02/20223

# INTRODUCCIÓN A DOCKER

DANILO RIVERO PÉREZ

Curso en YouTube realizado por Pelado Nerd

# ÍNDICE

## ***INTRODUCCIÓN***

Este documento se ha escrito a medida que iba avanzando en el curso de Pelado Nerd expuesto en YouTube. He decidido realizar este curso de Introducción a Docker debido a que me atrae bastante esta tecnología, es muy demandada a día de hoy y además quiero aprender *Kubernetes* (tecnología que se basa en Docker), y por todo ello he realizado este estupendo curso.

El objetivo principal de este documento es plasmar por escrito los conocimientos aprendidos durante el curso y apuntarlos para poder consultarlos en un futuro si hiciera falta.

## ***DIFERENCIAS ENTRE MVS, CONTENEDORES, DOCKER Y KUBERNETTES***

- MVS: Simula un S.O completo, soporta hardware virtualidad, software, la imagen del S.O, etc.

- Contenedores: Son como las MVS pero son mas ligeros, ya que no utilizan imágenes creo, solo virtualidad el software y no el hardware.

- Docker: Utiliza contenedores y es portable, ya que si yo por ejemplo tengo una app que requiere para su funcionamiento una determinada versión de JAVA, lo que hago esquema mi app en el contenedor de docker va a funcionar en mi PC que si tiene esa version de Java instalado, en el PC DE mi vecino que no la tiene instalado, en cualquiera, ya que ese contenedor ya tiene esas dependencias instaladas sin necesidad que tu PC las tenga, por lo cual las apps funcionan siempre y en cualquier lado. Además puedes probar diferentes software sin instalarlo en tu PC, solo usando las imágenes de Docker Hub.

- Kubernetes: Cluster de contenedores que facilita mucho la vida cuando se tiene muchos contenedores docker y para automatizar el despliegue de las apps en contenedores. Utiliza nodos, namespaces, pods, etcd (bd de kubernettes), una API, servicios, agentes, etc. Permite crear un cluster de dos

que implementa determinadas funcionalidades sobre los contenedores como: alta disponibilidad, tolerancia a fallos, permite escalar cuando se queda corto de recursos, trabaja de forma eficiente, se pueden realizar modificaciones en caliente, etc.

## ***PARA EMPEZAR CON DOCKER***

- Descargar e instalar Docker Desktop: https://www.docker.com/products/docker-desktop/

- Docker Hub (donde están las imágenes): https://hub.docker.com/

## ***CORRER IMÁGENES DOCKER***

- docker run nginx

- docker run -e POSTGRES_PASSWORD=password postgres (Si sale un error)

- docker run Postgres:9.1 (Corre una version especifica de Postgres, indicando el TAG después de ":")

- docker run -d ubuntu (Corre la imagen en background, en segundo plano)

- docker run -d --name Ubuntu ubuntu (Corre la imagen en background, y le da un nombre, en este caso Ubuntu)

## ***DESCARGAR IMÁGENES DOCKER***

- docker pull postgres

- docker pull ubuntu

## ***VER IMAGENES DESCARGADAS***

- docker images

## ***VER QUE IMAGENES ESTÁN CORRIENDO***

- docker ps (solo se ven las imágenes o contenedores que están corriendo actualmente)

- docker ps -a (Se ven las imágenes que estaban corriendo alomejor hace 1 semana también)

## ***VOLVER A CORRER/INICIAR UNA IMAGEN/CONTENEDOR QUE ESTÁ PARADO***

- docker start CONTAINER_ID/NAME (El containerID o el name se ve haciendo docker ps o docker ps -a)

## ***PARAR UNA IMAGEN/CONTENEDOR DOCKER***

- docker stop CONTAINER_ID/name

## ***VER LOGS DE UNA IMAGEN/CONTENEDOR DE DOCKER***

- docker logs CONTAINER_ID/name

- docker logs -f CONTAINER_ID/name (Se ven los logs pero se queda ahí esperando, como una consola interactiva, como un tail -f /var/log/messages de Linux)

