1 - PERSISTENCIA Y DOCKER COMPOSE

Migración de persistencia en base de datos

Instalar un contenedor de la imagen 'postgres', con un volumen con nombre 'db-persistance', usando la versión 9.6.1 de Postgres. Tras ello, eliminar el contenedor y levantar un nuevo postgres, pero con versión 9.6.2 usando el volumen anteriormente creado.

La entrega de este ejercicio debe ser un pdf compuesto por las capturas de:

- *Instalación de ambos postgres con log*
- *Salida de 'docker volume inspect db-persistance'*
- *Logs de ambos postgres indicando que se ha iniciado correctamente, con 'docker logs*

<container>

Creación de Volume ‘db-persistance’

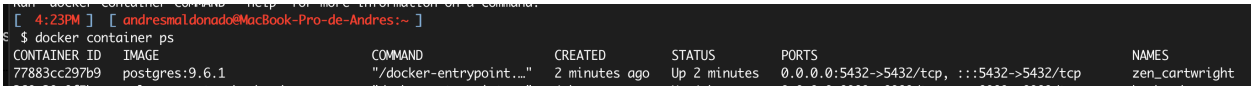
```
[  3:55PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker volume create db-persistance
db-persistance
```

```
Run 'docker volume COMMAND --help' for mo
[  4:11PM ]  [ andresmaldonado@MacBook-Pr
 $ docker volume ls
DRIVER    VOLUME NAME
local     0a928c96c1df2ad4a0e33cf730a37f4
local     1d6a4b9c82894c3d7ba5a60e6e9bfd7
local     3d13a7bb053eddab5dc75645e6e5e40
local     4fc1f96ae8c90a16eb392672d7e35aa
local     5a9ea6d4507f9212e4bf7da5c10fc64
local     5e611147d0daeaaecf7c20d2ebc1d17
local     7ead26fa685e29b2cf2175e8c11bd60
local     8ef43b7a0191b81a72e78962271ce22
local     9db207b45c8a94929f3ed6bdc313657
local     27fe8611ecfcad4dab2546c3835fd59
local     28c3147ccbe218b8cb20d036f7e1657
local     38e84ebd32f91e9b1c0e569288d8883
local     753e5420715f86a75f68954147304d5
local     981207f9bae8e28574bc6476047b910
local     55418258de9b2abec675a330ca48e1b
local     81355596551e1dea5d6b0943a9886b8
local     b7b8cfbd4750e10e7cd001d43303d74
local     c48e61a9f734a28060260cb7eea869e
local     ce1894727f1b1ed617ef4234df28c5e
local     d91951f2d1780c91470dd738b0f3bf7
local     dadc6705da29e94f3090c2a4d4b0155
local     db-persistance
local     dc5b4b26160267555c38eed6ae963de
```

Se crea postgres 9.6.1 usando como volumen para la base de datos el anterior creado:

```
[  4:20PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker run -p 5432:5432 -d -e POSTGRES_PASSWORD=pgpass -e POSTGRES_USER=pguser -e
POSTGRES_DB=exercicedb -v db-persistance:/var/lib/postgresql/data postgres:9.6.1
Unable to find image 'postgres:9.6.1' locally
9.6.1: Pulling from library/postgres
5040bd298390: Pull complete
f08454c3c700: Pull complete
4db038cdfe03: Pull complete
e1d9ba315f03: Pull complete
25e0ee93170e: Pull complete
3f28084c3f51: Pull complete
78c91f0aedcd: Pull complete
93ab52dbcbb8: Pull complete
27ec75825613: Pull complete
```

Se adjunta captura con contenedor corriendo



Ejecuto "docker volume inspect db-persistance”

```
[  4:26PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker volume inspect db-persistance
[
    {
        "CreatedAt": "2022-03-04T19:21:11Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/db-persistance/_data",
        "Name": "db-persistance",
        "Options": {},
        "Scope": "local"
    }
]
```

El container tiene id “77883cc297b9”. Ejecuto “docker logs 77883cc297b9” para pintar el log:

```
[  4:29PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker logs  77883cc297b9
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting dynamic shared memory implementation ... posix
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
syncing data to disk ... ok

Success. You can now start the database server using:

    pg_ctl -D /var/lib/postgresql/data -l logfile start

--auth-local and --auth-host, the next time you run initdb.
waiting for server to start....LOG:  could not bind IPv6 socket: Cannot assign requested address
HINT:  Is another postmaster already running on port 5432? If not, wait a few seconds and retry.
LOG:  database system was shut down at 2022-03-04 19:21:09 UTC
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
 done
server started
CREATE DATABASE

CREATE ROLE


/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

waiting for server to shut down...LOG:  received fast shutdown request
LOG:  aborting any active transactions
.LOG:  autovacuum launcher shutting down
LOG:  shutting down
LOG:  database system is shut down
 done
server stopped

PostgreSQL init process complete; ready for start up.

LOG:  database system was shut down at 2022-03-04 19:21:10 UTC
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

Como solicitado, elimino el contenedor

