

MÓDULO 1: GitHub – Docker

Docente: Héctor Frías

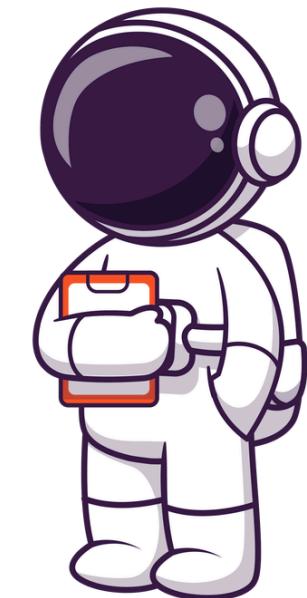


Clase 1:

- * ¿Qué es Git y para qué sirve?
- * Instalación y configuración
- * Git en local (uso diario)
- * Ramas (branches)
- * Conflictos

Clase 2:

- * GitHub y trabajo en equipo
- * Pull Requests
- * Flujos reales



Clase 3:

- * Introducción a Docker
- * Instalación + primeros comandos
- * Manejo de contenedores
- * Imágenes y Dockerfile
- * Práctica guiada + cierre

Clase 4 :

- * ¿Qué es Docker y para qué sirve?
- * Instalación y configuración
- * Git en local (uso diario)
- * Ramas (branches)
- * Conflictos



Git es un sistema de control de versiones. Sirve para guardar el historial de cambios de un proyecto y poder:

- Volver atrás si algo se rompe
- Trabajar en equipo sin pisarse
- Probar cambios sin miedo

Git trabaja localmente en tu máquina. **GitHub** es un servidor remoto para compartir ese código.

Ejemplo real:

Tenés un proyecto Angular y rompés todo con un cambio. Con Git podés volver al estado anterior en segundos



Instalación de Git Windows

1. Entrar a la página oficial de Git <https://git-scm.com/install/windows>
2. Descargar Git for Windows
3. Ejecutar el instalador
4. Durante la instalación:
 - o Aceptar la licencia
 - o Dejar las opciones por defecto
 - o Editor recomendado: Visual Studio Code (si lo usás)

5. Finalizar instalación Git se podrá usar desde:

- Git Bash
- PowerShell
- Terminal de VS Code



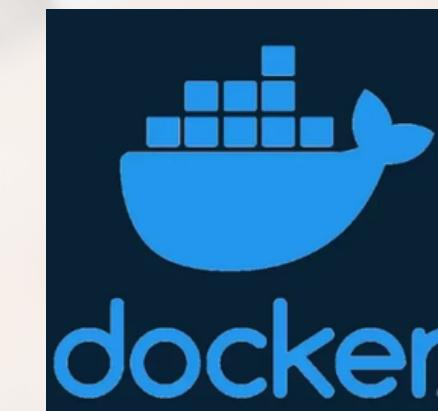
Crear cuenta en GitHub

1. Entrar a <https://github.com/>
 2. Click en Sign up
 3. Completar:
 - o Email
 - o Usuario
 - o Contraseña
 4. Confirmar email
- Con esto ya tenés tu cuenta de GitHub creada.



Descargar Docker Desktop

1. Entrar a <https://docs.docker.com/desktop/setup/install/windows-install/>
2. Descargar Docker Desktop for Windows
3. Ejecutar el instalador Durante la instalación:
 - Usar WSL 2 (recomendado)
 - No usar Hyper-V si no es necesario



configuración



Antes de usar Git, hay que decirle quién sos. Esa info queda en cada commit.

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tu@email.com"
```

--global significa que aplica a todos los proyectos
Cada commit queda firmado con tu nombre

Crear y versionar un proyecto



Inicializar repositorio:

git init

Esto crea una carpeta oculta .git donde vive todo el historial.

Ver estado:

git status

Te dice qué archivos están modificados o sin guardar.
Pasar cambios al "área de preparación":

git add .

Esto significa: estos cambios quiero guardarlos.

Guardar cambios:

git commit -m "mensaje"

Un commit es una foto del proyecto en ese momento.

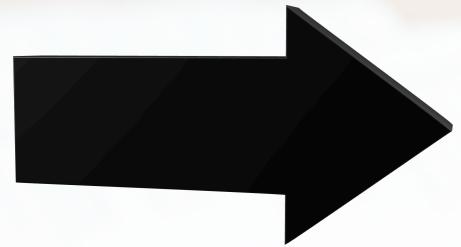
Buenas prácticas de mensajes:

feat: nueva funcionalidad

fix: corrección

refactor: mejora interna

git clone



El comando git clone se usa para crear una copia local de un repositorio Git remoto. Este comando descarga todos los archivos, ramas e historial de commits de la URL especificada a un nuevo directorio en tu equipo local.

Hector-Frias / FrontAtleta Public

Code Issues Pull requests Actions Projects Security Insights

main 1 Branch 0 Tags

Go to file

Clone

HTTPS GitHub CLI

<https://github.com/Hector-Frias/FrontAtleta.git>

Clone using the web URL.

Open with GitHub Desktop

Download ZIP

3 years

