101

course[IT]

Juani Gallo

@JuaniGallo

Programador
Emprendedor

Founder

course[It]

1. Que es Docker
2. Como escribir un Dockerfile
3. Manejo de contenedores e imagenes

course[It]

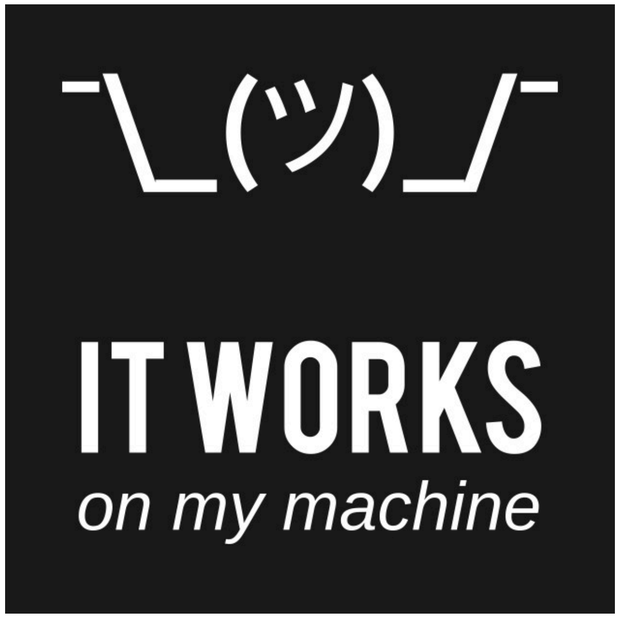
¿Qué es Docker?

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Docker crea una computadora virtual adentro de tu computadora con todo lo necesario para correr tu aplicación.

¿Para que sirve Docker?

- Para correr nuestra aplicación con la misma infraestructura que producción



- Para simplificar el onboarding de nuestra aplicación

- Levantar servicios en distintos lenguajes sin tener que instalar todo lo relacionado a cada uno de ellos

1. Para correr nuestra aplicación con la misma infraestructura que producción

2. Para simplificar el onboarding de nuestra aplicación

3. Levantar servicios en distintos lenguajes sin tener que instalar todo lo relacionado a cada uno de ellos

Conceptos básicos

Container

Un contenedor es una pieza de software que contiene todo lo necesario para correr una aplicación (código + dependencias).

Un container siempre tiene un estado asignado y los estados disponibles son: “**created**”, “**restarting**”, “**running**”, “**removing**”, “**paused**”, “**exited**” y “**dead**”

Created

El contenedor se creo sin errores pero no esta corriendo. Este estado es útil para cuando queremos crear un contenedor pero todavía no utilizarlo

Restarting

El contenedor se esta reiniciando. Este estado puede darse en el caso que desde un comando reiniciemos el contenedor o bien por un error relacionado a código en el mismo.

Running

Nuestro contenedor esta creado y corriendo sin ningún problema

Removing

El contenedor se esta removiendo pero todavía sigue creado, en este momento ya no podemos acceder al mismo

Paused

El contenedor esta creado pero pausado. En este estado podemos acceder al contenedor pero su código no está en ejecución

