

Docker-Compose y Dockers a usar

Se utilizó el Docker que nos dieron en clase para la base de datos MySQL y Airflow

[GitHub - obedaeg/airflow: Airflow Test project](https://github.com/obedaeg/airflow: Airflow Test project)

Para esto usamos el Dockerfile – en las cuales se ponen los paquetes de postgres (que usa Airflow), Airflow y MySQL.

Para la parte de dashboards usamos Shiny por lo que usamos un Docker diferente y la configuración estaría en Dockerfile-shiny en la que usamos rocker/shiny. Luego de eso instalamos las librerías que necesitamos para lo que usamos en Shiny R y los paquetes de R.

También copiamos nuestro Dashboard al Docker donde se lanzan los Shiny app, exponemos el puerto y levantamos el servicio.

```
FROM rocker/shiny:4.0.5

# system libraries
# Try to only install system libraries you actually need
# Package Manager is a good resource to help discover system deps
RUN apt-get update && apt-get install -y \
    libcurl4-gnutls-dev \
    libssl-dev \
    libmysqlclient-dev \
    libarmadillo-dev \
    libgeos-dev \
    libudunits2-dev \
    libproj-dev \
    libfontconfig1-dev \
    libnode-dev \
    libjq-dev \
    libgdal-dev

# install R packages required
# Change the packages list to suit your needs
RUN R -e
'install.packages(c("magrittr", "rvest", "readxl", "dplyr", "maps", "ggplot2",
"reshape2", "ggiraph", "RColorBrewer", "leaflet", "plotly", "geojsonio", "shiny",
"shinyWidgets", "shinydashboard", "shinythemes", "plyr", "RMySQL", "DBI", "DT"),
repos="https://packagemanager.rstudio.com/cran/___linux___/focal/2021-04-23") '

EXPOSE 3838
# copy the app directory into the image
ADD Dashboard /srv/shiny-server/

# run app
CMD ["/usr/bin/shiny-server"]
```

En la parte de Docker-compose agregamos el Docker de shiny en donde especificamos el Dockerfile, con los paquetes y librerías que necesitamos. Por lo que por último mapeamos el puerto interno a externo para poder acceder desde nuestra máquina.

```
shiny:
  build:
    context: .
    dockerfile: Dockerfile-shiny
  image: rocker/shiny
  container_name: shiny
  restart: unless-stopped
  ports:
    - 3838:3838
```

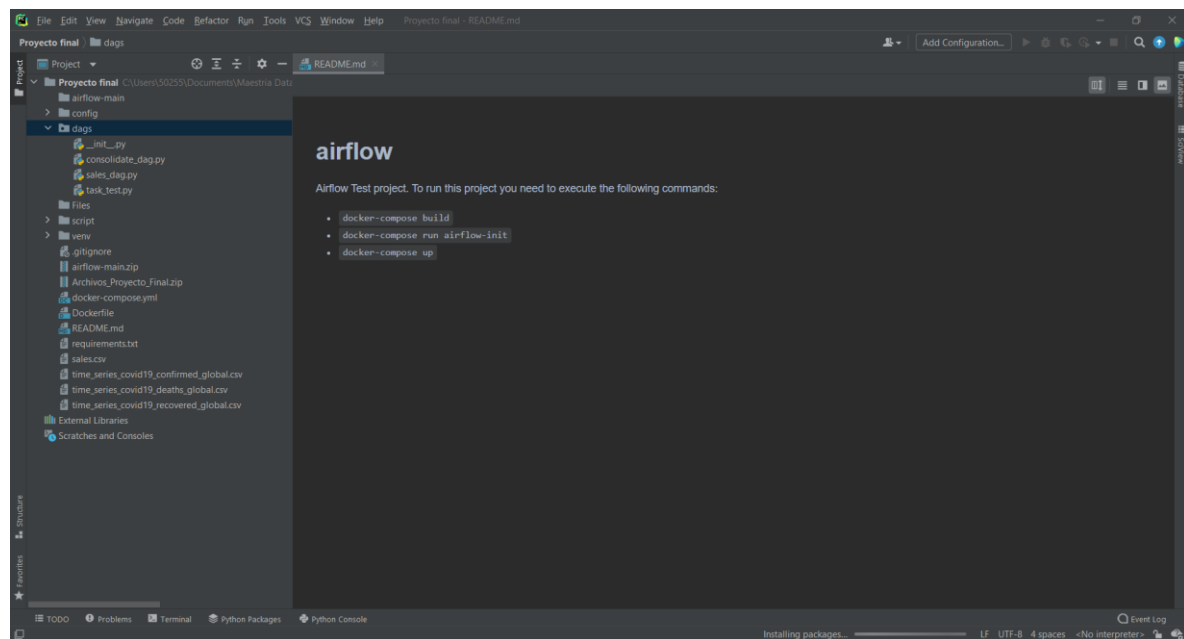
1. Para construirlo por primera vez corremos = Docker-compose build
 - a. Cuando corre la primera vez construye los Dockers y ya que están en el mismo compose comparten la misma red.
2. Para levantarlo le damos = Docker-compose up
 - a. Levantará todos los Dockers bajo la misma red virtual interna.

