

1- Instalar Docker Community Edition

Diferentes opciones para cada sistema operativo

<https://docs.docker.com/>

Ejecutar el siguiente comando para comprobar versiones de cliente y demonio.

```
C:\Users\franc>docker version
Client:
 Version:           27.1.1
 API version:       1.46
 Go version:        go1.21.12
 Git commit:        6312585
 Built:             Tue Jul 23 19:57:57 2024
 OS/Arch:           windows/amd64
 Context:           desktop-linux
```

3- Obtener la imagen BusyBox

Ejecutar el siguiente comando, para bajar una imagen de DockerHub

```
C:\Windows\System32>docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
```

Verificar qué versión y tamaño tiene la imagen bajada, obtener una lista de imágenes locales:

docker images

```
C:\Windows\System32>docker images
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
busybox       latest    65ad0d468eb1  15 months ago  4.26MB

C:\Windows\System32>
```

4- Ejecutando contenedores

-Ejecutar un contenedor utilizando el comando run de docker:

docker run busybox

```
C:\Windows\System32>docker run busybox
C:\Windows\System32>
```

-Explicar porque no se obtuvo ningún resultado

Cuando ejecutas el comando `docker run busybox` sin especificar ningún comando adicional, Docker inicia un contenedor con la imagen de BusyBox y lo ejecuta. Sin embargo, dado que no le has proporcionado ninguna instrucción específica (como un comando para que BusyBox lo ejecute), el contenedor se inicia, no hace nada y luego se detiene inmediatamente.

-Especificamos algún comando a correr dentro del contenedor, ejecutar por ejemplo:

`docker run busybox echo "Hola Mundo"`

```
C:\Windows\System32>docker run busybox echo "Hola Mundo"
Hola Mundo
```

-Ver los contenedores ejecutados utilizando el comando ps:
docker ps

```
C:\Windows\System32>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

-Vemos que no existe nada en ejecución, correr entonces:

docker ps -a

Mostrar el resultado y explicar que se obtuvo como salida del comando anterior.

```
C:\Windows\System32>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED        STATUS       PORTS   NAMES
88778f7553b6   busybox   "echo 'Hola Mundo'"   About a minute ago   Exited (0)   About a minute ago   gallant_cohen
df3a74477968   busybox   "sh"       About a minute ago   Exited (0)   About a minute ago   sharp_edison
```

nos muestra contenedores ejecutados anteriormente

5- Ejecutando en modo interactivo

-Ejecutar el siguiente comando

docker run -it busybox sh

-Para cada uno de los siguientes comandos dentro de contenedor, mostrar los resultados:

ps

uptime

free

ls -l /

```
C:\Windows\System32>docker run -it busybox sh
/ # ps
PID   USER     TIME   COMMAND
    1  root      0:00   sh
    7  root      0:00   ps
/ # uptime
 16:53:26 up 6 min,  0 users,  load average: 0.00, 0.04, 0.01
/ # free
              total        used        free      shared  buff/cache   available
Mem:      8092864       571496       5997688         3168       1523680       7274144
Swap:      2097152           0       2097152
/ # ls -l /
total 40
drwxr-xr-x  2 root    root      12288 May 18  2023 bin
drwxr-xr-x  5 root    root        360 Aug 27 16:53 dev
drwxr-xr-x  1 root    root      4096 Aug 27 16:53 etc
drwxr-xr-x  2 nobody nobody     4096 May 18  2023 home
drwxr-xr-x  2 root    root      4096 May 18  2023 lib
lrwxrwxrwx  1 root    root         3 May 18  2023 lib64 -> lib
dr-xr-xr-x 238 root    root         0 Aug 27 16:53 proc
drwx----- 1 root    root      4096 Aug 27 16:53 root
dr-xr-xr-x 11 root    root         0 Aug 27 16:53 sys
drwxrwxrwt  2 root    root      4096 May 18  2023 tmp
drwxr-xr-x  4 root    root      4096 May 18  2023 usr
drwxr-xr-x  4 root    root      4096 May 18  2023 var
/ #
```

-Salimos del contenedor con:
exit