```
[  4:33PM ]  [ andresmaldonado@MacBook-Pro
 $ docker kill 77883cc297b9
77883cc297b9
```

Ejecuto el comando anterior, cambiando esta vez la versión de postgres a 9.6.2

```
[  4:34PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker run -p 5432:5432 -d -e POSTGRES_PASSWORD=pgpass -e POSTGRES_USER=pguser -e POSTGRES_DB=exercicedb
-v db-persistance:/var/lib/postgresql/data postgres:9.6.2
Unable to find image 'postgres:9.6.2' locally
9.6.2: Pulling from library/postgres
10a267c67f42: Pull complete
e9a920522e33: Pull complete
6888e696bd71: Pull complete
798096eed143: Pull complete
fb58419959b5: Pull complete
97f9ec09cb68: Pull complete
d58678d9d3ab: Pull complete
ece2bc4a78f4: Pull complete
eadac36b8440: Pull complete
4da13987a6ca: Pull complete
bd2eab93fc5a: Pull complete
2efd8a94a8d7: Pull complete
cd1f07c4ebbe: Pull complete
Digest: sha256:5284ba74a1065e34cf1bfccd64caf8c497c8dc623d6207b060b5ebd369427d34
Status: Downloaded newer image for postgres:9.6.2
d3bda8aca606c603df5efa61e4d566fcc7b6e8019d9239b46186137faec6f1b4
```

Adjunto captura del nuevo contenedor corriendo

```
[  4:35PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker container ps
CONTAINER ID   IMAGE                COMMAND                  CREATED              STATUS              PORTS                                       NAMES
d3bda8aca606   postgres:9.6.2       "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   busy_torvalds
```

Adjunto “docker volume inspect”

```
[  4:36PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker volume inspect db-persistance
[
    {
        "CreatedAt": "2022-03-04T19:35:25Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/db-persistance/_data",
        "Name": "db-persistance",
        "Options": {},
        "Scope": "local"
```

El container tiene id “d3bda8aca606”. Ejecuto “docker logs d3bda8aca606” para pintar el log:

```
[  4:39PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~ ]
 $ docker logs d3bda8aca606
LOG:  database system was interrupted; last known up at 2022-03-04 19:21:11 UTC
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  invalid record length at 0/14EEB48: wanted 24, got 0
LOG:  redo is not required
LOG:  MultiXact member wraparound protections are now enabled
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

Vemos que esta vez no me imprime el log de creación porque dicho proceso ocurrió en la instancia anterior. Considerando que el log de creación aparece únicamente cuando la base de datos es creada, se deduce que aceptó el volume “db-persistance” de la instancia anterior.

2 - PERSISTENCIA Y DOCKER COMPOSE

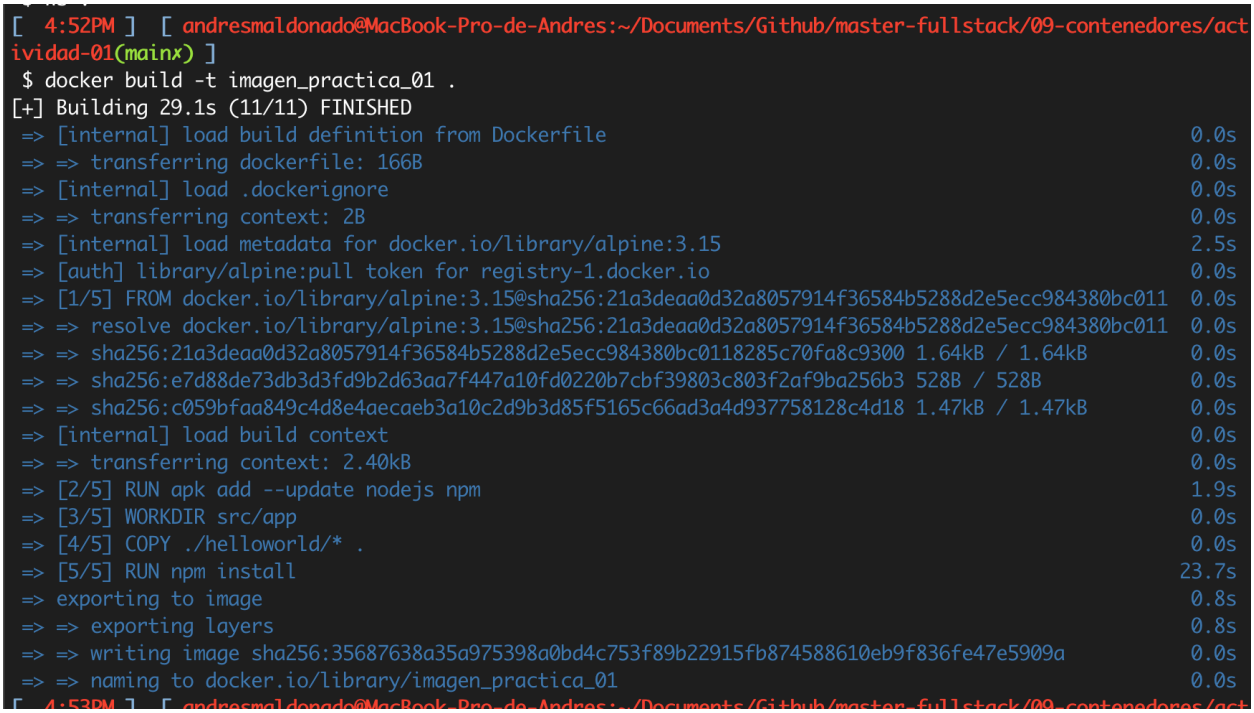
Realizar un bind mount para modificar código dinámicamente

