Contenedores Docker. Conectando aplicaciones

1. Vamos a emplear el parametro –link de docker para enlazar distintos contenedores. En este caso enlazaremos phpmyadmin con mysql. Para ello:

   a) Descargamos con docker pull las imágenes de phpmyadmin y de mysql
   b) Creamos el contenedor de mysql con:

   ```
   docker run --name bbdd -e MYSQL_ROOT_PASSWORD=naranco -d mysql
   ```

   Le estamos asignando el nombre “bbdd” al nuevo contenedor y estamos estableciendo un valor de entorno de para la contraseña de root (“naranco” en este caso)

   c) Creamos ahora el contenedor para phpmyadmin con:

   ```
   docker run --name pma -d --link bbdd:db -p 8080:80 phpmyadmin
   ```

   Le estamos poniendo el nombre “pma” al nuevo contenedor, mapeando el puerto 80 del contenedor con el puerto 8080 de la maquina anfitrión y con la opción –link estamos conectándolo con el contenedor “bbdd” y con el valor de alias “db”

   d) Comprobamos con docker ps que los dos contenedores estan funcionando y finalmente accedemos a la interfaz de phpmyadmin desde un navegador cualquiera con http://localhost:8080. Nos loguearemos con el usuario root y la password asignada (naranco)

2. Vamos a descargar las imágenes de mariadb y wordpress y lanzar los contenedores, pero en este caso vamos a crear previamente una red particular (dentro de la red brigde) para conectar las dos aplicaciones. Para ello:

   a) Creamos la red privada con:

   ```
   docker network create red_wp
   ```

   Comprobaremos con `docker inspect red_wp` la nueva dirección de red para los contenedores de esta red

   b) Creamos ahora el contenedor de mariadb con:

```
docker run -d --name servidor_mysql \
              --network red_wp \
              -v /opt/mysql_wp:/var/lib/mysql \
              -e MYSQL_DATABASE=bd_wp \
              -e MYSQL_USER=user_wp \
              -e MYSQL_PASSWORD=asdasd \
              -e MYSQL_ROOT_PASSWORD=asdasd \
              mariadb
```

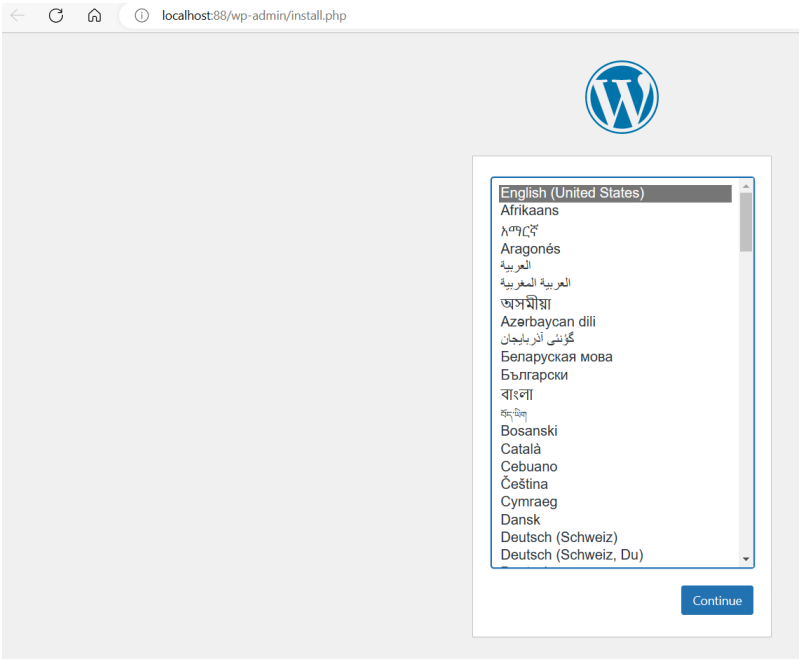
donde –network indica la red privada que empleamos. La opción -v debe ajustarse con una ruta real de nuestro sistema de archivos y los demás parámetros permite crear una base de datos, crear un usuario con contraseña y ajustar la contraseña del root

c) Creamos ahora el contenedor de wordpress con:

```
docker run -d --name servidor_wp \
              --network red_wp \
              -v /opt/wordpress:/var/www/html/wp-content \
              -e WORDPRESS_DB_HOST=servidor_mysql \
              -e WORDPRESS_DB_USER=user_wp \
              -e WORDPRESS_DB_PASSWORD=asdasd \
              -e WORDPRESS_DB_NAME=bd_wp \
              -p 80:80 \
              wordpress
```

