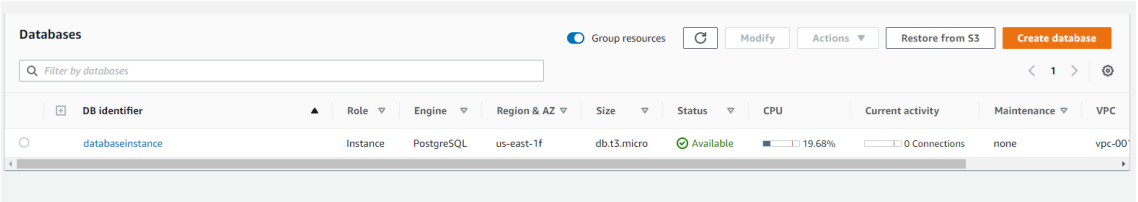
AWS Database

The server that provides the information is uploaded to a Postgres instance in Amazon Relational Database Service (RDS). The connection can be made using the following URL:

postgresql://london:as3fgd6@databaseinstance.c8611n47i7sr.us-east-1.rds.amazonaws.com:5432/database_ds4a

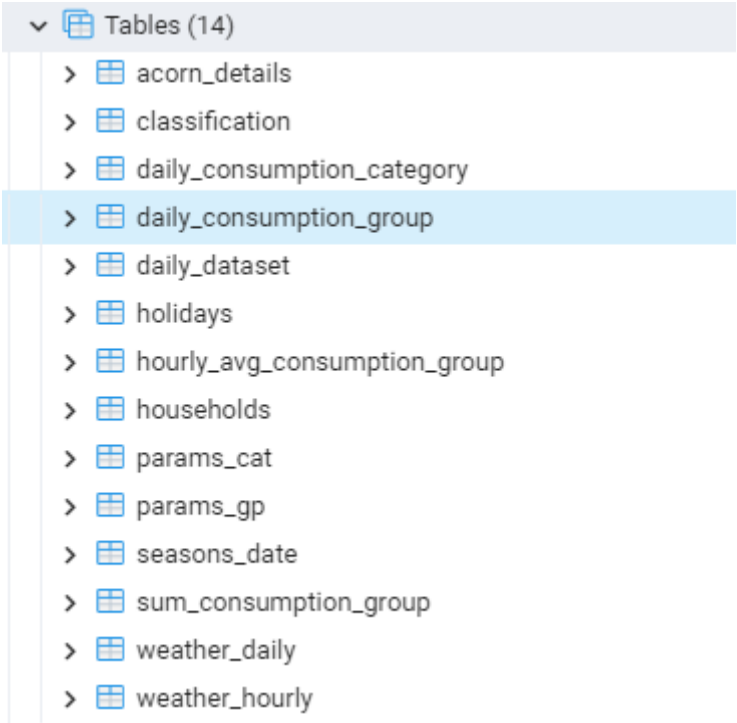
Base de Datos	database_ds4a
Usuario	london
Contraseña	xxxxxx
Instancia	databaseinstance.c8611n47i7sr.us-east-1.rds.amazonaws.com
Puerto	5432



For the correct operation of the dashboard, the instance must be turned on.

Due to the storage limits of the instance, it was necessary to update the database tables with the information transformed into the database. In the ***staging*** scheme.

The database is made up of:



Deployment:

Once the application is finished, it is necessary to use Docker to create an image and from this its respective container.