Instalar un contenedor con la imagen de la Práctica 1, usando un bind mount para ser capaz de cambiar el texto que salga por pantalla por un mensaje diciendo el nombre del alumno.

La entrega de este ejercicio debe ser compuesta por las capturas de:

- *Lanzamiendo del docker*
- *Salida de 'docker volume inspect <container>*
- *Captura de imqgen del display de pantalla del servicio web*

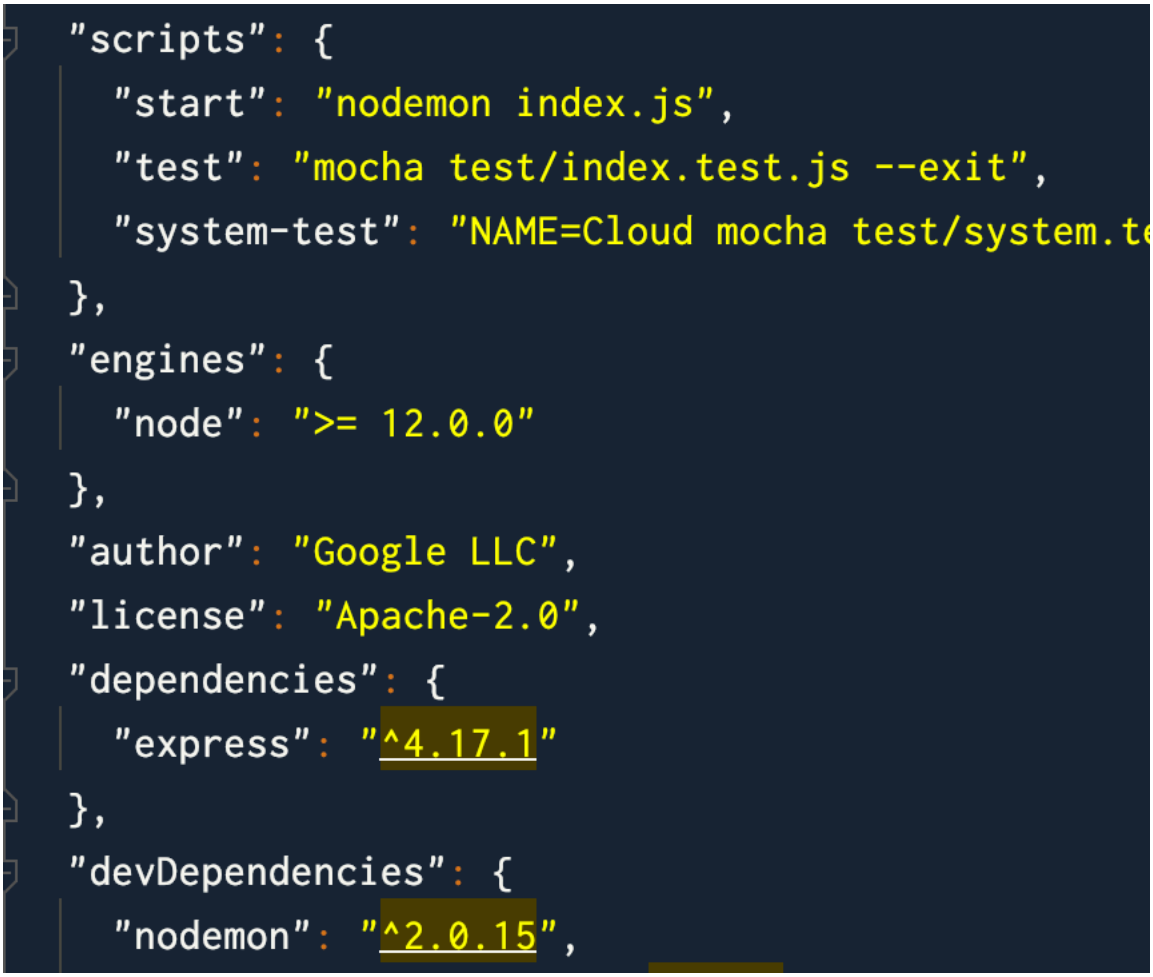
Empiezo por construir la imagen de la práctica 1, le he nombrado “imagen_practica_01”



