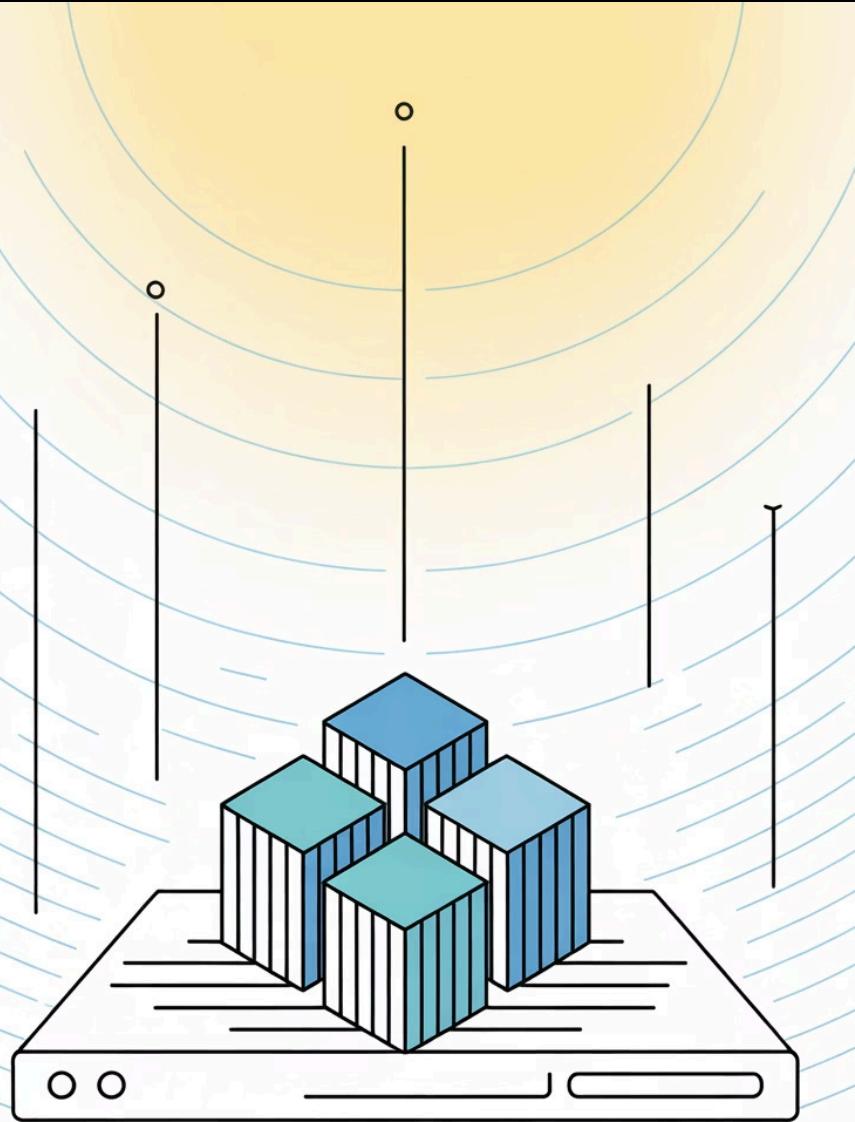


# Curso de Desarrollo Frontend con React



Docker

# Introducción a Docker para Desarrolladores Frontend

# Agenda de Hoy



## 1 Fundamentos de Docker

Conceptos esenciales y diferencias con VMs



## 2 Conceptos Clave

Imágenes, contenedores y Dockerfile



## 3 Docker en Frontend

Aplicación práctica en desarrollo web



## 4 Comandos Básicos

Herramientas esenciales del día a día



## 5 Ejemplo Práctico

Dockerfile para aplicación React



## 6 Buenas Prácticas

Optimización y eficiencia

# ¿Qué es Docker?

Docker es una plataforma que utiliza contenedores para empaquetar aplicaciones junto con todas sus dependencias, garantizando que funcionen de manera consistente en cualquier entorno.



# Docker en el Desarrollo Moderno

## Antes de Docker

- Conflictos de versiones
- Configuración manual repetitiva
- "En mi máquina funciona"
- Dependencias complejas



