

# Trabajo Práctico 6 - Construcción de Imágenes de Docker

## Desarrollo:

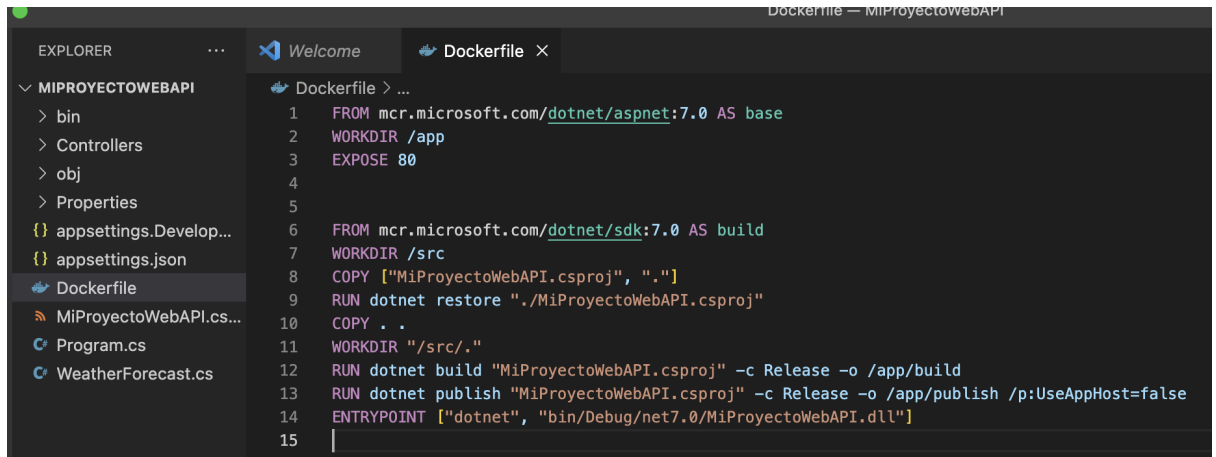
### 1- Conceptos de Dockerfiles

- Leer <https://docs.docker.com/engine/reference/builder/> (tiempo estimado 2 horas)
- Describir las instrucciones
  - **FROM:** La instrucción FROM se utiliza para especificar la imagen base desde la cual se construirá la nueva imagen de Docker. La imagen base es el punto de partida para la construcción de la nueva imagen y contiene un sistema operativo y un conjunto inicial de herramientas y bibliotecas.
  - **RUN:** La instrucción RUN se utiliza para ejecutar comandos en un nuevo contenedor durante el proceso de construcción de la imagen. Se pueden ejecutar múltiples comandos RUN en una sola imagen para instalar software, configurar el entorno, y realizar otras tareas de configuración.
  - **ADD y COPY:** Estas dos instrucciones se utilizan para copiar archivos y directorios desde el sistema de archivos del host al sistema de archivos del contenedor durante la construcción de la imagen. La principal diferencia entre ellas es que ADD permite la posibilidad de realizar algunas operaciones de extracción automática de archivos, mientras que COPY simplemente copia archivos desde el host al contenedor.
  - **EXPOSE:** La instrucción EXPOSE se utiliza para especificar qué puertos de red deben estar disponibles para las conexiones entrantes en el contenedor cuando se ejecute. No abre automáticamente los puertos, pero es útil para documentar qué puertos se deben exponer y se utiliza comúnmente en conjunción con la opción -p al ejecutar contenedores para mapear puertos del host al contenedor.
  - **CMD:** La instrucción CMD se utiliza para proporcionar un comando o una lista de comandos que se ejecutarán cuando se inicie un contenedor basado en la imagen. Puede haber solo una instrucción CMD en un Dockerfile. Si se especifica más de una, la última prevalecerá.
  - **ENTRYPOINT:** La instrucción ENTRYPOINT se utiliza para especificar un comando o una lista de comandos que se ejecutarán como un ejecutable cuando se inicie un contenedor basado en la imagen. A diferencia de CMD, los argumentos proporcionados en el momento de

ejecución se pasan como argumentos al comando especificado en ENTRYPOINT. Si se utiliza ENTRYPOINT, se pueden anular sus efectos en tiempo de ejecución utilizando la instrucción CMD.

