CFGS ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

BIENVENIDOS!

Manuel Domínguez.

mftienda@gmail.com

@mafradoti

www.linkedin.com/in/mftienda

https://github.com/mftienda

Ud.- Creación de imágenes propias

Índice

1.- Introducción

Si queremos hacer una imagen de nuestra aplicación y distribuirla,

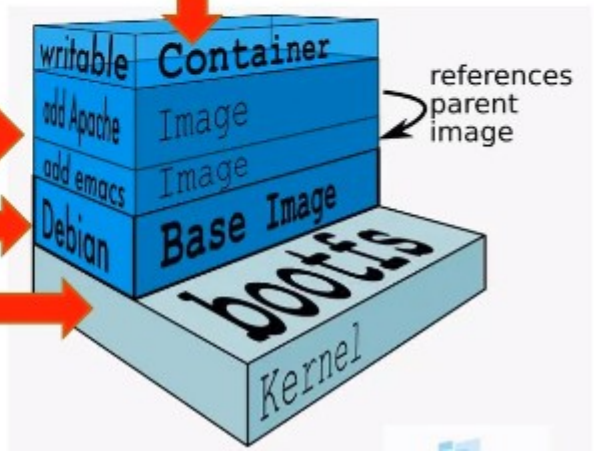
¿Cómo lo hacemos?

¿Cómo creamos nuestras propias imágenes?

1.- Introducción

Capas de una imagen:

- **Imágenes en Docker**
  - Las imágenes en Docker están formadas por varias capas solo de lectura.
  - Por ejemplo:
    - Monta una capa por encima (lectura/escritura) → el contenedor
    - Monta las n capas de imagen (lectura)
    - Monta el sistema de ficheros de root: rootfs (lectura)
    - Monta el sistema de ficheros de arranque: bootfs (lectura)



2.- Creación de imágenes manualmente.

Una primera idea seria:

1.- Elegimos una imagen base de un repositorio.

2.- Añadimos las aplicaciones que nos interese.

3.- La convertimos en una imagen: **docker commit ……..**

4.- La subimos a un repositorio público.

2.- Creación de imágenes manualmente.

Ejemplo:

1.- Creamos el contenedor: # docker run -it --name mi-debian debian bash

2.- Lo actualizamos: #apt update

3.- Instalamos wget: # apt install wget

4.- Instalamos curl: # apt install curl

wget y curl son dos herramientas para descargar archivos. Además curl lo podemos utilizar para comprobar si es accesible una página web.

5.- Lo comprobamos: # wget google.es → Se descarga el index.html

6.- Lo comprobamos # curl google.es → Nos dice si es accesible.

7.- Lo paramos: #docker stop mi-debian

2.- Creación de imágenes manualmente.

Ejemplo:

8.- Para ver los cambios en un contenedor:

#docker diff mi-debian

A → Creación de archivo.

C→ Cambio.

D→ Eliminación.

```
root@debian:~# docker diff mi-ubuntu-1|grep wget
A /etc/wgetrc
A /usr/share/doc/wget
A /usr/share/doc/wget/copyright
A /usr/share/doc/wget/changelog.Debian.gz
A /usr/share/info/wget.info.gz
A /usr/bin/wget
```

2.- Creación de imágenes manualmente.

9.- Creamos la imagen de forma manual:

 # docker commit mi-debian debian-manolo

mi-debian → Nombre del contenedor

debian-manolo → Nombre que le ponemos a la imagen.

10.- Comprobamos: # docker images

11.- Creamos un contenedor a partir de esa imagen y comprobamos que está wget y curl

```
root@debian:~# docker run -it debian-manolo bash
root@6624d2fc7c55:/# apt policy curl
curl:
  Installed: 7.64.0-4+deb10u1
  Candidate: 7.64.0-4+deb10u1
  Version table:
 *** 7.64.0-4+deb10u1 500
        500 http://deb.debian.org/debian buster/main amd64 Packages
        500 http://security.debian.org/debian-security buster/updates/main amd64
 Packages
        100 /var/lib/dpkg/status
root@6624d2fc7c55:/#
```

12.- Ya solo nos quedaría subirla al repositorio de imágenes privado o público.

