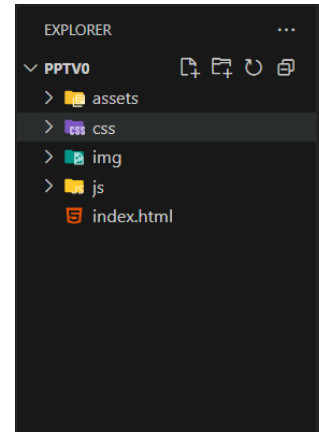


CONVERSION A NODE Y SUBIDA A DOCKER DE UNA APLICACIÓN

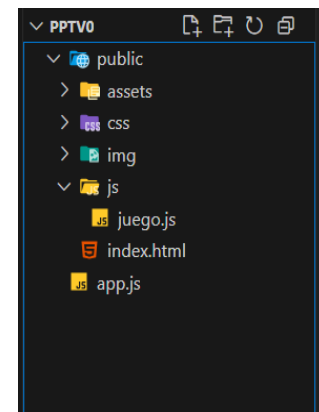
1. Abrimos nuestro VSC y nos encontraremos con la imagen estatica (html) y sus carpetas para el funcionamiento desde el index.html

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



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.

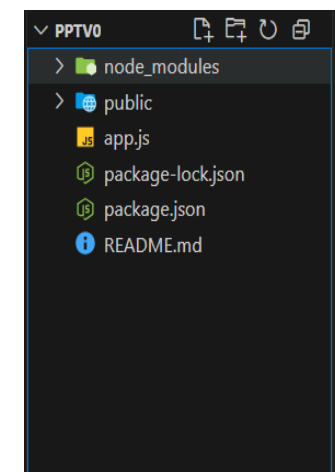
Uno de los pasos que también vamos a dejar preparado es crear un repositorio de GitHub para indicarlo en el proceso de iniciación del proyecto



3. 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 – que hemos creado antes-, palabras clave, autor y licencia. Todo es de libre elección, y por ejemplo el campo licencia no hace falta ponerlo, se rellena por defecto al apretar enter)



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

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

5. Configuramos el funcionamiento del archivo app.js como ya hemos realizado en otros ejercicios de node.js

```
appjs > ...
1 // Dependencias del servidor
2 const path = require('node:path');
3 const express = require('express');
4 const app = express();
5
6 // Middleware
7
8 process.loadEnvFile()
9 const PORT = process.env.PORT || 3000;
10
11 // Indicar ruta de los archivos estáticos
12 app.use(express.static(path.join(__dirname, "public")))
13
14 //Lanzar
15 app.get('/', (req, res) => {
16   res.sendFile('index.html')
17 })
18
19 // Indicar Puerto de escucha
20 app.listen(PORT, () => {console.log(`Servidor escuchando en http://localhost:${PORT}`)})
```

6. Creamos el archivo ".gitignore" donde incluiremos:
 - los archivos que empiecen por punto "."(como .env .dockerignore...)
 - la carpeta "node_modules"
7. Creamos los archivos:
 - "Dockerfile" (donde indicaremos la configuración de la aplicación para subirla)
 - ".dockerignore" (donde indicamos archivos que no queremos subir a dockerhub).
8. Configuramos el archivo Dockerfile del siguiente modo:

-Especificación de versiones:

FROM **node:24-slim** (versión 24 de node)-(versión 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) al contenedor.

COPY **package*.json .**

-Instalar dependencias usando npm install, y limpiar caché para peso 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 **..**

