TP6 - Construcción de imágenes en Docker

1 - Leer https://docs.docker.com/engine/reference/builder/ y describir las instrucciones

FROM

Establece la base para la construcción de una imagen Docker, y es el punto de partida desde el cual se agrega software y configuración adicional para crear una imagen personalizada.

Especifica la imágen principal a partir de la cual estoy construyendo. Puede que venga detras de una o mas instrucciones **ARG,** que declaran los argumentos usados en las lineas FROM en el dockerfile.

RUN

Esta instrucción ejecuta comandos dentro de una nueva capa (layer) en la parte superior de la imagen actual y luego compromete (commit) los resultados. La imagen resultante comprometida se utilizará para el siguiente paso en el Dockerfile.

> 💡 Cada instrucción RUN genera una nueva capa en la imágen con esos cambios y se genera un commit que representa esa capa.

ADD

Esta instrucción copia nuevos archivos, directorios o archivos URLs remoto desde una ubicación de orígen ***<src>*** y los agrega al sistema de archivos de la imágen Docker en la ruta de destino ***<dest>.***

Cada `<src>` puede contener comodines (wildcards) y la coincidencia se realizará utilizando las reglas de `filepath.Match` de Go.

Por ejemplo, para agregar todos los archivos que comienzan con "hom":

```
ADD hom* /mydir/
```

Por ejemplo, si deseas agregar "test.txt" a un directorio llamado `relativeDir/` en el contenedor, puedes usar una ruta relativa:

```
ADD test.txt relativeDir/
```

`ADD`**,** en comparación con **COPY** tiene algunas funcionalidades adicionales, como la capacidad de copiar archivos desde una URL remota o extraer archivos comprimidos automáticamente

COPY

Esta instrucción se usa para copiar nuevos archivos o directorios desde una ubicación de origen `<src>` y agregarlos al sistema de archivos del contenedor en una ubicación de destino `<dest>`.

Es mejor utilizar `COPY` para mantener la simplicidad y la claridad en el Dockerfile, a menos que especifique que se necesita ***ADD***.

EXPOSE

Informa a Docker que el contenedor escuchará en los puertos de red específicados en tiempo de ejecución. Puedo especificar puerto **TCP, UDP.**

Si no se especifica por defecto es el **TCP.**

Por sí sola no publica el puerto para que sea accesible desde fuera del contenedor.

En realidad, actúa como una especie de documentación que informa a la persona que crea la imagen y a la persona que ejecuta el contenedor sobre qué puertos están destinados a ser publicados.

Para publicar un puerto específico cuando ejecuto el contenedor, puedo usar la opción `-p` seguida del número de puerto cuando ejecuto el comando `docker run`. Por ejemplo:

```
docker run -p 8080:80 mi-imagen
```

CMD

Puede haber un solo una instrucción **CMD** por dockerfile. Si se agregan más, solo la última generará efectos.

> 💡 Proporciona valores predeterminados para un contenedor en ejecución, pueden incluir un ejecutable o no y en ese caso se debe especificar tambien una instrucción ***ENTRYPOINT.***

Si el **CMD** es usado para proporcionar argumentos predeterminados a lainstrucción **ENTRYPOINT,** ambas instrucciones deben estar espeecificadas en formato **JSON**

ENTRYPOINT

Se usa para configurar un contenedor para que se ejecute como un ejecutable. Esto significa que defino qué comando se ejecutará cuando se inicie el contenedor.

Hay 2 formas:

1. Forma **exec:**

   Es la forma preferida y **se utiliza como una lista JSON.** Acá, especifico el comando ejecutable y sus argumentos como una lista de cadenas.

   Por ejemplo:

   ```
   ENTRYPOINT ["executable", "param1", "param2"]
   ```

2. Forma **shell:**

   Se utiliza como una línea de comando en sí misma. Puedo escribir el comando ejecutable y sus argumentos directamente en la instrucción `ENTRYPOINT`.

   Por ejemplo:

   ```
   ENTRYPOINT command param1 param2
   ```

Los argumentos de línea de comando que pase al iniciar un contenedor con `docker run <imagen>` se agregarán después de todos los elementos de la instrucción `ENTRYPOINT` en forma de lista JSON.

Estos argumentos anularán cualquier elemento especificado usando `CMD`. Esto permite pasar argumentos al punto de entrada del contenedor.