2.- Creación de imágenes manualmente.

Ahora, bien esta forma de crear las imágenes **tiene un problema:**

Si alguién quiere reproducirla:

1.- No sabe bien qué hay de esa imagen.

2.- No tiene un mecanismo para reproducirla o modificarla.

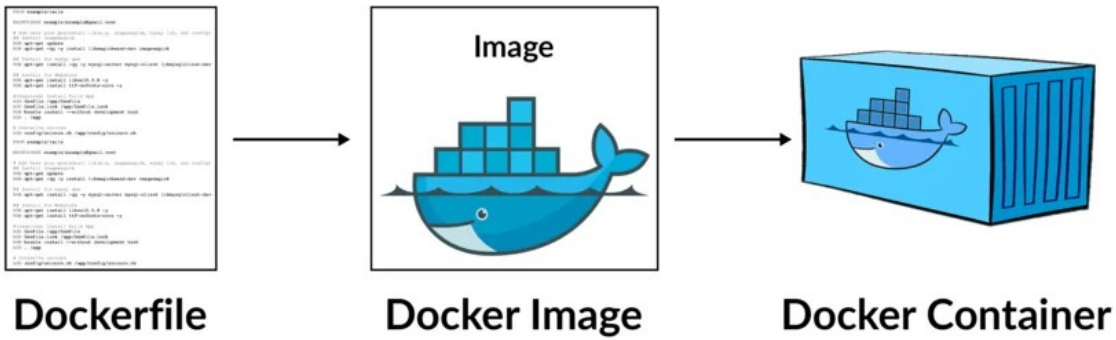
3.- Creación automática de imágenes. Dockerfile

Creación de imágenes automatizadas

Docker nos permite **automatizar** el proceso de **creación de imágenes.**

Para ello utilizaremos un fichero llamado **Dockerfile.**

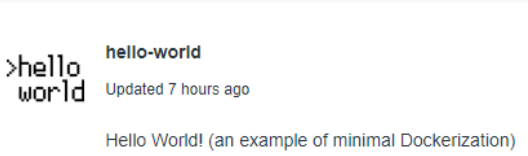
Dockerfile es un archivo donde se detalla todo el proceso a seguir en la creación de la imagen.



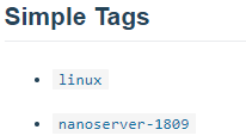
3.- Creación automática de imágenes. Dockerfile

Dockerfile en Docker Hub: hello-world

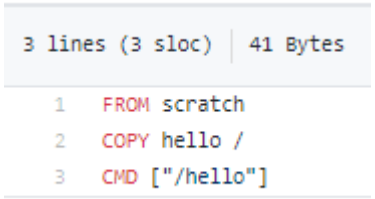
1.- Accedemos a Docker Hub y buscamos esta imagen.



2.- Pinchamos sobre ella y después en Linux.



3.- Veremos en Github, el Dockerfile:



Cada línea de ese fichero, nos creará una capa.

4.- Ver las capas de una imagen:

#docker image history Nombre-Imagen

3.- Creación automática de imágenes. Dockerfile

Procedimiento Dockerfile

1.- Creamos una carpeta para nuestro proyecto: docker-miweb

2.- Introducimos los archivos de nuestra aplicación.

2.- Accedemos a ella y nos creamos el archivo Dockerfile: Vim Dockerfile

```
FROM httpd:latest
MAINTAINER Manuel Dominguez
RUN ……..
```

4.- Para crear la imagen: **docker build -t miweb:latest .**

t → Indicamos el nombre de la imagen

. → Indicamos que el Dockerfile se encuentra en el directorio actual.

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

FROM: indicamos la imagen base.
MAINTAINER: indicamos el creador de la imagen.

