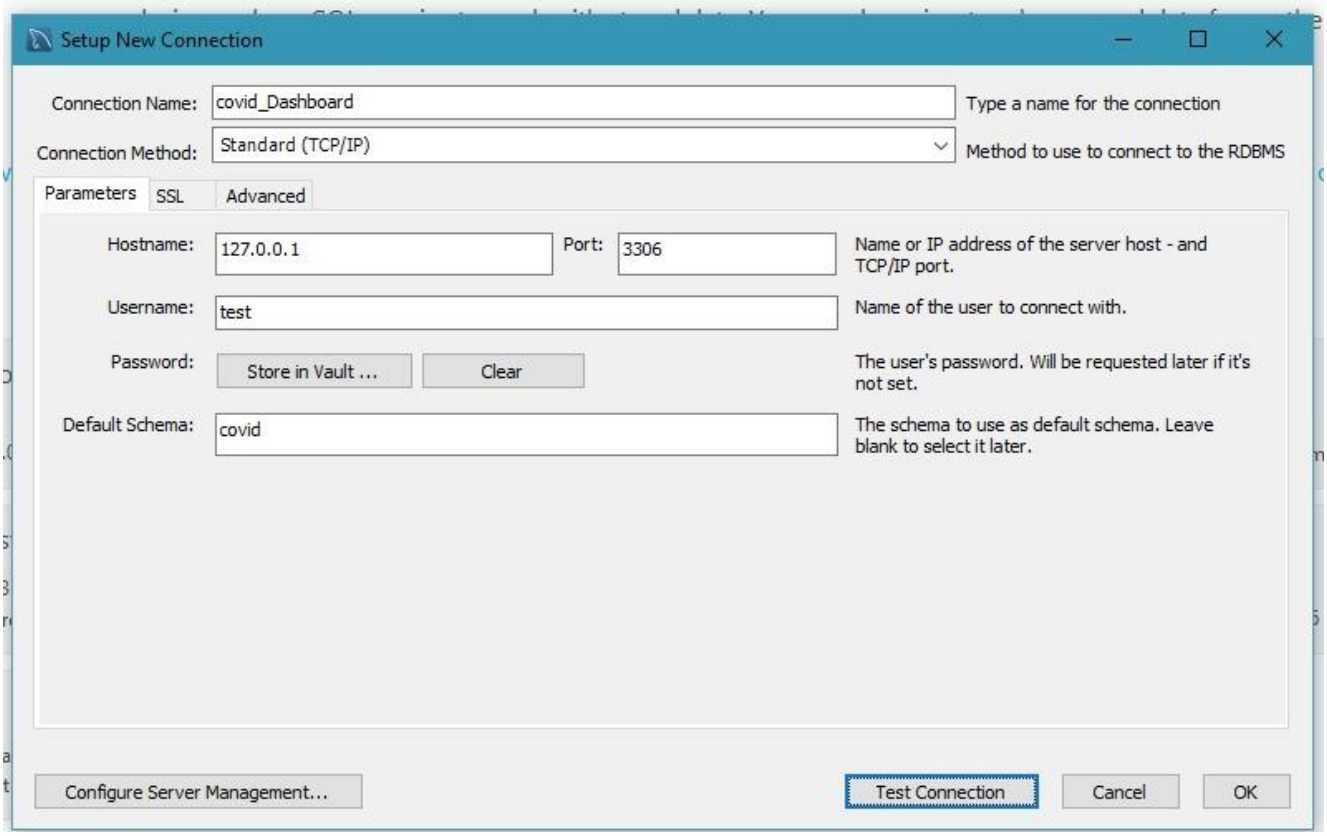
```
webserver)
  airflow initdb
  airflow connections --d --conn_id=fs_default
  airflow connections --d --conn_id=airflow_db
  airflow connections --add --conn_id=airflow_db --conn_type=mysql --conn_host=db --conn_schema=covid --conn_login=test --conn_password=test123 --conn_port=
  airflow connections --add --conn_id=fs_default --conn_type=fs --conn_extra='{"path": "/home/airflow/monitor/"}'
  if [ "$AIRFLOW__CORE__EXECUTOR" = "LocalExecutor" ] || [ "$AIRFLOW__CORE__EXECUTOR" = "SequentialExecutor" ]; then
    # With the "Local" and "Sequential" executors it should all run in one container.
    airflow scheduler &
  fi
  exec airflow webserver
  ;;
```

