

Nombre	2023-2024 Facultad de Ciencias Físicas y Matemáticas	Fecha	10/10/2023
Profesor	Alvaro M. Gómez	Materia	Introducción a la Computación
Institución	Universidad de Guadalajara	Carrera	Ing. en Sistemas
		Curso	2023-2024
		Nota	

## Introducción a Docker

Docker es una tecnología que permite encapsular aplicaciones completas y sus entornos dentro de contenedores individuales. Cuando se ejecutan múltiples versiones de estos contenedores en una sola máquina, están aisladas entre sí como si estuvieran funcionando en sus propias máquinas dedicadas.

Docker es de código abierto, lo cual encaja bien con la ejecución de Linux en contenedores, así como con numerosos componentes de código abierto disponibles que ayudan a construir sistemas complejos.

Es la evolución lógica de las tecnologías utilizadas para el alojamiento y desarrollo de backend durante la última década o más. Esta evolución ha pasado de un tipo de alojamiento físico a uno lógico y ha sido impulsada por varios requisitos. Estos requisitos incluyen confiabilidad, accesibilidad, escalabilidad y seguridad. Este libro está dividido en tres secciones. La primera es una introducción a Docker, centrándose en el desarrollo local. La Segunda describe la metodología para probar, desplegar y escalar aplicaciones. La tercera entra en detalles sobre la seguridad al utilizar un diseño basado en contenedores.

## Los impulsores de Docker

La gama de servicios de alojamiento se limitó originalmente a servidores auto-alojados, alojamiento de servidores cubicados y alojamiento compartido. En 1994 y 1995, Best Internet Communications pasó de la nada a alojar 18,000 sitios web en un par de servidores pentium, que eran los servidores más poderosos de la época. Best también ofrecía alojamiento de servidor dedicado a través de la cuadrigación, conectividad de banda ancha dedicada y servicios premium de primer nivel. La mayoría de los sitios web alojados por Best eran de la variedad de alojamiento compartido. Todos estos sitios compartían el mismo servidor, los mismos discos duros, el mismo sistema de archivos, la misma RAM, las mismas CPU, las mismas conexiones de red, etc. No era raro que cualquiera de estos sitios web tuviera barras diagonales o contuviera un enlace al sitio desde un sitio muy popular al sitio ajeno. Esto causaría un gran aumento en el tráfico a uno de los aproximadamente 18,000 sitios, y un impacto en el rendimiento de los demás. A medida que la calidad de los sitios crecía y erigía más recursos, sus administradores pasaban a un alojamiento dedicado en el mismo lugar o al auto-alojamiento.

## Uso de la virtualización para economizar el uso de recursos

La virtualización es el proceso de exponer una porción de una máquina física como una máquina lógica o virtual que actúa lo suficientemente como una máquina real para que soporte la instalación de sistemas operativos completos, sus sistemas de archivos y el software que se ejecuta en el sistema operativo. Por ejemplo, una máquina con 64 GB de RAM y 4 CPUs podría ejecutar software de virtualización que se disfraza como cuatro máquinas de 16 GB de RAM y 1 CPU cada una. Esta máquina podría ejecutar cuatro instancias de Linux. La virtualización no es un concepto nuevo, ya que fue implementado por IBM a principios de la década de 1960.

Probablemente ganó popularidad general durante la década de 1980 cuando se utilizó para ejecutar MS-DOS y luego Windows en sistemas informáticos como el Apple Macintosh Original (Mac) y computadoras Unix como las estaciones

de trabajo de Sun y Silicon Graphics. El software de virtualización inicial utilizó las características disponibles en los CPUs de la época, pero a menudo simplemente emulaba el Conjunto de instrucciones de x86 en la familia 68000 o CPUs personalizadas de las estaciones de trabajo de Unix profesionales. SoftPC fue una de las ofertas más populares en la década de 1990. SoftPC era bastante lento, pero la capacidad de ejecutar aplicaciones de Windows o MS-DOS en una computadora Mac permitió el uso de estas máquinas en entornos empresariales y educativos.

