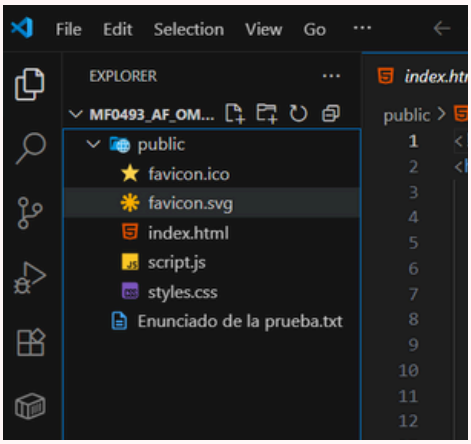
Despliegue de aplicaciones en contenedores Docker

Guía paso a paso

Evaluacion Final - MF0493

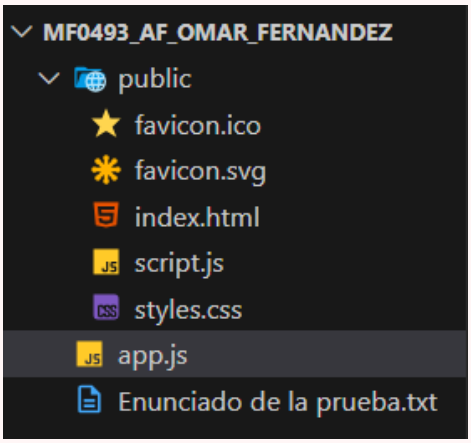
Omar Fernandez Ballester



1. Abrimos nuestro **VSC** y nos encontraremos con la imagen estatica (html) y sus carpetas para el funcionamiento desde el index.html

Todo esto lo vamos a meter dentro de la **carpeta public,** para poder utilizarlo como una aplicación node.

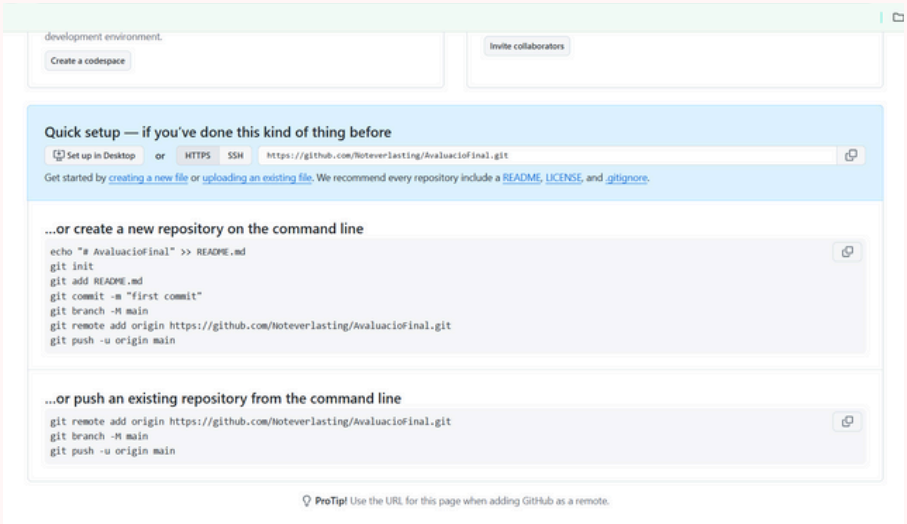
(en este caso ya venia todo dentro de public)



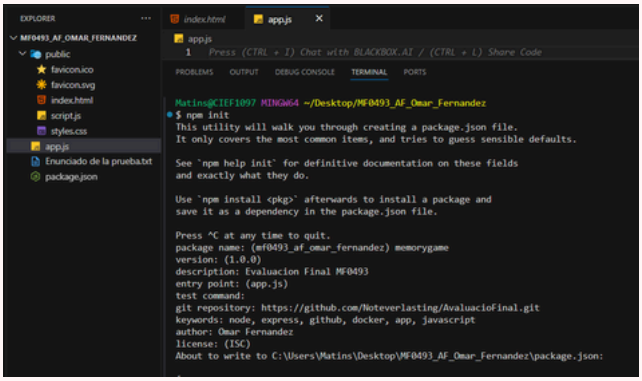
2. Después de meter todos esos ítems en la carpeta public, vamos a crear el archivo **app.js**.