# La Solución Docker

## Consistencia

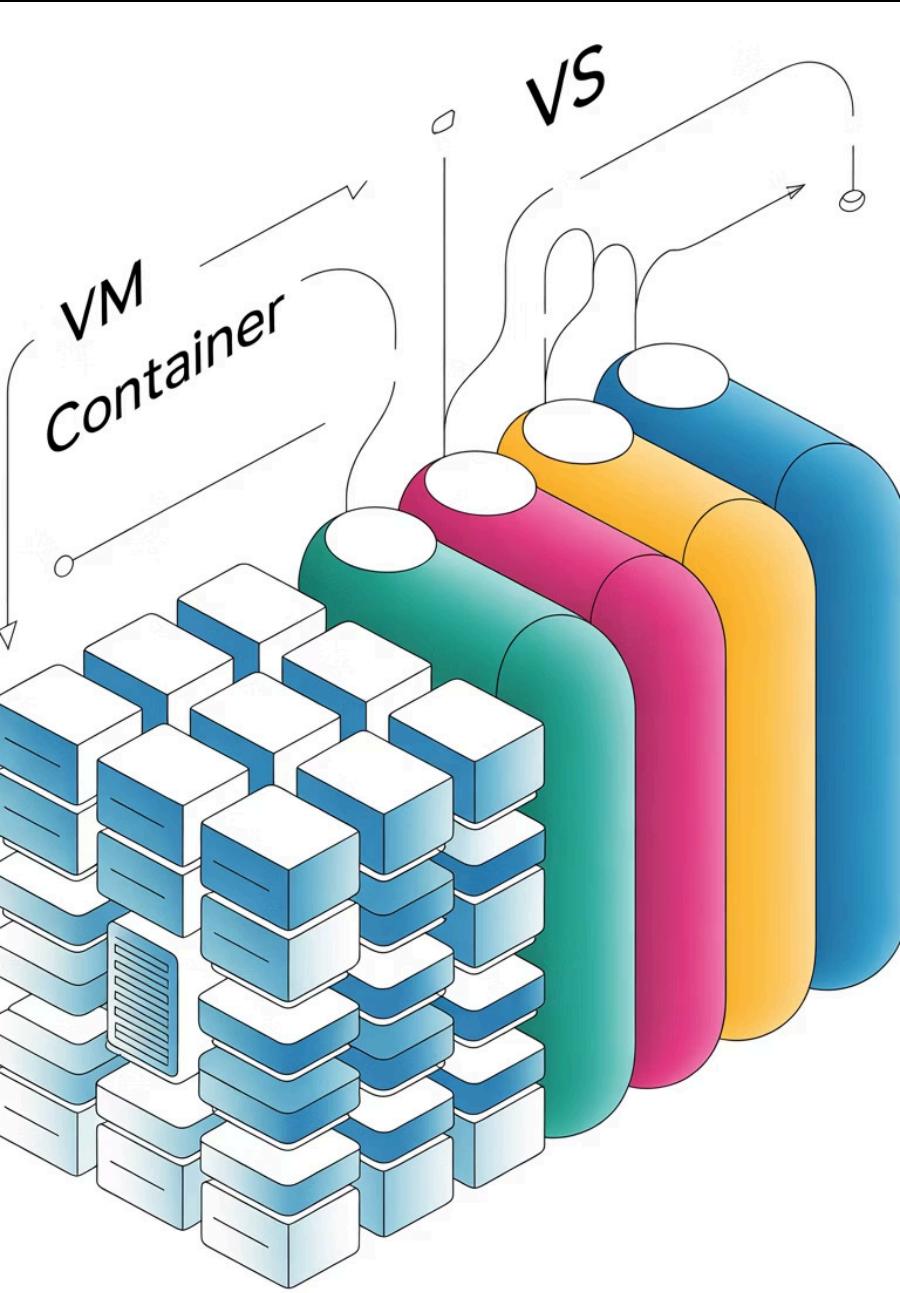
El mismo entorno en desarrollo, testing y producción

## Portabilidad

Funciona igual en cualquier sistema operativo

## Eficiencia

Menor consumo de recursos que las máquinas virtuales



# Máquinas Virtuales vs Contenedores

## Máquinas Virtuales

Virtualizan hardware completo, incluyendo sistema operativo.  
Más pesadas pero mayor aislamiento.

## Contenedores

Comparten el kernel del host.  
Más livianos, arranque rápido y uso eficiente de recursos.

