Tema 3 Contenedores y Docker

Indice

Introducción a la contenedorización.

Conceptos básicos de Docker: imágenes, contenedores, Dockerfiles

Qué es Docker

¿QUÉ ES DOCKER?

- **FUNDAMENTOS DE DOCKER**

- Docker es una plataforma de software que permite crear, implementar y gestionar aplicaciones mediante contenedores. Facilita la portabilidad y escalabilidad de aplicaciones complejas.

Qué es Docker

Qué **NO** es Docker

- Hypervisor
- Virtualización (Pura)
- Clon de MS Paint

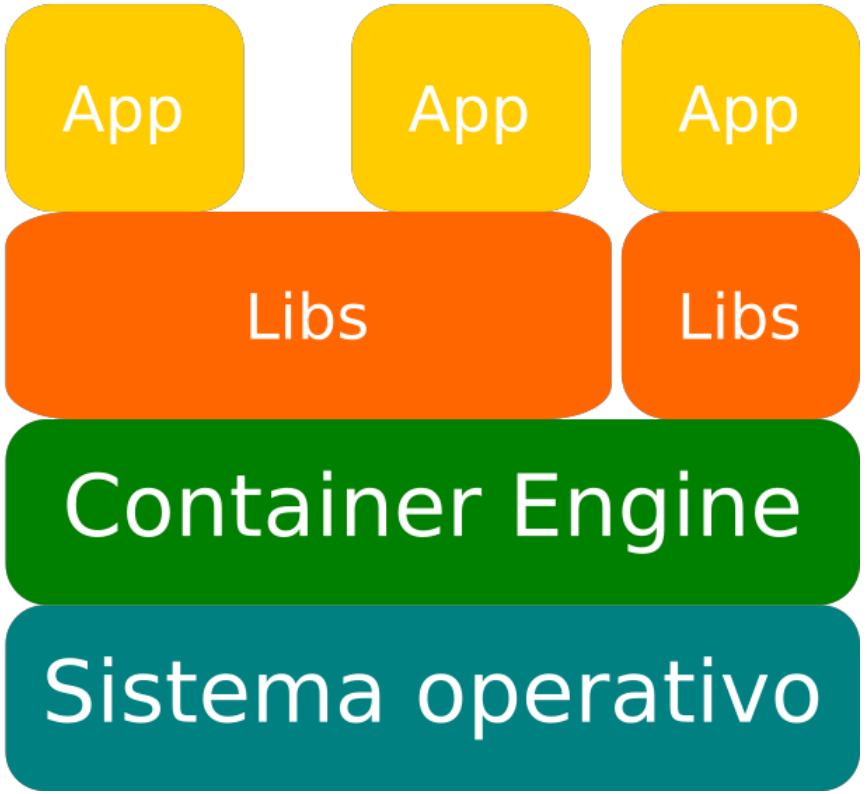
Qué **ES** Docker

- Manejador de contenedores
- Paravirtualización
- Abtracción de nombres de espacios a nivel de sistema operativo

