Documentación Proyecto Final
Product Development
Dashboard Covid-19

Maestría Data Science

Cuarto Trimestre 2020

Número grupo	**Nombre Integrante**	**Carné**
4	Elda Magally Calderon Motta	16003182
	Ligia Deyanira Aguilar Estrada	19004789
	Victor Rene Pérez Mayen	19005180

1 Objetivos

- Aplicar conocimientos adquiridos en la clase magistral para utilizar técnicas de diseño e implementación avanzadas en la construcción de un dashboard funcional siguiendo las especificaciones de herramientas y características dadas.
- Combinar el uso de herramientas para el desarrollo de software como lo es el uso de lenguatjes de programación (R y Python), bases de datos (PostgreSQL y MySQL) y herramientas de ETL (Airflow).
- Utilizar técnicas de análisis y procesamiento de datos para el correcto despliegue de la información presentada en el dashboard.

2 Resumen

El presente proyecto consiste en poder visualizar la cantidad de casos confirmados, casos fallecidos y casos recuperados del virus COVID-19. Para ello se utilizó un dashboard de Shiny Apps creado en un contenedor de Docker, este dashboard despliega los datos almacenados en una Base de Datos de MySQL la cual tiene comunicación con nuestro dashboard, dicha base de datos tiene la característica que también se encuentra ubicada en un contenedor de Docker; dicha base de datos recibe una ingesta de datos por medio de Airflow , ubicado entro del contenedor de Docker ya mencionado. La librería de Airflow contiene funciones para trabajar y desarrollar procesos de ETL para alimentar la base de datos. Se obtuvo como resultado una visualización a nivel global de los casos de COVID-19, el cual puede recibir información de manera dinámica gracias a las configuraciones y manejo de DAGS (Directed Acyclic Graph por sus siglas en inglés) de Airflow.

3 Introducción

Docker es una aplicación que simplifica el proceso de gestionar los procesos de aplicaciones en contenedores. Los contenedores le permiten ejecutar sus aplicaciones en procesos aislados de recursos. Se parecen a las máquinas virtuales, sin embargo, los contenedores son más portátiles, tienen más recursos y son más dependientes del sistema operativo host.

Para el proyecto Docker toma un papel fundamental en su desarrollo debido a su portabilidad y fácil manejo de contenedores en los cuales se pueden combinar varias herramientas y recursos de software sin que estos tengan conflictos de compatibilidad. Docker adicionalmente es sencillo de mantener y aporta al proyecto servicios de red, manejo optimizado de los recursos a utilizar y autonomía.

A partir de Docker y Docker Compose se manejan diferentes servicios para que nuestro dashboard los utilice y se puedan realizar las diferentes tareas de carga y visualización de datos. Para este proyecto se combinan lenguajes de programación como lo es R y Python, manejo y mantenimiento de base de datos.

En el presente trabajo se explicará cada uno de los servicios contenidos en Docker así como los resultados y conclusiones obtenidos al desarrollar nuestro dashboard.

4 Desarrollo del Dashboard

4.1 Docker y Docker Compose

4.1.1 Dockerfile

Utilizamos un Dockerfile para que pueda crear de forma automática las imágenes necesarias con las instrucciones descritas en el. Se puede decir que es un archivo compuesto por los comandos que el usuario tendría que ingresar manualmente en la terminal de su máquina. Al ejecutar las instrucciones de este archivo se ejecutan de forma secuencial y automatizada.

A continuación se puede observar las instrucciones a ejecutar en nuestro Dockerfile.

```
# VERSION 1.10.9
# AUTHOR: Matthieu "Puckel_" Roisil
# DESCRIPTION: Basic Airflow container
# BUILD: docker build --rm -t puckel/docker-airflow .
# SOURCE: https://github.com/puckel/docker-airflow

FROM python:3.7-slim-buster
LABEL maintainer="Puckel_"

# Never prompt the user for choices on installation/configuration of packages
ENV DEBIAN_FRONTEND noninteractive
ENV TERM linux

# Airflow
ARG AIRFLOW_VERSION=1.10.9
ARG AIRFLOW_USER_HOME=/usr/local/airflow
ARG AIRFLOW_DEPS=""
ARG PYTHON_DEPS=""
ENV AIRFLOW_HOME=${AIRFLOW_USER_HOME}

# Define en_US.
ENV LANGUAGE en_US.UTF-8
ENV LANG en_US.UTF-8
ENV LC_ALL en_US.UTF-8
ENV LC_CTYPE en_US.UTF-8
ENV LC_MESSAGES en_US.UTF-8