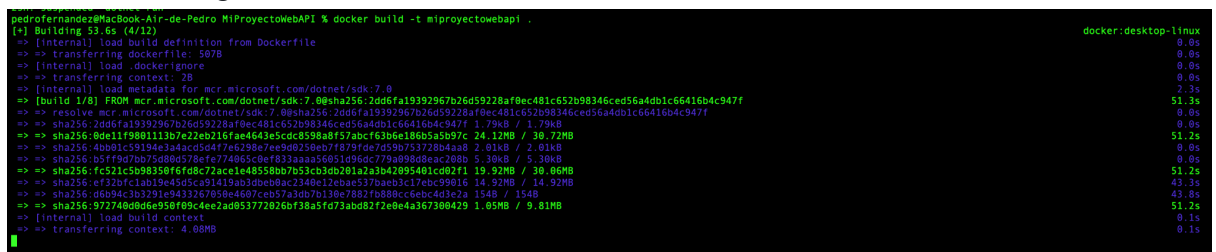
## 2- Generar imagen de docker

- Utilizar el resultado del paso 1 del TP 5
- Agregar un archivo llamado Dockerfile (en el directorio raiz donde se encuentran todos los archivos y directorios)



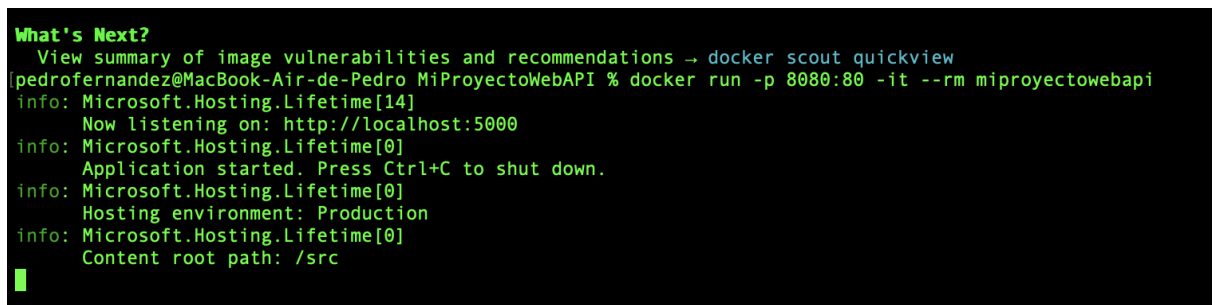
```
Dockerfile — MiProyectoWebAPI
EXPLORER
  > MIPROYECTOWEBAPI
    > bin
    > Controllers
    > obj
    > Properties
    {} appsettings.Develop...
    {} appsettings.json
    Dockerfile
    MiProyectoWebAPI.csproj
    Program.cs
    WeatherForecast.cs
  Dockerfile > ...
1 FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
2 WORKDIR /app
3 EXPOSE 80
4
5
6 FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
7 WORKDIR /src
8 COPY ["MiProyectoWebAPI.csproj", "."]
9 RUN dotnet restore "./MiProyectoWebAPI.csproj"
10 COPY . .
11 WORKDIR "/src/"
12 RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build
13 RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false
14 ENTRYPOINT ["dotnet", "bin/Debug/net7.0/MiProyectoWebAPI.dll"]
15
```

