

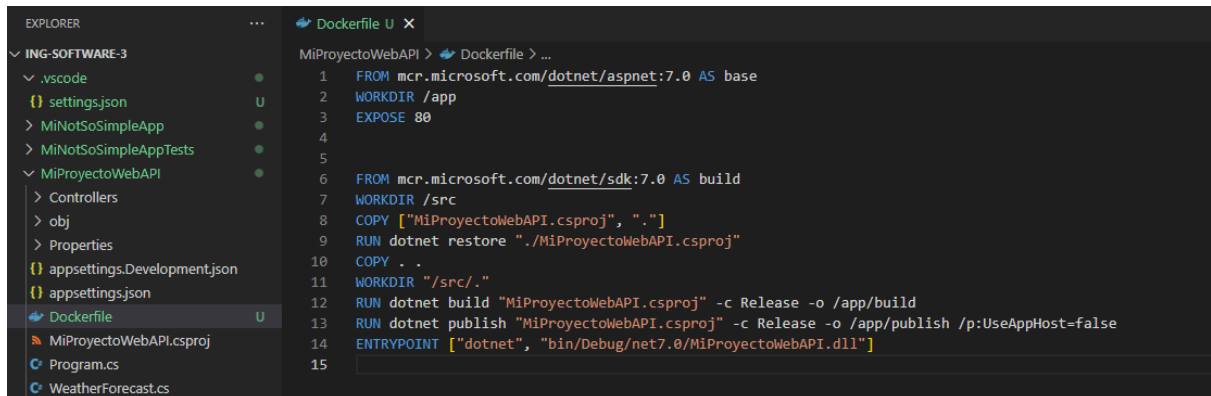
TP6

1- Conceptos de Dockerfiles

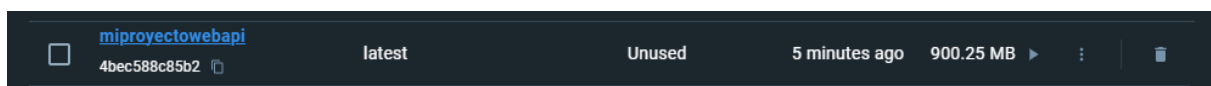
1. **FROM** : Esta es la primera instrucción en un Dockerfile y especifica la imagen base que se utilizará como punto de partida para construir tu imagen de contenedor. Por ejemplo, puedes usar **FROM ubuntu:20.04** para comenzar con la imagen base de Ubuntu 20.04.
2. **RUN** : La instrucción **RUN** se utiliza para ejecutar comandos en el contenedor durante la construcción de la imagen. Puedes utilizarla para instalar paquetes, configurar software o realizar otras tareas de configuración. Por ejemplo, **RUN apt-get update && apt-get install -y apache2** instalaría Apache en el contenedor.
3. **ADD** : La instrucción **ADD** se utiliza para copiar archivos y directorios desde el sistema de archivos del host al sistema de archivos del contenedor. Puedes usar esta instrucción para agregar archivos necesarios para tu aplicación a la imagen.
4. **COPY** : Al igual que **ADD**, la instrucción **COPY** también se utiliza para copiar archivos y directorios desde el sistema de archivos del host al sistema de archivos del contenedor. Sin embargo, a diferencia de **ADD**, **COPY** solo funciona con archivos locales y no admite la URL ni la descompresión automática de archivos remotos.
5. **EXPOSE** : La instrucción **EXPOSE** se utiliza para indicar qué puertos deben exponerse cuando se ejecute un contenedor basado en la imagen. Esto no abre automáticamente los puertos en el host, pero proporciona información sobre qué puertos deberían estar disponibles para ser mapeados cuando se ejecute el contenedor.
6. **CMD** : La instrucción **CMD** define el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en la imagen. Puede especificar un comando y sus argumentos, por ejemplo, **CMD ["python", "app.py"]**. Si se proporciona más de una instrucción **CMD** en el Dockerfile, solo se ejecutará la última.
7. **ENTRYPOINT** : Similar a **CMD**, la instrucción **ENTRYPOINT** se utiliza para especificar el comando que se ejecutará cuando se inicie un contenedor. Sin embargo, a diferencia de **CMD**, los argumentos pasados en la línea de comandos al ejecutar el contenedor se agregarán al final del comando **ENTRYPOINT**. Esto permite que el

contenedor sea más flexible y pueda recibir argumentos personalizados al ejecutarse.

