

# DOCKER

## • Introducción a Docker

Docker se ha consolidado en una herramienta fundamental en el desarrollo. Consiste en empaquetar aplicaciones y sus dependencias dentro de contenedores ligeros, asegurando que puedan ejecutarse de manera uniforme en cualquier entorno, desde el desarrollo local hasta la producción en la nube. Esto resuelve problemas clásicos de compatibilidad entre entornos y facilita la portabilidad, escalabilidad y estandarización de los despliegues.

## Conceptos Clave

- **Imagen:** Plantilla inmutable que contiene el Sistema base, librerías y la aplicación.
- **Contenedor:** Instancia en ejecución de una imagen; equivalente a un proceso aislado.
- **Dockerfile:** Archivo de instrucciones para construir imágenes personalizadas.
- **Docker Hub:** Repositorio público que aloja imágenes compartidas por la comunidad y oficiales.
- **Volumen:** Mecanismo para la persistencia de datos entre ejecuciones.

## • Instalación y configuración

En sistemas Linux, Docker se instala mediante repositorios oficiales. Una vez instalado, Docker se configura principalmente a través de archivos del Sistema:

Archivo

Propósito

/etc/docker/docker.json → Configuración avanzada (ejemplo: drivers de log(s), mirrors de registro)

/var/lib/docker

→ Ubicación por defecto de imágenes, capas y volúmenes

/var/run/docker.sock

→ Socket de comunicación entre la CLI y el demonio de Docker.

## Ejemplo de configuración de logs en daemon.json

```
{  
  "log-driver": "json-file",  
  "log-opt": {  
    "max-size": "10m",  
    "max-file": "3"  
  }  
}
```

Evita que los contenedores generen archivos de logs excesivos que saturen el disco.

## • Imágenes

Son plantillas inmutables que contienen el sistema operativo base, librerías, configuraciones y la aplicación lista para ejecutarse.

Características principales.

- Inmutables: Una vez creadas no cambian; cualquier ajuste requiere construir una nueva imagen.

- Capas (layers): Cada instrucción del Dockerfile genera una capa. Estas capas se cachearán, lo que optimiza la construcción y reduce el espacio usado.

- Reutilizables: Múltiples contenedores pueden levantarse a partir de una misma imagen.

Flujo de uso de imágenes.

1. Buscar y descargar:

```
docker pull nginx:1.25
```

2. Ejecutar el contenedor:

```
docker run -d -p 8080:80 nginx:1.25
```

3. Modificar y personalizar: creando un dockerfile.

4. Construir nueva imagen:

```
docker build -t mi-app:1.0 .
```

Buenas prácticas con imágenes

- Usar etiquetas (tags) fijas en lugar de la `latest` para evitar problemas de compatibilidad.

- Preferir imágenes oficiales y minimalistas (`alpine`, `slim`) para reducir vulnerabilidades.

- Escanear imágenes regularmente con:

```
docker scan mi-app:1.0
```

## • Contenedores

Los contenedores son instancias en ejecución de imágenes. Se pueden ver como procesos aislados que corren en el host, pero que comparten el kernel del sistema operativo.

Propiedades clave:

- Aislados: Cada contenedor tiene su propio sistema de archivos, red y procesos.
- Efímeros: Pueden eliminarse y recrearse fácilmente sin afectar la aplicación.
- Escalables: Se pueden ejecutar múltiples instancias de un mismo servicio.

Ejemplos de creación de contenedores:

1. Contenedor Simple.

```
docker run hello-world
```

2. Contenedor en background:

```
docker run -d --name web -p 8080:80 nginx
```

3. Contenedor con variables de entorno.

```
docker run -d -e MYSQL_ROOT_PASSWORD=secret mysql:8
```

Gestión de Contenedores.

• Listar contenedores activos.

```
docker ps
```

• Listar todos (incluyendo detenidos)

```
docker ps -a
```

• Detener y eliminar:

```
docker stop web docker rm web
```

Buenas Prácticas

- Usar `--restart always` en producción.
- Nombrar los contenedores (`--name`) para facilitar su gestión.
- Limitar recursos (`--cpus, -m`) para evitar que consuman todo el host.

## • Dockerfile

El Dockerfile es un archivo de texto que define paso a paso como construir una imagen.

Instrucciones más comunes:

- FROM : define la imagen base.
- WORKDIR : establece el directorio de trabajo.
- COPY / ADD : copia archivos al contenedor.
- RUN : ejecuta comandos en la construcción.
- CMD : comando por defecto al ejecutar el contenedor.
- EXPOSE : documenta el puerto expuesto.
- ENV : define variables de entorno.

## Ejemplo (Node.js)

```
FROM node:18-alpine
WORKDIR /app
COPY package.json / 
RUN npm install --production
COPY .
EXPOSE 3000
CMD ["node", "server.js"]
```

## Ejemplo con .dockerignore

- dockerignore (evita copiar basura a la imagen)
- ```
node_modules
*.git
*.log
```

## Buenas prácticas

- Usar imágenes ligeras (alpine, slim)
- Agrupar comandos RUN para reducir capas

```
RUN apt-get update && apt-get install -y \
curl \
git \
88 &m -rf /var/lib/apt/lists/*
```

- Evitar copiar todo el contexto (COPY...) sin .dockerignore.
- Separar entornos de build y runtime (multistage builds).

## • Conexión entre imágenes, contenedores y Dockerfile

- 1- El Dockerfile define como se construye una imagen.
- 2- Una imagen sube de plantilla estética lista para ejecutar.
- 3- El contenedor es la instancia dinámica de esa imagen en ejecución.