- Generar la imagen de docker con el comando build



```
pedrofernandez@MacBook-Air-de-Pedro MiProyectoWebAPI % docker build -t miprojectowebapi .
[+] Building 53.6s (4/12)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 507B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0
=> [build 1/8] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c66416b4c947f
=> => resolve mcr.microsoft.com/dotnet/sdk:7.0@sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c66416b4c947f
=> => sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c66416b4c947f 1.79kB / 1.79kB
=> => sha256:d6b11f98b113b7a22eb01f9ae4643e5cd8598a8f57abcf63b6e186b5a5b97c 24.12MB / 30.72MB
=> => sha256:4bb01c59194e3a4acd5d4f7e6298e7ee9d0250eb7f079fde7d59b753728b4aa8 2.01kB / 2.01kB
=> => sha256:b5f7f9d7b75d8bd578efe774b65c0ef833aaaa56051096dc779a098d8eac208b 5.30kB / 5.30kB
=> => sha256:ef32bfc1ab19e45d5ca91419ab3dbb8ac2340e12ebac537baeb3c17ebc99016 14.92MB / 14.92MB
=> => sha256:d6b94c3b3291e433267050e4607ceb57a3db7b130e7882fb880cc6ebc4d3e2a 154B / 154B
=> => sha256:9774bd8d06e958f09c4ee2ad853777026bf38a5fd73abd82f2e8e4a367300429 1.05MB / 9.81MB
=> [internal] load build context
=> => transferring context: 4.08MB
```

- Ejecutar el contenedor



```
What's Next?
View summary of image vulnerabilities and recommendations → docker scout quickview
[pedrofernandez@MacBook-Air-de-Pedro MiProyectoWebAPI % docker run -p 8080:80 -it --rm miprojectowebapi
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
```

## 3- Dockerfiles Multi Etapas

- Modificar el dockerfile para el proyecto anterior de la siguiente forma

```

1 FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
2 WORKDIR /app
3 EXPOSE 80
4
5
6 FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
7 WORKDIR /src
8 COPY ["MiProyectoWebAPI.csproj", "."]
9 RUN dotnet restore "/MiProyectoWebAPI.csproj"
10 COPY . .
11 WORKDIR "/src/."
12 RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build
13
14 FROM build AS publish
15 RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false
16
17 FROM base AS final
18 WORKDIR /app
19 COPY --from=publish /app/publish .
20 ENTRYPOINT ["dotnet", "MiProyectoWebAPI.dll"]
21
22

```

- Construir nuevamente la imagen

```

Start a build
pedrofernandez@MacBook-Air-de-Pedro MiProyectoWebAPI % docker build -t miprojectowebapi .
[+] Building 5.6s (18/18) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 582B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0
=> [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c66416b4c947f
=> [internal] load build context
=> => transferring context: 3.98kB
=> [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:933ae169296fe093776749cc47fd60e0a6e85bde607651a3fc6cd97bb5bc1ce0
=> => resolve mcr.microsoft.com/dotnet/aspnet:7.0@sha256:933ae169296fe093776749cc47fd60e0a6e85bde607651a3fc6cd97bb5bc1ce0
=> => sha256:933ae169296fe093776749cc47fd60e0a6e85bde607651a3fc6cd97bb5bc1ce0 1.79kB / 1.79kB
=> => sha256:b8f282041842ae92d66b5715c071d208d375baa62ab66478448794f4bb69aca8 1.37kB / 1.37kB
=> => sha256:2a32215d5e8c8bd94752c8de9ce69abfa607bf9a38583574262a1efc000c033e 2.36kB / 2.36kB
=> CACHED [build 2/7] WORKDIR /src
=> CACHED [build 3/7] COPY [MiProyectoWebAPI.csproj, .]
=> CACHED [build 4/7] RUN dotnet restore "/MiProyectoWebAPI.csproj"
=> [build 5/7] COPY . .
=> [base 2/2] WORKDIR /app
=> [build 6/7] WORKDIR /src/
=> [final 1/2] WORKDIR /app
=> [build 7/7] RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build
=> [publish 1/1] RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false
=> [final 2/2] COPY --from=publish /app/publish .
=> exporting to image
=> => exporting layers
=> => writing image sha256:fbd947ae1ffbb5aaf0732d08745775f95dd47ca98e24b2a1bb18697275776973
=> => naming to docker.io/library/miprojectowebapi

What's Next?
View summary of image vulnerabilities and recommendations -> docker scout quickview
pedrofernandez@MacBook-Air-de-Pedro MiProyectoWebAPI %

```

- Analizar y explicar el nuevo Dockerfile, incluyendo las nuevas instrucciones.

