Docker Compose

Alejandro Sainz Sainz

26/01/2026

[**INTRODUCCIÓN 2**](#_x1wbrpfrg1pv)

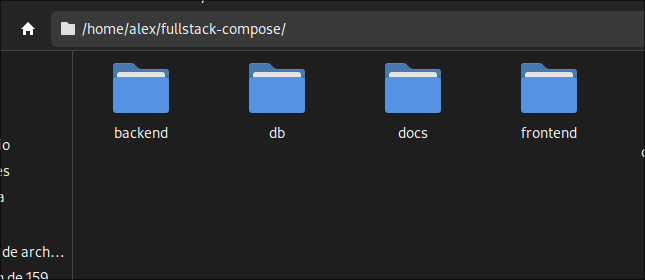
[**COMENZANDO CON LA PRÁCTICA 2**](#_o01r1qru1q2w)

# INTRODUCCIÓN

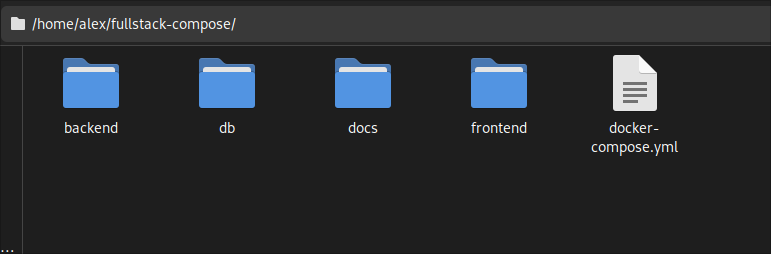
Para esta práctica debemos de crear una estructura a base de contenedores. En mi caso, voy a usar uno para alojar la base de datos, MariaDB. Después crearé otro para el back que he realizado en SpringBoot. Ya por último habrá que crear uno para el front, en este caso en Angular v20. Una vez hecho esto, deberemos de hacer que se comuniquen entre ellos de la forma deseada para así poder ejecutar la práctica de forma correcta.

# COMENZANDO CON LA PRÁCTICA

Lo primero es asegurarse de que todo está correcto como con otras prácticas con docker --v para ver la versión. Acto seguido compruebo que no hay ningún contenedor arrancado con doker ps. Una vez hecho esto debemos de comenzar creando una estructura.

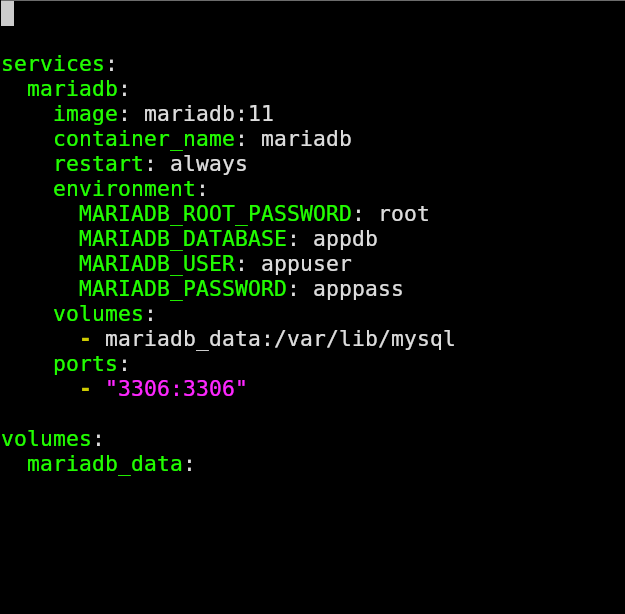


Para la siguiente parte debo de crear ya lo que será mi archivo de configuración, en este caso le he llamado docker-compose.yml.

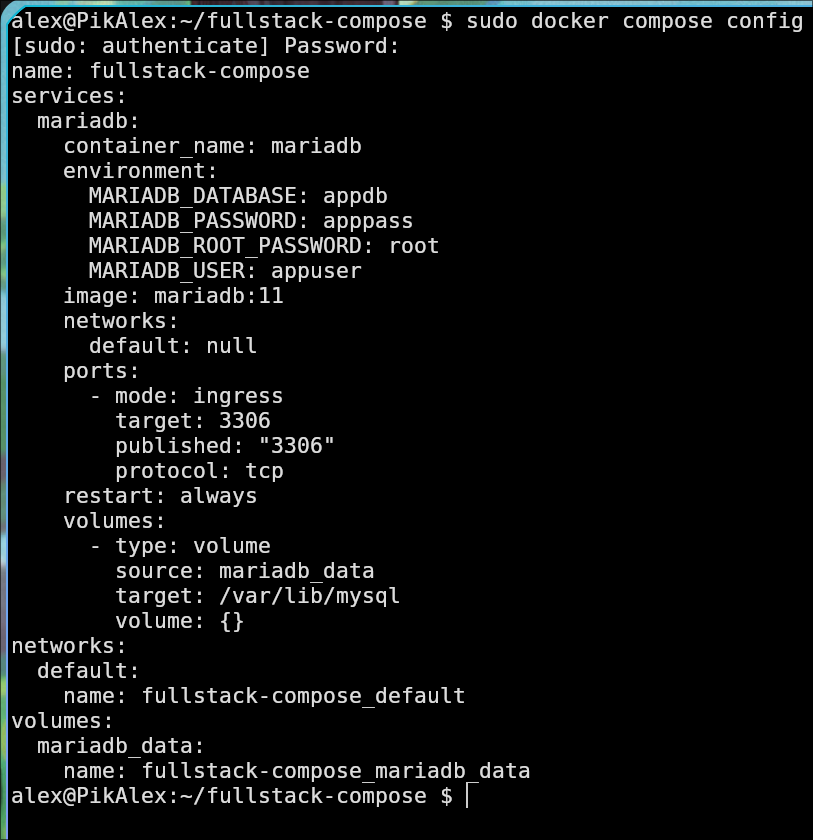


Ahora vamos a escribir una pequeña configuración básica para el. Como la base de todo proyecto es la BD empezaré configurando eso mismo.

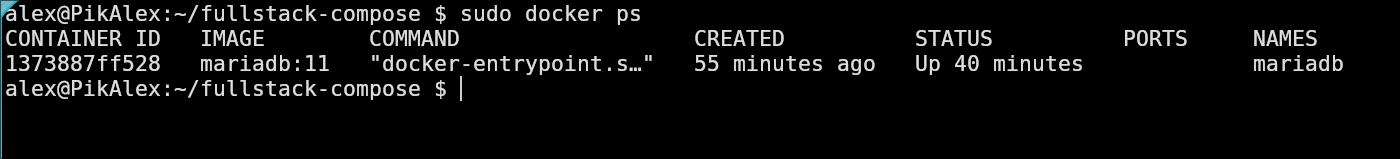
Debemos de editar el archivo de texto y añadiendo las siguiente lineas.