## ***EJECUTAR COMANDOS INTERACTIVOS EN UNA TERMINAL DE UN CONTENEDOR/IMAGEN QUE ESTÉ CORRIENDO DE DOCKER***

- docker exec -it CONTAINER_ID/name sh (En este caso, ejecuto en el contenedor CONTAINER_ID el comando sh, que es una shell, en una consola interactiva)

## ***FICHERO DOCKERFILE***

- Es un fichero con todas las especificaciones y dependencias que necesita nuestra app para poder ser ejecutada. Es muy buena práctica que este fichero se llame "Dockerfile"

*-EJEMPLO de fichero Dockerfile para correr una app echa en Node JS:*

```
FROM node:12.22.1-alpine3.11 # (Descargamos la imagen de Node que ya tiene todas las dependencias: npm, etc.)

WORKDIR /app # (Donde se va a hacer el trabajo, en que directorio. El directorio de trabajo del contenedor. Si el directorio de trabajo no existe se crea y los comandos RUN, COPY, CMD ADD, ENTRYPOINT ... se ejecutan dentro de ese directorio, /app, en este caso)

COPY . . # (Copia todos los archivos que hay en el directorio en el que estoy en mi aplicación dentro del contenedor. Que el directorio de trabajo del contenedor en este caso es /app. Por lo que copia todos los archivos que están en el directorio donde está el DockerFile en /app)
RUN yarn install --production # (Se ejecuta el comando yarn install ...)
```

CMD ["node", "/app/src/index.js"] # (Especificar el comando que vaya a correr cuando se inicie el contenedor, en este caso se ejecuta el comando node /app/src/index.js cuando se inicie el contenedor)

## ***CONSTRUIR UN CONTENEDOR/IMAGEN***

- docker build . (Construe un contenedor, que podemos ver una vez inicializado con docker ps o en la app docker dashboard. Se construye sin nombre, solo con el ID)

- docker build -t NOMBRE_CONTENEDOR . (Así se le especifica para taguearlo y que así tenga nombre el contenedor. Usa el dockerfile que esté en la ruta especificada. En este caso ".", por eso hay que llamarlo Dockerfile el fichero de definición del contenedor)

- Para correr la imagen creada en el docker file y construidas, usamos docker run NOMBRE_IMAGEN/CONTENEDOR

## ***SOLUCION DE PROBLEMAS DE PUERTOS DE DOCKER AL CORRER UNA APP POR LOS PUERTOS***

- Puede ser que mi PC no tenga el puerto que utiliza mi app para correr abierto o algo de eso, por eso hay que especificárselo.

- docker run -dp 3000:3000 NOMBRE_CONTENEDOR/IMAGEN (-d para background y -p para especificar el puerto. De esta manera se expone el puerto 3000 a la red de mi PC)

## ***SOLUCION SI QUIERO QUE LOS DATOS DE MI APP SE GUARDEN PERMANENTEMENTE EN LA BD SI SE DESTRUYE EL CONTENEDOR Y SE VUELVE A INICIAR. VOLUMENES***

- docker run -d -v RUTA_LOCAL_DONDE_QUEREMOS_GUARDAR:RUTA_REMOTA_DE_LA_DB -p 3000:3000 NOMBRE_CONTENEDOR/IMAGEN

- docker build -t NOMBRE:CONETENEDOR:v2 . (Si al final del día he hecho cambios en mi app, en us archivos y demás y demás y quiero que sean persistentes en el contenedor, construyo una version 2 de contenedor con esos cambios "v2". Usando el dockerfile, ".", en este caso)

## ***SUBIR MI IMAGEN CREADA A DOCKER HUB PARA PODER SER DESCARGADA EN OTRO PC O POR OTRA PERSONA***

1) docker tag NOMBRE_QUE_QUIERO_DE_LA_IMAGEN_A_SUBIR:v2 IMAGE_ID (Image_id de la imagen que quiero subir. Sepuede mirar haciendo docker images. Esto ni hace falta, con solo el push y hacer un docker login antes bastaría)

2) docker push NOMBRE_QUE_LE_PUSE_DE_LA_IMAGEN_A_SUBIR (VERIFICARLO EN DOCKER HUB EN MY PROFILES - REPOSITORIES)

## ***MULTICONTENEDORES***

- En muchos casos, es buena práctica correr dos servicios en 2 contenedores diferentes, por ejemplo por un lado el código de la app y en otro contenedor la BD que usa esa app.