Carga de datos y la transformación de datos, (Airflow)

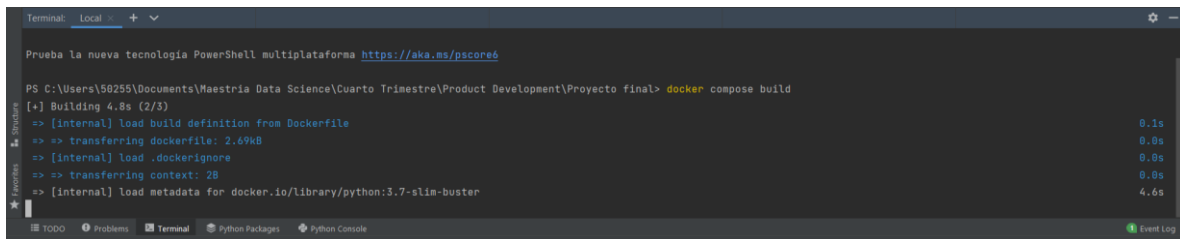
Para poder realizar la transformación es necesario contar con el ambiente listo para esto como nos indicó Obed en clase estaremos utilizando como base el repositorio compartido:

[GitHub - obedaeg/airflow: Airflow Test project](https://github.com/obedaeg/airflow)

Lo descargamos para usarlo de base del proyecto



Ejecutamos un **Docker compose Build**

A screenshot of a terminal window with a dark background. The title bar says 'Terminal: Local'. The text inside shows a command to build a Docker image: 'docker compose build'. The output shows progress: '[+] Building 4.8s (2/3)', followed by steps like 'load build definition from Dockerfile', 'transferring dockerfile: 2.69kB', 'load .dockerignore', 'transferring context: 2B', and 'load metadata for docker.io/library/python:3.7-slim-buster'. A progress bar on the left shows the build progress. The bottom of the terminal has tabs for 'TODO', 'Problems', 'Terminal', 'Python Packages', and 'Python Console'.

```
Terminal: Local + -
Prueba la nueva tecnologia PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\S0255\Documents\Maestria Data Science\Cuarto Trimestre\Product Development\Proyecto final> docker compose build
[+] Building 4.8s (2/3)
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 2.69kB 0.8s
=> [internal] load .dockerignore 0.8s
=> => transferring context: 2B 0.8s
=> [internal] load metadata for docker.io/library/python:3.7-slim-buster 4.6s
Event Log
```

Al finalizar un Docker corremos el comando **Docker compose up** para levantar la infraestructura de Airflow. En este ambiente estamos configurando una base de datos Postgres que es para el funcionamiento interno de Airflow y una base de datos MySQL que es donde estaremos guardando la data de los archivos de COVID que queremos trabajar.