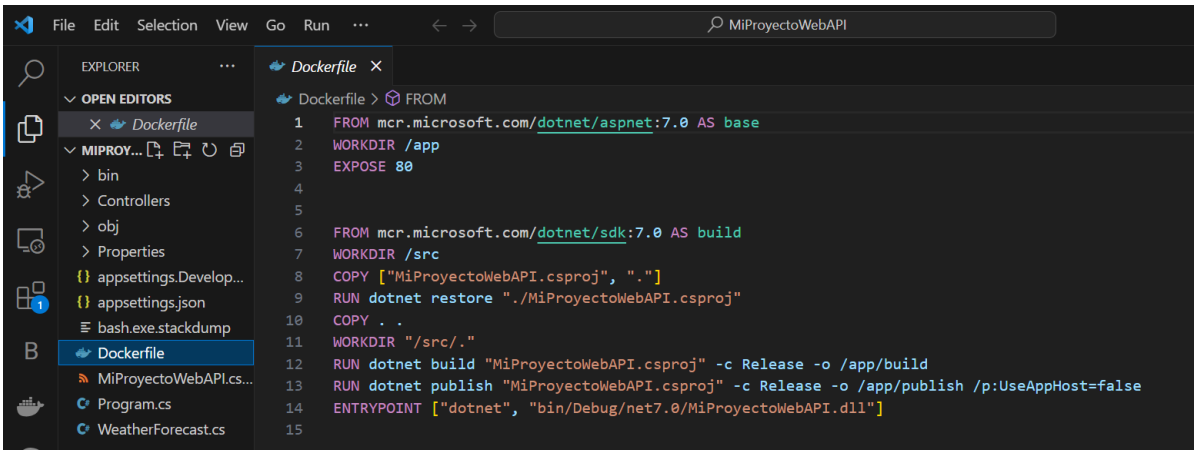
Por ejemplo:

```
docker run <imagen> -d
```

💡 Solo la última instrucción `ENTRYPOINT` en el Dockerfile tendrá efecto.

2- Generar imagen de docker.

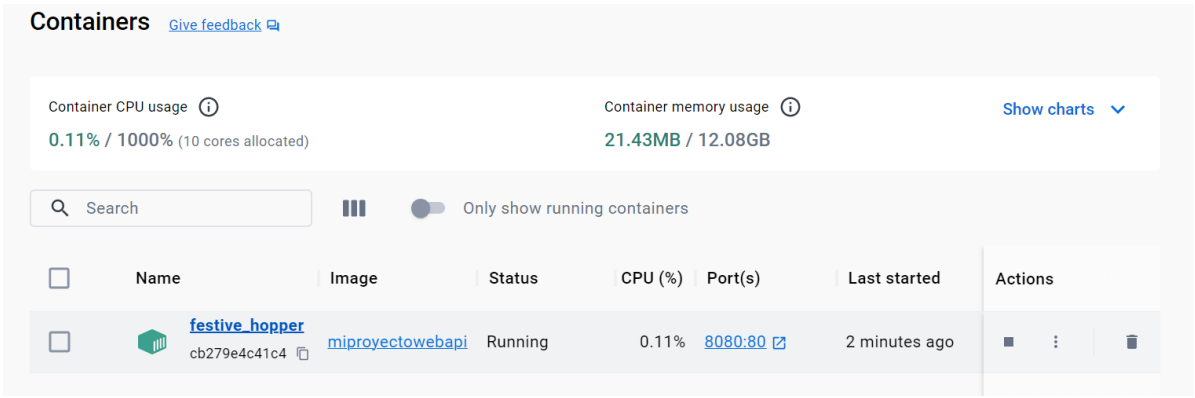
Utilizar el resultado del paso 1 del TP5 y agregar un archivo llamado Dockerfile (en el directorio raiz donde se encuentran todos los archivos y directorios)