```
MINGW64:/c/Users/HFrias/Downloads
HFrias@NOT0245 MINGW64 ~/Downloads
$ git clone https://github.com/Hector-Frias/FrontAtleta.git
```

En Git, una rama (branch)

Es esencialmente un puntero móvil que apunta a una de tus confirmaciones (commits). Permite separar el trabajo en diferentes líneas de desarrollo sin interferir con el código principal.

Comandos básicos de git branch

El comando principal para gestionar ramas es `git branch`, pero se complementa con otros para navegar entre ellas:

Acción Comando

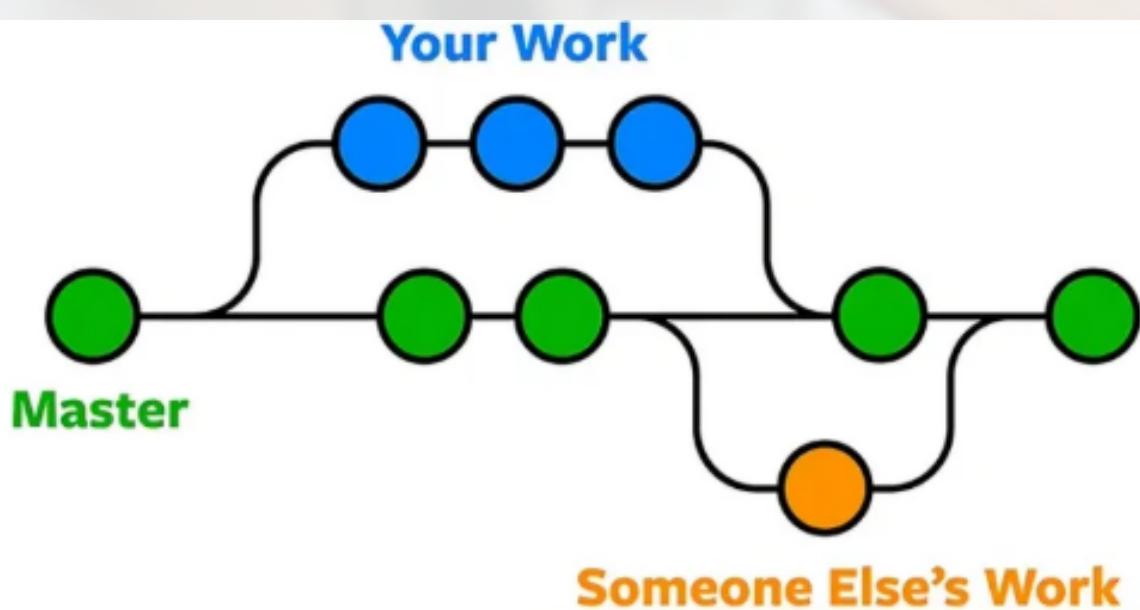
Listar ramas locales => **git branch**

Crear una nueva rama => **git branch <nombre>**

Eliminar una rama (ya fusionada) => **git branch -d <nombre>**

Forzar eliminación (no fusionada) => **git branch -D <nOMBRE>**

Renombrar rama actual => **branch -m <nuevo-nombre>**



Flujo de trabajo esencial

Crear y cambiar: Para empezar a trabajar en algo nuevo, usa **git switch -c <nombre-rama>**. Esto crea la rama y te mueve a ella inmediatamente.

Navegar: Para volver a la rama principal, usa **git switch main**.

Fusionar: Una vez terminado el trabajo, vuelve a la rama principal y usa **git merge <nombre-rama>** para integrar los cambios.

Sincronizar: Para ver ramas remotas (del servidor), usa **git branch -a**.

¿Por qué es importante?



Aislamiento: Puedes probar funciones experimentales o corregir errores sin afectar la versión estable del proyecto.