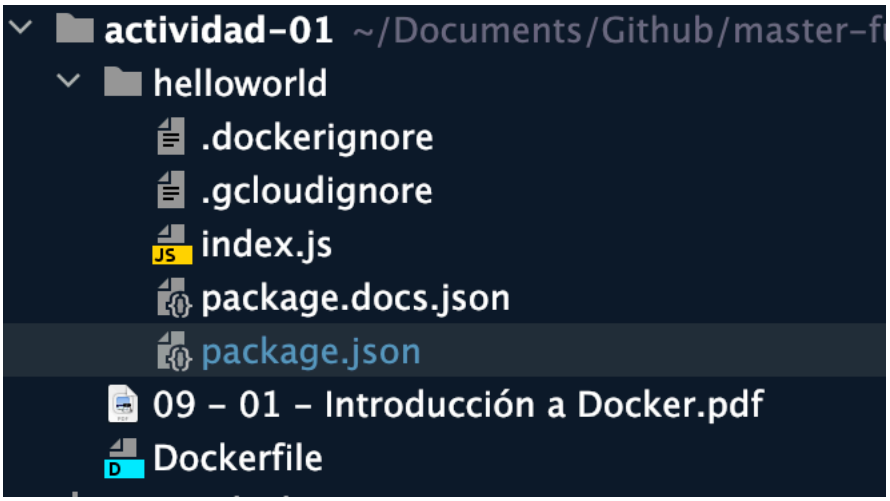
Antes que continuar realizo el siguiente análisis:

- El contenido a cambiar se encuentra en el archivo “index.js” de “Práctica 1”
- Si utilizo **bind mount** a nivel de la **carpeta**, tendré que instalar las dependencias **NPM** en mi máquina ya que no estaré usando las que se encuentran en la imagen (porque viven a nivel de la carpeta), por lo tanto decido utilizar bind mount sobre el **archivo index.js**
- Actualmente, el proyecto corre con node, pero esto impide que se vean cambios en tiempo real si realizo alguna modificación en index.js. En consecuencia, para garantizar el ejercicio, incluyo **nodemon** a mi lista de dependencias en el npm y lo utilizo para iniciar la aplicación en lugar de **node**

Adjunto capturas de las desiciones técnicas anteriormente nombradas en package.json



Estructura de carpeta por la que usaré bind mount en index.js, como se aprecia, los node_modules no se consumirán desde mi local



Me posiciono en proyecto “helloworld”

```
[  5:20PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~/Documents/Github/master-fullstack/09-contened
es/actividad-01(mainx) ]
 $ cd helloworld
[  5:20PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~/Documents/Github/master-fullstack/09-contened
es/actividad-01/helloworld(mainx) ]
 $ pwd
/Users/andresmaldonado/Documents/Github/master-fullstack/09-contenedores/actividad-01/helloworld
```

Levanto la imagen configurando como bind mount únicamente el archivo “index.js” que editaré en este ejercicio:

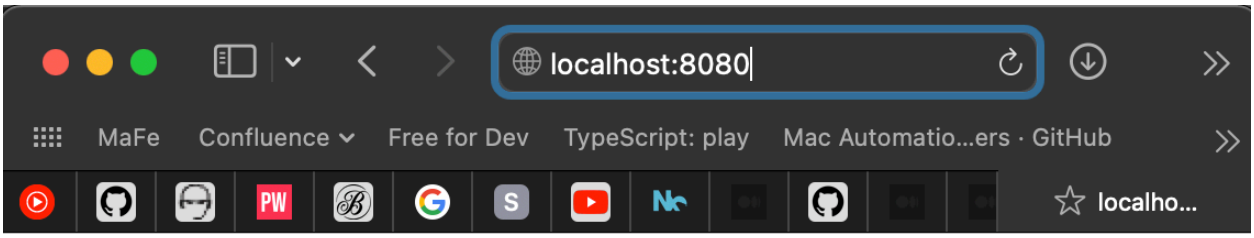
```
[  5:22PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~/Documents/Github/master-fullstack/09-contenedor
es/actividad-01/helloworld(mainx) ]
 $ docker run -it -p 8080:8080 -v $(pwd)/index.js:/src/app/index.js imagen_practica_01


> helloworld@1.0.0 start
> nodemon index.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
helloworld: listening on port 8080
```

La instancia se encuentra exitosamente corriendo en Docker exponiendo el puerto 8080

```
Last login: Fri Mar  4 16:58:31 on ttys002
[  5:24PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~/Documents/Github/master-fullstack/09-contenedores/actividad-01/helloworld(m
ainx) ]
 $ docker ps
CONTAINER ID   IMAGE               COMMAND       CREATED              STATUS              PORTS                    NAMES
71319f1f384d   imagen_practica_01   "npm start"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp   great_wilson
```



Hello World!

