


# Prácticas Docker Compose

## MEAN STACK

### 1. Preparación de los componentes y de Angular CLI

- Primero vamos a crear una aplicación Angular CLI que resida en el propio PC de trabajo y luego vamos a “dockerizarla”
- Vamos a usar node 8 pero es fácilmente modificable a la versión que se quiera utilizar.
- Primero debemos instalar en el sistema el paquete npm. Dependiendo del sistema operativo que tengamos en el PC. Si instalamos Node.js, el producto “npm” se instala también.
- Accedemos a la página de Node y descargamos la versión 8

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.



August 2018 security releases available, upgrade now

Descargar para Windows (x64)

<b>8.11.4 LTS</b> Recomendado para la mayoría	<b>10.9.0 Actual</b> Últimas características
--	---

Otras Descargas | Cambios | Documentación del API

Ó revise la Agenda de LTS.

- Descomprimos el fichero en algún directorio:

[www.apasoft-training.com](http://www.apasoft-training.com)

apasoft.training@gmail.com

```
tar xvf node-v8.11.4-linux-x64.tar.xz
```

- Cambiamos el nombre a node para que sea más sencillo
- Ponemos el directorio “bin” en el PATH para poder trabajar con npm. Por ejemplo

```
export
PATH=$PATH:/directorio_donde_hemos_instalado_node/bin
```

- Instalamos Angular-cli

```
npm install -g @angular/cli
/usr/bin/ng -> /usr/lib/node_modules/@angular/cli/bin/ng
/usr/lib
└─ @angular/cli@6.1.5
   └─ @angular-devkit/architect@0.7.5
      └─ @angular-devkit/core@0.7.5
         └─ ajv@6.4.0
            └─ fast-deep-equal@1.1.0
               └─ fast-json-stable-stringify@2.0.0
                  └─ json-schema-traverse@0.3.1
                     └─ uri-js@3.0.2
                        └─ punycode@2.1.1
                           └─ chokidar@2.0.4
                              └─ anymatch@2.0.0
.....
```

- Crear un directorio para poner todos los componentes de la aplicación y que usaremos para acceder desde los contenedores docker

```
mkdir mean
```

- Creamos una aplicación, llamada por ejemplo angular-cli

```
ng new angular-cli
CREATE angular-cli/README.md (1027 bytes)
CREATE angular-cli/angular.json (3593 bytes)
CREATE angular-cli/package.json (1316 bytes)
CREATE angular-cli/tsconfig.json (408 bytes)
```

```
CREATE angular-cli/tslint.json (2805 bytes)
CREATE angular-cli/.editorconfig (245 bytes)
CREATE angular-cli/.gitignore (503 bytes)
CREATE angular-cli/src/favicon.ico (5430 bytes)
CREATE angular-cli/src/index.html (297 bytes)
CREATE angular-cli/src/main.ts (370 bytes)
CREATE angular-cli/src/polyfills.ts (3194 bytes)
CREATE angular-cli/src/test.ts (642 bytes)
CREATE angular-cli/src/styles.css (80 bytes)
CREATE angular-cli/src/browserslist (388 bytes)
CREATE angular-cli/src/karma.conf.js (964 bytes)
CREATE angular-cli/src/tsconfig.app.json (166 bytes)
CREATE angular-cli/src/tsconfig.spec.json (256 bytes)
CREATE angular-cli/src/tslint.json (314 byt ....
....
.
```

