Dockerizar una aplicación Angular

# Fuentes

**Create a new project**

<https://angular.io/tutorial/toh-pt0>

**Como Dockerizar una aplicación en Angular**

<https://youtu.be/CIKj9ftIMY8>

**Dockerize an Angular Application**

<https://jakecyr.medium.com/dockerize-an-angular-application-a45fbbf46de2>

**Docker with an Angular Application**

<https://youtu.be/MbA71IuYUhg>

**“Containerizing” Angular with Docker - Dan Wahlin**

<https://youtu.be/cLT7eUWKZpg>

Revisar este

<https://youtu.be/3NobJzI3_68>

# Resumen

Crear una aplicación Angular.

Dockerizarla.

# Preparación

Arrancar Docker Desktop.

Abrir en VS Code la carpeta 40-docker\angular-docker\angular-tour-of-heroes, que es donde está la aplicación. Ejecutar el comando:

npm ci

para limpiar y reinstalar de cero todos los paquetes. Arrancar la aplicación con:

ng serve --o

La aplicación aparecerá en el browser.

# angular-tour-of-heroes\Dockerfile

Crear y editar el archivo Dockerfile como sigue:

# Especificar la base image. No vamos a trabajar con una

# imagen que está completamente vacía, sino con una que

# ya tiene cosas que necesitamos.

# https://docs.docker.com/engine/reference/builder/#from

FROM node:alpine

# Especificar el working directory

# https://docs.docker.com/engine/reference/builder/#workdir

# La instrucción WORKDIR establece el directorio de

# trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT,

# COPY y ADD que le sigue en el Dockerfile. Si WORKDIR

# no existe, se creará incluso si no se usa en

# ninguna instrucción posterior de Dockerfile.

# Este el directorio en donde todos nuestros archivos van a estar

# y desde donde va a correr nuestro container. O sea, tal como

# se dijo antes, los comandos COPY y RUN que vienen después

# van a estar referidos a WORKDIR.

# https://docs.docker.com/engine/reference/builder/#workdir

# El directorio es del file system del container, no

# de la PC local. La documentación no lo dice directamente,

# sino que hay que deducirlo pensando sobre los

# ejemplos. En el caso de la siguiente instrucción, se

# trata de un path absoluto, porque empieza con /,

# y ese directorio habrá que buscarlo después en el

# file system del container, una vez que esté corriendo.

WORKDIR /usr/src/app

# https://docs.docker.com/engine/reference/builder/#copy

# COPY <src> <dest> copia archivos o directorios

# desde <src> y los agrega al file system del

# container en el path <dest>.

# El punto (.) como segundo parámetro, o destination, es el working directory.

# Copiamos primero este file, porque Docker trabaja en layers,

# y de este modo si luego cambiamos otro file que no sea este,

# no necesitaremos correr de nuevo npm install, que es time consuming.

# El resto de los files los copiamos más adelante, y de ese modo los

# podemos cambiar sin incurrir en ese overhead.

# Primero copia los archivos y después corre 'npm install'.

# De ese modo, los node\_modules estarán cacheados hasta

# que package.json o package-lock.json cambie.

#

# https://www.geeksforgeeks.org/difference-between-package-json-and-package-lock-json-files/

#

# En Node.js, package.json es un archivo de control

# de versiones que se usa para instalar varios

# paquetes en su proyecto. A medida que inicializa

# su aplicación de Node, verá tres archivos

# instalados en su aplicación que son node\_modules,

# package.json y package.lock.json.

#

# package.json contiene metadatos sobre el proyecto

# y también las dependencias funcionales que

# requiere la aplicación.

#

# package-lock.json se crea para bloquear la

# dependencia con la versión instalada. Instalará

# la última versión exacta de ese paquete en su

# aplicación y la guardará en package.json. Digamos

# que si la versión actual del paquete es 1.3.2,

# guardará la versión con el signo (^). Aquí caret(^)

# significa que admitirá cualquier versión superior

# con la versión principal 1, por ejemplo. 1.2.2.

#

# Sin package-lock.json, puede haber algunas

# diferencias en las versiones instaladas en

# diferentes entornos. Para superar este problema,

# se crea package-lock.json para obtener los mismos

# resultados en todos los entornos. Debe estar en el

# control de código fuente con el archivo package.json

# porque si cualquier otro usuario clona el proyecto

# e instala las dependencias, instalará exactamente

# las mismas dependencias que en el

# package-lock.json para evitar diferencias.

#

COPY package.json .

COPY package-lock.json .

# https://docs.docker.com/engine/reference/builder/#run

# La instrucción RUN ejecuta los comandos que se le indican

# en una nueva capa superpuesta a los anteriores resultados

# y confirma todo.

# La imagen resultante confirmada será usada en el

# siguiente paso en Dockerfile.

# Ahora instalamos todas las dependencias establecidas

# en package.json. Esto es lo que toma más tiempo, salvo

# que estén cacheadas.

RUN npm install

# copy over all code files

# Copio todo lo que haya en este directorio, menos lo que ya

# he copiado antes.

COPY . .

# https://docs.docker.com/engine/reference/builder/#expose

# La instrucción EXPOSE informa a Docker que el container

# atiende a los puertos especificados del container en

# runtime.

# Uno puede especificar si el puerto atiende on TCP

# o UDP. Si no se especifica, el default es TCP.

#

# You can specify whether the port listens on TCP or UDP,

# and the default is TCP if the protocol is not specified.

EXPOSE 4200

# https://docs.docker.com/engine/reference/builder/#cmd

# Ejecuta el comando ng serve

# specify default command to run when we run the image

# https://angular.io/cli/serve

# --host                  Host to listen on.              string    localhost

# --disable-host-check  Don't verify connected clients  boolean false

#                       are part of allowed hosts.

CMD /usr/src/app/node\_modules/.bin/ng serve --host 0.0.0.0

#  --disableHostCheck

# El parámetro anterior lo comenté porque da error.

# Crear la imagen de la aplicación

Abrimos una terminal de Windows en la carpeta angular-docker\angular-tour-of-heroes. Pegamos el comando:

docker build -t myapp -f Dockerfile .

* -t especifica el tag name de la imagen que está creando.
* -f especifica la ubicación del Dockerfile que se usará para generar la imagen.
* . al final del comando anterior especifica el contexto para enviar al docker builder. Por ejemplo, si tuviera su código en una carpeta code, podría decir:

docker build -t myap -f Dockerfile code

Podemos verificar en la extensión de Docker de VS Code que efectivamente se ha creado la imagen.

# Arrancar el container

Corremos la imagen con el comando:

docker run --rm -it -p 4200:4200 myapp

Podemos verificar en la extensión de Docker de VS Code que efectivamente el container está corriendo. Si navegamos hasta /usr/src/app en el file system del container, veremos que allí están los archivos que copiamos.

Vamos en el browser a la URL <http://localhost:4200/home> y allí está nuestr aplicación. Notar que no la arrancamos con ng serve, como antes. Está corriendo dentro del container.