# Ventajas de los Contenedores



## Arranque Rápido

Los contenedores inician en segundos, no minutos como las VMs



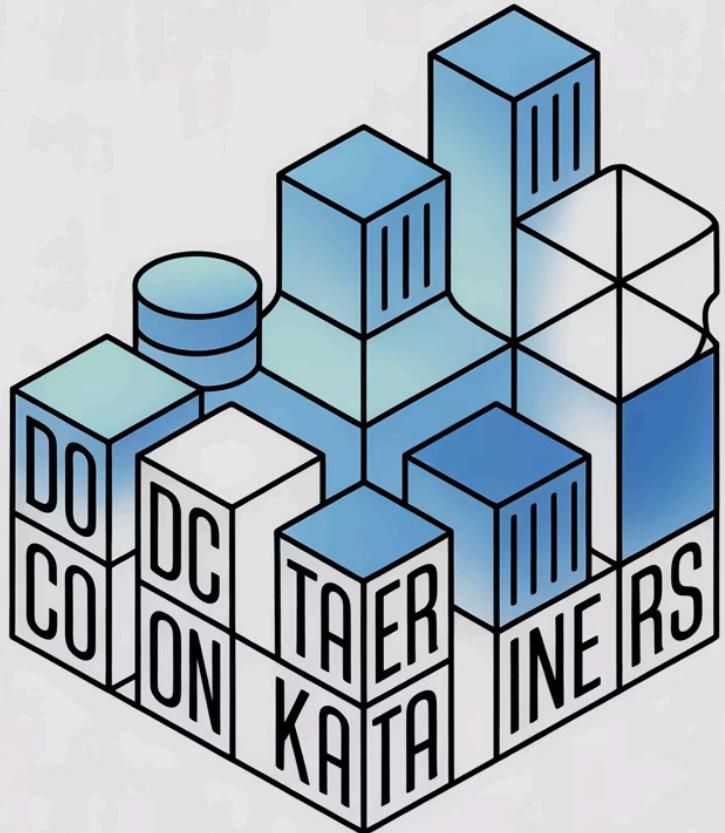
## Livianos

Ocupan mucho menos espacio en disco y memoria



## Escalables

Fácil creación de múltiples instancias según demanda



# Conceptos Fundamentales

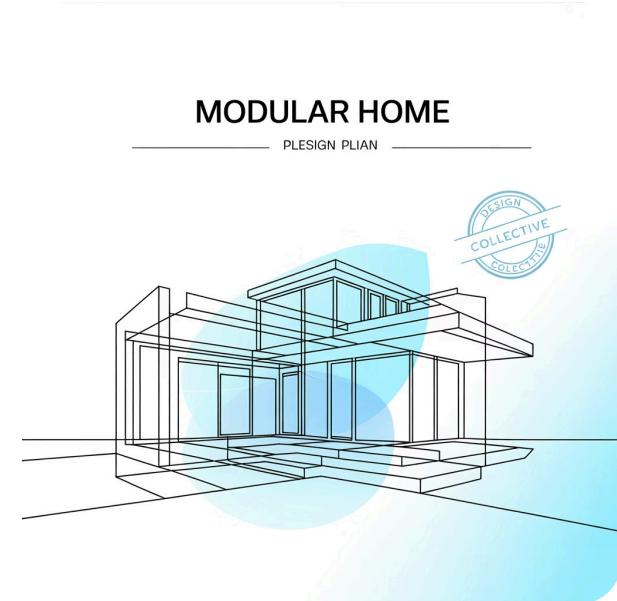
Antes de empezar a usar Docker, es crucial entender estos cuatro elementos básicos que forman el ecosistema.

# ¿Qué es una Imagen?

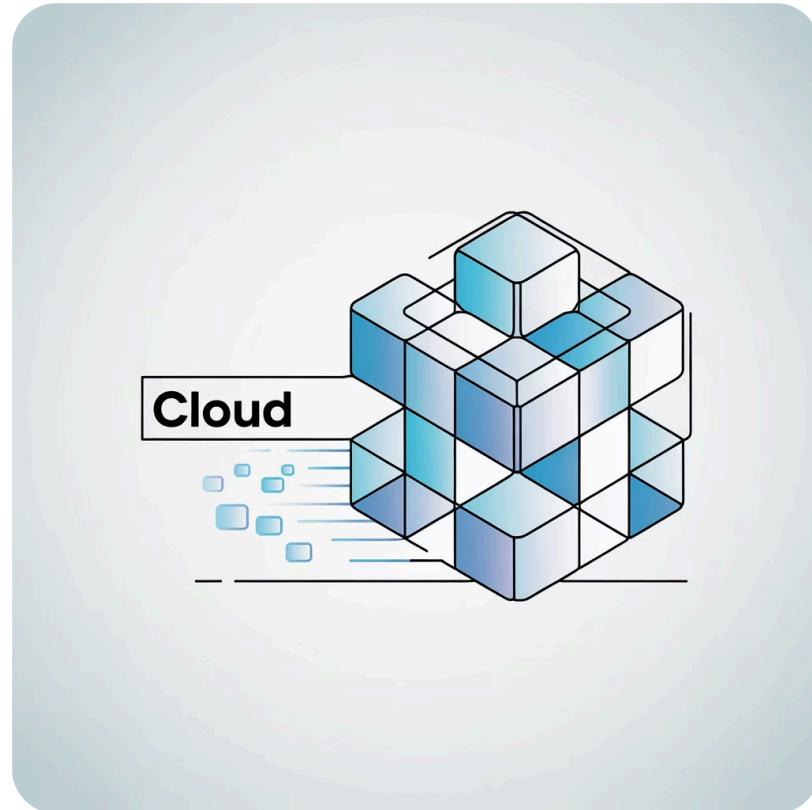
Una **imagen** es como una plantilla o molde que contiene:

- Sistema operativo base
- Aplicación y su código
- Dependencias y librerías
- Configuraciones necesarias

Es *inmutable* - no cambia una vez creada.



