

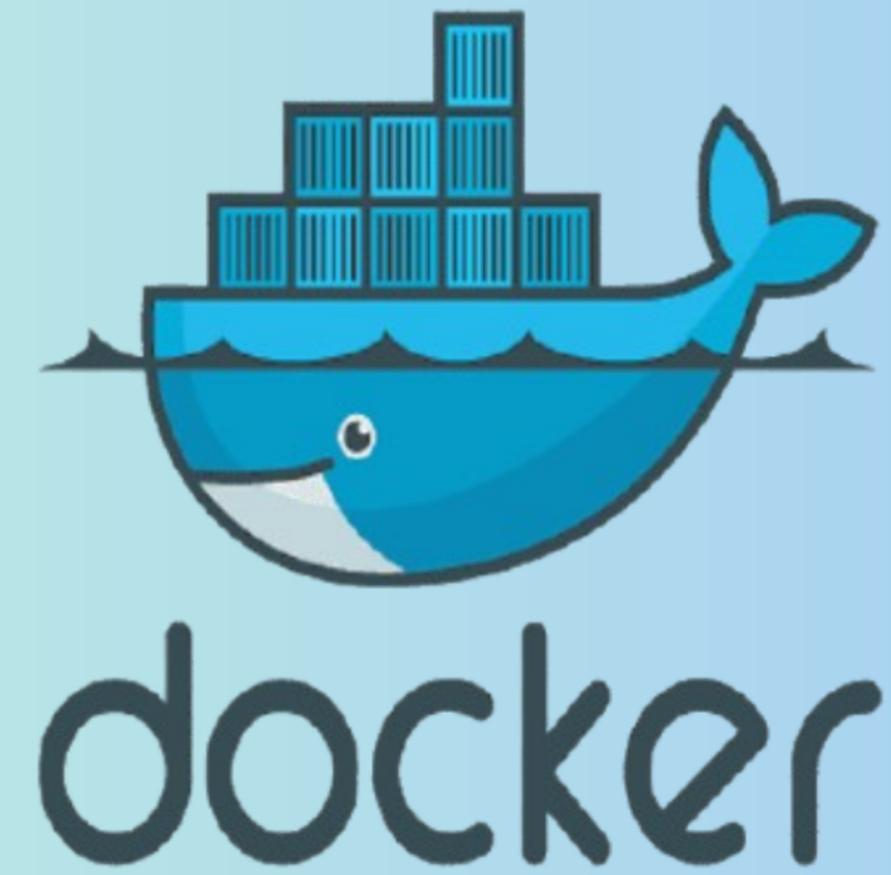
---

# **DOCKER**

---

# Que es Docker?

Docker es una plataforma que permite crear, ejecutar y gestionar aplicaciones dentro de contenedores.

