**Conceptos**

**Imagen**: Es la que contiene las dependencias, bibliotecas, etc. Básicamente es una plantilla o (modelo) que solo tiene lo necesario para ejecutar la aplicación.

**Contenedor**: Son instancias o (objetos como en POO), el cual es básicamente una instancia de una imagen.

**Dockerfile:** Es la configuración para la creación de la imagen, acá se define que va a llevar, las instrucciones, dependencias, y configuración del entorno, es un archivo que lleva texto y este después se ejecuta.

**Red**: Esta red es virtual, permite comunicación entre contenedores esto es usado en aplicaciones grandes que van a tener varios servicios, entonces en un contenedor se crea el micro servició estos llevan los scripts (básicamente el backend o la lógica del negocio).

**Volumen:** Es donde están los datos generados y usados por los contenedores.

**Nota:**

* Cree una cuenta de docker hub.
* Los comandos sin comillas.
* Algunos comandos como por ejemplo build, compose requieren que estes en el directorio donde creaste los archivos, para que funcionen.

**Curso**

**contenedores**

**Crear contenedor**: “docker run nombre\_contenedor”.

**Alias (nuevo nombre)**: “docker run –name nombre\_alias nombre\_contenedor” lo usas para cambiar el nombre en NAMES y no usar el CONTAINER ID.

**Listar contenedores**: Contenedores que están corriendo con “docker ps” y los que están apagados agrega “-a” al final “docker ps -a”.

**Borrar contenedor:** “docker rm nombre\_contenedor” y con “docker container prune” borra todos los contenedores.

**Correr contenedor:** “docker run –d --name Linux Ubuntu” en este caso se está corriendo un contenedor que tiene Ubuntu y no tener que ponerlo a correr en modo interactivo “docker run –it Ubuntu”, el cual te cambiara a la terminal de Ubuntu.

Para entrar en modo interactivo.

Si quieres entrar en modo interactivo después de ejecutar el comando “docker run –d --name run Linux Ubuntu”, escribes “docker exec –it nombre\_contenedor bash”.

**Detener contenedor:** Para detener el contenedor en Linux es “kill 1234”, este número es el id del proceso que está corriendo la máquina virtual de docker para obtenerlo escribe “docker inspect nombre\_contenedor –format ‘{{.State.Pid}}’”.

En Windows para detenerlo “docker kill nombre\_contenedor”.

O con “docker stop”.

**Exponer contenedor:** “docker run –d –name proxy –p 8001:80 nginx”, vamos a exponer un puerto del contenedor usando nginx, de los números a la izquierda puerto del computador, derecha puerto del contenedor y puedes acceder con “<http://localhost:8001>” en un navegador web.

**Datos**

**Persistir datos (Bind mount):**

“docker run –d –name db –v C:\ruta\directorio\carpeta\_persistir\_datos\mongo mongo”, esto persiste datos que se guarde en la base de datos mongodb localmente si haces un “docker rm –f db”, y creas un nuevo contenedor y buscas el documento de mongo los datos persistirán.

**Crear volumen:** “docker volumen créate nombre\_volumen”, creación básica de volumen. Un ejemplo más práctico “docker run -d --name db --mount src=nombre\_volumen,dst=/data/db mongo”, en src es el nombre del volumen y en dst el directorio dentro del contenedor donde se montará el volumen, esto te sirve para guardar datos.

**Insertar archivos:** Crea archivo externamente con “touch nombre.extension”, y es importante que el contenedor este corriendo, después copias el archivo de tu computador y lo pegas en el contenedor de docker con “docker cp ruta\_del\_archivo\_en\_host nombre\_container:/ruta/dentro/del/contenedor”.

**Extraer archivos:** Ahora para extraer archivos haces lo mismo de forma opuesta con

“docker cp nombre\_container:/ruta/dentro/del/contenedor ruta\_para\_guardar\_archivo\_en\_host”.

**Imágenes**

**Listar imágenes:** “docker image ls” ver imágenes descargadas, de docker hub.

**Descargar imagen:** “docker pull nombre\_imagen” descarga imagen de docker hub.

**Crear imagen:** Para crear imagen se crea un dockerfile

“

#descargar imagen de docker hub

FROM ubuntu:latest

#ejecutar un comando (este se ejecuta en tiempo de construcción)

RUN touch /usr/src/holamrxd.txt

”.