Cambio el código en index.js con mi nombre como se solicita

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  const name = 'Andrés Maldonado';
  res.send(`Hello ${name}!`);
});
```

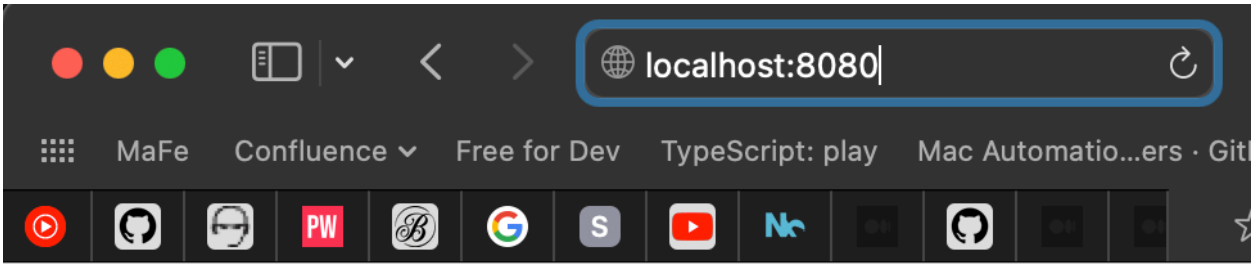
La instancia **NODEMON** que corre actualmente en Docker, escucha el cambio en el archivo y vuelve a lanzar el servidor

```
[  5:22PM ]  [ andresmaldonado@MacBook-Pro-de-And
ainx) ]
 $ docker run -it -p 8080:8080 -v $(pwd)/index.js