RUN: Ejecuta un comando **durante la construcción de la imagen.**

3.- Creación automática de imágenes. Dockerfile

Ejemplo: FROM y RUN

1.-# mkdir -p docker/prueba

2.- Escribimos el Dockerfile dentro de prueba

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
RUN mkdir /datos
```

(Utilizar espacios, no tabulaciones)

3.- Construimos la imagen a partir del Dockerfile: **#docker build -t mi-imagen:v1 .**

4.- Comprobamos las capas de esa imagen: **docker image history mi-imagen:v1**

5.- Construimos un contenedor con esa imagen: #d**ocker run -it mi-imagen:v1 bash**

6.- Comprobamos que tiene instalado, por ejemplo el git

3.- Creación automática de imágenes. Dockerfile

CMD |ENTRYPOINT: Indicamos los comandos que se van a ejecutar una vez **inicializado el contenedor.**

ENTRYPOINT indica la aplicación principal que se desea ejecutar, y CMD para indicar los parámetros que le acompaña.

Si sólo utilizamos CMD, entonces Docker ejecutará dicho comando usando el ENTRYPOINT por defecto, que es **/bin/sh -c.**

Con **CMD y ENTRYPOINT** estamos diciendo cuál es el **proceso principal del contenedor.**

3.- Creación automática de imágenes. Dockerfile

Ejemplo: CMD

1.- Modificamos el Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
CMD echo "Welcome to this container"
```

2.- Creamos la imagen: **#docker build -t mi-imagen:v2 .**

3.- # docker images

4.- #docker run -it mi-imagen:v2 → Creamos el contenedor

3.- Creación automática de imágenes. Dockerfile

Ejemplo: CMD [Formato json]

1.- Modificamos el Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
CMD ["echo","Welcome to this container"]
```

2.- Creamos la imagen: **#docker build -t mi-imagen:v3 .**

3.- # docker images

4.- #docker run -it mi-imagen:v3 → Creamos el contenedor

3.- Creación automática de imágenes. Dockerfile

Observaciones CMD:

¿Qué diferencia existe entre:?

```
CMD echo "Welcome to this container"

CMD ["echo","Welcome to this container"]
```

La diferencia es que la primera se ejecuta sobre una bash (/bin/bash) y el segundo no. Puede ser que la imagen que estemos arrancando no tenga bash**. Lo mejor es utilizar el segundo formato, Json.**

3.- Creación automática de imágenes. Dockerfile

DIFERENCIA ENTRE CMD Y ENTRYPOINT

1.- Modificamos el Dockerfile → CMD [“/bin/bash”] → Nos abrirá un bash.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
CMD ["/bin/bash"]
```

2.- Creamos la imagen: **#docker build -t mi-imagen:v4 .**

3.- # docker images

4.- **#docker run -it mi-imagen:v4** → Creamos el contenedor

5.- Aparece la bash → OK

DIFERENCIA ENTRE CMD Y ENTRYPOINT

6.- Pero si creo el contenedor de la siguiente forma:

#docker run -it mi-imagen:v4 ls → El contenedor ejecuta el comando ls y se sale.

Con esto, estamos sustituyendo el comando que pusimos en el CMD [“/bin/bash”] por el ls.

Esto no ocurre con **ENTRYPOINT**

3.- Creación automática de imágenes. Dockerfile

DIFERENCIA ENTRE CMD Y ENTRYPOINT

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
ENTRYPOINT["/bin/bash"]
~
```

2.- Creamos la imagen: **#docker build -t mi-imagen:v4 .**

3.- # docker images

4.- **#docker run -it mi-imagen:v4** → Creamos el contenedor

5.- Aparece la bash → OK

6.- Repetimos: **#docker run -it mi-imagen:v4** ls → No lo entiende

3.- Creación automática de imágenes. Dockerfile

Si definimos un `ENTRYPOINT`, lo que se defina como `CMD` será tomado como argumento para el ejecutable del `ENTRYPOINT`.

```
FROM debian:jessie-slim

RUN apt-get update                                  && \
    apt-get install -y --no-install-recommends      \
        cowsay                                      \
        screenfetch                             &&  \
    rm -rf /var/lib/apt/lists/*

ENV PATH "$PATH:/usr/games"

CMD ["Yo, CMD!!"]
ENTRYPOINT ["cowsay", "Yo, Entrypoint!!"]
```

```
 ____________________________
< Yo, Entrypoint!! Yo, CMD!! >
 ----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

3.- Creación automática de imágenes. Dockerfile

CONCLUSIONES: CMD Y ENTRYPOINT

Tanto CMD como ENTRYPOINT nos permite indicar qué comando se va a ejecutar **una vez inicializado el contenedor.**

CMD → Puede ser sustituido en la creación del contenedor y con ENTRYPOINT, No.