- Vamos al directorio creado

```
cd angular-cli
```

- Deberíamos tener un directorio parecido al siguiente

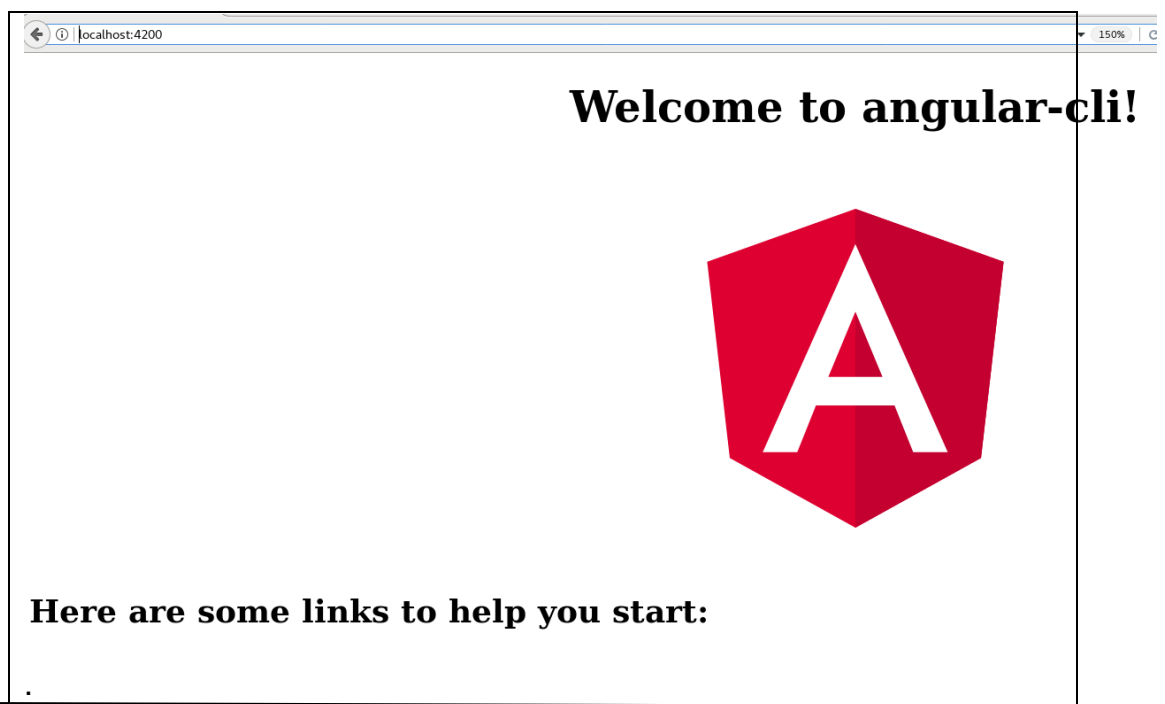
```
ls -l
total 412
-rw-r--r--.  1 root root  3593 ago 27 19:19 angular.json
drwxr-xr-x.  3 root root    68 ago 27 19:19 e2e
drwxr-xr-x. 795 root root 24576 ago 27 19:20 node_modules
-rw-r--r--.  1 root root  1316 ago 27 19:19 package.json
-rw-r--r--.  1 root root 357239 ago 27 19:20 package-lock.json
-rw-r--r--.  1 root root  1027 ago 27 19:19 README.md
drwxr-xr-x.  5 root root   267 ago 27 19:19 src
-rw-r--r--.  1 root root   408 ago 27 19:19 tsconfig.json
-rw-r--r--.  1 root root  2805 ago 27 19:19 tslint.json
```

- Para comprobar que la aplicación funciona podemos arrancarla y comprobarla. Arrancamos el servidor y probamos el navegador por el puerto 4200

```
ng serve
** Angular Live Development Server is listening on
localhost:4200, open your browser on
http://localhost:4200/ **

Date: 2018-08-27T17:22:49.616Z
Hash: 31b8594da2d7c92ebcb9
Time: 11739ms
chunk {main} main.js, main.js.map (main) 10.7 kB [initial]
[rendered]
chunk {polyfills} polyfills.js, polyfills.js.map
(polyfills) 227 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 5.22
kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 15.6 kB
[initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.27 MB
[initial] [rendered]
[i] [wdm]: Compiled successfully.
```

- Podemos acceder por el puerto 4200 para ver la aplicación funcionando



- Paramos el servidor con CTRL-C

## 2. Node.js y aplicación

- Dado que tenemos el fichero package.json generado en el paso anterior y que contiene las dependencias, no es necesario instalar angular-cli en contenedor, ya que lo hará automáticamente.
- Antes de empezar, debemos cambiar una línea de ese fichero para que puede ejecutarse dentro del contenedor, de lo contrario, la aplicación no se serviría correctamente.
- En concreto debemos modificar la línea donde se arranca la aplicación

```
"start": "ng serve",
```