> helloworld@1.0.0 start
> nodemon index.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
helloworld: listening on port 8080
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
helloworld: listening on port 8080
```

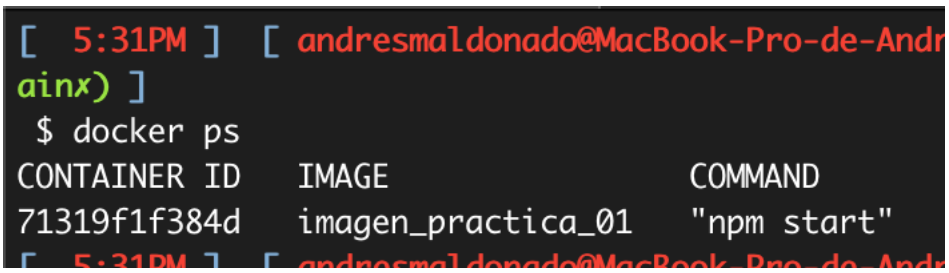
Mi localhost:8080 responde esta vez con mi nombre en pantalla, demostrando así que lee el index.js de mi máquina y no el de Docker



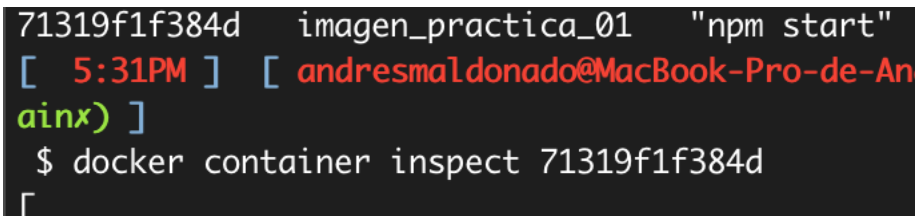
Hello Andrés Maldonado!

Finalmente, se solicita un la *"Salida de 'docker volume inspect <container>"* yo asumo que dado el contexto del ejercicio, se requiere "docker **container** inspect" en lugar de "docker **volume** inspect"

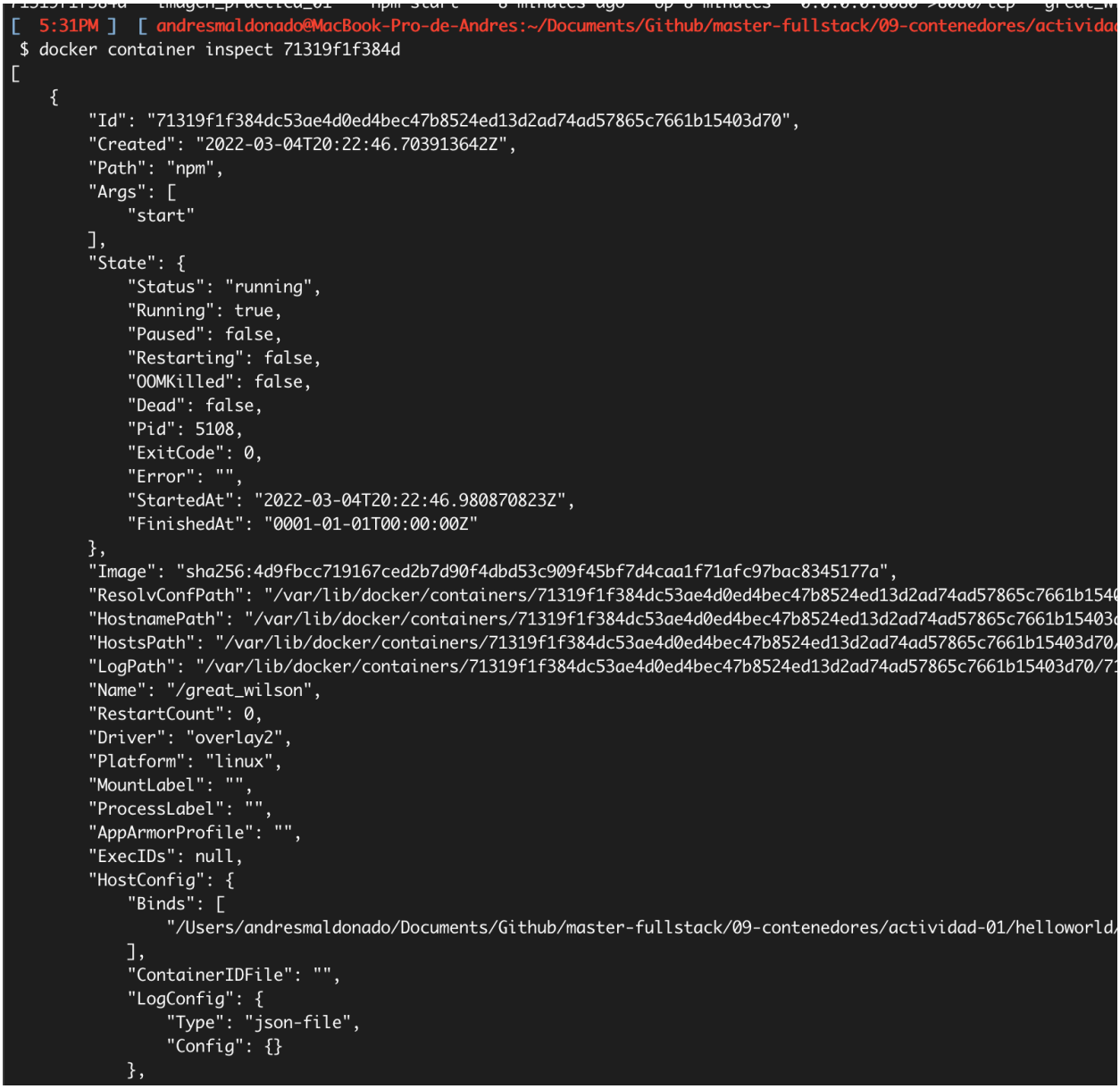
Primero, obtengo el container ID



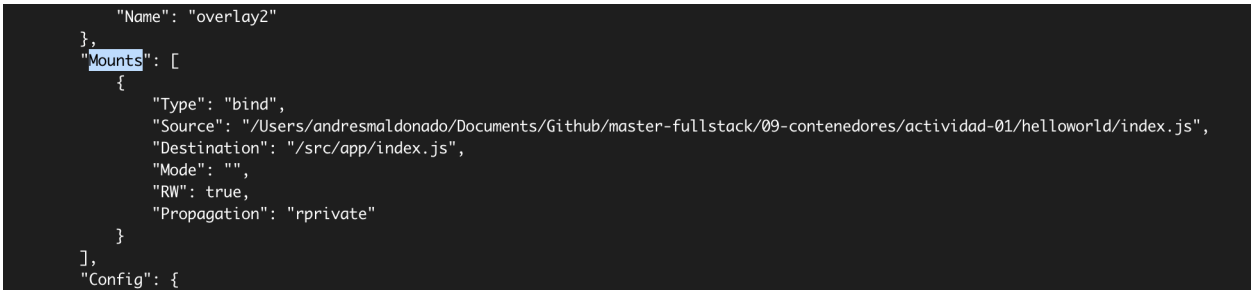
Procedo a ejecutar el comando "docker container inspect 71319f1f384d"



Esto devuelve un enorme JSON



Considerando el contexto del ejercicio, adjunto captura de “**Mounts**” para demostrar que se trabajó con **bind mount** en **index.js**



3 - PERSISTENCIA Y DOCKER COMPOSE

Docker Compose, build image y Persistencia

- *Realizar un Docker Compose que sea capaz de levantar el servicio de Node realizado en la práctica anterior, construyendo el mismo con el propio Docker Compose, no usando una imágen ya existente. Este servicio deberá incluir un Bind Mount, que permita alterar el contenido del archivo index.js.*
- *El contenido de este ejercicio deberá ser la carpeta conteniendo el Dockerfile, el código de node, así como el docker-compose.yml.*

*** ***NOTA*** ***

No entiendo por qué se solicita un Dockerfile si se solicita *"construyendo el mismo con el* ***propio Docker Compose*** *no usando una imagen ya existente"* pero se dejará de todas formas dicho Dockerfile en la carpeta

Si lo que se buscaba era crear la imagen en docker-compose a través de un Build, dejo captura de como hacerlo acá

En cualquier caso, esta opción o la que expongo llevan al mismo resultado

Como solicitado, se construye el proyecto "*construyendo el mismo con el **propio Docker Compose***" y decido utilizar como base del dockerhub un alpine:3.15

