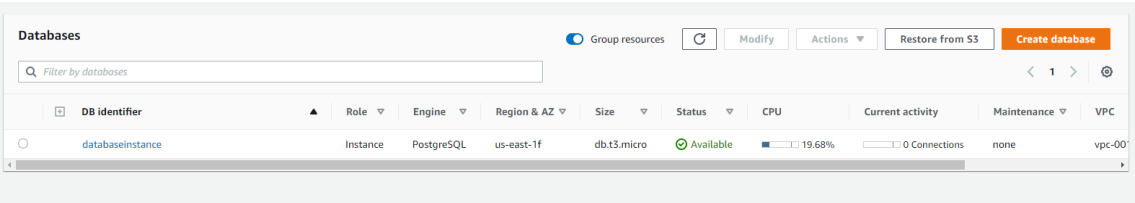


AWS Database

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:

- Tables (14)
- acorn\_details
  - classification
  - daily\_consumption\_category
  - daily\_consumption\_group
  - daily\_dataset
  - holidays
  - hourly\_avg\_consumption\_group
  - households
  - params\_cat
  - params\_gp
  - seasons\_date
  - sum\_consumption\_group
  - weather\_daily
  - weather\_hourly

## 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

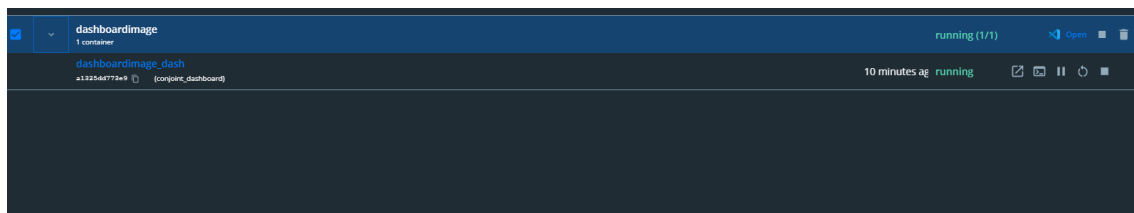
WORKDIR "/app"

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

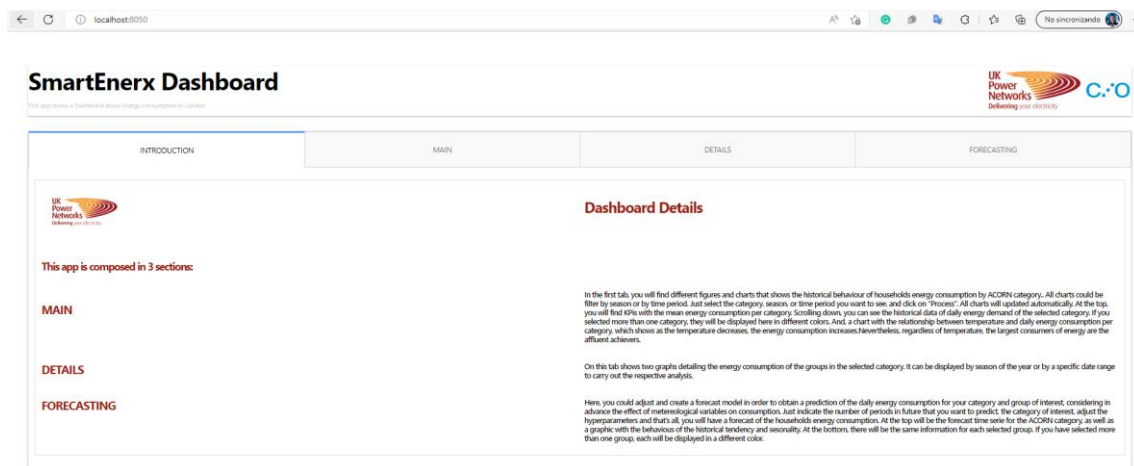
Once docker-compose is run, the image and container are created, and the app is run. In Docker Desktop the image is displayed like this:

NAME ↑		TAG	IMAGE ID	CREATED	SIZE
dashboardimage_dash	running	latest	07d1e99ef825	1 minute ago	630.27 MB

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:

dashboard

View push commandsEdit

Images (5)

Find images

< 1 >

<input type="checkbox"/>	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
<input type="checkbox"/>	latest	Image	07 de julio de 2022, 11:55:33 (UTC-05)	256.85	Copy URI	sha256:143343c65a3bcc237da99b8cb3a7b...	-	-

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

Container definition

Edit

Choose an image for your container below to get started quickly or define the container image to use.

sample-app

image : httpd:2.4

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

nginx

image : nginx:latest

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

tomcat-webserver

image : tomcat

memory : 2GB (2048)

cpu : 1 vCPU (1024)

containerFinal

image : 886090961082.dkr.ecr.us-east-1.amazonaws.com/dashboard:latest

memory :

cpu :

Configure

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

## Launch Status

We are creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

[Back](#) [View service](#)

## Additional features that you can add to your service after creation

### Scale based on metrics

You can configure scaling rules based on CloudWatch metrics.

Preparing service : 9 of 9 complete

ECS resource creation		complete
Cluster	clusterFinal	complete
Task definition	first-run-task-definition.7	complete
Service	containerFinal-service	complete
Additional AWS service integrations		complete
Log group	The log group [ /ecs/first-run-task-definition ] already exists	complete
CloudFormation stack	EC2ContainerService-clusterFinal	complete
VPC	vpc-09e6d336390eae64	complete
Subnet 1	subnet-07ae7c8677e112d82	complete
Subnet 2	subnet-0bb39aea8b22d3f91	complete
Security group	sg-0350b5508c7896d58	complete

The application would already be running on the public IP.