Se puede combinar, tomando ENTRYPOINT como proceso principal y CMD como argumentos del proceso principal.

3.- Creación automática de imágenes. Dockerfile

Observaciones:

¿Puede existir más de un CMD o ENTRYPOINT

Sí, pero solo se tiene en cuenta el último introducido.

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

WORKDIR: Nos situamos en una carpeta dentro del contenedor,

Es como un “cd”

Es el punto de partida para los comandos: RUN , CMD , ENTRYPOINT.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
RUN mkdir /datos
WORKDIR /datos
RUN touch f1.txt
RUN mkdir /datos1
WORKDIR /datos1
RUN touch f2.txt
ENTRYPOINT ["/bin/bash"]
```

3.- Creación automática de imágenes. Dockerfile

WORKDIR:

1- Creamos la imagen: **#docker build -t mi-imagen:v5 .**

2.- # docker images

3.- **#docker run -it mi-imagen:v5** → Creamos el contenedor

4.- Vemos que se ha colocado sobre el directorio /datos1 y se ha arrancado la bash.

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

COPY y ADD: nos permite añadir archivos al contenedor.

La diferencia es que ADD permite trabajar con URL y descomprime archivos en el destino.

3.- Creación automática de imágenes. Dockerfile

Ejemplo: COPY Y ADD

```
##WORKDIR##
RUN mkdir /datos
WORKDIR /datos
RUN touch f1.txt
RUN mkdir /datos1
WORKDIR /datos1
RUN touch f2.txt

##COPY##
COPY index.html .
COPY app.log /datos

##ADD##
ADD docs docs
ADD f* /datos/
ADD f.tar .

##ENTRYPOINT##
ENTRYPOINT ["/bin/bash"]
```

COPY index. html . → Se copiará en el directorio /datos1

ADD f.tar . → Descomprime el fichero f.tar en el directorio de trabajo del contenedor.

Ejercicio: ¿Dónde se guardaría el directorio docs?

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

ENV: permite declarar una variable de entorno en el contenedor.

```
##ENV##
ENV dir=/data dir1=/data1
RUN mkdir $dir [redacted] && mkdir $dir1
```

Creará el directorio /data y el directorio /data1

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

ARG: permite pasar variables en el momento crear **la imagen (--build-arg)**

```
##ENV##
ENV dir=/data dir1=/data1
RUN mkdir $dir   && mkdir $dir1

##ARG##
ARG dir2
RUN mkdir $dir2
```

No hemos definido el valor de la variable dir2. Ese valor se lo daremos al construir la imagen.

```
# docker build -t image:v6 --build-arg dir2=/data2 .
```

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

EXPOSE: Es **simplemente descriptivo**. Para él que ejecute esa imagen sepa el puerto con el que trabaja la aplicación.
Posteriormente cuando se crea el contenedor, podemos crear otros puertos.

```
##EXPOSE##
RUN apt-get install -y apache2
EXPOSE 80
```

3.- Creación automática de imágenes. Dockerfile

Estructura de Dockerfile

VOLUME:

Nos permite guardar los datos de forma persistente.

3.- Creación automática de imágenes. Dockerfile

Ejemplo: Servidor apache

1.- Nos descargamos el material complementario.

2.- Creamos la siguiente estructura de directorios: **docker/volumen/paginas**

3.- Descomprimimos el material complementario en dicha carpeta

 # unzip /home/usuario/Descargas/06.-Práctica_web_volume.zip -d /root/docker/volumen/paginas

4.- En el interior nos creamos el **Dockerfile.**

```
root@debian:~/docker/volume# ls
Dockerfile  paginas
root@debian:~/docker/volume#
```

3.- Creación automática de imágenes. Dockerfile

Ejemplo: Servidor apache