En lugar de agregar compatibilidad con Microsoft Office a todos los programas en el Mac para apoyar el intercambio de archivos entre usuarios de Windows/MS-DOS y usuarios de Mac, los usuarios podían ejecutar Microsoft Office. La gente lo vio en acción y vio el valor en ello. Windows era el sistema operativo dominante para el hogar y los negocios, y para encajar con Windows en el entorno corporativo, se necesitaba algo como SoftPC. El problema con SoftPC es que era una emulación de software pura, que era bastante lenta en el uso real. La virtualización era superior a la emulación en términos de rendimiento. Se fundaron empresas enteras alrededor de proporcionar soluciones de virtualización para consumidores o negocios. VMware, fundada en 1998, fue una de las primeras de estas empresas.

Innotek desarrolló VirtualBox y lo lanzó como código abierto en 2007, y luego fue adquirido por Sun Microsystems en 2008. Luego, Sun fue adquirida por Oracle en 2010. Parallels, una solución virtualizada (de virtualización) para Mac, fue desarrollada en 2004 y se volvió común en 2006.

El valor de la virtualización alentó a los fabricantes de chips a agregar gradualmente soporte de CPU para virtualización. Con el soporte de la CPU, un sistema basado en x86 podía ejecutar máquinas virtualizadas a software a una velocidad lo suficientemente cercana a la nativa como para ser mucho más tolerable. Esto a su vez llevó a las empresas de estaciones de trabajo (como Apple, Sun y Silicon Graphics) a cambiar a las CPU x86. Un componente clave del software de virtualización es el hipervisor. El hipervisor presenta la máquina virtual al sistema operativo elegido y luego gestiona los recursos y la ejecución de las máquinas virtuales a lo largo del tiempo. Las máquinas virtuales en sí son configurables, al menos en lo que respecta a la cantidad de RAM, el número de núcleos lógicos de CPU, la memoria de la tarjeta gráfica, los archivos del disco del sistema operativo host para actuar como unidades de disco virtual en la máquina virtual, el montaje y el desmontaje de CD-ROM en la unidad de CD-ROM virtual, y así sucesivamente.

El hipervisor asegura que estos recursos estén verdaderamente disponibles y que ninguna máquina virtual agote los recursos de la máquina host para las otras máquinas virtuales. Para la empresa, los requisitos eran algo diferentes. En lugar de proporcionar máquinas virtuales a través de un sistema operativo general como Linux, todo el sistema operativo podría optimizarse solo para ser el hipervisor. VMware ofreció su sistema operativo *Elastic Sky X Integrated* (*ESXi*) en 2004. El laboratorio de computación de la Universidad de Cambridge desarrolló el hipervisor Xen a fines de la década de 1990, y la primera versión estable fue lanzada en 2003. Xen fue originalmente el hipervisor utilizado por Amazon para su oferta de *Elastic Compute Cloud*, antes de mudarse a KVM.

## Introducción a los contenedores

Versiónes anteriores de Docker instalaban VirtualBox para crear su máquina virtual, pero la tecnología de virtualización más reciente implementada dentro de Docker para contenedores de Linux espera que el sistema operativo host o la máquina virtual esté ejecutando Linux.

Los contenedores comparten el núcleo de Linux con el host. Docker se puede utilizar para ejecutar contenedores nativos de Windows, de manera similar a los

Contenedores de Linux. El núcleo de windows se comparte entre el host y los invitados. Para fines de discusión, nos enfocaremos en el host y las invitadas de Linux. Los contenedores de Docker se utilizan típicamente para implementar algo como máquinas virtuales sin cabeza.

El uso de máquinas virtuales para cada aplicación para la cual podrías crear un contenedor es costoso: debes reservar una cantidad fija de RAM y espacio en disco para la máquina virtual. En un MacBook Pro de 16 gigabytes de RAM, puedes encajar aproximadamente tres máquinas virtuales de 4 gigabytes de RAM ejecutándose al mismo tiempo. Necesitas tener algo de RAM para que el sistema operativo host funcione. Privar a las máquinas virtuales host o invitadas de RAM hará que intercambien lo que aplastará el rendimiento.

### Usando Docker para el desarrollo

Una gran razón para usar Docker en desarrollo es que no tienes que instalar ningún programa, más allá de Docker mismo, en tu host para habilitar el desarrollo. Por ejemplo, puedes ejecutar Apache en tu contenedor sin instalarlo en tu estación de trabajo.

También puedes mezclar y combinar versiones de software dentro de tus contenedores. Una arquitectura de micro servicios podría requerir que un contenedor use la versión 10 de Node.js. Esto obviamente es problemático en un único host, pero es sencillo cuando se usa Docker. Un contenedor instala y ejecuta la versión 8, y otro contenedor instala y ejecuta la versión 10.