De momento no lo vamos a escribir, ya que pasaremos a inicializar el proyecto de node en el terminal.

3. **Acedemos a gitHub** y vamos a crear un **repositorio** para tenerlo listo y poder indicar la dirección en el proceso de iniciación del proyecto



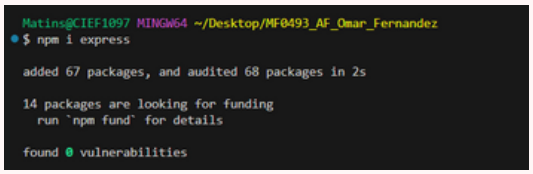
4. Abrimos en nuestro **terminal** (ctrl+ñ) un **GitBash** y pasaremos a escribir los comandos para iniciar npm (node package manager) de este modo:
$ **npm init**



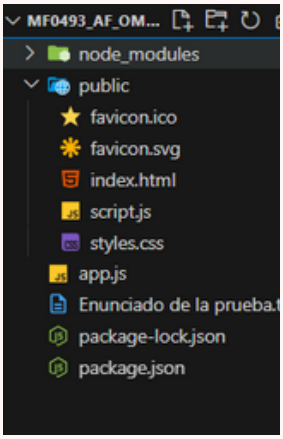
este comando nos irá guiando para completar la instalación, pidiéndonos los datos como:

nombre y versión del proyecto,
fichero de inicio,
fichero de test,
dirección del repositorio de GitHub,
palabras clave,
autor y licencia.

Todo es de libre elección, (por ejemplo el campo licencia no hace falta rellenarlo, se rellena por defecto al apretar enter)



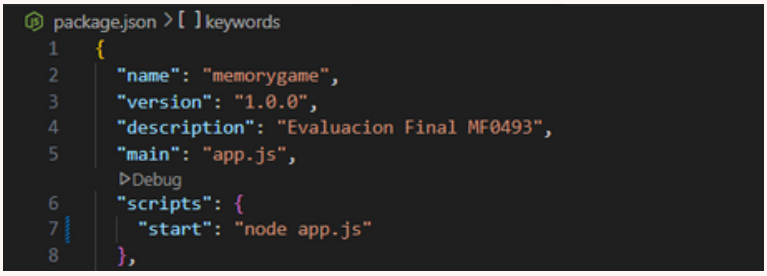
5. Siguiendo en el terminal, instalamos express en el proyecto con el comando:
$ npm i express

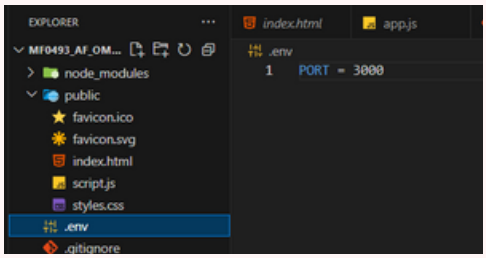


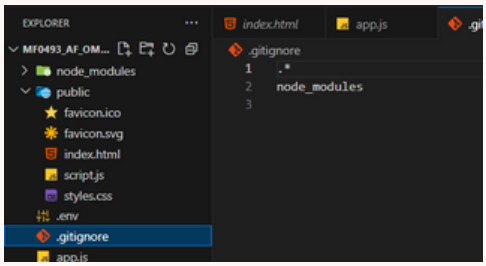
6. Al finalizar la instalación, se nos crearán los archivos **package.json** y **package-lock.json** (de configuracion) así como la carpeta **node_modules**
(donde se guardan los modulos necesarios y la cual NO HAY QUE SUBIR A GITHUB incluyéndola en un .gitignore)

7. Entramos al archivo **package.json** y modificamos en el apartado **“scripts”** el script “test” por el **“start”** indicando después de los dos puntos **“node app.js”**

Esto lo realizamos para que la orden “npm start” ejecute app.js desde node.







