

TP 6 - Construcción de Imágenes de Docker

Conceptos de Dockerfiles

- Describir las instrucciones

FROM: Esta es la primera instrucción en un Dockerfile y especifica la imagen base que se utilizará como punto de partida para construir la imagen de Docker.

RUN: La instrucción RUN se utiliza para ejecutar comandos dentro del contenedor durante la construcción de la imagen.

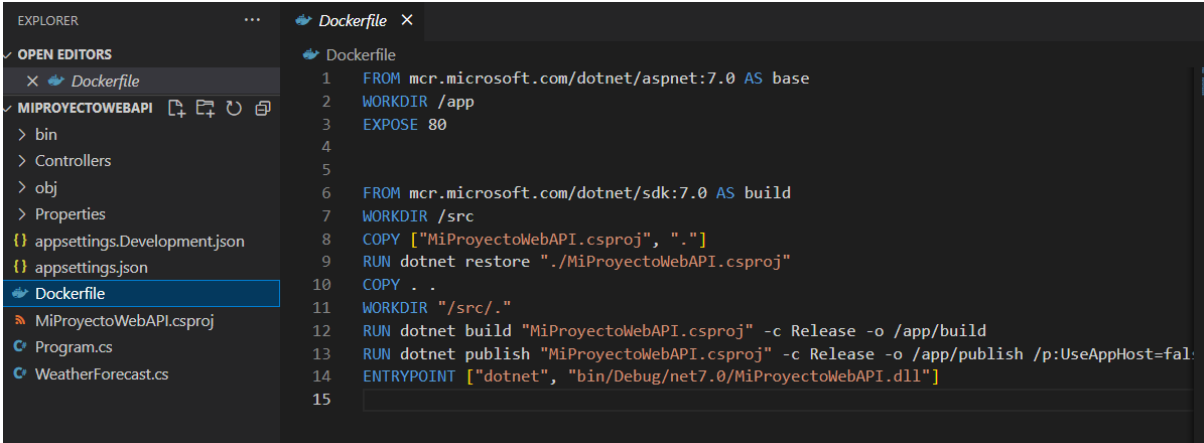
ADD y COPY: Estas instrucciones se utilizan para copiar archivos y directorios desde el sistema de archivos del host al sistema de archivos del contenedor. La diferencia principal entre ellas es que ADD puede realizar algunas operaciones adicionales, como la descompresión de archivos TAR automáticamente.

EXPOSE: La instrucción EXPOSE especifica los puertos en los que el contenedor escuchará las conexiones entrantes. No abre realmente el puerto en el host, pero documenta que el contenedor debe escuchar en esos puertos. Por ejemplo, EXPOSE 80 indica que el contenedor escuchará en el puerto 80.

CMD: Esta instrucción se utiliza para proporcionar un comando predeterminado que se ejecutará cuando se inicie el contenedor. Puede ser sobrescrito por un comando especificado en la línea de comandos al ejecutar el contenedor.

ENTRYPOINT: Similar a CMD, esta instrucción se utiliza para especificar el comando principal que se ejecutará cuando se inicie el contenedor. Sin embargo, a diferencia de CMD, los argumentos proporcionados en la línea de comandos al ejecutar el contenedor se pasan como argumentos al comando especificado en ENTRYPOINT.

Generar imagen de docker



```
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

