

Día	Mes	Año	Hora	Institución			
2							
Alumno					Código	Materia	
Curso	Bimestre	Semestre	Salón	Hoja No.	de		CALIFICACIÓN

Profesor

# DOCKER

Docker es una plataforma de software que permite crear, probar y desplegar aplicaciones rápidamente en entornos aislados llamados contenedores.

## ¿Qué es un contenedor?

Un contenedor es una unidad ligera y portátil que incluye todo lo necesario para ejecutar una aplicación, como el código, las bibliotecas y las dependencias, haciendo la consistente y fácil de mover entre diferentes entornos.

Docker facilita la creación, distribución y ejecución de aplicaciones en entornos consistentes y aislados. Esto mejora la eficiencia del desarrollo y la implementación, ya que las aplicaciones empacadas en contenedores pueden ejecutarse de manera consistente en cualquier entorno que admita Docker.

→ Crear contenedores de docker no se trata de crear una máquina virtual, que es algo que comúnmente tendremos a malentender, pero que definitivamente nos puede dar la formación para llegar a entender que se trata de un contenedor, un lugar en donde se está empaquetando toda la solución guardandola y dejándola lista para que cualquier otra persona pueda descargar sin problema.

## Principales Características de Docker

### 1. Contenedores:

\***Aislamiento:** los contenedores encapsulan una aplicación y su entorno, asegurando que las aplicaciones se ejecuten de manera independiente y sin interferencias entre sí.

\* **Ligereza:** A diferencia de las máquinas virtuales, los contenedores comparten el mismo sistema operativo subyacente, lo que los hace más eficientes en términos de recursos.

## 2. Portabilidad:

Los contenedores pueden ejecutarse en cualquier sistema que soporte Docker, garantizando que el software funcione de la misma manera en desarrollo, pruebas y producción.

## 3. Imagen de Docker:

Una imagen es una plantilla de solo lectura utilizada para crear contenedores. Incluye todo lo necesario para ejecutar una aplicación: código, dependencias, bibliotecas, configuraciones, etc.

Las imágenes se pueden almacenar y compartir a través de registros de imágenes, como Docker Hub.

## 4. Orquestación:

- Herramientas como Docker Compose permiten definir y ejecutar aplicaciones multi-contenedor.
- Kubernetes es una plataforma de orquestación más avanzada que gestiona la implementación, escalado y operación de aplicaciones contenedora.

## Beneficios de Docker

### 1. Consistencia:

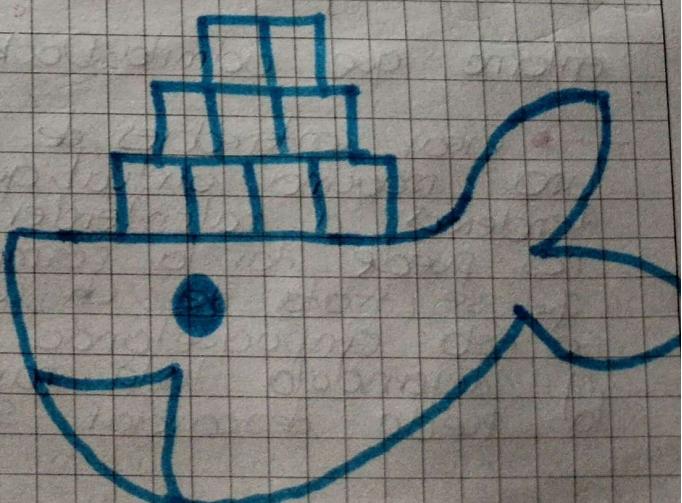
Garantiza que las aplicaciones se ejecuten de manera uniforme en diferentes entornos, eliminando problemas de "funciona en mi máquina".

### 2. Escalabilidad:

Facilita la escalabilidad horizontal al permitir la creación rápida de múltiples instancias de contenedores.

### 3. Desarrollo Ágil:

Permite a los desarrolladores crear y destruir entornos de desarrollo rápidamente, facilitando pruebas y desarrollo continuo.