```

```
COPY requirements.txt .

# Disable noisy "Handling signal" log messages:
# ENV GUNICORN_CMD_ARGS --log-level WARNING

RUN set -ex \
    && buildDeps=' \
        freetds-dev \
        libkrb5-dev \
        libsasl2-dev \
        libssl-dev \
        libffi-dev \
        libpq-dev \
        git \
    ' \
    && apt-get update -yqq \
    && apt-get upgrade -yqq \
    && apt-get install -yqq --no-install-recommends \
        $buildDeps \
        freetds-bin \
        build-essential \
        default-libmysqlclient-dev \
        apt-utils \
        curl \
        rsync \
        netcat \
        locales \
    && sed -i 's/^# en_US.UTF-8 UTF-8$/en_US.UTF-8 UTF-8/g' /etc/
    locale.gen \
    && locale-gen \
    && update-locale LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8 \
    && useradd -ms /bin/bash -d ${AIRFLOW_USER_HOME} airflow \
    && pip install -U pip setuptools wheel \
    && pip install 'SQLAlchemy==1.3.15' \
    && pip install pytz \
    && pip install pyOpenSSL \
    && pip install ndg-httpsclient \
    && pip install pyasn1 \
    && pip install apache-airflow[crypto,celery,postgres,hive,jdbc,
    mysql,ssh${AIRFLOW_DEPS:+,}${AIRFLOW_DEPS}]==${AIRFLOW_VERSION}
     \
    && pip install 'redis==3.2' \
    && pip install -r requirements.txt \
    && if [ -n "${PYTHON_DEPS}" ]; then pip install ${PYTHON_DEPS};
     fi \
    && apt-get purge --auto-remove -yqq $buildDeps \
    && apt-get autoremove -yqq --purge \
    && apt-get clean \
    && rm -rf \
        /var/lib/apt/lists/* \
        /tmp/* \
        /var/tmp/* \
        /usr/share/man \
        /usr/share/doc \
        /usr/share/doc-base

COPY script/entrypoint.sh /entrypoint.sh
```

```
COPY config/airflow.cfg ${AIRFLOW_USER_HOME}/airflow.cfg

RUN chown -R airflow: ${AIRFLOW_USER_HOME}
RUN mkdir -p /home/airflow/monitor
RUN chown -R airflow: /home/airflow/monitor

EXPOSE 8080 5555 8793 3306 33061

USER airflow
WORKDIR ${AIRFLOW_USER_HOME}
ENTRYPOINT ["/entrypoint.sh"]
CMD ["webserver"]
```

4.1.2 Docker compose

Compose es una herramienta para la cual permite ejecutar aplicaciones en Docker con varios contenedores y servicios corriendo. Con Compose, usa un archivo YAML para configurar los servicios de nuestro proyecto. Permite con un solo comando pueda crear e iniciar todos los servicios con las configuraciones definidas.

El archivo .YAML de nuestro proyecto levanta tres servicios que son:

- Airflow
- Postgres (Base de datos relacionada a Airflow)
- MySQL
- Shiny Server

En cada sección se verá como se encuentran configurados dentro del archivo docker-compose.yaml