Al tener listo el ambiente creo el archivo de Python en la carpeta de Dags llamada proyecto_dag.py donde estaremos trabajando el Dag que será nuestro ETL para los 3 archivos de COVID.

- Configurar los sensores para cada archivo.

```
sensor1 = FileSensor(task_id="file_sensor_covid_confirmed",
                    dag=dag,

                    filepath='time_series_covid19_confirmed_global.csv',
                    fs_conn_id='fs_default',
                    poke_interval=5,
                    timeout=60)

sensor2 = FileSensor(task_id="file_sensor_covid_deaths",
                    dag=dag,

                    filepath='time_series_covid19_deaths_global.csv',
                    fs_conn_id='fs_default',
                    poke_interval=5,
                    timeout=60)

sensor3 = FileSensor(task_id="file_sensor_covid_recovered",
                    dag=dag,

                    filepath='time_series_covid19_recovered_global.csv',
                    fs_conn_id='fs_default',
                    poke_interval=5,
                    timeout=60)
```

- Configuramos el operador que será el que manda a cargar los archivos el ETL se podría decir.

```
operador = PythonOperator(task_id='process_file',
                          dag=dag,
                          python_callable=process_file,
                          provide_context=True)
```

- Se configura el DAG con las características de corrida que vimos en clase

- ```
dag = DAG('dag_proyecto', description='primer Dag del proyecto',
 default_args={
 'owner': 'conchita.rainier',
 'depends_on_past': False,
 'max_active_runs': 1,
 'start_date': days_ago(5)
 },
 schedule_interval='0 1 * * *',
 catchup=False)
```

- se trabaja carga y manipulación de archivos con una mezcla de Python y SQL

- ```
def process_file(**kwargs):
    connection = MySQLHook('mysql_default').get_sqlalchemy_engine()
    file_path_confirmed =
    f"{FSHook('fs_default').get_path()}/time_series_covid19_confirmed_g
    lobal.csv"
    df_confirmed = pd.read_csv(file_path_confirmed)
    file_path_deaths =
    f"{FSHook('fs_default').get_path()}/time_series_covid19_deaths_glob
    al.csv"
    df_deaths = pd.read_csv(file_path_deaths)
    file_path_recovered =
    f"{FSHook('fs_default').get_path()}/time_series_covid19_recovered_g
    lobal.csv"
    df_recovered = pd.read_csv(file_path_recovered)
    #logger.info(df_recovered)

    #creo las fechas
    dates = df_confirmed.columns[4:]

    #realizo el melt para realizar el transpose de las fechas
    confirmed_df_long = df_confirmed.melt(
        id_vars=['Province/State', 'Country/Region', 'Lat',
        'Long'],
        value_vars=dates,
        var_name='Date',
        value_name='Confirmed'
    )
    deaths_df_long = df_deaths.melt(
        id_vars=['Province/State', 'Country/Region', 'Lat',
        'Long'],
        value_vars=dates,
        var_name='Date',
        value_name='Deaths'
    )
    recovered_df_long = df_recovered.melt(
        id_vars=['Province/State', 'Country/Region', 'Lat',
        'Long'],
        value_vars=dates,
        var_name='Date',
        value_name='Recovered'
    )
    logger.info(f'termino los melts')
```

```

logger.info(confirmed_df_long.tail())

#limpio los valores nulos por ceros
df_recovered['Recovered'] = df_recovered['Recovered'].fillna(0)
deaths_df_long['Deaths'] = deaths_df_long['Deaths'].fillna(0)
confirmed_df_long['Confirmed'] =
confirmed_df_long['Confirmed'].fillna(0)
#df_confirmed['Date'] = pd.to_datetime(df_confirmed['Date'],
errors='coerce')

#renombro las columnas con la estructura de la base de datos
confirmed_df_long = confirmed_df_long.rename(columns=COLUMNS1)
recovered_df_long = recovered_df_long.rename(columns=COLUMNS1)
deaths_df_long = deaths_df_long.rename(columns=COLUMNS1)
#logs para debuggear
logger.info(f'renombro columnas')
logger.info(confirmed_df_long.tail())
logger.info(f'insert')

#Creo la primer conexion a base de datos donde elimino las
tablas de confirmados e inserto el DF
with connection.begin() as transaction:
    transaction.execute('DELETE FROM test.confirmed where 1=1')
    transaction.execute('DELETE FROM test.resumen_confirmed
where 1=1')
    confirmed_df_long.to_sql('confirmed', con=transaction,
schema='test', if_exists='append', index=False)

logger.info(f'Records inserted {len(confirmed_df_long.index)}')

# Creo la segunda conexion a base de datos donde elimino las
tablas de recuperados e inserto el DF
with connection.begin() as transaction:
    transaction.execute('DELETE FROM test.recovered where 1=1')
    transaction.execute('DELETE FROM test.resumen_recovered
where 1=1')
    recovered_df_long.to_sql('recovered', con=transaction,
schema='test', if_exists='append', index=False)

logger.info(f'Records inserted {len(recovered_df_long.index)}')

# Creo la segunda conexion a base de datos donde elimino las
tablas de las muertes e inserto el DF
with connection.begin() as transaction:
    transaction.execute('DELETE FROM test.deaths where 1=1')
    transaction.execute('DELETE FROM test.resumen_deaths where
1=1')
    deaths_df_long.to_sql('deaths', con=transaction,
schema='test', if_exists='append', index=False)

logger.info(f'Records inserted {len(deaths_df_long.index)}')

#por comodidad y reducir el tiempo de desarrollo se realizo un
Storeprocedure en la base de datos en el que se realizan el resto
de transformaciones
#lo mandamos a llamar desde aqui.