Base de Datos

Generación de base de datos de forma manual

Para crear la base de datos se utilizará una importación de datos del ETL que generamos en el DAG de airflow del paso anterior con el objetivo de solo transmitir automáticamente la información a MySQL en el puerto 3306.

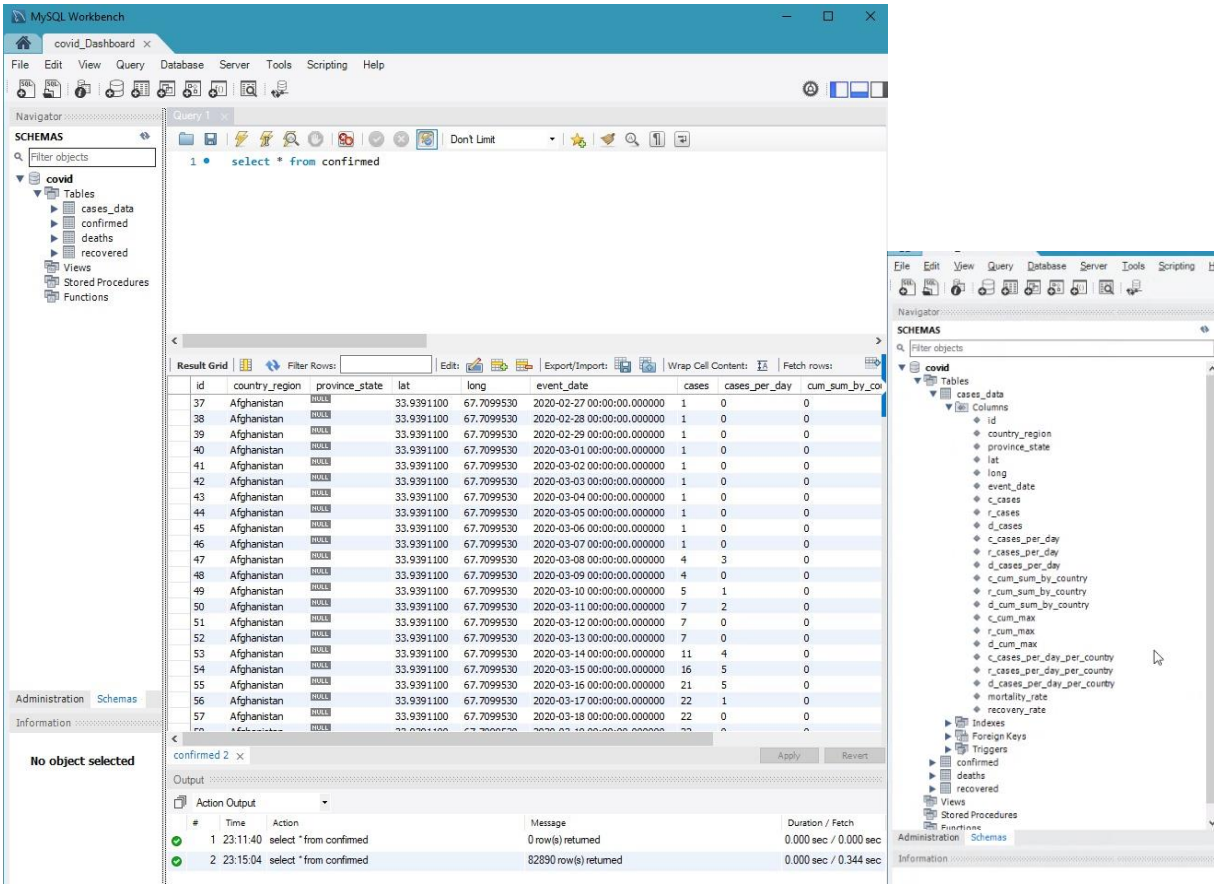




Con esto ya tenemos creadas la tabla global que se utilizará para generar parámetros de visualización para el streamlit a utilizar.

