**Docker**

Imagina que debes entregar un trabajo en equipo en la escuela, tu y tu grupo hacen el trabajo por separado y lo unen en la escuela. Seguramente no cuadre un pingo. En las empresas de software ocurre lo mismo, para que cuadre el trabajo, los desarrolladores deben trabajar conjuntamente en el mismo entorno de trabajo. Y la solución mas sencilla a esto son los contenedores.

**Contenedores**

Contienen todo lo que necesita una aplicación para funcionar, en palabras sencillas, son un entorno de ejecución para un software que contiene todas las dependencias que necesita la aplicación y este esta separado del sistema, puedes tener diferentes contenedores con diferentes versiones de dependencias y no van a hacer conflicto entre ellos, ya que todos los entornos están separados de los demás.

Así que si quiero compartir mi aplicación en otras maquinas pero necesito asegurarme que este funcione, se envía dentro de una réplica del contenedor con todas sus dependencias donde lo desarrolle, esto asegura que funcione en cualquier máquina.

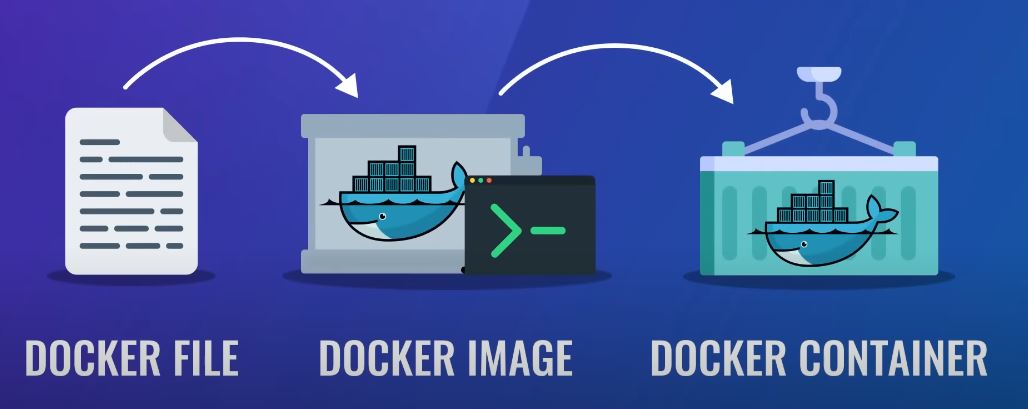
Los contenedores son extremadamente livianos así que los hacen una vía recomendable para compartir tu proyecto con demás personas.

***Dockerfile***

Es un archivo de texto que contiene todos los comandos e instrucciones para instalar todas las dependencias, es como un manual. Con ese manual se crea la imagen Docker.

***Docker Image***

Es un ejecutable que contiene todas las dependencias y el software, a partir de esa imagen se crea los contenedores.

*****Docker Container***

**Aplicación monolítica**

Una aplicación monolítica es un tipo de software donde todas sus partes, como el frontend, backend y base de datos, están integradas en una sola unidad. En lugar de dividir la aplicación en servicios independientes, todo funciona como un solo bloque. Por lo tanto, cualquier cambio o actualización en una parte de la aplicación generalmente requiere modificar y desplegar todo el sistema.

**Microservicios**

Los microservicios son una arquitectura de software en la que una aplicación se compone de muchos servicios pequeños e independientes, cada uno ejecutando un proceso único y específico. En contraste con las aplicaciones monolíticas, donde todo está integrado en una sola unidad, en una arquitectura de microservicios, cada servicio opera de manera autónoma y se comunica con otros servicios a través de API (Interfaz de Programación de Aplicaciones).

**DOCKERFILE**

*# Especifica la imagen base que se usará para construir esta imagen.*

**FROM** ubuntu:latest

*# Instalar paquetes necesarios*

**RUN** apt-get update && \

apt-get install -y \

package1 \

package2 \

&& rm -rf /var/lib/apt/lists/\*

*# Configuración de entorno*

**ENV** ENV\_VAR\_NAME="valor"

*# Directorio de trabajo*

**WORKDIR** /app

*# Copiar archivos*

**COPY** script.sh /app

*# Establecer permisos y ejecutar script*

**RUN** chmod +x /app/script.sh

**CMD** ["/app/script.sh"]

**COMANDOS DE DOCKERFILE**

**RUN** Ejecuta comandos durante la construcción de la imagen. Aquí, se actualiza el sistema, establece permisos de ejecución en el script, se instalan paquetes y se limpian los archivos de cache de apt-get.