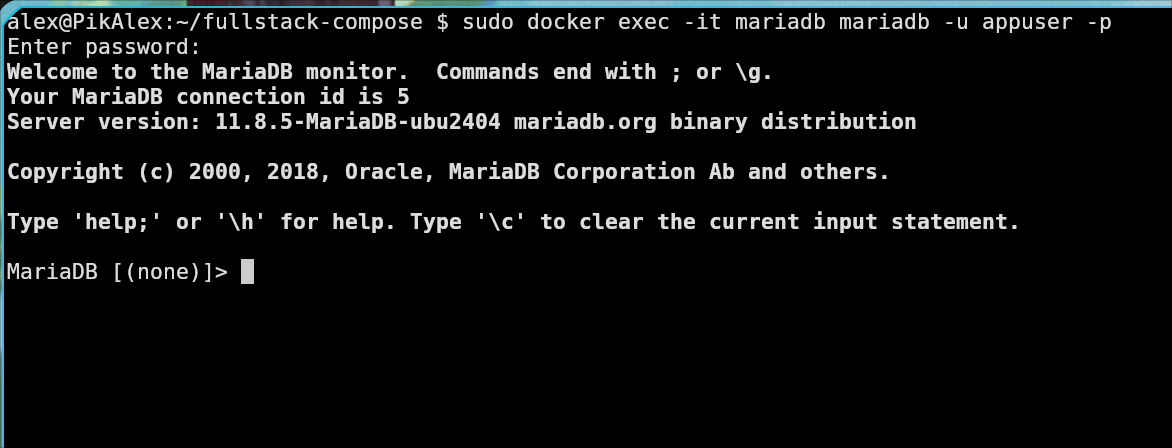
Esto es configuración básica. Los datos todavía no concuerdan con los datos de mi BD. Esos no son ni la BD, ni el usuario ni el pass. El problema es que, hasta que no tenga después en los contenedores la configuración de red, me es indistinto indicar todas estas cosas. Eso sí, el puerto le pongo como debe de ser, ya que va a ser siempre el mismo. Otra parte muy importante es la persistencia. Por eso debo de crear el volumen como se ve en la imagen. Aunque para las pruebas, mientras hacía el back y el front, la BD ya tiene datos, hay que crear un volumen para que en caso de que se añadan nuevos datos estos se mantengan en el tiempo.



Después de guardar el archivo, con sudo docker compose config, comprobamos que se ejecuta bien. Este es el resultado.



Por último, con docker ps vemos los contenedores que están levantados. Como resultado, vemos que el contenedor de mariadb está funcionando. Repito, da igual la config que tenga por ahora. Lo importante es ver que hemos escrito una configuración mínima, indicando un contenedor y, cuando ejecutamos ese archivo de configuración, se levanta sin que nosotros usemos docker start “nombre-contenedor”. Otra cosa que se me olvidaba comentar. Al crear el archivo de configuración, si no hemos descargado la imagen ni creado el contenedor, si ejecutamos docker compose, la imagen se descarga automáticamente. Así podemos ahorrarnos algún paso y ganar tiempo.



Como última prueba. Si ejecutamos el comando que vemos arriba se supone que nos conectamos a la BD. Nos pide el pass, introducimos el que hemos incluido en el archivo de config por ahora y como vemos, estamos dentro.

Ahora tocan una serie de cosas que es importante saber antes de seguir. Como en otras prácticas, esto ha sido buscar por internet, pedir a la IA que filtre resultados y mirar alguna guía. Ahora explico porque esto es importante.

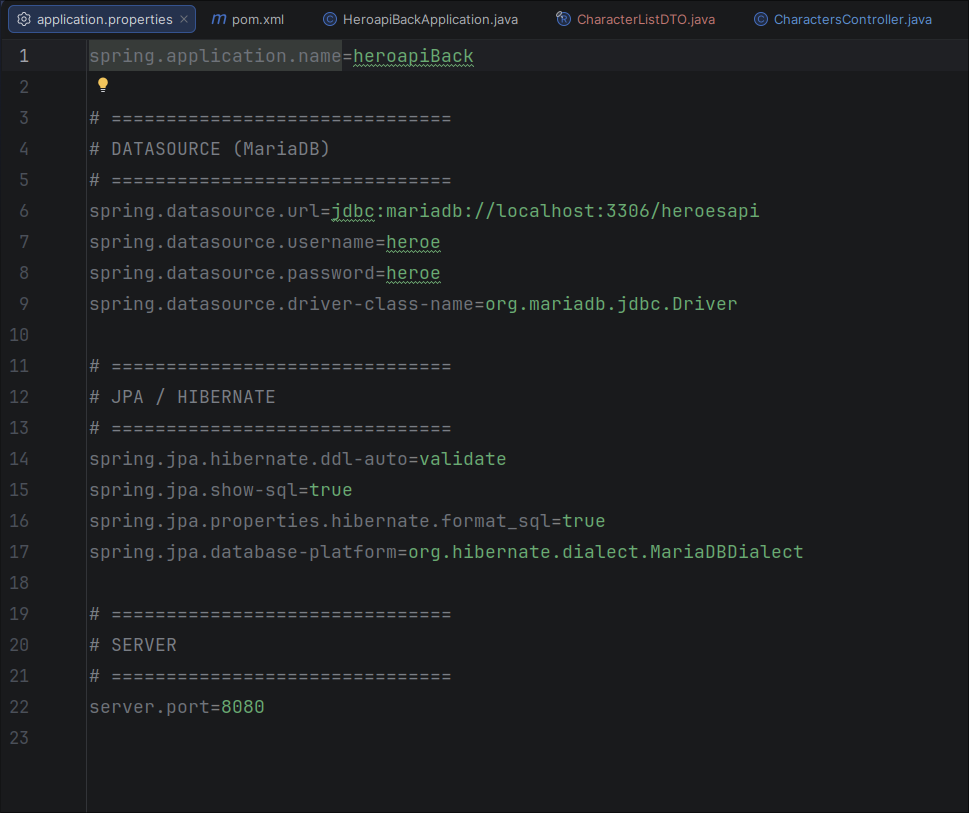
<https://docs.docker.com/engine/containers/run/#env-environment-variables>

Vamos por partes. En secciones anteriores dentro de la misma página he visto una cosa interesante. Por lo menos para la parte que nos toca. La primera es que los contenedores que forman parte de uno mayor con compose no necesitan configuración de red adicional, en principio. Esto es porque docker compose funciona como servidor dns interno y el resuelve.

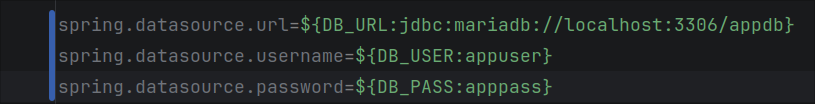
Porque son importantes las variables de entorno, que están bastante abajo en la página. Porque para el back, en un archivo de SpringBoot llamado application.properties yo tengo que indicarle cual es la BD, el user y el pass. Allí ya lo tengo preparado, pero todo está pensado para trabajar en local, así que todo está configurado con localhost. Pero, si configuramos variables de entorno, si no lo he entendido mal, SpringBoot o MariaDB o lo que sea que use, tratará primero de usar la variable de entorno, si no la hay, usará la otra que yo le indico en application.properties. Eso es como yo he entendido que funciona.