7. Creamos los archivos .env y .gitignore:

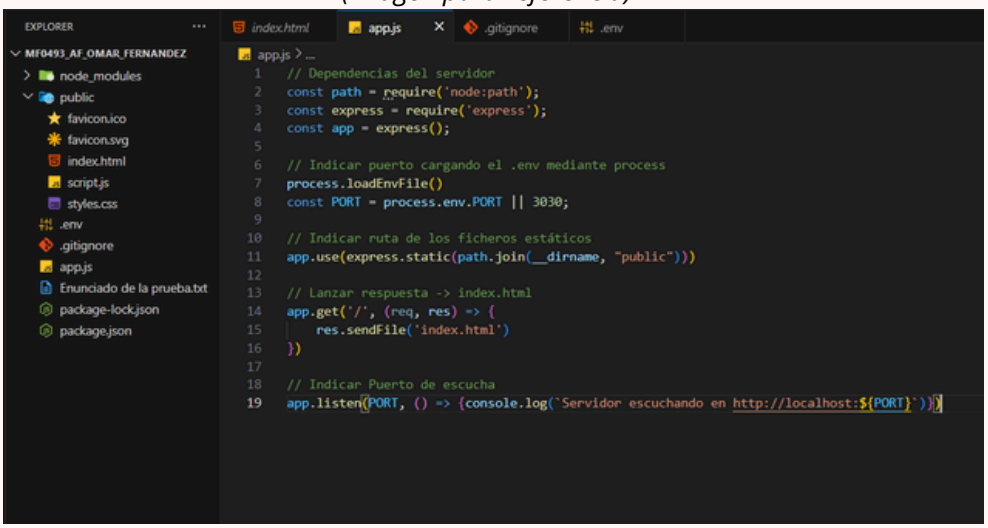
Entramos en el archivo **.env** (el cual sirve para guardar las variables de entorno):
-escribimos la variable de entorno PORT

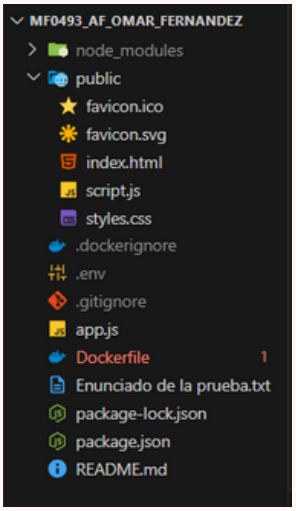
Entramos en el archivo **.gitignore** (en el cual se escriben los ficheros o carpetas que no queremos que sean subidos a github) e indicamos los archivos a ignorar:

- los archivos que empiecen por punto "."*
- la carpeta "node_modules"

8. Configuramos el funcionamiento del archivo **app.js** para que la conexión se realice y el navegador responda, como ya hemos realizado en otros ejercicios de node.js

(imagen para referencia)





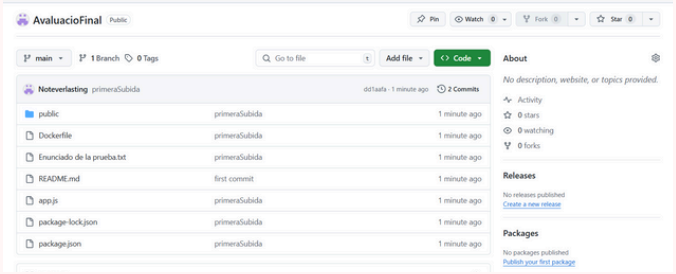
9. Creamos los archivos:

- "**Dockerfile**" (donde indicaremos la configuración de la imagen para poder crearla y que funcione correctamente)

- "**.dockerignore**" (donde indicaremos archivos que no queremos subir a dockerhub).

10. Para no acumular trabajo, es un buen momento de hacer la **primera subida** (commit+push) **a github**, y dejar nuestra primera versión en el repositorio creado. Para ello podemos seguir las instrucciones que nos da el propio git primero:

```
echo "# AvaluacioFinal" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Noteverlasting/AvaluacioFinal.git
git push -u origin main
```



Y seguidamente usamos los comandos para añadir, hacer commit y subir:

$ **git add .**

$ **git commit -m "mensaje"**

$ **git push**

Ya tenemos subida nuestra primera commit al repositorio de github

11. **Configuramos el archivo Dockerfile** del siguiente modo:

```
Dockerfile > ...
1  # Indicamos la imagen de node que se va a utilizar, en este caso alpine3.22 (mas ligera)
2  FROM node:24-alpine3.22
3
4  # Establecemos el directorio de trabajo dentro del contenedor
5  # Todo lo que se copie o ejecute después será relativo a esta ruta
6  WORKDIR /app
7
8  # Copiamos todos los archivos del proyecto local al directorio de trabajo del contenedor
9  COPY . .
10
11 # Instalamos las dependencias del proyecto con npm, limpiamos el cache y eliminamos carpetas temporales,
12 # para reducir el tamaño de la imagen final y que sea lo más ligera posible
13 RUN npm install \
14     && npm cache clean --force \
15     && rm -rf /tmp/ /root/.npm/_cacache
16
17 # Definimos una variable de entorno para el puerto que usará la aplicación
18 ENV PORT=3000
19
20 # Exponemos el puerto definido anteriormente para indicarle que puerto interno usa la aplicacion
21 EXPOSE $PORT
22
23 # Comando por defecto al iniciar el contenedor
24 CMD ["npm", "start"]
```

-Especificación de versiones:

FROM node:24--alpine3.22 (*versión 24 de node*)-(*versión alpine ligera*)

- Indicar carpeta desde la que funciona la imagen dentro del contenedor. ("app")

WORKDIR /app

-Copiar los archivos package.json y package-lock.json (si existe) dentro del contenedor.

COPY package*.json .

-Instalar dependencias usando npm install, y limpiar caché para pese menos la imagen.

RUN npm install \ && npm cache clean --force \ && rm -rf /tmp/ /root/.npm/_cacache

-Copiar todo el resto del proyecto al contenedor (código fuente, carpetas, etc.).

COPY . .

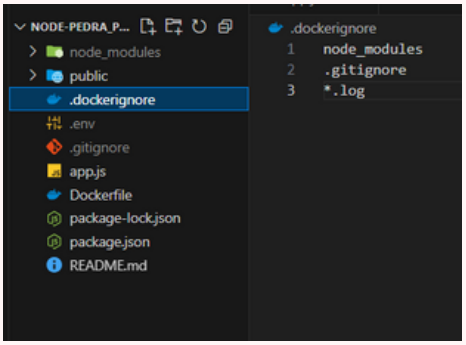
- Indicar a docker que esta app usará el puerto 3000.

ENV PORT=3000 *(variable para definir PORT)*

EXPOSE PORT

- Definir el comando que se ejecutará cuando arranque el contenedor:

CMD ["npm", "start"]

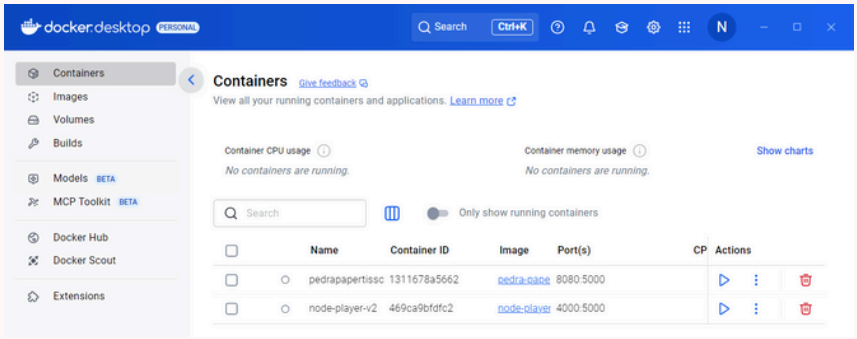


12. En el archivo .dockerignore indicamos que no se suban los siguientes tipos de archivo:

-la carpeta "node_modules"
-el archivo ".gitignore"
-los archivos ".log" (contienen informes)*

13. IMPORTANTE

Abrimos y nos logueamos en **docker desktop** ya que no se **inicia el servicio** hasta entonces