De nuevo indicamos la red a usar y con la opción -v (que debemos ajustar) creamos un volumen para los datos de wordpress. Loa demás valores permiten indicar el HOST (contenedor) donde está mariadb y los valores de usuario, contraseña y base de datos a emplear. En este caso se está mapeando el puerto 80 del contenedor con el puerto 80 de la maquina (esto puede ajustarse si es necesario)

d) Acceder desde un navegador de la maquina a la dirección http://localhost y completar la instalación de Wordpress para verificar que todo funciona bien



3. En este caso vamos a usar una aplicación de visualización de temperaturas que emplea dos componenetes un backend con una API REST que permite la consulta de temperaturas en España y un frontend escrito en Python que permite hacer el acceso a municipios. Para ello:

   a) Creamos una red brigde privada para los dos contenedores con

   ```
   docker create network red_temperaturas
   ```

   b) Lanzamos el conedor del backend con:

   ```
   docker run -d --name temperaturas-backend --network red_temperaturas
   avelinopef/temperaturas_back
   ```

   c) Creamos ahora el contendor del frontend con:

   ```
   docker run -d -p 80:3000 --name temperaturas-frontend --network
   red_temperaturas avelinopef/temperaturas_front
   ```

   En este caso se puede mapear el puerto 3000 del servicio con otro puerto diferente del 80 si es necesario

   d) Con un navegador accedemos a http://localhost y verificamos que se puede acceder a la aplicación

4. Vamos ahora a lanzar un contenedor de tomcat. Para ello:

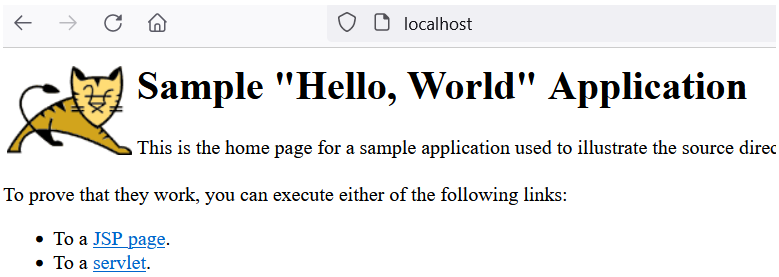
   a) `docker run -d --name aplicacionjava -v c:\users\avelino\tomcat:/usr/local/tomcat/webapps/ tomcat:9.0`

      En este cado debe ajustarse la ruta del volumen para emplear una carpeta de nuestro sistema de archivos

   b) copiaremos en nuestra carpeta del volumen el fichero sample.war que se entrega

   c) A continuación, vamos a emplear el contenedor de nginx para hacer que las peticiones a http://localhost se redirijan a http://IP_aplicacionjava:8080/sample/

   d) Finalmente probaremos desde un navegador que al cargar la URL http://localhost se nos redirige a la aplicación java sample



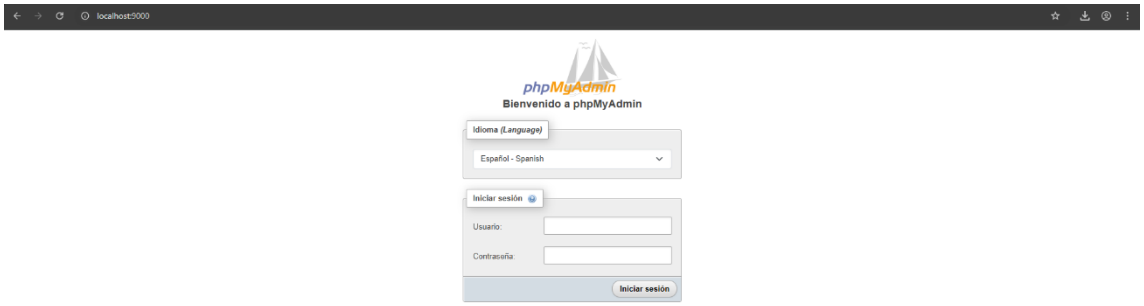
1.- Primero lanzamos nuestro contenedor de mysql.

```
C:\Users\guillermosr26>docker run --name bbdd -e MYSQL_ROOT_PASSWORD=naranco -d mysql
c70b9ad4a1619c07eb07abdc4e279b85eb242c5fb0fa57c9f2c9a7a2c1adfc76
```

