

Nombre Esteban Palomar Muñoz Fecha 24/08/25

Profesor

Institución

Nota

1) Concepto de Docker e importancia en el desarrollo moderno

Docker es una plataforma para crear, empaquetar y ejecutar aplicaciones dentro de contenedores. Un contenedor es un proceso del sistema operativo aislado mediante namespaces y limitado por cgroups, que incluye el código, dependencias, binarios y configuraciones necesarios para ejecutar una aplicación de forma consistente.

A diferencia de una máquina virtual, el contenedor comparte el kernel de host y solo agrega lo imprescindible para la app, por lo que inicia rápido, consume menos recursos y se transporta fácilmente entre máquinas y nubes. Docker se ha convertido en un estándar de facto en prácticas DevOps y cloud-native.

Historia breve: el aislamiento de procesos en Linux existe hace años (chroot, luego LXC, cgroups, namespaces). Docker popularizó un flujo de trabajo sencillo (construir, distribuir, ejecutar imágenes) y un ecosistema de herramientas (CLI, Engine, compose, BuildKit).

2) Arquitectura y funcionalidades

Componentes principales:

- Docker Engine (dockerd): daemon que gestiona imágenes, contenedores, redes, volúmenes.
- CLI (docker): interfaz de línea de comandos que se comunica con daemon vía API.
- Runtime (containerd | runc): ejecuta contenedores siguiendo el estándar OCI.
- BuildKit: motor de construcción moderno (cache eficiente, secretos, mounts, multi-stage).
- Registros (Docker Hub, GitHub, privados): distribución de

Imagenes (pull/push).

Storage drivers (overlay2, btrfs, zfs) administran las capas del sistema de archivos; loggin drivers definen el destino de logs; Network drivers (bridge, host, macvlan) controlan tráfico.

Funcionalidades clave

- Empaquejado en imágenes inmutables por capas (reproducibilidad)
- Aislamiento de procesos (namespaces) y control de recursos (cgroups).
- Portabilidad y distribución mediante registros de imágenes.
- Redes definidas por software (bridge, host, none, macvlan); DNS interno entre servicios
- Persistencia con volúmenes y bind mounts
- Orquestación local y multi-contenedor con Docker Compose.

3) Beneficios de usar Docker

- Arranque en segundos frente a minutos de una VM.
- Consistencia entre desarrollo, pruebas y producción (misma imagen).
- Menor consumo de recursos al compartir kernel del host.
- Escalabilidad horizontal rápida (más replicas del mismo servicio).
- Productividad: levanta bases de datos o colas efimeras para tests en segundos.
- Ecosistema maduro: CI/CD, nubes, observabilidad, seguridad, scanning
- Trazabilidad y rollback mediante versionado de imágenes

4) Posibles problemas o limitaciones

- Curva de aprendizaje (imágenes, redes, volúmenes, seguridad).

- En macOS Windows corre dentro de una VM (Hyper-V / WSL2), menor IO con bind mounts.
- Persistencia: Si no usas volúmenes, al borrar contenidos pierdes datos.
- Seguridad: Imágenes no confiables o ejecución como root incrementan riesgos.
- Compatibilidad: Contenedores Linux vs Windows son distintos; no mezclables.
- Debug de redes y puertos puede requerir práctica (NAT, firewalls, DNS).

5) Guía de instalación (macOS, windows)

MacOS: Instala Docker Desktop (Intel o Apple Silicon), abre la app y valida:
docker --version

docker run hello-world

Windows (WSL2 recomendado): activa WSL2 y Virtual Machine Platform, instala Docker Desktop y valida:
docker --version

docker run hello-world

6) Conceptos clave (profundizado)

Imágenes

Plantillas inmutables compuestas por capas. Cada instrucción del Dockerfile (RUN, COPY, etc.) añade una capa. Las imágenes tienen tags y pueden ser multiarquitectura (amd64/arm64) usando buildx. Evita usar 'latest' en producción; versiona claramente.

Contenedores

Instancias en ejecución de una imagen. Estados típicos: created, running, stopped, removed. Los cambios dentro del contenedor no alteran la imagen; para cambios permanentes

Nombre

Fecha dia mes año

Profesor

Institución

Introducción a Docker

Curso 2012/2013

Volumenes y montajes

Volumenes (named) son gestionados por Docker y recomendados para datos persistentes. Bind mounts enlazan carpetas del host al contenedor, útiles en desarrollo, pero con cautela por permisos y rendimiento. TMPFS guarda datos en RAM (efímeros)

Redes

Bridge (por defecto), host (Linux; comparte red del host), none (sin red) y macvlan (IP propia en la LAN). Los contenedores en la misma red bridge se resuelven por nombre de servicio (DNS interno).