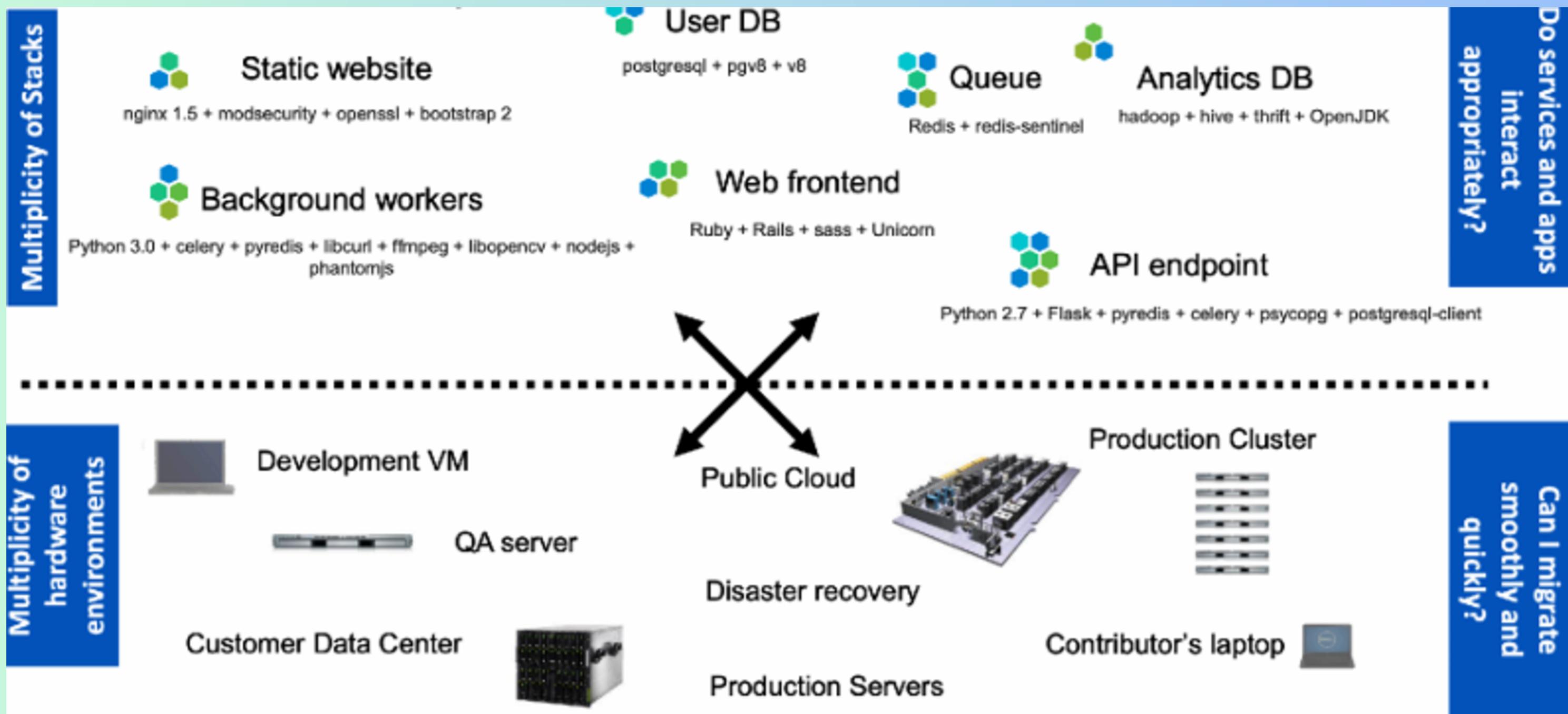


# Funcionamiento Basico

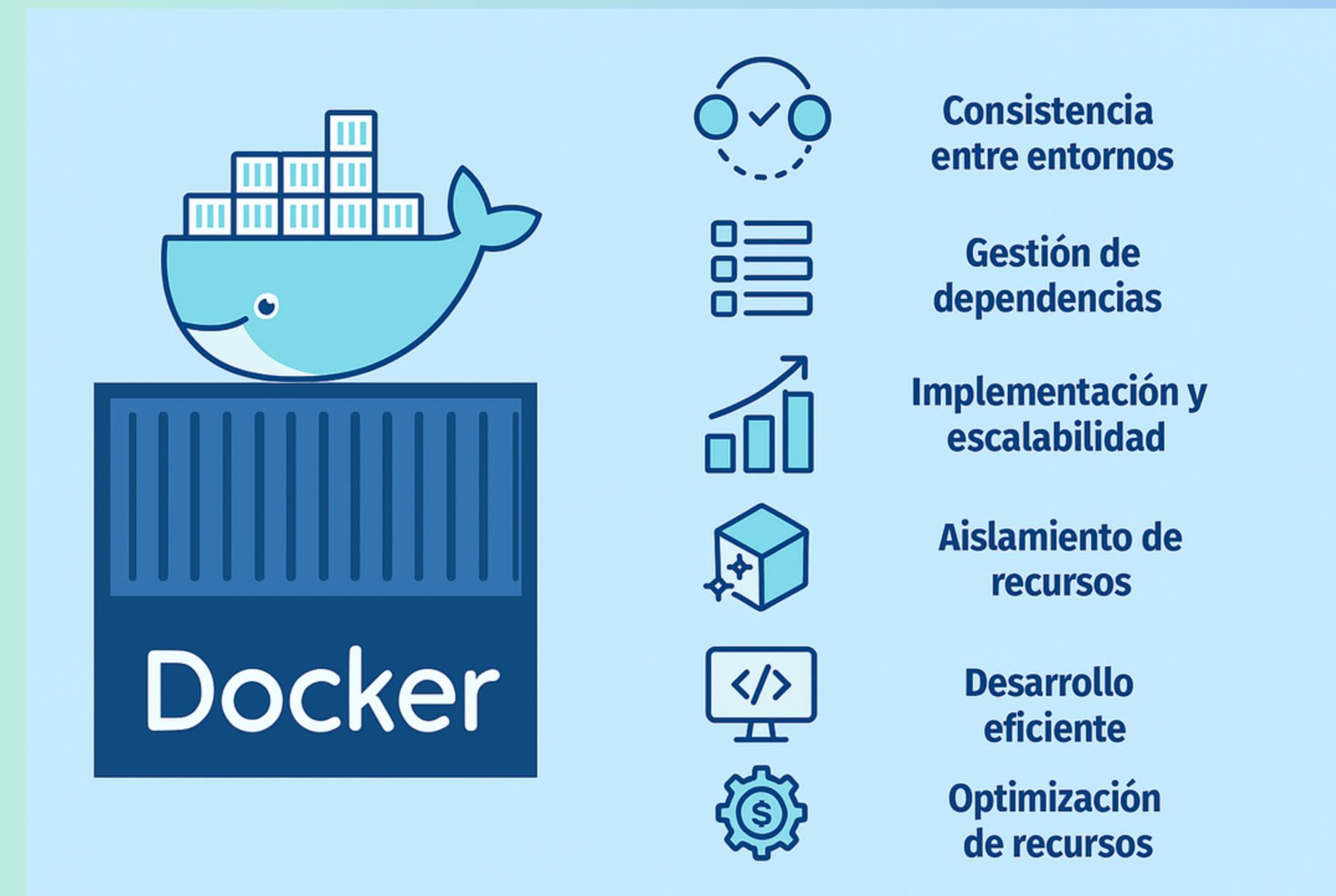
Docker utiliza contenedores para empacar aplicaciones y sus dependencias. Se crea una imagen a partir de un archivo llamado Dockerfile, se almacena en un registro y luego se ejecuta como un contenedor. Esto proporciona un entorno consistente y eficiente para desarrollar, distribuir y ejecutar aplicaciones.



# Que problema solventa?



# Que problema solventa?



# El Container



# Docker Proporciona



**Retorno de Inversión  
y ahorro de costos**



**Estandarización y  
Productividad**



**Eficiencia de Continuous  
Integration**



**Simplicidad y  
configuraciones mas rapida**



**Despliegue rápido**



**Despliegue continuo  
y pruebas**



**Plataformas multi-  
nube**



**Aislamiento  
Seguridad**

# Contenedores VS VM

## Contenedores

- Son completamente aislados
- Comparten el mismo SO (osea ocupan los recursos proporcionados de la máquina sin necesidad de crear una sub máquina (MV))
- Despliegues más rápidos, menos sobrecoste, más fáciles de migrar y reiniciar

## Maquinas Virtuales

- Básicamente son la emulación de un sistema operativo dentro de otro. Ej: linux en windows.
- Problema de recursos, ya que emulan todo un SO, y tener muchas MV satura la máquina servidor.
- Cada cambio requiere una nueva máquina virtual