# ¿Qué es un Contenedor?



Un **contenedor** es una instancia en ejecución de una imagen:

- Entorno aislado y funcional
- Puede ejecutar procesos
- Tiene estado temporal
- Se puede iniciar, parar y eliminar

# Imagen vs Contenedor



## Imagen

Plantilla estática

Solo lectura

Se almacena una vez

## Contenedor

Instancia ejecutable

Puede escribir datos

Múltiples por imagen

# Dockerfile: Tu Receta de Imagen

Un **Dockerfile** es un archivo de texto con instrucciones paso a paso para construir una imagen Docker automáticamente.



# Estructura Básica del Dockerfile

```
FROM node:16-alpine
WORKDIR /app
COPY package*.json .
RUN npm install
COPY ..
EXPOSE 3000
CMD ["npm", "start"]
```

Cada línea es una instrucción que agrega una capa a la imagen final.

# Docker Hub: Tu Biblioteca

## ¿Qué es Docker Hub?

Es el repositorio oficial donde puedes:

- Descargar imágenes públicas
- Subir tus propias imágenes
- Colaborar con tu equipo
- Automatizar builds





# Docker en Frontend

# ¿Por Qué Docker en Frontend?



## Consistencia

Mismo Node.js, npm y dependencias para todo el equipo



## Portabilidad

Tu app funciona igual en cualquier máquina o servidor



## Despliegue Simple

Una sola imagen contiene todo lo necesario

# Problemas que Docker Soluciona

## Versiones de Node.js

"Mi proyecto necesita Node 14 pero tengo Node 18 instalado"

## Dependencias Globales

"¿Por qué mi compañero no puede ejecutar el proyecto?"

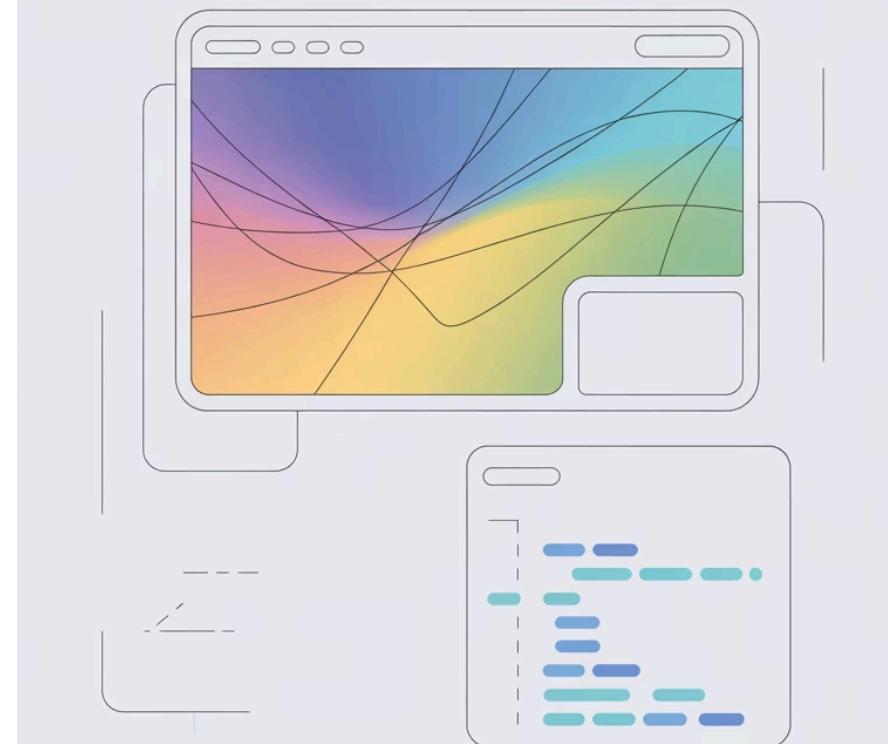
## Configuración Manual

"Pasé 2 horas configurando el entorno de desarrollo"

# Ejemplo: Aplicación React Containerizada

Veamos cómo Docker transforma el desarrollo de una app React típica, eliminando problemas comunes de configuración.