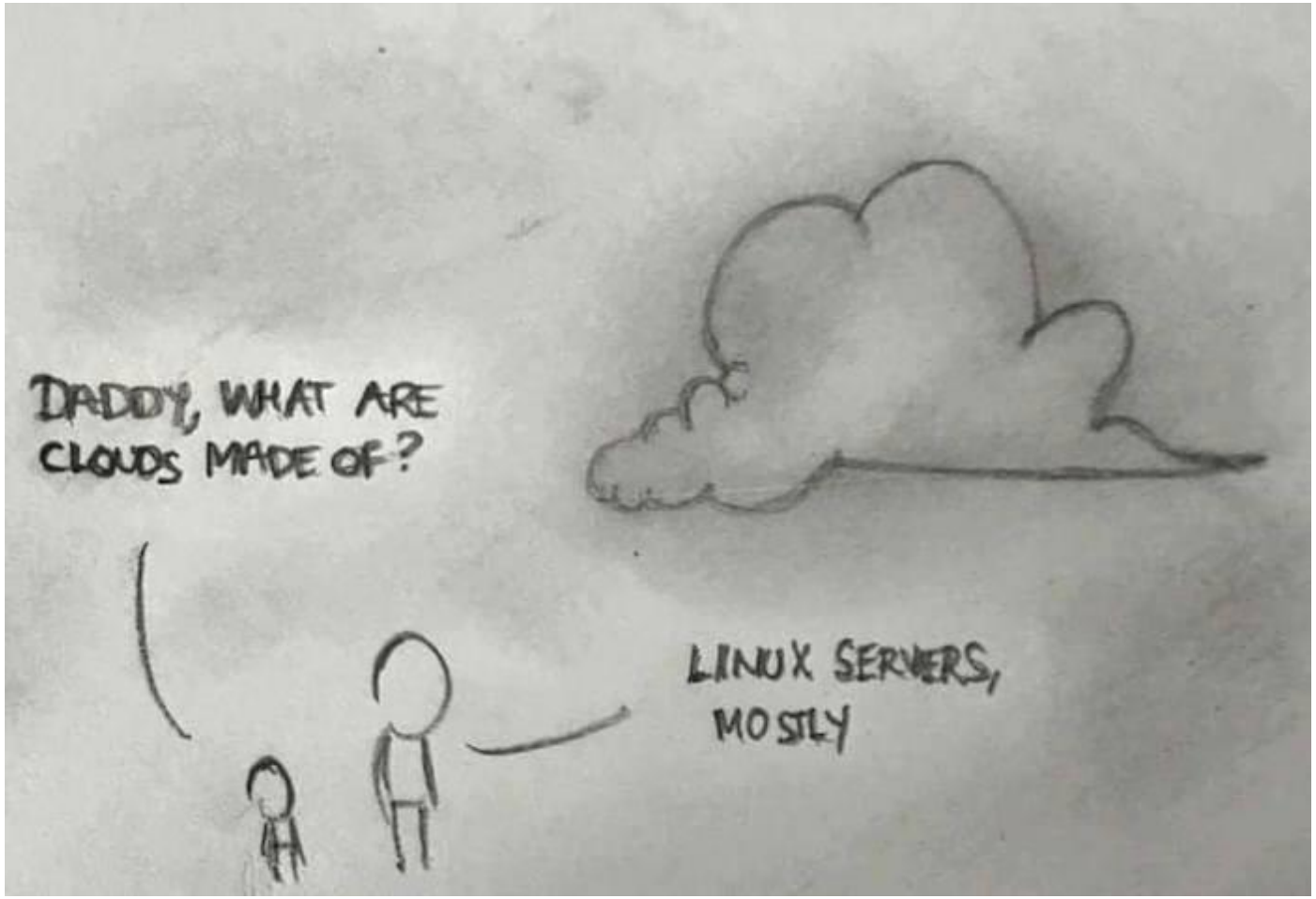
Características

- Contenedores autogestionados
- Abstracción de dependencias instaladas
- Escalabilidad horizontal
- Abstracción y encapsulación de clusters de servicios
- Almacenamiento ligéro (imágenes versionadas)
- Abstracción del sistema host
- Habilidad de compartir imágenes con la comunidad
- FOSS; Es Software Libre

Paravirtualización



Docker es Linux

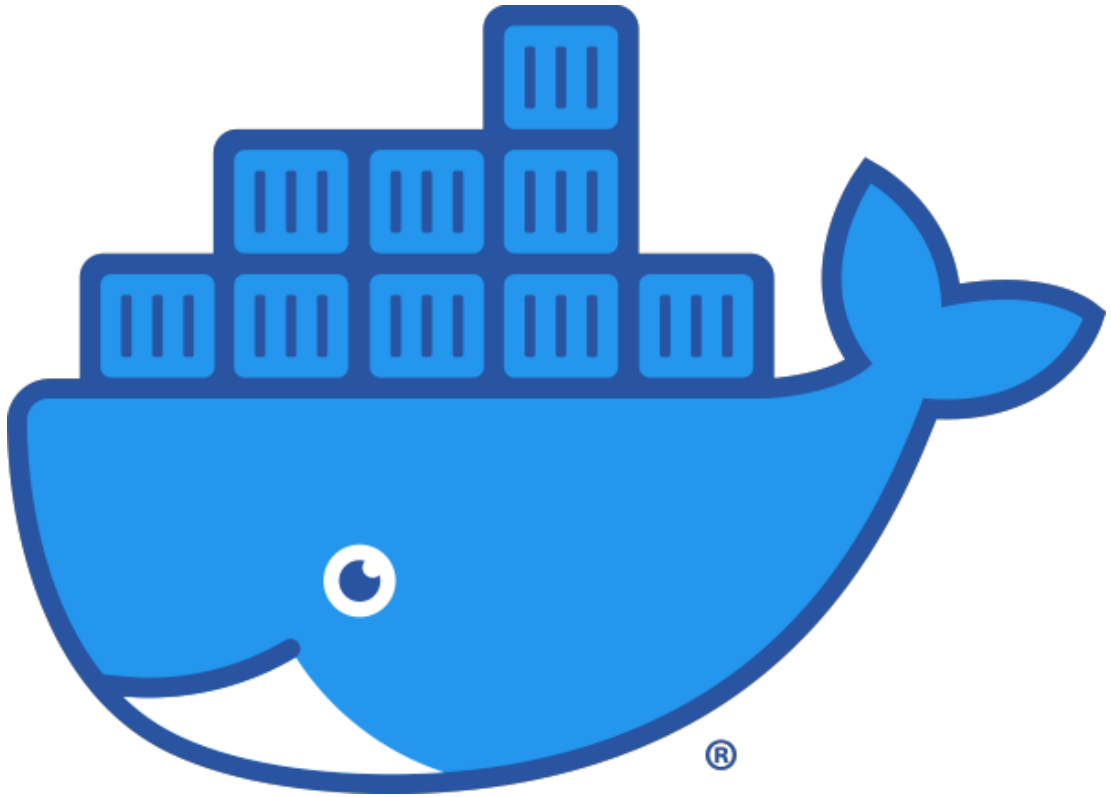


- **Linux**: Nativo
- **Windows < 10, 2016 Server**: Linux sobre VirtualBox
- **Windows 10 & 2016 Server**: Hyper-V (Linux subsystem)
- **MacOS**: Linux sobre VirtualBox