```
PS C:\Users\sofia\OneDrive\Escritorio\IdS3\miwebapi\MiProyectoWebAPI> docker build -t miproyectowebapi .
[+] Building 41.5s (4/12)
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 521B 0.0s
=> [internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0 2.4s
=> [build 1/8] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:bdcfb498261ca18f023ac67615d814ea743aa3288eb880855fa2eb86c6 38.8s
=> => resolve mcr.microsoft.com/dotnet/sdk:7.0@sha256:bdcfb498261ca18f023ac67615d814ea743aa3288eb880855fa2eb86c6 0.0s
=> => sha256:892c71ccd8bab154b490e6764192f3f558834dd2ealb187f6e088ae42158462c 32.45MB / 32.45MB 4.7s
=> => sha256:987e56a754ab7921b37177ef5fb9380120864c87dd476e0658fd06a165895b31 7.17kB / 7.17kB 0.0s
=> => sha256:024f4b1106d8ee7f101c2c0e0cdc9a74906f119b53ded8c261631ee008327883 2.01kB / 2.01kB 0.0s
=> => sha256:7d97e254a0461b0a30b3f443f1daa0d620a3cc6ff4e2714cc1cfd96ace5b7a7e 31.42MB / 31.42MB 8.6s
=> => sha256:bdcfb498261ca18f023ac67615d814ea743aa3288eb880855fa2eb86c6313ccc 1.82kB / 1.82kB 0.0s
=> => sha256:3970ea6bbab9f0c58ea7c4734b7f41f1e66939f1e670b093c6166b19d16f375b 14.97MB / 14.97MB 10.6s
=> => sha256:0258bbc9dba08a056cd0ff86fa4de21f24355193eefa187b417a699241c8c502 156B / 156B 5.0s
=> => sha256:14c0c27e2667f4cc64cba0bfbdb9d6fcd6853edeb97fda00747efd6d6730f819b 10.12MB / 10.12MB 6.5s
=> => sha256:7454f91b71b3937c0ba871aba2436ee265cd1e7e5cb7404de14acc4feb5a7e62 25.37MB / 25.37MB 9.6s
=> => sha256:0c788bf06741d85c041ea1f1223672478aee79dd9621f9c7ef48067c62ae783 181.16MB / 181.16MB 27.9s
=> => extracting sha256:7d97e254a0461b0a30b3f443f1daa0d620a3cc6ff4e2714cc1cfd96ace5b7a7e 3.2s
=> => sha256:62d5b12f5d497593fd625fd558fe5bf212dcd39eaff524092db0bf5774435444 13.90MB / 13.90MB 11.6s
=> => extracting sha256:3970ea6bbab9f0c58ea7c4734b7f41f1e66939f1e670b093c6166b19d16f375b 0.9s
=> => extracting sha256:892c71ccd8bab154b490e6764192f3f558834dd2ealb187f6e088ae42158462c 2.0s
=> => extracting sha256:0258bbc9dba08a056cd0ff86fa4de21f24355193eefa187b417a699241c8c502 0.0s
=> => extracting sha256:14c0c27e2667f4cc64cba0bfbdb9d6fcd6853edeb97fda00747efd6d6730f819b 0.6s
=> => extracting sha256:7454f91b71b3937c0ba871aba2436ee265cd1e7e5cb7404de14acc4feb5a7e62 4.8s
```

- Ejecutar el contenedor

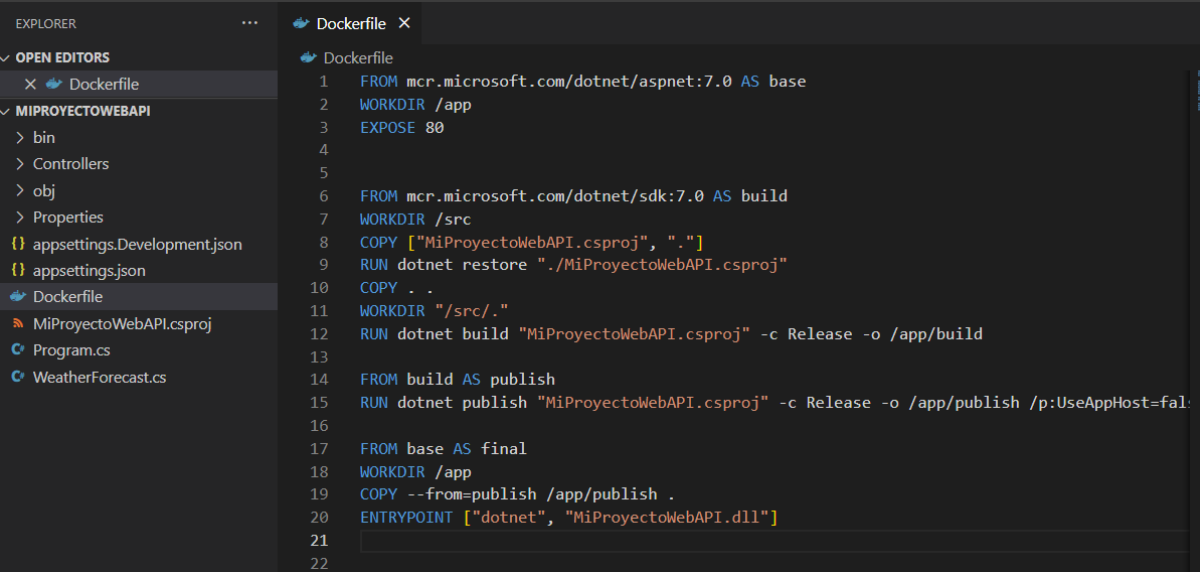
```
PS C:\Users\sofia\OneDrive\Escritorio\IdS3\miwebapi\MiProyectoWebAPI> docker run -p 8080:80 -it --rm miproyectowebapi
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
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

```
# 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
# |
```