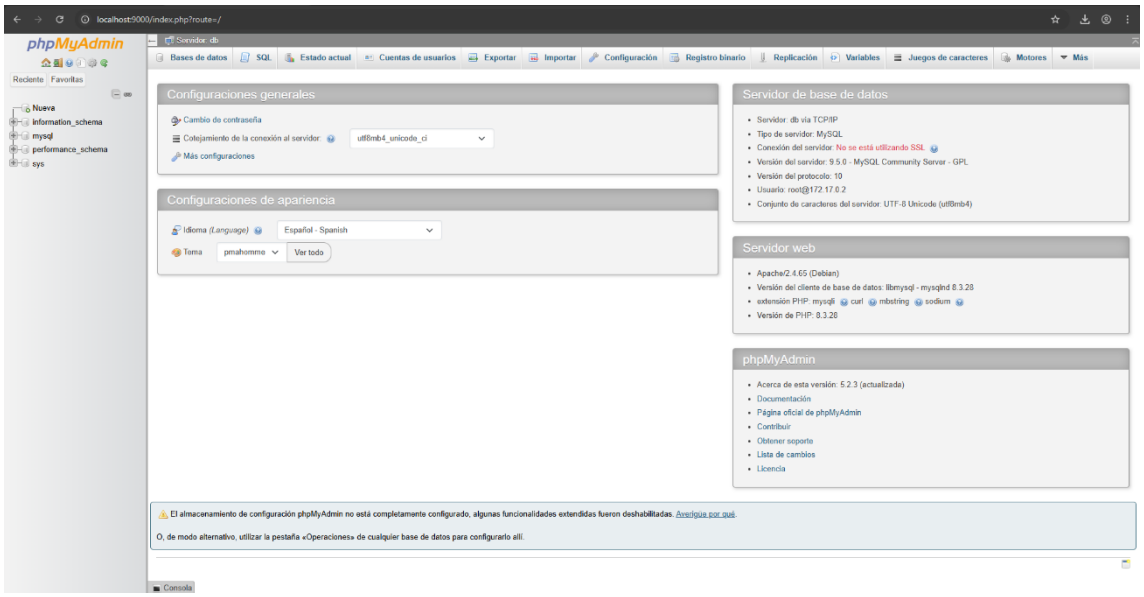
Tras ello lo linkeamos con php my admin, en este caso con el puerto 9000.

```
C:\Users\guillermosr26>docker run --name pma -d --link bbdd:db -p 9000:80 phpmyadmin
```

Ahora iniciamos sesión en phpmyadmin con el usuario y contraseña que hemos puesto.



Y finalmente veremos la interfaz con nuestra base de datos creada a la izquierda.



2.- Ahora creamos nuestra red con el comando docker network create

```
C:\Users\guill>docker network create red_wp
```

Con docker inspect podemos ver la ip de la red.

```
Símbolo del sistema
C:\Users\guill>docker inspect red_wp
[
    {
        "Name": "red_wp",
        "Id": "08579476bfa918b912eefcaf21870ba34070a02f161c3922c16c8c8e170510cb",
        "Created": "2026-01-09T18:00:41.572849557Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
```

Ahora lanzamos el contenedor de maríadb, yo le he puesto las \ para que se vea bien, pero a la hora de ejecutar el comando hay que quitarlas.