**Construir imagen:** Con “docker build –t ubuntu:tutag .” construyes la imagen y va a tener acceso a lo que tenga la carpeta ”.” donde creaste el dockerfile.

**Borrar imagen:** Con “docker image rm nombre\_imagen:tag” borras la imagen es importante que pongas el nombre de repositorio y el tag esto aparece cuando lista la imagen.

**Cambiar nombre del tag:** Con “docker tag ubuntu:mrxd name:tag” cambia el nombre a uno personal y ya se puede publicar.

**Publicar imagen:** La imagen se publicará en docker hub que es el repositorio público de imágenes,

se hace con “docker login” y después con “docker push usernamedelhub/name:tag” nombre de repositorio y el tag esto aparece cuando lista la imagen.

**Historial de una imagen:** Con “docker history name: tag”

**Ejemplo práctico para desarrollo**

Clona repositorio “git clone https://github.com/platzi/docker”

Construye la imagen “docker build -t mrnode .”

Evitar reinstalación y hacer cambios de código

“

FROM node:14

COPY ["package.json","package-lock.json", "/usr/src/"]

WORKDIR /usr/src

RUN npm install

COPY [".","/usr/src/"]

EXPOSE 3000

CMD ["node", "index.js"]

”

Hacer reload

“CMD ["npx","nodemon", "index.js"]”

Y ejecuta

“docker run -–rm –p 3000:3000 –v C:/Users/…/docker/index.js:/usr/src/index.js mrnode” el “--rm” es para eliminar contenedor apenas se detenga.

**Redes**

**Listar Redes:** Con “docker network ls”.

**Crear una red:** Con “docker network créate –-attachable nombre\_red”, –-attachable esto es para que cualquier contenedor se conecte a esta red.

**Conectarse a red:** ahora crea dos contenedores ejemplo uno con mongo y el otro con node, conecta el de mongo “docker network connect nombre\_red nombre\_contenedor”, y con node puedes hacer algo así “docker run -d --rm --name appweb --network cable -p 3000:3000 --env MONGO\_URL=mongodb://db:27017/test -v C:/…/index.js:/usr/src/index.js mrnode”, env es una variable de entorno y así conectas dos contenedores para crear una red.

**Docker compose**

Esta herramienta nos ayuda a hacer todo de forma más sencilla de forma declarativa, es importante la indentacion como en Python. ejemplo:

Crea un archivó con el nombre de “docker-compose.yml”

“

version: "3.8"

services:

  app:

    image: mrnode

    environment:

      MONGO\_URL: "mongodb://db:27017/test"

    depends\_on:

      - db

    ports:

      - "3000:3000"

  db:

    image: mongo

”

Ahora para ejecutar esto lo haces con “docker-compose up -d” el “–d” evita entrar al modo interactivo, para entrar al modo interactivo es con “docker-compose exec app bash”, “app” es el nombre del servicio.

Para destruir todo con “docker-compose down”.

**Construir imagen:** “docker-compose build” y el archivo quedaría así

“

version: "3.8"

services:

  app:

    build: .

    environment:

      MONGO\_URL: "mongodb://db:27017/test"

    depends\_on:

      - db

    ports:

      - "3000:3000"

  db:

    image: mongo

”

Cuando necesites un entorno personal, usted puede crear el archivo “docker-compose.override.yml”

El archivo “docker-compose.yml” lo deja como viene y en “docker-compose.override.yml” modifica lo que necesita ejemplo:

En “docker-compose.yml”

“

version: "3.8"

services:

  app:

    image: mrnode

    environment:

      MONGO\_URL: "mongodb://db:27017/test"

    depends\_on:

      - db

    ports:

      - "3000:3000"

  db:

    image: mongo

”

En “docker-compose.override.yml”

“

version: "3.8"

services:

  app:

    build: .

    environment:

      MONGO\_URL: "mongodb://db:27017/test"

    depends\_on:

      - db

    volumes:

      - .:/usr/src

      - /usr/src/node\_modules

    command: npx nodemon index.js

  db:

    image: mongo

”