```
/ # exit  
C:\Windows\System32>
```

6- Borrando contenedores terminados

-Obtener la lista de contenedores

docker ps -a

```
C:\Windows\System32>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dbf11f4f8016	busybox	"sh"	2 minutes ago	Exited (0) 48 seconds ago		nervous_tesla
88778f7553b6	busybox	"echo 'Hola Mundo'"	4 minutes ago	Exited (0) 4 minutes ago		gallant_cohen
df3a74477968	busybox	"sh"	5 minutes ago	Exited (0) 5 minutes ago		sharp_edison

-Para borrar podemos utilizar el id o el nombre (autogenerado si no se especifica) de contenedor que se desee, por ejemplo:

docker rm elated_lalande

-Para borrar todos los contenedores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:

docker rm \$(docker ps -a -q -f status=exited)

docker container prune

```
C:\Windows\System32>docker container prune  
WARNING! This will remove all stopped containers.  
Are you sure you want to continue? [y/N] y  
Deleted Containers:  
dbf11f4f80163c5b8ee2baccce63b5965ac311eb04a50e02ba589c85213fe62b  
88778f7553b65d01386f8df836d93bdf922ed0ff57cac54600f34f657008cc6f  
df3a744779689d507c87f4edd223dca5b9d305671a1f0e49e22a498b54859755  
  
Total reclaimed space: 28B
```

7- Construir una imagen

Conceptos de DockerFile

Leer <https://docs.docker.com/engine/reference/builder/>

Describir las instrucciones

FROM: Especifica la imagen base para construir la nueva imagen.

RUN: Ejecuta un comando durante la construcción de la imagen.

ADD: Copia archivos/directorios desde la máquina local o URL al contenedor.

COPY: Copia archivos/directorios desde la máquina local al contenedor (sin manejar URLs).

EXPOSE: Informa a Docker del puerto en el que el contenedor escucha.

CMD: Define el comando que se ejecutará cuando se inicie el contenedor.

ENTRYPOINT: Establece el comando principal que siempre se ejecutará cuando el contenedor inicie, permitiendo que el contenedor funcione como un ejecutable.

A partir del código <https://github.com/ingsoft3ucc/SimpleWebAPI> crearemos una imagen.

Clonar repo

Crear imagen etiquetándola con un nombre. El punto final le indica a Docker que use el dir actual