Velocidad: En Git, las ramas son extremadamente ligeras y rápidas de crear, ya que solo guardan una referencia (hash) al commit.

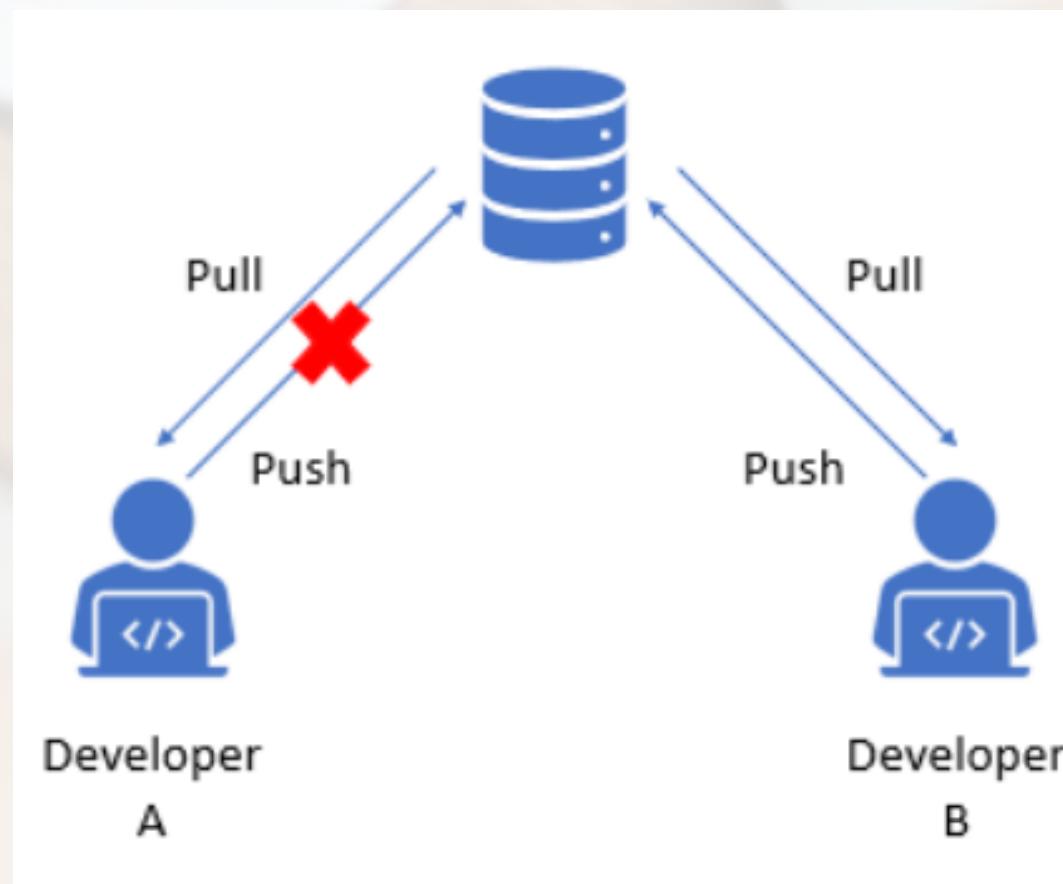
Colaboración: Permite que varios desarrolladores trabajen en el mismo proyecto simultáneamente en diferentes tareas.

conflicto de fusión

Los conflictos de fusión ocurren cuando se hacen cambios contrapuestos en la misma línea de un archivo o cuando una persona edita un archivo y otra persona borra el mismo archivo.

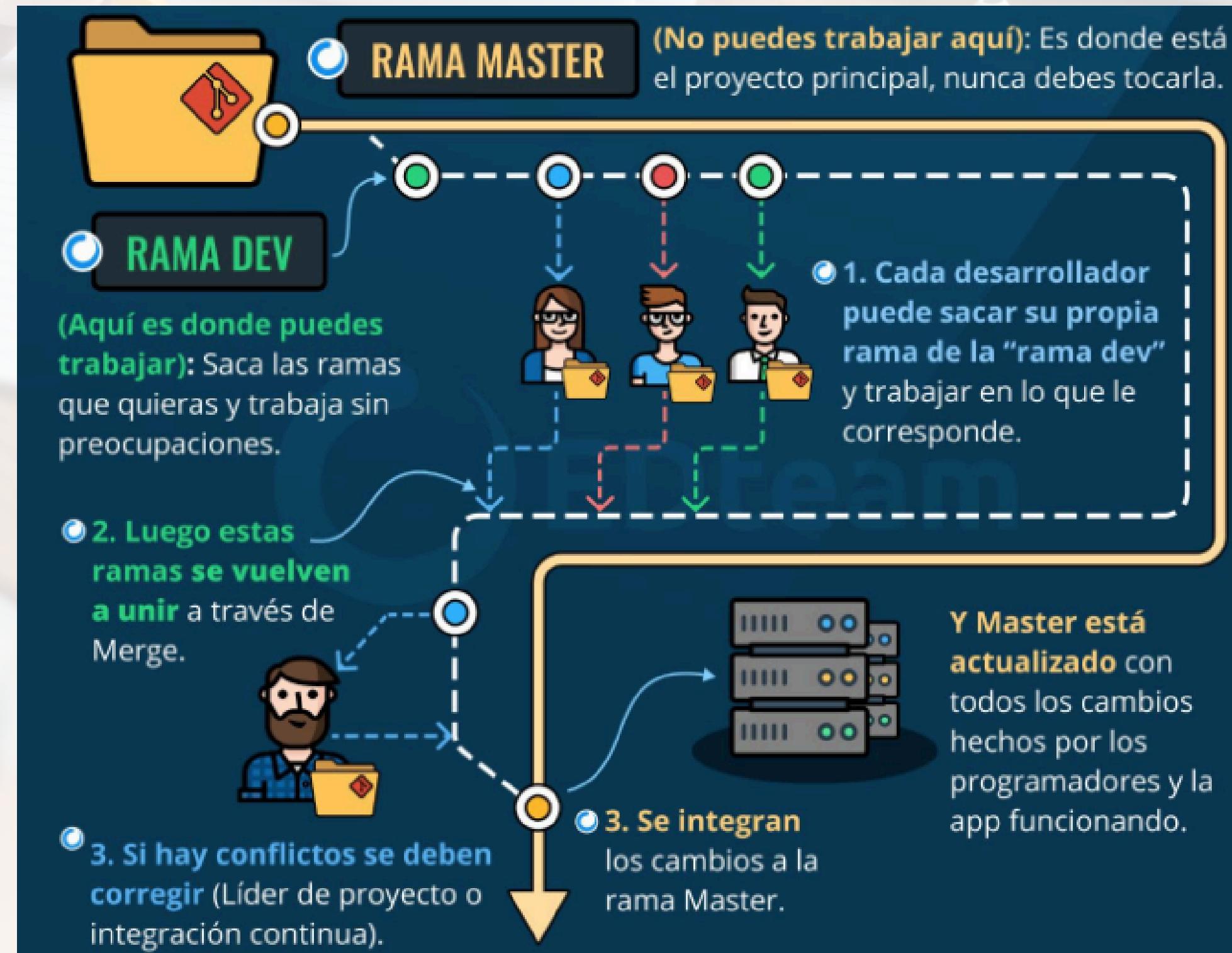
Para resolver un conflicto de fusión causado por cambios de líneas contrapuestos, debes decidir qué cambios incorporar desde las diferentes ramas de una confirmación nueva.

Por ejemplo, si tú y otra persona habéis editado en el archivo styleguide.md las mismas líneas de diferentes ramas del mismo repositorio de Git, recibirás un error de conflicto de fusión mediante combinación cuando intentes fusionar mediante combinación estas ramas. Debes resolver este conflicto de fusión con una confirmación nueva antes de que puedas fusionar estas ramas.



GitHub y trabajo en equipo

Organizaciones y Equipos: Permiten agrupar usuarios con diferentes niveles de permisos (lectura, escritura, administración) y mencionar a todo un equipo mediante @nombre-del-equipo para notificaciones rápidas.



GitHub y trabajo en equipo

Prefijos

El uso de prefijos en los nombres de las ramas ayuda a identificar rápidamente el propósito de la misma. A continuación, se muestran algunos tipos comunes de prefijos:

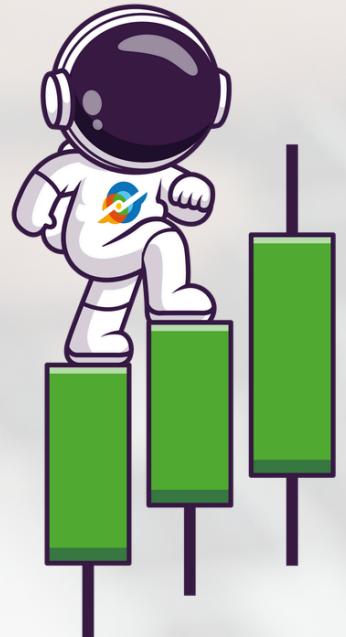
Feature: se utiliza para mejorar, actualizar o desarrollar nuevas funciones.

Bugfix: utilizado para corregir errores en el código.

Hotfix: usado para corregir errores críticos en el entorno de producción.

Release: se utiliza para ramas de lanzamiento de versiones.

Documentation: usado para escribir, actualizar o corregir documentación.



GitHub y trabajo en equipo

Orden y partes de los nombres de las ramas

Siempre incluya el texto "DIC-" al inicio del nombre de la rama, haciendo alusión a Dicsys.
Incluya el prefijo correspondiente según la tarea a realizar. Al finalizar agregue una barra lateral "/".
Descripción de la tarea.

Ejemplos

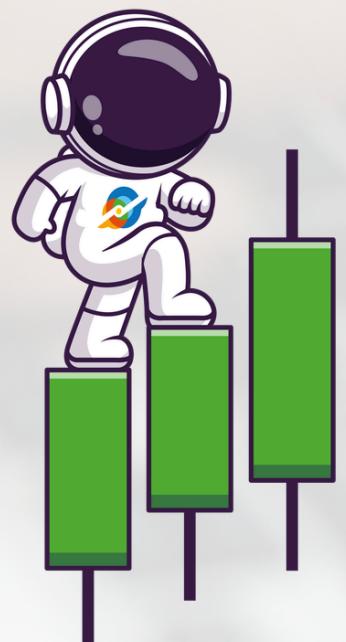
DIC-feature/login_mockup

DIC-bugfix/error-when-closing-session

DIC-hotfix/error-when-save-new-client

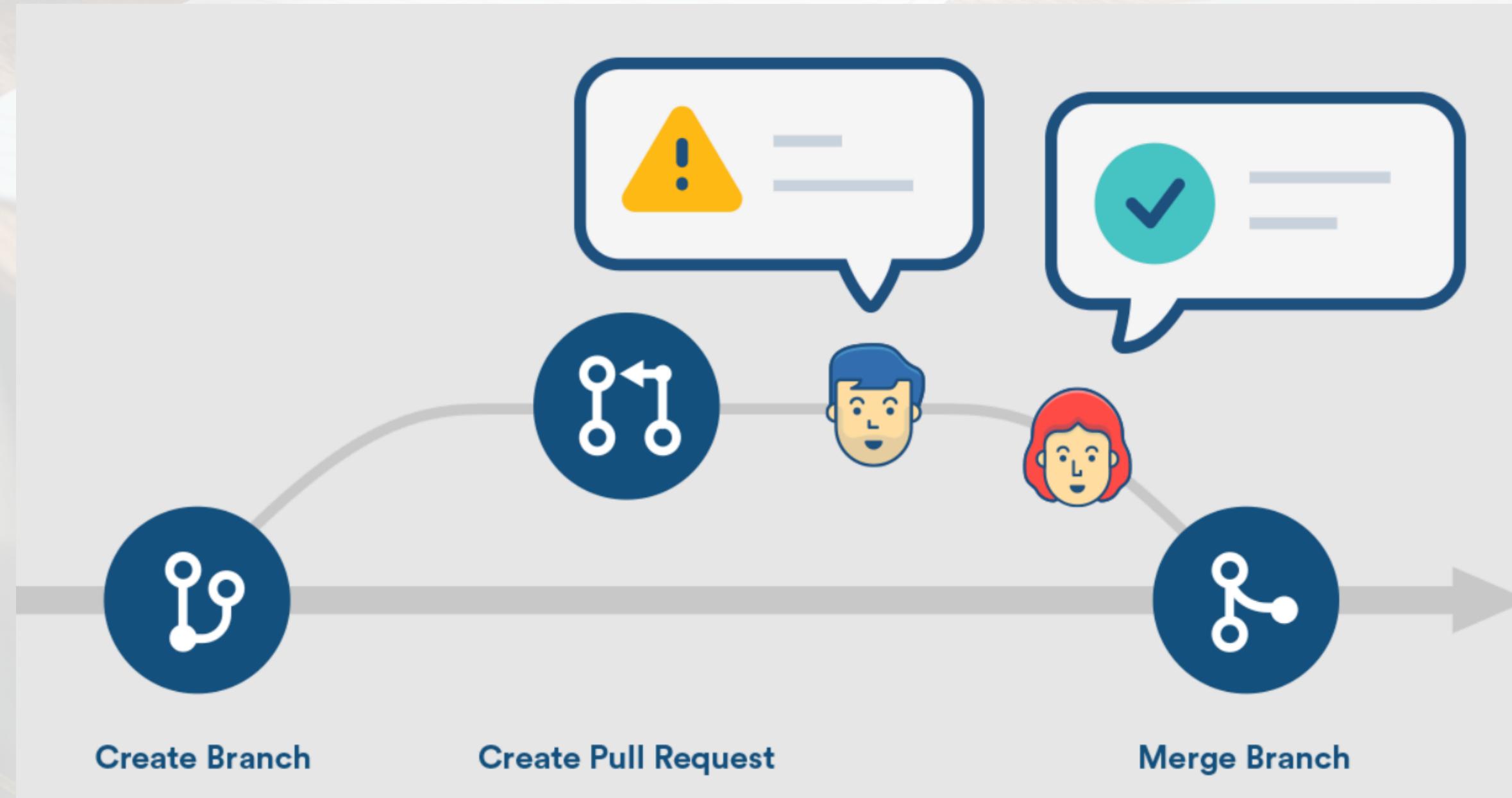
DIC-release/new-version-1.0.0

DIC-documentation/requirements-readme-updated



Pull Request (PR)

Es una propuesta para fusionar cambios de código de una rama a otra, comúnmente a la rama principal (main o master) en plataformas como GitHub, permitiendo a otros desarrolladores revisar, discutir y aprobar esos cambios antes de integrarlos, asegurando la calidad y el control del proyecto.



Docker



¿Qué problema resuelve Docker?

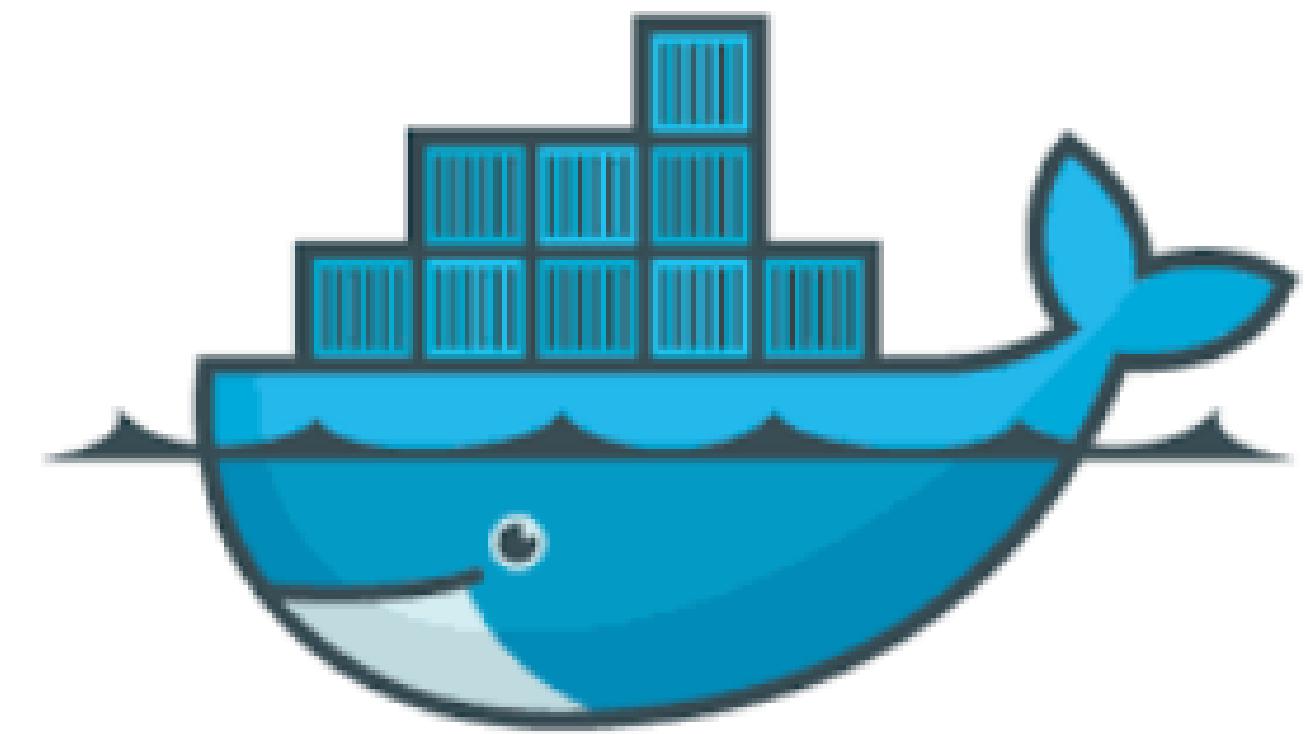
“Docker no sirve para programar mejor, sirve para que todo funcione igual en todos lados.”