# Diferencias entre máquinas virtuales y contenedores

## Docker

Una máquina virtual es un software que simula un computador completo dentro de otro computador. Tiene su propio sistema operativo invitado separado del sistema operativo del host, incluye su propia CPU virtual, disco duro y red virtual, dicho esto; es como tener una computadora dentro de otra computadora completamente aislada.

La mayor diferencia entre una máquina virtual y un contenedor radica en el aislamiento que tienen de sus entornos de trabajo:

Los VMs proporcionan un aislamiento completo, ya que ejecutan un sistema operativo completo independiente del host, consumen más recursos porque cada una tiene su propio sistema operativo; por ende presentan un arranque más lento.

Los contenedores presentan un aislamiento ligero, ya que comparten el mismo kernel del sistema operativo, son más eficientes en términos de recursos, ya que comparten el mismo sistema operativo y solo incluye las bibliotecas y dependencias necesarias.

## Funcionamiento de los contenedores a nivel de sistema operativo

→ Procesos apilados que comparten el kernel del sistema operativo original. Para lograr este comportamiento, Docker utiliza varias características del kernel de Linux:

1. **Namepaces**: Proporcionan aislamiento de recursos (PID, red, memoria, etc). Cada contenedor cree que tiene su propio sistema independiente.

2. **Cgroups (Control Groups)**: limitan y asignan recursos (CPU RAM, etc) para que un contenedor no consuma todo el sistema.

3. **UnionFS**: permite capas de solo lectura para imágenes y capas de escritura para contenedores.

## Diferencias entre Imagen y Contenedor:

### Imagen

Archivo estático e immutable que contiene:

- \* El sistema operativo base
- \* Binarios y librerías necesarios
- \* Tu aplicación y dependencias

- No se ejecuta por si sola, es solo la plantilla para crear contenedores.
- Se construye a partir de un Dockerfile

## Contenedor:

- Es una instancia en ejecución creada a partir de una imagen
- Es un proceso aislado que comparte el kernel del host
- Tiene su propia capa de lectura y escritura temporal (los cambios se pierden al detenerlo, salvo que uses volúmenes)

## Ejemplo:

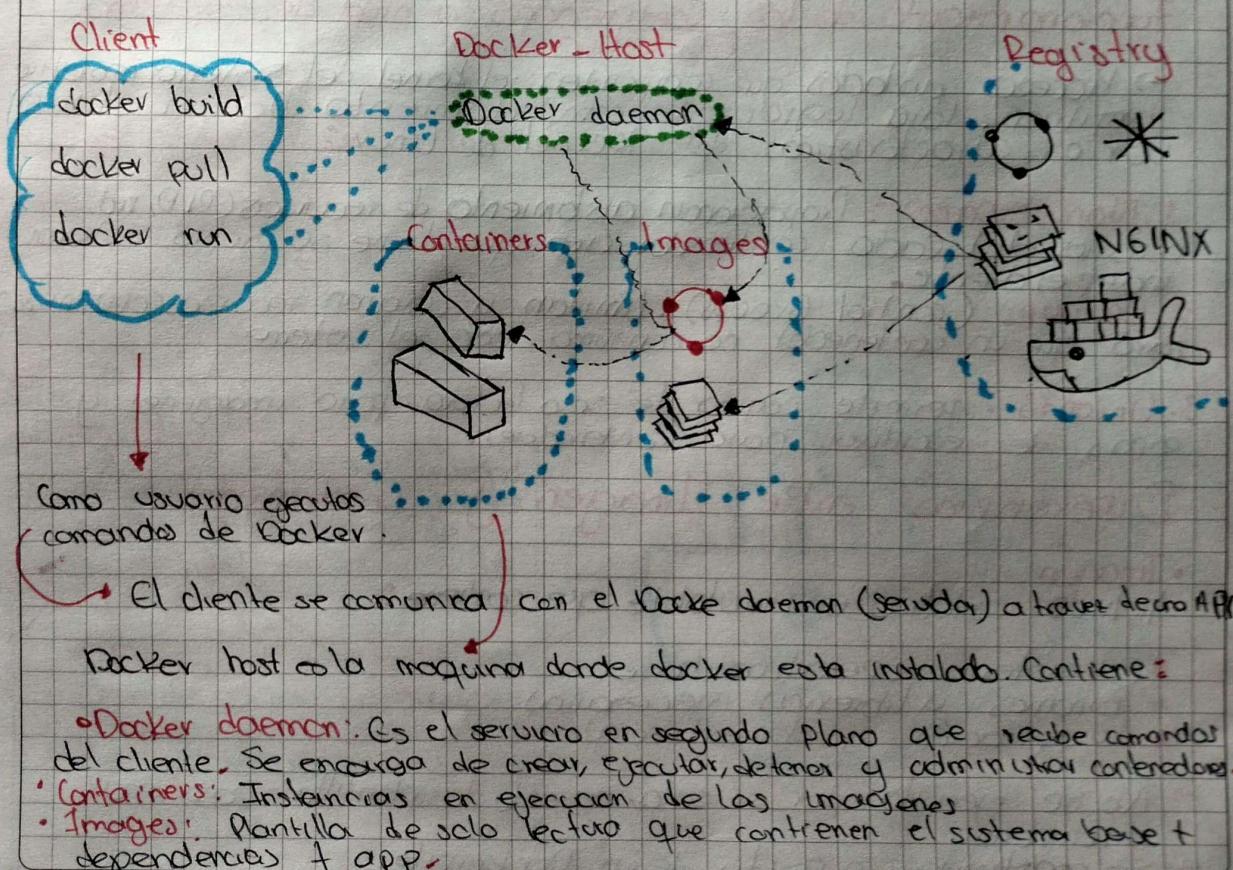
docker pull ubuntu:20.04 → descarga la imagen de Ubuntu  
docker run -it ubuntu:20.04 bash → crea un contenedor basado en la imagen y ejecuta un shell dentro.