<https://docs.docker.com/compose/how-tos/environment-variables/set-environment-variables/>

Vamos a explicarlo poco a poco.

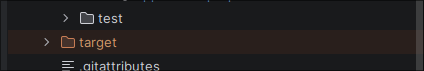


Este es el application properties que yo tengo configurado que trabaja en local. En el yo le indico la dirección y el puerto de la BD. El usuario y el pass. Docker, no entiende lo que hay dentro de los contenedores. Es decir, le da igual. El solo sabe que si tienes un contenedor llamado spring y otro mariadb y se quieren comunicar, él lo va a hacer por esos nombres. Parecido a como funciona un servidor DNS. Las otras variables las puede proporcionar mediante el archivo yml que hemos ido editando. Spring, al parecer, permite este tipo de variables de entorno. Las que nosotros usamos en Node en archivos .env. Luego, un servicio llama a otro, docker resuelve, y proporciona las variables necesarias para que ocurra. Docker proporciona el entorno idóneo y los demás, a lo suyo.



Cambio esas líneas para obtener variables de entorno. Espero estar haciéndolo bien.

Ahora toca compilar el proyecto en un archivo .jar, como ya hice en otra práctica. Después de unos problemillas, yo tenía la jdk 21 y el proyecto está en la 17, compilado y listo.

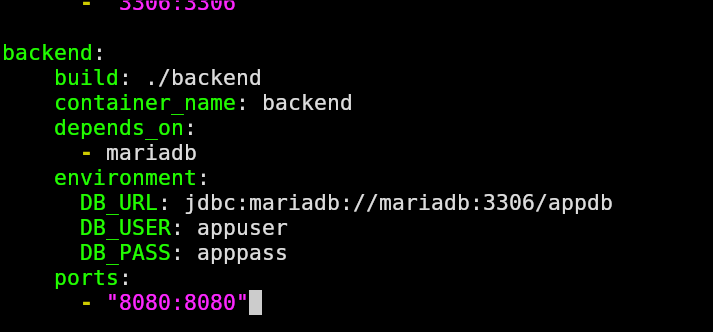


Después de compilarlo con maven ya tengo la carpeta target que me hace falta para el contenedor.

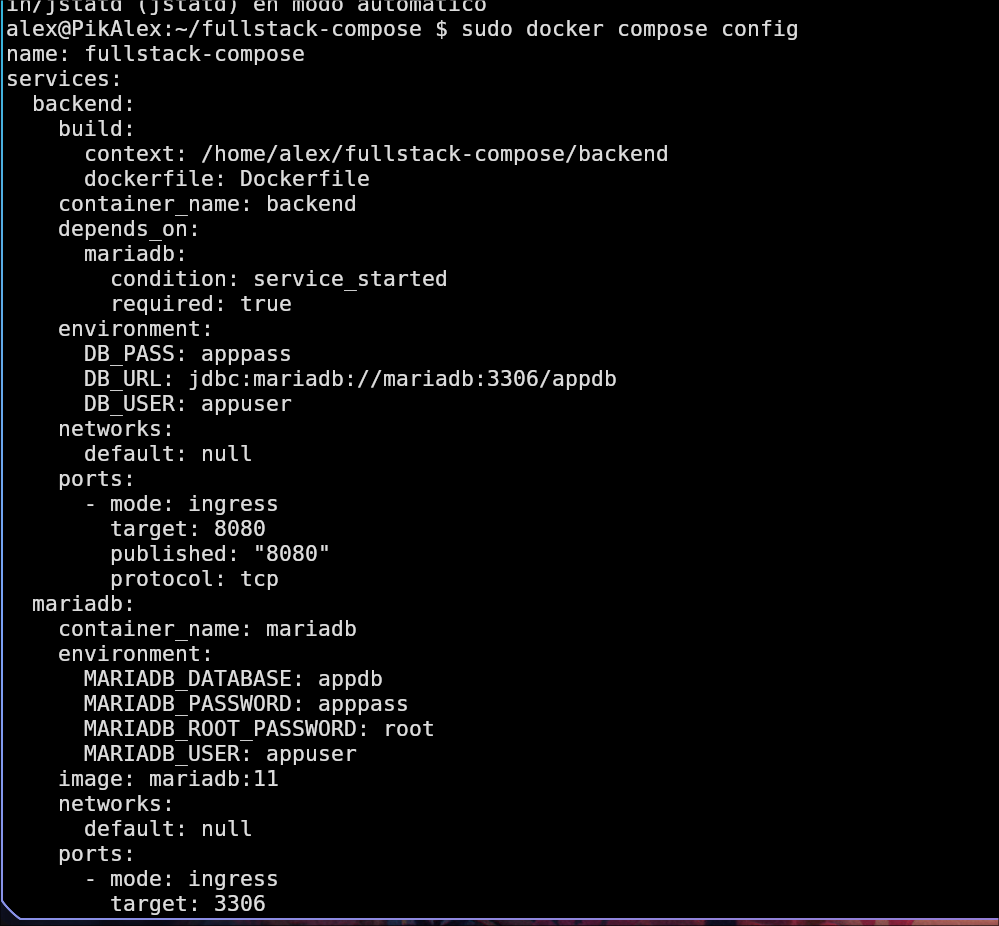
El comprimido .jar que está dentro lo copio a la carpeta backend que está creada en mi proyecto. Dentro, creo también un archivo llamado Dockerfile e introduzco esto.



Esta es la imagen de java 17, de eclipse temurin. Luego los archivos con los que va a trabajar, el puerto que va a exponer y los puntos de entrada o los archivos de inicio.

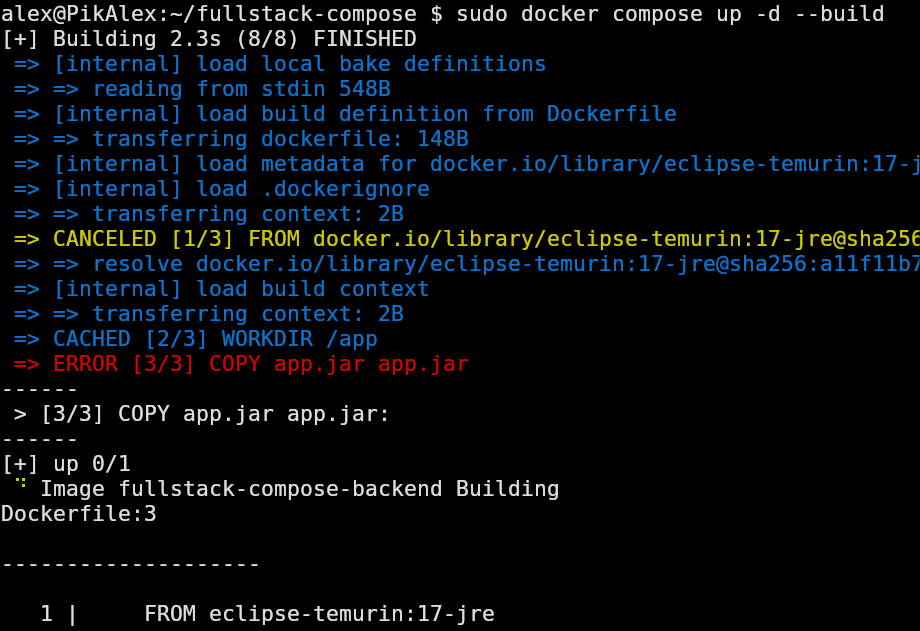


Añado esto al archivo docker-compose. En él tengo el directorio desde el que se construye, el nombre del contenedor, depende de mariadb(el contenedor) y las variables de entorno. Aunque ya las tengo definidas en el application, por si acaso, otra vez aquí. No vaya a ser que pasen cosas.