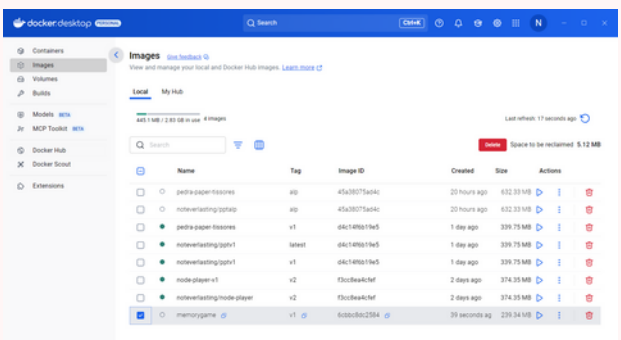


14. Abrimos terminal e introducimos el siguiente comando para construir la imagen:

$ **docker build -t nombre-imagen:tag directorio-origen**
En mi caso: "**docker build -t memorygame:v1 .**"

notas: el ":tag" es el control de versiones y el "." indica que se usen los archivos de esta carpeta actual

Pantalla de confirmacion de que el build fué bien



Con este comando ya tenemos creada la imagen "memorygame" con el tag "v1" para poder usarla

15. Volvemos a terminal, donde vamos a introducir este comando para crear el contenedor con lo necesario para que nuesta aplicación pueda funcionar al subirla:

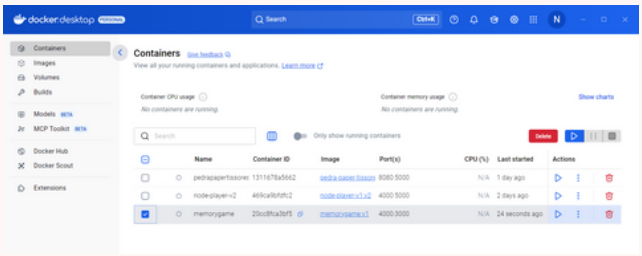
$ **docker run -it -p puerto-host:puerto-container --name nombre-container nombre-imagen:tag**

En mi caso:

“**docker run -it -p 4000:3000 --name memorygame memorygame:v1**”

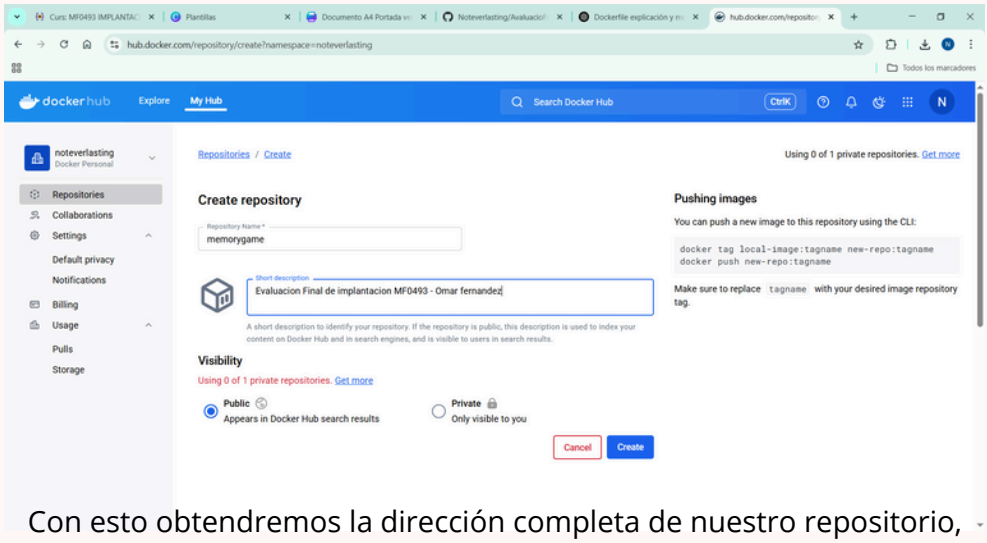
-it : Permite interactuar con el contenedor a través de la terminal (modo interactivo).
-p host:container : Asocia un puerto del host con un puerto interno del contenedor.

```
Matins@CIEF1097 MINGW64 ~/Desktop/MF0493_AF_Omar_Fernandez (main)
$ docker run -it -p 4000:3000 --name memorygame memorygame:v1

> memorygame@1.0.0 start
> node app.js

Servidor escuchando en http://localhost:3000
```



Con este comando ya tenemos creado el contenedor “memorygame” que contiene la imagen “memorygame:v1"