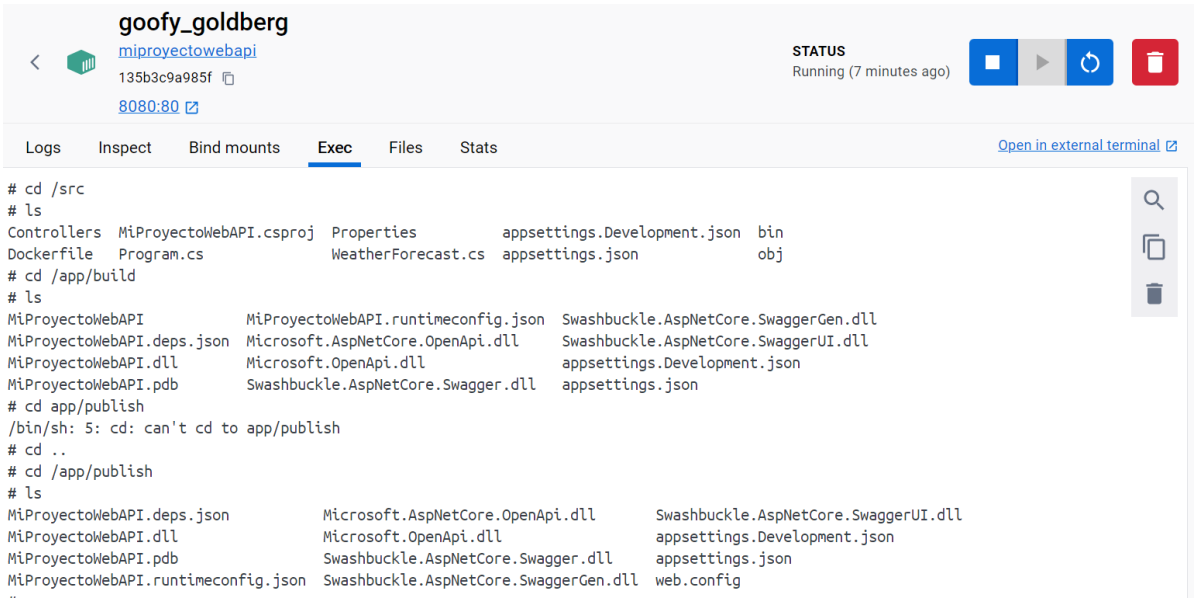
Dockerfile agregado

Generar la imagen de docker con el comando build y ejecutar el contenedor

```
BELU@belenaguilarv MINGW64 ~/go/src/github.com/belenaguilarv/IngenieriaDeSoftware3/TP6 - Creacion de imagenes con Docker
/MiProyectoWebAPI (main)
$ docker build -t miproyectowebapi .
[+] Building 1.4s (13/13) FINISHED                                                                          docker:default
 => [internal] load build definition from Dockerfile                                                                 0.0s
 => => transferring dockerfile: 521B                                                                                 0.0s
 => [internal] load .dockerignore                                                                                    0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0                                                    1.3s
 => [build 1/8] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c   0.0s
 => [internal] load build context                                                                                    0.0s
 => => transferring context: 3.47kB                                                                                  0.0s
 => CACHED [build 2/8] WORKDIR /src                                                                                  0.0s
 => CACHED [build 3/8] COPY [MiProyectoWebAPI.csproj, .]                                                             0.0s
 => CACHED [build 4/8] RUN dotnet restore "./MiProyectoWebAPI.csproj"                                                0.0s
 => CACHED [build 5/8] COPY . .                                                                                      0.0s
 => CACHED [build 6/8] WORKDIR /src/.                                                                                0.0s
 => CACHED [build 7/8] RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build                           0.0s
 => CACHED [build 8/8] RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=fals   0.0s
 => exporting to image                                                                                               0.0s
 => => exporting layers                                                                                              0.0s
 => => writing image sha256:5083a09c318cdf5253275ee69a383a2e3078b344537e369ced087935c32a2119                         0.0s
 => => naming to docker.io/library/miproyectowebapi                                                                  0.0s

What's Next?
[2023-10-03T12:56:15.955483300Z][docker-credential-desktop.system][W] Windows version might not be up-to-date: The syste
m cannot find the file specified.
  View a summary of image vulnerabilities and recommendations → docker scout quickview

BELU@belenaguilarv MINGW64 ~/go/src/github.com/belenaguilarv/IngenieriaDeSoftware3/TP6 - Creacion de imagenes con Docker
/MiProyectoWebAPI (main)
$ docker run -p 8080:80 -it --rm miproyectowebapi
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
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: /src
```