```
FROM debian
RUN apt-get update
RUN apt-get install -y apache2
EXPOSE 80
###Volúmenes###
ADD paginas /var/www/html
VOLUME ["/var/www/html"]
#########################################################
#apache es el servidor web
#apache2ctl es el script que controla al servidor apache
#########################################################
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Otra alternativa:

```
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

ADD → Copiamos el contenido de la carpeta paginas, que está en el host a /var/www/html del contenedor.

VOLUME → Nos crea un **nuevo volumen** en el host con la información de /var/www/html

3.- Creación automática de imágenes. Dockerfile

Ejemplo: Servidor apache

1.- Construimos la imagen: **#docker build -t mi-imagen:v6 .**

2.- Creamos un contenedor: **# docker run -d --name c1 -p 8080:80 mi-imagen:v6**

3.- Comprobamos: localhost:8080

4.- Podemos ver que se ha creado el volumen.#docker volume ls

3.- Creación automática de imágenes. Dockerfile

Ejemplo: Servidor apache → COMPARTIR VOLÚMENES

Ahora vamos a crear otro contendor que tire del mismo volumen.

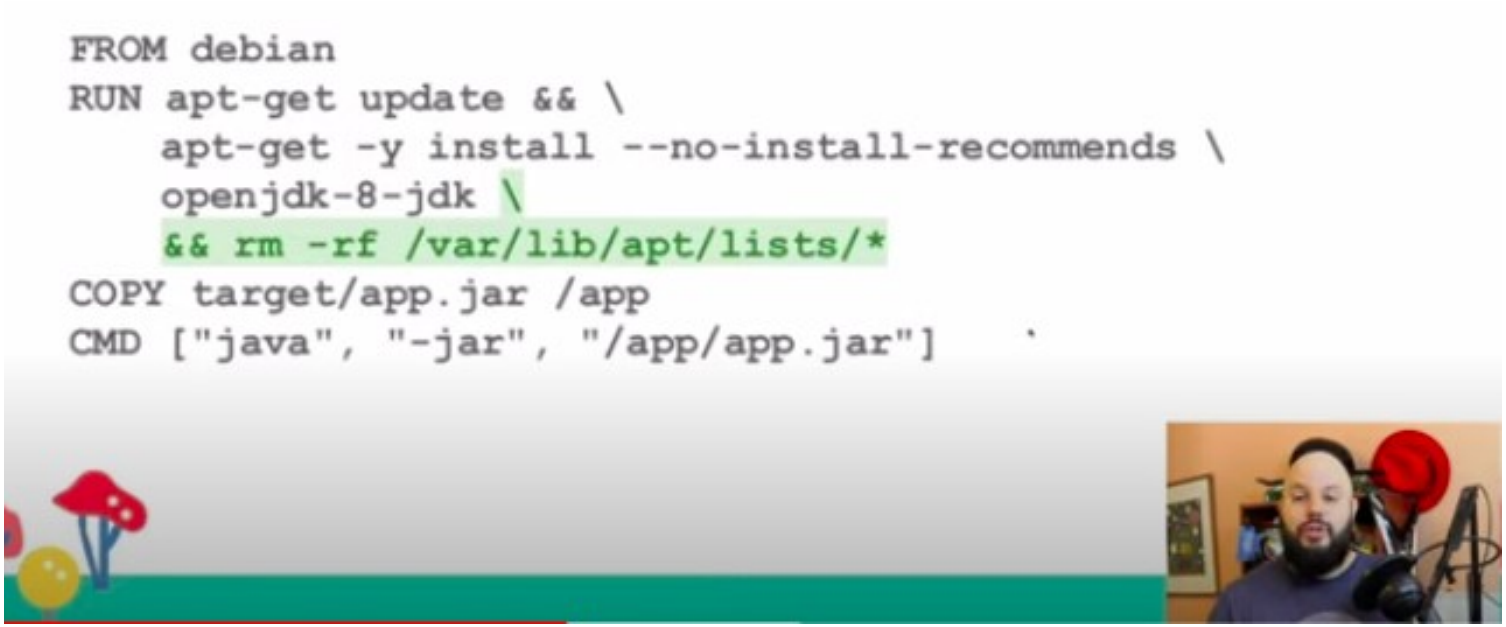
 # docker run -d --name c2 -p 9080:80 --volumes-from c1 mi-imagen:v6
(Otro puerto en el host)

--volumes-from → Carga los mismo volúmenes que c1

Comprobación:
localhost:9080
No ha creado ningún volumen. Está tirando del anterior.
Buscamos el index.html en /var/lib/docker(volume/ID/_data/index.html y cambiamos: “Welcome” por Manuel. vim → :/welcome **¿Qué le ocurre a las páginas?**

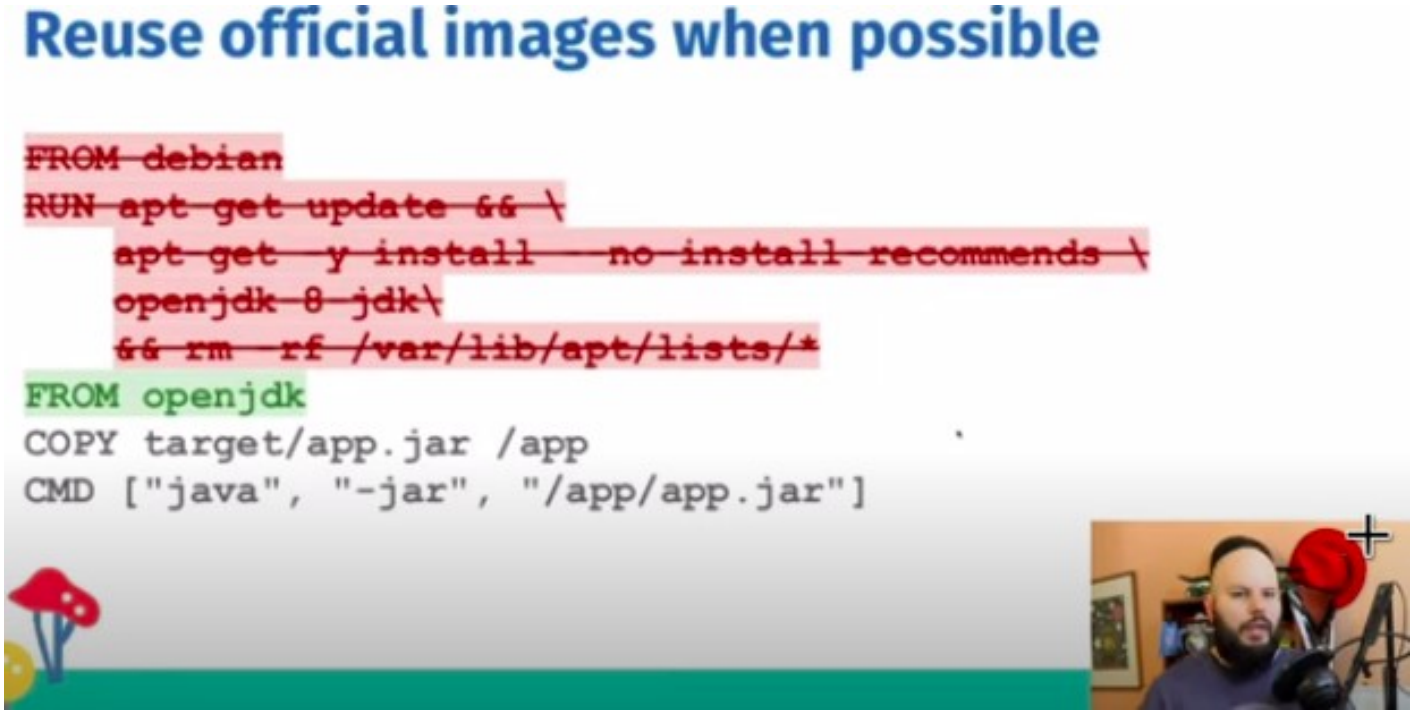
4.- Consejos para escribir Dockerfile

Eliminamos las listas de paquetes



4.- Consejos para escribir Dockerfile

Utilizar imágenes ya preparadas.



4.- Consejos para escribir Dockerfile

Se debe indicar la versión para la que funciona nuestra aplicación.

Resumen

1.- ¿Qué es dockerfile?

Sugerencias/mejoras del tema

Sugerencias /mejoras del tema

Dudas

Referencias

- Programador novato
- Albert Coronado
- El atareao
- Programacionymas
- Pelado Nerd