Realizo una comprobación. En la imagen la vemos bien, pero hay que saber que yml es un poco puñetero. Tenía mal indentada una línea y me daba error. Pásale el archivo a chappie y que te diga cual es el error. Un espacio mal puesto y explota.

Ahora vuelvo a ejecutar docker compose up -d –build y vemos que pasa.



Explota. Supongo que tendré que descomprimir el .rar donde va el .jar. No se porqué me da en la nariz.

Pues no. Esto pasa por seguir guías al pie de la letra sin fijarse del todo bien. El nombre de mi .jar era algo así como heroApi0.0.1snapshot, etc. Mi archivo compose busca algo llamado app.jar. Normal que explote. Volvemos a intentarlo.



Mira tu por donde.

Ahora, supuestamente, viene otra parte que me ha costado entender.

En una práctica de despliegue de aplicaciones, yo hice ya algo parecido, pero todo en local. Un servidor Nginx, una build de Angular y un back de Spring que se ejecuta solo por el servidor Tomcat que lleva embebido. Hasta aquí, bien, más o menos.

Cuando buscas por internet como desplegar Angular en Docker, la gente indica de tener node dentro del contenedor. Eso yo no lo entendía de ninguna manera. Hasta que dí con el motivo. El objetivo de los contenedores es que no dependan de nada del host. Por lo tanto, lo que se suele hacer es, crear el contenedor, que va a tener una imagen de Nginx. Pero añaden node, para que al arrancar el contenedor, el instale las dependencias dentro del contenedor, haga el ng build, descargue todo lo necesario y así, angular funcionará sin problemas sin que tu tengas instalado [node.js](http://node.js) en tu equipo. Ahí ya lo entendí. Tiene que estar preparado para que funcione en un equipo vacío, sin instalar nada.

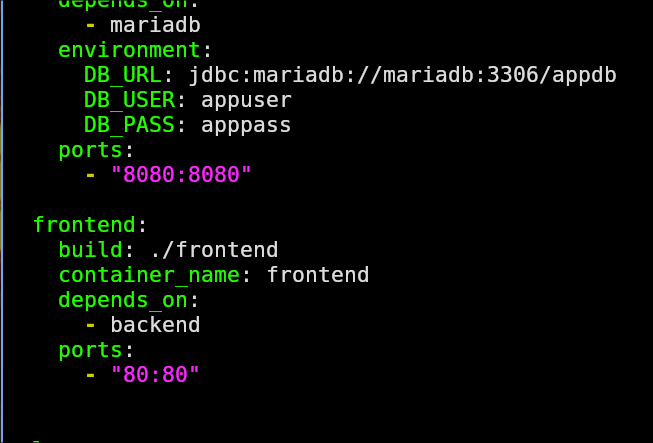


Comenzando con lo primero. Docker es como git. Se puede crear un archivo .dockerignore para que ignore todo lo que copias en el. Lo más importante es node\_modules, para que docker no tenga que cargar con 300 megas de archivos que no le van a hacer falta. Así que creo un archivo .dockerignore dentro de la carpeta front y añado lo que se ve en la imagen.

Después de copiar mi proyecto de front debo de crear un Dockerfile dentro.



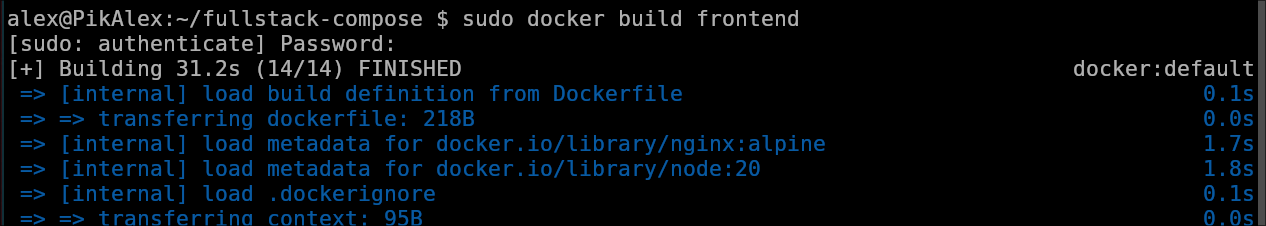
Como ya he explicado antes. Lo que hace es tener Node. Preparará todo el entorno según lo que tengo en package.json y luego servirá desde la imagen de nginx.



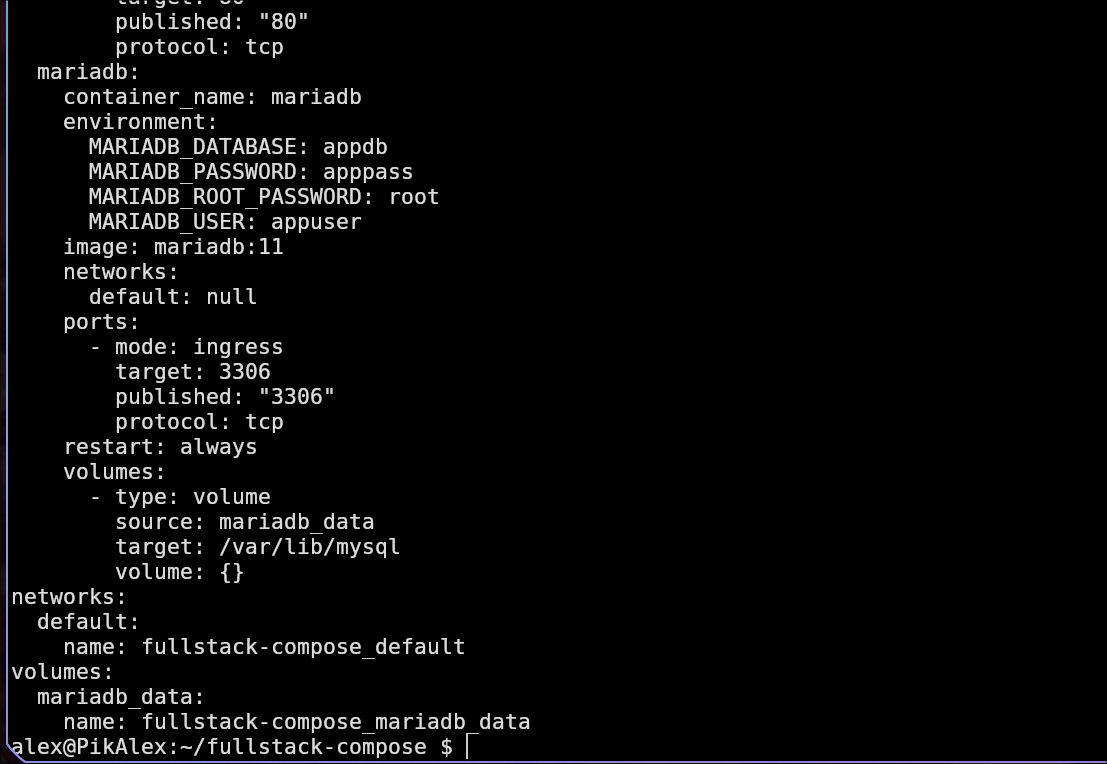
Añado el nuevo servicio, o contenedor a mi archivo docker-compose.