Contenedores e Imágenes

Containers

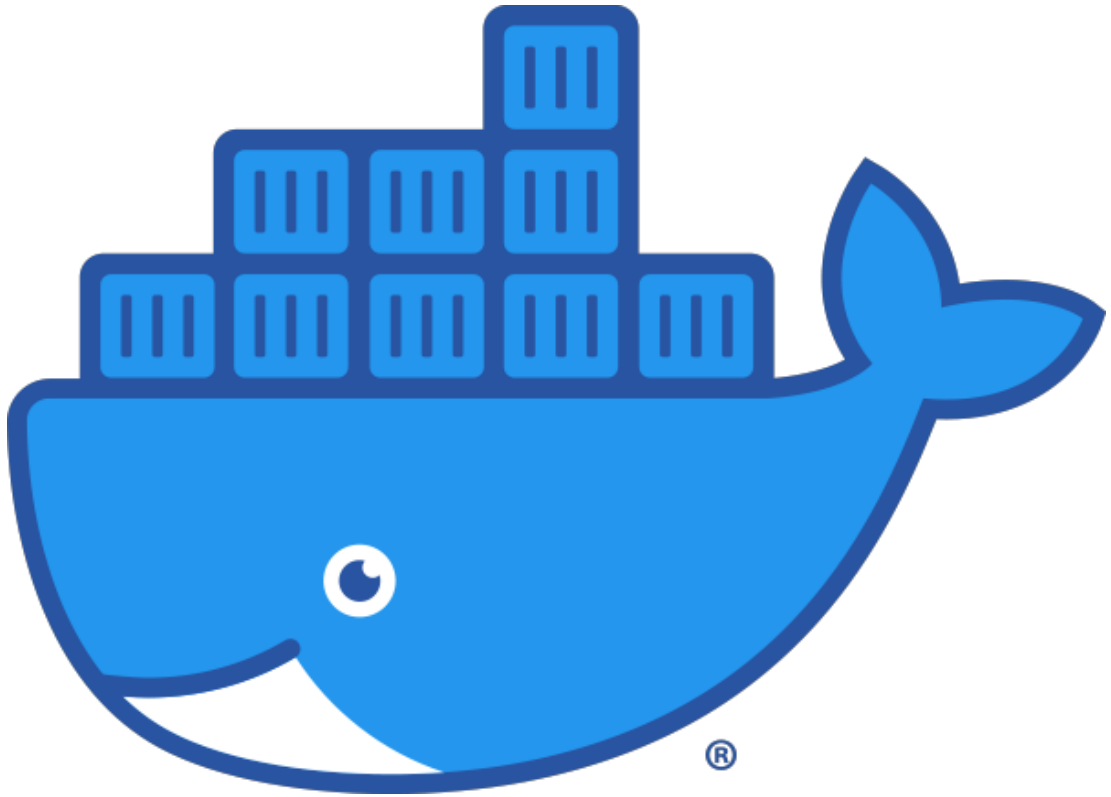
Un *conteiner* es una unidad estándar de software en la cuál están empaquetadas las dependencias necesarias para correr **una aplicación específica**.



