DESPLIEGUE DE APLICACIONES WEB

DESPLIEGUE DE APLICACIONES CON NODE EXPRESS

AUTOR: Alberto Martínez Pérez

CURSO: 2º CFGS Desarrollo de Aplicaciones Web (DAW)

MÓDULO: Despliegue de Aplicaciones Web

ÍNDICE

INTRODUCCIÓN

En esta práctica vamos a realizar un despliegue de aplicaciones Node.js sobre un servidor Node Express. En este caso el despliegue cambia respecto a lo visto hasta ahora en prácticas anteriores ya que no se realiza sobre un servidor, sino que la aplicación es en sí el servidor.

A. NODE.JS

Node.js es un entorno de ejecución de JavaScript que se utiliza para construir aplicaciones de backend. Por sí mismo no sabe cómo debe servir archivos o manejar peticiones HTTP de ahí que necesite de la ayuda de Express.

Hay que tener en cuenta que Node.js no es ni un lenguaje de programación ni un framework sino que es un entorno de ejecución que se utiliza para ejecutar JavaScript fuera del navegador. Además, su tiempo de ejecución se construye sobre el propio JavaScript y ayuda en la ejecución de los frameworks.

B. EXPRESS.JS

Express.js es un framework de Node.js diseñado para construir aplicaciones web de API's y aplicaciones móviles multiplataforma de forma rápida.

Con él se consigue que el uso de Node.js sea más sencillo.

C. NPM (Node Package Master)

NPM es la herramienta por defecto de JavaScript para la tarea de compartir e instalar paquetes.

Se compone de dos partes:

1) Un repositorio online donde se publican paquetes de software libre para ser utilizados en proyectos con Node.js.
2) Una herramienta de terminal con la que se interactúa con el repositorio para instalar utilidades, dependencias e incluso realizar publicaciones de paquetes.

Por tanto, NPM es un gestor de paquetes de JavaScript similar a lo que es Maven para Java permitiendo instalar y gestionar versiones de paquetes y librerías .js.

a. *package.json*

Cada proyecto en JavaScript puede enfocarse como un paquete npm con su propia información de paquete y su archivo package.json para describir el proyecto.

Este archivo .json se generará cuando se ejecute npm init para inicializar un proyecto JavaScript/Node.js, con los siguientes metadatos básicos proporcionados por los desarrolladores:

- name: el nombre de la librería/proyecto JavaScript.
- version: la versión del proyecto.
- description: la descripción del proyecto.
- license: la licencia del proyecto.

b. *NPM scripts*

package.json también soporta la propiedad scripts que puede definirse para ejecutar herramientas de línea de comandos que se instalan en el contexto local del proyecto.

D. REQUISITOS PARA LA PRÁCTICA

En esta práctica vamos a utilizar una nueva máquina virtual con el sistema operativo Debian 12 instalado desde cero para impedir posibles interferencias con los servidores NGINX y Tomcat instalados en las prácticas anteriores[1].

Lo que se va a realizar es una instalación de Node.js y de Express, así como el despliegue de dos aplicaciones una básica que crearemos paso a paso y otra más compleja que clonaremos de un repositorio de GitHub.

---

[1] Si se quisiera utilizar la máquina de la práctica anterior simplemente habría que parar los servicios NGINX y Tomcat con los comandos *sudo systemctl stop nginx.service* y *sudo systemctl stop tomcat.service* respectivamente.
Alternativamente a esto podríamos decidir desinstalar los paquetes para ello deberemos usar los siguientes comandos:

1. *sudo apt purge nginx nginx-common tomcat:* para desinstalar y eliminar completamente los paquetes especificados y sus archivos de configuración.
2. *sudo apt autoclean:* para limpiar la caché de archivos .deb que ya no se encuentran disponibles en los repositorios.
3. *sudo apt autoremove:* para eliminar los paquetes que fueron instalados como dependencias de otros paquetes, pero que ya no son necesarios debido a que los paquetes que los necesitaban han sido desinstalados.