```

```
with connection.begin() as transaction:
    transaction.execute('CALL TRANSFORMACION()')
```

- los scripts de las tablas utilizadas en base de datos están en el archivo de **SCHEMA.SQL**

```
• CREATE TABLE test.sales(
    id int primary key auto_increment,
    order_number int,
    quantity_ordered int,
    price_each decimal(20,2),
    order_line_number int,
    sales decimal(20,2),
    order_date datetime,
    status varchar(32),
    qtr_id int,
    month_id int,
    year_id int,
    product_line varchar(64),
    msrp int,
    product_code varchar(64),
    customer_name varchar(256),
    phone varchar(32),
    address_line_1 varchar(256),
    address_line_2 varchar(256),
    city varchar(32),
    state varchar(32),
    postal_code varchar(32),
    country varchar(32),
    territory varchar(32),
    contact_last_name varchar(64),
    contact_first_name varchar(64),
    deal_size varchar(32)
);
```

```
CREATE TABLE test.covid(
    Province_State varchar(32),
    Country_Region varchar(32),
    Lat DECIMAL(8,6),
    Lon DECIMAL(9,6),
    date_observation datetime,
    confirmed int,
    deaths int,
    recovered int);
```

```
CREATE TABLE test.confirmed(
    Province_State varchar(100),
    Country_Region varchar(100),
    Lat DECIMAL(8,6),
    Lon DECIMAL(9,6),
    Date varchar(10),
    confirmed int
);
```

```
CREATE TABLE test.deaths(
    Province_State varchar(100),
    Country_Region varchar(100),
    Lat DECIMAL(8,6),
    Lon DECIMAL(9,6),
```

```
Date varchar(10),
Deaths int
);

CREATE TABLE test.recovered(
Province_State varchar(100),
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date varchar(10),
Recovered int
);

CREATE TABLE test.resumen_confirmed(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date date,
confirmed int
);

CREATE TABLE test.resumen_deaths(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date date,
Deaths int
);

CREATE TABLE test.resumen_recovered(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date date,
Recovered int
);

CREATE TABLE test.consolidado(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date date,
confirmed int,
Deaths int,
Recovered int
);

CREATE TABLE test.resumen(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6),
Date date,
confirmed int,
Deaths int,
Recovered int,
confirmed_today int,
```

```

Deaths_today int,
Recovered_today int
);

CREATE TABLE test.países(
Country_Region varchar(100),
Lat DECIMAL(8,6),
Lon DECIMAL(9,6)
);
CREATE INDEX resumen_confirmed ON test.resumen_confirmed
(Date,Lat,lon);
CREATE INDEX resumen_deaths ON test.resumen_deaths (Date,Lat,lon);
CREATE INDEX resumen_recovered ON test.resumen_recovered
(Date,Lat,lon);
CREATE INDEX resumen ON test.resumen (Date, Country_Region);
CREATE INDEX consolidado ON test.consolidado (Date,
Country_Region);
CREATE INDEX resumen_confirmed_ ON test.resumen_confirmed
(Date,Country_Region);
CREATE INDEX resumen_deaths_ ON test.resumen_deaths
(Date,Country_Region);
CREATE INDEX resumen_recovered_ ON test.resumen_recovered
(Date,Country_Region);