4.2 Airflow

4.2.1 Configuración Airflow en docker compose

```
    postgres:
      image: postgres:9.6
      environment:
        - POSTGRES_USER=airflow
        - POSTGRES_PASSWORD=airflow
        - POSTGRES_DB=airflow
      logging:
        options:
          max-size: 10m
          max-file: "3"

    webserver:
      build: .
      restart: always
      depends_on:
```

```
      - postgres
    environment:
      - LOAD_EX=n
      - EXECUTOR=Local
    logging:
      options:
        max-size: 10m
        max-file: "3"
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./monitor:/home/airflow/monitor
      # - ./plugins:/usr/local/airflow/plugins
    ports:
      - "8080:8080"
    command: webserver
    healthcheck:
      test: [ "CMD-SHELL", "[ -f /usr/local/airflow/airflow-
webserver.pid ]" ]
      interval: 30s
      timeout: 30s
      retries: 3
```

4.2.2 DAGS

En Airflow, es un conjunto de tareas que se desean ejecutar, todas estas tareas se encuentran relacionadas unas con otras y se encuentran organizadas para determinar el momento o las acciones que hacen que cada tarea se ejecute.

En esta situación cada DAG se encuentra definido en un archivo de código escrito en el lenguaje de programación Python. A continuación se puede ver cada archivo de Python dentro de la carpeta de DAGS de nuestro proyecto. Cada archivo gobierna y determina las funciones de cada tarea para cargar una tabla diferente para nuestra base de datos.

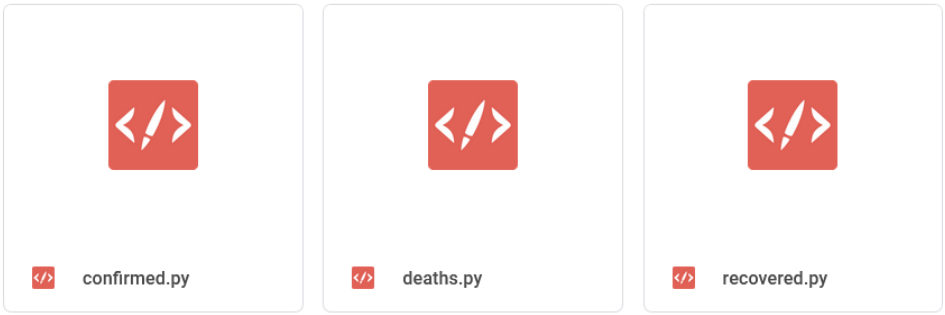

Figure 1: DAGS - Scripts de Python

A continuación se presenta un ejemplo de la construcción de cada DAG para el proyecto.

```
import os
import pandas as pd
from airflow import DAG
from airflow.contrib.hooks.fs_hook import FSHook
from airflow.contrib.sensors.file_sensor import FileSensor
from airflow.hooks.mysql_hook import MySqlHook
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago


FILE_CONNECTION_ID = "fs_default"
FILE_NAME = "time_series_covid19_confirmed_global.csv"
OUTPUT_TRANSFORM_FILE = "_confirmed_tmp.csv"

dag = DAG('confirmed_dag', description='ETL for confirmed table cases',
          default_args={
              'owner': 'elda.calderon',
              'depends_on_past': False,
              'max_active_runs': 1,
              'start_date': days_ago(5)
          },
          schedule_interval='0 1 * * *',
          catchup=False)

file_sensor_task = FileSensor(dag = dag,
                              task_id = 'file_sensor',
                              fs_conn_id= FILE_CONNECTION_ID,
                              filepath= FILE_NAME,
                              poke_interval= 10,
                              timeout= 300
                              )

def transform_func(**kwargs):
    folder_path = FSHook(conn_id = FILE_CONNECTION_ID).get_path()
    file_path = f"{folder_path}/{FILE_NAME}"
    destination_file = f"{folder_path}/{OUTPUT_TRANSFORM_FILE}"
    df = pd.read_csv(file_path, header = 0, encoding= 'ISO-8859-1')
    df.to_csv(destination_file, index = False)
    os.remove(file_path)
    return destination_file


transform_process = PythonOperator(dag = dag,
                                   task_id = 'transform_process',
                                   python_callable = transform_func,
                                   provide_context= True
                                   )

def insert_process(**kwargs):
    ti = kwargs['ti']
    source_file = ti.xcom_pull(task_ids = 'transform_process')
    db_connection = MySqlHook('airflow_db').get_sqlalchemy_engine()

    df = pd.read_csv(source_file)