Las instrucciones son las mismas que en el Dockerfile original, pero están organizadas en etapas:

La primera etapa (base) establece la imagen base que se utilizará para la ejecución de la aplicación. Se configura el directorio de trabajo en /app y se expone el puerto 80 para que la aplicación escuche las conexiones entrantes en ese puerto.

La segunda etapa (build) se utiliza para compilar la aplicación. Se configura el directorio de trabajo en /src, se copia el archivo de proyecto

MiProyectoWebAPI.csproj y se realiza una restauración (dotnet restore) para obtener las dependencias necesarias. Luego se copian todos los archivos al directorio de trabajo y se compila la aplicación con el comando dotnet build.

La tercera etapa (publish) se utiliza para publicar la aplicación. Se basa en la etapa build y ejecuta el comando dotnet publish para crear los archivos de la aplicación publicada en el directorio /app/publish.

La cuarta etapa (final) se basa en la etapa base y copia los archivos publicados desde la etapa publish al directorio de trabajo en /app. Finalmente, establece el punto de entrada (ENTRYPOINT) para iniciar la aplicación cuando se ejecute el contenedor.

#### 4- Imagen para aplicación web en Nodejs

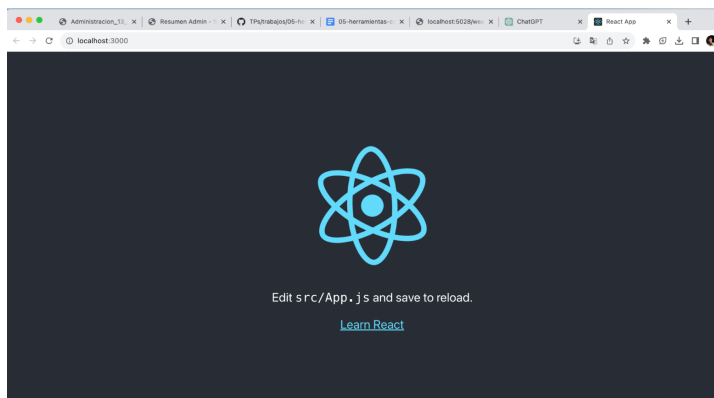
- Crear una la carpeta `trabajo-practico-06/nodejs-docker`



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

```
pedrofernandez@MacBook-Air-de-Pedro: trabajo-practico-06:nodejs-docker % npx create-react-app my-app
Creating a new React app in /Users/pedrofernandez/Desktop/Facu Pedro/Ing de Software III/ing-sw-3/06-construccion-imagenes-docker/trabajo-practico-06:nodejs-docker/my-app.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1459 packages in 3m
241 packages are looking for funding
  run `npm fund` for details
Installing template dependencies using npm...
added 69 packages, and changed 1 package in 9s
245 packages are looking for funding
  run `npm fund` for details
Removing template package using npm...
removed 1 package, and audited 1528 packages in 1s
```



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

```

Welcome  Dockerfile x
Dockerfile > ...
1  # Etapa 1: Construir la aplicación
2  FROM node:13.12.0-alpine AS build
3
4  WORKDIR /app
5
6  # Copiar archivos de dependencias
7  COPY package*.json ./
8
9  # Instalar las dependencias
10 RUN npm install
11
12 # Copiar el código fuente de la aplicación
13 COPY . .
14
15 # Compilar la aplicación (si es necesario)
16 # RUN npm run build
17
18 # Etapa 2: Imagen de producción
19 FROM node:13.12.0-alpine AS production
20
21 WORKDIR /app
22
23 # Copiar los archivos de la etapa de construcción
24 COPY --from=build /app .
25
26 # Exponer el puerto en el que la aplicación escuchará
27 EXPOSE 3000
28
29 # Comando para iniciar la aplicación
30 CMD ["npm", "start"]
31