2- Generar imagen de docker



```
PS D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiProyectoWebAPI> docker build -t miprojectowebapi .
[+] Building 1.6s (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 1.4s
=> [build 1/8] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:2dd6fa19392967b26d59228af0ec481c652b98346ced56a4db1c66416b4c947f 0.0s
=> [internal] load build context                                  0.1s
=> => transferring context: 696B                                       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
```

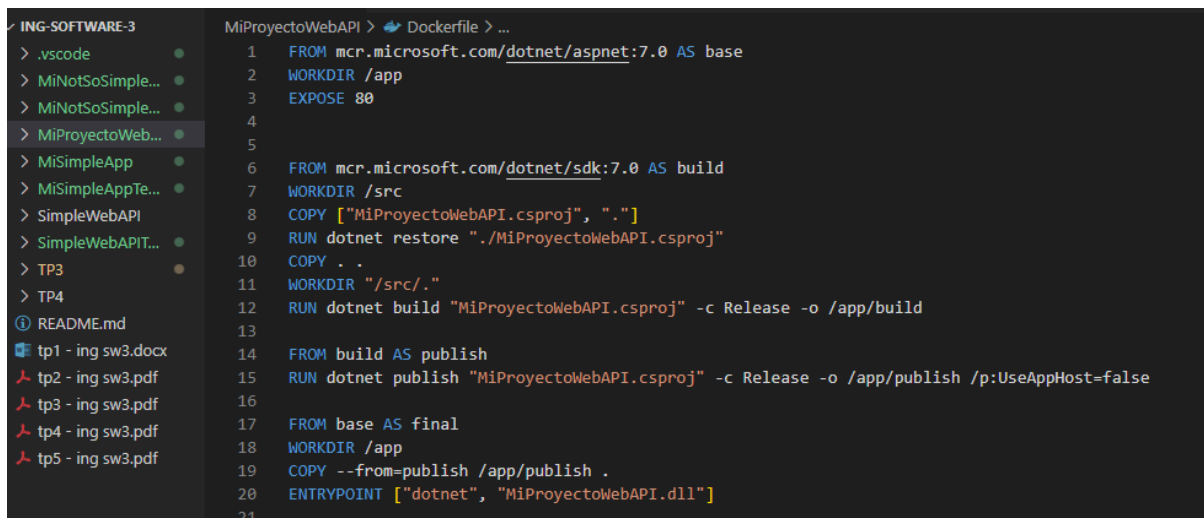


```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiProyectoWebAPI> docker run -p 8080:80 -it --rm miprojectowebapi
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      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
```

Ahora entramos a la terminal del contenedor y vemos los directorios.

```
# cd /src
# ls
Controllers  MiProyectoWebAPI.csproj  Properties          appsettings.Development.json  bin
Dockerfile   Program.cs               WeatherForecast.cs  appsettings.json              obj
# cd /app/build
# ls
MiProyectoWebAPI               Microsoft.AspNetCore.OpenApi.dll  Swashbuckle.AspNetCore.SwaggerUI.dll
MiProyectoWebAPI.deps.json     Microsoft.OpenApi.dll            appsettings.Development.json
MiProyectoWebAPI.dll           Newtonsoft.Json.dll              appsettings.json
MiProyectoWebAPI.pdb           Swashbuckle.AspNetCore.Swagger.dll
MiProyectoWebAPI.runtimeconfig.json  Swashbuckle.AspNetCore.SwaggerGen.dll
# cd /app/publish
# ls
MiProyectoWebAPI.deps.json     Microsoft.OpenApi.dll            appsettings.Development.json
MiProyectoWebAPI.dll           Newtonsoft.Json.dll              appsettings.json
MiProyectoWebAPI.pdb           Swashbuckle.AspNetCore.Swagger.dll  web.config
MiProyectoWebAPI.runtimeconfig.json  Swashbuckle.AspNetCore.SwaggerGen.dll
Microsoft.AspNetCore.OpenApi.dll     Swashbuckle.AspNetCore.SwaggerUI.dll
#
```