Tabla generada por Dag Transform

De la tabla generada podemos observar que la data transformada ha sido correctamente insertada según las variables que habíamos colocado para relacionarlas y generar las nuevas columnas para utilizarlas en streamlit.



Streamlit

Uso de Streamlit

Para la representación visual de nuestro dashboard se utilizará streamlit que es un marco de aplicación de código abierto para equipos de ciencia de datos y aprendizaje automático. Es una biblioteca que hace fácil crear aplicaciones web para mostrar resultados análisis propios de una forma gráfica elegante y más detallada a como el usuario requiera.

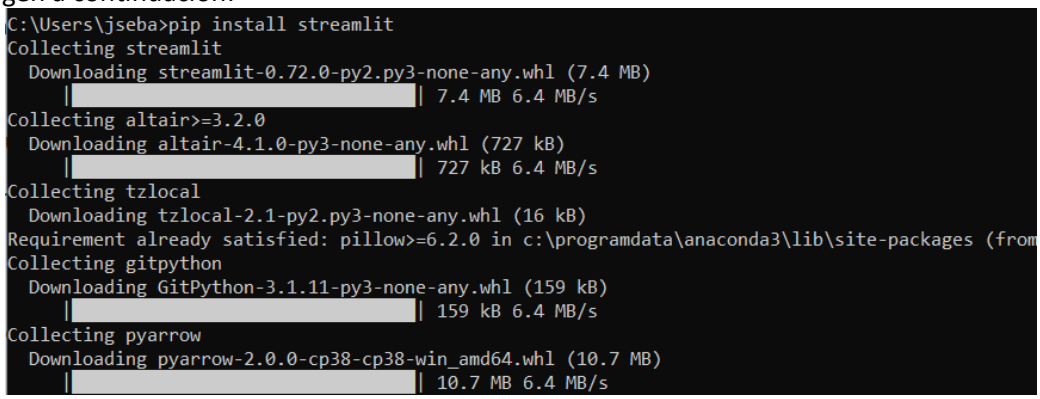
Documentación de Streamlit

Para mayor información de la documentación que tiene actualmente Streamlit se puede visitar el siguiente link el cual ofrece toda la recopilación de funciones y parámetros que maneja esta herramienta.
https://docs.streamlit.io/_/downloads/en/0.62.0/pdf/

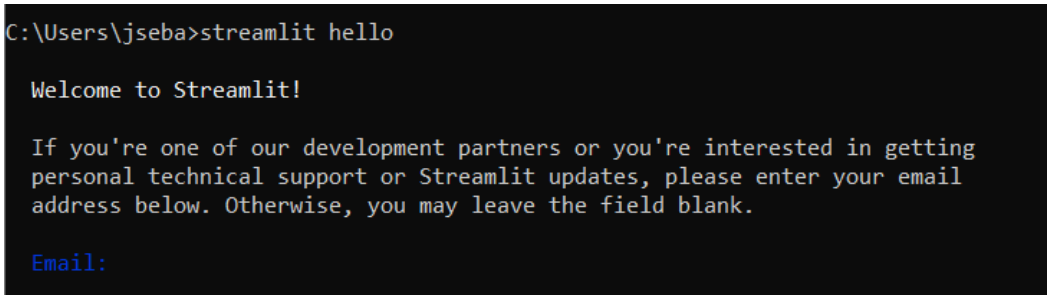
Instalación de Streamlit

Para realizar la instalación de esta biblioteca realizamos el siguiente procedimiento:

- Realizar la instalación por medio de un entorno virtual de Python o directamente en el cmd como en la imagen a continuación:



- Realizamos el primer comando de Streamlit para verificar que esté debidamente instalado con Streamlit hello



Para esta aplicación utilizando streamlit nos centramos en realizar un dashboard que muestre datos como la mortalidad según país, los países con más casos reportados, número de casos totales por provincia, etc…

Uso de Streamlit para dashboard

Se utilizó Streamlit como una herramienta para realizar la visualización de los casos COVID-19 hasta la actualidad en la que se encuentra la base de datos. Para ellos realizamos

- Datos de casos confirmados, recuperados y muertes por medio de tablas.
- Comparativas entre casos recuperados por medio de gráficos.
- Gráfico de burbujas que indican:
    - Casos confirmados por país.
    - Casos de recueprados por país.
    - Casos de muertes por país.

Resultados obtenidos

Product Development: Project

Este dashboard permitirá visualizar la situación del COVID-19

En esta primera tabla podrá visualizar los casos a la fecha máxima cargada

	State	latitude	longitude	Confirmed	Recovered	Deaths
14	Iran	32.4279	53.6880	866821	610406	45255
15	South Africa	-30.5595	22.9375	769759	711195	20968
16	Ukraine	48.3794	31.1656	653442	303782	11423
17	Belgium	50.8333	4.4699	559902	0	15755
18	Chile	-35.6751	-71.5430	542080	517524	15106
19	Iraq	33.2232	43.6793	537457	467654	11996
20	Indonesia	-0.7893	113.9213	502110	422386	16002
21	Czechia	49.8175	15.4730	496638	405982	7360
22	Netherlands	52.1326	5.2913	489818	0	8945
23	Turkey	38.9637	35.2433	453535	377891	12511
24	Bangladesh	23.6850	90.3563	449760	364611	6416

El coronavirus o COVID-19, es una enfermedad infecciosa causada por un coronavirus descubierto recientemente. La mayoría de personas afectadas por el virus experimentarán síntomas moderados. Sin embargo, la enfermedad puede presentar formas graves en pacientes con comorbilidades o edades avanzadas.

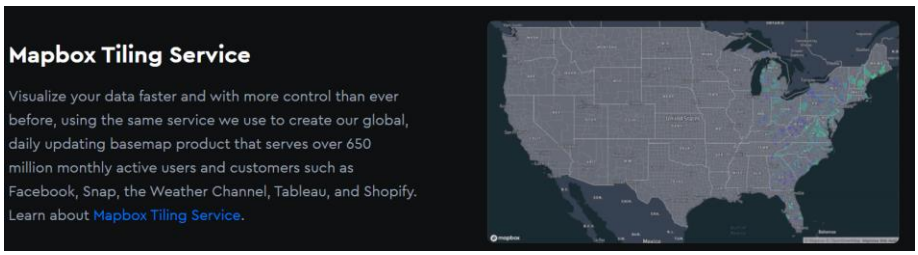
Tabla de visualización para la situación por estado en general de casos confirmados, recuperados y muertes.

Funciona básicamente con un scroll para ver el total del cuadro sobre cada estado y casos.

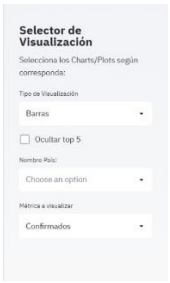
Mapa de casos mundiales a la última fecha en sistema

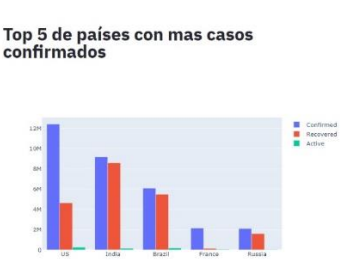
Mapa de casos totalizados hasta la última fecha de la que se tiene registro en la base de datos.

Se utiliza la librería mapbox la cual nos brinda la referencia gráfica con su geolocalización insertada que nos brinda un mayor detalle al momento de plotear la información scon sus logitudes y latitudes.