Problemas clásicos

- “En mi máquina funciona”
- Diferencias entre:
 - SO
 - Versiones de Node / Java / DB
- Setups largos



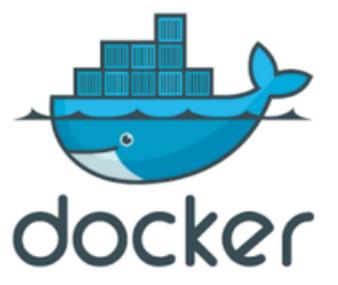
Docker



docker

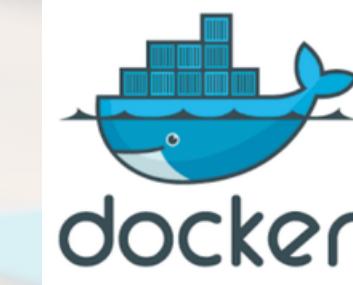
Docker es una plataforma abierta para desarrollar, distribuir y ejecutar aplicaciones. Docker permite separar las aplicaciones de la infraestructura para que puedas entregar software rápidamente. Con Docker, puedes gestionar tu infraestructura de la misma forma que gestiones tus aplicaciones. Al aprovechar las metodologías de Docker para distribuir, probar e implementar código, puedes reducir significativamente el tiempo entre la escritura del código y su ejecución en producción.

Docker



VM vs Contenedores

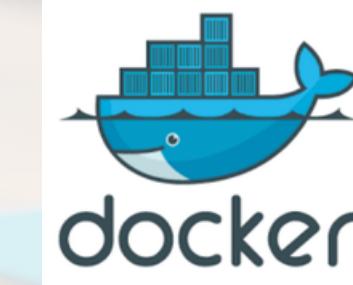
| Virtual Machine | Docker |
|----------------------------|----------------------|
| Sistema operativo completo | Comparte kernel |
| Pesadas | Livianas |
| Arranque lento | Arranque en segundos |
| Mucho consumo | Poco consumo |



BENEFICIOS

- **Cada contenedor está aislado de los demás.**
- **Es posible ejecutar varias instancias de la misma versión o diferentes versiones sin configuraciones adicionales.**
- **Con un comando, puedes descargar, levantar y correr todo lo que necesitas.**
- **Cada contenedor contiene todo lo que necesita para ejecutarse.**
- **Indiferente el sistema operativo HOST.**

Docker

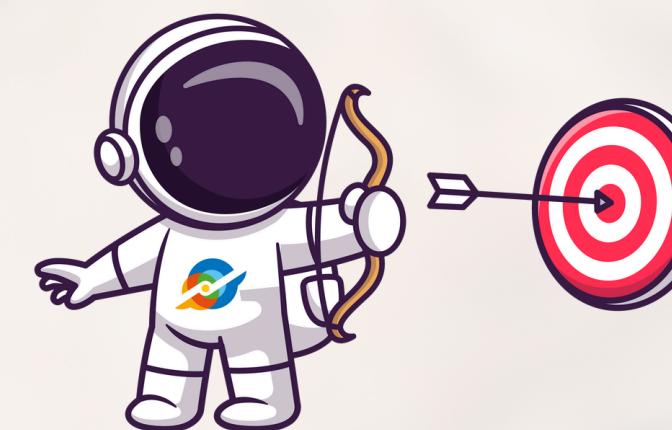