También se comprobará el funcionamiento de estas aplicaciones haciendo una conexión anfitrión - máquina virtual por lo que el tipo red que tengan nuestra máquina virtual debe ser uno que permita este tipo de conexión, por ejemplo, adaptador puente o red NAT.

INSTALACIÓN DE NODE.JS

Para poder utilizar Express.js primero necesitamos instalar Node.JS para ello vamos a comprobar si existen actualizaciones de Debian o no:

sudo apt-get update

```
albertom-servidor@debian-deaw:~$ sudo apt-get update
Des:1 http://security.debian.org/debian-security bookworm-secu
Des:2 http://deb.debian.org/debian bookworm InRelease [151 kB]
Des:3 http://deb.debian.org/debian bookworm-updates InRelease
Des:4 http://security.debian.org/debian-security bookworm-secu
Des:5 http://security.debian.org/debian-security bookworm-secu
Des:6 http://security.debian.org/debian-security bookworm-secu
Des:7 http://security.debian.org/debian-security bookworm-secu
Des:8 http://security.debian.org/debian-security bookworm-secu
```

sudo apt-get upgrade

```
albertom-servidor@debian-deaw:~$ sudo apt-get upgrade
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Calculando la actualización... Hecho
Los siguientes paquetes se han retenido:
  linux-headers-amd64 linux-image-amd64
Se actualizarán los siguientes paquetes:
  base-files cups cups-client cups-common cups-core-dr
  cups-server-common distro-info-data exfatprogs ffmpe
```

También comprobaremos la versión de nuestro Debian:

sudo cat /etc/debian_version

```
albertom-servidor@debian-deaw:~$ sudo cat /etc/debian_version
12.3
```

En este caso estamos trabajando sobre un Debian en su versión 12.3.

Una vez hecho esto vamos a instalar Node.js en nuestra máquina para ello ejecutamos el siguiente comando:

sudo apt-get install nodejs

```
albertom-servidor@debian-deaw:~$ sudo apt-get install nodejs
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libc-ares2 libnode108 node-acorn node-busboy node-cjs-modul
Paquetes sugeridos:
  npm
Se instalarán los siguientes paquetes NUEVOS:
  libc-ares2 libnode108 node-acorn node-busboy node-cjs-modul
  nodejs-doc
0 actualizados, 9 nuevos se instalarán, 0 para eliminar y 2 r
Se necesita descargar 14,5 MB de archivos.
Se utilizarán 67,9 MB de espacio de disco adicional después d
¿Desea continuar? [S/n]
```

Una vez finalizada la instalación vamos a comprobar la versión de Node.js que se ha instalado:

node --version

```
albertom-servidor@debian-deaw:~$ node --version
v18.13.0
```

INSTALACIÓN DE npm

Para instalar npm en nuestra máquina virtual ejecutamos el comando:

sudo apt-get install npm

```
albertom-servidor@debian-deaw:~$ sudo apt-get install npm
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  eslint gyp handlebars libjs-async libjs-events libjs-inhe
  libjs-regenerate libjs-source-map libjs-sprintf-js libjs-
  libssl-dev libuv1-dev node-abbrev node-agent-base node-aj
  node-ampproject-remapping node-ansi-escapes node-ansi-reg
  node-aproba node-archy node-are-we-there-yet node-argpars
  node-async-each node-auto-bind node-babel-helper-define-p
  node-babel-plugin-add-module-exports node-babel-plugin-lo
  node-babel-plugin-polyfill-corejs3 node-babel-plugin-poly
  node-babel7-runtime node-balanced-match node-base node-ba
  node-brace-expansion node-braces node-browserslist node-b
```