```

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

```

pedrofernandez@MacBook-Air-de-Pedro my-app % docker build -t test-node .

[+] Building 17.5s (4/10)                                docker:desktop-linux
=> [internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 638B                     0.0s
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 2B                           0.0s
=> [internal] load metadata for docker.io/library/node:13.12.0-alpine 2.9s
=> [build 1/5] FROM docker.io/library/node:13.12.0-alpine@sha256:cc85e7 14.5s
=> => resolve docker.io/library/node:13.12.0-alpine@sha256:cc85e728fab38 0.0s
=> => sha256:cc85e728fab3827ada20a181ba280cae1f8b625f256 1.19kB / 1.19kB 0.0s
=> => sha256:c8b4c2938b54a8a457c601dd5cf10e2890d1f750d24 1.16kB / 1.16kB 0.0s

```

- Ejecutar la imagen test-node publicando el puerto 3000.

```
Compiled successfully!

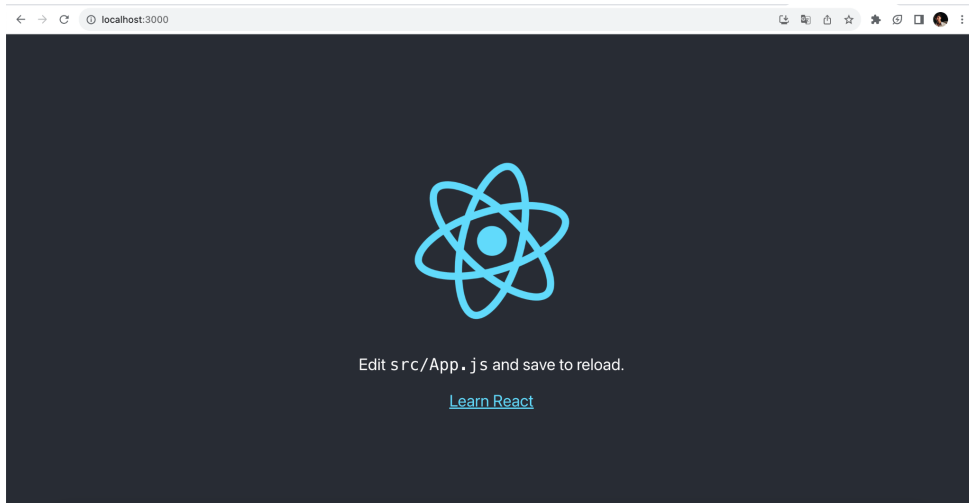
You can now view my-app in the browser.

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
```

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



- Proveer el Dockerfile y los comandos ejecutados como resultado de este ejercicio.

```
docker build -t test-node .
```

```
docker run -p 3000:3000 -it --rm test-node
```

## 5- Publicar la imagen en Docker Hub.

- Crear una cuenta en Docker Hub si no se dispone de una.
- Registrarse localmente a la cuenta de Docker Hub:

```
Username: piterfmz
[Password:
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
pedrofernandez@MacBook-Air-de-Pedro my-app %
```

- Crear un tag de la imagen generada en el ejercicio 3. Reemplazar <mi\_usuario> por el creado en el punto anterior.

```
n more at https://docs.docker.com/go/access-tokens/  
[pedrofernandez@MacBook-Air-de-Pedro my-app % docker tag test-node piterfmz/test-]  
node:latest  
[pedrofernandez@MacBook-Air-de-Pedro my-app % ]
```

- Subir la imagen a Docker Hub con el comando

```
[pedrofernandez@MacBook-Air-de-Pedro my-app % docker tag test-node piterfmz/test-]  
node:latest  
[pedrofernandez@MacBook-Air-de-Pedro my-app % docker push piterfmz/test-node:late]  
st  
The push refers to repository [docker.io/piterfmz/test-node]  
d613f4ff5ab4: Pushing 143.6MB/244.6MB  
4e8c52a2b011: Pushed  
970cce441d1c: Mounted from library/node  
6ed2d5b46491: Mounted from library/node  
e012a64bf318: Mounted from library/node  
294ac687b5fc: Mounted from library/node  
[ ]
```

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