Exited

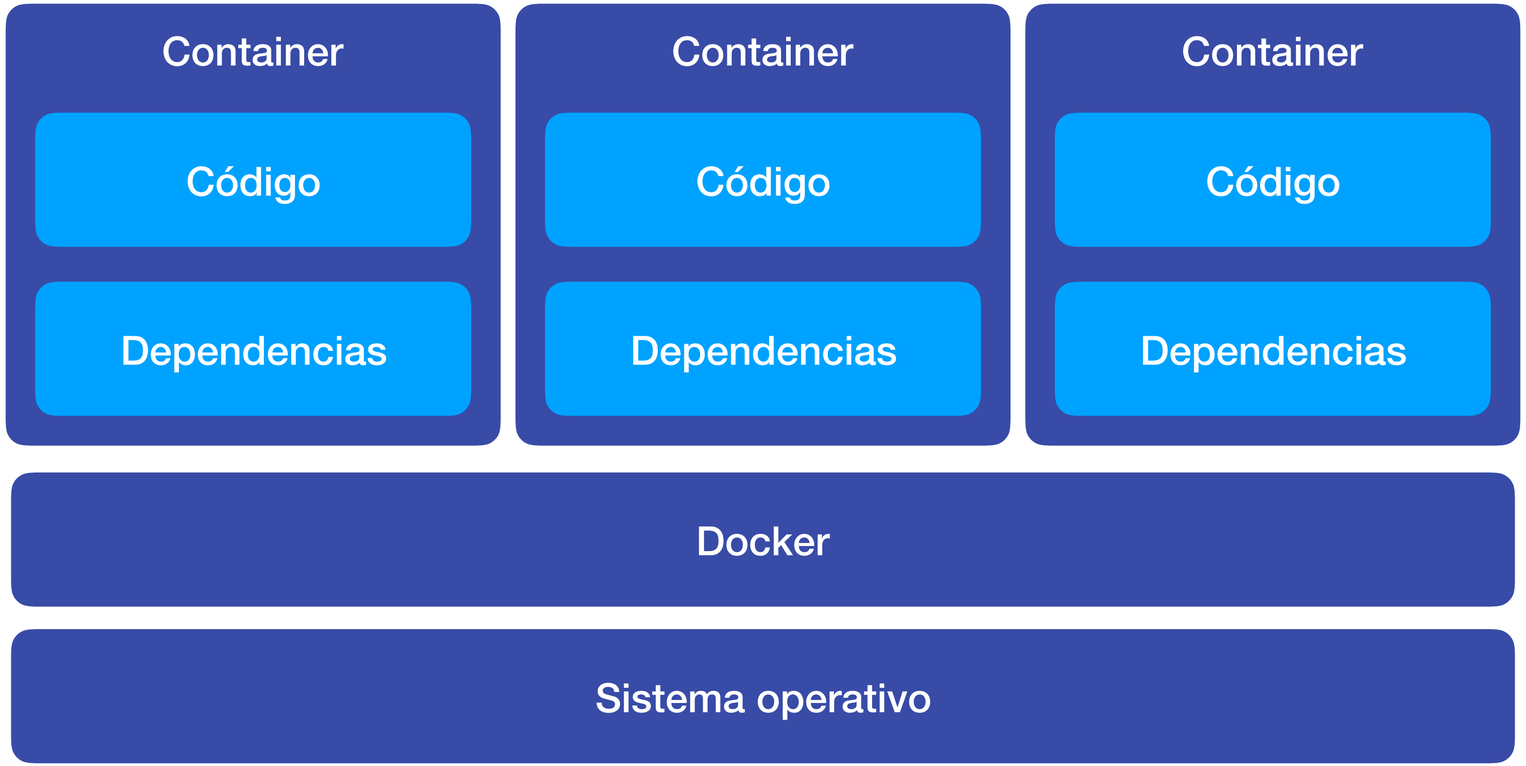
Podemos llegar a este estado porque el container termino de ejecutar su código de forma correcta o bien porque se encontró con algún problema durante su ejecución. Si su “status code” es **0** quiere decir que el container termino sin problemas, cualquier otro numero distinto de 0 implica que el container no termino su ejecución de forma correcta.

Dead

Es el estado terminal del container. Usualmente se da cuando el container pierde acceso de escritura a archivos necesarios para funcionar.

Aplicación con un único container

Aplicación con tres containers



containers

Docker crea ~~una computadora virtual~~ adentro de tu computadora con todo lo necesario para correr tu aplicación.

Image

Una imagen es una plantilla creada a partir de una serie de instrucciones para luego crear containers en base a esta. Para crear imágenes vamos a crear un archivo llamado **Dockerfile**.

En https://hub.docker.com/ podes encontrar una biblioteca con muchísimas imágenes públicas (por ejemplo: Ubuntu, NGINX, Mongo, Node)