Streamlit ofrece distintas formas de graficar nuestros datos con cierta estructura y al decidirse utilizar histogramas y bubble map plot se lograron obtener los siguientes resultados. Se deja la documentación al lado de cada resultado el cual indica la forma en que se llegó a la solución. En el archivo main.py puede verse el detalle final del código utilizado.





13.1.5 Draw a histogram

Now that you've had a chance to take a look at the dataset and observe what's available, let's take things a step further and draw a histogram to see what Uber's busiest hours are in New York City.

1. To start, let's add a subheader just below the raw data section:

```
st.subheader('Number of pickups by hour')
```

2. Use NumPy to generate a histogram that breaks down pickup times binned by hour:

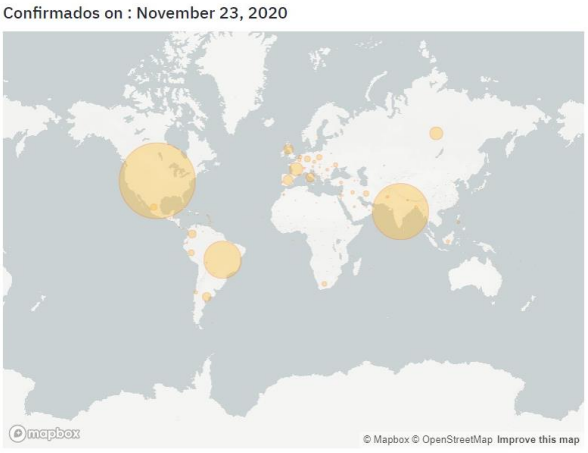
```
hist_values = np.histogram(
    data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))[0]
```

3. Now, let's use Streamlit's `st.bar_chart()` method to draw this histogram.

```
st.bar_chart(hist_values)
```

4. Save your script. This histogram should show up in your app right away. After a quick review, it looks like the busiest time is 17:00 (5 P.M.).

To draw this diagram we used Streamlit's native `bar_chart()` method, but it's important to know that Streamlit supports more complex charting libraries like Altair, Bokeh, Plotly, Matplotlib and more. For a full list, see supported charting libraries.



13.1.6 Plot data on a map

Using a histogram with Uber's dataset helped us determine what the busiest times are for pickups, but what if we wanted to figure out where pickups were concentrated throughout the city. While you could use a bar chart to show this data, it wouldn't be easy to interpret unless you were intimately familiar with latitudinal and longitudinal coordinates in the city. To show pickup concentration, let's use Streamlit `st.map()` function to overlay the data on a map of New York City.

1. Add a subheader for the section:

```
st.subheader('Map of all pickups')
```

2. Use the `st.map()` function to plot the data:

```
st.map(data)
```

3. Save your script. The map is fully interactive. Give it a try by panning or zooming in a bit.

After drawing your histogram, you determined that the busiest hour for Uber pickups was 17:00. Let's redraw the map to show the concentration of pickups at 17:00.

1. Locate the following code snippet:

```
st.subheader('Map of all pickups')
st.map(data)
```

2. Replace it with:

```
hour_to_filter = 17
filtered_data = data[data[DATE_COLUMN].dt.hour == hour_to_filter]
st.subheader(f'Map of all pickups at {hour_to_filter}:00')
st.map(filtered_data)
```

Conclusiones y Recomendaciones

- Streamlit es un ambiente óptimo para entornos visuales.
- Streamlit al ser una herramienta relativamente nueva no cuenta con mucha documentación desarrollada y una comunidad fuerte como lo es con Shiny por lo que requirió un poco más de investigación.
- La curva de aprendizaje con Streamlit es menor en comparación de Shinyapps por lo que fue ideal para realizar un entorno gráfico amigable y rápidamente.
- La forma de implementar Dags en airflow es un proceso extenso, complicado y que requiere de múltiples iteraciones para obtener un resultado esperado.
- Cuando hay acumulación de container se requiere de un equipo más robusto para procesarlo.