# Beneficios Docker

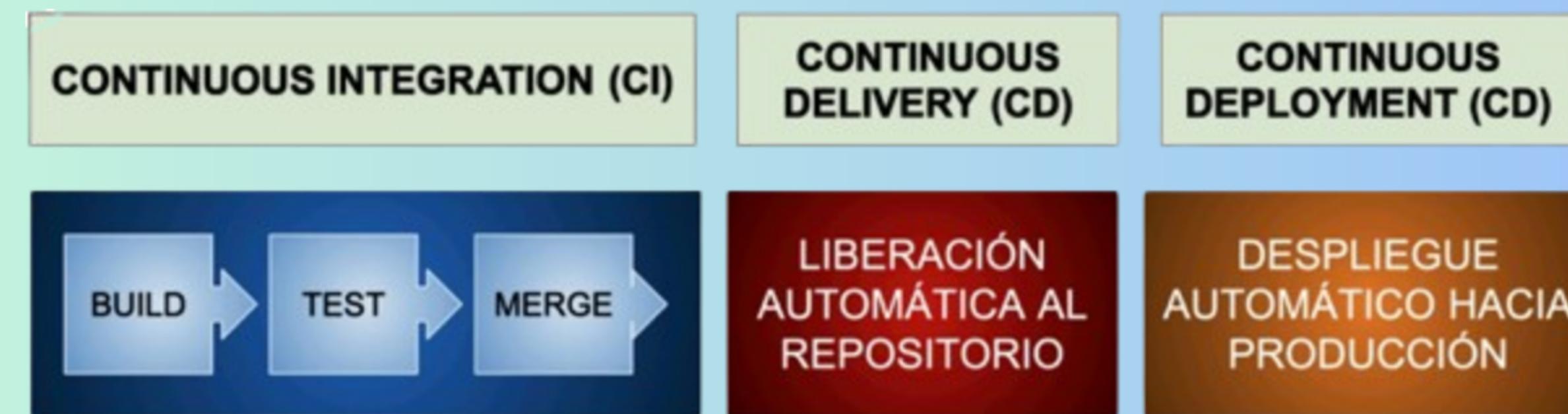
## Desarrolladores

- Entornos limpios, seguros y portables
- Despliegues reproducibles
- Aislamiento de aplicaciones
- Test, integración, empaquetado automatizado
- Menores problemas de compatibilidad
- Despliegues rápidos y baratos

## Devops

- Configura una vez, corre cientos
- Despliegues estandarizados y repetitivos
- Elimina inconsistencia entre entornos (Devel, qa, prod)
- Permite segregación de responsabilidades
- Mejora la velocidad de CI y CD
- Más ligeros que una MV

# CI - CD - CD



# Separación de responsabilidades

## Desarrolladores

- Preocupado por lo que hay dentro del container
  - Código
  - Librerías
  - Gestor de paquetes
  - Apps
  - Datos
- Todos los servidores linux son iguales

## Devops

- Preocupado por lo que hay fuera del contenedor
  - Logging
  - Acceso remoto
  - Monitorización
  - Configuración de red
- Todos los contenedores arrancan, paran, copian y migran del mismo modo

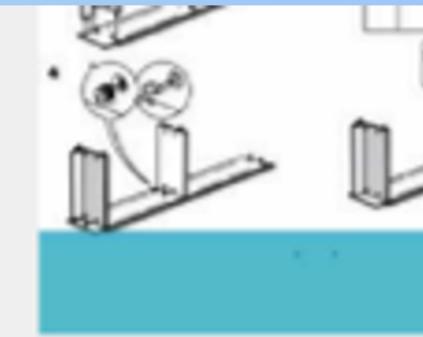
---

# ¿Por qué Docker?

- Corre en cualquier sitio (Linux)
  - Sin importar V kernel.
  - Sin importar distribución.
  - Container & Arquitectura de host deben emparejar.
  - Puede tener su propio /sbin/init.
- Corre cualquier cosa:
  - Si corre en el host, corre en el container.
  - Si corre en un kernel linux, correra.