```

- La Transformación que se dejó a nivel de base de datos en MySQL esta en un store procedure llamado transformación () y en el repositorio está el script **transformación.sql** Este SP se manda a llamar desde el DAG

```

DELIMITER //
CREATE PROCEDURE TRANSFORMACION ()
BEGIN
insert into resumen_confirmed
select Country_Region, max(lat) as lat, max(lon) as lon, str_to_date(Date,
'%m/%e/%y') as Date, sum(confirmed) confirmed
from confirmed
group by Country_Region, str_to_date(Date, '%m/%e/%y');

insert into resumen_deaths
select Country_Region, max(lat) as lat, max(lon) as lon, str_to_date(Date,
'%m/%e/%y') as Date, sum(Deaths) as Deaths
from deaths
group by Country_Region, str_to_date(Date, '%m/%e/%y');

insert into resumen_recovered
select Country_Region, max(lat) as lat, max(lon) as lon, str_to_date(Date,
'%m/%e/%y') as Date, sum(Recovered) as Recovered
from recovered
group by Country_Region, str_to_date(Date, '%m/%e/%y');

DELETE FROM consolidado where 1=1;

insert into consolidado
select c.Country_Region, c.Lat, c.Lon, c.Date, c.confirmed, d.Deaths, r.Recovered
from resumen_confirmed c left join resumen_deaths d on
c.Date = d.Date and
c.Lat = d.Lat and
c.Lon = d.Lon
left join resumen_recovered r on
c.Date = r.Date and
c.Lat = r.Lat and
c.Lon = r.Lon ;

DELETE FROM resumen where 1=1;

```



```

insert into resumen
SELECT R1.Country_Region, R1.Lat, R1.Lon, R1.Date, R1.confirmed, R1.Deaths,
R1.Recovered,
IFNULL(R1.confirmed,0)-IFNULL(R2.confirmed,0) as confirmed_today,
IFNULL(R1.Deaths,0)-IFNULL(R1.Deaths,0) as deaths_today,
IFNULL(R1.Recovered,0)-IFNULL(R2.Recovered,0) as recovered_today
FROM consolidado R1 JOIN consolidado R2 ON
    R1.Date = date_add(R2.Date,INTERVAL +1 DAY) and
    R1.Country_Region = R2.Country_Region ;

update resumen
set confirmed_today=confirmed,
    Deaths_today=Deaths,
    recovered_today=recovered
where date=(select min(date) from consolidado);

DELETE FROM paises where 1=1;

insert into paises
select Country_Region, lat, lon
from resumen
group by Country_Region, lat, lon;

END
//
DELIMITER ;

```

Del lado de la parte Web de Airflow. Debemos configurar las 2 conexiones que utilizaremos.