Instalemos Docker

MacOS

Desde MacOS Sierra 10.12 para arriba

https://hub.docker.com/editions/community/docker-ce-desktop-mac

Si tiene un OS más antiguo que Sierra 10.12, hay que instalar Docker Toolbox

https://docs.docker.com/toolbox/overview/

Ubuntu

```
sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl
gnupg-agent software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -

sudo apt-key fingerprint 0EBFCD88

sudo add-apt-repository "deb [arch=amd64] https://
download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io
```

https://cutt.ly/nerdearla-docker

Otras distros de Linux

Ubuntu: https://docs.docker.com/install/linux/docker-ce/ubuntu/

Debian: https://docs.docker.com/install/linux/docker-ce/debian/

Fedora: https://docs.docker.com/install/linux/docker-ce/fedora/

CentOS: https://docs.docker.com/install/linux/docker-ce/centos/

Windows 7, 8 o 10 versión “Pro” 64 bits

https://hub.docker.com/

Hay que crearse una cuenta y descargar el Docker Desktop

Windows 7, 8 o 10 versión “Home” 64 bits

Para versiones que no son Pro o versiones viejas de Windows, hay que instalar el Docker Toolbox

https://github.com/docker/toolbox/releases/download/v19.03.1/DockerToolbox-19.03.1.exe

El Docker Toolbox instala 3 archivos, Docker Quickstart, Oracle VM y Kitematic.

Una vez instalado hay que activar el Hyper-V en la BIOS de la PC.

Activar Hyper-V (Windows)

En todas las PCs se entra al BIOS de una manera diferente, HP suele ser F1, Lenovo F8, muchas otras es Supr, hay que apretar este botón varias veces cuando apenas se prende la PC. Vamos a entrar a una pantalla azul con varias opciones, cada PC tiene su propia configuración, lo que vamos a tener que buscar es la opción “Security” o “Configuration”, una vez dentro vamos a tener que pasar a “enable” la opción que diga: “Hyper-V”, puede también llamarse “Virtualization”, “Intel Vitualization” o “VTX”.

Una vez que Hyper-V está activado, ya podemos volver a iniciar el Windows y abrir el ejecutable: “Docker Quickstart”.

https://courseit.com.ar/static/docker.pdf

Dockerfile

¿Qué es un Dockerfile?

Es un documento de texto que contiene todos los **comandos** que vamos a ejecutar a la hora de crear nuestra imagen. Se podría decir que nuestro Dockerfile va a ser la receta que Docker va a seguir para poder crear nuestra imagen.

FROM

El comando **FROM** nos va a servir para basar nuestra nueva imagen en una imagen ya existente (las podemos ver en: https://hub.docker.com). Este comando **siempre** tiene que ser el primero en nuestro Dockerfile

```
FROM node:11
```

RUN

El comando **RUN** nos va a servir para correr comandos en una terminal dentro de nuestro *container.* Esto es util para cambiar configuración a nivel sistema operativo o bien instalar paquetes de forma global.

```
RUN npm install -g pm2 --silent
```

CMD

El comando **CMD** nos va a servir para indicarle a nuestra *imagen* que comando tiene que correr por defecto al crear nuestro *container.* En caso de que nuestro comando contenga parámetros hay que escribir el comando en formato de *array[]*

```
CMD ["node", "server.js"]
```

EXPOSE

El comando **EXPOSE** nos va a servir para indicarle a nuestro container que puerto escuchar mientras este corriendo. Esto es especialmente útil para cuando estamos corriendo un servidor.

```
EXPOSE 3000
```

WORKDIR

El comando **WORKDIR** nos va a servir para indicarle a nuestra imagen que directorio tiene que usar como base para los comandos que modifiquen el sistema de archivos. En caso de que no exista el directorio indicado, Docker lo va a crear.

```
WORKDIR /usr/src/app
```

COPY

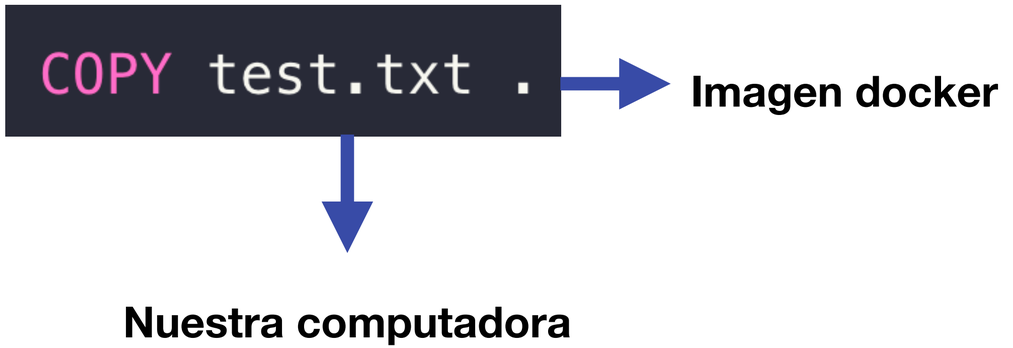
El comando **COPY** nos va a servir para copiar archivos desde nuestra computadora a nuestra imagen de Docker. Con el comando **COPY** podemos copiar archivos o directorios completos.

```
COPY test.txt .
```

```
COPY . .
```