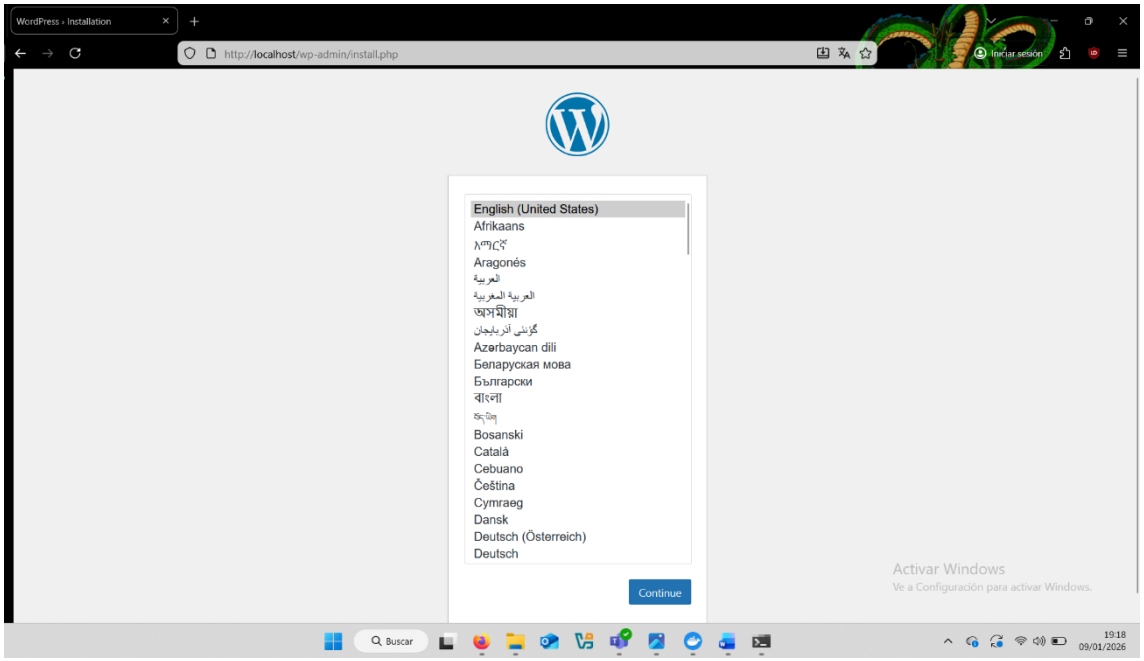
```
C:\Users\guill>docker run -d \ --name servidor_mysql \ --network red_wp \ -v C:\Users\guill\docker\mariadb:/var/lib/mysq
l \ -e MYSQL_DATABASE=bd_wp \ -e MYSQL_USER=user_wp \ -e MYSQL_PASSWORD=asdasd \ -e MYSQL_ROOT_PASSWORD=asdasd \ mariadb
```

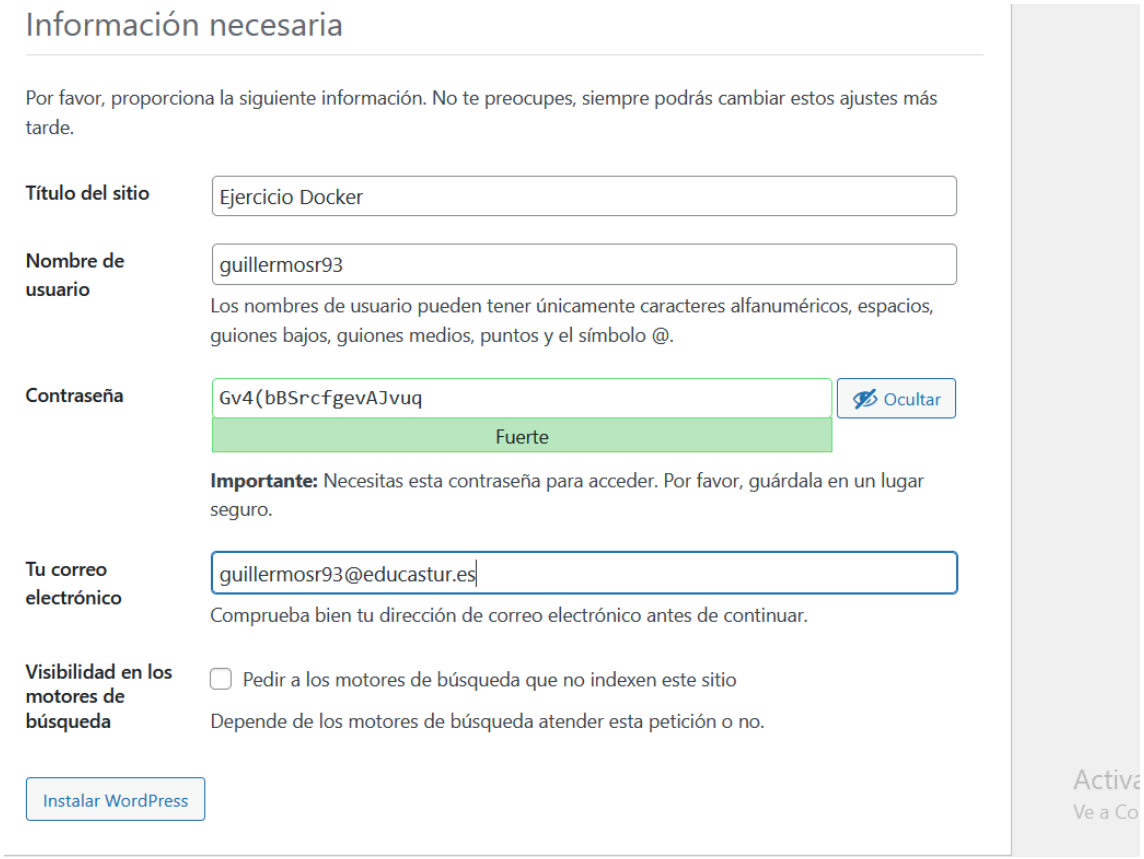
Lanzamos también el contenedor de wordpress.