16. Ahora vamos a **crear un repositorio nuevo en dockerhub**.
Accedemos con nuestro usuario y vamos a: **[Create a repository]**
Una vez allí indicamos *Nombre de repositorio*, *descripción* y *visibilidad*.
Clicamos **CREATE**



Con esto obtendremos la dirección completa de nuestro repositorio, que se compone de usuario/nombre-repositorio
(en mi caso **noteverlasting/memorygame**)

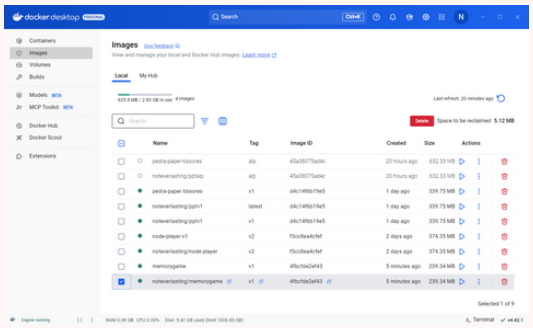
17. Abrimos **terminal**, ya que ahora vamos a es reetiquetar la imagen para subirla a Docker Hub y que esté referenciada correctamente:

$ **docker tag nombreimagen:tag usuario/nombre-repositorio:tag**

En mi caso:

“**docker tag memorygame:v1 noteverlasting/memorygame:v1**”

```
Matins@CIEF1097 MINGW64 ~/Desktop/MF0493_AF_Omar_Fernandez (main)
$ docker tag memorygame:v1 noteverlasting/memorygame:v1
```



Con este comando ya tenemos la imagen renombrada con nuestra ruta completa de dockerhub

18. El siguiente comando que introducimos en la terminal es nuestro login, en principio con este comando ya se hace solo:

$ **docker login**

(si no es así, deberiamos usar: $ docker login -u “nombre-usuario” -p “password”)

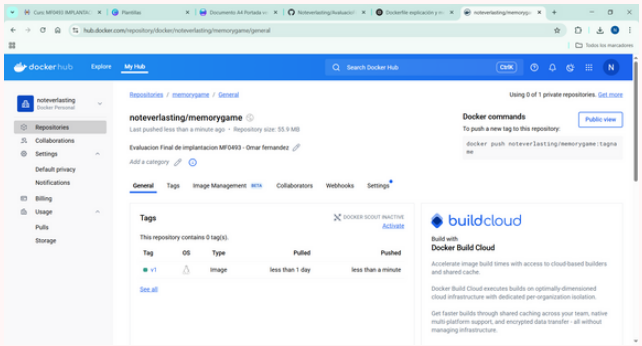
Y seguidamente hacemos el push a dockerhub con este comando:

$ **docker push usuario/nombre-repositorio:tag**

En mi caso:

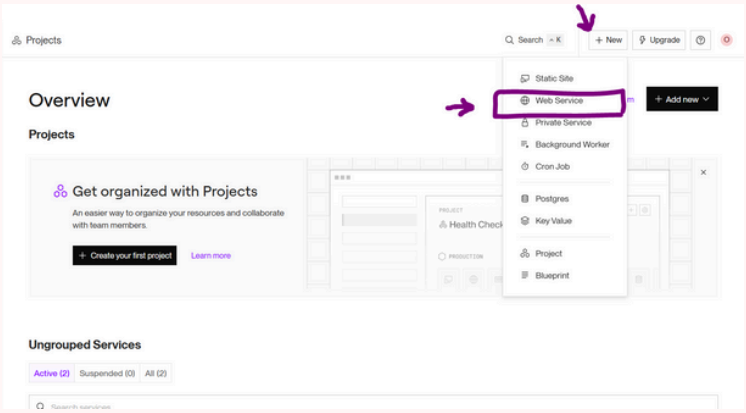
“**docker push noteverlasting/memorygame:v1**”

```
Matins@CIEF1097 MINGW64 ~/Desktop/MF0493_AF_Omar_Fernandez (main)
$ docker login
Authenticating with existing credentials... [Username: noteverlasting]

i Info → To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded

Matins@CIEF1097 MINGW64 ~/Desktop/MF0493_AF_Omar_Fernandez (main)
$ docker push noteverlasting/memorygame:v1
The push refers to repository [docker.io/noteverlasting/memorygame]
fe07684b16b8: Mounted from noteverlasting/pptalp
1bb848e16bc6: Pushed
b0252ce8e5cb: Mounted from noteverlasting/pptalp
6a1e58d27f9f: Mounted from noteverlasting/pptalp
d33b9420a165: Mounted from noteverlasting/pptalp
54bcf4cfad6f: Pushed
4387918b708a: Mounted from noteverlasting/pptalp
d83282c7462a: Pushed
```



