

Nombre:

Fecha:

Profesor:

Materia:

Institución:

Curso:

Nota:

Docker:

• Introducción:

Docker: Es una plataforma de software que permite crear, probar e implementar aplicaciones de manera rápida y consiste utilizando contenedores.

surgió en 2013 gracias a Solomon Hykes en la empresa dotCloud, y en poco tiempo se convirtió en una de las herramientas más influyentes del mundo de VOPS.

La idea principal detrás de Docker es aislar aplicaciones y todas sus dependencias en un mismo paquete, llamado contenedor, de forma que se pueda ejecutar igual en cualquier máquina sin preocuparse por las diferencias de sistema operativo, librerías o configuraciones.

El problema que se suele resumir en la frase: "Funciona en mi máquina". Muchas veces, un software funciona correctamente en el computador de un desarrollador, pero falla al pasarlo a otro entorno. Docker elimina este riesgo porque el contenedor incluye todo lo necesario para que la aplicación se ejecute de forma idéntica en cualquier lugar.

• Virtualización:

Contenerización

Antes de Docker, lo habitual era usar máquinas virtuales (VMs). Una VM utiliza un hipervisor para crear una copia completa de un sistema operativo. Esto garantiza aislamiento, pero es pesado: consume memoria, CPU y espacio en disco.

Los contenedores, en cambio, no replican todo un sistema operativo. Comparten el kernel del host y solo empaquetan las librerías y dependencias necesarias para la aplicación.

Comparación directa:

• Máquina Virtual (VM):

• Tamaño: 5 - 70 GB.

• Arranque: Lento (Minutos).

• Cada VM requiere un sistema operativo completo.

• Contenedor:

• Tamaño: Pocos MB.

• Arranque: Segundos.

• Comparte el kernel de host, solo lleva lo esencial.

Este enfoque hace que Docker sea mucho más eficiente y ligero que las máquinas virtuales, sin perder la portabilidad ni el aislamiento.

Arquitectura de Docker

Docker tiene varios componentes fundamentales:

- **Docker Engine:** es el motor Principal de Docker. Permite construir imágenes, ejecutarlas en contenedores y gestionarlos.
- **Docker Daemon (dockerd):** proceso que se ejecuta en Segundo Plano: Administra imágenes, contenedores, redes y volúmenes.
- **Docker CLI:** interfaz de línea de comandos que el usuario utiliza Para interactuar con Docker. Ejemplo: `docker run`, `docker ps`.
- **Docker Hub:** repositorio oficial de imágenes, similar a GitHub Pero Para contenedores. Allí puedes encontrar imágenes ya listas (Ubuntu, MySQL, Redis, etc.) o subir las tuyas.

Flujo de trabajo básico:

- El desarrollador escribe un DockerFile.
- con `docker build` se crea una imagen a partir del DockerFile.
- con `docker run` esa imagen se ejecuta como un contenedor.

Imágenes y Contenedores

Imágenes

Una imagen es como una plantilla inmutable que contiene lo necesario Para ejecutar un programa: sistema base, dependencias, configuraciones y códigos de la aplicación.

Ejemplo de imágenes: `ubuntu:20.04`, `node:18`, `mysql:8.0`,

Contenedores

Un contenedor es una instancia en ejecución de una imagen. Puedes tener varios contenedores corriendo a partir de la misma imagen.

DOCKERFILE

El DockerFile es un archivo de texto con instrucciones para construir imágenes.

Ejemplo:

```
FROM node:78
WORKDIR /app
COPY . .
RUN npm install
CMD ["npm", "start"]
```

ESTE DOCKERFILE:

- usa como la imagen oficial de Node.js version 78.

- usa como base la imagen oficial de Node.js

- crea un directorio de trabajo /app;

- copia el código dentro del contenedor.

- instala dependencia con npm install.

- Arranca la aplicación con npm start.

Comandos básicos de Docker

- ver contenedores en ejecución

bash

☞ copiar código

docker ps

- ver todos los contenedores (incluyendo detenidos)

bash

☞ copiar código

docker ps -a

Nombre:

Fecha:

Profesor:


Materia:

Institución:

Curso:

Nota:

- Descargar una imagen desde Docker Hub

bash  copiar código

docker pull ubuntu:20.04


- Ejecutar un contenedor

bash  copiar código

docker run -it ubuntu:20.04 bash

Esto inicia un contenedor interactivo con Ubuntu y abre la terminal bash.