- Estándar
- Liviana
- Segura

Images

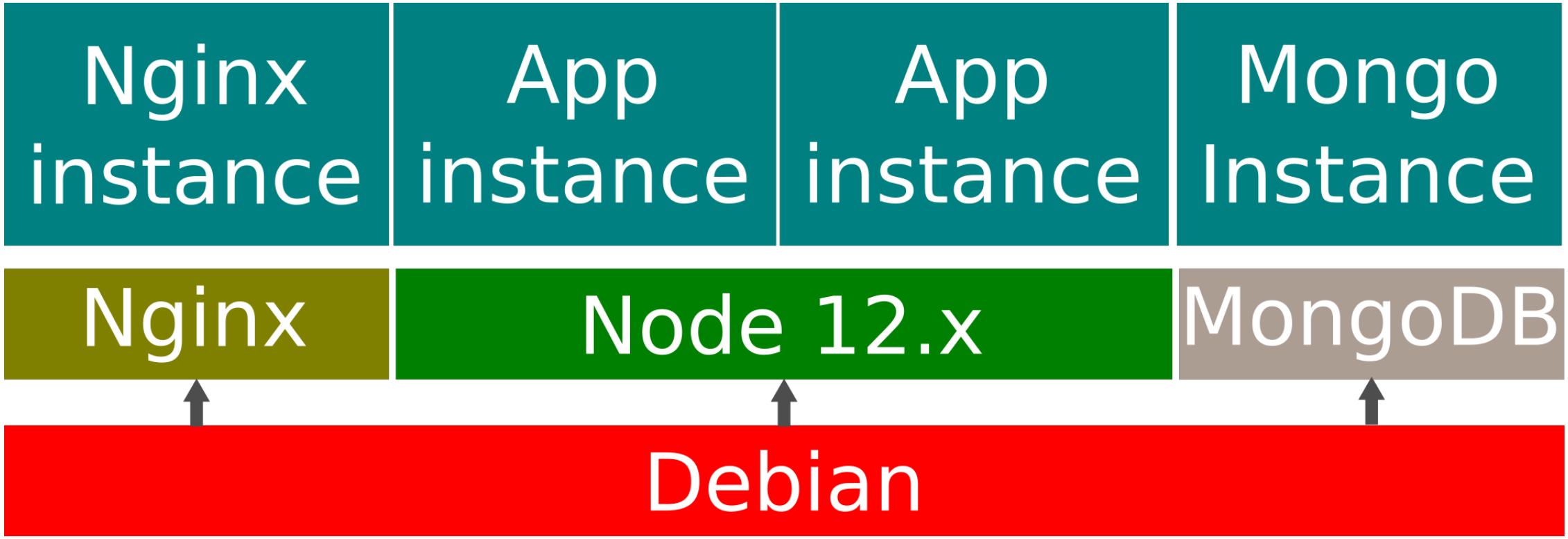
Una *imágen* es una especie de *snapshot* de un contenedor, de la que basa el container, los archivos necesarios para poder ser instanciados en forma de containers, inmutables y armados desde una receta.



Esta imágenes pueden ser

extendidas desde otras, ya que están conformadas por layers.

Images


Nginx instance
App instance
App instance
Mongo Instance
Nginx
Node 12.x
MongoDB
Debian

Volúmenes (Volumes)

- Los volúmenes se utilizan para almacenar datos persistentes en Docker,
- El sistema de archivos de un contenedor es temporal.
- Los volúmenes permiten compartir datos entre contenedores o entre un contenedor y el sistema anfitrión, garantizando que los datos persistan aunque el contenedor se elimine.

Dockerfile

Dockerfile

Un **Dockerfile** es un archivo que contiene una definición, al estilo *receta* de como se va a crear una imágen. Esta siempre depende de otra, extendiendo la “base”.

Cada linea representa un estado intermedio (una revisión), quedando guardado solo los cambios aplicados. Esto acelera mucho el proceso de buildeo y actualización.

Dockerfile

```
1 FROM debian:latest
2
3 RUN apt-get update &&\
4     apt-get install -y toilet && \
5     apt-get clean -y
6
7 ENV MSG="Hello World"
8
9 CMD /usr/bin/toilet $MSG
```

Dockerfile

```
# Usar una imagen base de Node.js
FROM node:20-alpine

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copiar los archivos necesarios al contenedor
COPY package.json .
COPY package-lock.json .

# Instalar las dependencias
RUN npm install

# Copiar el resto de los archivos del proyecto
COPY public
COPY views
COPY . .

# Crear las carpetas uploads y logs si no existen


# Exponer el puerto 3000
EXPOSE 3000

# Comando para iniciar el servidor
CMD ["node", "app.js"]
```

docker build -t prueba/cmb:v1 .

Docker build

- El comando docker build se utiliza para construir imágenes Docker a partir de un Dockerfile y un "contexto“

```
docker build [OPCIONES] CONTEXTO
```

CONTEXTO:
•Es el conjunto de archivos y directorios que Docker necesita para construir la imagen. Generalmente, es el directorio donde se encuentra el Dockerfile.
•Docker solo puede acceder a los archivos dentro del contexto durante la construcción.

Parámetros más comunes:

-t, --tag nombre:etiqueta:

Asigna un nombre y opcionalmente una etiqueta a la imagen construida.
Ejemplo: docker build -t mi-aplicacion:v1 .

-f, --file ruta/al/Dockerfile:

Especifica la ruta al Dockerfile si no está en el directorio actual o si tiene un nombre diferente a Dockerfile.
Ejemplo: docker build -f mi-dockerfile .

--build-arg clave=valor:

Define variables de construcción que se pueden usar dentro del Dockerfile.
Ejemplo: docker build --build-arg VERSION=1.2.3 .

--no-cache:

No utiliza la caché de construcción, forzando la reconstrucción de todas las capas de la imagen.
Útil cuando necesitas asegurarte de que se aplican los últimos cambios.

--pull:

Siempre intenta descargar las imágenes base más recientes especificadas en el Dockerfile.
Ejemplo: docker build --pull -t mi-aplicacion:v1 .

--target etapa:

Especifica una etapa de compilación objetivo. Si hay varias etapas en el Dockerfile, solo se construye la etapa objetivo y las etapas anteriores.
Ejemplo: docker build --target builder -t mi-aplicacion:v1 .