**ENV** Define variables de entorno que estarán disponibles durante la ejecución de los contenedores basados en esta imagen.

**COPY** Copia archivos desde el contexto de construcción del Dockerfile al sistema de archivos de la imagen.

**ADD** Igual a COPY pero este intentará descomprimir automáticamente los archivos si son archivos comprimidos conocidos.

**CMD** Proporciona un comando predeterminado que se ejecutará cuando inicies un contenedor basado en esa imagen. Se puede definir como un comando ejecutable o como un array JSON de strings.

**ENTRYPOINT**

A diferencia de CMD, el comando o script especificado en ENTRYPOINT no se sobrescribe fácilmente cuando se inicia el contenedor. En su lugar, cualquier argumento proporcionado al iniciar el contenedor se pasa como argumento al comando o script especificado en ENTRYPOINT.

**WORKDIR** Establece el directorio de trabajo dentro del contenedor.

***DOCKER VOLUMES***

En resumen, los volúmenes en Docker son como carpetas especiales que proporcionan una forma fácil y confiable de almacenar y compartir datos entre contenedores y con el host. Esto hace que Docker sea más versátil y adecuado para una amplia gama de aplicaciones y escenarios de uso.

***Como crearlos***

Crear un volumen en Docker es bastante simple. Puedes crear un volumen utilizando el comando docker volume create seguido del nombre que desees asignar al volumen.

**docker volume create nombre\_del\_volumen.**

***Como usarlos***

**Docker run****-d --name mi\_contenedor -v volumen:/ruta img:tag.**

PACTICA

**Crear tu propio Dockerfile**

Ve a un directorio con cd.

**cd Docker/.**

Crea un archivo Dockerfile con touch.

**touch Dockerfile.**

Entra al archivo Dokerfile con nano

**nano Dockerfile.**

Edítalo como quieras, para salir de la edición deberás poner Cntr + X, luego S, y luego guardar el archivo con un ENTER.

**Crear la imagen**

Construye la imagen con:

**docker build -t nombre:etiqueta .**

Docker build : Construye una imagen a partir de un Dockerfile.

**-t**  : Le asigna un nombre y etiqueta a la imagen.

Name:tag : Tanto el nombre como el tag pueden ser a elección.

**•**  : El punto significa que el Dockerfile esta en el directorio actual.

**COMANDOS DOCKER**

**Docker save**

Este comando guarda una o varias imágenes dentro de un archivo comprimido que se podrá compartir entre máquina, o utilizar como copia de seguridad de esas imágenes.

docker save ID > archivo.tar**.**

**Docker load**

Se utiliza para cargar una imagen Docker desde un archivo tar.

docker load -i archivo.tar

O

docker load < archivo.tar

**Docker top**

Con este comando puedes ver los procesos que esta ejecutando un contenedor en ejecución.

docker top ID**.**

**Docker events**

Muestra eventos en tiempo real relacionados con los contenedores Docker y otros objetos del sistema Docker. Como: Visualización de eventos en tiempo real, Información sobre eventos, Monitoreo del sistema Docker.

docker events

**Docker diff**

Muestra las diferencias que tiene un contenedor actual comparado a cuando se creó.

docker diff ID**.**

**Docker Compose**

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multi contenedor de forma sencilla. Con Compose, puedes utilizar un archivo YAML para configurar los servicios de tu aplicación, sus dependencias y las redes que los conectan. Luego, con un simple comando, puedes levantar toda la infraestructura de tu aplicación con todos los servicios necesarios, en lugar de tener que iniciar y configurar cada contenedor individualmente.

Utiliza un archivo YAML para definir los servicios, volúmenes, redes y otras configuraciones necesarias para tu aplicación. Esta definición es fácil de entender y compartir entre los miembros del equipo.

**version: '3'**

**services:**

**backend:**

**image: miimagenbackend:latest**

**ports:**

**- "3000:3000"**

**depends\_on:**

**- db**

**db:**

**image: mysql:5.7**

**environment:**

**MYSQL\_ROOT\_PASSWORD: mysecretpassword**

**Kubernetes**

**Orquestación de Contenedores:** Kubernetes administra y orquesta la ejecución de contenedores, lo que permite a los desarrolladores definir cómo deben ejecutarse sus aplicaciones y cómo deben interactuar entre sí.

**Escalabilidad Automática:** Kubernetes puede escalar automáticamente la cantidad de contenedores que ejecutan una aplicación en función de la demanda de recursos, lo que garantiza un rendimiento óptimo incluso en situaciones de alta carga.