Durante el desarrollo, puedes compartir los archivos de desarrollo de tu proyecto con el contenedor para que cuando edits estos archivos, el contenedor vea que los archivos han cambiado. Cada contenedor tiene su propio conjunto de variables ambientales globales.

Es típico configurar la aplicación usando variables de entorno, en lugar de en el código fuente o archivos de configuración dentro del contenedor. Cuando estés listo para desplegar o publicar un contenedor, puedes enviarlo a un servicio de hospedaje de contenedores, como Docker Hub. De hecho, Docker Hub es una excelente fuente de contenedores ya existentes que pueden ayudarte en el desarrollo de tu proyecto.

Hay imágenes de contenedor pre-hechas para MongoDB, Node.js (varias versiones), Apache, y así sucesivamente. La construcción de contenedores es efectivamente orientada a objetos. Heredas de un contenedor base y agregas la funcionalidad que necesitas a eso. Puedes crear una aplicación de Node.js en un contenedor que comienza con un contenedor de Node.js listo para usar, instalar paquetes de npm en el contenedor y ejecutar tu código personalizado en el contenedor.

### Introducción a Docker

Antes de entrar en el código, asegúremos de que Docker esté instalado correctamente. El comando `Docker ps` imprime una lista de todos los contenedores de Docker en ejecución. Podemos ver que no tenemos contenedores en ejecución y hay un comando `docker` real:

```
% docker ps
```

CONTAINER

ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			

Un Dockerfile es un archivo de texto que define cómo construir una imagen de contenedor Docker. El contenedor no se inicia. Simplemente se crea en el disco. Una vez construido, puedes iniciar tantas instancias como desees.

## Automatizando comandos de Docker a través de scripts de sh

Vamos a hacer un uso intensivo del comando de la línea de comandos de docker y de scripts en sh para automatizar el uso de la línea de comandos. El uso de archivos de script en sh tiene algunas ventajas. Una vez que se crea el archivo de script, no tienes que recordar cuáles son todos los comandos de la línea de comandos. Una vez que el script es correcto, no tendrás problemas debido a errores tipográficos o a comandadores de línea de comandos incorrectos. Escribir el nombre del archivo de script es mucho más corto y tu shell debería autocompletarlo cuando escribas los primeros caracteres del nombre y presiones la tecla tab. Finalmente, los nombres de los scripts son muy técnicos: build.sh significa construir el contenedor, run.sh significa ejecutar el contenedor, y así sucesivamente.

Los Scripts sh que proporcionamos son los siguientes:

- ./build.sh: Esto construye el contenedor a partir del Dockerfile. Querrás ejecutar este script cada vez que edits el Dockerfile, o si el contenedor necesita ser construido de otra manera.
- ./debug.sh: Esto ejecuta el contenedor en modo de depuración. En modo de depuración, Apache se ejecuta en modo de primer plano y puedes presionar Ctrl+C para detener el contenedor.
- ./run.sh: Esto ejecuta el contenedor como un demonio. A diferencia del script ./debug.sh, se te devolverá al símbolo del sistema, con el contenedor en ejecución en Docker. Usarás este script para ejecutar el contenedor localmente, como si estuvieras en producción, para que puedas probar el comportamiento en producción.
- ./stop.sh: Cuando tienes tu contenedor en ejecución en segundo plano, este script se puede usar para detenerlo.
- ./shell.sh: A veces, al crear tu contenedor y editar el Dockerfile, las cosas no funcionan como se esperaba. Puedes usar este script para obtener una línea de comando Bash ejecutándose dentro del contenedor. Desde esta línea de comando, puedes inspeccionar y diagnosticar los problemas.
- ./persist.sh: Este script demuestra el uso de un volumen nombrado para persistir el estado de la aplicación dentro del contenedor. Es decir, con un volumen nombrado, puedes detener y reiniciar el contenedor y el contenido del volumen se persiste. El volumen se monta en el contenedor como si fuera un disco.