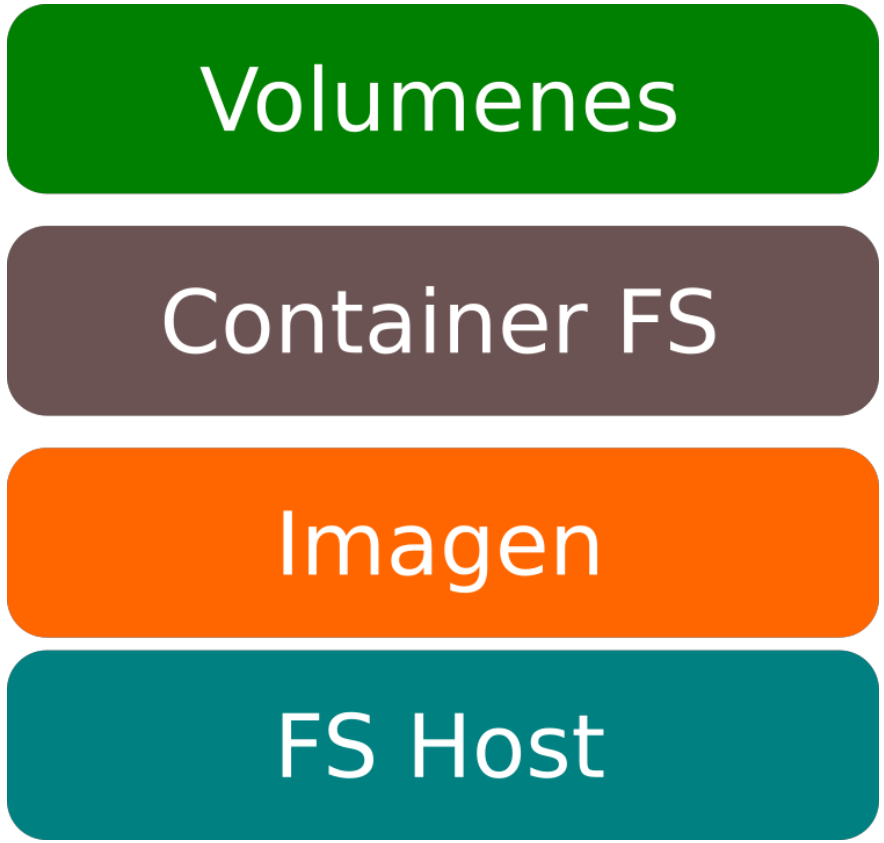
--network modo:

Establece el modo de red para las instrucciones RUN durante la compilación.
Ejemplo: docker build --network=host -t mi-aplicacion:v1 .

--label clave=valor:

Agrega metadatos a la imagen.
Ejemplo: docker build --label "com.example.version=0.1"

Storage

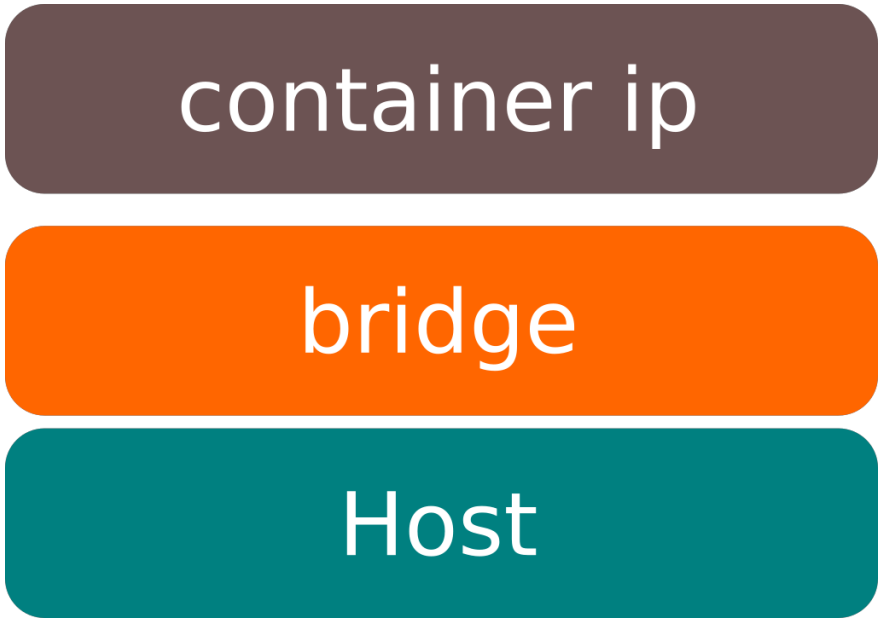


Son espacios de almacenamiento **manejados por Docker** para persistir datos generados por contenedores (ej: bases de datos, archivos de configuración).

No dependen del sistema de archivos del host (tu máquina física), lo que los hace portables y seguros.