## Ejemplo:

Imagen → clase  
contenedor → Objeto.

## Docker Engine

Docker Engine es la tecnología cliente-servidor para construir y containerizar aplicaciones en Docker. Esencialmente, soporta todas las tareas relacionadas con la ejecución de tu aplicación basada en contenedores.



Día	Mes	Año	Hora	Institución			
Alumno					Código	Materia	
Curso	Bimestre	Semestre	Salón	Hoja No.	de	CALIFICACIÓN	
Profesor							

**Registry:** Es un repositorio donde se almacenan las imágenes Docker. Ejemplo: **Docker hub**. De aquí se pueden descargar imágenes o subir imágenes.

## Flujo de Trabajo Representado en la Imagen

1. Cliente ejecuta comandos
2. El docker daemon recibe esos comandos
3. Si ejecutas docker pull el daemon busca la imagen en el Registry y la descarga.
4. La imagen descargada se guarda en la sección Images del Docker host
5. Cuando haces docker run, el daemon usa esa imagen para crear un contenedor en ejecución.
6. Los contenedores funcionan de manera aislada, pero todos estar gestionados por el mismo Docker daemon.

### En Resumen:

- El cliente manda comandos → daemon los ejecutan → las imágenes se descargan de un registry → a partir de las imágenes se crean los contenedores.

### Dockerfile:

Es un documento de texto que incluye instrucciones para construir imágenes Docker. Al leer este archivo, Docker construirá automáticamente nuevas imágenes.

El comando docker build crea una imagen a partir de un dockerfile y un contexto. Este contexto es un conjunto de archivos que se encuentra en una ruta o URL especificada.

### Volumen en Docker:

Es un mecanismo que usa docker para persistir datos fuera del ciclo de vida de un contenedor.

No realmente, cuando creas un contenedor y guardas algo dentro él de él (por ejemplo, un archivo en app/data), esos datos se pierden cuando el contenedor se elimina.

Para evitar eso, se usan volúmenes, que son áreas de almacenamiento gestionadas por Docker que no se borran al eliminar el contenedor.