```yaml
version: "3.9"

services:
  custom_node:
    image: alpine:3.15
    container_name: only_docker_compose_container
    volumes:
      - ./helloworld/package.json:/helloworld/package.json
      - ./helloworld/index.js:/helloworld/index.js
    ports:
      - 8080:8080
    command: sh -c "
      apk add --update nodejs npm
      && npm i --prefix /helloworld
      && npm start --prefix /helloworld
      "
```

Al no poder utilizar una imagen personalizada, no puedo utilizar el comando para copiar los archivos como lo haría en **Dockerfile**. Sin embargo, puedo utilizar volúmenes para permitirle a mi instancia de docker acceder a mis archivos.
Por una parte, necesito que el contenedor lea "**package.json**" para instalar las dependencias y controlar el inicio de la aplicación. Por otra parte, debo poder alterar "**index.js**" y mi contenedor debe escuchar dichos cambios.

Para responder al enunciado del ejercicio, decido crear 2 volúmenes con las rutas específicas de los elementos que participan, de esta manera los node_modules quedarán instalados en Docker y no en mi local pero podré alterar a voluntad "**index.js**"
Igualmente configuro los puertos **8080** ya que mi proyecto corre ahí y lo expongo a mi host.

Finalmente, utilizo un comando para:

- Instalar las dependencias del sistema (**nodejs** y **npm**)
- Instalar las dependencias del proyecto vía **npm**
- Lanzar el proyecto con **npm start**.

Ahora en mi carpeta, utilizo en comando “docker-compose up”

```
[  6:53PM ]  [ andresmaldonado@MacBook-Pro-de-Andres:~/Documents/Github/master-fullstack/09-contenedores/actividad-
 $ docker-compose up
Recreating only_docker_compose_container ... done
Attaching to only_docker_compose_container
only_docker_compose_container | fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/main/x86_64/APKINDEX.tar.gz
only_docker_compose_container | fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/community/x86_64/APKINDEX.tar.gz
only_docker_compose_container | (1/8) Installing ca-certificates (20211220-r0)
only_docker_compose_container | (2/8) Installing nghttp2-libs (1.46.0-r0)
only_docker_compose_container | (3/8) Installing brotli-libs (1.0.9-r5)
only_docker_compose_container | (4/8) Installing c-ares (1.18.1-r0)
only_docker_compose_container | (5/8) Installing libgcc (10.3.1_git20211027-r0)
only_docker_compose_container | (6/8) Installing libstdc++ (10.3.1_git20211027-r0)
only_docker_compose_container | (7/8) Installing nodejs (16.14.0-r0)
only_docker_compose_container | (8/8) Installing npm (8.1.3-r0)
only_docker_compose_container | Executing busybox-1.34.1-r3.trigger
only_docker_compose_container | Executing ca-certificates-20211220-r0.trigger
only_docker_compose_container | OK: 58 MiB in 22 packages
only_docker_compose_container |
only_docker_compose_container | added 290 packages, and audited 291 packages in 14s
only_docker_compose_container |
only_docker_compose_container | 47 packages are looking for funding
only_docker_compose_container |   run `npm fund` for details
only_docker_compose_container |
only_docker_compose_container | found 0 vulnerabilities
only_docker_compose_container |
only_docker_compose_container | > helloworld@1.0.0 start
only_docker_compose_container | > nodemon index.js
only_docker_compose_container |
only_docker_compose_container | [nodemon] 2.0.15
only_docker_compose_container | [nodemon] to restart at any time, enter `rs`
only_docker_compose_container | [nodemon] watching path(s): *.*
only_docker_compose_container | [nodemon] watching extensions: js,mjs,json
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
```

Se levanta el proyecto con éxito y está escuchando en el puerto 8080

NOTA: hasta este punto, se llega al mismo resultado con el Dockerfile

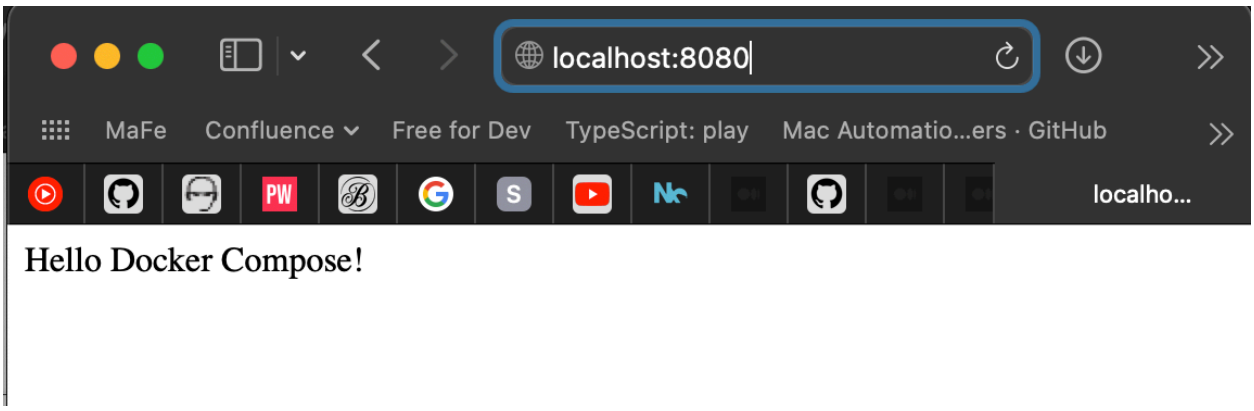
Realizo un cambio en mi archivo “index.js”

