2025-2026

DAW

Introducción a Docker (Contenedores)

Introducción

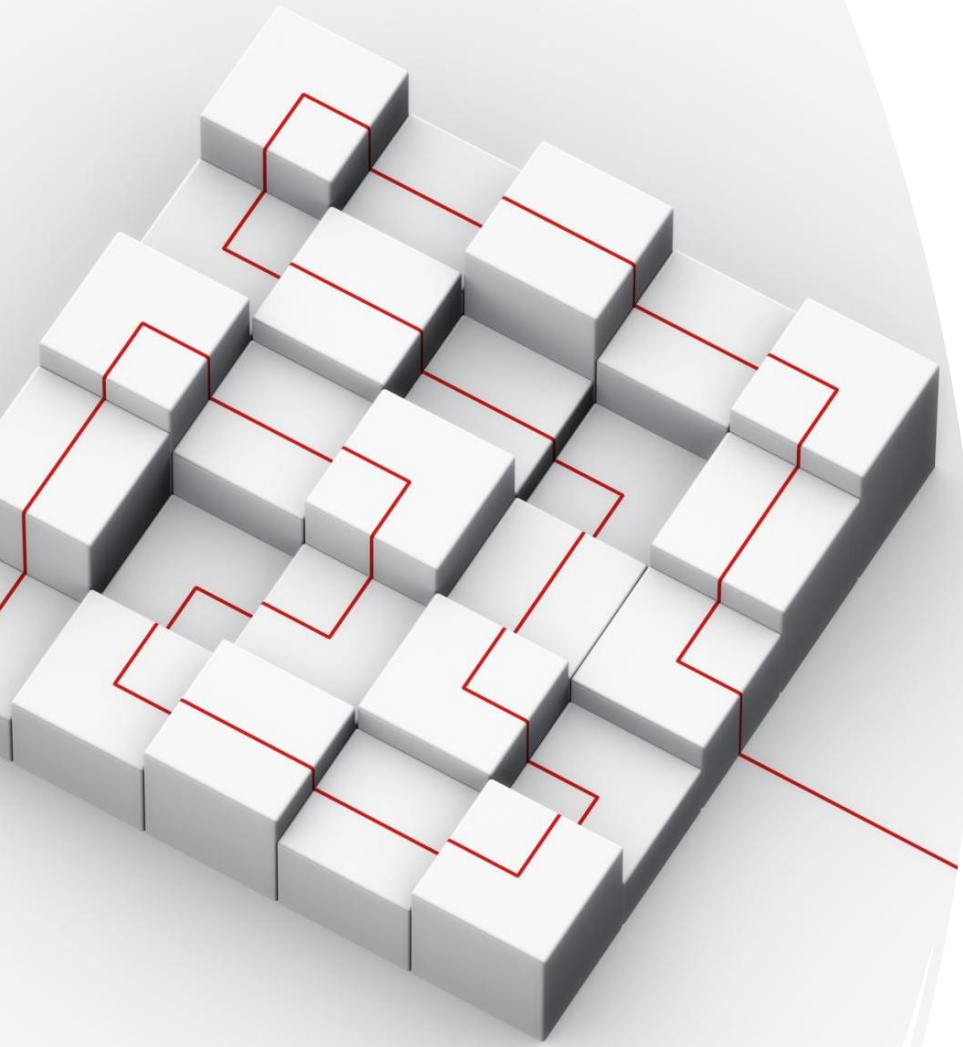
- Docker es una plataforma de contenedores que permite empaquetar, distribuir y ejecutar aplicaciones y sus dependencias en entornos aislados.
- Facilita la creación y el despliegue rápido y consistente de aplicaciones en diferentes entornos, desde el desarrollo hasta la producción.

Ventajas de Docker

- **Portabilidad**: Los contenedores Docker encapsulan aplicaciones y sus dependencias, lo que asegura que funcionen de manera idéntica en cualquier entorno.
- **Eficiencia**: Los contenedores comparten recursos del sistema operativo subyacente, lo que los hace más livianos y eficientes que las máquinas virtuales.
- **Aislamiento**: Los contenedores proporcionan un nivel de aislamiento que evita conflictos entre aplicaciones y sus dependencias.
- **Escalabilidad**: Docker facilita la escalabilidad horizontal, permitiendo la replicación y distribución de contenedores para manejar cargas variables.
- **Desarrollo Ágil**: Los equipos de desarrollo pueden trabajar en entornos consistentes y reproducibles, evitando problemas de "funciona en mi máquina".