Veamos lo que hace el dockerfile, paso a paso:

1. El dockerfile hereda de la imagen de Debian en Docker Hub.
2. Establecemos la zona horaria del contenedor para que coincida con la zona horaria del host; en otras palabras, aseguramos que las marcas de tiempo de los archivos dentro del contenedor y en el host coincidan. Esto es importante al mapear directorios del host al sistema de archivos del contenedor.
3. Luego instalamos Apache y PHP 7.3. Estos se instalan en el sistema de archivos del contenedor y no en el sistema de archivos del host. Hemos evitado el problema de contaminación de tener una versión de ambas instaladas en el host que luego se vuelven no utilizadas cuando no estamos trabajando en este proyecto.
4. También instalamos algunas utilidades de línea de comandos que nos permiten examinar el estado del contenedor construido desde una shell bash.

Nombre		Fecha	
Profesor		Materia	
Institución		Curso	Nota

5. Por defecto, el usuario y grupo que ejecutarán el proyecto en el contenedor es root. Para proporcionar una seguridad típica de Unix/Linux, queremos ejecutar como un usuario real; en nuestro caso, el nombre del usuario es app. Así que añadimos el usuario al entorno del contenedor con user add.

6. Vamos a poner en nuestros scripts PHP en /home/app, con la capacidad de mapear nuestro directorio de trabajo con nuestros scripts PHP en el host sobre /home/app.

7. Nuestra aplicación de demostración escribe su estado en /data, por lo que necesitamos crearla y asegurarnos de que el script PHP que se ejecutan como una aplicación de usuario pueda leer y escribir archivos allí.

8. Creamos un archivo de configuración PHP personalizado que queremos usar dentro del contenedor, por lo que lo copiamos al contenedor en la ubicación correcta en el sistema de archivos.

9. Necesitamos habilitar los módulos userdir y php7.3. Esto nos permite ejecutar scripts PHP desde Apache, así como acceder a nuestros scripts PHP en /home/app/public\_html a través de una URL como <http://localhost/~app/index.php>.

10. Cuando se inicia el contenedor, necesita ejecutar algún programa o script dentro del contenedor. Usamos un script sh llamado entrypoint.sh en el directorio /home/app para iniciar la aplicación. Podemos editar este archivo para adaptarlo a nuestras necesidades durante el desarrollo.

Podríamos haber elegido entre una variedad de versiones de Linux para comenzar. Elegimos Debian aquí porque los comandos de configuración deberían ser familiares para la mayoría de los lectores.

Si instalas Debian en una máquina virtual, usarías los mismos comandos para instalar y mantener su sistema. Debian no es la imagen de Linux más pequeña o ligera para comenzar. Alpine es una excelente opción si quieres que tu contenedor use menos recursos. Si decides usar Alpine, asegúrate de informarte sobre cómo instalar paquetes y mantener el sistema usando Alpine. Ten en cuenta que, independientemente de la imagen de Linux desde la cual comiences, estás compartiendo el núcleo de Linux con tu máquina anfitrión. Solo dentro del contenedor es Debian; tu sistema operativo anfitrión puede ser otra distribución de Linux. Lo que instales dentro del contenedor no se instala en tu estación de trabajo, solo dentro del contenedor.

Obviamente no deberías mezclar, por ejemplo, comandos de Debian y paquetes instalados directamente en una estación de trabajo Arch Linux.

## Imagen

Una imagen es un paquete inmutable con todo lo que una app necesita para ejecutarse: Sistema base (por ejemplo, Debian o Alpine), runtime (por ejemplo, .NET), bibliotecas, archivos de tu proyecto y configuraciones predeterminadas. Se construye a partir de un Dockerfile, que indica paso a paso cómo armalar (por ejemplo: "parte de esta base", "copia estos archivos", "ejecuta este comando de instalación", "define este comando de arranque"). Cada paso genera una capa; Docker guarda estas capas en caché, de modo que si cambias solo el código, no recomponer las capas anteriores, acelerando los builds.

Las imágenes se identifican por nombre y tag (ej. your-api:1.0) y también por un digest (un hash que garantiza que el contenido no cambia). Se almacenan y distribuyen desde registries (Docker Hub, GitHub container Registry, tu propio registry privado). Como son inmutables, son ideales para tener versiones reproducibles: si hoy y mañana ejecutas la misma imagen, obtienes exactamente el mismo entorno.

En tu API de C#, la imagen típicamente incluye: una imagen base con .NET SDK para compilar tu código publicado, y una imagen final más ligera con el runtime de .NET para ejecutar. Eso te asegura que el servidor donde despliegas no necesita tener nada instalado aparte de Docker.