```

```
    with db_connection.begin() as transaction:
        transaction.execute("DELETE FROM test.confirmed_table WHERE 1=1")
        df.to_sql("confirmed_table", con = transaction, schema = "test", if_exists= "replace", index = False)

    os.remove(source_file)

insert_process = PythonOperator(dag = dag,
                                task_id= 'insert_process',
                                provide_context= True,
                                python_callable = insert_process
                                )

file_sensor_task >> transform_process >> insert_process
```

4.3 Base de Datos - MySQL

MySQL es un sistema de gestión de bases de datos de código abierto. Gestiona sus datos usando una base de datos relacional y SQL (Lenguaje de consulta estructurada).

4.3.1 Configuración MySQL en docker compose

```
    db:
      image: mysql:5.7
      volumes:
        #- ./db_data:/var/lib/mysql
        - ./script/schema.sql:/docker-entrypoint-initdb.d/1.sql
      restart: always
      ports:
        - 33061:3306
      environment:
        MYSQL_ROOT_PASSWORD: test123
        MYSQL_DATABASE: test
        MYSQL_USER: test
        MYSQL_PASSWORD: test123
```

4.3.2 Tablas y ejecución de scripts

Para la conexión de la base de datos con los contenedores, se procedió a utilizar DataGrip. Se configuró la conexión para MySQL, con valores similares a los que se muestran a continuación:

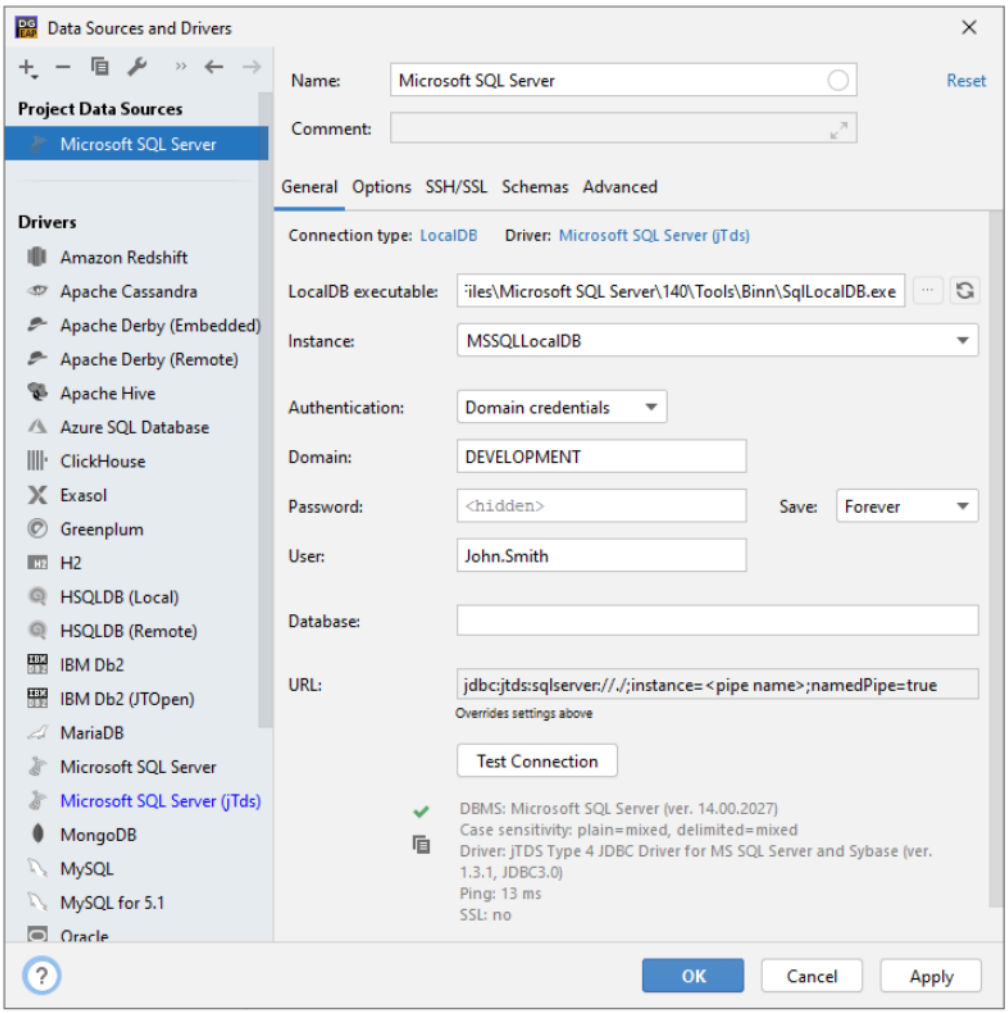

Figure 2: Conexión a la base de datos

Para poder accedera la base de datos se utilizó MySQL Workbenchutilizando los parámetros de conexión que se indicaron al momentode crearla.

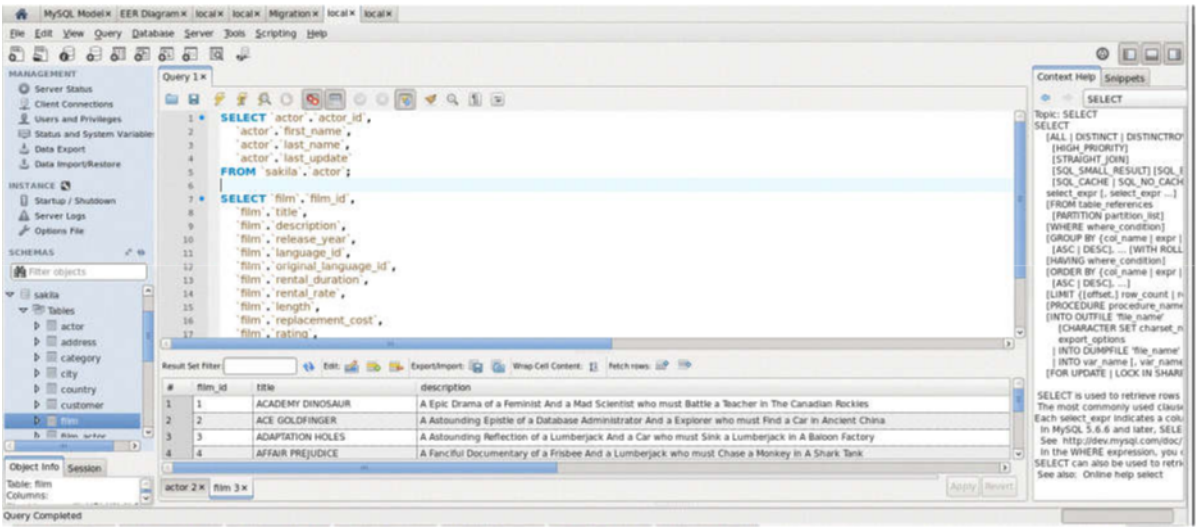

Figure 3: Workbench

4.3.3 Ingesta de informaciónpara la base de datos

Por medio de los DAGS de Airflow, se realizael proceso de ETL hacia la base de datos.Gracias a este proceso, la información de los casos confirmados, recuperados y fallecidosse cargan a la base de datos.