```
app.get('/', (req, res) => {
  const name = 'Docker Compose';
  res.send(`Hello ${name}!`);
});
```

Con dicho cambio, se refresca nodemon

```
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
only_docker_compose_container | [nodemon] restarting due to changes...
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
```

Mi local muestra “Hello Docker Compose!”



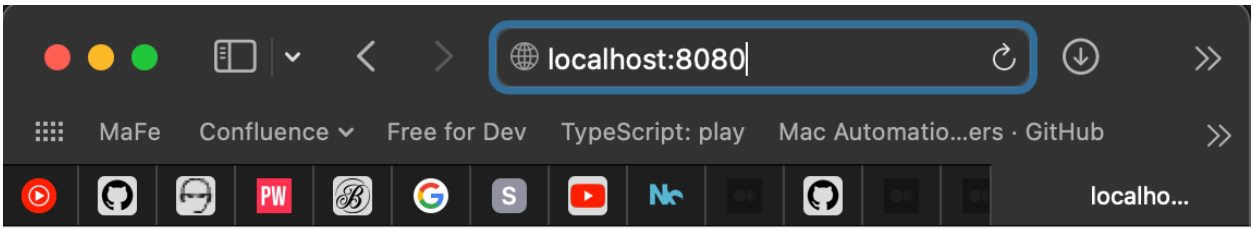
Realizo otro cambio en el mensaje

```
app.get('/', (req, res) => {
  const name = 'From My Computer';
  res.send(`Hello ${name}!`);
});
```

Nodemon vuelve a refrescar la instancia

```
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
only_docker_compose_container | [nodemon] restarting due to changes...
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
only_docker_compose_container | [nodemon] restarting due to changes...
only_docker_compose_container | [nodemon] starting `node index.js`
only_docker_compose_container | helloworld: listening on port 8080
```

El servidor responde con este nuevo mensaje



Hello From My Computer!

4 - PERSISTENCIA Y DOCKER COMPOSE

Realizar un Docker Compose de Drupal y Postgres.

En esta última actividad trabajaremos en preparar un servidor de Drupal y Postgres orquestados dentro de Docker Compose. Este Docker Compose tendrá las siguientes características adicionales:

- *El servicio debe exponerse en el puerto 32400 del hosts*
- *La base de Postgres deberá tener un volumen llamado "drupal-db"*
- *Ambos deberán estar dentro de una red llamada "drupal-network"*

El contenido de este ejercicio deberá ser una carpeta llamada docker-compose_2, con un docker- compose.yml conteniendo la solución al ejercicio.

Se crea una base de datos prostgres según lo solicitado, con red "drupal-network" y data en volumen drupal-db

```yaml
version: '3.9'

services:
  postgres:
    image: postgres
    restart: always
    environment:
      POSTGRES_DB: postgresdb
      POSTGRES_USER: postgresuser
      POSTGRES_PASSWORD: postgrespassword
    volumes:
      - drupal-db:/var/lib/postgresql/data
    ports:
      - 5432:5432
    networks:
      - drupal-network
```

Se vinculan los volúmenes que participan de acuerdo al script de ejemplo. Se asocia a la red “drupal-network”. Finalmente, se expone en puerto 32400 según lo solicitado y como adicional se genera dependencia con la instancia de postgres.

```
drupal:
  image: drupal
  restart: always
  ports:
    - 32400:80
  volumes:
    - drupal-modules:/var/www/html/modules
    - drupal-profiles:/var/www/html/profiles
    - drupal-sites:/var/www/html/sites
    - drupal-themes:/var/www/html/themes
  depends_on:
    - postgres
  networks:
    - drupal-network
```

Al final, se declaran los volúmenes que participan y se configura el driver “bridge” para el network solicitado

```
volumes:
  drupal-db:
  drupal-modules:
  drupal-profiles:
  drupal-sites:
  drupal-themes:

networks:
  drupal-network:
    driver: bridge
```