COPY

El comando **COPY** nos va a servir para copiar archivos desde nuestra computadora a nuestra imagen de Docker. Con el comando **COPY** podemos copiar archivos o directorios completos.




```
COPY . .
```

course[It]

ENV

El comando **ENV** nos va a servir para crear variables de ambiente dentro de nuestra imagen. Por ejemplo para indicarle si estamos en un ambiente de testing o productivo.

```
ENV NODE_ENV production
```

¿Qué hace este Dockerfile?

```
1
2    FROM node:10.15-alpine
3
4    WORKDIR /usr/src/app
5
6    RUN npm install  --silent
7
8    COPY . .
9
10   RUN npm run build
11
12   EXPOSE 3001
13
14   CMD ["npm", "start"]
```

course[It]

¿Qué hace este Dockerfile?

```
# Buscamos la imagen de node version 10.15
FROM node:10.15-alpine
# Establecemos una carpeta de trabajo
WORKDIR /usr/src/app
# Instalamos las dependencias
RUN npm install  --silent
# Copiamos todos los archivos
COPY . .
# Ejecutamos el comando de npm build
RUN npm run build
# Exponemos el puerto 3001
EXPOSE 3001
# Corremos nuestra imagen con el comando npm start
CMD ["npm", "start"]
```

Se rompe

¿Qué hace este Dockerfile?

```
# Buscamos la imagen de node version 10.15
FROM node:10.15-alpine
# Establecemos una carpeta de trabajo
WORKDIR /usr/src/app
# Instalamos las dependencias
RUN npm install  --silent
# Copiamos todos los archivos
COPY . .
# Ejecutamos el comando de npm build
RUN npm run build
# Exponemos el puerto 3001
EXPOSE 3001
# Corremos nuestra imagen con el comando npm start
CMD ["npm", "start"]
```

course[IT]

https://github.com/juanigallo/bad-dockerfile-nerdearla

https://labs.play-with-docker.com/

Instrucciones básicas

docker ps

El comando **docker ps** nos sirve para listar todos los contenedores que actualmente estén corriendo.