instalación + primeros comandos

Docker Desktop
Verificación:



`docker --version`
`docker info`



Docker



Primer contenedor

**Busca la imagen local
Si no existe → Docker Hub
Crea contenedor
Ejecuta
Finaliza**



`docker pull hello-world`
`docker run hello-world`

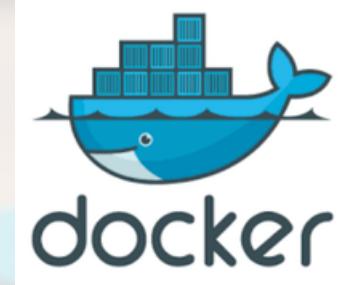


imagen de Docker

Es una plantilla ejecutable e immutable de solo lectura que contiene todo lo necesario para ejecutar una aplicación: código, bibliotecas, dependencias y archivos de configuración, ligero para crear contenedores.

Una imagen es como:

📦 una plantilla / receta / sistema operativo en miniatura

Contiene:

el sistema base (Linux)

el software (MySQL, Node, Nginx, etc.)

configuraciones necesarias

NO tiene datos dinámicos

👉 Es de solo lectura

Pensalo así:

Mundo real

Instalador .exe

Programa instalado

Documentos

Docker

Imagen

Contenedor

Volúmenes



contenedor

Un contenedor es:
una instancia en ejecución de una imagen

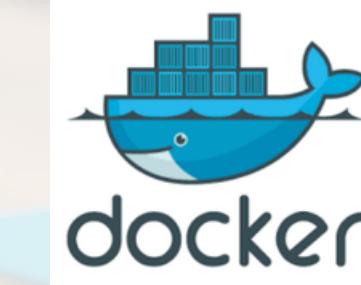
Es decir:

- la imagen es la plantilla
- el contenedor es esa plantilla corriendo

Características clave:

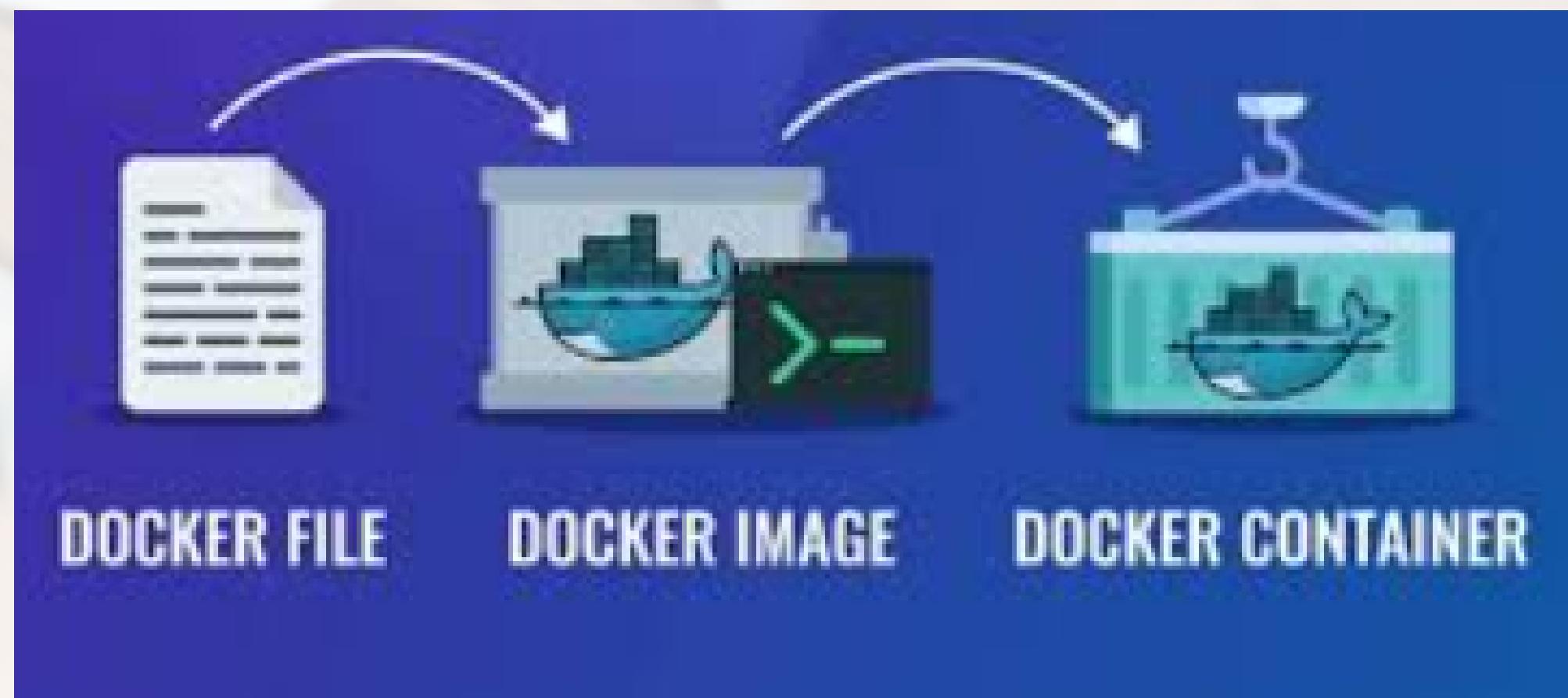
- Empaquetado unificado: Agrupa todo lo que la aplicación necesita para funcionar.
- Aislamiento: Separa la aplicación de su entorno, evitando conflictos de dependencias.
- Portabilidad: Se ejecuta consistentemente en cualquier lugar (portátil, nube, servidor).