# DockerFile

```
# A basic apache server.  
  
FROM ubuntu:14.04  
  
MAINTAINER Kimbro Staken version: 0.1  
  
RUN apt-get update && apt-get install -y apache2  
  
ENV APACHE_LOG_DIR /var/log/apache2  
  
EXPOSE 80  
  
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```



# Imagen

Fichero Binario que contiene todo el sistema de ficheros de un contenedor

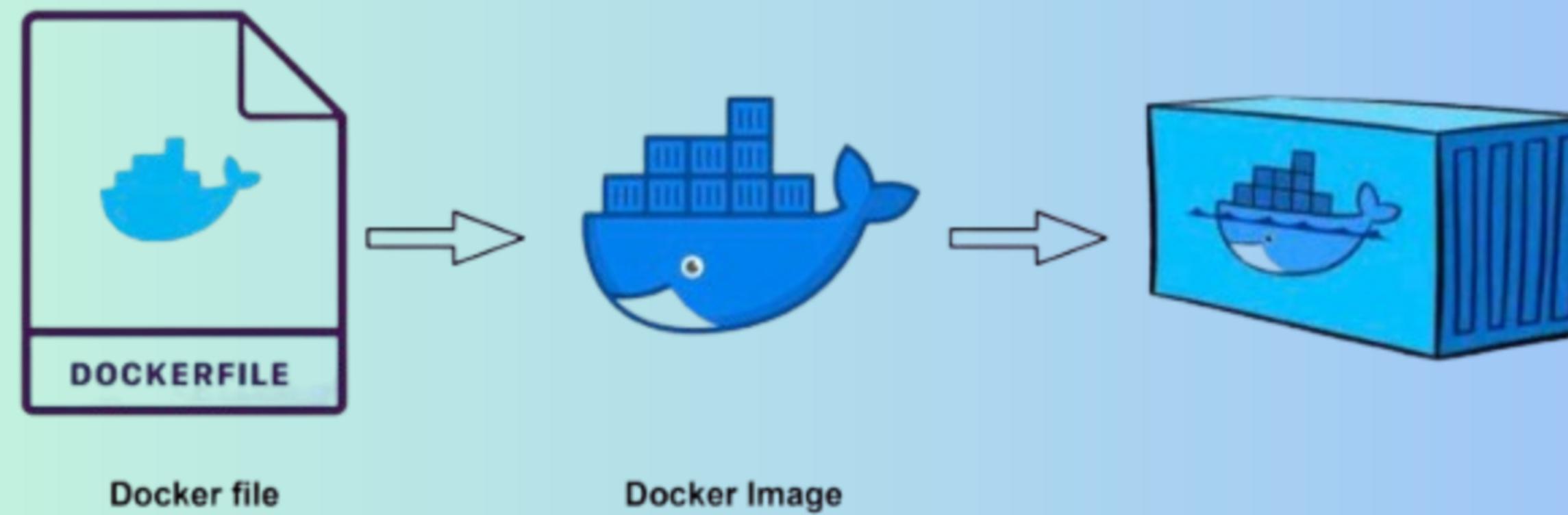
# Volumen

- Discos o directorios externos que podemos montar en el contenedor
- Recursos Externos (Alojados en el host) que sobreviven al contenedor
- Configuracion / DATOS / Recursos

# Registro

- Biblioteca de imágenes de contenedor
  - Listas para ser usadas
- Registro público
  - Compartidas por la comunidad
  - Libre Acceso
- Registro privado
  - Contenedores corporativos o privados

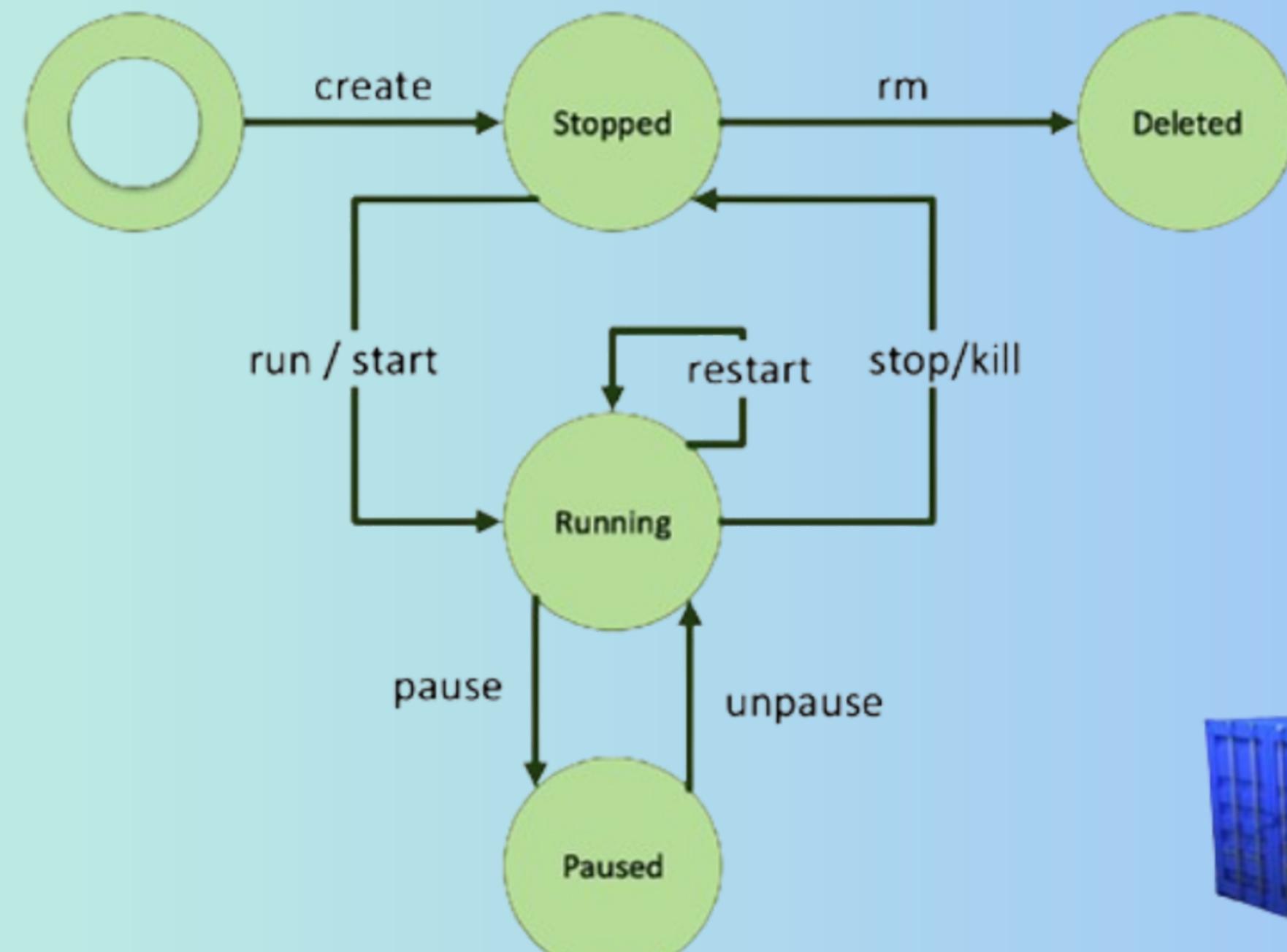
# Jerarquía



Docker file

Docker Image

# Ciclo de vida de un Contenedor



# Como hacer un Docker

- Instalación y actualización de paquetes del sistema operativo
- Establecer variable de entorno llamada Debian\_frontend
- Instalacion git