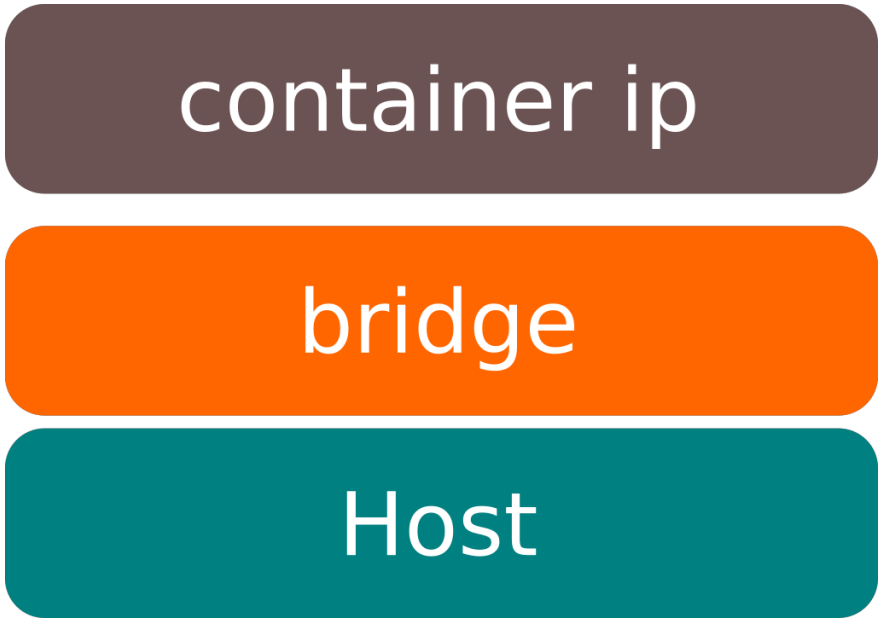
Networking

- bridge
- host
- none
- ~~overlay~~
- ~~macvlan~~



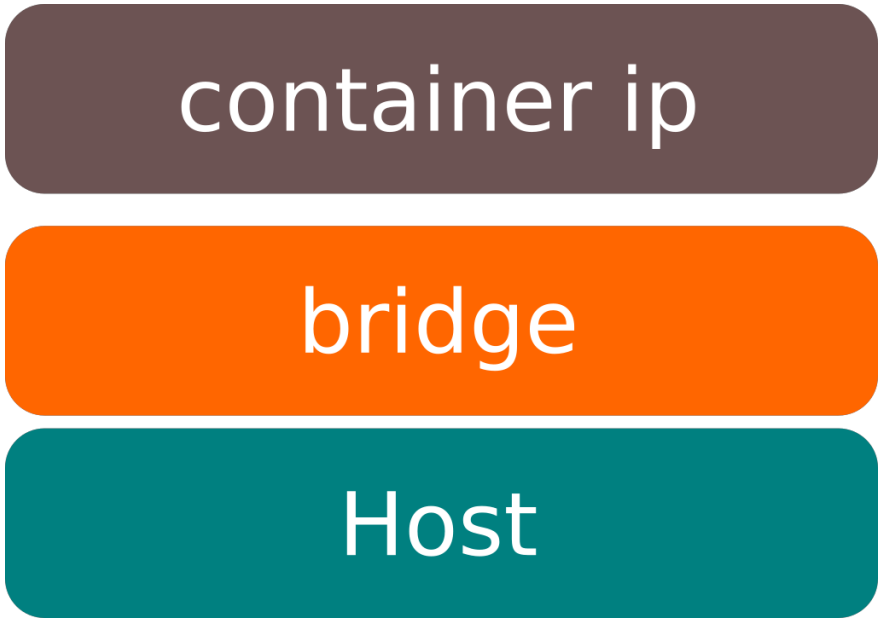
Networking

- binding
- linking

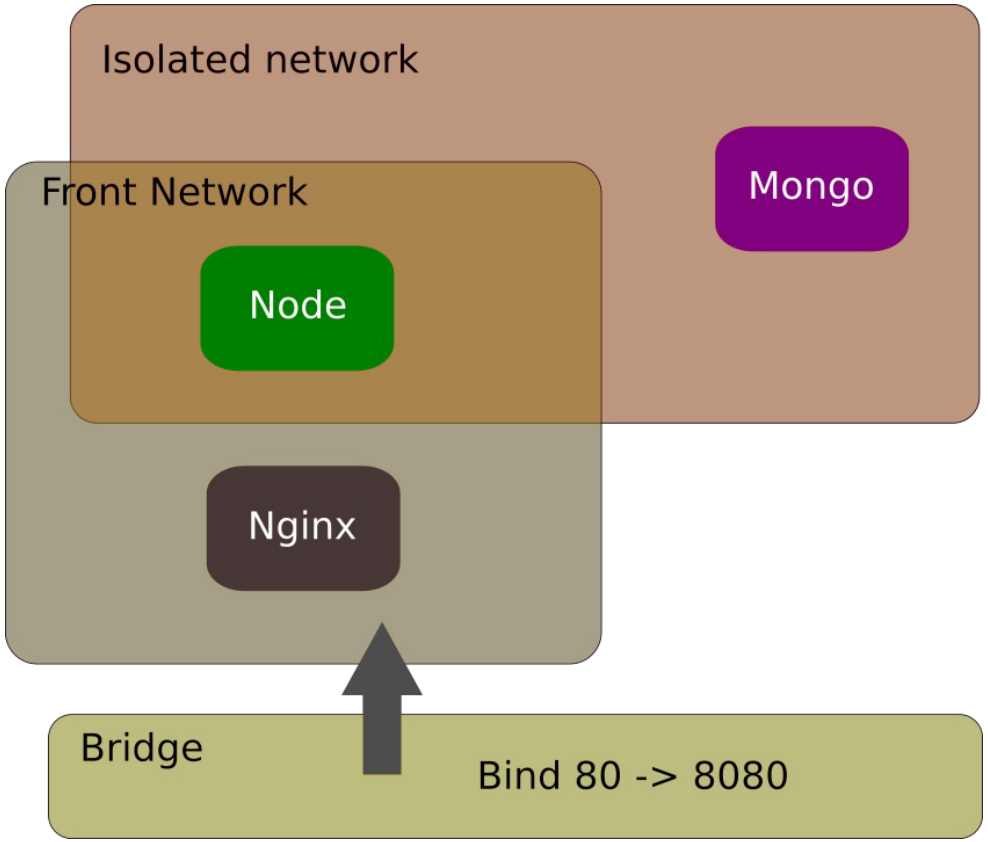


Networking

- bridge
- host
- none
- ~~overlay~~
- ~~macvlan~~

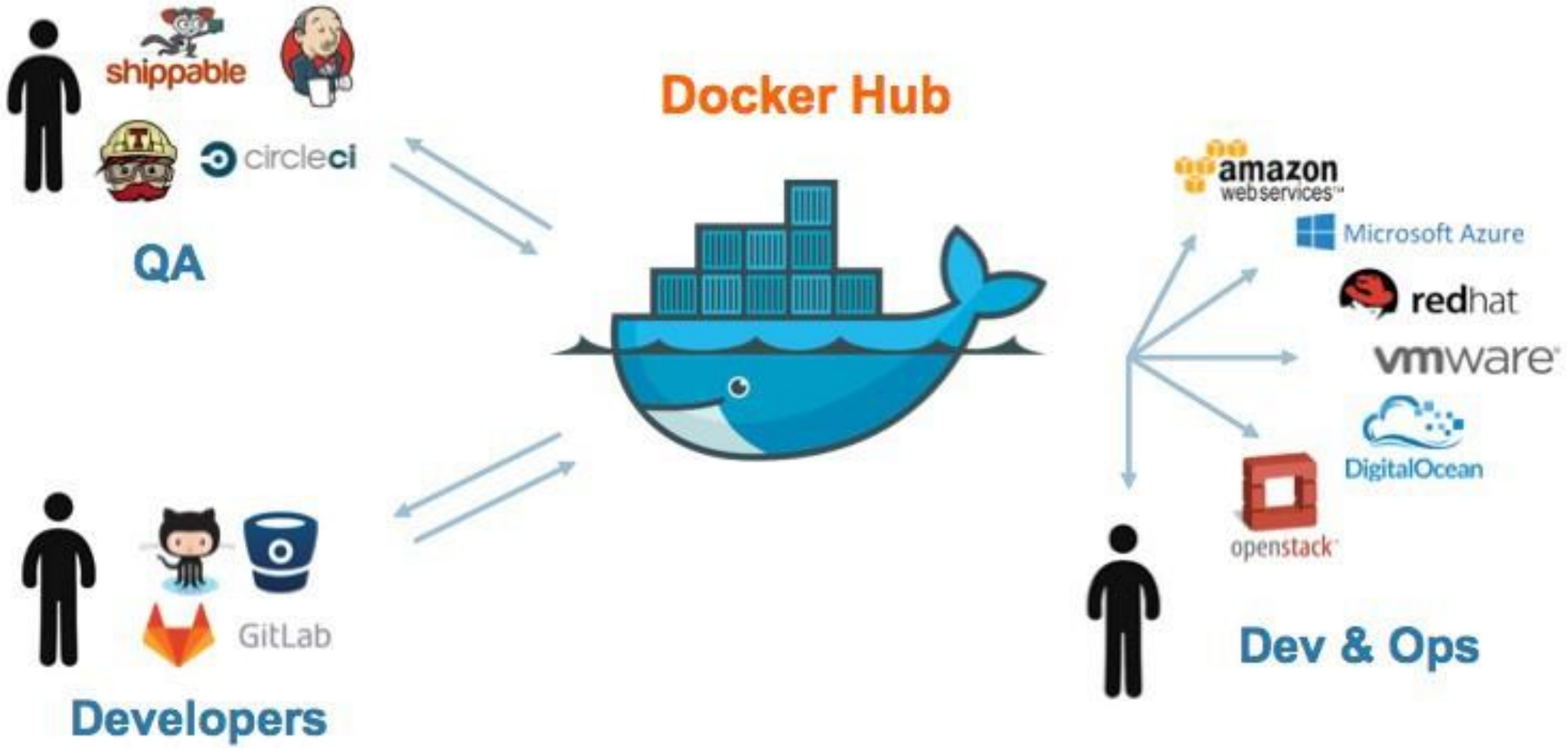


Networking


Isolated network
Front Network
Mongo
Node
Nginx
Bridge
Bind 80 -> 8080

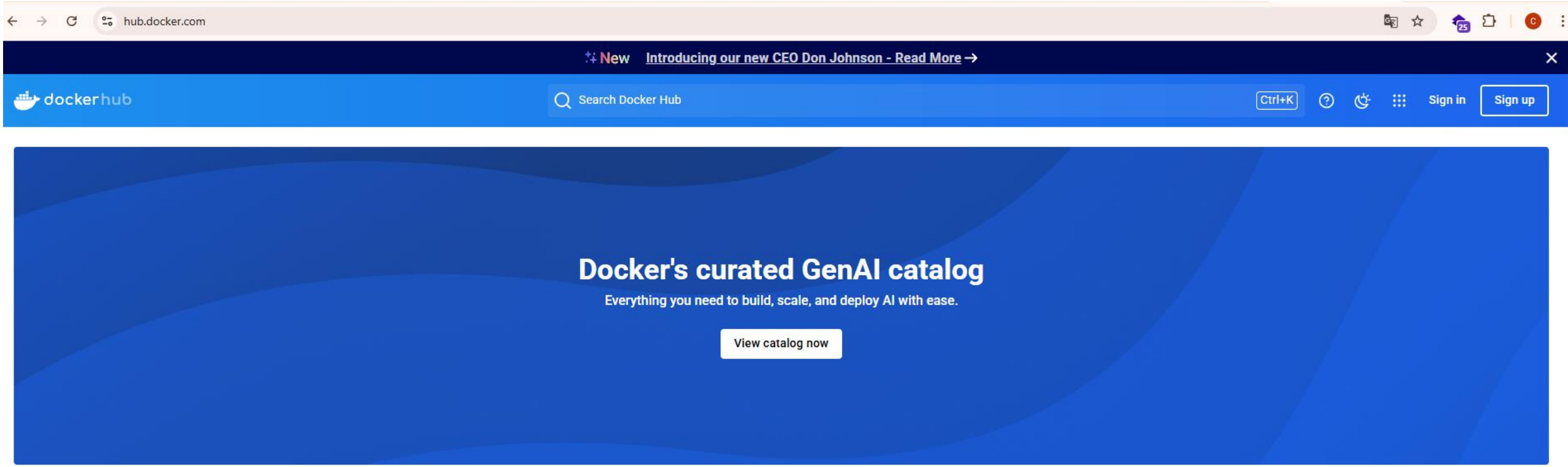
Docker HUB

Docker-HUB


shippable
circleci
QA
Docker Hub
amazon web services
Microsoft Azure
redhat
vmware
DigitalOcean
openstack
GitLab
Developers
Dev & Ops

Docker-HUB

Docker Hub es un servicio en la nube proporcionado por Docker que actúa como un **registro centralizado (registry)** para almacenar, compartir y administrar **imágenes de Docker**. Es similar a GitHub, pero en lugar de código, se usan imágenes de contenedores.



Para que sirve Docker Hub

