

Tarea 17: Dockerización de una app con Node.js

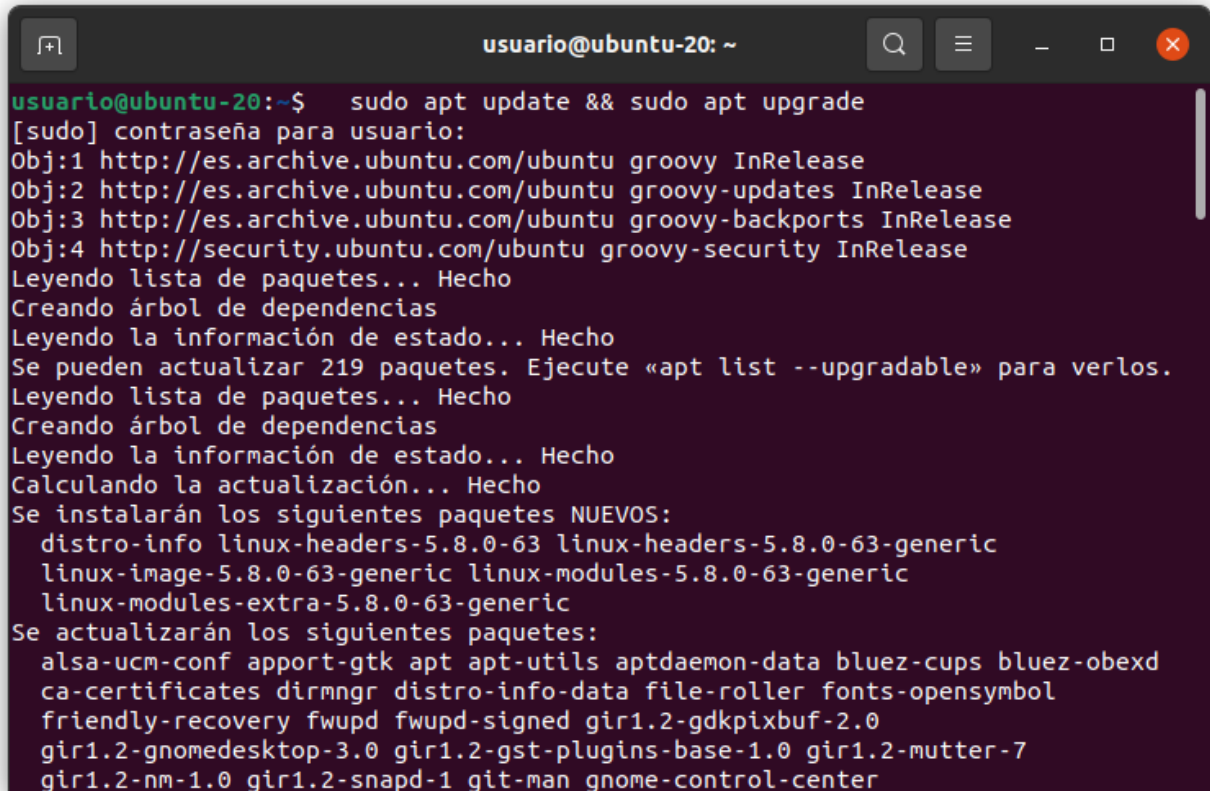


Índice

1. Actualización de repositorios
2. Creación de una app node
3. Creación del Dockerfile
4. Creación del archivo .dockerignore
5. Construcción de la imagen
6. Lanzamiento de la imagen

1. Actualización de repositorios

Primero vamos a actualizar el repositorio y el sistema operativo para que la instalación de apache sea mas segura.