docker build -t mywebapi .

```

=> => sha256:c7d9ee6cd01afe9aa80642e577c7cec9f5d87f88e5d70bd36fd61072079bc55b 1.79kB / 1.79kB 0.0s
=> => sha256:8a7717ff21c245feacd25dae5ff23306aae0e058578725ad987fd4b8509c36ec 1.37kB / 1.37kB 0.0s
=> => sha256:ec861be017681c5da7a762bd29eb07b4acd2391a9fa5ca9150ad6e5554c0506c 2.34kB / 2.34kB 0.0s
=> => sha256:534ba947de6ac79fd6168f4a93847954b23bab2782700bbfff7f31e61a03e8d4 32.46MB / 32.46MB 22.9s
=> => sha256:728328ac3bde9b85225b1f0d60f5c149f5635a191f5d8eae00e095d36ef9fd 31.43MB / 31.43MB 23.4s
=> => sha256:82bb7a80de578404d92b5ae5e67f3de90eab30027694d2609be35ad25b09e3bc 14.97MB / 14.97MB 13.4s
=> => sha256:f1b39e168c1c776458e172f157167607b9fd3cc550af8e6ff0a7dd363c1e64ea 153B / 153B 13.6s
=> => sha256:f194078e85f8008c084163778aaf266434f7182e0d4f783646f41b388c88a13a 10.12MB / 10.12MB 20.0s
=> => extracting sha256:728328ac3bde9b85225b1f0d60f5c149f5635a191f5d8eae00e095d36ef9fd 1.8s
=> => extracting sha256:82bb7a80de578404d92b5ae5e67f3de90eab30027694d2609be35ad25b09e3bc 117.7s
=> => extracting sha256:534ba947de6ac79fd6168f4a93847954b23bab2782700bbfff7f31e61a03e8d4 117.3s
=> => extracting sha256:f194078e85f8008c084163778aaf266434f7182e0d4f783646f41b388c88a13a 116.7s
=> [base 2/2] WORKDIR /app 1.5s
=> [final 1/2] WORKDIR /app 0.1s
=> [build 2/7] WORKDIR /src 4.0s
=> [build 3/7] COPY [SimpleWebAPI/SimpleWebAPI.csproj, SimpleWebAPI/] 0.1s
=> [build 4/7] RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj" 5.4s
=> [build 5/7] COPY . . 0.1s
=> [build 6/7] WORKDIR /src/SimpleWebAPI 0.1s
=> [build 7/7] RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /app/build 4.9s
=> [publish 1/1] RUN dotnet publish "SimpleWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false 2.5s
=> [final 2/2] COPY --from=publish /app/publish . 0.3s
=> exporting to image 0.3s
=> => exporting layers 0.2s
=> => writing image sha256:27cffe9ca77d63661df562c1b8587bf17c57591902a3358a73c0733026bb1ca9 0.0s
=> => naming to docker.io/library/mywebapi 0.0s

```

Revisar Dockerfile y explicar cada línea

#See <https://aka.ms/containerfastmode> to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.

FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base # (Usa la imagen base de ASP.NET Core 7.0 runtime para la etapa final)

WORKDIR /app # (Establece el directorio de trabajo en /app)

EXPOSE 80 # (Expone el puerto 80 para HTTP)

EXPOSE 443 # (Expone el puerto 443 para HTTPS)

EXPOSE 5254 # (Expone el puerto 5254, utilizado por la aplicación)

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build # (Usa la imagen base del SDK de .NET 7.0 para la etapa de construcción)

WORKDIR /src # (Establece el directorio de trabajo en /src)

COPY ["SimpleWebAPI/SimpleWebAPI.csproj", "SimpleWebAPI/"] # (Copia el archivo de proyecto al contenedor)

RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj" # (Restaura las dependencias del proyecto)

COPY . . # (Copia todos los archivos del proyecto al contenedor)

WORKDIR "/src/SimpleWebAPI" # (Cambia el directorio de trabajo a /src/SimpleWebAPI)

RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /app/build # (Compila la aplicación en modo Release y coloca los archivos en /app/build)

FROM build AS publish # (Crea una nueva etapa de construcción basada en la etapa build)

RUN dotnet publish "SimpleWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false # (Publica la aplicación en modo Release y coloca los archivos en /app/publish)

FROM base AS final # (Crea la etapa final basada en la imagen base)

WORKDIR /app # (Establece el directorio de trabajo en /app en la etapa final)

COPY --from=publish /app/publish . # (Copia los archivos publicados desde la etapa publish al directorio actual)

```
ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"] # (Establece el comando de inicio para ejecutar la aplicación)
#CMD ["/bin/bash"] #
```

Ver imágenes disponibles

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mywebapi	latest	27cffe9ca77d	7 minutes ago	216MB
busybox	latest	65ad0d468eb1	15 months ago	4.26MB

Ejecutar un contenedor con nuestra imagen

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker run -d -p 8080:80 mywebapi 3307bee3ce93c0df3d1ac3d10da8fe32d4a0672a37e7b7ef6ba6e66c3969483
```

Subir imagen a nuestra cuenta de dockerhub

7.1 Inicia sesión en Docker Hub

Primero, asegúrate de estar autenticado en Docker Hub desde tu terminal:

docker login

7.2 Etiquetar la imagen a subir con tu nombre de usuario de Docker Hub y el nombre de la imagen. Por ejemplo:

docker tag <nombre_imagen_local> <tu_usuario_dockerhub>/<nombre_imagen>:<tag>

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker tag mywebapi francotallone/mywebapi:latest
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> |
```