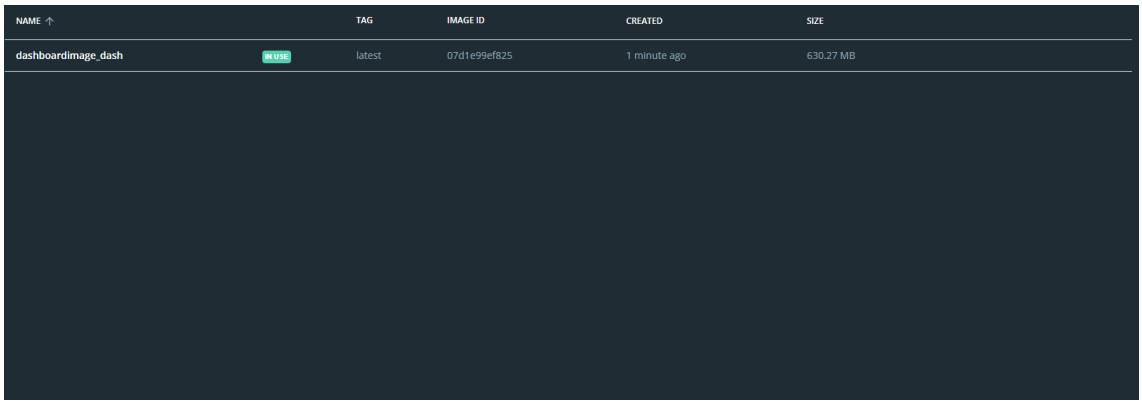
To create the image, and container and run the app, it was done using docker-compose, which instantiates the name of the Docker file, the name of the container and the application to open once the container is deployed. As well as instantiating port 8050. As follows:

```
version: '3'

services:


  dash:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: conjoint_dashboard
    command: app.py
    volumes:
      - .:/code
    ports:
      - "8050:8050"
```

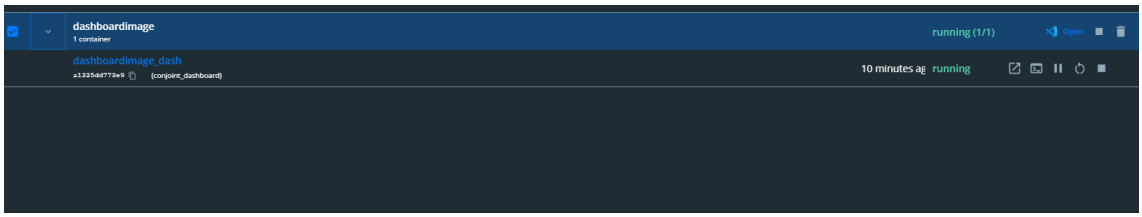
The package from which the libraries will be obtained is instantiated in the Docker file. Then, on the container server, the mkdir path is opened, the requirements and the files that make up the application are copied to the app folder. After doing the installation and update of pip for the installation of packages contained in the text file requirements. Finally, in the container in the app folder, the Python file app.py is opened, as follows:

```
FROM python:3.8-slim-buster


RUN mkdir wd
WORKDIR wd


COPY requirements.txt .
COPY ./app /app

RUN pip install --upgrade pip
RUN pip install -r requirements.t

WORKDIR "/app"

ENTRYPOINT ["python3"]
CMD [ "app.py" ]
```

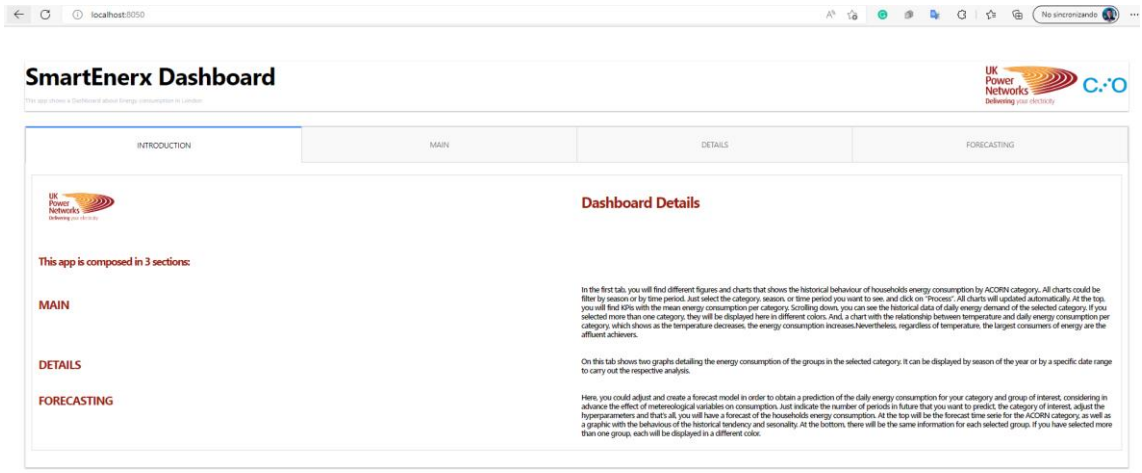
Once docker-compose is run, the image and container are created, and the app is run. In Docker Desktop the image is displayed like this:



Docker Desktop shows the creation of the container and the evidence that it runs.