The screenshot shows the Apache Airflow web interface. The top navigation bar includes 'Airflow', 'DAGs', 'Data Profiling', 'Browse', 'Admin', 'Docs', and 'About'. The 'Admin' menu is open, showing options like 'Pools', 'Configuration', 'Users', 'Connections', 'Variables', and 'XComs'. The 'DAGs' section is active, displaying a table with columns: DAG, Schedule, Recent Tasks, Last Run, DAG Runs, and Links. The table lists three DAGs: 'consolidate_dag', 'sales_ingestion_dag', and 'task_test'. The 'task_test' DAG is highlighted. At the bottom, there is a pagination control showing 'Showing 1 to 3 of 3 entries' and a 'Hide Paused DAGs' link.

- Sensor

Connection [edit]

List Create Edit

Conn Id * fs_default

Conn Type File (path)

Host

Schema

Login

Password

Port

Extra {"path": "/home/airflow/monitor"}

Save Save and Add Another Save and Continue Editing Cancel

- MySQL

Connection [edit]

List Create Edit

Conn Id * mysql_default

Conn Type MySQL

Host db

Schema test

Login test

Password

Port 3306

Extra

Save Save and Add Another Save and Continue Editing Cancel

En esta ventana observamos los DAGs creados, como mencionamos al inicio del documento el dag se llama dag_proyecto.

Airflow DAGs Data Profiling Browse Admin Docs About 2021-12-07 05:33:29 UTC

DAGs

Search:

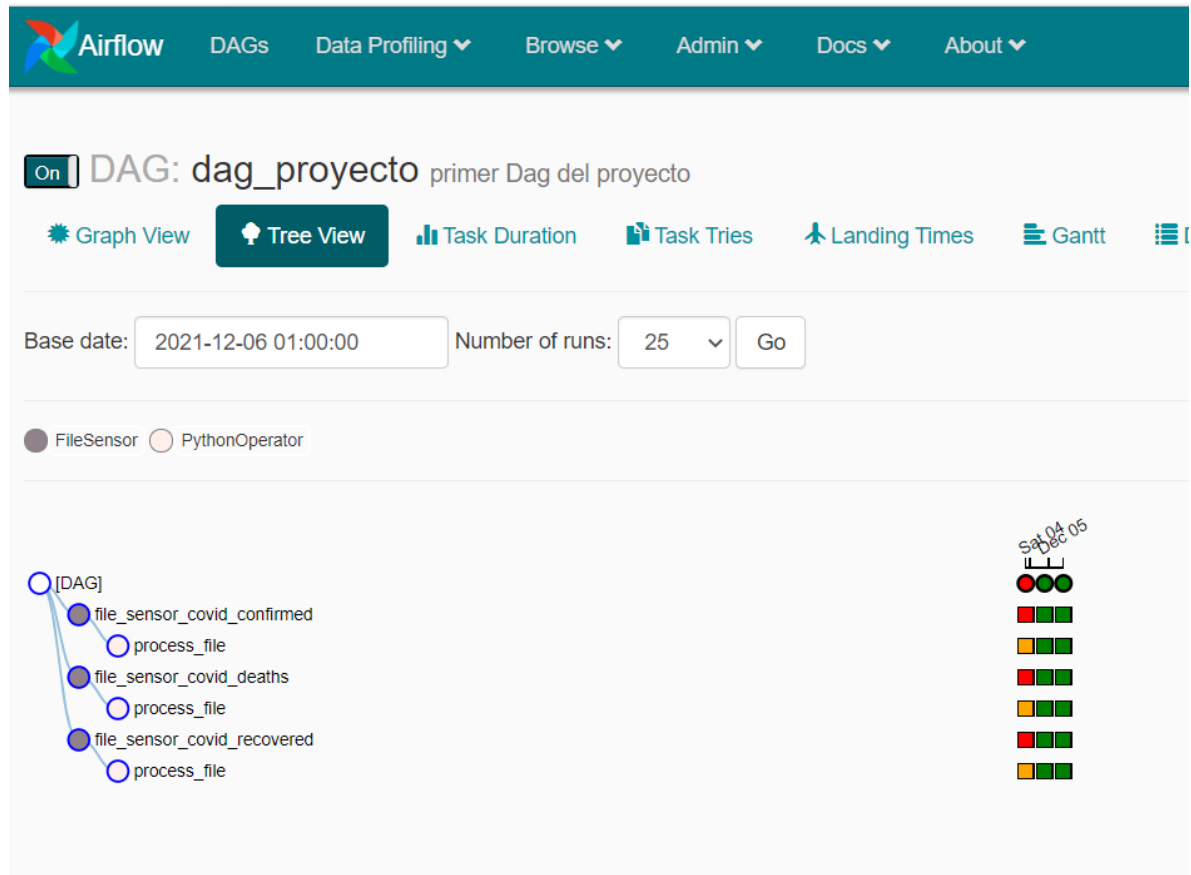
	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	consolidate_dag	@1***	obed.espinosa, airflow		2021-12-04 00:00		
	dag_proyecto	@1***	conchita.rainier		2021-12-06 01:00		
	sales_ingestion_dag	@1***	obed.espinosa		2021-12-06 01:00		
	task_test	@1***	obed.espinosa				