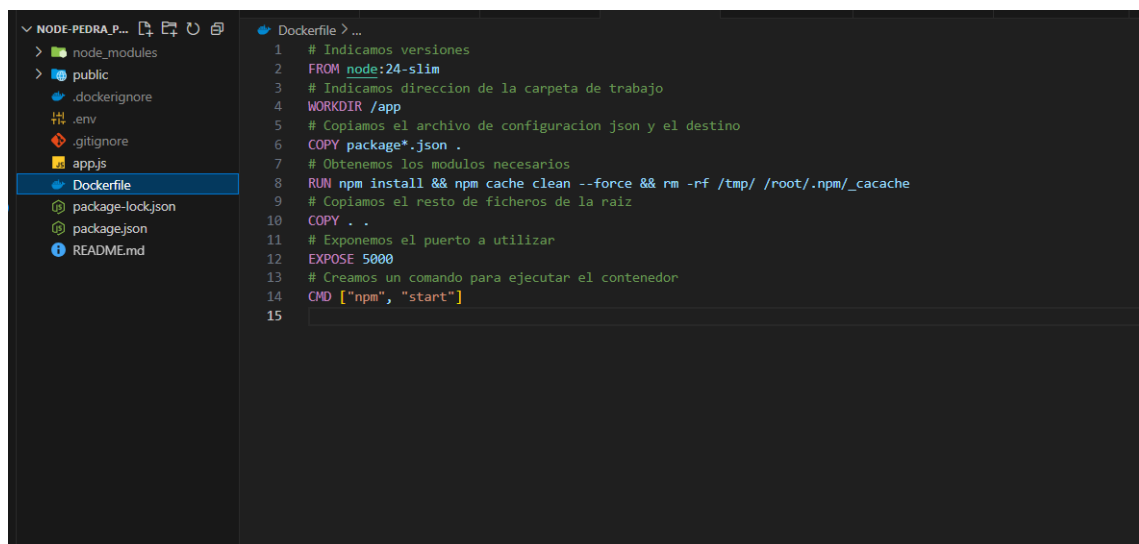
- Indicar a docker que esta app usará el puerto 5000.

EXPOSE **5000**

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

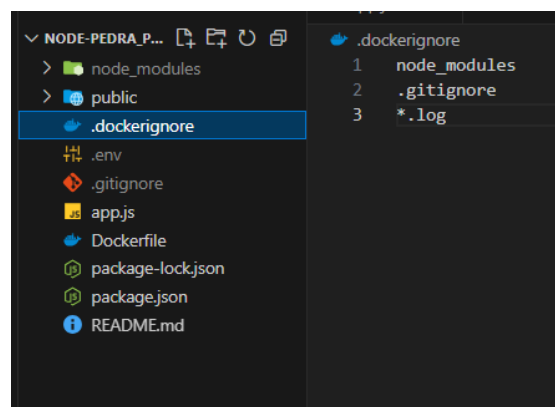
CMD **["npm", "start"]**

Este es un ejemplo de como queda el archivo Dockerfile



```
1 # Indicamos versiones
2 FROM node:24-slim
3 # Indicamos direccion de la carpeta de trabajo
4 WORKDIR /app
5 # Copiamos el archivo de configuracion json y el destino
6 COPY package*.json .
7 # Obtenemos los modulos necesarios
8 RUN npm install && npm cache clean --force && rm -rf /tmp/ /root/.npm/_cacache
9 # Copiamos el resto de ficheros de la raiz
10 COPY . .
11 # Exponemos el puerto a utilizar
12 EXPOSE 5000
13 # Creamos un comando para ejecutar el contenedor
14 CMD ["npm", "start"]
15
```

9. Y en el archivo .dockerignore indicamos que no se suban los siguientes tipos de archivo:



```
1 node_modules
2 .gitignore
3 *.log
```

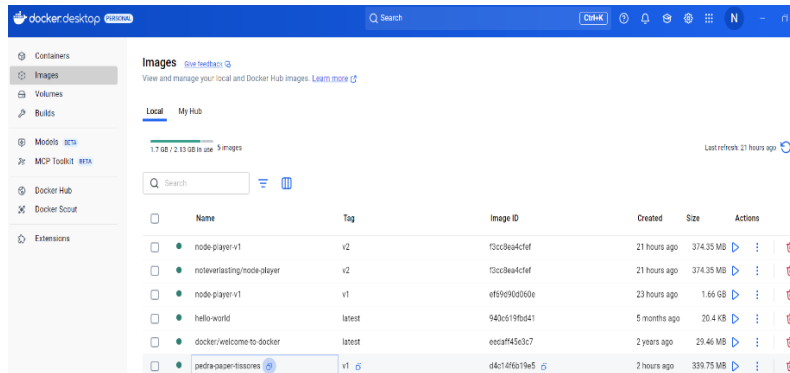
10. Ahora vamos de nuevo al terminal e introducimos el siguiente comando para construir la imagen de nuestro proyecto.