Y, al igual que con Node.js, una vez que se haya completado la instalación, comprobamos la versión de npm:

npm --version

```
albertom-servidor@debian-deaw:~$ npm --version
9.2.0
```

INSTALACIÓN DE EXPRESS.JS

Para instalar Express en nuestro sistema de forma global (opción -g) debemos usar npm, para ello utilizamos el siguiente comando:

sudo npm install -g express

```
albertom-servidor@debian-deaw:~$ sudo npm install -g express

added 62 packages in 5s

11 packages are looking for funding
  run `npm fund` for details
```

CREACIÓN DE UNA APLICACIÓN .JS DE PRUEBA

Una vez hechos todos los pasos anteriores vamos a crear una aplicación JavaScript que nos sirva como prueba de despliegue para vez que todo funciona de forma correcta, para ello lo primero que vamos a hacer es crear un directorio, en nuestro caso lo vamos a crear en el directorio home/ del usuario principal del sistema:

mkdir ./proyectoNJS

```
albertom-servidor@debian-deaw:~$ mkdir ./proyectoNJS
albertom-servidor@debian-deaw:~$ ls -la
total 132
drwx------ 18 albertom-servidor albertom-servidor 4096 dic 10 08:06 .
drwxr-xr-x  3 root              root              4096 nov  1 11:15 ..
-rw-------  1 albertom-servidor albertom-servidor  494 dic 10 07:56 .bash_history
-rw-r--r--  1 albertom-servidor albertom-servidor  220 nov  1 11:15 .bash_logout
-rw-r--r--  1 albertom-servidor albertom-servidor 3526 nov  1 11:15 .bashrc
drwx------ 11 albertom-servidor albertom-servidor 4096 dic 10 07:53 .cache
drwxr-xr-x 12 albertom-servidor albertom-servidor 4096 nov  1 11:30 .config
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Descargas
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Documentos
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Escritorio
-rw-r--r--  1 albertom-servidor albertom-servidor 5290 nov  1 11:15 .face
lrwxrwxrwx  1 albertom-servidor albertom-servidor    5 nov  1 11:15 .face.icon -> .face
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 dic 10 07:53 .fontconfig
drwx------  2 albertom-servidor albertom-servidor 4096 nov  1 11:27 .gnupg
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Imágenes
drwx------  4 albertom-servidor albertom-servidor 4096 nov  1 11:19 .local
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Música
drwxr-xr-x  3 albertom-servidor albertom-servidor 4096 dic 10 08:05 .npm
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Plantillas
-rw-r--r--  1 albertom-servidor albertom-servidor  807 nov  1 11:15 .profile
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 dic 10 08:06 proyectoNJS
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Público
drwx------  2 albertom-servidor albertom-servidor 4096 nov  1 11:27 .ssh
```

Dentro de este directorio debemos inicializar el proyecto ejecutando el siguiente comando:

npm init -y

```
albertom-servidor@debian-deaw:~$ cd ./proyectoNJS/
albertom-servidor@debian-deaw:~/proyectoNJS$ npm init -y
Wrote to /home/albertom-servidor/proyectoNJS/package.json:

{
  "name": "proyectonjs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Esto nos habrá creado el archivo package.json con la información que vemos en la imagen de arriba.

El siguiente paso será instalar express de forma local para este proyecto:

sudo npm install express

```
albertom-servidor@debian-deaw:~/proyectoNJS$ sudo npm install express

added 62 packages, and audited 63 packages in 1s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Esto ha creado dentro de nuestro proyecto tanto el archivo package-lock.json (que guarda la última versión del archivo package.json) como un directorio node_modules (que contiene los archivos de las bibliotecas y sus dependencias necesarias para que el proyecto funcione).

```
albertom-servidor@debian-deaw:~/proyectoNJS$ ls -la
total 40
drwxr-xr-x  3 albertom-servidor albertom-servidor  4096 dic 10 08:08 .
drwx------ 18 albertom-servidor albertom-servidor  4096 dic 10 08:06 ..
drwxr-xr-x 64 root              root               4096 dic 10 08:08 node_modules
-rw-r--r--  1 albertom-servidor albertom-servidor   275 dic 10 08:08 package.json
-rw-r--r--  1 root              root              24286 dic 10 08:08 package-lock.json
```