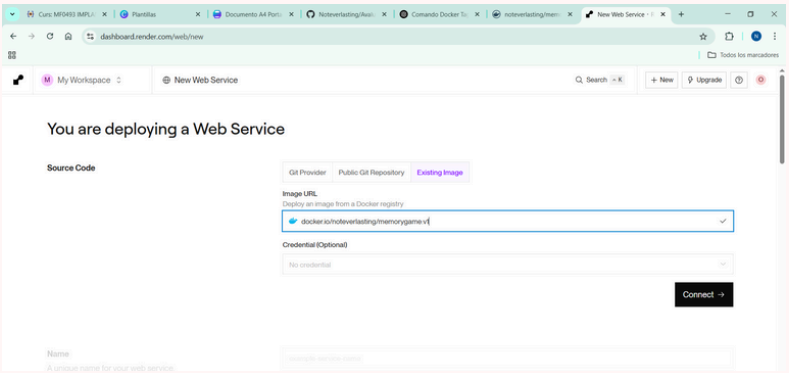
Esto subirá la imagen con el tag que le hemos indicado *(si no se le indica, por defecto será “latest”)* **a nuestro repositorio de dockerhub.**

19. **Accedemos a Render** y **creamos un nuevo Web Service**

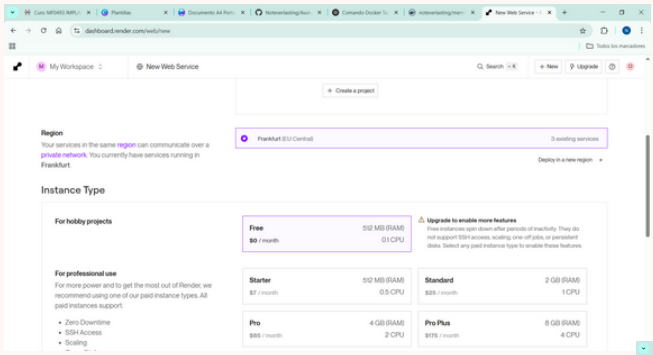


20. Le indicamos que vamos a crearlo a través de una **Imagen existente.**

En el campo Image URL escribiremos:
docker.io/usuario/nombre-repositorio:tag
En mi caso:
“**docker.io/noteverlasting/memorygame:v1**”

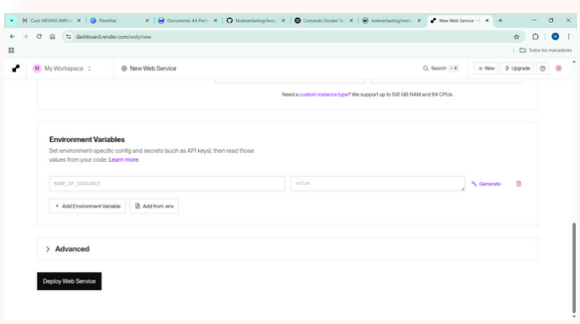


Si todo va bien, en el campo saldrá un **tick de color verde**, entonces pulsamos **Connect**.