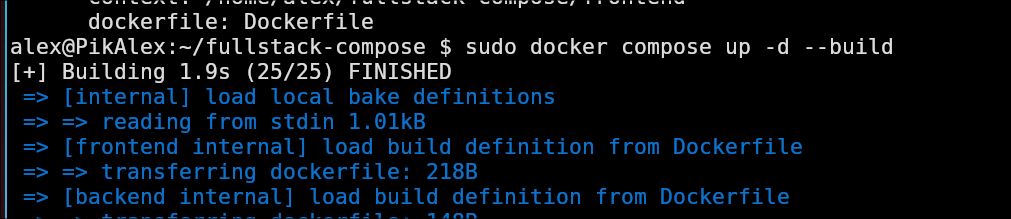
Para hacer la prueba, uso sudo docker build frontend. Entonces veo como va node descargando dependencias que el front de angular va a necesitar.



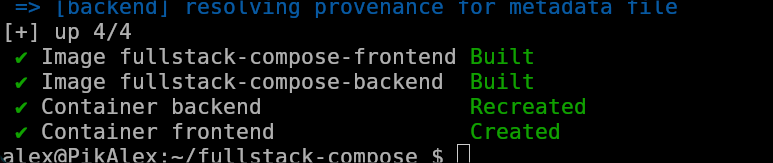
Ha tardado un poco, pero que son 30 segundos.



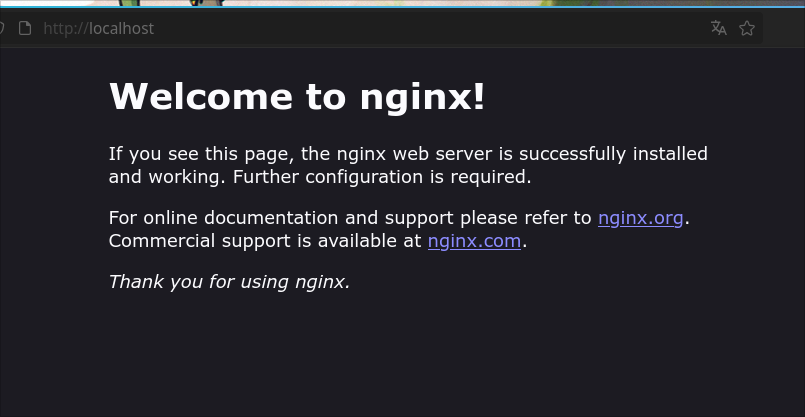
Otra validación. Hay que ver de nuevo que usando docker compose config nos devuelve todo correcto. Hay que validar cada paso que se da para que esto no salte por los aires.



Ahora compruebo que ejecutando compose build funciona y construye todo.



Ahora tendremos que hacer una prueba en el navegador. Si entro a localhost, debería de funcionar, Angular por lo menos.



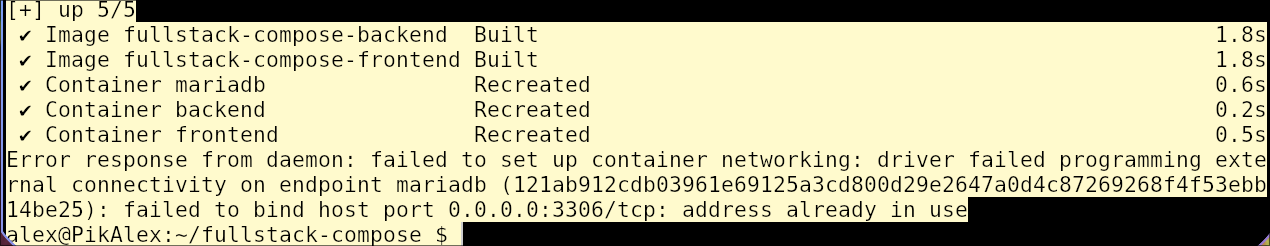
Pues no. Nginx funciona. Pero no se está sirviendo Angular. A revisar toca.

De nuevo consulta suelta por la red. Mi Dockerfile tiene esto:



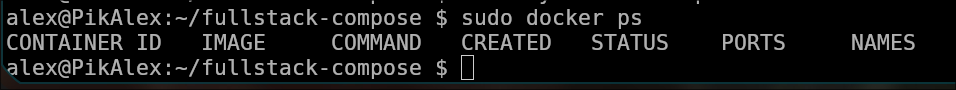
Esto quiere decir que está usando todo el contenido de la carpeta dist para servir desde nginx. Pero en dist, hay muchos archivos, y ni siquiera le he indicado el nombre de mi app.

Después de arreglar eso vamos a ver que sale.



Pinta a roto. Conflicto con mi servicio local de mi BD. A buscar.

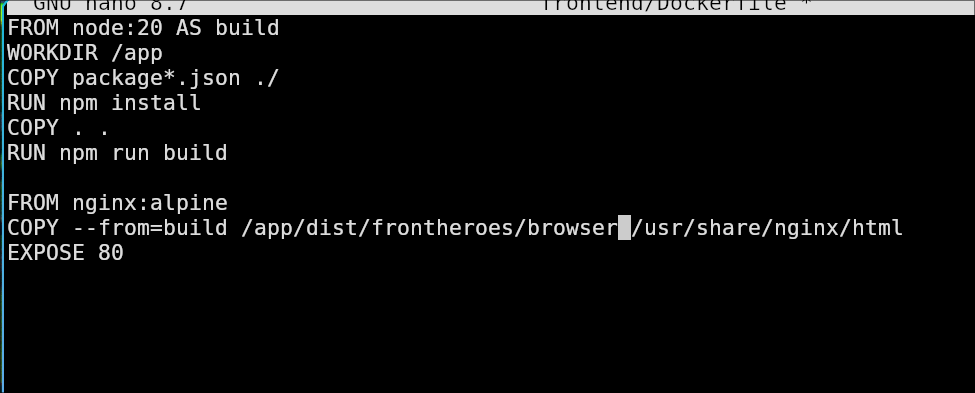
Probé parando el servicio local de mariaDB con systemctl stop mariadb pero seguía igual, porque al dar error el docker compose no se levantaba el servidor de mariadb del contenedor. Así que tengo que buscar otra cosa. Es que además, al dar ese error, los contenedores ni arrancaban. Vamos a probar con un docker ps.