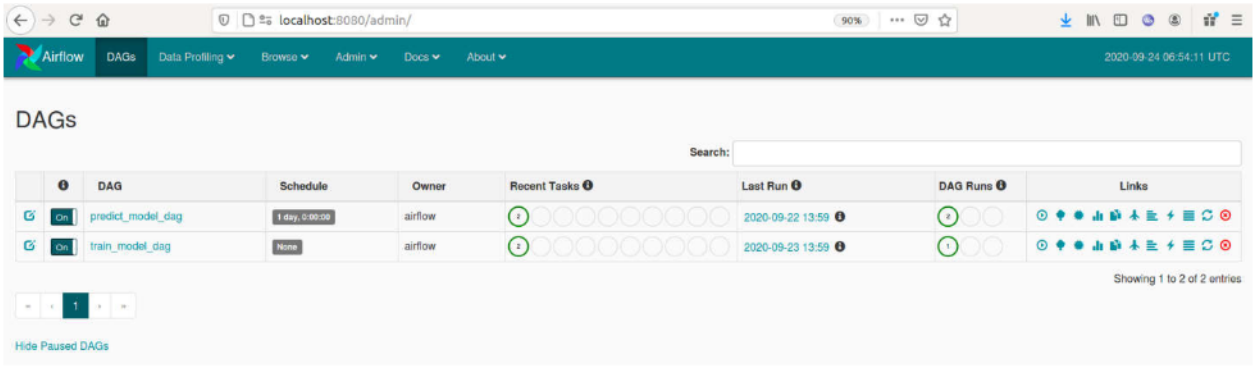

Figure 4: ETL

4.4 Shiny Server

En el apartado de Rsecreo undashboard por medio de Shiny Apps, este dashboard visualiza la cantidad de casos mencionados anteriormente por País, regióny por fecha. En el dashboard también se pueden visualizar la cantidad de casos por medio de una gráfica acumulativa, y los 5 países con más casos por fechas específicas.

4.4.1 Configuración Shiny Server en docker compose

```
    dashboard:
      build: ./dashboard
      ports:
        - 3838:3838 # izquierda puerto maquina, derecha puerto
    contenedor
      volumes:
      - ./prueba:/srv/shiny-server/
      - ./logs:/var/log/shiny-server/
```

4.4.2 Librerías

```
library(shiny)
library(leaflet)
library(RColorBrewer)
library(dplyr)
library(RMySQL)
```

4.4.3 Características del Dashboard

Panel con opciones donde el usuario puede determinar el tipo de datos a desplegar.

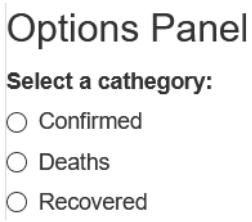

Figure 5: Option Panel

Selección de país y provincia (si es que está disponible la información) para el despliegue de las estadisticas y visualización del mapa.

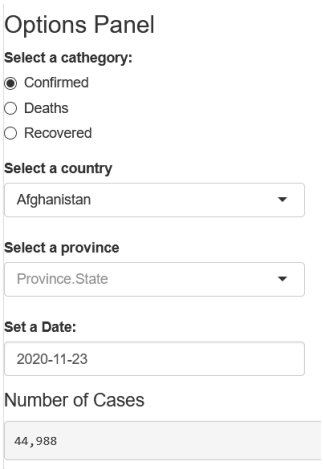

Figure 6: País, provincia y número de casos según las opciones seleccionadas

Gráfica acumulativa de los casos según el país y provincia seleccionados.

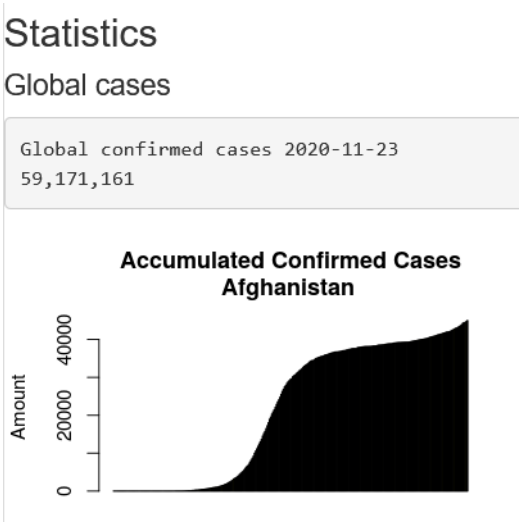

Figure 7: Estadísticas

Top 5 de los países con más casos según la fecha especificada.

Top 5 Country Accumulated Cases

Country.Region	X11.23.20
US	12418311
India	9177840
Brazil	6087608
France	2144979
Russia	2096749

Figure 8: Top 5

Mapa interactivo en el cúal se puede ver la información por país.