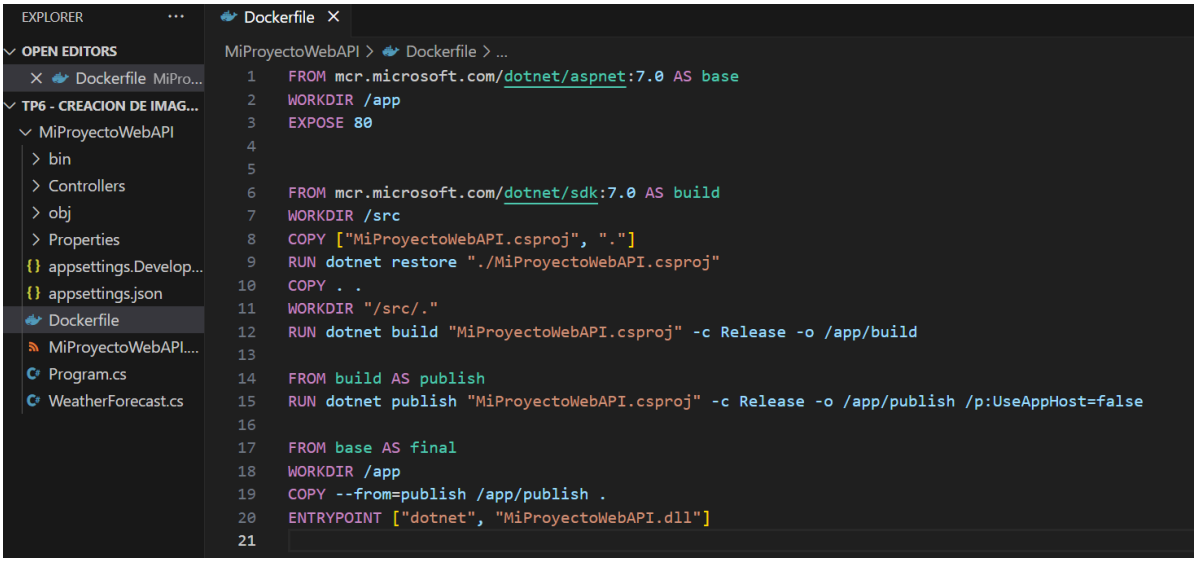


Entrar a la terminal del contenedor y ver directorios src, app/build y app/publish



3- Dockerfiles Multi Etapas

Modificar el dockerfile para el proyecto anterior, analizar y explicar el nuevo Dockerfile, incluyendo las nuevas instrucciones.



Este Dockerfile consta de cuatro etapas:

Etapa 1: Base

Comienza igual que en el Dockerfile original. Configura el directorio de trabajo en `/app` y expone el puerto 80.

Etapa 2: Build

Se copian los archivos de proyecto, se ejecuta el proceso de restauración (`dotnet restore`), se copian los archivos fuente, se realiza la compilación (`dotnet build`) y se publica la aplicación (`dotnet publish`) en la carpeta `/app/build`.

Etapa 3: Publish

Aca se toma como base la imagen de la etapa "build" y se ejecuta nuevamente el proceso de publicación (`dotnet publish`) en la carpeta `/app/publish`.

> 💡 Esta etapa solo copia los archivos necesarios para la aplicación publicada, no los archivos de desarrollo y se contruye una imagen más pequeña.

Etapa 4: Final

Se toma como base la imagen "base" y se copian los archivos publicados desde la etapa "publish" (`/app/publish`) a la carpeta `/app` de la imagen final. Esto crea una imagen final que solo contiene los archivos necesarios para ejecutar la aplicación publicada.