Seleccionamos Frankfurt
y la version gratis,

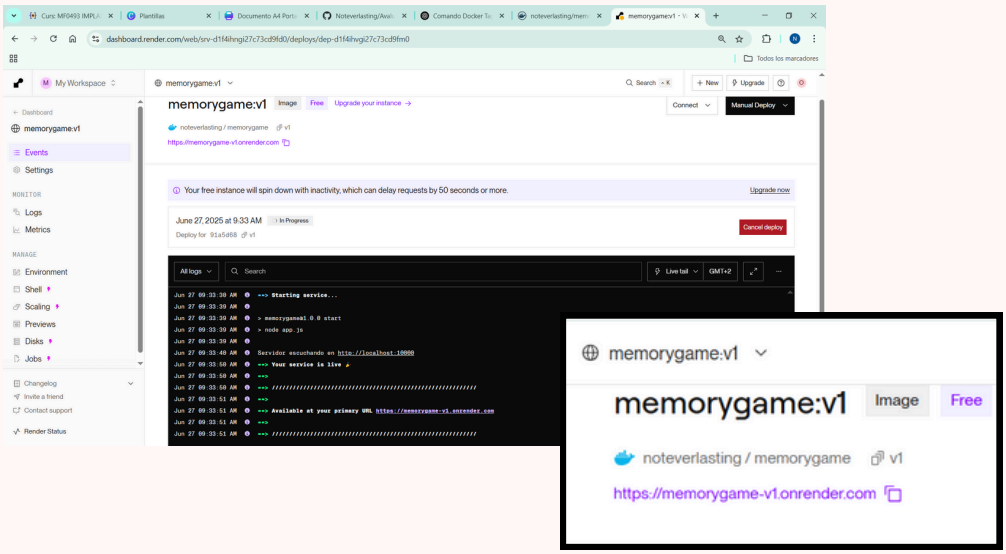
clicamos **Deploy web Service**



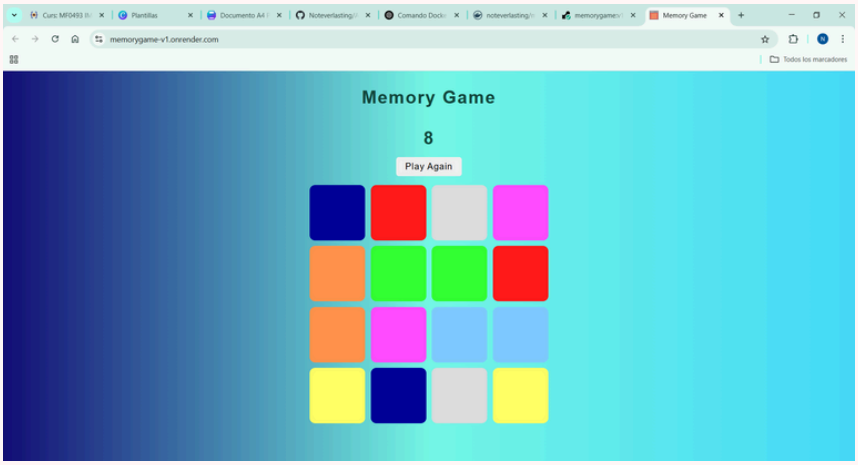
21. Se abrirá una nueva ventana donde se nos mostrará el progreso de la ejecución y creación del web service, y si todo funciona correctamente aparecerá el mensaje:

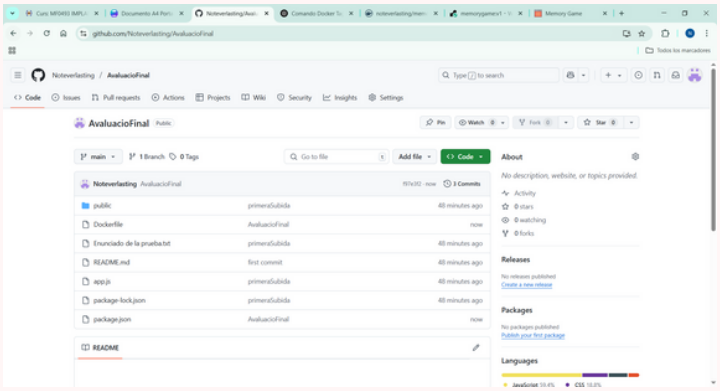
Your server is Live!

y justo debajo de la direccion de nuestro service se nos facilitará una dirección donde podremos acceder a probar nuestra aplicación ya subida y funcional.



https://memorygame-v1.onrender.com/





Para finalizar, **actualizamos github** desde terminal con los comandos correspondientes:

git add .
git commit -m “mensaje”
git push

LINKS DE ACCESO A LOS REPOSITORIOS

GITHUB:

https://github.com/Noteverlasting/AvaluacioFinal

DOCKERHUB:

https://hub.docker.com/repository/docker/noteverlasting/memorygame/general

RENDER:

https://memorygame-v1.onrender.com/