\$ docker build -t *nombre-imagen:tag* *directorio-origen*

En mi caso: **“docker build -t *pedra-paper-tissores:v1* .”**

```
Matias@TEF1807 MINGW64 ~/Desktop/node-pedra_paper_tissores (main)
$ docker build -t pedra-paper-tissores:v1 .
[+] Building 10.5s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 512B
=> [internal] load metadata for docker.io/library/node:24-slim
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load dockerignore
=> transferring context: 71B
=> [1/5] FROM docker.io/library/node:24-slim@sha256:d773e49e39dafc0e425b9fac1d74705886b9aebac960786ab146c5d4fa0632d
=> resolve docker.io/library/node:24-slim@sha256:d773e49e39dafc0e425b9fac1d74705886b9aebac960786ab146c5d4fa0632d
=> sha256:435aee359ac7c3c12c5273692e561ba1cb0f218ee8558455804f9f6e9f265a9 447B / 447B
=> sha256:18f50900c3495786d1ef3c196a268f23f43706b350005e33db3a839da26e1d04 1.71MB / 1.71MB
=> sha256:dbf4ae3706bc24278d0b932068b33e2fbedd4b6362c7506ca79be8479616ed 4.9s / 51.49MB
=> sha256:a5aee185c1369093d8268fec68a440bc7eda5ebac5f75617818500b840echea 3.31kB / 3.31kB
=> extracting sha256:a5aee185c1369093d8268fec68a440bc7eda5ebac5f75617818500b840echea
=> extracting sha256:dbf4ae3706bc24278d0b932068b33e2fbedd4b6362c7506ca79be8479616ed
=> extracting sha256:18f50900c3495786d1ef3c196a268f23f43706b350005e33db3a839da26e1d04
=> extracting sha256:435aee359ac7c3c12c5273692e561ba1cb0f218ee8558455804f9f6e9f265a9
=> [internal] load build context
=> transferring context: 1.54MB
=> [2/5] WORKDIR /app
=> [3/5] COPY package*.json .
=> [4/5] RUN npm install && npm cache clean --force && rm -rf /tmp/.npm/_cacache
=> [5/5] COPY . .
=> exporting to image
=> exporting layers
=> exporting manifest sha256:b6ddf3581639f0ae002c4c955e8afa3b0234d5c3887a264fba59eb4fa
=> exporting config sha256:a6e17f6ba254c9992158fcb42b63dc701c43b9f9a8d16cbda465a569f57
=> exporting attestation manifest sha256:280bde1e36c5c6e2a69a6c51098411320b1b5631fbc801de98be2b07fb340bf1
=> exporting manifest list sha256:d4c14f6b19e53260decf85d7b7564577bcffa01260262faae25c6232bc35768a
=> naming to docker.io/library/pedra-paper-tissores:v1
=> unpacking to docker.io/library/pedra-paper-tissores:v1
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/t3bd1024a00y1fhyq0211b15
```

Este comando lo que ha hecho es crear la imagen para poder usarla.



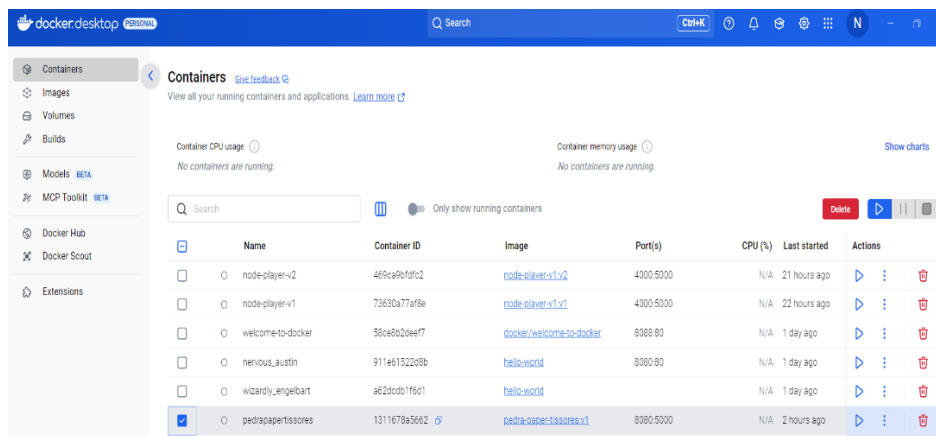
11. De nuevo, volvemos a la terminal, donde vamos a introducir un comando, esta vez para crear el contenedor con lo necesario para que nuestra aplicación pueda funcionar .