Ahora vamos a crear un archivo JavaScript sencillo que nos sirva de prueba:

sudo nano app.js

```
albertom-servidor@debian-deaw:~/proyectoNJS$ sudo nano app.js
```

```
 GNU nano 7.2                                    app.js *
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('Hola, bienvenido al blog de Alberto Martínez');
});

app.listen(port, () => {
    console.log(`Ejemplo de app escuchando por http://localhost:${port}`);
});
```

Para lanzar el servidor de nuestra aplicación utilizamos el comando:

sudo node app.js

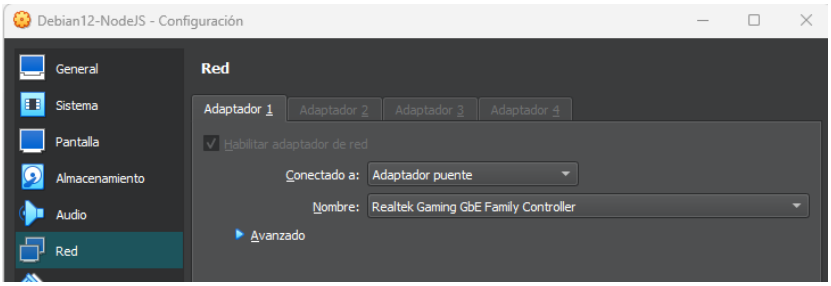
```
albertom-servidor@debian-deaw:~/proyectoNJS$ sudo node app.js
Ejemplo de app escuchando por http://localhost:3000
```

El mensaje que vemos es el correspondiente al método listen() del archivo app.js.

A. CONEXIÓN A NUESTRA APLICACIÓN DESDE LA MÁQUINA LOCAL

Para comprobar el funcionamiento de la aplicación desde la máquina local debemos conectarnos a la dirección IP de nuestra máquina virtual y al puerto 3000.

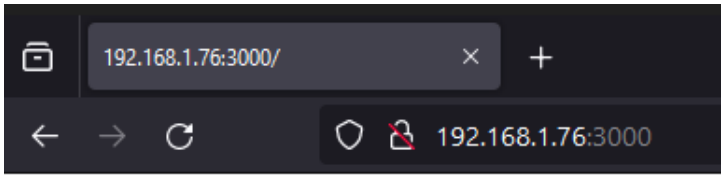
Para ello como se comentó en la introducción la máquina debe tener un tipo de red que permita su visión desde fuera, en nuestro caso hemos decidido utilizar una red de tipo "Adaptador puente":



Para obtener la ip se utiliza el comando ip a como hemos hecho en otras prácticas:

```
albertom-servidor@debian-deaw:~/proyectoNJS$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue stat
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1
 1000
    link/ether 08:00:27:76:a3:3b brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.76/24 brd 192.168.1.255 scope global en
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe76:a33b/64 scope link
       valid_lft forever preferred_lft forever
```

Si entramos desde el navegador del equipo anfitrión, veremos el siguiente mensaje:



Hola, bienvenido al blog de Alberto Martínez

El mensaje que vemos es el correspondiente al método get() del archivo app.js.

Por último, paramos la ejecución del servidor usando la combinación Ctrl + C:

```
albertom-servidor@debian-deaw:~/proyectoNJS$ sudo node app.js
Ejemplo de app escuchando por http://localhost:3000
^C
albertom-servidor@debian-deaw:~/proyectoNJS$
```

DESPLIEGUE DE UNA APLICACIÓN DE TERCEROS

Ahora vamos a desplegar una aplicación de terceros, concretamente la que se encuentran en [este repositorio de GitHub](https://github.com/contentful/the-example-app.nodejs.git).