7.3 Subir la Imagen

Para subir la imagen etiquetada a Docker Hub, utiliza el comando docker push:

docker push <tu_usuario_dockerhub>/<nombre_imagen>:<tag>

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker push francotallone/mywebapi:latest
The push refers to repository [docker.io/francotallone/mywebapi]
b526a4218946: Pushed
5f70bf18a086: Pushed
db430c853fcd: Pushed
270f7fde987a: Pushed
4d29f6e29d10: Pushed
b4ec6db9c251: Pushed
ba941484fba1: Pushed
123eef91533f: Pushed
latest: digest: sha256:f99e467cda1f3985eeff400af6af551b623ffdd39b1de9fdf63c5d296039e5b3 size: 1995
```

7.4 Verificar la Subida

docker pull <tu_usuario_dockerhub>/<nombre_imagen>:<tag>

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker pull francotallone/mywebapi:latest
latest: Pulling from francotallone/mywebapi
Digest: sha256:f99e467cda1f3985eeff400af6af551b623ffdd39b1de9fdf63c5d296039e5b3
Status: Image is up to date for francotallone/mywebapi:latest
docker.io/francotallone/mywebapi:latest
```

8- Publicando puertos

En el caso de aplicaciones web o base de datos donde se interactúa con estas aplicaciones a través de un puerto al cual hay que acceder, estos puertos están visibles solo dentro del contenedor. Si queremos acceder desde el exterior deberemos exponerlos.

Ejecutar la siguiente imagen, en este caso utilizamos la bandera -d (detach) para que nos devuelva el control de la consola:

docker run --name myapi -d mywebapi

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker run --name myapi -d mywebapi
1956c6414111808557d8bc192e263fb50aa0a3287e8345068f92481a2d5c8550
```

Ejecutamos un comando ps:

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
1956c6414111   mywebapi   "dotnet SimpleWebAPI..." 32 seconds ago Up 31 seconds 80/tcp, 443/tcp, 5254/tcp
3307bee3ce93   mywebapi   "dotnet SimpleWebAPI..." 10 minutes ago Up 10 minutes 443/tcp, 5254/tcp, 0.0.0.0:8080->80/tcp
condescending_jepsen
```

Vemos que el contenedor expone 3 puertos el 80, el 5254 y el 443, pero si intentamos en un navegador acceder a <http://localhost/WeatherForecast> no sucede nada.

Procedemos entonces a parar y remover este contenedor:

docker kill myapi

docker rm myapi

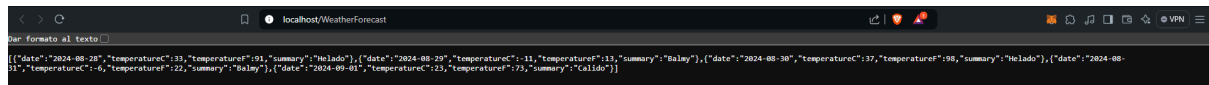
```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker kill myapi
myapi
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker rm myapi
myapi
```

Vamos a volver a correrlo otra vez, pero publicando el puerto 80

docker run --name myapi -d -p 80:80 mywebapi

```
PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker run --name myapi -d -p 80:80 mywebapi
dc8b22e1c9cd93242c64603f173e8533e308c6692d6086a3a929b30944f861bb
```

Accedamos nuevamente a <http://localhost/WeatherForecast> y vemos que nos devuelve datos.



```
[[{"date": "2024-08-28", "temperature": 22, "summary": "Helado"}, {"date": "2024-08-29", "temperature": 11, "summary": "Bale"}, {"date": "2024-08-30", "temperature": 37, "summary": "Helado"}, {"date": "2024-08-31", "temperature": 6, "summary": "Bale"}, {"date": "2024-09-01", "temperature": 23, "summary": "Calido"}]]
```