- y dejarla de esta forma

```
"start": "ng serve --host 0.0.0.0",
```

- Dentro del directorio angular-cli creamos un fichero dockerfile con el siguiente contenido

```
# Lo iniciamos con la imagen oficial de Node 8
FROM node:8

# Vamos a crear un directorio donde dejar la aplicación
Angular
RUN mkdir -p /usr/mi-app

# Nos cambiamos a ese directorio
WORKDIR /usr/mi-app

# Copiamos el paquete json para gestionar las dependencias
COPY package.json /usr/mi-app

# Instalamos esas dependencias
RUN npm install
```

```
# Copiamos el código que hemos generado en el punto
anterior, al crear la aplicación angular-cli
COPY . /usr/mi-app

# Exponemos el Puerto
EXPOSE 4200

# Arrancamos
CMD ["npm", "start"]
```

- Creamos la imagen de la forma habitual a como hemos hecho durante el curso. Lo llamamos angular-cli:v1

```
docker build -t angular-cli:v1 .
Sending build context to Docker daemon 276.1MB
Step 1/8 : FROM node:8
8: Pulling from library/node
d660b1f15b9b: Already exists
46dde23c37b3: Already exists
6ebaeb074589: Already exists
e7428f935583: Already exists
eda527043444: Already exists
f3088daa8887: Already exists
5b9236fe759e: Pull complete
bd3513780305: Pull complete
Digest:
sha256:cd8ebd022c01f519eb58a98fcb05c1d1195ac356ef01851036
671ec9e9d5580
Status: Downloaded newer image for node:8
---> 55791187f71c
Step 2/8 : RUN mkdir -p /usr/mi-app
---> Running in 6866fe0f438f
Removing intermediate container 6866fe0f438f
---> 21017973566b
Step 3/8 : WORKDIR /usr/mi-app
Removing intermediate container 79d4a99510ca
---> a6ef40ede3fd
```

```
Step 4/8 : COPY package.json /usr/mi-app
```

```
---> cd6a6caaae30
```

```
Step 5/8 : RUN npm install
```

```
---> Running in 48e0ea94ea45
```

```
...
```

```
...
```

- Podemos comprobar si la imagen se ha creado

```
docker image ls angular-cli:v1
```

REPOSITORY	TAG	IMAGE ID
CREATED	SIZE	
angular-cli	v1	3a14232adac9
About a minute ago	1.25GB	

- Para probar que funciona podemos crear un contenedor

```
docker run -d --name a1 -p 4200:4200 angular-cli:v1
```

```
fed3062a2dc7558d145aca618cfc46fe4ebe5a4097e2ed636c942c6b82  
d1057e
```

- Desde el navegador intentamos acceder al contenedor con localhost: 4200 y comprobamos que accedemos sin problemas
- Después de la prueba, paramos el contenedor y lo borramos

### 3. Servidor express

- Ahora vamos a crear una imagen para la parte del servidor
- Creamos un directorio denominado express

```
mkdir express
```

```
cd express
```

- Creamos un fichero "package.json" con el siguiente contenido:

```
{
  "name": "express-server",
  "version": "0.0.0",
  "private": true,
  "scripts": {
```

```

    "start": "node server.js"
  },
  "dependencies": {
    "body-parser": "~1.15.2",
    "express": "~4.14.0"
  }
}

```

- Instalamos express

```
npm install express-generator -g
```

- Creamos una aplicación express

```

express myapp

warning: the default view engine will not be jade in
future releases
warning: use '--view=jade' or '--help' for additional
options

create : myapp/
create : myapp/public/
create : myapp/public/javascripts/
create : myapp/public/images/
create : myapp/public/stylesheets/
create : myapp/public/stylesheets/style.css
create : myapp/routes/
create : myapp/routes/index.js
create : myapp/routes/users.js
create : myapp/views/
create : myapp/views/error.jade
create : myapp/views/index.jade
create : myapp/views/layout.jade
create : myapp/app.js
create : myapp/package.json

