INTRODUCCIÓN A DOCKER

Contenedores para el desarrollo moderno

Duración: 4 horas | Teoría + Práctica

AGENDA DEL CURSO

1	**Introducción a Docker** Qué es, arquitectura y conceptos clave	*30 min*
2	**Imágenes y Contenedores** Ciclo de vida y comandos básicos	*30 min*
3	**Laboratorio 1: Primeros pasos** Ejecutar y gestionar contenedores	*45 min*
4	**Dockerfile y construcción** Crear imágenes personalizadas	*30 min*
5	**Laboratorio 2: Tu primera imagen** Construir aplicación containerizada	*45 min*
6	**Docker Compose** Aplicaciones multi-contenedor	*30 min*
7	**Laboratorio 3: App completa** Base de datos + Backend + Frontend	*60 min*

¿QUÉ ES DOCKER?

Docker es una plataforma de **contenedores** que permite empaquetar aplicaciones con todas sus dependencias en unidades estandarizadas llamadas **contenedores**.

Portabilidad

Funciona igual en desarrollo, testing y producción

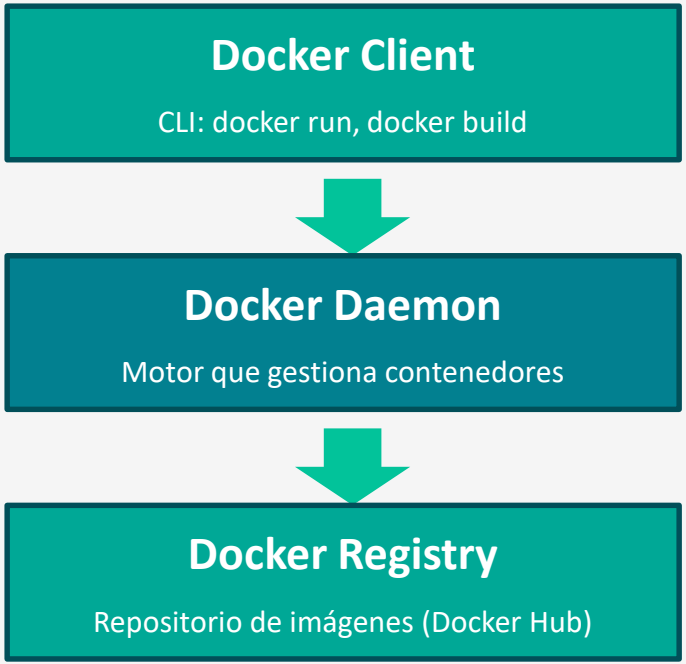
Ligereza

Comparte el kernel del SO, no necesita virtualización completa

Consistencia

Misma imagen = mismo comportamiento en cualquier entorno

ARQUITECTURA DE DOCKER



FLUJO DE TRABAJO

1. 1. Escribes un Dockerfile
1. 2. Construyes una imagen
1. 3. Subes la imagen al registro
1. 4. Ejecutas contenedores desde la imagen
1. 5. Los contenedores se ejecutan aislados

CONCEPTOS CLAVE

Imagen	Plantilla de solo lectura con instrucciones para crear un contenedor. Incluye código, runtime, librerías y dependencias.	*ubuntu:20.04, nginx:latest, node:18*
Contenedor	Instancia ejecutable de una imagen. Proceso aislado que ejecuta tu aplicación.	*docker run -d nginx*
Dockerfile	Archivo de texto con instrucciones para construir una imagen Docker.	*FROM, RUN, COPY, CMD*
Volume	Almacenamiento persistente para datos que sobreviven al ciclo de vida del contenedor.	*-v /data:/app/data*

IMAGEN vs CONTENEDOR

IMAGEN

- Plantilla inmutable
- Se construye una vez
- Se almacena en registro
- Compuesta por capas
- Similar a una clase en POO

CONTENEDOR

- Instancia ejecutable
- Se puede crear/detener/eliminar
- Tiene estado en runtime
- Añade capa escribible
- Similar a un objeto en POO