Dockerfile

Receta para construir imágenes. instrucciones clave: FROM, WORKDIR, COPY/ADD, RUN, ENV, EXPOSE (documenta puertos), USER (elevar root), HEALTHCHECK, ENTRYPOINT y CMD. Prefiere formato exec ([bin, -arg]).

Registros (Docker Hub y privados)

Distinguen imágenes (push/pull). repos públicos o privados, autenticación con 'docker login'. Alternativas: GHCR, Gitlab, ECR, ACR, GCR. Implementa políticas de retención y escaneo

7) Comandos de Docker (esenciales y avanzados)

Información y diagnóstico

docker version

- Versión cliente / servidor

docker info

- detalles del entorno, drivers y V

docker system df

- uso de disco por imágenes / cont.

docker system volumes

docker events

- Stream de eventos del daemon

Imagenes

docker search nginx

- buscar en Docker Hub

docker pull nginx:1.27

- descargar imagen específica

docker images

- listar imágenes locales

docker image ls --digests

- incluir digest sha256

docker history miapp:1.0

- capas y tamaños

docker tag miapp:1.0 repo/miapp:1.0

- agregar tag

docker push repo/miapp:1.0

- subir al registro

docker rmi nginx:1.27

- borrar imagen

docker image prune

- borrar imágenes no usadas

docker save -o img.tar nginx:1.27

- exportar imagen a tar

docker load -i img.tar

- importar imagen desde tar

Construcción de imágenes (Buildkit)

- build estándar

docker build -t miapp:dev

- sin cache

docker build --no-cache -t miapp:clean

- Dockerfile alternativo

docker build -f Dockerfile.prod -t miapp:prod

- Multi-arquitectura con buildx

docker buildx create --use

docker buildx build --platform linux/amd64,linux/arm64 -t

repo/miapp:1.0 -push

Contenedores: ciclo de vida

docker run --name web -d -p 8080:80 nginx - corre en background

docker ps - contenedores activos

docker ps -a - todos (incluidos detenidos)

docker stop web - detener

docker start web - iniciar

docker restart web - reiniciar

docker rm web - eliminar

docker logs -f web - Seguir logs en vivo

docker top web - procesos dentro del contenedor

docker inspect web - JSON con metadatos

docker diff web - cambios en el FS del contenedor

EXEC y copia de archivos

docker exec -it web bash - shell interactivo

docker cp web:/etc/nginx/nginx.conf - copiar al host

conf. /

docker cp ./index.html web:/usr/share/nginx/html/ - copiar al contenedor

nginx

Recursos y políticas

docker run --cpus=1 --memory=512m - limitar CPU/RAM

nginx

docker update --restart=always web - Políticas de reinicio: no on-failure[:N] 1a

Volumenes

docker volume create datos

docker volume ls

docker volume inspect datos

docker run -d --name db -v datos:/var/lib/postgresql/data

Postgres:16

docker volume rm datos

Bind mounts

-linux/macOS

```
docker run -it --rm -v $PWD/app:/app node:20 bash
```

-windows Powershell

```
docker run -it --rm -v $PWD/app:/app node:20 bash
```

Redes

docker network ls

```
docker network create appnet
```

```
docker run -d --name api --network=appnet m/api:1.0
```

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

```
docker network inspect appnet
```

```
docker network connect appnet web
```

```
docker network disconnect appnet web
```

```
docker network prune
```

EXPORT/IMPORT contenedores

```
docker export web -o web.tar
```

-exporta FS del contenedor
(sin capas)

```
docker import web.tar web:flat
```

-crea imágenes desde FS
planas

SISTEMA Y LIMPIEZA

```
docker system prune -m 10G
```

-limpia recursos no usados
(con confirmación)

```
docker system prune -a
```

-incluye imágenes no referen-
ciadas

```
docker buildx prune
```

-limpia cache de build

compose (plugin V2)

```
docker compose up -d
```

```
docker compose ps
```

```
docker compose logs -f
```

```
docker compose exec web sh
```

```
docker compose down
```

```
docker compose build
```

Aspecto	Docker	Maquinas Virtuales
Aislamiento	Nivel de proceso (namespaces/cgroups)	A nivel de hardware (hipervisor)
Kernel	Comparte kernel del host	Kernel propio por VM
Arranque	Segundos	Minutos
Consumo	Bajo (Sistema completo por instancia)	Alto (Sistema completo por VM)
Portabilidad	Muy alto (estándar OCI)	Alta, pero imágenes pesadas
Seguridad	Buena con buenas prácticas; menor que VM	Aislamiento fuerte por hipervisores
Compatibilidad SO	Linux > Linux/Windows < Windows	Casi cualquiera dentro de la VM
Uso ideal	Microservicios, CI/CD, Cloud-native	Apps legacy, kernels distintos, aislamiento