## Características Principales:

1. **Persistencia:** Los datos sobreviven aunque el contenedor muera, se reinicie o lo elimines.
2. **Compartición de datos:** Varios contenedores pueden acceder al mismo volumen, lo cual sirve para que comparten información. Ejemplo: un contenedor de aplicación y uno de base de datos compartiendo archivos de configuración.

## 3. Independencia del contenedor:

El volumen está fuera del sistema de archivos temporales del contenedor.

## Redes en Docker:

Las redes en docker permiten que los contenedores se comuniquen entre sí y con el mundo exterior (internet o tus host). En otras palabras son como los cables virtuales que conectan los contenedores.

## Tipos de Redes en Docker:

### 1. Bridge:

- Es la red que docker crea por defecto
- Los contenedores en esta red pueden comunicarse entre sí usando el nombre del contenedor como host name.

### 2. Host:

El contenedor comparte la misma red del host. No hay aislamiento de red, el contenedor usa la IP del host directamente, es más rápido pero menos seguro.

## None:

El contenedor no tiene red, solo funciona con procesos internos.

## Overlay:

Conecta contenedores que están en distintos host (máquinas físicas / VMs). Muy usado en clústeres.

## Macvlan:

Le da al contenedor una IP propia en la red física, como si fuera otro dispositivo conectado a tu router. Ideal para integrarlo a redes ya existentes.

## Orquestación de Contenedores:

Es la gestión automatizada de múltiples contenedores, asegurando escalado, balanceo de carga, alta disponibilidad y despliegues consistentes.

## ¿Por qué se necesita?

- Escalar horizontalmente aplicaciones
- Manejar fallos automáticamente
- Coordinar redes y volúmenes entre múltiples nodos.

## Herramientas:

- Docker Compose: ideal para desarrollo, define (múltiples) contenedores con docker-compose.yml
- Kubernetes: estándar de la industria para producción. Escala masivamente con pods, servicios y despliegues.

## Conclusión:

Docker revolucionó la forma en que se despliegan aplicaciones:

- Introdujo portabilidad, rapidez y consistencia en el desarrollo.
- Permite ejecutar aplicaciones en cualquier entorno sin conflictos.
- En combinación con Kubernetes y otros orquestadores, es el pilar de la computación nativa en la nube.

## Comandos Básicos de Docker:

### Comandos Básicos de Información:

```
docker --version # Ver la versión de docker instalada  
docker info # Información general del sistema docker  
docker ps # Listar contenedores en ejecución  
docker ps -a # Listar todos los contenedores.  
docker images # Listar imágenes descargadas  
docker volume ls # Listar volúmenes  
docker network ls # Listar redes
```

### Trabajar con imágenes:

```
docker pull ubuntu # Descargar imagen de Ubuntu  
docker images # Ver imágenes locales  
docker rmi ubuntu # Eliminar imagen  
docker search nginx # Buscar imágenes en Docker Hub
```

### Crear y manejar contenedores

```
docker run hello-world # Ejecutar contenedor de prueba  
docker run -it ubuntu bash # Correr un contenedor  
docker run -d nginx # Interactuar con Ubuntu.  
# Correr contenedor en segundo plano  
docker run -d -p 8080:80 nginx # Mapear puerto host:contenedor.  
docker exec -it cids bash # Entrar al contenedor en ejecución
```

```
docker stop <ids> # Detener un contenedor  
docker start <ids> # Iniciar un contenedor detenido  
docker restart <ids> # Reiniciar un contenedor  
docker rm <ids> # Eliminar contenedor.
```

### Construcción de imágenes propias

```
docker build -t mi-app . # Construir img desde Dockerfile.  
docker tag mi-app miusuario/mi-app:v1 # Etiquetar img  
docker push miusuario/mi-app:v1 # Subir img a Docker Hub.
```

### Volumenes:

```
docker volume create mi-volumen # Crear un volumen.  
docker run -d -v mi-volumen:/data nginx # Montar volumen en contenedor.  
docker volume inspect mi-volumen # Ver detalles del volumen.  
docker volume rm mi-volumen # Eliminar volumen.
```