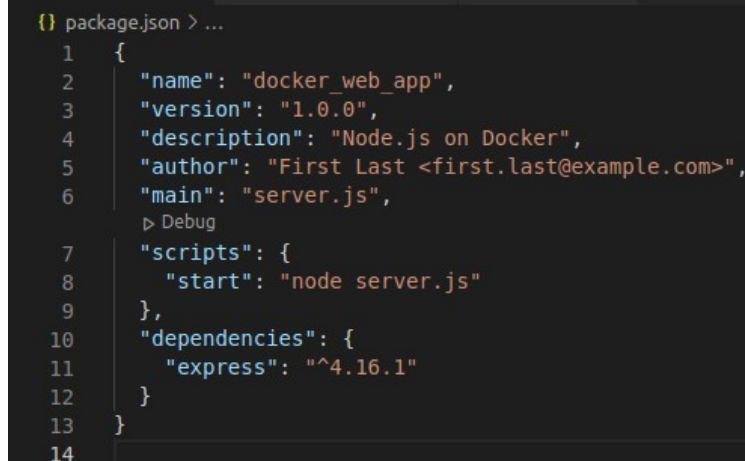


```
usuario@ubuntu-20: ~  
usuario@ubuntu-20:~$ sudo apt update && sudo apt upgrade  
[sudo] contraseña para usuario:  
Obj:1 http://es.archive.ubuntu.com/ubuntu groovy InRelease  
Obj:2 http://es.archive.ubuntu.com/ubuntu groovy-updates InRelease  
Obj:3 http://es.archive.ubuntu.com/ubuntu groovy-backports InRelease  
Obj:4 http://security.ubuntu.com/ubuntu groovy-security InRelease  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se pueden actualizar 219 paquetes. Ejecute «apt list --upgradable» para verlos.  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Calculando la actualización... Hecho  
Se instalarán los siguientes paquetes NUEVOS:  
  distro-info linux-headers-5.8.0-63 linux-headers-5.8.0-63-generic  
  linux-image-5.8.0-63-generic linux-modules-5.8.0-63-generic  
  linux-modules-extra-5.8.0-63-generic  
Se actualizarán los siguientes paquetes:  
  alsa-ucm-conf apport-gtk apt apt-utils aptdaemon-data bluez-cups bluez-obexd  
  ca-certificates dirmngr distro-info-data file-roller fonts-opensymbol  
  friendly-recovery fwupd fwupd-signed gir1.2-gdkpixbuf-2.0  
  gir1.2-gnomedesktop-3.0 gir1.2-gst-plugins-base-1.0 gir1.2-mutter-7  
  gir1.2-nm-1.0 gir1.2-snapd-1 git-man gnome-control-center
```

2. Creacion de una app node

Ahora vamos a crear nuestra app con node.

Primero creamos el archivo .json



```
{  
  "name": "docker_web_app",  
  "version": "1.0.0",  
  "description": "Node.js on Docker",  
  "author": "First Last <first.last@example.com>",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js"  
  },  
  "dependencies": {  
    "express": "^4.16.1"  
  }  
}
```

Una vez creado el archivo .json tendremos que realizar un npm install

```
usuario@ubuntu-20:~/Escritorio/nodem$ npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN docker_web_app@1.0.0 No repository field.
npm WARN docker_web_app@1.0.0 No license field.
```

added 50 packages from 37 contributors and audited 50 packages in 7.083s

Una vez terminado el proceso se habra creado un archivo package-lock.json y otros ademas de este.

```
{ package.json Dockerfile package-lock.json x JS server.js
package-lock.json > ...
1 {
2   "name": "docker_web_app",
3   "version": "1.0.0",
4   "lockfileVersion": 1,
5   "requires": true,
6   "dependencies": {
7     "accepts": {
8       "version": "1.3.7",
9       "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
10      "integrity": "sha512-Il80Qs2WjYlJIBNzNkK6KYqlVMTbZLXgHx2oT0pU/fjRHyeP+PEfEPY0R3WCwAGV0tauxh1h0xNgIf5bv7dQpA==",
11      "requires": {
12        "mime-types": "~2.1.24",
13        "negotiator": "0.6.2"
14      }
15    },
16    "array-flatten": {
17      "version": "1.1.1",
18      "resolved": "https://registry.npmjs.org/array-flatten/-/array-flatten-1.1.1.tgz",
19      "integrity": "sha1-m19pkFGx5wcZKPKgCJaLZ0opVdI="
20    },
21    "body-parser": {
22      "version": "1.19.0",
23      "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-1.19.0.tgz",
24      "integrity": "sha512-dhEPs72UPbDnAQJ9ZKMNTp6ptJaionhP5cBb541nXPLW60Jepo9RV/a4fX4XWW9CuFNK22krhjl+rgzifNCsw==",
25      "requires": {
26        "bytes": "3.1.0",
27        "content-type": "~1.0.4",
28        "debug": "2.6.9",
29        "depd": "~1.1.2",
30        "http-errors": "1.7.2",
31        "iconv-lite": "0.4.24",
32        "on-finished": "~2.3.0",
33        "qs": "6.7.0",
34        "raw-body": "2.4.0",
35        "type-is": "~1.6.17"
36      }
37    }
38  }
39 }
```