3- Dockerfiles Multi Etapas



```
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
```

Este Dockerfile se utiliza para construir una imagen de Docker para una aplicación ASP.NET Core utilizando .NET 7.0. A continuación, analizaré y explicaré las instrucciones presentes en este Dockerfile:

1. `FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base`: Esta instrucción define la imagen base que se utilizará como punto de partida. En este caso, se utiliza la imagen `mcr.microsoft.com/dotnet/aspnet:7.0`, que es una imagen oficial de Microsoft que contiene el entorno ASP.NET Core.
2. `WORKDIR /app`: Establece el directorio de trabajo dentro del contenedor en `/app`. Todas las siguientes instrucciones se ejecutarán en este directorio.

3. `EXPOSE 80` : Esta instrucción expone el puerto 80 del contenedor para que las solicitudes web puedan ser dirigidas a este puerto cuando se ejecute la aplicación ASP.NET Core.
4. `FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build` : Esta instrucción define una segunda etapa llamada "build" utilizando la imagen `mcr.microsoft.com/dotnet/sdk:7.0` . Esta etapa se utilizará para compilar la aplicación.
5. `WORKDIR /src` : Establece el directorio de trabajo dentro de la etapa "build" en `/src` .
6. `COPY ["MiProyectoWebAPI.csproj", "."]` : Copia el archivo de proyecto `MiProyectoWebAPI.csproj` al directorio de trabajo actual en la etapa "build". Esto permite que Docker pueda restaurar las dependencias de la aplicación.
7. `RUN dotnet restore "./MiProyectoWebAPI.csproj"` : Ejecuta el comando `dotnet restore` para restaurar las dependencias del proyecto.
8. `COPY . .` : Copia todo el contenido del directorio de construcción actual al directorio de trabajo en la etapa "build". Esto incluye el código fuente de la aplicación.
9. `WORKDIR "/src/."` : Cambia el directorio de trabajo en la etapa "build" al directorio raíz del proyecto.
10. `RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build` : Compila la aplicación utilizando el perfil de compilación "Release" y la salida se almacena en el directorio `/app/build` en la etapa "build".
11. `FROM build AS publish` : Esta instrucción define una tercera etapa llamada "publish" que se basa en la etapa "build". Se utilizará para publicar la aplicación.
12. `RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false` : Publica la aplicación en el directorio `/app/publish` en la etapa "publish". La opción `/p:UseAppHost=false` se utiliza para evitar la creación de un ejecutable de host para la aplicación, ya que se ejecutará en el contenedor de Docker.
13. `FROM base AS final` : Esta instrucción define una cuarta etapa llamada "final" que se basa en la etapa "base". Esta etapa será la imagen final que se utilizará para ejecutar la aplicación.
14. `WORKDIR /app` : Establece el directorio de trabajo en la etapa "final" en `/app` .

15. `COPY --from=publish /app/publish .` : Copia los archivos publicados desde la etapa "publish" al directorio de trabajo en la etapa "final".
16. `ENTRYPOINT ["dotnet", "MiProyectoWebAPI.dll"]` : Establece el punto de entrada de la aplicación cuando se inicie el contenedor. En este caso, se ejecutará el archivo `MiProyectoWebAPI.dll` utilizando el comando `dotnet` .