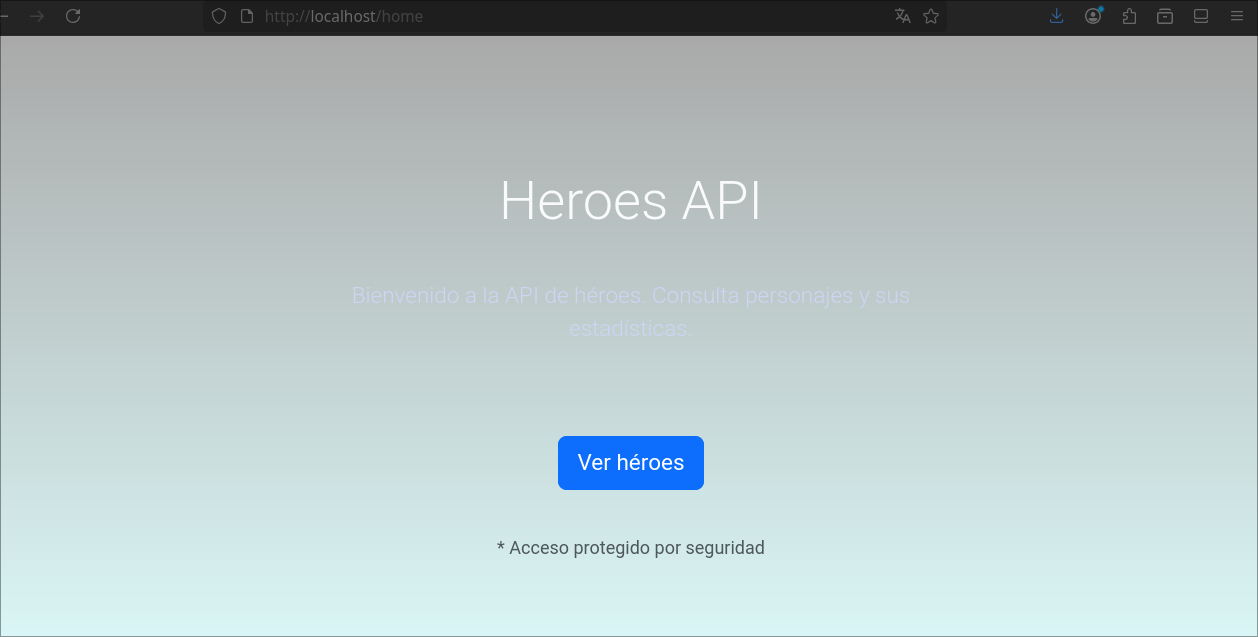
No hay ninguno levantado.

Buscando, parece ser, que los contenedores tienen por defecto conectividad exterior. Eso quiere decir que estoy tratando de exponer dos veces el mismo puerto. De ahí viene el problema. Lo que voy a hacer es modificar el puerto o eliminar el de docker-compose.yml.

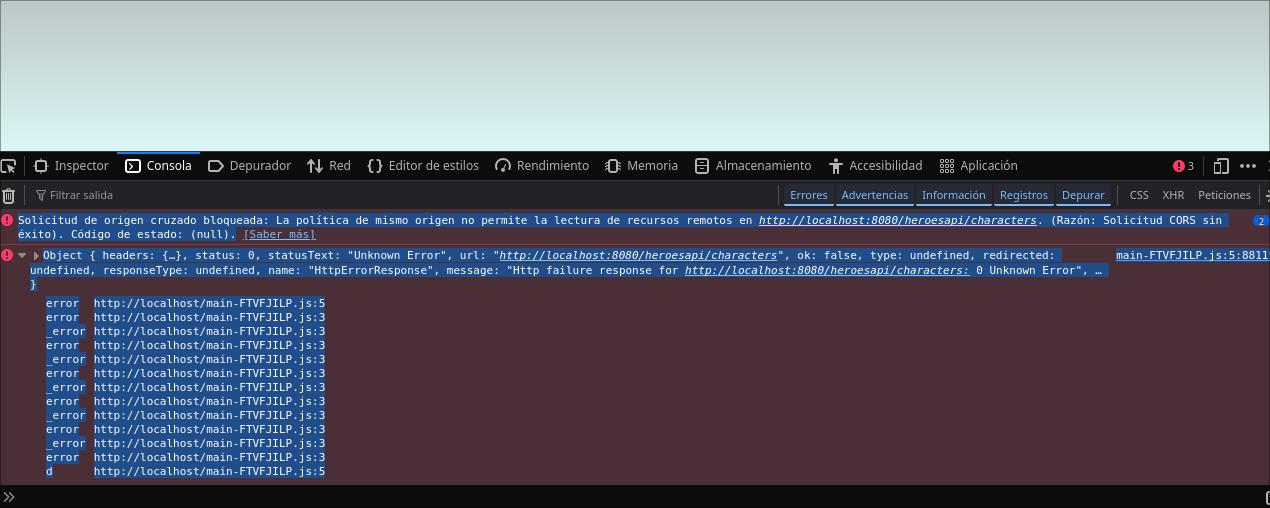
Después de hacer eso buildea, pero sigo sin ver la interfaz de angular.



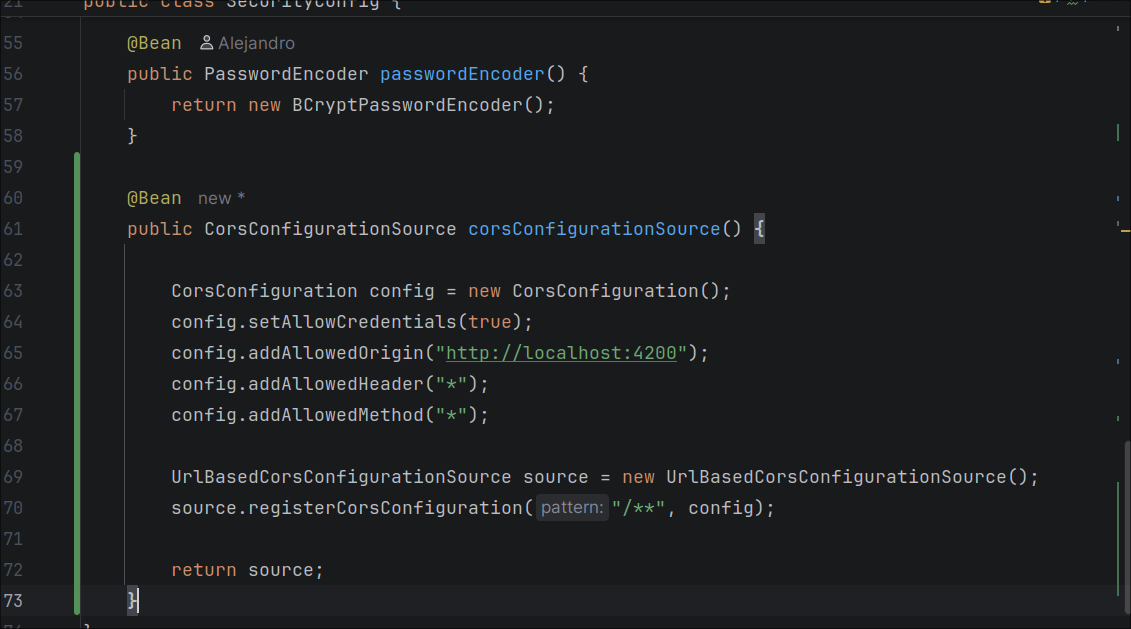
Una de las cosas que encontré por ahí, es que cuando haces build de un proyecto de angular, lo principal está dentro de una carpeta browser. Vamos a ver si añadiendo eso al Dockerfile de front obtengo resultado.