- Crear una red de Docker para que 2 contenedores corran en la misma red y se puedan ver entre ellos: docker network create NOMBRE_RED

- Corremos un contenedor de MySQL que use esa red: docker run -d --network NOMBRE_RED --network-alias NOMBRE_ALIAS -v ./mysql/data(DIRECTORIO_LOCAL):/var/lib/mysql(directorio del contenedor) -e MYSQL_ROOT_PASSWORD=secret-e MYSQL_DATABASE=todos mysql:5.7 (La primera -e es una variable de entorno para setear la password del usuario root de mysql y ponerle de contraseña secret y la segunda -e crea una BD llamada todos)

- Ejecutar comandos en el contenedor de MySQL: docker exec -it CONTAINER_ID mysql -p

- Correr una imagen de docker HUB usando nuestra red creada anteriormente: docker run -it --network MI_RED Nicolasa/netshoot (Si aquí hiciéramos dig ALIAS_DE_LA_RED, nos resuelve y nos da la IP)

- Creación del contenedor del código de la app que utiliza la BD del otro contenedor: docker run -dp 3000:3000 --network MI_RED - e MYSQL_HOST=NOMBRE_CONTENEDOR_MYSQL -e MYSQL_USER=root -e MYSQL_PASSWORD=secret -e MYSQL_DB=todos NOMBRE_IMAGEN_DEL_CONTEENDR_DEL_CODIGO_CREADO_CON_DOCKERFILE

## ***DOCKER COMPOSE***

- Herramienta para definir y ejecutar aplicaciones de Docker de varios contenedores. Para ello se utiliza un archivo YAML. Con este archivo me pedo ahorrar todas las líneas de docker run para correr las imágenes además, con docker compose defino todos los contenedores/servicios que van a correr a la vez y además no es necesario especificar una red ni crearla, ya que docker compose la crea y crea todos los contenedores/servicios en la misma red.

- *EJEMPLO FICHERO docker-compose.yaml* (Son dos contenedores, una para la pp, el código y otro contenedor que usa la BD de la app):

```
Version: "3.7" (Version de la sintaxis del archivo YAML)

Services:

        app: (Nombre servicio 1 o contenedor 1. Además así se llamará el network-alias)
            image:  pablokbs/getting-started:v1  (IMAGEN  DE DOCKER)
            ports:
                - 3000:3000
              environment:
                MYSQL_HOST: mysql
                MYSQL_USER: root
                MYSQL_PASSWORD: secret
                MYSQL_DB: todos (Nombre BD)

        mysql: (Nombre servicio 2 o contenedor 2. Además así se llamará el network-alias)
            image: mysql:5.7
              volumes:
                - ./mysql-data:/var/lib/mysql
              environment:
                MYSQL_ROOT_PASSWORD: secret
                MYSQL_DATABASE: todos
```

- LEVANTAR y CORRER LOS CONTENEDORES CREADOS EN docker-compose.yaml: docker-compose up -d (Lo puedo usar todas

las veces que desee, y si hay alguno caído levanta solo el contenedor caído)

- FRENAR TODO LOS CONTENEDORES Y BORRAR LA RED Y DEMAS: docker-compose down

## ***CONCLUSIONES***

Tras finalizar el curso y este documento, he aprendido bastante acerca de Docker, de contenedores, de virtualización, a crear mis propias imágenes Docker, a subirlas a DockerHub, a ejecutar contenedores, a usar Docker Compose, etc. Por todo ello, estos conocimientos me han servido de base para seguir aprendiendo y avanzando en la tecnología de Docker y poder adentrarme en el mundo de Kubernetes.