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
```

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

Este Dockerfile se utiliza para construir una imagen de Docker para una aplicación web ASP.NET Core. Está organizado en varias etapas para optimizar el tamaño de la imagen y separar las tareas de construcción y ejecución de la aplicación. La última etapa utiliza la imagen base de ASP.NET Core para ejecutar la aplicación, mientras que las etapas anteriores se utilizan para compilar y publicar la aplicación.

FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base:

Esta instrucción especifica la imagen base a partir de la cual se construirá la imagen de Docker. En este caso, se utiliza la imagen oficial de ASP.NET Core 7.0.

La parte AS base le da un nombre a esta etapa o paso en la construcción de la imagen, lo que permitirá referenciar más adelante.

WORKDIR /app:

Establece el directorio de trabajo dentro del contenedor en /app. Todas las instrucciones siguientes se ejecutarán en este directorio.

EXPOSE 80:

Esta instrucción documenta que el contenedor escuchará en el puerto 80. Sin embargo, no abre el puerto en el host, simplemente informa sobre el puerto en el que la aplicación dentro del contenedor debe escuchar.

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build:

Se inicia una nueva etapa en la construcción de la imagen utilizando otra imagen base, en este caso, la imagen SDK de .NET Core 7.0. Esto se hace para separar las etapas de compilación y ejecución de la aplicación.

WORKDIR /src:

Establece el directorio de trabajo dentro de esta nueva etapa en /src.

COPY ["MiProyectoWebAPI.csproj", "."]:

Copia el archivo de proyecto MiProyectoWebAPI.csproj desde el sistema de archivos del host al directorio de trabajo en el contenedor.

RUN dotnet restore "./MiProyectoWebAPI.csproj":

Ejecuta el comando dotnet restore para restaurar las dependencias del proyecto.

COPY . .:

Copia todos los archivos del directorio actual del host al directorio de trabajo en el contenedor. Esto incluye el código fuente de la aplicación.

WORKDIR "/src/":

Cambia el directorio de trabajo al directorio raíz del proyecto en el contenedor.

RUN dotnet build "MiProyectoWebAPI.csproj" -c Release -o /app/build:

Ejecuta el comando dotnet build para compilar la aplicación en modo Release y guarda la salida en el directorio /app/build en el contenedor.

FROM build AS publish:

Inicia una nueva etapa utilizando la misma imagen que la etapa de compilación (build) con el nombre publish.

RUN dotnet publish "MiProyectoWebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false:

Ejecuta el comando dotnet publish para publicar la aplicación en modo Release y guarda la salida en el directorio /app/publish en el contenedor. El argumento /p:UseAppHost=false es utilizado para evitar la creación de un host de aplicación en la publicación.

FROM base AS final:

Inicia una nueva etapa utilizando la imagen base nombrada como base.

WORKDIR /app:

Establece el directorio de trabajo en /app dentro de esta última etapa.

COPY --from=publish /app/publish .:

Copia los archivos publicados desde la etapa publish al directorio de trabajo en esta última etapa.

ENTRYPOINT ["dotnet", "MiProyectoWebAPI.dll"]:

Define el comando de entrada que se ejecutará cuando se inicie el contenedor. En este caso, ejecutará la aplicación ASP.NET Core contenida en el archivo MiProyectoWebAPI.dll.

Imagen para aplicación web en Nodejs

- Crear una 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"]
```

Este **Dockerfile** consta de dos etapas: una para la **construcción** de la aplicación y otra para la **producción**.

En la primera etapa (build), utilizamos la imagen base node:13.12.0-alpine para instalar las dependencias de Node.js y copiar los archivos de la aplicación.

En la segunda etapa (production), utilizamos la misma imagen base y copiamos solo los archivos necesarios desde la etapa de construcción (build). También exponemos el puerto 3000 y especificamos el comando para iniciar la aplicación.

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

docker build -t test-node .

```
PS C:\Users\sofia\OneDrive\Escritorio\IdS3\apptp6> docker build -t test-node "C:\Users\sofia\OneDrive\Escritorio\IdS3\apptp6"
[+] 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

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```

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

docker run -p 3000:3000 test-node

```

PS C:\Users\sofia\OneDrive\Escritorio\Id53\apptp6> 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 'setupMiddlewares' option.
(node:31) [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 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:

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 sofiacersofios/test-node:latest

- Subir la imagen a Docker Hub con el comando

docker push sofiacersofios/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.

```

PS C:\Users\sofia\OneDrive\Escritorio\Id53\apptp6> docker tag test-node sofiacersofios/test-node:latest
PS C:\Users\sofia\OneDrive\Escritorio\Id53\apptp6> docker push sofiacersofios/test-node:latest
The push refers to repository [docker.io/sofiacersofios/test-node]
83056ad0677a: Pushed
1308da6627db: Pushed
65d358b7de11: Mounted from library/node
f97384e8ccbc: Mounted from library/node
d56e5e720148: Mounted from library/node
beee9f30bc1f: Mounted from library/node
latest: digest: sha256:4e0c2c407f1ca99b5ca27a06502ae4245ccbd03c2becd8f37116af857fc363be size: 1577

```