9- Modificar Dockerfile para soportar bash

Modificamos dockerfile para que entre en bash sin ejecutar automaticamente la app

#ENTRYPOINT ["dotnet", "SimpleWebAPI.dll"]

CMD ["/bin/bash"]

Rehacemos la imagen

docker build -t mywebapi .


```

PS C:\Users\franc\OneDrive\Desktop\SimpleWebAPI> docker build -t mywebapi .
[+] Building 9.3s (18/18) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 813B                             0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0 1.6s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0 2.1s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                   0.0s
=> [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a 0.0s
=> [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:c7d9ee6cd01afe9aa80642e577c7cec9f5d87f88e5d70bd36f 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 3.65kB                             0.0s
=> CACHED [build 2/7] WORKDIR /src                             0.0s
=> CACHED [build 3/7] COPY [SimpleWebAPI/SimpleWebAPI.csproj, SimpleWebAPI/] 0.0s
=> CACHED [build 4/7] RUN dotnet restore "SimpleWebAPI/SimpleWebAPI.csproj" 0.0s
=> [build 5/7] COPY . .                                        0.1s
=> [build 6/7] WORKDIR /src/SimpleWebAPI                      0.1s
=> [build 7/7] RUN dotnet build "SimpleWebAPI.csproj" -c Release -o /app/build 4.7s
=> [publish 1/1] RUN dotnet publish "SimpleWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false 2.2s
=> CACHED [base 2/2] WORKDIR /app                             0.0s
=> CACHED [final 1/2] WORKDIR /app                             0.0s
=> CACHED [final 2/2] COPY --from=publish /app/publish .      0.0s
=> exporting to image                                          0.0s
=> => exporting layers                                         0.0s
=> => writing image sha256:blcde54e1134ae8d34df7bad3eebaa65f42fecdeec16f865a10b8ee4a074c1f0 0.0s
=> => naming to docker.io/library/mywebapi                    0.0s

```

Corremos contenedor en modo interactivo exponiendo puerto

docker run -it --rm -p 80:80 mywebapi

```

PS C:\Windows\system32> docker run -it --rm -p 80:80 mywebapi
root@50705fd1ae94:/app#

```

Navegamos a <http://localhost/weatherforecast>

Vemos que no se ejecuta automáticamente

Ejecutamos app:

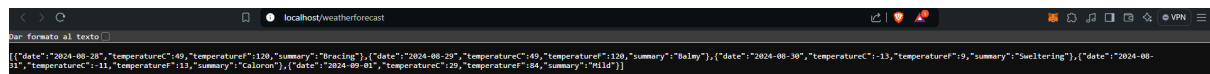
dotnet SimpleWebAPI.dll

```

PS C:\Windows\system32> docker run -it --rm -p 80:80 mywebapi
root@50705fd1ae94:/app# dotnet SimpleWebAPI.dll
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.

```

-Volvemos a navegar a <http://localhost/weatherforecast>



Salimos del contenedor

10- Montando volúmenes

Hasta este punto los contenedores ejecutados no tenían contacto con el exterior, ellos corrían en su propio entorno hasta que terminaran su ejecución. Ahora veremos cómo montar un volumen dentro del contenedor para visualizar por ejemplo archivos del sistema huésped:

Ejecutar el siguiente comando, cambiar myusuario por el usuario que corresponda. En Mac puede utilizarse /Users/miusuario/temp):

```
docker run -it --rm -p 80:80 -v /Users/miuser/temp:/var/temp mywebapi
```

```
PS C:\Users\franc\OneDrive\Desktop> docker run -it --rm -p 80:80 -v /Users/franc/temp:/var/temp mywebapi
root@eff689102779:/app# |
```

