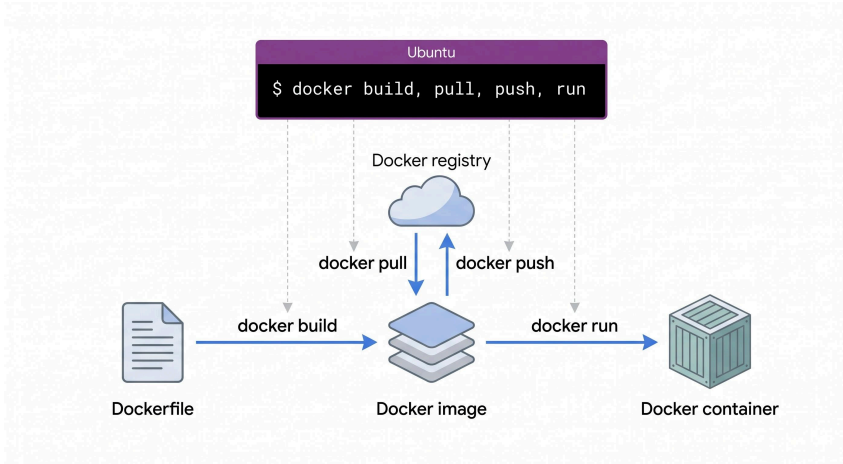


Guía Completa de Docker

<https://gemini.google.com/app/7e0b7dd86bac7519>

User prompt: dime todo lo que sabes sobre docker

Response: Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones mediante tecnología de **containerización**. A diferencia de las máquinas virtuales tradicionales, Docker aísla procesos a nivel del kernel del sistema operativo anfitrión, garantizando que el entorno de ejecución sea idéntico desde el desarrollo hasta la producción.



Conceptos Fundamentales

Componente	Descripción
Dockerfile	Script con las instrucciones necesarias para compilar una imagen paso a paso.
Imagen	Plantilla ejecutable de solo lectura que empaqueta el código, binarios, librerías y dependencias.
Contenedor	Instancia de ejecución de una imagen aislada del sistema. Añade una capa de lectura/escritura sobre la imagen.
Docker Registry	Almacén distribuido para publicar y descargar imágenes (ej. Docker Hub, Amazon ECR).
Docker Engine	Aplicación cliente-servidor con un daemon (dockerd), una API REST y la CLI para interactuar con la plataforma.

Contenedores vs. Máquinas Virtuales

- **Máquinas Virtuales (VMs):** Virtualizan el hardware. Cada VM incluye un sistema operativo invitado (*Guest OS*) completo, lo que genera un consumo elevado de disco/RAM y tiempos de arranque de minutos.
- **Contenedores:** Virtualizan el sistema operativo. Comparten el **kernel del OS anfitrión** aprovechando primitivas del kernel de Linux (*namespaces* para aislamiento de procesos y *cgroups* para asignación de CPU/memoria), lo que permite arrancar en milisegundos con una huella mínima.

Ecosistema Clave

- **Docker Compose:** Permite definir y orquestar aplicaciones multicontenedor mediante un archivo declarativo `docker-compose.yml`.

- **Persistencia (Volúmenes):** Almacenamiento independiente del ciclo de vida del contenedor para evitar la pérdida de datos al destruirlo.
- **Redes (Networking):** Controla el aislamiento y la comunicación entre contenedores o con redes externas (modos bridge, host, overlay).
- **Orquestación:** Para entornos complejos a gran escala, herramientas como **Kubernetes** o **Docker Swarm** gestionan el escalado automático, balanceo de carga y autorecuperación.

Comandos Fundamentales

```
# Construir una imagen a partir del Dockerfile del directorio actual
docker build -t mi-aplicacion:v1 .

# Descargar una imagen oficial desde Docker Hub
docker pull python:3.11-slim

# Ejecutar un contenedor mapeando puertos (Host:Contenedor) en segundo plano (detached)
docker run -d -p 8080:80 --name servidor_web mi-aplicacion:v1

# Listar contenedores activos
docker ps

# Acceder a una terminal interactiva dentro de un contenedor en ejecución
docker exec -it servidor_web /bin/bash

# Limpieza de recursos no utilizados (imágenes huérfanas, contenedores detenidos)
docker system prune
```

User prompt: dame mas

Response: Profundizando en el uso profesional de Docker, estos son los patrones avanzados de optimización, seguridad, orquestación y diagnóstico esenciales para entornos de producción.

Optimización de Imágenes con Multi-Stage Builds

Los *Multi-stage builds* dividen la compilación en varias etapas para mantener las imágenes de producción extremadamente ligeras y seguras, separando las herramientas de compilación del artefacto final.

```
# Etapa 1: Build (incluye herramientas de compilación y SDKs)
FROM golang:1.22-alpine AS builder
WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o mi-app .
```

```
# Etapa 2: Runtime (mínima y reducida)
FROM alpine:3.19
RUN adduser -D appuser
WORKDIR /app
COPY --from=builder /app/mi-app .
USER appuser
EXPOSE 8080
CMD ["/mi-app"]
```

Buenas Prácticas de Seguridad y Desempeño

- **Principio de menor privilegio:** Nunca ejecutes procesos como root dentro del contenedor. Declara un usuario sin privilegios (USER appuser).
- **Uso de .dockerignore:** Excluye .git, node_modules, archivos .env y temporales para evitar que se incluyan en el contexto de compilación o se filtren credenciales.
- **Aprovechamiento del Layer Caching:** Ordena las instrucciones del Dockerfile desde las que menos cambian a las que más cambian (copia dependencias/paquetes *antes* de copiar el código fuente).
- **Imágenes base mínimas:** Prefiere imágenes alpine o distroless sobre distribuciones Linux completas para reducir la superficie de vulnerabilidad (CVEs).

Orquestación Local con Docker Compose y Healthchecks

Permite gestionar múltiples servicios dependientes coordinando su orden de inicio según la disponibilidad real del servicio y no solo del estado del contenedor.

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8080:8080"
    environment:
      - DB_HOST=db
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres:16-alpine
    environment:
      POSTGRES_DB: app_db
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: secretpassword
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U admin -d app_db"]
      interval: 5s
      timeout: 5s
      retries: 5

volumes:
  postgres_data:
```

Monitoreo y Inspección Avanzada

Comando	Función Principal
docker stats	Telemetría en tiempo real de CPU, memoria, E/S de red y disco por contenedor.
docker inspect <id>	Devuelve la configuración JSON completa del objeto (redes, montajes, variables).
docker logs -f --tail 100 <id>	Transmite las últimas 100 líneas del log del contenedor en tiempo real.

Comando	Función Principal
<code>docker diff <id></code>	Muestra las alteraciones realizadas en el sistema de archivos del contenedor respecto a su imagen base.
<code>docker run --memory="512m" - -cpus="1.5"</code>	Limita de forma estricta los recursos consumibles por el contenedor.