\$ docker run -it -p *puerto-ejecucion:puerto-interno* --name *nombre-container* *nombre-imagen:tag*

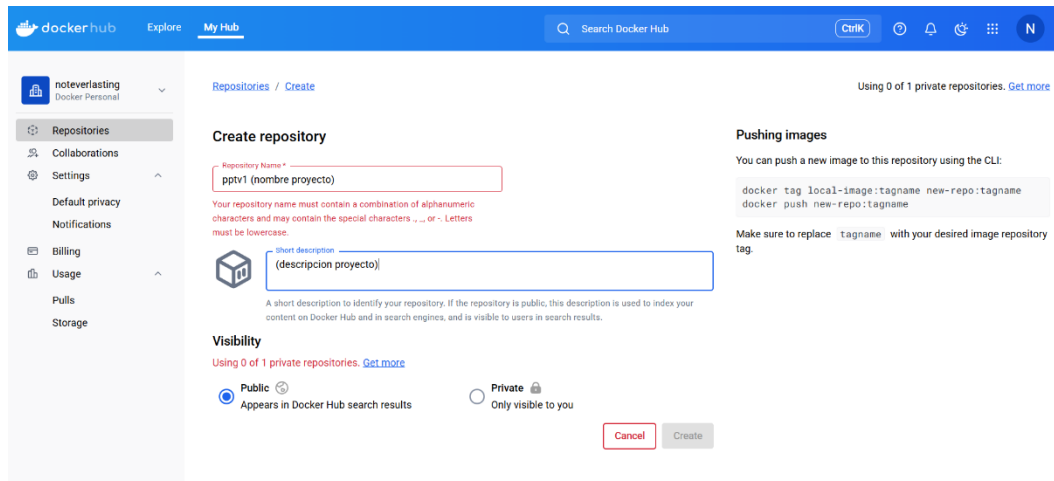
En mi caso:

“docker run -it -p 8080:5000 --name *pedrapapertissores* *pedra-paper-tissores:v1*”

Este comando crea el container:



12. Ahora vamos a crear un repositorio nuevo en dockerhub.
Accedemos y vamos a: **Create a repository**
Una vez allí indicamos Nombre de repositorio, descripción y visibilidad.
Clicamos CREATE



13. Cuando ya tengamos la dirección de nuestro repositorio, que se compone de **usuario/nombre-repositorio** (en mi caso **noteverlasting/pptv1**)

[Repositories](#) / [pptv1](#) / [General](#)

noteverlasting/pptv1

joc de pedra, paper i tissors interactiu funcionant amb node.

[Add a category](#)

Abrimos terminal y escribimos el siguiente comando para reetiquetarla, ya que es necesario etiquetar la imagen con “usuario/nombre-repositorio:tag para subirla a Docker Hub y que esté referenciada correctamente:

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

En mi caso:

“docker tag pedra-paper-tissors:v1 noteverlasting/pptv1:v1”

14. El siguiente comando que introducimos en la terminal es nuestro login:
\$ docker login -u “nombre-usuario” -p “password”

15. Por último en la terminal, realizamos un push a dockerhub para subir la imagen.

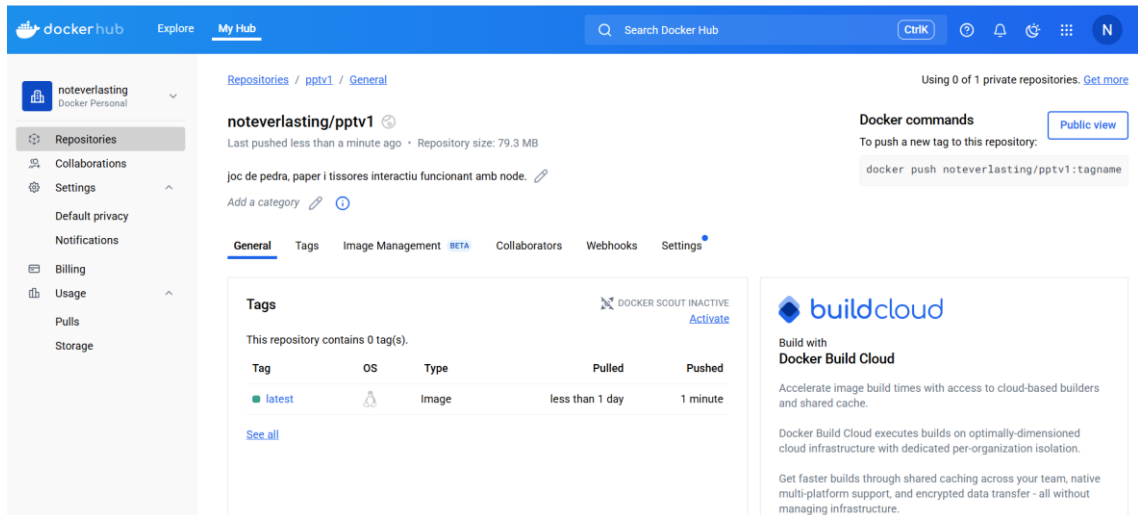
\$ docker push usuario/nombre-repositorio:tag