```



```
create : myapp/bin/  
create : myapp/bin/www
```

- Cambiamos al directorio

```
$ cd myapp
```

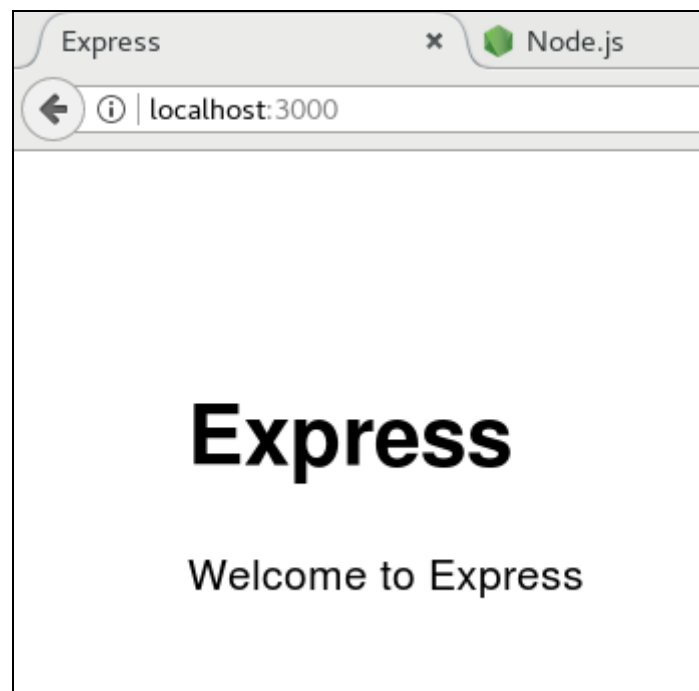
- Instalamos las dependencias

```
$ npm install
```

- Ejecutamos la aplicación

```
$ DEBUG=myapp:* npm start  
> myapp@0.0.0 start /root/mean/express/myapp  
> node ./bin/www  
  
myapp:server Listening on port 3000 +0ms
```

- Podemos escuchar por el Puerto 3000



- Paramos el servidor con CTRL+C

[www.apasoft-training.com](http://www.apasoft-training.com)

apasoft.training@gmail.com

- Una vez que hemos comprobado que el servidor funciona, podemos crear un dockerfile.
- Es muy parecido al del cliente pero escuchando a través del puerto 3000

```
# Lo iniciamos con la imagen oficial de Node 8
FROM node:8

# Vamos a crear un directorio donde dejar la aplicación
Angular
RUN mkdir -p /usr/mi-app

# Nos cambiamos a ese directorio
WORKDIR /usr/mi-app

# Copiamos el paquete json para gestionar las dependencias
COPY package.json /usr/mi-app

# Instalamos esas dependencias
RUN npm install

# Copiamos el código que hemos generado en el punto anterior,
al crear la aplicación express
COPY . /usr/mi-app

# Exponemos el Puerto
EXPOSE 3000

# Arrancamos
CMD ["npm", "start"]
```

- Creamos la imagen

```
docker build -t server:v1 .
```

- Creamos un contenedor para probar que funciona

```
docker run -d --name e1 -p 3000:3000 express:v1
```

- Una vez que hemos comprobado que accedemos desde localhost:3000 paramos el contenedor

## 4. mongoDB

- En el caso de Mongo no es necesario que creamos ninguna imagen, es suficiente con descargarnos la imagen oficial de Docker hub

```
docker pull mongo
```

## 5. Conectar los componentes. Docker Compose.

- Ahora vamos a construir la arquitectura completa con Docker Compose
- Volvemos al directorio “mean” que creamos en los primeros pasos y que contiene el resto de directorios
- Creamos un fichero “docker-compose.yml”
- Ponemos el siguiente contenido

```
# Vamos a definir los servicios y contenedores a usar
version: '3'

services:
  # Preparamos el cliente
  angular:
    image: angular-cli:v1 #    ports:
      - "4200:4200" # Puerto del cliente

  # Preparamos el servidor
  servidor:
    image: server:v1
    ports:
      - "3000:3000" #Puerto del servidor

  #Contenedor de Mongo
  database:
    image: mongo
    ports:
```

```
- "27017:27017"
```

- Arrancamos

```
docker-compose up
```

- Comprobamos que todo está arrancado a través de los navegadores:  
http://localhost:4200, <http://localhost:3000>, http://localhost:27017
- También podemos verlo con Docker

docker-compose ps			
Name	Command	State	Ports
-----			
mean_angular_1	npm start	Up	0.0.0.0:4200->4200/tcp
mean_database_1	docker-entrypoint.sh mongod	Up	0.0.0.0:27017->27017/tcp
mean_servidor_1	npm start	Up	0.0.0.0:3000->3000/tcp

- Ya tenemos un entorno MEAN para poder crear nuestras aplicaciones