- Ligereza y eficiencia: Comparte el sistema operativo anfitrión, consumiendo menos recursos que una VM.
- Ciclo de vida: Se crean a partir de imágenes (plantillas inmutables) y son instancias ejecutables, efímeras y dinámicas.

Docker



¿Cómo funciona?

- Imagen (Image): Es una plantilla de solo lectura que contiene el código y las dependencias.
- Contenedor (Container): Es una instancia en ejecución de una imagen, un entorno vivo y ejecutable.
- Motor Docker: Ejecuta múltiples contenedores sobre el mismo sistema operativo.





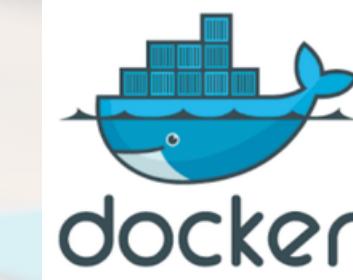
volúmenes

Un volumen es la forma que tiene Docker de guardar datos fuera del contenedor.

Porque los contenedores son efímeros:

- Si borrás el contenedor → se pierden los datos
- Si recreás el contenedor → arranca vacío

Los volúmenes evitan eso: los datos viven en el host, no dentro del contenedor.



volúmenes

✗ Problema sin volúmenes

Ejemplo:

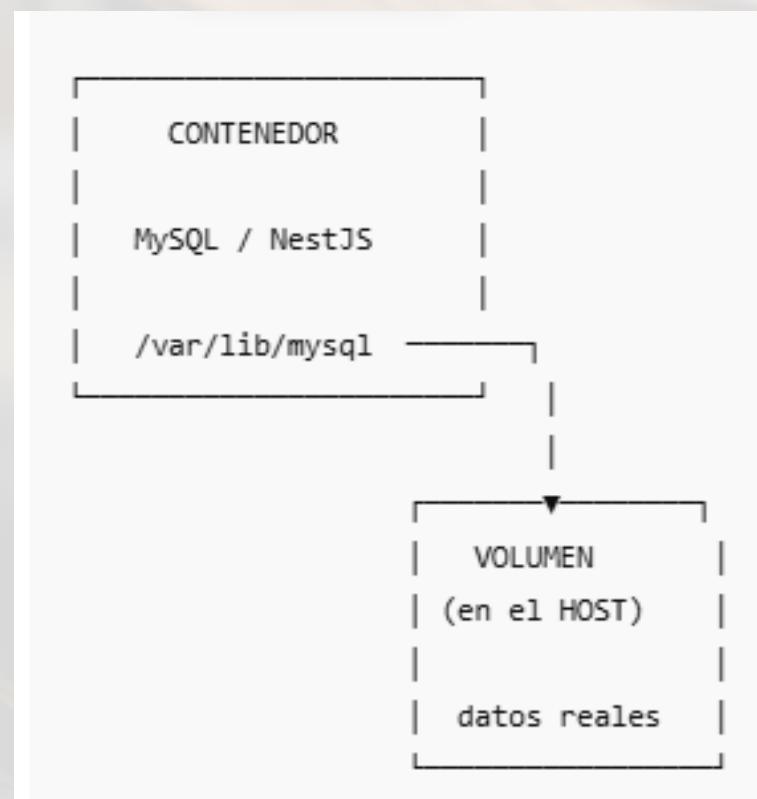
```
docker run -it mysql
```

MySQL guarda datos dentro del contenedor
Parás o borrás el contenedor → chau base de datos

✓ Solución: usar volúmenes

Idea clave

El contenedor usa una carpeta,
pero la carpeta está fuera del contenedor.



Qué pasa en la vida real:

El contenedor es descartable

El volumen es la caja fuerte

Si matás el contenedor:

✗ app se borra

✓ datos siguen vivos

El volumen es como un disco rígido externo enchufado al contenedor.

Docker



volúmenes

SIN volumen (mal)

yaml