Lo primero que debemos hacer es una clonación del repo, en nuestro caso, lo hacemos en el directorio home/ del usuario:

git clone https://github.com/contentful/the-example-app.nodejs.git

```
albertom-servidor@debian-deaw:~$ git clone https://github.com/contentful/the-example-app.nodejs.git
Clonando en 'the-example-app.nodejs'...
remote: Enumerating objects: 2935, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 2935 (delta 12), reused 2 (delta 0), pack-reused 2914
Recibiendo objetos: 100% (2935/2935), 6.81 MiB | 9.26 MiB/s, listo.
Resolviendo deltas: 100% (1770/1770), listo.
```

Esto nos ha creado un directorio en nuestro home/ con el nombre the-example-app.nodejs:

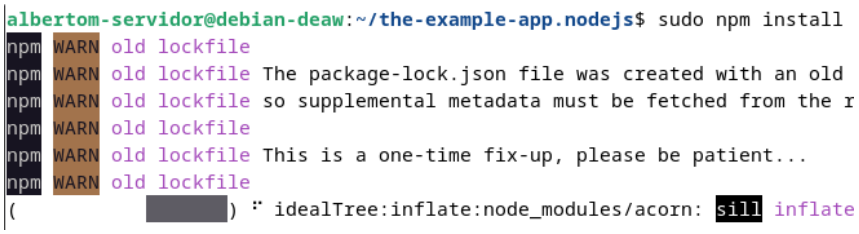
```
albertom-servidor@debian-deaw:~$ ls -la
total 136
drwx------ 19 albertom-servidor albertom-servidor 4096 dic 10 08:21 .
drwxr-xr-x  3 root              root              4096 nov  1 11:15 ..
-rw-------  1 albertom-servidor albertom-servidor  521 dic 10 08:20 .bash_history
-rw-r--r--  1 albertom-servidor albertom-servidor  220 nov  1 11:15 .bash_logout
-rw-r--r--  1 albertom-servidor albertom-servidor 3526 nov  1 11:15 .bashrc
drwx------ 11 albertom-servidor albertom-servidor 4096 dic 10 07:53 .cache
drwxr-xr-x 12 albertom-servidor albertom-servidor 4096 nov  1 11:30 .config
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Descargas
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Documentos
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Escritorio
-rw-r--r--  1 albertom-servidor albertom-servidor 5290 nov  1 11:15 .face
lrwxrwxrwx  1 albertom-servidor albertom-servidor    5 nov  1 11:15 .face.icon -> .face
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 dic 10 07:53 .fontconfig
drwx------  2 albertom-servidor albertom-servidor 4096 nov  1 11:27 .gnupg
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Imágenes
drwx------  4 albertom-servidor albertom-servidor 4096 nov  1 11:19 .local
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Música
drwxr-xr-x  3 albertom-servidor albertom-servidor 4096 dic 10 08:05 .npm
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Plantillas
-rw-r--r--  1 albertom-servidor albertom-servidor  807 nov  1 11:15 .profile
drwxr-xr-x  3 albertom-servidor albertom-servidor 4096 dic 10 08:14 proyectoNJS
drwxr-xr-x  2 albertom-servidor albertom-servidor 4096 nov  1 11:19 Público
drwx------  2 albertom-servidor albertom-servidor 4096 nov  1 11:27 .ssh
drwxr-xr-x 13 albertom-servidor albertom-servidor 4096 dic 10 08:21 the-example-app.nodejs
```