Analogía: La imagen es la receta, el contenedor es el plato preparado

COMANDOS BÁSICOS

`docker pull <imagen>`	Descargar imagen desde Docker Hub
`docker images`	Listar imágenes locales
`docker run <imagen>`	Crear y ejecutar un contenedor
`docker ps`	Listar contenedores en ejecución
`docker ps -a`	Listar todos los contenedores
`docker stop <id>`	Detener un contenedor
`docker rm <id>`	Eliminar un contenedor
`docker rmi <imagen>`	Eliminar una imagen
`docker logs <id>`	Ver logs de un contenedor
`docker exec -it <id> bash`	Ejecutar comando en contenedor

DOCKERFILE

Archivo que define cómo construir tu imagen

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

FROM	Imagen base
WORKDIR	Directorio de trabajo
COPY	Copiar archivos al contenedor
RUN	Ejecutar comandos al construir
EXPOSE	Documentar puerto expuesto
CMD	Comando por defecto al iniciar

DOCKER COMPOSE

Herramienta para definir y ejecutar aplicaciones multi-contenedor

```yaml
version: '3.8'
services:
  web:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - db
  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: secret
```

BENEFICIOS

- ✓ Define toda tu aplicación en un archivo
- ✓ Levanta múltiples servicios con un comando
- ✓ Gestiona redes y volúmenes automáticamente
- ✓ Ideal para desarrollo y testing

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

VENTAJAS DE DOCKER

Despliegue rápido

Desde commit hasta producción en minutos

Aislamiento

Cada contenedor ejecuta en su propio espacio

Eficiencia

Mejor uso de recursos vs VMs

DevOps

Integración perfecta con CI/CD

Microservicios

Arquitectura modular y escalable

Portabilidad

Funciona en cualquier infraestructura

CASOS DE USO

1

Desarrollo Local

Replicar el entorno de producción en tu laptop. No más 'en mi máquina funciona'.

```
docker run -d -p
5432:5432 postgres
```

2

CI/CD

Ejecutar tests en contenedores aislados. Construir, testear y desplegar automáticamente.

Jenkins + Docker = Pipeline potente

3

Microservicios

Cada servicio en su propio contenedor. Escala independientemente según demanda.

Auth, API, Workers, cada uno en Docker

4

Aplicaciones Legacy

Containeriza apps antiguas sin modificarlas. Extiende su vida útil.

PHP 5.6 en producción 2025? ✓

LABORATORIOS PRÁCTICOS

Lab 1: Primeros Pasos

Ejecutar contenedores, gestión básica | 45 min

Lab 2: Tu Primera Imagen

Crear Dockerfile, construir imagen | 45 min

Lab 3: App Multi-Contenedor

Docker Compose, base de datos + app | 60 min

MEJORES PRÁCTICAS

Imágenes pequeñas

Usa imágenes alpine cuando sea posible

Multi-stage builds

Separa build de runtime para reducir tamaño

No secrets en imágenes

Usa variables de entorno o Docker secrets

.dockerignore

Excluye node_modules, .git, etc del contexto

Tags específicos

Evita :latest en producción, usa :1.2.3

Limpieza de capas

RUN apt-get update && apt-get install && rm -rf /var/lib/apt

Usuario no-root

USER node para ejecutar como usuario sin privilegios

Health checks

HEALTHCHECK para monitoreo de contenedores

RECURSOS Y PRÓXIMOS PASOS

📚 DOCUMENTACIÓN

- docs.docker.com - Documentación oficial
- hub.docker.com - Repositorio de imágenes
- play-with-docker.com - Playground online
- docker.com/blog - Blog oficial

🚀 TEMAS AVANZADOS

- Kubernetes y orquestación
- Docker Swarm
- Seguridad en contenedores
- Optimización de imágenes
- Monitoreo y logging
- Registro privado (Harbor)

🏆 *Certificación Docker (DCA)*

¡MANOS A LA OBRA!

Es hora de practicar con los laboratorios

Recuerda: La mejor forma de aprender Docker es usándolo