DOCKER FILE

Un Dockerfile es un archivo legible por el demonio Docker, que contiene una serie de instrucciones para automatizar el proceso de creación de un contenedor.

El comando docker build irá siguiendo las instrucciones del Dockerfile y armando la Imagen. El demonio de Docker es el que se encarga de construir la imagen siguiendo las instrucciones línea por línea y va lanzando los resultados por pantalla. Cada vez que ejecuta una nueva instrucción se hace en una nueva imagen, son imágenes intermedias, hasta que muestra el ID de la imagen resultante de ejecutar todas las instrucciones. El daemon irá haciendo una limpieza automática de las imágenes intermedias.

COMANDOS

FROM

Indicamos una imagen base para construir el contenedor y opcionalmente un tag (si no la indicamos docker asumirá “latest” por defecto, es decir buscará la última versión).

Lo que hace docker al leer esto es buscar en la máquina local una imagen que se llama, si no la encuentra la descargará de los repositorios.

Sintaxis:

FROM <imagen>

FROM <imagen>:<tag>

RUN

Ejecuta directamente comandos dentro del contenedor y luego aplica/persiste los cambios creando una nueva capa encima de la anterior con los cambios producidos de la ejecución del comando y se hace un commit de los resultados. Posteriormente sigue con la siguiente instrucción.

Sintaxis:

RUN <comando> → modo shell, /bin/sh -c

RUN [“ejecutable”,”parámetro1”,”parámetro2”] → modo ejecución, que permite correr comandos en imágenes base que no tengan /bin/sh o hacer uso de otra shell.

ENV

Establece variables de entorno del contenedor. Dichas variables se pasarán a todas las instrucciones RUN que se ejecuten posteriores a la declaración de las variables. Podemos especificar varias variables de entorno en una sóla instrucción ENV.

Sintaxis:

ENV <key> <value> <key> <value>

ENV <key>=<value> <key>=<value>

Si queremos sustituir una variable aunque esté definida en el Dockerfile, al ejecutar un contenedor podemos especificarla y tomará dicho valor, tenga el que tenga en el Dockerfile.

docker run -env <key>=<valor>

También podemos pasar variables de entorno con el comando “docker run” utilizando el parámetro -e, para especificar que dichas variables sólo se utilizarán en tiempo de ejecución.

Podemos usar estas variables en otras instrucciones llamándolas con $nombre_var o ${nombre_var}. Por ejemplo:

ENV DESTINO_DIR /opt/app

WORKDIR $DESTINO_DIR

ADD

Esta instrucción copia los archivos o directorios de una ubicación especificada en <fuente> y los agrega al sistema de archivos del contenedor en la ruta especificada en <destino>. En fuente podemos poner una URL, docker se encargará de descargar el archivo y copiarlo al destino.

Si el archivo origen está comprimido, lo descomprime en el destino cómo si usáramos “tar -x”.

Sintaxis:

ADD <fuente>..<destino>

ADD [“fuente”,...”destino”]

El parámetro <fuente> acepta caracteres comodín tipo ?,*, etc.

Una de las cosas a tener en cuenta (entre otras) es que el <origen> debe estar donde esté el Dockerfile, no se pueden añadir archivos desde fuera del directorio de construcción. Si el destino no existe, Docker creará la ruta completa incluyendo cualquier subdirectorio.

Los nuevos archivos y directorios se crearán con los permisos 0755 y un UID y GID de 0. También tenemos que tener en cuenta que si los archivos o directorios agregados por una

instrucción ADD cambian, entonces se invalida la caché para las siguientes instrucciones del Dockerfile.

COPY

Es igual que ADD, sólo que NO admite URLs remotas y archivos comprimidos como lo hace ADD.