Cuyo contenido es el siguiente:

```
albertom-servidor@debian-deaw:~$ ls -la ./the-example-app.nodejs/
total 416
drwxr-xr-x 13 albertom-servidor albertom-servidor   4096 dic 10 08:21 .
drwx------ 19 albertom-servidor albertom-servidor   4096 dic 10 08:21 ..
-rw-r--r--  1 albertom-servidor albertom-servidor   5362 dic 10 08:21 app.js
-rw-r--r--  1 albertom-servidor albertom-servidor    416 dic 10 08:21 app.json
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 bin
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 .circleci
-rw-r--r--  1 albertom-servidor albertom-servidor    135 dic 10 08:21 cypress.json
-rw-r--r--  1 albertom-servidor albertom-servidor    164 dic 10 08:21 Dockerfile
-rw-r--r--  1 albertom-servidor albertom-servidor     27 dic 10 08:21 .dockerignore
-rw-r--r--  1 albertom-servidor albertom-servidor    256 dic 10 08:21 .eslintrc.js
drwxr-xr-x  8 albertom-servidor albertom-servidor   4096 dic 10 08:21 .git
-rw-r--r--  1 albertom-servidor albertom-servidor   3178 dic 10 08:21 .gitignore
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 handlers
-rw-r--r--  1 albertom-servidor albertom-servidor   2326 dic 10 08:21 helpers.js
drwxr-xr-x  3 albertom-servidor albertom-servidor   4096 dic 10 08:21 i18n
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 lib
-rw-r--r--  1 albertom-servidor albertom-servidor   1067 dic 10 08:21 LICENSE
-rw-r--r--  1 albertom-servidor albertom-servidor      6 dic 10 08:21 .nvmrc
-rw-r--r--  1 albertom-servidor albertom-servidor   1619 dic 10 08:21 package.json
-rw-r--r--  1 albertom-servidor albertom-servidor 307556 dic 10 08:21 package-lock.json
drwxr-xr-x  7 albertom-servidor albertom-servidor   4096 dic 10 08:21 public
-rw-r--r--  1 albertom-servidor albertom-servidor   4814 dic 10 08:21 README.md
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 routes
drwxr-xr-x  2 albertom-servidor albertom-servidor   4096 dic 10 08:21 services
drwxr-xr-x  4 albertom-servidor albertom-servidor   4096 dic 10 08:21 test
-rw-r--r--  1 albertom-servidor albertom-servidor    344 dic 10 08:21 variables.env
drwxr-xr-x  3 albertom-servidor albertom-servidor   4096 dic 10 08:21 views
```

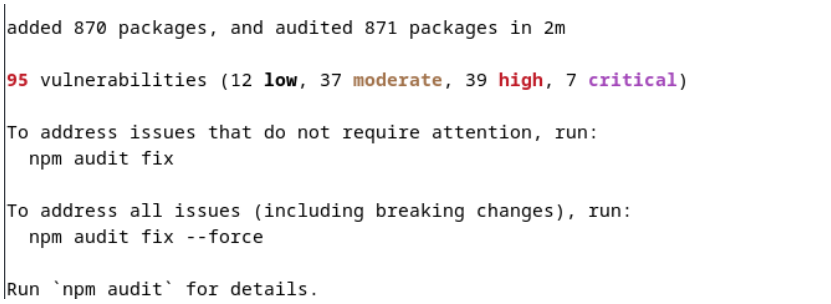
Nos movemos al interior del directorio e instalamos las librerías necesarias con:

sudo npm install



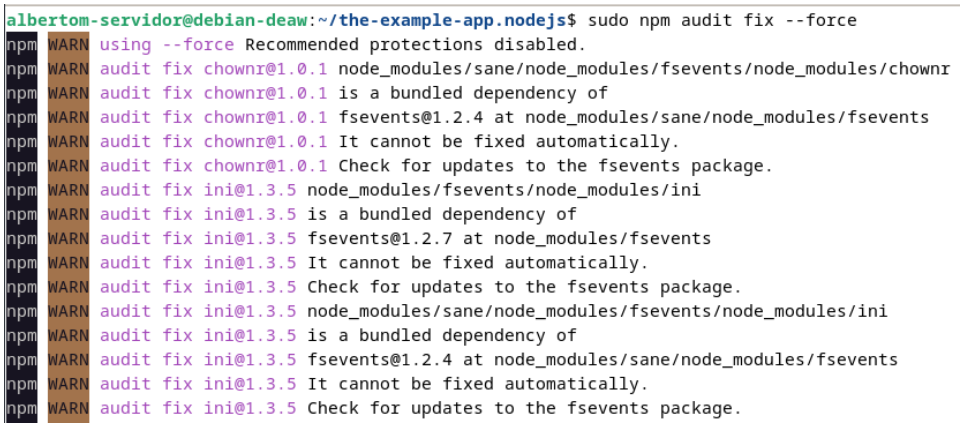
Este proceso puede tardar unos minutos y a lo largo del proceso recibiremos varias alertas de tipo old lockfile y deprecated como se ve en la imagen.