React Application  
Development Environment



# Flujo Tradicional vs Docker

## Flujo Tradicional

- Instalar Node.js localmente
- npm install en cada máquina
- Configurar variables de entorno
- Rezar que funcione en producción

## Flujo con Docker

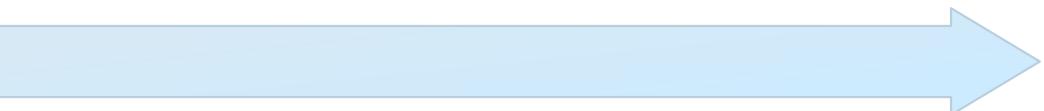
- Crear Dockerfile una vez
- docker build para crear imagen
- docker run en cualquier lugar
- Funciona idéntico siempre

# El Flujo Docker en Frontend



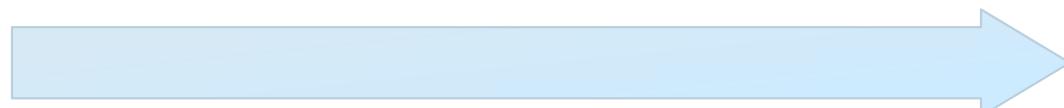
## Código Fuente

Tu aplicación React con package.json y dependencias



## Dockerfile

Instrucciones para construir el entorno perfecto



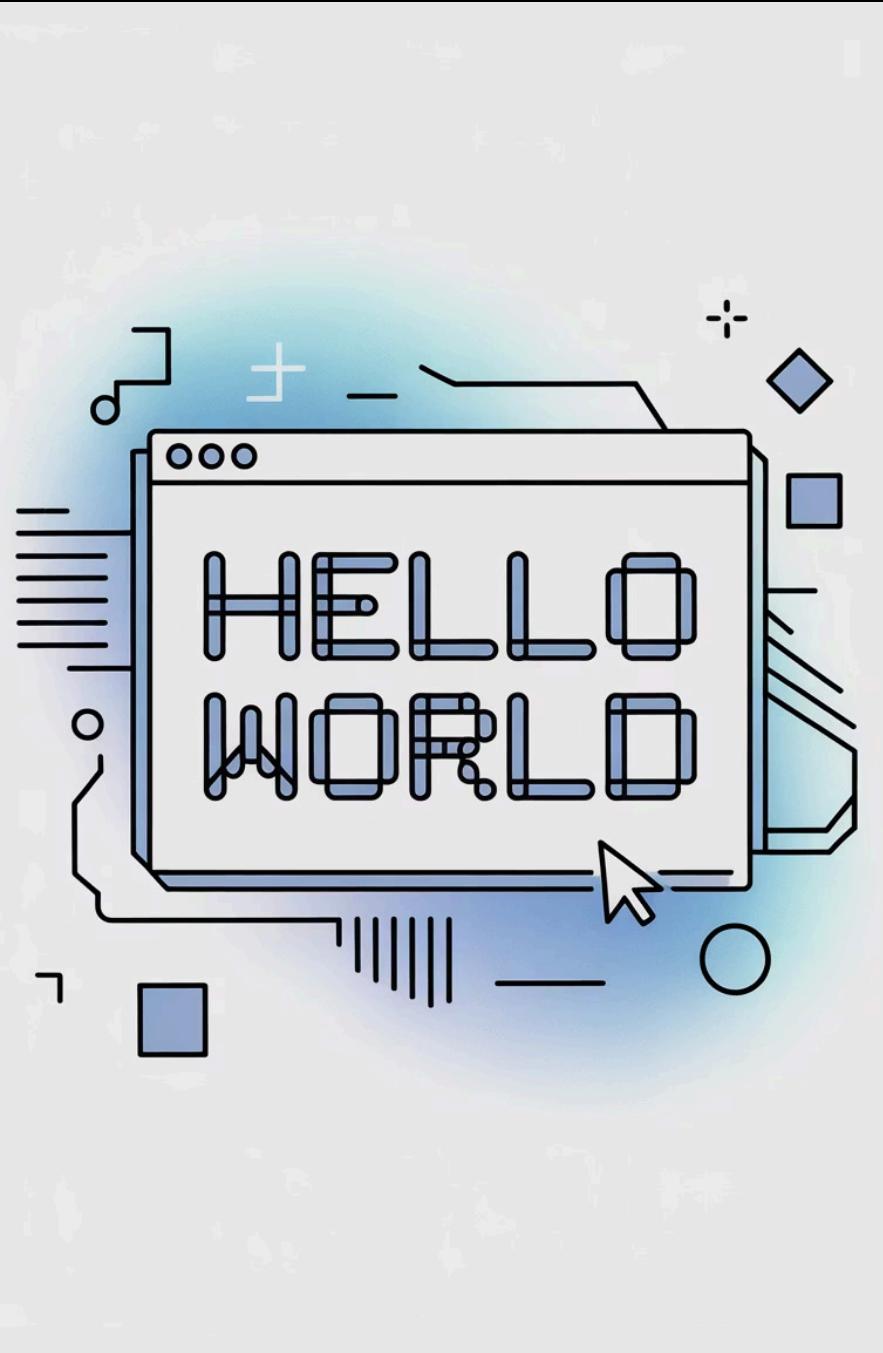
## Imagen Docker

Plantilla lista con tu app y todas las dependencias



## Contenedor

Tu aplicación ejecutándose de forma aislada



# Comandos Básicos

Domina estos comandos esenciales para trabajar eficientemente con Docker en tu día a día.