### Qué es un Dockerfile:

Un Dockerfile es un archivo de texto (sin extensión especial, simplemente llamado Dockerfile) donde le das instrucciones a Docker para construir una imagen. Piensa en él como una receta paso a paso: defines de qué parte arrancas, qué ingredientes agregas y qué acciones ejecutar, hasta que dejes lista la imagen final que luego se convertirá en contenedor.

### Cómo funciona en esencia:

Cuando ejecutas docker build, Docker lee el dockerfile de arriba hacia abajo y ejecuta cada instrucción creando una capa. Cada capa se guarda en caché, de modo que si algo no cambia (por ejemplo, las librerías que ya instalaste), no se vuelve a reconstruir, acelerando el proceso.

### Instrucciones más comunes

- FROM: Indica la imagen base desde que la partes (por ejemplo, mcr.microsoft.com/dotnet/aspnet:8.0).
- WORKDIR: Define la carpeta de trabajo dentro de la imagen.
- RUN: Ejecuta comandos en el proceso de construcción (instalar dependencias, compilar códigos).
- EXPOSE: Documenta el puerto en el que escuchará tu aplicación.
- ENV: Define variables de entorno.
- CMD o ENTRYPOINT: Define el comando que se ejecutará cuando arranque el contenedor a partir de esta imagen.

### Contenedor

Un contenedor es una instancia en ejecución de una imagen. Si la imagen es el "moldé", el contenedor es "el producto funcionando". Cuando arrancas un contenedor, Docker crea una capa de escritura efímera sobre las capas de la imagen (que son solo de lectura). Esta capa se borra cuando eliminas el contenedor. Por eso se dice que los contenedores son efímeros y recreables: si algo sale mal, lo paras y lo vuelves a crear; el entorno queda limpio y consistente.

El contenedor corre aislado del host y de otros contenedores mediante tecnologías del Kernel (namespaces y cgroups). Este aislamiento implica su propio espacio de procesos, red virtual y sistema de archivos. Para persistir datos (por ejemplo, la base de datos), no confíes en la capa efímera: usa volúmenes o bind mounts para que los datos queden fuera del ciclo de vida del contenedor. Para exponer servicios, el contenedor escucha en un puerto interno y Docker hace el mapeo de puertos hacia el host (por ejemplo, contenedor en 8080 → host en 8080).

En tu escenario: el contenedor de la API .NET expone :8080 y el contenedor de la base de datos expone su puerto interno (5432 Si es Postgres, 1433 Si es SQL Server). Ambos viven en la misma red docker (bridge), se resuelven por nombre de servicios (por ejemplo, db) y no necesitan abrir puertos al host para hablar entre sí. Solo abres hacia el host los puertos que quieras consumir desde afuera (Swagger, PgAdmin, etc.).

## Docker Engine

Docker Engine es el motor que hace posible todo. Consta del daemon (dockerd) que corre en segundo plano, expone una API y orquesta imágenes, contenedores, redes y volúmenes; y del CLI (docker) que es el cliente con el que tú das órdenes. Cuando ejecutas un comando como docker build o docker run, el CLI habla con el daemon, y este desencadena lo necesario: Compilar Capas Con Buildkit, crear redes, asignar recursos (CPU, RAM), conectar volúmenes, descargar imágenes del registry, etc.

A nivel de sistema, el Engine usa componentes como containerd y runc para la ejecución de bajo nivel. En Linux corre de forma nativa. En Windows y macOS se apoya en una pequeña VM. Para ti, la experiencia es la misma: el daemon mantiene un inventario local de imágenes, contenedores, redes y volúmenes; tú defines qué correr y cómo, y el Engine se encarga del resto real.

En el día a día con tu API: El Engine compila tu Dockerfile, guarda la imagen en tu equipo, arranca los contenedores con la red adecuada, inventa las variables de entorno (como cadenas de conexión y secretos JWT) y mantiene los procesos activos. Si usas docker compose, el Engine orquesta varios servicios como un todo (API, DB, PgAdmin), resolviendo dependencias y reinicios.