- Almacenar imágenes: Subir tus imágenes personalizadas (ej: una app en Node.js con una base de datos).
- Descargar imágenes públicas: Acceder a imágenes oficiales como node:20-alpine, mysql, mongo, etc.
- Colaboración: Compartir imágenes con tu equipo o la comunidad.
- Integración con CI/CD: Automatizar despliegues usando herramientas como GitHub Actions o Jenkins.

Docker compose

Docker Compose

- Docker Compose es una herramienta de Docker que simplifica la gestión de aplicaciones multi-contenedor.
- En lugar de ejecutar manualmente múltiples comandos docker run para cada contenedor.
- Definis todos los servicios de tu aplicación en un archivo YAML llamado docker-compose.yml.

docker-compose.yml

Este archivo define los servicios que componen tu aplicación, sus dependencias, redes y volúmenes.

Cada servicio representa un contenedor, y puedes especificar la imagen Docker que se utilizará, las variables de entorno, los puertos que se expondrán y otros parámetros.

Comandos de Docker Compose

docker-compose up: Inicia todos los servicios definidos en el archivo docker-compose.yml.

docker-compose down: Detiene y elimina los contenedores, redes y volúmenes creados por docker-compose up.

docker-compose ps: Lista los contenedores en ejecución definidos en el archivo docker-compose.yml.

docker-compose logs: muestra los logs de los contenedores.

```
version: '3.8'

services:
  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: tu_password_root
      MYSQL_DATABASE: tu_basededatos
      MYSQL_USER: tu_usuario
      MYSQL_PASSWORD: tu_password
    ports:
      - "3306:3306"
    volumes:
      - db-data:/var/lib/mysql

  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      DB_HOST: db
      DB_USER: tu_usuario
      DB_PASSWORD: tu_password
      DB_NAME: tu_basededatos
    depends_on:
      - db

volumes:
  db-data:
```