WORKDIR

Permite especificar en qué directorio se va a ejecutar una instrucción RUN, CMD o ENTRYPOINT. Puede ser usada varias veces dentro de un Dockerfile. Si se da una ruta relativa, esta será la ruta relativa de la instrucción WORKDIR anterior. Podemos usar variables de entorno previamente configuradas, por ejemplo:

ENV rutadir /ruta

WORKDIR $rutadir

VOLUME

Crea un punto de montaje con un nombre especificado que permite compartir dicho punto de montaje con otros contenedores o con la máquina anfitriona. Es un directorio dentro de uno o más contenedores que no utiliza el sistema de archivos del contenedor, aunque se integra en el mismo

para proporcionar varias funcionalidades útiles para que los datos sean persistentes y se puedan compartir con facilidad.

Esto se hace para que cuando usemos el contenedor podamos tener acceso externo a un determinado directorio del contenedor.

Las características de estos volúmenes son:

• Los volúmenes pueden ser compartidos y reutilizados entre los contenedores.

• Un contenedor no tiene que estar en ejecución para compartir sus volúmenes.

• Los cambios en un volumen se hacen directamente.

• Los cambios en un volumen no se incluirán al actualizar una imagen.

• Los volúmenes persisten incluso cuando dejan de usarlos los contenedores.

Esto permite añadir datos, BBDD o cualquier otro contenido sin comprometer la imagen.

El valor puede ser pasado en formato JSON o como un argumento, y se pueden especificar varios volúmenes.

VOLUME [“/var/tmp”]

VOLUME /var/tmp

EXPOSE

Se utiliza para asociar puertos, permitiéndonos exponer un contenedor al exterior (internet, host,etc.). Esta instrucción le especifica a Docker que el contenedor escucha en los puertos especificados. Pero EXPOSE no hace que los puertos puedan ser accedidos desde el host, para esto debemos mapear los puertos usando la opción -p en docker run.

Por ejemplo:

EXPOSE 80 443

docker run centos:centos7 -p 8080:80

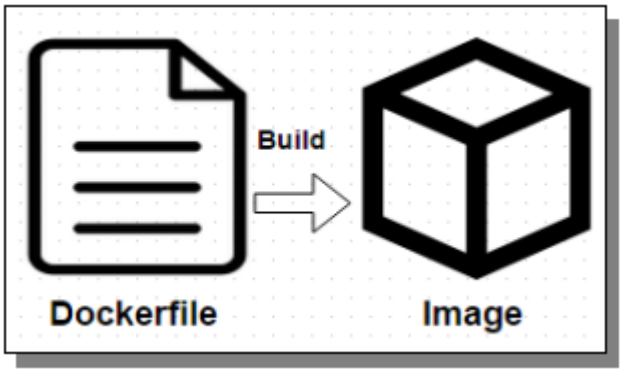
CMD

Esta instrucción es similar al comando RUN pero con la diferencia de que se ejecuta cuando instanciamos o arrancamos el contenedor, no en la construcción de la imagen.

Sólo puede existir una única instrucción CMD por cada Dockerfile y puede ser útil para ejecutar servicios que ya estén instalados o para correr archivos ejecutables especificando su ubicación.

EJEMPLO

1) Creamos el archivo Dockerfile




```
FROM ubuntu:latest
RUN apt update
RUN apt install -y apache2
EXPOSE 80
COPY bootstraphtml /var/www/html
CMD ["/usr/sbin/apache2ctl", "-
DFOREGROUND"]
```