# docker build: Construyendo Imágenes

```
docker build -t mi-app-react .
```

Este comando:

- `-t` asigna un nombre a la imagen
- `.` usa el directorio actual
- Lee el Dockerfile automáticamente



# **docker run: Ejecutando Contenedores**

```
docker run -p 3000:3000 mi-app-react
```



**-p 3000:3000**

Mapea el puerto 3000 del contenedor al puerto 3000 de tu máquina



**mi-app-react**

Nombre de la imagen que quieres ejecutar

# **docker ps: ¿Qué Está Corriendo?**

`docker ps`

Muestra todos los contenedores activos con información útil:

- ID del contenedor
- Imagen utilizada
- Puertos expuestos
- Tiempo de ejecución

# Gestión de Contenedores

## **docker stop**

```
docker stop [ID]
```

Detiene un contenedor en ejecución de forma elegante

## **docker rm**

```
docker rm [ID]
```

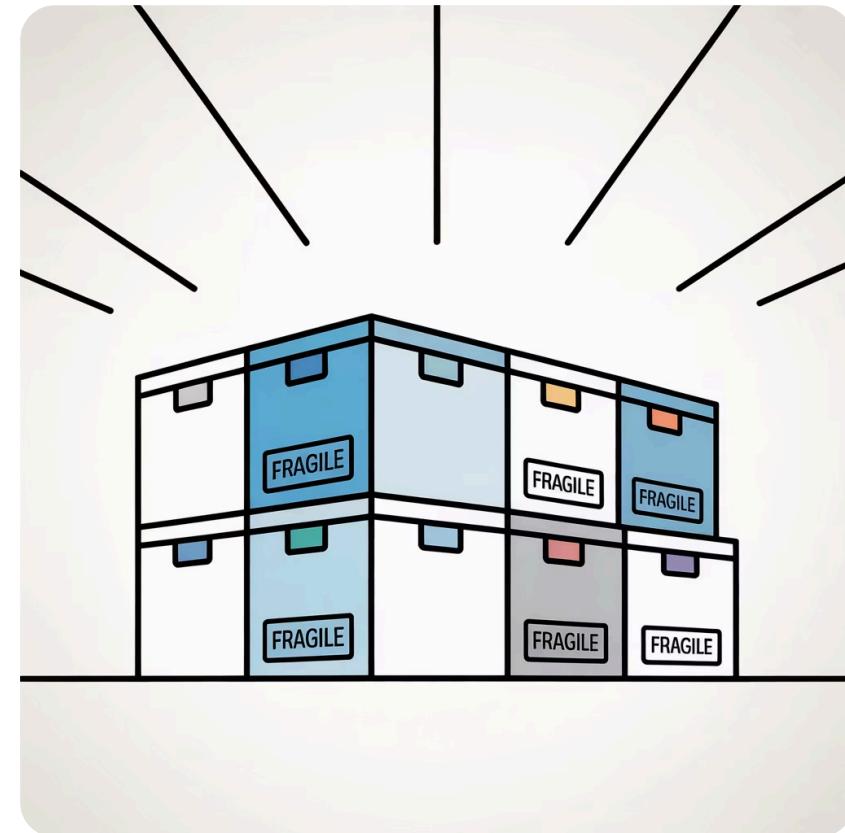
Elimina completamente un contenedor detenido

# docker images: Tu inventario

docker images

Lista todas las imágenes almacenadas localmente mostrando:

- Nombre y tag
- ID único
- Fecha de creación
- Tamaño en disco



# Dockerfile para React

Vamos a crear paso a paso un Dockerfile optimizado para una aplicación React moderna.



# Estructura del Proyecto React

```
mi-app-react/
├── package.json
├── package-lock.json
├── public/
├── src/
└── Dockerfile
    └── .dockerignore
```

Organización típica de un proyecto React que vamos a containerizar.

# Dockerfile Paso a Paso

```
FROM node:16-alpine
```

```
WORKDIR /app
```

```
COPY package*.json ./
```

```
RUN npm install
```

```
COPY ..
```

```
EXPOSE 3000
```

```
CMD ["npm", "start"]
```