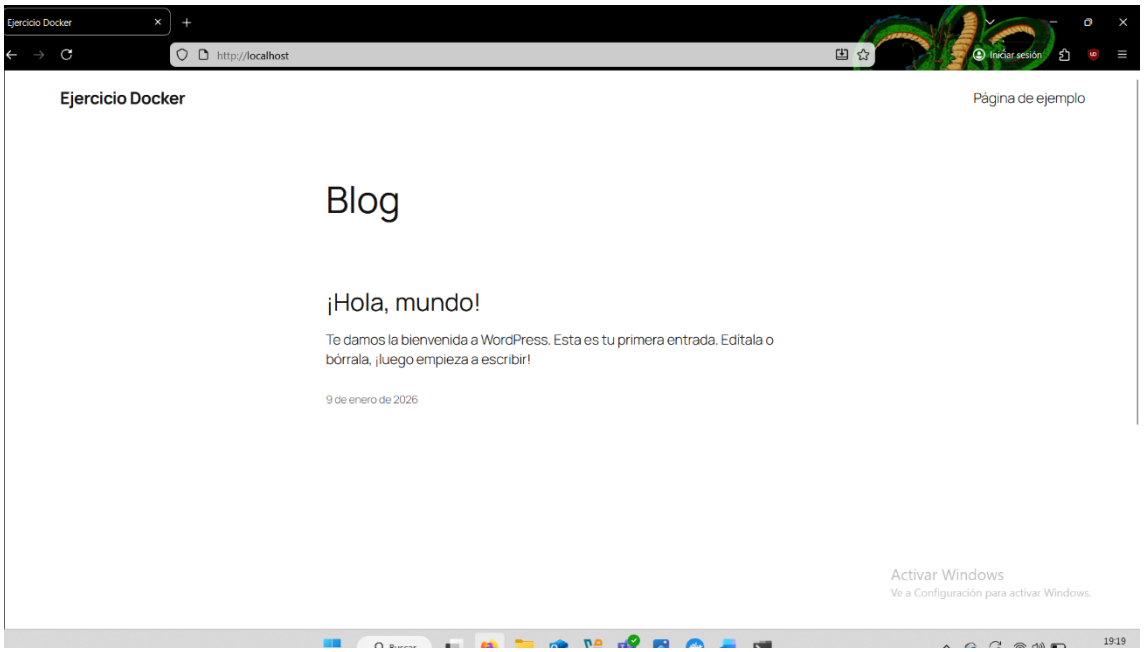
```
C:\Users\guill>docker run -d --name servidor_wp --network red_wp -v C:\Users\guill\docker\wordpress:/var/www/html/wp-cpn
tent -e WORDPRESS_DB_HOST=servidor_mysql -e WORDPRESS_DB_USER=user_wp -e WORDPRESS_DB_PASSWORD=asdasd -e WORDPRESS_DB_NA
ME=bd_wp -p 80:80 wordpress
```

En el volumen, yo he creado una carpeta para ambos, una llamada mariadb y otra wordpress, para introducir ahí todos los archivos.