```
# Descarga la imagen de Ubuntu 14.04
FROM ubuntu:14.04

# Actualiza la imagen base de Ubuntu 14.04
RUN apt-get update

# Definir ambiente de entorno
ENV DEBIAN_FRONTEND noninteractive

# Instalar Git
RUN apt-get -qqy install git
```

---

# Como hacer un Docker

- **From:** indica la imagen base sobre la que se construirá la aplicación dentro del contenedor
- **RUN:** nos permite ejecutar comandos en el contenedor, ejemplo instalar paquetes o librerías. Hay dos formas de colocarlo
  - RUN <comando>
  - [“ejecutable”, “parametro1”, “parametro2”]
- **ENV:** establece variables de entorno para nuestro contenedor.
  - Ej: en un entorno DEBIAN\_FRONTEND noninteractive, el cual permite instalar un montón de archivos .deb sin tener que interactuar con ellos

---

# Como hacer un Docker

```
# Descarga la imagen de Ubuntu 16.04
FROM ubuntu:16.04

# Actualiza la imagen base de Ubuntu 16.04
RUN apt-get update

# Ejecuta el comando apt-get install y elimina determinados archivos y temporales
RUN apt-get install -y nginx \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Indica los puertos TCP/IP los cuales se pueden acceder a los servicios del container
EXPOSE 80