```
services:  
  mysql:
```

```
    image: mysql:8
```

Resultado:

- MySQL guarda datos dentro del contenedor
- `docker down` → 😞 chau DB

docker-compose.yml

```
version: '3.8'  
  
services:  
  mysql:  
    image: mysql:8  
    container_name: mysql_db  
    environment:  
      MYSQL_ROOT_PASSWORD: root  
      MYSQL_DATABASE: app_db  
    ports:  
      - "3306:3306"  
    volumes:  
      - mysql_data:/var/lib/mysql  
  
  api:  
    build: .  
    container_name: nest_api  
    depends_on:  
      - mysql  
    ports:  
      - "3000:3000"  
  
volumes:  
  mysql_data:
```

Qué está pasando acá (mentalmente)

bash

```
NestJS —> MySQL —> /var/lib/mysql —> mysql_data (HOST)
```

- NestJS no sabe nada del volumen
- MySQL escribe en `/var/lib/mysql`
- Docker redirige eso al volumen
- Los datos sobreviven reinicios

Docker



volúmenes

Conceptos Clave de Volúmenes Docker

- **Persistencia:** Los datos no se pierden al detener o eliminar un contenedor.
- **Gestión:** Se administran con comandos docker volume (crear, ls, rm, inspect).
- **Ubicación:** Generalmente se almacenan en /var/lib/docker/volumes/ en Linux.
- **Uso:** Se montan al iniciar contenedores (-v o --mount en docker run).

Docker



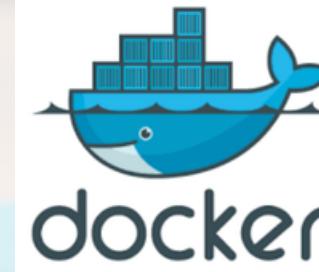
volúmenes

Tipos de Almacenamiento y Diferencias

Volúmenes Nombrados (Named Volumes): Son los mejores para persistir datos. Docker crea un nombre específico y lo gestiona, ideal para bases de datos o datos compartidos.

Bind Mounts: Montan una ruta específica de la máquina host en el contenedor. Son útiles para desarrollo, pero dependen de la estructura del host.

Docker



volúmenes

Comandos Básicos

- **Crear:** docker volume create mi_volumen
- **Listar:** docker volume ls
- **Eliminar:** docker volume rm mi_volumen
- **Uso en contenedor:** docker run -d --name mi_contenedor -v mi_volumen:/ruta/en/contenedor imagen

MUCHAS
GRACIAS