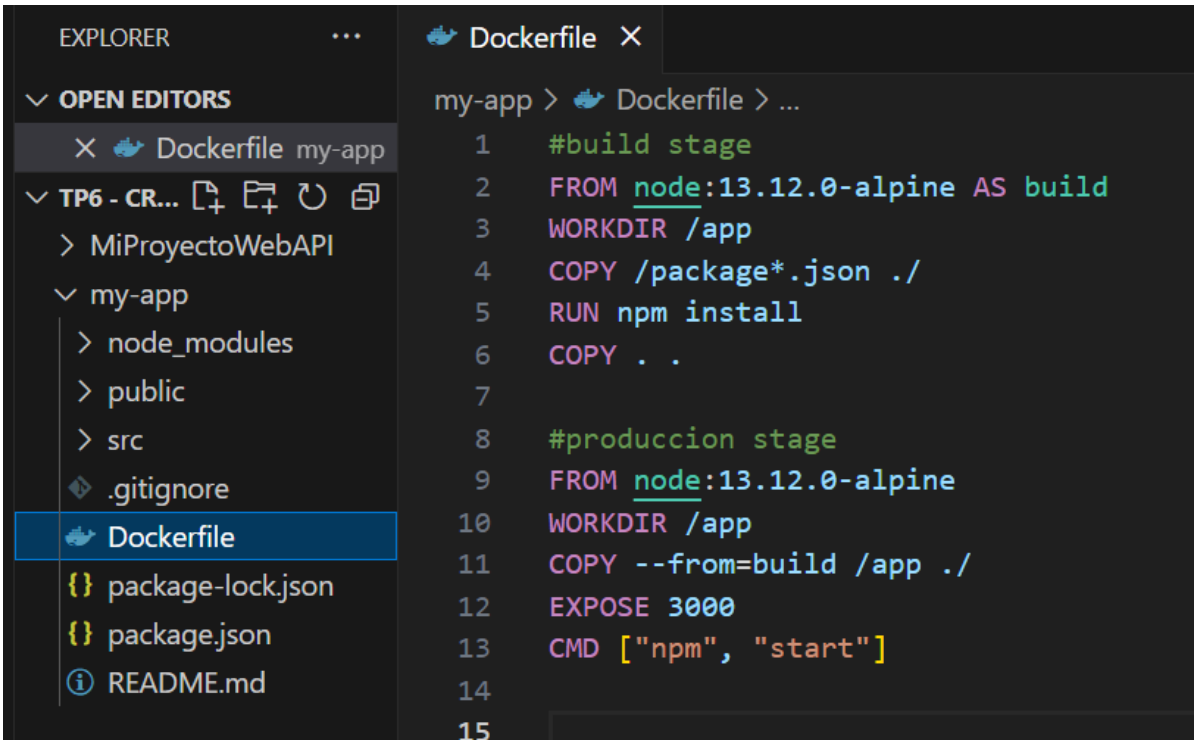
4- Imagen para aplicación web en Nodejs

Crear una la carpeta `trabajo-practico-06/nodejs-docker`

Generar un proyecto siguiendo los pasos descriptos en el trabajo práctico 5 para Nodejs

Escribir un Dockerfile para ejecutar la aplicación web localizada en ese directorio

- Idealmente que sea multistage, con una imagen de build y otra de producción.
- Usar como imagen base **node:13.12.0-alpine**
- Ejecutar **npm install** dentro durante el build.
- Exponer el puerto 3000