# Explicando Cada Instrucción

01

## **FROM node:16-alpine**

Imagen base ligera con Node.js 16

03

## **COPY package\*.json ./**

Copia archivos de dependencias primero

05

## **COPY ..**

Copia el resto del código fuente

02

## **WORKDIR /app**

Establece el directorio de trabajo

04

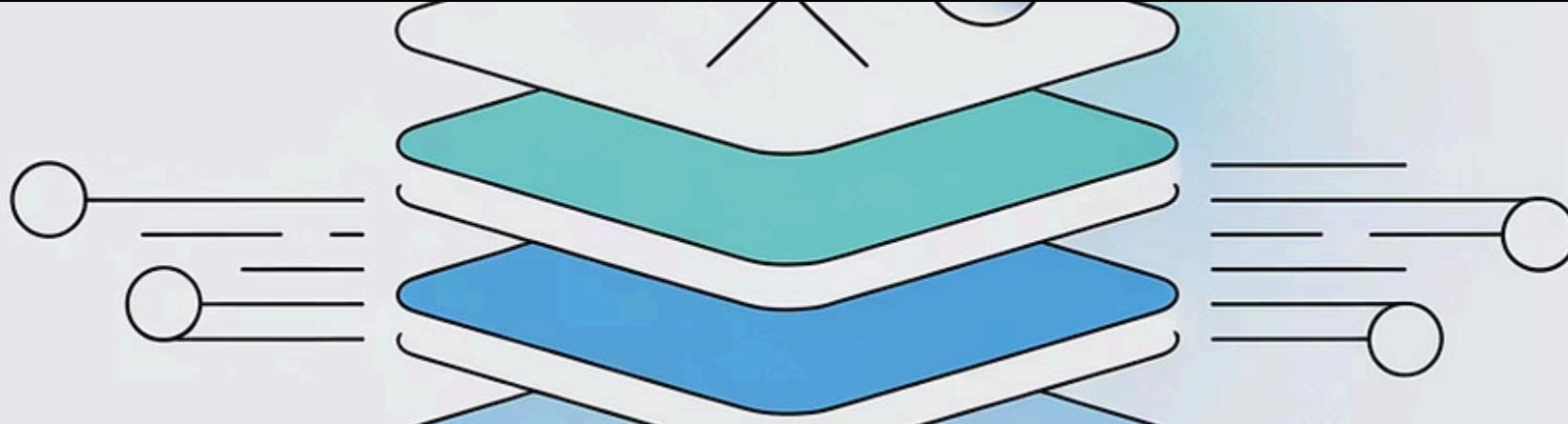
## **RUN npm install**

Instala dependencias (se cachea)

06

## **CMD ["npm", "start"]**

Comando que ejecuta al iniciar



## ¿Por Qué Este Orden?

Copiar `package.json` primero aprovecha el **cache de capas** de Docker. Si solo cambias código, no reinstala dependencias.

# Construir y Ejecutar

1

**Construir**

```
docker build -t react-app .
```

Crea la imagen con todas las dependencias

2

**Ejecutar**

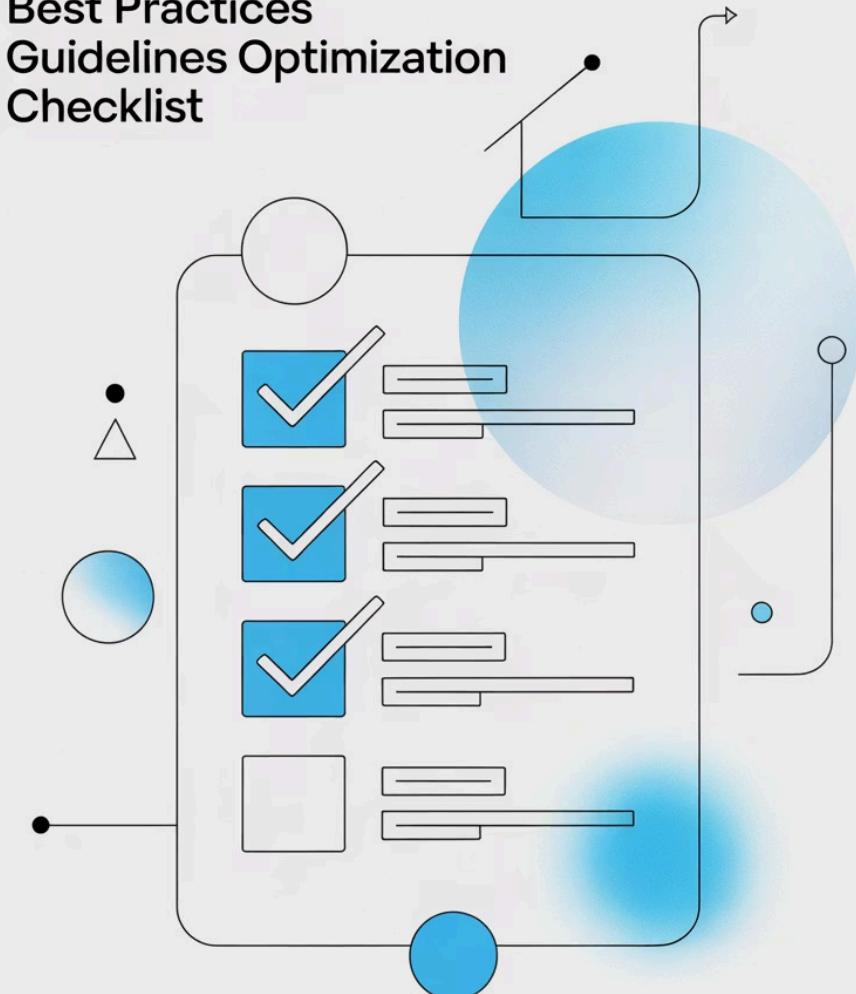
```
docker run -p 3000:3000 react-app
```

Inicia el contenedor y mapea puertos

# Buenas Prácticas

Optimiza tus contenedores siguiendo estas recomendaciones probadas por la comunidad.