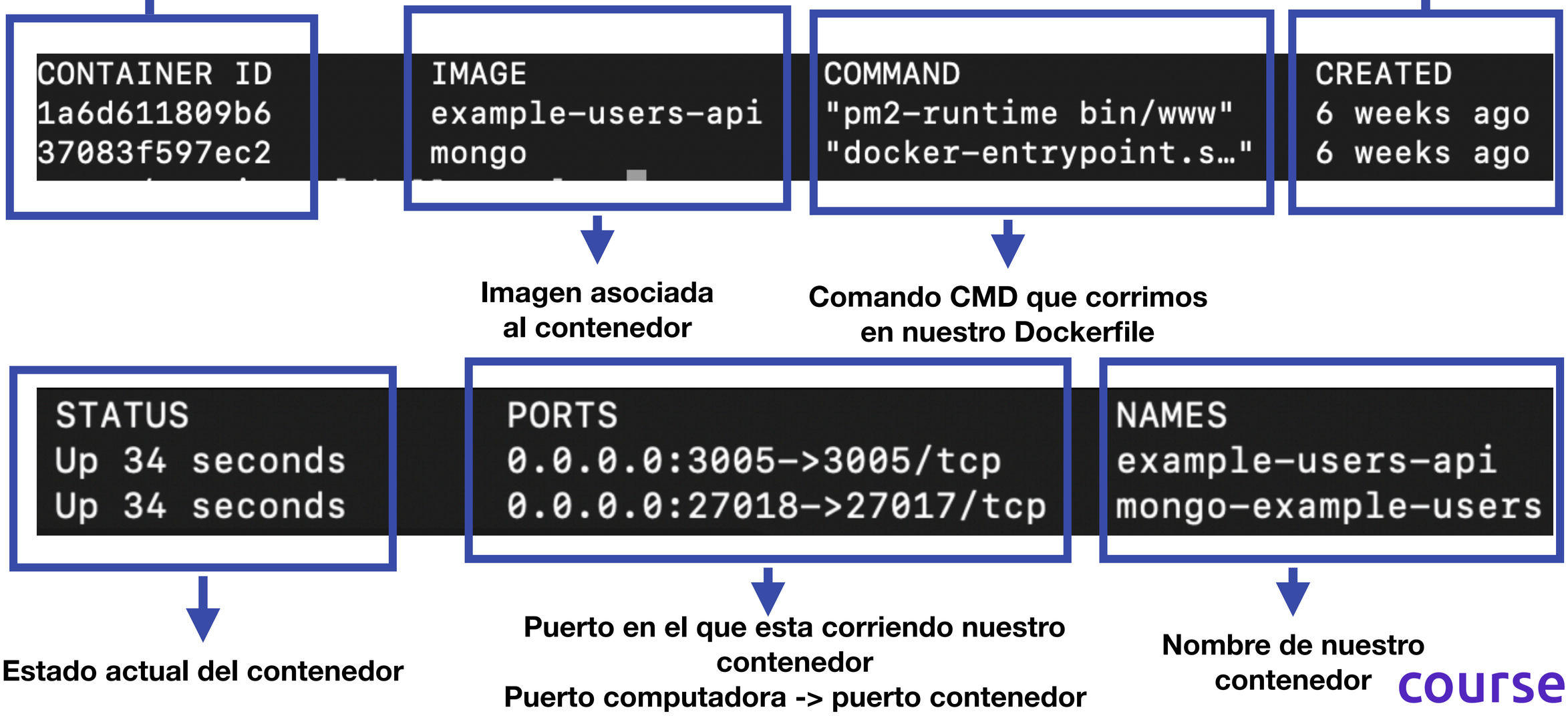
En caso de querer ver los contenedores que ya terminaron o murieron en el pasado, podemos correr el comando **docker ps -a**

docker ps

```
CONTAINER ID        IMAGE                  COMMAND                  CREATED
1a6d611809b6        example-users-api      "pm2-runtime bin/www"    6 weeks ago
37083f597ec2        mongo                  "docker-entrypoint.s…"   6 weeks ago
```

```
STATUS              PORTS                      NAMES
Up 34 seconds       0.0.0.0:3005->3005/tcp     example-users-api
Up 34 seconds       0.0.0.0:27018->27017/tcp   mongo-example-users
```

docker ps



course[It]

docker build

El comando **docker build** nos sirve para crear una imagen en base a un Dockerfile

El uso *normal* del comando es: **docker build .** lo que nos va a generar una imagen en base a un Dockerfile situado en esa misma carpeta.

Para crear una imagen

```
FROM node:10.15-alpine
ENV NODE_ENV ci
WORKDIR /usr/src/app
RUN npm install --production --silent
COPY . .
RUN npm run build
EXPOSE 3001
CMD ["node", "server.js"]
```

+

```
docker build .
```

course[IT]

docker build

```
Sending build context to Docker daemon   15.8MB
Step 1/10 : FROM node:10.15-alpine
 ---> 288d2f688643
Step 2/10 : ENV NODE_ENV production
 ---> Using cache
 ---> 0cfea731a3f6
Step 3/10 : WORKDIR /usr/src/app
 ---> Using cache
 ---> f9c39ce52759
```

...

```
Successfully built c2aa558e8f74
```

docker images

El comando **docker images** nos sirve para listar todas las imágenes que creamos

docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	c2aa558e8f74	27 minutes ago	140MB
example-users-api	latest	0a737297ee55	6 weeks ago	140MB
mongo	latest	0fb47b43df19	7 weeks ago	411MB

docker images

El nombre de nuestra imagen

Hace cuanto fue creada nuestra imagen

El tag/version de nuestra imagen

El identificador unico de nuestra imagen

Peso de nuestra imagen

course[IT]

docker run

El comando **docker run** nos sirve para crear un container a partir de una imagen

```
docker run -p PUERTO_EXPOSE:SERVIDOR IMAGE_ID
```

```
docker run -p 3000:3000 05a56348ea50
```

Proyecto a Dockerizar

GitHub

https://github.com/juanigallo/actividad-docker-nerdearla

¡A codear! 💪

Slides

https://courseit.com.ar/static/docker.pdf