En mi caso:

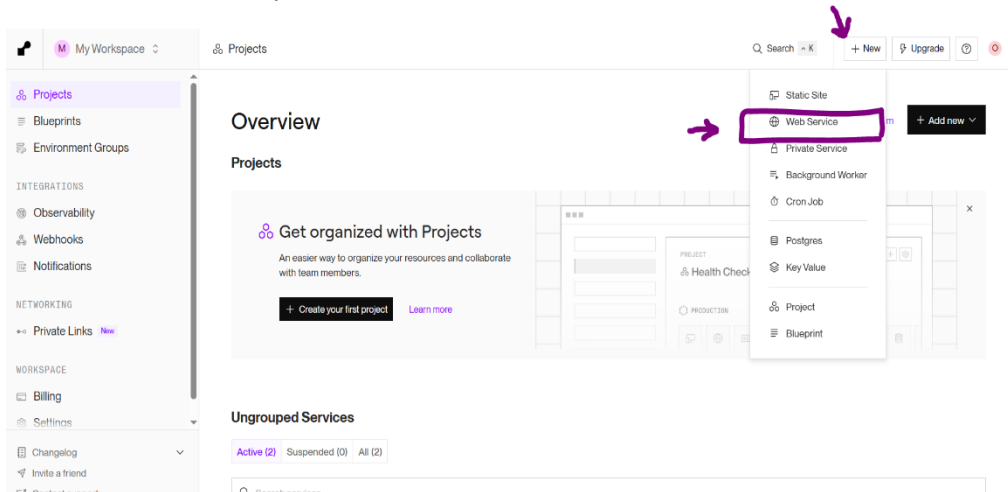
“docker push noteverlasting/pptv1:v1”

Esto hará que nuestro repositorio tenga la imagen subida

(NOTA: si, como yo, te olvidas de poner la tag después del nombre del repositorio, se aplica por defecto la tag “latest”, así que esta será la que se ha de usar en Render)



16. Accedemos a Render y creamos un nuevo Web Service.



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

En el campo Image URL escribiremos:

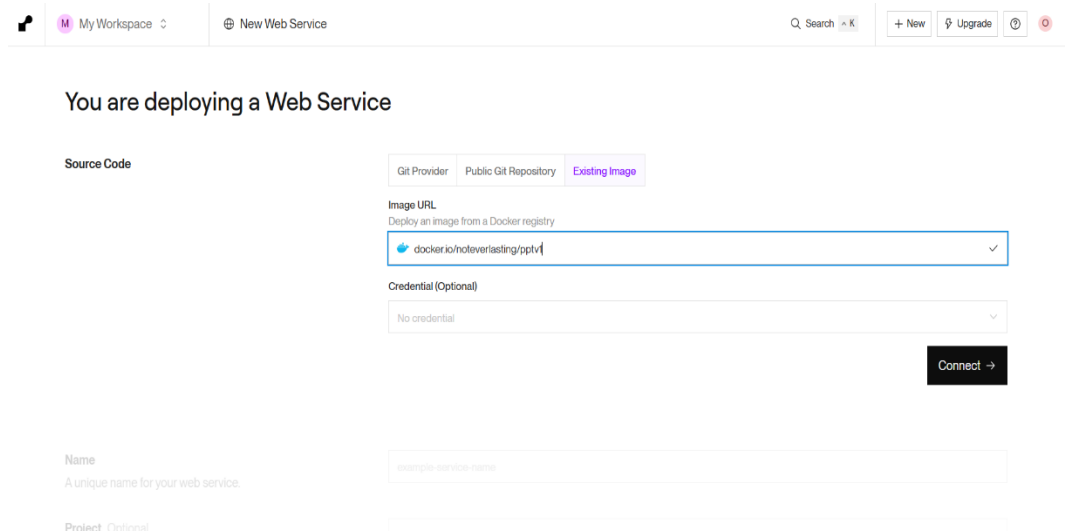
docker.io/usuario/nombre-repositorio:tag