Luego creamos un fichero server.js y le añadimos el siguiente contenido.

```
{ package.json Dockerfile package-lock.json JS server.js x
JS server.js > [?] PORT
1  'use strict';
2
3  const express = require('express');
4
5  // Constants
6  const PORT = 8085;
7  const HOST = '0.0.0.0';
8
9  // App
10 const app = express();
11 app.get('/', (req, res) => {
12   res.send('Hello World');
13 });
14
15 app.listen(PORT, HOST);
16 console.log(`Running on http://${HOST}:${PORT}`);
17
```

3. Creación del Dockerfile

Ahora crearemos un Dockerfile con el siguiente contenido (Nota: el puerto realmente es 8085 en vez de 8080, lo modifique porque en mi caso estaba ocupado el puerto 8080).

```

package.json  Dockerfile x  package-lock.json  JS server.js

Dockerfile
1  FROM node:16
2
3  # Create app directory
4  WORKDIR /usr/src/app
5
6  # Install app dependencies
7  # A wildcard is used to ensure both package.json AND package-lock.json are copied
8  # where available (npm@5+)
9  COPY package*.json ./
10
11  RUN npm install
12  # If you are building your code for production
13  # RUN npm ci --only=production
14
15  # Bundle app source
16  COPY . .
17
18  EXPOSE 8080
19  CMD [ "node", "server.js" ]
20

```

4. Creación del archivo .dockerignore

Ahora vamos a crear nuestro archivo .dockerignore con el siguiente contenido

```

package.json  Dockerfile  package.dockerignore x

package.dockerignore
1  node_modules
2  npm-debug.log
3

```

5. Construcción de la imagen

A continuación, vamos a construir nuestra imagen docker con el siguiente comando

```

usuario@ubuntu-20:~/Escritorio/nodem$ sudo docker build . -t nodem
[sudo] contraseña para usuario:
Sending build context to Docker daemon  2.018MB
Step 1/7 : FROM node:16
16: Pulling from library/node
07471e81507f: Pulling fs layer
07471e81507f: Downloading [=====>] 37.33MB/50.44MB
13a51f13be8e: Download complete

```

Y ahora listamos la imagen

```
usuario@ubuntu-20:~/Escritorio/nodem$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nodem         latest    032c902ceb27   37 seconds ago 911MB
```

6. Lanzamiento de la imagen

Ahora que ya tenemos construida la imagen podremos lanzar la app con la imagen docker. Para ello ejecutaremos el siguiente comando donde indicaremos el puerto donde se ejecutara nuestra app y el nombre que tendrá.

```
usuario@ubuntu-20:~/Escritorio/nodem$ sudo docker run -p 49160:8085 -d nodem
5a851bd4df1e222d4869152d4101f068ed30cf9c6c469f897569fd63e2d9c8a6
```

Una vez lanzado comprobamos que al dirigiarnos a la siguiente direccion web se ejecutara la app.

← → ↻ localhost:49160

Hello World

Y con el comando docker ps vemos que se esta ejecutando en el puerto 8085, sin embargo si utilizamos esa dirección pues no se puede acceder en cambio si utilizamos 49160 si podemos acceder.

```
usuario@ubuntu-20:~/Escritorio/nodem$ sudo docker run -p 49160:8085 -d nodem
5a851bd4df1e222d4869152d4101f068ed30cf9c6c469f897569fd63e2d9c8a6
usuario@ubuntu-20:~/Escritorio/nodem$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
5a851bd4df1e   nodem     "docker-entrypoint.s..." 4 minutes ago Up 4 minutes   0.0.0.0:49160->8085/tcp, :::49160->8085/tcp   quirky_brattain
```

← → ↻ localhost:8085

No se puede conectar

Firefox no puede establecer una conexión con el servidor en localhost:8085.

- El sitio podría estar no disponible temporalmente o demasiado ocupado. Vuelva a intentarlo en unos momentos.
- Si no puede cargar ninguna página, compruebe la conexión de red de su equipo.
- Si su equipo o red están protegidos por un cortafuegos o proxy, asegúrese de que Firefox tiene permiso para acceder a la web.

Reintentar