Ahora sí. Algunas letras se ven fatal, pero me vale. Por lo menos esto ya funciona.



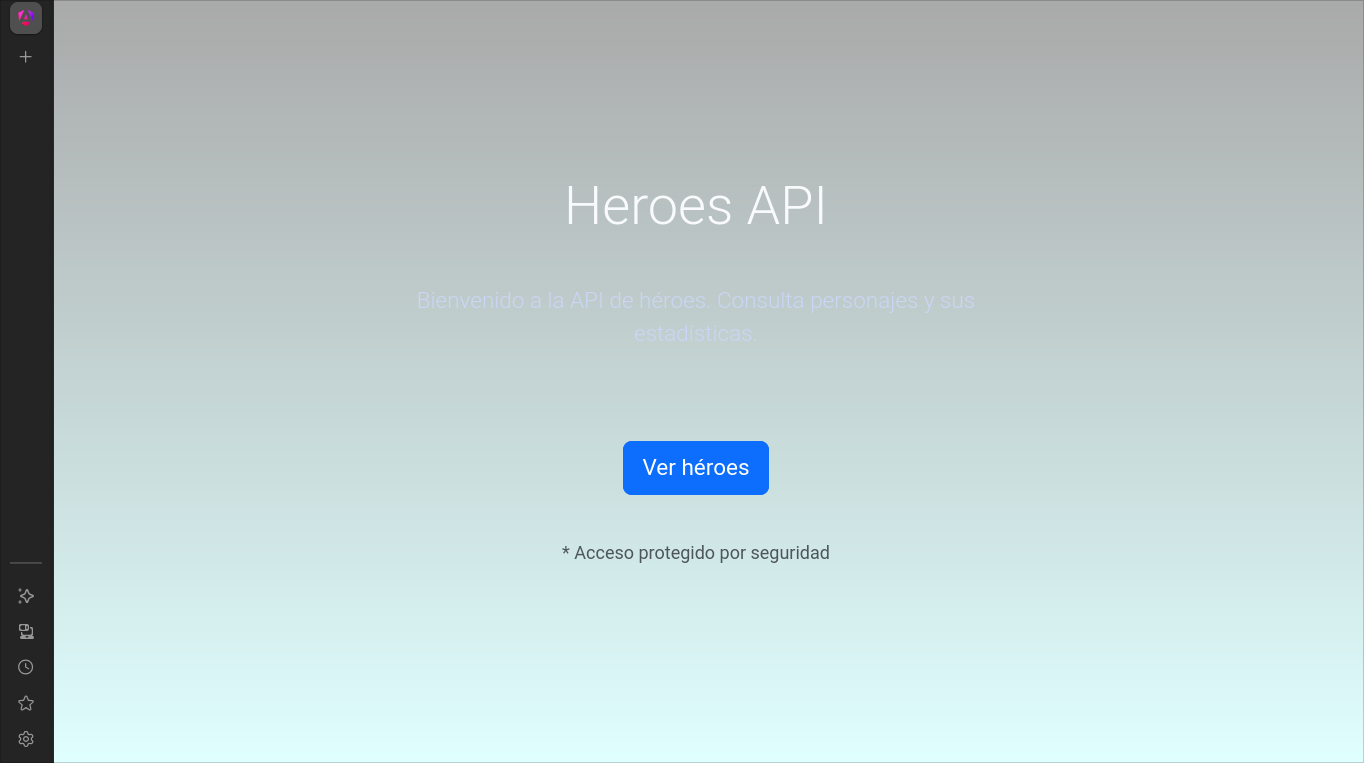
Primera prueba de conectividad. Fallo de las CORS. Pensé que las tenía bien configuradas en el back, pero parece que no es así.



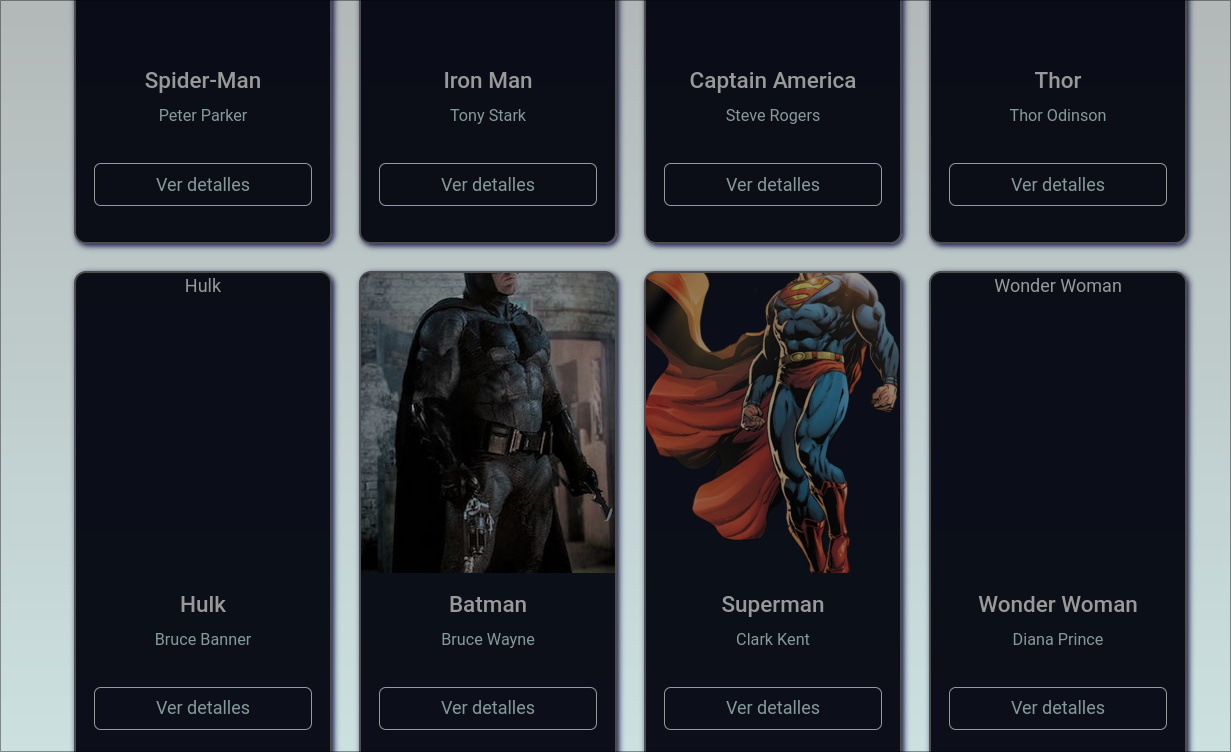
Esta era mi configuración de las CORS en SpringBoot. El problema es que esta valía cuando yo servía el front desde la terminal. Con el puerto 4200 entraba todo. Pero ahora el front lo sirve Nginx, que usa el puerto 80. Por lo tanto, tengo que quitar de ahí el puerto o simplemente cambiarlo por :80. Tendré que compilar de nuevo y volver a levantar el contenedor.