Una vez finalizada la instalación deberíamos recibir un resultado similar al siguiente:



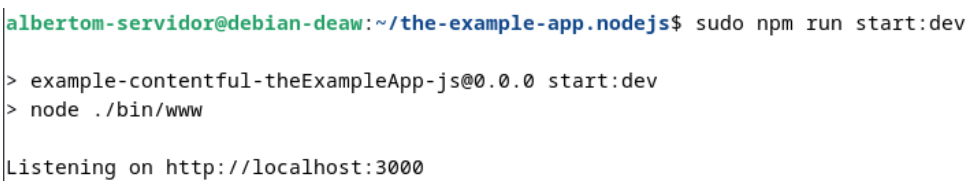
Para solucionar las vulnerabilidades encontradas ejecutamos el comando:

sudo npm audit fix --force

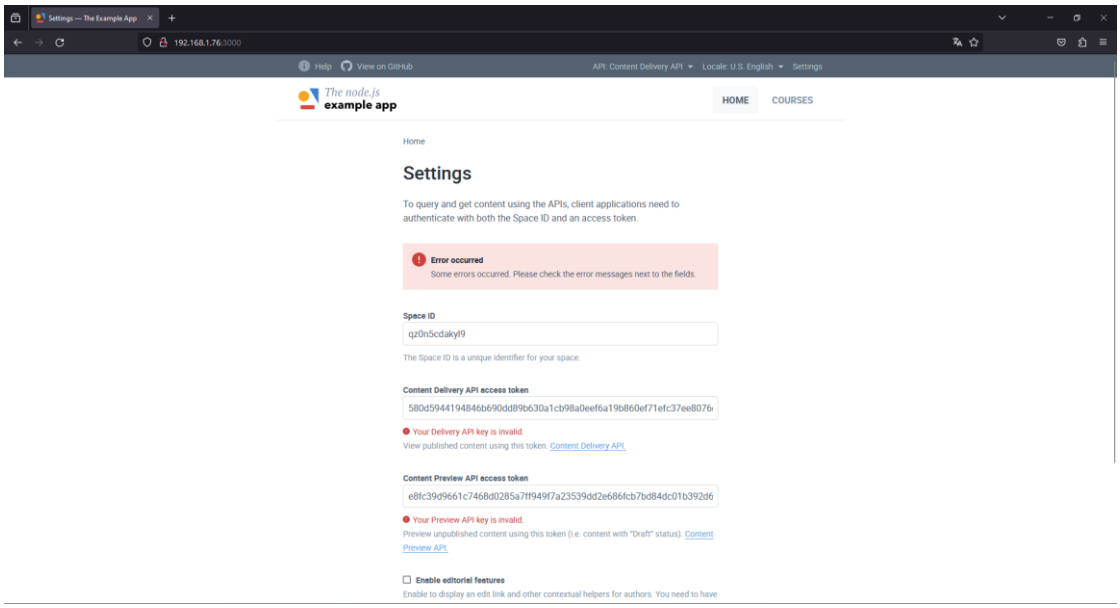


Por último, lanzamos la aplicación usando el comando:

sudo npm run start:dev



Si accedemos desde el equipo anfitrión a la dirección IP de la máquina virtual por el puerto 3000 veremos el siguiente resultado:



Al ejecutarse este acceso al servidor, veremos esta información en la consola de nuestra máquina virtual:

```
GET / 200 1057.691 ms - 11710
GET /stylesheets/style.css 200 25.923 ms - 34765
GET /scripts/index.js 200 8.521 ms - 64309
GET /images/icon-php.svg 200 13.384 ms - 2440
GET /icons/icons.svg 200 20.466 ms - 5336
GET /images/icon-swift.svg 200 24.880 ms - 1855
GET /images/icon-ruby.svg 200 29.178 ms - 7731
GET /images/icon-python.svg 200 30.833 ms - 1370
GET /images/the-example-app-logo-nodejs.svg 200 56.161 ms - 12556
GET /fonts/roboto-regular-webfont.woff2 200 30.160 ms - 19748
GET /fonts/roboto-bold-webfont.woff2 200 25.852 ms - 19884
GET /fonts/roboto-medium-webfont.woff2 200 27.640 ms - 19824
GET /images/contentful-logo.svg 200 19.854 ms - 4969
GET /images/icon-java.svg 200 16.138 ms - 1691
GET /images/icon-nodejs.svg 200 2.316 ms - 1614
GET /images/icon-dotnet.svg 200 6.029 ms - 21475
GET /images/icon-android.svg 200 12.893 ms - 2732
GET /favicon-16x16.png 200 2.134 ms - 477
GET /apple-touch-icon.png 200 3.759 ms - 1569
```

Esto quiere decir que se ha producido una petición de tipo GET sobre diferente archivos de nuestro sitio web, así como el tipo de respuesta que se ha ejecutado y el tiempo que ha tardado en ejecutarse.

CUESTIONES FINALES

a. *Cuando se ejecuta el comando npm run start:dev, lo que se hace es ejecutar un script:*

- *¿Dónde se puede ver que script se está ejecutando?*

  Se puede ver en el archivo package.json, concretamente en la sección relativa a los scripts, ahí debemos buscar el que tenga el nombre "start:dev":

```
"scripts": {
  "start:watch": "nodemon ./bin/www --ignore public/",
  "start:dev": "node ./bin/www",
  "debug": "node debug ./bin/www",
  "start": "NODE_ENV=production node ./bin/www",
  "start:production": "NODE_ENV=production node ./bin/www",
  "lint": "eslint ./app.js routes",
  "format": "eslint --fix . bin --ignore public node_modules",
  "pretest": "npm run lint",
  "test": "npm run test:unit && npm run test:integration && npm run test:e2e",
  "test:e2e": "node test/run-e2e-test.js",
  "test:e2e:dev": "node test/run-e2e-test.js --dev",
  "test:integration": "jest test/integration",
  "test:integration:watch": "jest test/integration --watch",
  "test:unit": "jest test/unit",
  "test:unit:watch": "jest test/unit --watch"
},
```

- *¿Qué comando está ejecutando?*

  Se está ejecutando el comando asociado en la pareja clave:valor, en este caso node ./bin/www, este comando ejecuta el archivo www que se encuentra en el directorio bin/:

```
  GNU nano 7.2                                    ./bin/www
#!/usr/bin/env node

/**
 * Module dependencies
 */
const app = require('../app')
const http = require('http')

/**
 * Get port from environment and store in Express
 */
const port = normalizePort(process.env.PORT || '3000')
app.set('port', port)

/**
 * Create HTTP server
 */
const server = http.createServer(app)

/**
 * Listen on provided port, on all network interfaces
 */
server.listen(port)
server.on('error', onError)
server.on('listening', onListening)

/**
 * Normalize a port into a number, string, or false
 */
function normalizePort (val) {
  const port = parseInt(val, 10)

  if (isNaN(port)) {
    // Named pipe
    return val
  }
```