

# Construye y corre un contenedor con un Dockerfile

## • Crear el contenedor ▾

Se creó primero una app sencilla con Node.js usando el modulo de **Express**

```
npm init
npm install -save express
```

En la raíz se creó un archivo con el servidor **main.js** y se configuró la ruta en el package.json para que se ejecutara este archivo al escribir en consola

```
npm start
```

```
'use strict';
const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(PORT, HOST, () => {
  console.log(`Running on http://${HOST}:${PORT}`);
});
```

La app de Node.js ya está terminada

## • Creando Dockerfile ▾

En la misma carpeta de la app se creó el archivo Dockerfile, en ella se especificó las características del contenedor

```
#Especifica la imagen a usar como base, Node 16
FROM node:16
#Especifica la carpeta de trabajo dentro del contenedor
WORKDIR /usr/src/app
#Copia los ficheros de origen en destino. Copia solo el package.json y package-lock.json
COPY package*.json ./
#Ejecuta el comando en el momento de la creación de la imagen
RUN npm install
#Para agrupar el código adentro de la imagen de Docker
COPY . .
#Informamos que el puerto que usa es el 8080
EXPOSE 8080
#Ejecuta el comando al ejecutar el contenedor. Inicia el archivo main.js
CMD [ "node", "main.js" ]
```

- .dockerignore

Se crea el .dockerignore, que evita que ciertos archivos especificados se copien en la imagen de Docker

```
node_modules
npm-debug.log
```

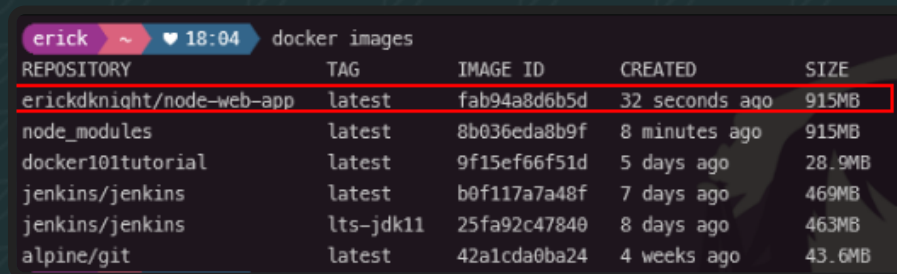
## • Construir y ejecutar la imagen

En el CMD en la ubicación donde se encuentra el **Dockerfile** se ejecuta el comando

```
docker build . -t nombreDeImagen
```

Podemos ver que la imagen ya esta instalada con:

```
docker images
```



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
erickdknight/node-web-app	latest	fab94a8d6b5d	32 seconds ago	915MB
node_modules	latest	8b036eda8b9f	8 minutes ago	915MB
docker101tutorial	latest	9f15ef66f51d	5 days ago	28.9MB
jenkins/jenkins	latest	b0f117a7a48f	7 days ago	469MB
jenkins/jenkins	lts-jdk11	25fa92c47840	8 days ago	463MB
alpine/git	latest	42a1cda0ba24	4 weeks ago	43.6MB

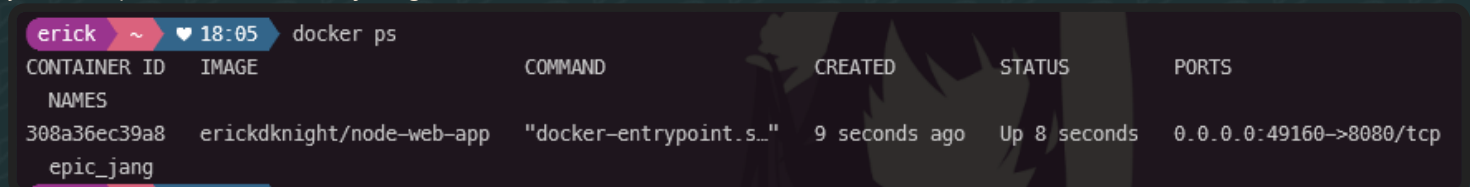
Para ejecutar la imagen, se escribe el comando

```
docker run -p 49160:8080 -d nombreDeImagen
```

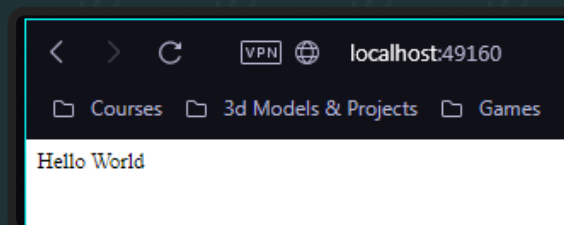
Podemos ver el contenedor activado con el comando

```
docker ps
```

y tambien podemos ver el mensaje asignado en el localhost:49160



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
308a36ec39a8	erickdknight/node-web-app	"docker-entrypoint.s..."	9 seconds ago	Up 8 seconds	0.0.0.0:49160->8080/tcp



Con la función **curl**, redirigimos la respuesta del localhost al cmd, pudiendo ver la respuesta del servidor en consola

```
erick ~ ♥ 18:07 curl -i localhost:49160
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 11
ETag: W/"b-Ck1VqNd45QIvq3AZd8XYQLvEhtA"
Date: Wed, 23 Nov 2022 00:08:37 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

```
Hello World erick ~ ♥ 18:08 docker kill 308a36ec39a8
```

Con docker kill cerramos el contenedor

```
docker kill CONTAINER_ID
```