Showing 1 to 4 of 4 entries

« 1 »

Hide Paused DAGs

Como podemos observar el árbol esta creado de la forma esperada y hemos corrido durante 3 días los procesos. La primera ejecución no fue exitosa ya que se tenían problemas en unas configuraciones y al solucionarlo ya los otros dos días han sido éxitos y los datos se ven cargados a nivel de base de datos.



Dashboard en Shiny.

Para esto usamos un mapa de calor y los ejemplos que encontramos es usando un archivo JSON en el cual se tiene mapeado el mundo entero allí tienen los valores de los diámetros de cada país para que podamos usarlo como valores para esto.

También como mejores prácticas en público con el archivo JSON también se tiene un archivo con LAT y LON oficial por cada país amarrando al archivo JSON, el cual nos da información adicional como continente al que pertenece el país que nos da más opciones de filtrado, que se usa en mapa de calor y en graficas.

```
countries = read.csv("input_data/json_countries_lat_lon.csv", check.names = FALSE, fileEncoding="UTF-8-BOM", sep = ";")
worldcountry = geojsonio::geojson_read("input_data/50m.geojson", what = "sp")
```

Luego de esto el Dashboard obtiene todos los datos de casos COVID de la base de datos con la tabla final con la información concentrada.

```

#Definicion de conexion
con <- dbConnect(MySQL(),
                  host="db", user="test", password="test123",
                  dbname="test", port=3306)

# funcion para obtener informacion de Mysql
GetValues <- function(Table) {
  queries <- sprintf("SELECT *
                      FROM %s", Table)
  res <- dbGetQuery(con, queries)
  dbDisconnect(con)
  return(res)
}

```

Ya que estamos en la misma red, solo colocamos el host = DB , que es la definición en el docker-compose con el puerto interno.

Para todos los datos alineados, encontramos que países en el data set de la base de datos es diferente al de JSON y los alineamos para que haga el mapeo correcto en el mapa.

Colocamos todas las funciones que servirán para graficas en el MAPA y graficas por filtro.

Luego hacemos cálculos adicionales que se usaran en la presentación de los datos, como seria Tasa de mortalidad y tasa de recuperación tanto acumulada, como por día, hacemos merge con el archivo de mapeo de JSON y con eso construimos una data set con la información para las diferentes opciones.

Creamos Data sets adicionales como agrupación por continente y global (Que serían todos los países).

Por último, tenemos el UI / SERVER en la cual definimos como se vería la parte grafica y los reactive que se usaran de acuerdo a los eventos que se registren.