## Windows

En Windows, la forma recomendada es instalar Docker Desktop. Este programa incluye todo lo necesario: el motor Docker (Docker Engine), la interfaz gráfica y la integración con WSL2 (Windows Subsystem for Linux). Primero debes asegurarte de que tu sistema operativo sea Windows 10 Pro, Enterprise o Education a partir de la versión 2004, o Windows 11 en cualquiera de sus ediciones. También necesitas tener virtualización habilitada en la BIOS, porque Docker se apoya en ella para ejecutar contenedores. Una vez cumples estos requisitos, descargas el instalador de Docker Desktop desde la página oficial de Docker, lo ejecutas y sigues el asistente. Durante la instalación te pedirá habilitar o confirmar la activación de WSL2. Al finalizar, podrás abrir Docker Desktop, verificar que esté corriendo y usar la terminal de PowerShell o CMD para ejecutar comandos como docker run hello-world y confirmar que funciona.

## Linux

En Linux la instalación se hace directamente desde los repositorios oficiales de Docker. El proceso varía un poco según la distribución, pero el flujo general es este: primera actualizas el sistema, luego instalas los paquetes necesarios para permitir repositorios HTTPS, después agregas la clave GPG oficial de Docker y configuras el repositorio de Docker en tu distribución. Con eso listo, instalas Docker -ce (Community Edition), que es la versión gratuita y completa del motor. Después habilitas y arrancas el servicio con systemctl enable docker y systemctl start docker. Finalmente, agregar tu usuario al grupo docker para poder ejecutar comandos sin necesidad de sudo. Por ejemplo, en Ubuntu el comando sería sudo usermod -aG docker \$USER. Tras cerrar la sesión y volver a entrar, ya puedes probar docker run hello-world para confirmar que el motor funciona correctamente.

## Mac

En macOS también se utiliza Docker Desktop, muy parecido a Windows. Descargas el archivo .dmg desde el sitio oficial de Docker, lo instalas arrastrando el ícono de Docker a la carpeta de aplicaciones y luego lo abres. Docker Desktop en Mac usa una máquina virtual ligera llamada HyperKit o en los equipos más recientes con chip Apple Silicon (M1/M2/M3), aprovecha la virtualización nativa. Los requisitos principales son: macOS 11.0 Superior y al menos 4 GB de memoria. Una vez que Docker Desktop esté en ejecución aparecerá el ícono de la ballena en la barra Superior. Desde la terminal te puedes ejecutar comandos de Docker igual que en Linux, y también puedes probar docker run hello-world para verificar que la instalación esté correcta.

## Docker Version

Este comando muestra información sobre la versión de Docker que tienes instalada. Verás dos bloques: Client y Server. El primero corresponde al cliente (herramienta de línea de comandos que usar) y el segundo al daemon (Docker Engine, motor que gestiona imágenes y contenedores).

Si solo aparece el cliente y no el servidor, significa que Docker Engine no está corriendo correctamente. Usualmente, al instalar Docker Desktop en Windows/Mac o el servicio en Linux, este comando es la primera prueba de que Docker quedó bien instalado.

## Docker run hello-world

Este comando es el clásico primer test de funcionamiento. Lo que hace es descargar la imagen llamada hello-world desde el Docker Hub (si no la tienes ya en tu máquina), crear un contenedor a partir de ella y ejecutarlo. La salida será un mensaje que confirma que tu instalación de Docker está operativa. Intermamente pasan tres cosas:

1. Docker busca la imagen hello world localmente, y si no está, la descarga.
2. Crea un contenedor con esa imagen.
3. Lo ejecuta, el contenedor imprime el mensaje y luego se apaga.

Ej. La manera más sencilla de comprobar que tanto el cliente como el servidor están funcionando y que puedes crear contenedores sin problemas.

## Docker ps

Este comando lista los contenedores en ejecución. Te muestra el ID del contenedor, la imagen desde la que se creó, el comando que está ejecutando, el estado (Up, Exited, etc.), los puertos externos y el nombre asignado.

Si no aparece nada en la lista, significa que en ese momento no tienes contenedores corriendo. Para ver todos (también los detenidos) se suele usar la variante docker ps -a.

## Docker stop

Sirve para detener un contenedor en ejecución. Los llamas junto con el ID o el nombre del contenedor que viiste con docker ps. Ejemplo: docker stop mi-contenedor o docker stop 123abc456. Lo que hace es enviar un señal para apagarlo de manera ordenada, permitiendo que el proceso dentro del contenedor se cierre correctamente.