Once verified that the application runs from the container:



The connection to the user who has the resources to use is proceeded through the AWS Management Console:

• Elastic Container Registry: Amazon repository where the code that was uploaded to the container is uploaded.

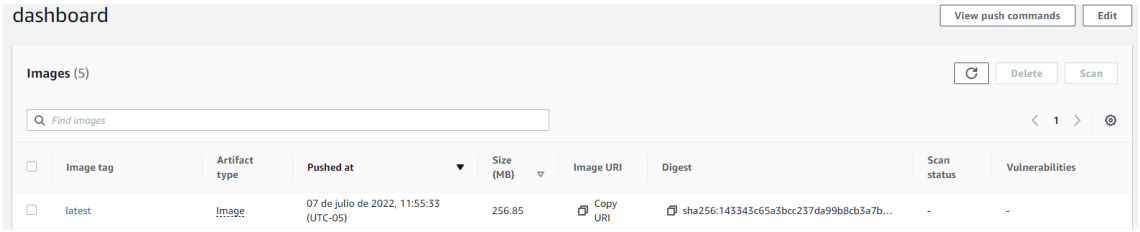
• Elastic Container Service: Amazon service where the cluster is created where the service (server) will be located that will allow the visualization of the dashboard in any part of the world.

The connection to the user from the AWS management Console is necessary to have the access key id and secret access key available, once the connection is created, it is started in the Elastic Container Registry to then tag the container with the path of the repository previously created in AWS.

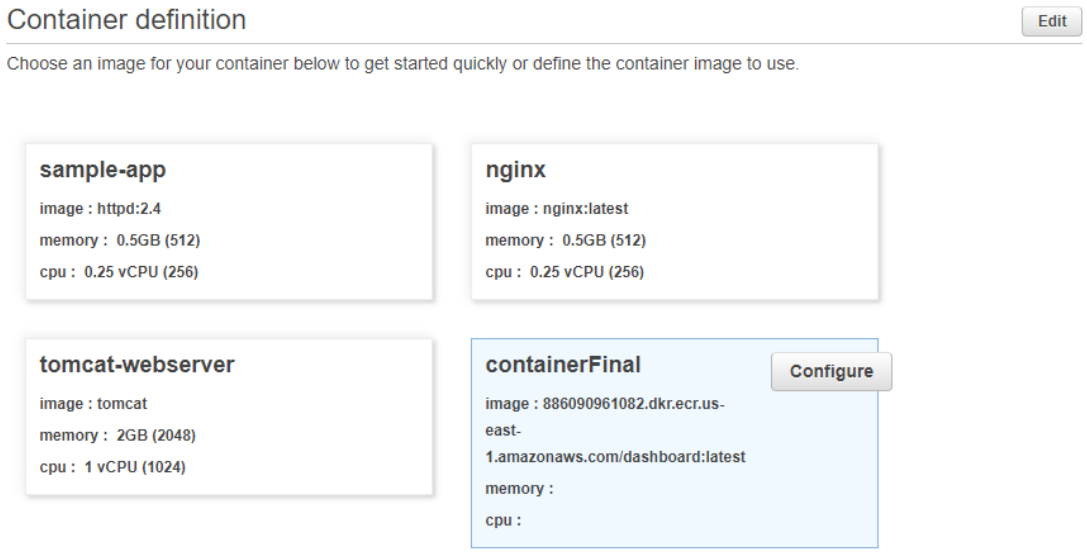
Finally, the image is uploaded to the repository, as follows:

```
5f70bf18a086: Pushed
5c53bde3c515: Pushed
7039139b1c2a: Pushed
3fdbfc2b6a61: Pushed
bca26bccfc2b: Pushed
c0670a146f48: Pushed
9b2fdb0faae2: Layer already exists
000a13d58723: Layer already exists
4f3455051638: Layer already exists
7ba8fd3fc230: Layer already exists
12cbeae46147: Layer already exists
latest: digest: sha256:143343c65a3bcc237da99b8cb3a7b9eda59844b61b2b376d3d46145842e81af9 size: 2831
```

On AWS it looks like this:



In Elastic Container Registry, a cluster is created where the definition of the container in AWS is specified:



Finally, the deployment is done, giving a successful status:

The application would already be running on the public IP.