2) Construimos la nueva imagen

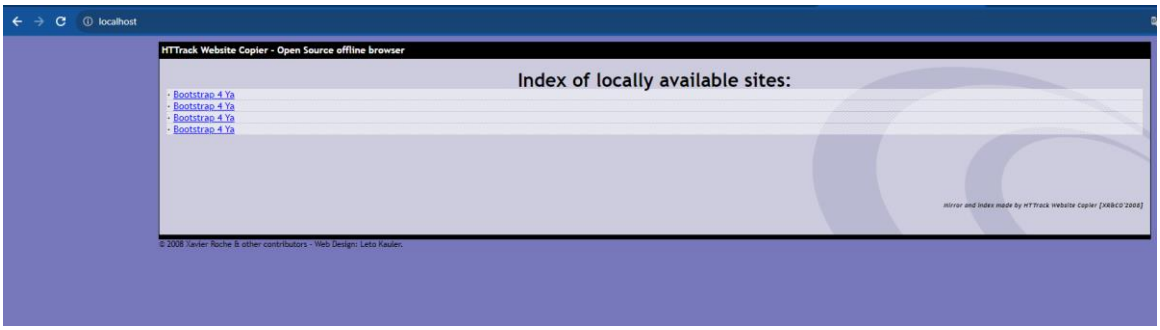
   # docker build -f Dockerfile -t apachefiledeb .

3) Construimos el contenedor basado en la imagen creada.

   # docker run -d --name webapachefiledeb -p 80:80 apachefiledeb

4) Accedemos el servidor mediante el puerto 80

En este caso lo que hicimos es copiar el contenido de la carpeta bootstraphtml (ayuda de Bootstrap en html) a la carpeta /var/www/html obteniendo el siguiente resultado.



VOLUMENES

Podemos montar volúmenes de datos en nuestros contenedores. Con el flag -v podemos montar un directorio dentro de nuestro contenedor. Además, seguirá apareciendo después de un reinicio del contenedor, serán persistentes.

También podemos compartir volúmenes de datos entre contenedores. Si dichos volúmenes están anclados al sistema operativo anfitrión, no serán eliminados por Docker cuando desaparezca el contenedor.

Vamos a crear un contenedor pasándole un volumen

docker run -d --name= webservervolume -p 8080:80 -v //d/bootstraphtml:/usr/share/nginx/html **nginx**

docker run -d --name webservervolume -p 80:80 -v d//bootstraphtml:/usr/local/apache2/htdocs **httpd**

Con el siguiente comando vemos los volúmenes creados en nuestra instalación, algunos tienen nombres explícitos ya que fueron creados de esa manera por nosotros y otros tienen nombres largos que son los creados implícitamente por las imágenes que hemos instalado.

```
C:\Users\alfre>docker volume ls
DRIVER    VOLUME NAME
local     0ebb5ac72b8d78cd35f6ad51100ea4d20d2f3659c54ffec0cf13c03713ad2411
local     0f5d109ebb3cb4ae7574e947b5d1ffca6cb139a8692970922f5ea7d243d45b96
local     6db4373ad53a9c3d46038f86686cd7f47f9e033753c00e85581da8be2ed97133
local     10f13fceea457c221df87e70467711c3b0dc5f03951c6f51d0c47b15a0c09461
local     85aedb10bfccc7a25d475747270ca87f8232bcc6f6b756fa6856a4bd3c96b343
local     89f995a4004500ee6dd80d12323b85d4598e8c93a28c33b8e809b3ada8367a6e
local     503d1acf07bb30b79320c295eaf48ee14774e0c8fba478de6048efcf7337c25e
local     822aa55c28b651014f89fb4d358273decc1a3207931369a9cf3ae1ab1a9738b4
local     8326668b250b71fef56c7a1578af91f301672935eda60db0142ec0663893a0d1
local     b534e9bd426ce6612656b9ecafd567eefa0207ca3c795d58f18a8e4cdcfcd9bb
local     bootstraphtml
local     ca6d2669846f18b92bc5c4993b845749fefb02fc5a0ab874e295a7a8751dfbaf
local     cedcc167a3f34219258f9e0f00558dda4ed90ae216e1e31ae4f0375788164757
local     d89e8f56e68fcee73fd93db5236991cd0d6c69496295c1763cd7c887620aa76a
local     f54604c345618017b8d061ee309b122d65cac87c2a38a952e0ecd66ebcfff349
local     fd7af5714060a96697bc0bbd476e941073b4476ff70b7e02730231ec4ff17159

C:\Users\alfre>
```