- Detener y eliminar contenedores

bash  copiar código

docker stop <id contenedor>


docker rm <id contenedor>

- Eliminar imágenes

bash  copiar código

docker rmi ubuntu:20.04

- Construir una imagen desde un Dockerfile

bash  copiar código

docker build -t mi_app:1.0

- Correr un contenedor mapeando puertos

bash  copiar código

docker run -d -p 8080:80 nginx

Esto levanta un servidor Nginx en el puerto 80 del contenedor, accesible desde el puerto 8080 de la máquina.

- Ventajas de Docker

- Portabilidad: Se ejecuta en cualquier sistema operativo o nube.
- Escalabilidad: ideal para arquitectura de microservicios.
- Rapidez: arranque casi instantáneo.
- Eficiencia: consume menos recursos que VMs.
- Integración con CI/CD: se acopla fácilmente a Pipelines de integración continua.
- Ecosistema amplio: Compatible con Kubernetes, Swarm, compose, etc.

• Casos de uso

- Desarrollo: todos los desarrolladores trabajan con el mismo entorno.
- Testing: Se replican entornos de prueba idénticos a Producciones.
- Producción: despliegues escalables y aislados.
- Big Data: levantar Clústeres de procesamiento rápidamente.
- Machine Legacy: ejecutar Software antiguo en contenedores modernos.

• Herramientas relacionadas


- Docker Compose: define y administra múltiples contenedores con un solo archivo `docker-compose.yml`.
- Docker Swarm: Orquestación nativa de Docker para múltiples nodos.
- Kubernetes: el Orquestador más popular y nivel empresarial, con mayor complejidad y escalabilidad.
- Podman: alternativa a Docker sin daemon y con foco en seguridad.

• Limitaciones y desafíos:

- No Sustituye Completamente a las máquinas Virtuales.
- Persistencia de datos: Los contenedores son efímeros, Se requiere configurar volúmenes.
- Seguridad: Contenedores mal configurados Pueden abrir vulnerabilidades.
- Curva de aprendizaje: Para quienes vienen de entornos tradicionales, Pueden ser complejos al inicio.

• Ejemplo Práctico con Docker Compose

Archivo docker-compose.yml Para levantar una aplicación Node, JS con base de datos MongoDB:

yaml  copiar código.

version: '3'

services:

app:

build: .

ports:

- "3000:3000"

depends-on:

- mongo


mongo:

image: mongo

ports:

- "27017:27017"

con este archivo basta ejecutar:


- bash  copia código

docker-compose up

y automáticamente se levantan los servicios (la app y mongo DB).


• Ejemplo de red y volúmenes en Docker

Crear una red Personalizada:

bash  copiar código


docker network create mi-red

Levantar un contenedor conectado a esta red:

bash  copiar código

docker run -d --name mi-app --network mi

montrar un volumen para Persistencia de datos:

bash  copiar código

docker run -d -v mi-volumen:/var/lib/mysql

de esta manera, aunque el contenedor se borre, los datos siguen guardados en el volumen.

• Conclusiones:

Docker ha transformado la forma en que se desarrolla, prueba y despliega software. Gracias a la contenerización, es posible tener entornos ligeros, portátiles y reproducibles en cualquier parte del mundo.

Aunque limitaciones (Persistencia de datos, Seguridad, curva de aprendizaje), sus beneficios superan ampliamente las desventajas. Su integración con herramientas de orquestación como Kubernetes y su adopción masiva lo consolidan como una de las tecnologías más importantes en el presente y futuro del desarrollo del software.

Con todo lo anterior ya tienes:

- Historia y contexto
- Comparaciones y técnicas
- Arquitectura y flujo
- Ventajas, usos, limitaciones
- Ejemplos de Dockerfile, comandos y docker-compose.
- Ejemplos Prácticos con redes y volúmenes.

Esto es word, con títulos y espaciado normal, ~~no~~ sin problemas.