En mi caso:

“docker.io/noteverlasting/pptv1:latest”

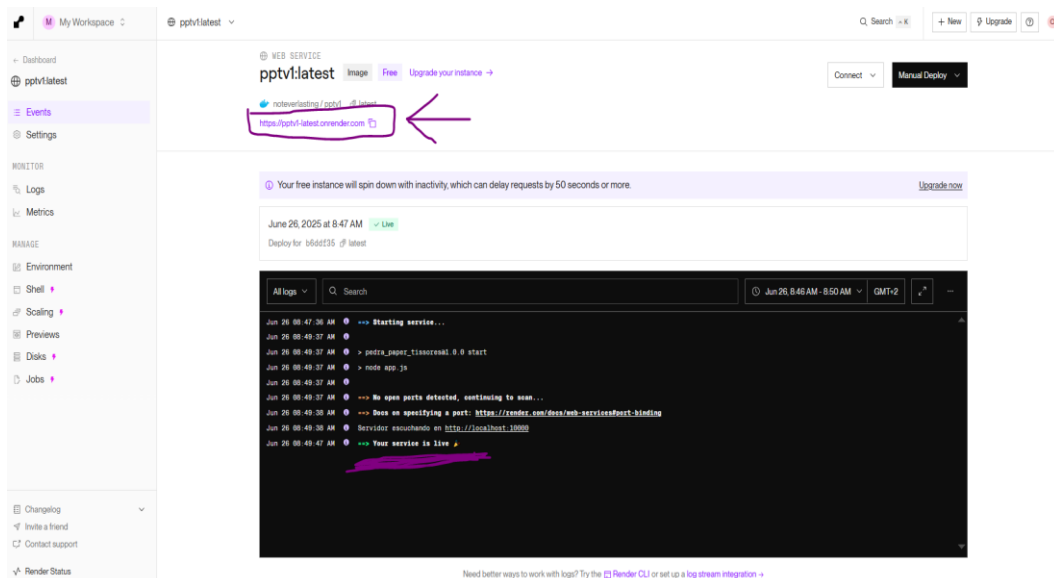
(recordemos que indico “latest” porque se me olvidó indicar el tag en el anterior comando, como se ve, el tag es totalmente personalizable (es la versión), solo tenemos que asegurarnos de definir siempre la misma a la hora de enlazarlos)

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



The screenshot shows the 'New Web Service' form in the Render dashboard. At the top, there's a header with 'My Workspace' and 'New Web Service'. Below this, a message says 'You are deploying a Web Service'. The 'Source Code' section has three tabs: 'Git Provider', 'Public Git Repository', and 'Existing Image'. The 'Existing Image' tab is selected. Under 'Image URL', there's a text input field containing 'docker.io/noteverlasting/pptv1' and a dropdown arrow. Below this is a 'Credential (Optional)' field with 'No credential' selected. A 'Connect' button is at the bottom right. Further down, there are fields for 'Name' (with a placeholder 'example-service-name') and 'Project' (with a placeholder 'Optional').

18. Se abrirá una nueva ventana donde se nos mostrará el progreso de la ejecución y creación del web service, y si todo funciona correctamente nos facilitará una dirección donde podremos acceder a probar nuestra aplicación ya subida y funcional.



The screenshot shows the Render deployment progress page. At the top, there's a header with 'My Workspace' and 'pptv1latest'. Below this, there's a section for 'WEB SERVICE' with 'pptv1latest' and 'Image Free' tabs. A link 'https://pptv1-latest.onrender.com' is highlighted with a red box and a red arrow. Below this, there's a message: 'Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more.' A 'Upgrade now' link is to the right. The main section shows the deployment progress: 'June 26, 2025 at 8:47 AM' with a green 'Live' status. Below this, there's a log stream showing the deployment process: 'Starting service...', 'Deploying b6d6135', 'No open ports detected, continuing to scan...', 'Done on specifying a port: https://render.com/docs/web-servicesPort-binding', 'Server is running on http://localhost:10000', and 'Your service is live'.

En mi caso el enlace es: <https://pptv1-latest.onrender.com/>