# Establece el comando del proceso de inicio del contenedor
CMD ["nginx"]
```

- 
- **CMD:**
    - instrucción nos provee valores por defecto a nuestro contenedor
    - Define una serie de comandos que solo se ejecutarán una vez que el contenedor se ha inicializado

---

# Docker Compose

- Orquesta contenedores en un mismo cliente
- Consiste en un archivo Yaml, este define de forma declarativa los contenedores que se van a desplegar, así como la dependencia entre ellos.
- Para despliegue, se basa en la definición de servicios, que referencian imágenes docker de un registro y las características de los contenedores que se desean desplegar

# En qué momento usar docker compose?

- Entornos que se encuentran en producción
- Entornos que se encuentran en desarrollo
- En entornos que se encuentren en fase de pruebas
- En flujos de Continuous integration



# Servicios y recursos

- Dentro de docker-compose.yaml podemos definir diferentes recursos a desplegar
- A la hora de desplegar nuestras aplicaciones, lo primero es definir los servicios que vamos a desplegar.



---

# Servicios

- Cada servicio despliega un contenedor, asociado a una imagen de docker.
  - Dentro del servicio se puede definir características de cada contenedor
    - Nombre
    - Imagen
    - Puertos que expone
    - Volúmenes
    - Redes a las que se conecta

---

# Servicios

- Para definir un servicio en docker-compose.yaml, debemos definir un bloque con el nombre del servicio, y dentro una serie de propiedades que definen las características del contenedor:
  - **image**
  - **[ container name ]**
  - **[ build ]**
  - **[ command ]**
  - **[ ports ]**
  - **[ volumenes ]**
  - **[ environment ]**
  - **[ depends\_on ]**
  - **[ networks ]**
  - **[ restart ]**

# Volúmenes

- Mecanismo que tienen los contenedores para persistir y compartir datos
- Consumen o generan ficheros en un directorio del sistema de ficheros del host.
- Dentro del bloque volumen se pueden definir las siguientes propiedades
  - [ **driver** ]
  - [ **driver\_opts** ]
    - **type**
    - **device**
    - **o**
  - [ **external** ]
  - [ **labels** ]
  - [ **name** ]
  - [ **scope** ]

**Todas son opcionales, pero es recomendable definir al menos un nombre y la ruta en el sistema de archivos del host.**

# Ejemplo

```
volumes:  
  my-volume:  
    driver: local  
    driver_opts:  
      type: none  
    device: /path/to/my-volume  
    o: bind  
  external: false  
labels:  
  - "com.example.description=Volume for my service"  
name: my-volume  
scope: local
```

Referenciado en el servicio como:

```
services:  
  my-service:  
    image: my-image  
    volumes:  
      - my-volume:/path/in/container
```

Rutas definidas en la propiedad device del bloque volumes debe existir en el sistema de archivos del host.

---

# Redes

- Se usa para que los contenedores puedan comunicarse entre si, mediante una red común, siguiendo el modelo Container Network Model
  - SandBox
  - EndPoints
  - Networks
- Otras varias implementaciones

| Nombre  | Alcance | Descripción  |
|---------|---------|--|
| bridge  | Local   | Red por defecto de Docker. Crea una red virtual en el host, que conecta los contenedores por medio de un bridge virtual.                       |
| host    | Local   | Deshabilita el aislamiento entre los contenedores y el host, no hace falta exponer puertos   |
| overlay | Global  | Permite conectar múltiples demonios Docker entre sí y habilitar la comunicación entre servicios distribuidos .                                 |
| ipvlan  | Global  | El usuario obtiene el control total del direccionamiento IPv4 e IPv6 de la red.  |
| macvlan | Global  | Permite asignarle una dirección MAC a un contenedor, haciendo que aparezca como un dispositivo físico en la red. El tráfico se dirige por MAC. |
| none    | Local   | Desactiva toda la gestión de red, normalmente usado en conjunto con un driver de red propio.   |