## Best Practices Guidelines Optimization Checklist



# Usa Imágenes Base Ligeras

 **Pesado**

FROM node:16

~900MB - Incluye herramientas innecesarias

Alpine Linux es una distribución minimalista perfecta para contenedores.

 **Optimizado**

FROM node:16-alpine

~170MB - Solo lo esencial

# Archivo .dockerignore

```
node_modules  
npm-debug.log  
.git  
.gitignore  
README.md  
.env  
coverage  
.nyc_output
```

Evita copiar archivos innecesarios que aumentan el tamaño de la imagen y tiempo de build.

# Separar Desarrollo y Producción

## Dockerfile.dev

Hot reload, debugger, herramientas de desarrollo

## Dockerfile.prod

Build optimizado, servidor nginx, tamaño mínimo

# Ejemplo Dockerfile de Producción

```
# Build stage
FROM node:16-alpine as builder
WORKDIR /app
COPY package*.json .
RUN npm ci --only=production

# Production stage
FROM nginx:alpine
COPY --from=builder /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

# Ventajas de la Containerización



## Colaboración

Todo el equipo trabaja en el mismo entorno exacto, eliminando el "en mi máquina funciona"



## CI/CD Simple

Integración y despliegue continuo más confiable con entornos consistentes



## Escalabilidad

Fácil replicar contenedores para manejar mayor carga de usuarios

# Alternativas a Docker

Ante los cambios en el modelo de precios de Docker y la creciente necesidad de flexibilidad, diversas herramientas open source han ganado terreno como opciones robustas para la containerización.



## Podman

Una alternativa sin daemon que permite ejecutar contenedores sin privilegios de root, compatible con OCI y con una CLI similar a Docker.



## Containerd

Un runtime de contenedores de nivel industrial, utilizado por Docker. Ofrece una base sólida para gestionar el ciclo de vida de los contenedores.



## LXC/LXD

Tecnologías de virtualización a nivel de sistema operativo que ofrecen contenedores muy ligeros y eficientes, ideales para entornos Linux.

## Buildah & Skopeo

Herramientas modulares para construir (Buildah) y gestionar (Skopeo) imágenes de contenedores OCI, sin depender de un demonio.

# Cuándo Elegir Qué Herramienta

La elección de tu herramienta de containerización dependerá de tus necesidades específicas, el entorno de desarrollo y despliegue, y las prioridades de tu equipo.



## Startups y Equipos Pequeños

Prioriza la velocidad de desarrollo y la facilidad de uso. Docker Desktop y su amplio ecosistema son ideales para una adopción rápida y mínima curva de aprendizaje.



## Grandes Corporaciones

Busca control granular, seguridad y optimización de costos. Aunque Docker es común, explorar alternativas como Containerd o Podman puede ofrecer mayor flexibilidad y eficiencia a escala.



## Proyectos Open Source

Alinea con principios de software libre y evita licencias restrictivas. Podman y Buildah son excelentes opciones sin daemon, ofreciendo compatibilidad con la sintaxis de Docker.



## Entornos de Seguridad Crítica

Requiere contenedores sin privilegios de root (rootless) y una superficie de ataque mínima. Podman es superior en este aspecto, junto con Containerd para gestión de bajo nivel.



## Desarrollo Personal y Aprendizaje

Benefíciate de la comunidad masiva y la vasta documentación. Docker sigue siendo la puerta de entrada más accesible al mundo de la containerización para principiantes.



## Reflexión Final

¿Qué ventajas ofrece contenerizar una aplicación frontend en comparación con ejecutarla directamente en el entorno local de cada desarrollador?

Piensa en tu experiencia actual: ¿cuánto tiempo has perdido por problemas de configuración?

# #EDCOUNIANDES

<https://educacioncontinua.uniandes.edu.co/>

Contacto: [educacion.continua@uniandes.edu.co](mailto:educacion.continua@uniandes.edu.co)

© - Derechos Reservados: La presente obra, y en general todos sus contenidos, se encuentran protegidos por las normas internacionales y nacionales vigentes sobre propiedad Intelectual, por lo tanto su utilización parcial o total, reproducción, comunicación pública, transformación, distribución, alquiler, préstamo público e importación, total o parcial, en todo o en parte, en formato impreso o digital y en cualquier formato conocido o por conocer, se encuentran prohibidos, y solo serán lícitos en la medida en que se cuente con la autorización previa y expresa por escrito de la Universidad de los Andes.