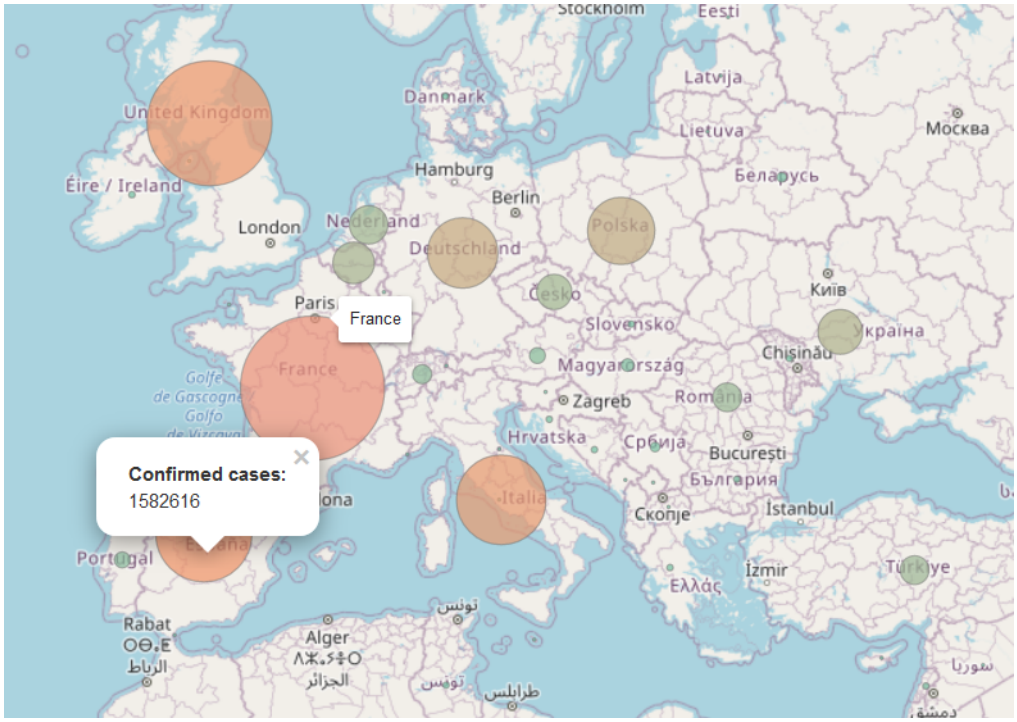

Figure 9: Mapa interactivo

5 Justificación

El proyecto presentado se realizó con el fin de poder realizar una conexión entre los DAGS de Airflow con una base de datos con contenedores hechos en Docker, y que los datos se puedan visualizar de una manera ordenada y simplificada para que al usuario final pueda sacar sus propias conclusiones o inferencias a partir de la data mostrada. Utilizando dashboards elaborados en Shiny Apps de R en conjunto con las otras herramientas descritas anteriormente el fin principal es que toda la aplicación sea portatil, fácil de mantener y el usuario pueda visualizar de manera asequible por medio de una virtualización óptima en cuanto al uso de recursos gracias a los contenedores administrados en Docker.

El despliegue de los casos recuperados, confirmados y fallecidos a causa del virus de COVID-19 son visualizados de tal forma que se espera que contribuya al entendimiento de este problema que nos ha afectado durante el presente año.

6 Recomendaciones

- Se recomienda utilizar contenedores de Docker para base de datos MySQL o SQL Server, ya que son los que tienen mayor cantidad de documentación.
- Es recomendable corroborar la correcta conexión entre los dashboards con los contenedores, para que la data sea visualizada en tiempo real.
- Se debe tener especial cuidado en la configuración de los DAGS para que no tenga errores el proceso de ETL en los contenedores.

7 Conclusiones

- Se instaló Docker, se trabajó con imágenes y contenedores haciendo push del material visto en el curso.
- Ahora se tiene una configuración de MySQL para base de datos la cual fácilmente puede ser adaptada para el desarrollo de otros proyectos.
- Se realizó una conexión de la base de datos con los contenedores de Docker usando DataGrip
- Se hizo la conexión de R server con la base de datos, ambos levantados en Docker container.
- Se demostró que Airflow es permite el proceso de ETL hacia la base de datos.