Dentro del contenedor correr


```
ls -l /var/temp
```

```
PS C:\Users\franc\OneDrive\Desktop> docker run -it --rm -p 80:80 -v C:/Users/franc/temp:/var/temp mywebapi
root@3788bbaade51:/app# ls -l /var/temp
total 0
```

```
touch /var/temp/hola.txt
```

Verificar que el Archivo se ha creado en el directorio del guest y del host.

```
root@3788bbaade51:/app# ls -l /var/temp
total 0
root@3788bbaade51:/app# touch /var/temp/hola.txt
```

Nombre	Fecha de modificación	Tipo	Tamaño
 hola	27/8/2024 14:51	Documento de te...	0 KB

11- Utilizando una base de datos

Levantar una base de datos PostgreSQL

```
mkdir $HOME/.postgres
```

```
docker run --name my-postgres -e POSTGRES_PASSWORD=mysecretpassword -v $HOME/.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
```

```
PS C:\Windows\system32> mkdir %env:USERPROFILE%\.postgres
>>

Directorio: C:\Users\franc

Mode                LastWriteTime         Length Name
----                -
d-----          27/8/2024   14:53             .postgres

PS C:\Windows\system32> docker run --name my-postgres -e POSTGRES_PASSWORD=mysecretpassword -v %env:USERPROFILE%\.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
>
Unable to find image 'postgres:9.4' locally
9.4: Pulling from library/postgres
519b14093c92: Pull complete
7ec0fe6664f6: Pull complete
7ac7ba0f7764: Pull complete
9e1155d037e2: Pull complete
febcbf7f8870: Pull complete
9c70c79412b5: Pull complete
5a35744496c5: Pull complete
27717922e067: Pull complete
36f9ac565550: Pull complete
4bf0a396f422: Pull complete
e4e86ea33e5: Pull complete
e8dd33eba6d1: Pull complete
51c81b3b2c20: Pull complete
2483ad76f5d7: Pull complete
Digest: sha256:42a7a6a647a602efa9592eddf56359000d079b93fa52c5d92244c50ac4a2ab9
Status: Downloaded newer image for postgres:9.4
507f17de9bf2 postgres:9.4 17 seconds ago Up 12 seconds 0.0.0.0:5432->5432/tcp
PS C:\Windows\system32> docker ps
>
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
3788bbaade51   mywebapi  "/bin/bash"              4 minutes ago Up 4 minutes   443/tcp, 0.0.0.0:80->80/tcp, 5254/tcp   distracted_gauss
PS C:\Windows\system32>
```

Ejecutar sentencias utilizando esta instancia

```
docker exec -it my-postgres /bin/bash
```

```
psql -h localhost -U postgres
```

#Estos comandos se corren una vez conectados a la base

\\


```
postgres=# \l
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	en_US.utf8	en_US.utf8	
template0	postgres	UTF8	en_US.utf8	en_US.utf8	=c/postgres +
template1	postgres	UTF8	en_US.utf8	en_US.utf8	postgres=C/c/postgres +

```
<3 rows>
```

create database test;

```
postgres=# create database test;
CREATE DATABASE
```

\connect test

```
postgres=# \connect test
You are now connected to database "test" as user "postgres".
```

create table tabla_a (mensaje varchar(50));

```
test=# create table tabla_a (mensaje varchar(50));
CREATE TABLE
```

insert into tabla_a (mensaje) values('Hola mundo!');

```
test=# insert into tabla_a (mensaje) values('Hola Mundo!');
INSERT 0 1
```

select * from tabla_a;

```
test=# select * from tabla_a;
 mensaje
-----
 Hola Mundo!
(1 row)
```

\q

```
test=# \q
root@507f17de9bf2:/#
```

exit

Conectarse a la base utilizando alguna IDE (Dbeaver - <https://dbeaver.io/>, Azure DataStudio - <https://azure.microsoft.com/es-es/products/data-studio>, etc). Interactuar con los objetos creados.

Explicar que se logro con el comando docker run y docker exec ejecutados en este ejercicio.