Ahora si accedemos a localhost ya nos aparecerá wordpress, debemos continuar con la instalación, yo lo he hecho de la siguiente manera.







3.- Ahora crearemos una red llamada red_temperaturas para visualizar una aplicación distinta, asique la creamos.

```
C:\Users\guill>docker network create red_temperaturas
7e1a00d1b71e80956b0f29ccf2d7c0df9bff4428cdf24c53998c76125397f30b

C:\Users\guill>
```

Tras esto lanzamos el contenedor del backend y del frontend.

Backend:

```
C:\Users\guill>docker run -d --name temperaturas-backend --network red_temperaturas avelinopef/temperaturas_back
```

Frontend:

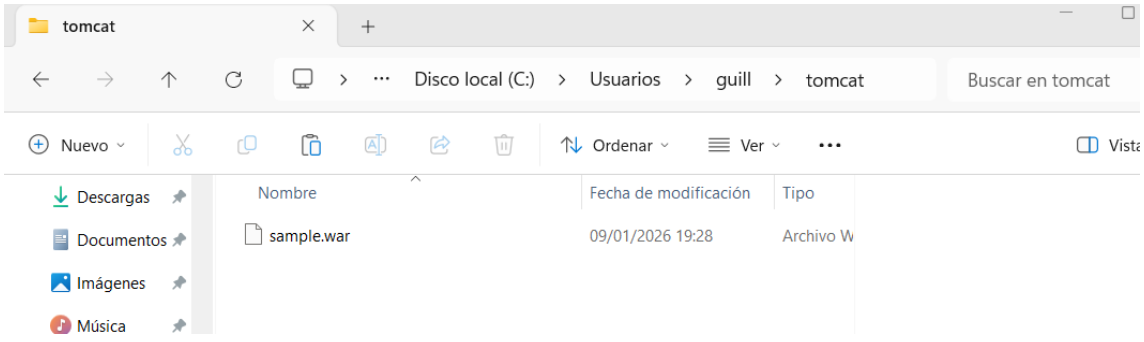
```
C:\Users\guill>docker run -d -p 80:3000 --name temperaturas-frontend --network red_temperaturas avelinopef/temperaturas_
front
Unable to find image 'avelinopef/temperaturas_front:latest' locally
latest: Pulling from avelinopef/temperaturas_front
d724eea63276: Download complete
1b0ea791d129: Downloading [================>                                  ]  62.91MB/192.9MB
18ffb3e704a1: Download complete
0ecb575e629c: Download complete
```

Ahora si accedemos a localhost nos saldrá la aplicación.

4.- Finalmente vamos a lanzar otra aplicación, esta vez de java, para ello lanzaremos un contenedor tomcat.

```
C:\Users\guill>docker run -d --name aplicacionjava -v C:\Users\guill\tomcat:/usr/local/tomcat/webapps/ tomcat:9.0
```

Yo he creado una carpeta en mi sistema llamada tomcat, ahí copiaremos el sample.war que nos da el ejercicio.



Como estoy en casa y no tengo contenedor de nginx, voy a hacerlo de una manera distinta.

Creo una carpeta llamada nginx en mi sistema y le añado el archivo default.conf con lo siguiente:

```
server {
        listen 80;

        location / {
                proxy_pass http://aplicacionjava:8080/sample/;
        }
}
```

Ahora creo la red con docker network create.

```
C:\Users\guill>docker network create red_proxy
4fabe9842128313fbaa38b04d8ec4edf01b9157a3ffaf38e12d96868469df2a9

C:\Users\guill>
```

Y ahora conecto el contenedor de la aplicación con la red proxy que hemos creado.

```
C:\Users\guill>docker network connect red_proxy aplicacionjava

C:\Users\guill>
```

Finalmente lanzamos nginx.

```
C:\Users\guill>docker run -d --name nginx --network red_proxy -v C:\Users\guill\docker\nginx\default.conf:/etc/nginx/con
f.d/default.conf -p 80:80 nginx
```

Y comprobamos que ahora en el localhost nos aparecerá nuestra aplicación de java.