Dockerfile → Docker build → Image → Docker run → Container

## Docker Compose

Docker Compose es una herramienta que permite definir y administrar aplicaciones multi-contenedor usando un solo archivo (`docker-compose.yml`). Es especialmente útil en entornos de desarrollo y prueba, donde varios servicios deben ejecutarse de manera coordinada.

Características principales:

- Declarativo: se describe la infraestructura en YAML.
- Reproducible: el mismo `docker-compose.yml` funciona en cualquier host.
- Orquestación básica: facilita levantar, detener y escalar servicios.
- Integración con Volumenes y redes: todo se gestiona de forma centralizada.

Comandos esenciales:

- Levantar servicios

```
docker-compose up -d
```

- Detener servicios

```
docker-compose down
```

- Escalar instancias

```
docker-compose up -d --scale api=3
```

- Ver logs de todos los servicios

```
docker-compose logs -f
```

Ejemplo: Aplicación web con API y Base de Datos.

version: "3.9"

services:

db:

image: postgres:15

environment:

POSTGRES\_USER: admin

POSTGRES\_PASSWORD: secret

POSTGRES\_DB: myapp

volumes:

- db-data:/var/lib/postgresql/data

api:

build: ./api

ports:

- "5000:5000"

environment:

- DB\_HOST = db
- DB\_USER = admin
- DB\_PASS = secret

depends\_on:

- db

frontend:

build: ./frontend

ports:

- "3000:80"

depends\_on:

- api

volumes:

db\_data:

Con este archivo, en un solo comando (docker-compose up -d) se levantan 3 servicios (frontend, API y base de datos) que se comunican entre sí en la misma red creada automáticamente.

## • Seguridad y Buenas prácticas en Docker

La Seguridad es crítica en entornos de contenedores ya que un error de configuración puede exponer vulnerabilidades en producción.

### 1 Manejo de imágenes.

- Usar imágenes oficiales o verificadas desde Docker Hub o repositorios privados.
- Escanear imágenes con herramientas como docker scan o Trivy.  
docker scan mi-app:1.0
- Evitar imágenes muy pesadas: reducen la superficie de ataque.

### 2 Reducción de privilegios

- No ejecutar procesos como root.
- En el Dockerfile, definir un usuario no privilegiado:  
RUN addgroup app 98 adduser -D -G app app  
USER app

### 3 Redes y Puertos.

- Evitar exponer todos los puertos con -P 0:0:0:port
- Usar redes personalizadas para aislar servicios:  
docker network create internal-net
- Permitir sólo la comunicación necesaria entre contenedores.

## 4. Gestión de Secretos

Nunca almacenar contraseñas en variables de entorno dentro del Dockerfile.

Usar Docker secrets o archivos externos.

Servicios:  
db:

image: mysql:8  
secrets:  
- db-password

Secrets:  
db-password:  
file: ./db-password.txt

## 5. Límites de recursos

Evitar que un contenedor consuma todos los recursos del host:

```
docker run -m 512m --cpus=1 my-app
```

## • CI/CD con Docker

La integración continua (CI) y la entrega/despliegue continuo (CD) son prácticas fundamentales en el desarrollo moderno de software. Su objetivo es garantizar que cada cambio en el código fuente pueda ser construido, aprobado y desplegado de manera automática, reduciendo errores humanos y acelerando la entrega de valor.

### Integración continua (CI)

CI asegura que cada vez que se incorpora código al repositorio:

1. El proyecto se compila en un entorno controlado.
2. Se ejecutan pruebas automáticas para validar la calidad del código.
3. Se genera una imagen Docker de la aplicación.
4. La imagen se almacena en un registro centralizado (ejemplo: Docker Hub, GitHub Container Registry, AWS ECR).

De esta manera cada commit produce una versión trazable y portable de la aplicación.

### Entrega y despliegue continuo (CD)

CD extiende el proceso anterior para que una vez validada la nueva versión, ésta sea desplegada automáticamente en el entorno de pruebas o producción.

El flujo típico es:

1. Descargar la nueva imagen desde el registro.
2. Detener el contenedor en ejecución.
3. Levantar un nuevo contenedor basado en la imagen actualizada.

Este ciclo permite que las aplicaciones se mantengan siempre en su versión más reciente, con cambios aplicados de forma confiable.

# Ejemplo de Pipeline con GitHub Actions.

El siguiente flujo construye y publica una imagen Docker cada vez que se realiza un push en la rama principal.

name: CI/CD Docker

on:

push:

branches: ["main"]

jobs:

build:

run: -on: ubuntu-latest

steps:

-name: Clonar repositorio

users: actions/checkout@v3

-name: Iniciar sesión en Docker Hub

users: docker/login-action@v2

with:

username: \${{ secrets.DOCKER\_USER }}

password: \${{ secrets.DOCKER\_PASS }}

-name: Construir imagen

run: docker build -t usuario/miapp:latest.

-name: Subir imagen a Docker Hub

run: docker push usuario/miapp:latest

- Cada push a la rama main activa el pipeline.

- Se construye la imagen Docker de la aplicación.

- La imagen se sube a un registro (Docker Hub).

- El servidor de despliegue solo necesita ejecutar:

docker pull usuario/miapp:latest

docker stop miapp 88 docker rm miapp

docker run -d --name miapp -P 80:80 usuario/miapp:latest

Cada cambio en el código se convierte en una imagen confiable y puede estar disponible en producción en minutos.