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
- 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

```
# Etapa de build
FROM node:13.12.0-alpine as build
WORKDIR /app
# Copia los archivos de package. json y package-lock.json para gestionar las dependencias
COPY /package*.json -/
# Instala las dependencias
RUN npm install
# Copia el resto de los archivos de la aplicación
COPY ..
# Etapa de producción
FROM node:13.12.0-alpine
WORKDIR /app
# Copia los archivos de la etapa de construcción (incluyendo "my-app")
COPY --from=build /app -/
# Exponer el puerto 3000
EXPOSE 3000
# Comando para iniciar la aplicación]
CMD ["npm", "start"]
```

```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker build
-t test-node .
[+] Building 71.0s (12/12) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 610B                                              0.0s
=> [internal] load .dockerignore                                                  0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/node:13.12.0-alpine           1.8s
=> [auth] library/node:pull token for registry-1.docker.io                     0.0s
=> [build 1/5] FROM docker.io/library/node:13.12.0-alpine@sha256:cc85e728fab3827ada20a181ba280cae1f8b625f256e2c8  0.0s
=> [internal] load build context                                                10.4s
=> => transferring context: 3.27MB                                              10.2s
=> 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 . .                                                        19.4s
=> [stage-1 3/3] COPY --from=build /app ./                                     20.4s
=> exporting to image                                                         9.0s
=> => exporting layers                                                         8.9s
=> => writing image sha256:e5fc2e5386467a4e552ace483ed49eccc8829e9db90c10723b26ec0b8eedb8a2  0.0s
=> => naming to docker.io/library/test-node                                    0.0s
```

```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker run -p
3000:3000 test-node
> my-app@0.1.0 start /app
> react-scripts start

(node:31) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setu
pMiddlewares' option.
(node:31) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'se
tupMiddlewares' 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 maintained 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.

Failed to compile.

[eslint] package.json » eslint-config-react-app/jest#overrides[0]:
  Environment key "jest/globals" is unknown
ERROR in [eslint] package.json » eslint-config-react-app/jest#overrides[0]:
  Environment key "jest/globals" is unknown

webpack compiled with 1 error
Compiling...
Compiled successfully!
webpack compiled successfully
```

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:

```
PS C:\Users\Hp> docker login
Authenticating with existing credentials...
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/acc
ess-tokens/
PS C:\Users\Hp> |
```

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

```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker tag te
st-node sanrivsal/test-node:latest
```

- Subir la imagen a Docker Hub con el comando

```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker tag miproyectowebapi sanrivsal/miproyectowebapi:latest

D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker push sanrivsal/miproyectowebapi:latest
The push refers to repository [docker.io/sanrivsal/miproyectowebapi]
d5118102a730: Pushed
556184aa94e0: Pushed
5f70bf18a086: Mounted from sanrivsal/mywebapi
974e00a843cc: Pushed
dc58c92fc67d: Pushed
0c95f9286fb3: Pushed
4d52c67ee14c: Pushed
6dc3f8471b68: Pushed
06c225fd853c: Pushed
9e71e1d7fd9d: Pushed
621ac519e7ac: Pushed
13073704c26c: Pushed
2429ab113701: Pushed
7a17569cab26: Pushed
10764c37bcbc: Pushed
latest: digest: sha256:bf7d9e16972aeb63da12d07571c8ec3c1b22392e62127779dca7dbdfb2f92971 size: 3475
```

```
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker tag miproyectowebapi sanrivsal/miproyectowebapi:latest

D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3>docker push sanrivsal/miproyectowebapi:latest
The push refers to repository [docker.io/sanrivsal/miproyectowebapi]
d5118102a730: Pushed
556184aa94e0: Pushed
5f70bf18a086: Mounted from sanrivsal/mywebapi
974e00a843cc: Pushed
dc58c92fc67d: Pushed
0c95f9286fb3: Pushed
4d52c67ee14c: Pushed
6dc3f8471b68: Pushed
06c225fd853c: Pushed
9e71e1d7fd9d: Pushed
621ac519e7ac: Pushed
13073704c26c: Pushed
2429ab113701: Pushed
7a17569cab26: Pushed
10764c37bcbc: Pushed
latest: digest: sha256:bf7d9e16972aeb63da12d07571c8ec3c1b22392e62127779dca7dbdfb2f92971 size: 3475
```