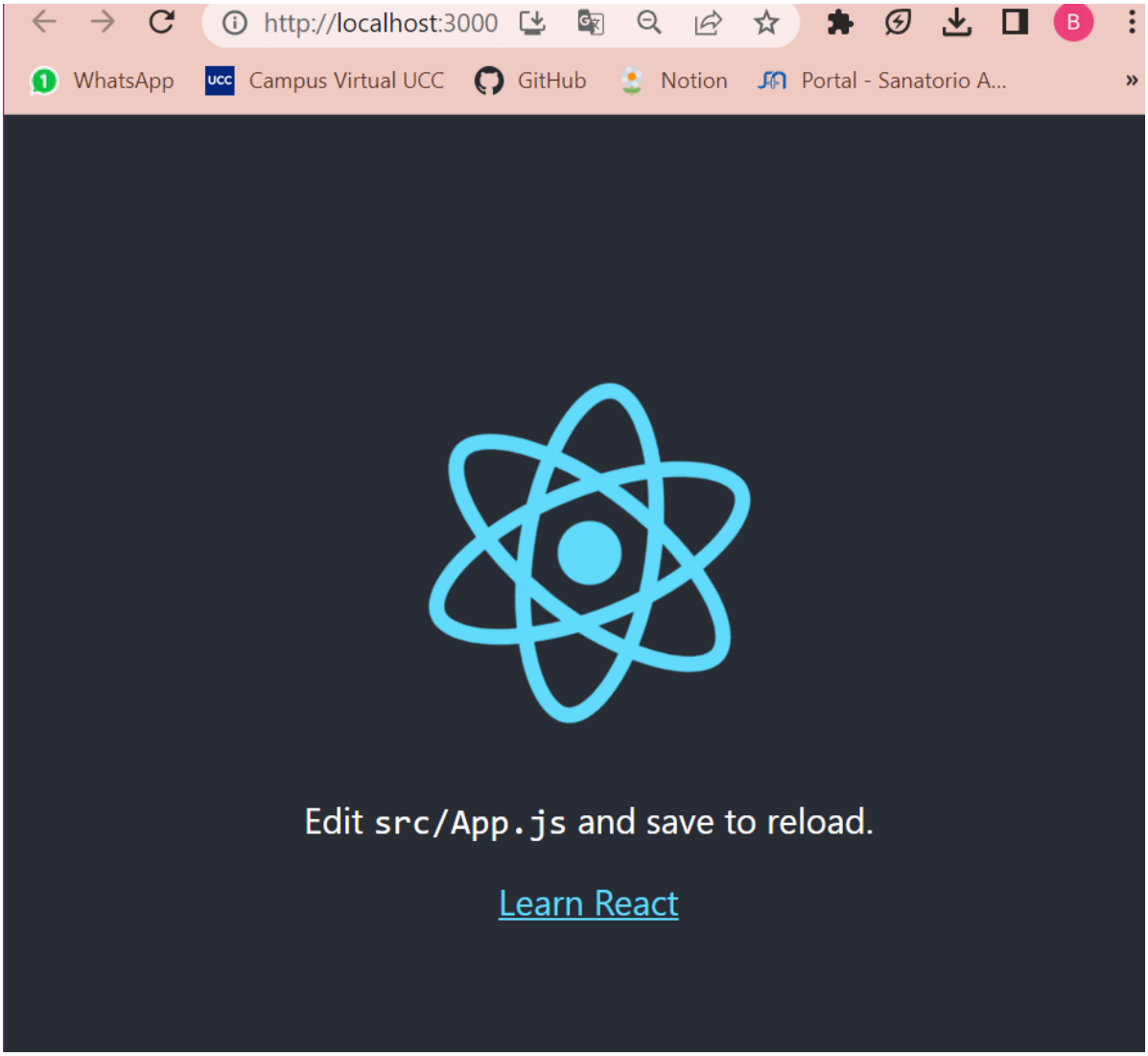
Hacer un build de la imagen, nombrar la imagen test-node.

```
$ docker build -t test-node .
[+] Building 0.0s (0/1)                                                                    docker:default
[2023-10-10T19:19:20.861751600Z][docker-credential-desktop.system][W] Windows version might not be up-to-date: The syste[
+] Building 33.7s (12/12) FINISHED                                                         docker:default
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 279B                                                                0.0s
 => [internal] load metadata for docker.io/library/node:13.12.0-alpine                              2.3s
 => [auth] library/node:pull token for registry-1.docker.io                                         0.0s
 => [internal] load build context                                                                   3.5s
 => => transferring context: 3.32MB                                                                 3.5s
 => [build 1/5] FROM docker.io/library/node:13.12.0-alpine@sha256:cc85e728fab3827ada20a181ba280cae1f8b625f256e2c8  0.0s
 => CACHED [build 2/5] WORKDIR /app                                                                 0.0s
 => CACHED [build 3/5] COPY /package*.json ./                                                       0.0s
 => CACHED [build 4/5] RUN npm install                                                              0.0s
 => [build 5/5] COPY . .                                                                            8.6s
 => [stage-1 3/3] COPY --from=build /app ./                                                         7.8s
 => exporting to image                                                                              6.9s
 => => exporting layers                                                                             6.9s
 => => writing image sha256:5d2835681bae9750f7adfc639758c8cfcc34a7809d3c170cd6ca0e5ffe43f80f        0.0s
 => => naming to docker.io/library/test-node                                                        0.0s

What's Next?
[2023-10-10T19:19:54.541043100Z][docker-credential-desktop.system][W] Windows version might not be up-to-date: The system
 cannot find the file specified.
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Ejecutar la imagen test-node publicando el puerto 3000.

```
BELU@belenaguilarv MINGW64 ~/go/src/github.com/belenaguilarv/IngenieriaDeSoftware3/TP6 - Creacion de imagenes con Docker/my-app (main)
$ docker run -p 3000:3000 test-node

> my-app@0.1.0 start /app
> react-scripts start

(node:30) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(node:30) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.17.0.2:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Compiling...
Compiled successfully!
webpack compiled successfully
```

Verificar en http://localhost:3000 que la aplicación está funcionando.



5- Publicar la imagen en Docker Hub.

Crear una cuenta en Docker Hub si no se dispone de una.

Registrase localmente a la cuenta de Docker Hub:

```
docker login
```

Crear un tag de la imagen generada en el ejercicio 3. Reemplazar <mi_usuario> por el creado en el punto anterior.

```
docker tag test-node belenaguilarv/test-node:latest
```

Subir la imagen a Docker Hub con el comando

```
docker push belenaguilarv/test-node:latest
```

Como resultado de este ejercicio mostrar la salida de consola, o una captura de pantalla de la imagen disponible en Docker Hub.