Las cosas como son. He tardado como cuatro horas. Fallos de cors, cambios de configuraciones, cambios en el front, en el back. He tenido que tirar de internet, de chatgpt, hasta de fé. Pero bueno, lo normal, no estaba preparado para esto. Son muchas cosas y, el proceso para salir lo que sería a producción, no estoy preparado, ni de lejos.

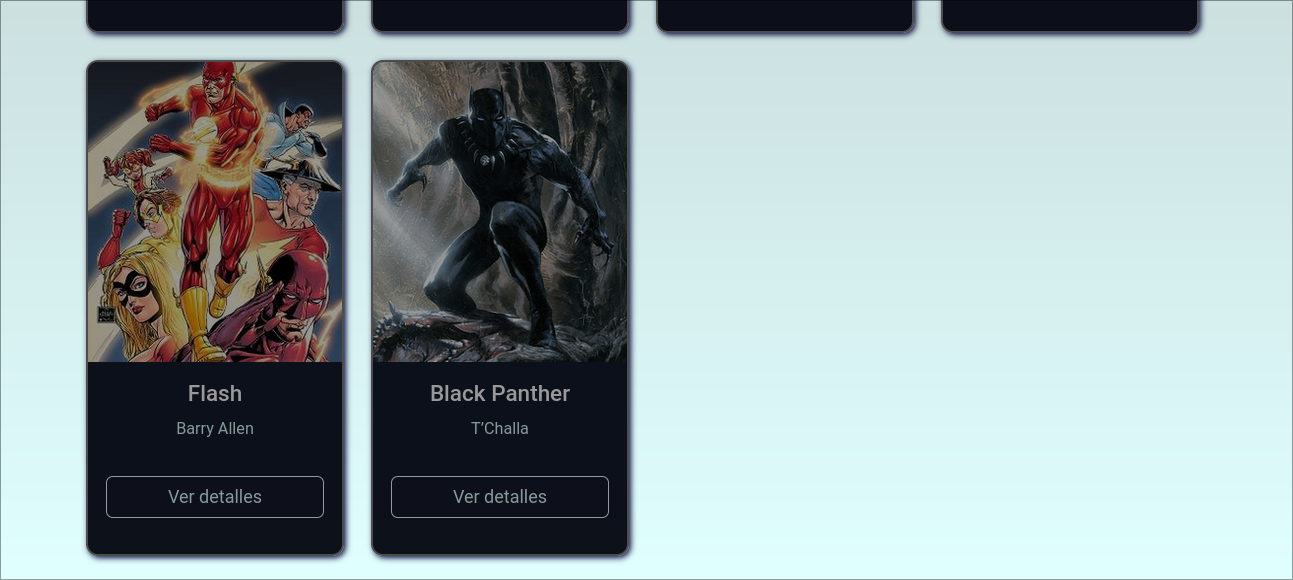
Tras muchos cambios, he vuelto a conseguir esto sin que salgan mil warnings en la consola del navegador.



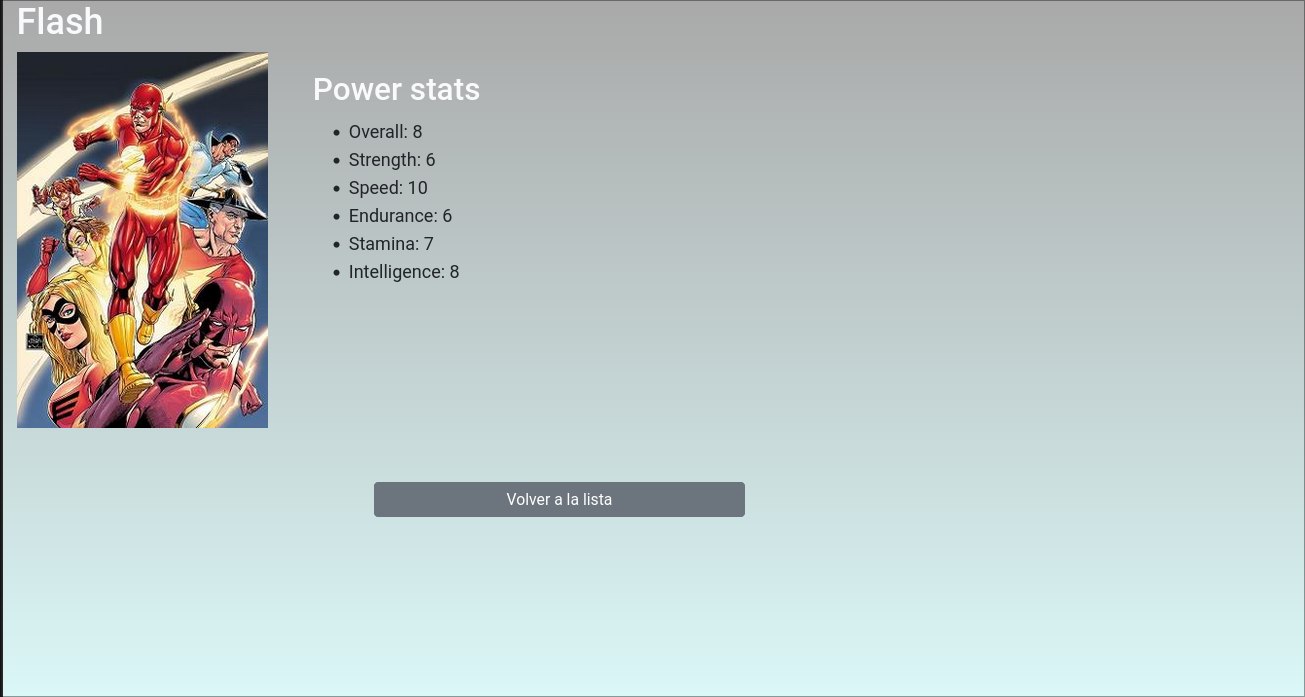
Y ahora ya por fín, si pulso en ver heroes:



Ya era hora. Eso si, algunas de las urls de las imágenes están protegidas y no se cargan. Tengo que cambiarlo.



Vamos a pulsar en Ver detalles, para ver las stats de un heroe.



Pues ya estaría. Incluso funciona el botón de volver a la lista. Por lo menos, ya se puede hacer algo.