Componentes Clave de Docker

1. **Imagen Docker**: Es un paquete ligero y autocontenido que incluye todo lo necesario para ejecutar una aplicación, como el código, las bibliotecas y las configuraciones. Las imágenes se utilizan para crear contenedores.
2. **Contenedor Docker**: Es una instancia en ejecución de una imagen Docker. Los contenedores son entornos aislados que comparten el mismo kernel del sistema operativo, pero tienen sus sistemas de archivos y procesos separados.
3. **Dockerfile**: Es un archivo de texto que contiene instrucciones para construir una imagen Docker. Especifica las capas y configuraciones que formarán la imagen final.
4. **Docker Hub**: Es un registro público de imágenes Docker donde los desarrolladores pueden compartir y descargar imágenes preconstruidas. También es posible utilizar registros privados.



Ciclo de Vida de una Aplicación Docker

1. **Creación de Imagen**: Se define una imagen usando un Dockerfile, que incluye instrucciones para instalar dependencias y configurar la aplicación.
2. **Construcción de la Imagen**: El Dockerfile se compila para crear una imagen Docker. Cada instrucción en el Dockerfile crea una nueva capa en la imagen.
3. **Creación de Contenedor**: A partir de una imagen, se crea un contenedor. Esto incluye la ejecución de la aplicación y la configuración de sus entornos.
4. **Ejecución y Pruebas**: La aplicación se ejecuta y se puede probar en el contenedor. Cualquier cambio en el código se reflejará en el contenedor.
5. **Despliegue**: Una vez que la aplicación está lista, se puede desplegar en diferentes entornos sin preocuparse por las diferencias en la infraestructura.

Casos de Uso de Docker

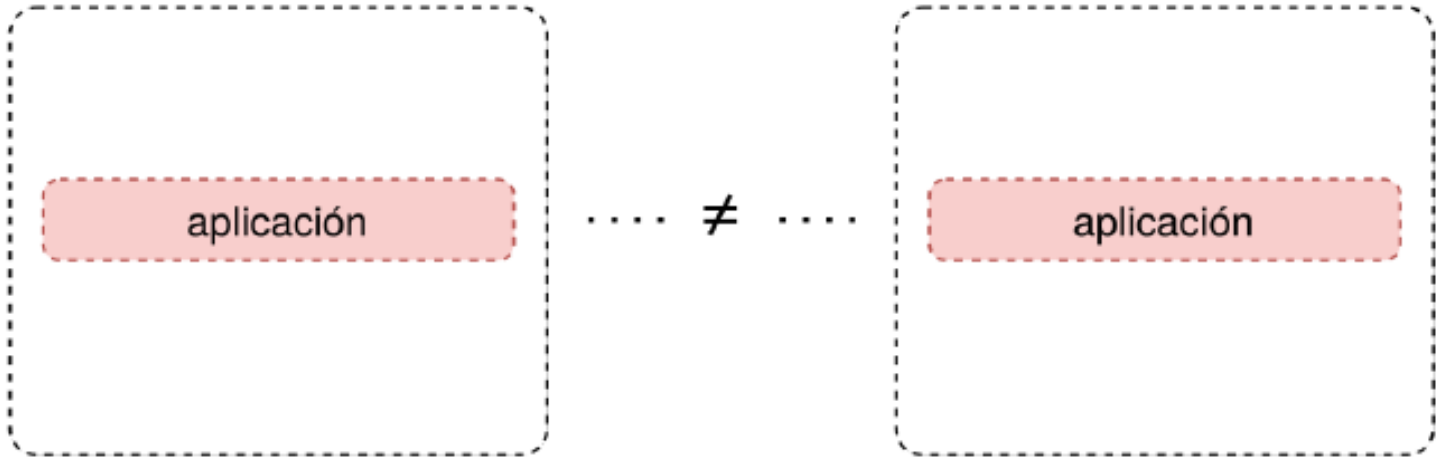
Desarrollo Local: Los desarrolladores pueden ejecutar aplicaciones en contenedores que replican los entornos de producción, evitando problemas de compatibilidad.

Entornos de Prueba y QA: Docker permite la creación de entornos de prueba aislados para probar nuevas características y realizar pruebas de regresión.

Despliegue en Producción: Docker simplifica el proceso de implementación y escalabilidad en servidores de producción.

Microservicios: Los contenedores son ideales para implementar microservicios, ya que permiten empaquetar y desplegar componentes individuales de una aplicación.


IN MY
MY LOCAL MACHINE
IT WORKS!



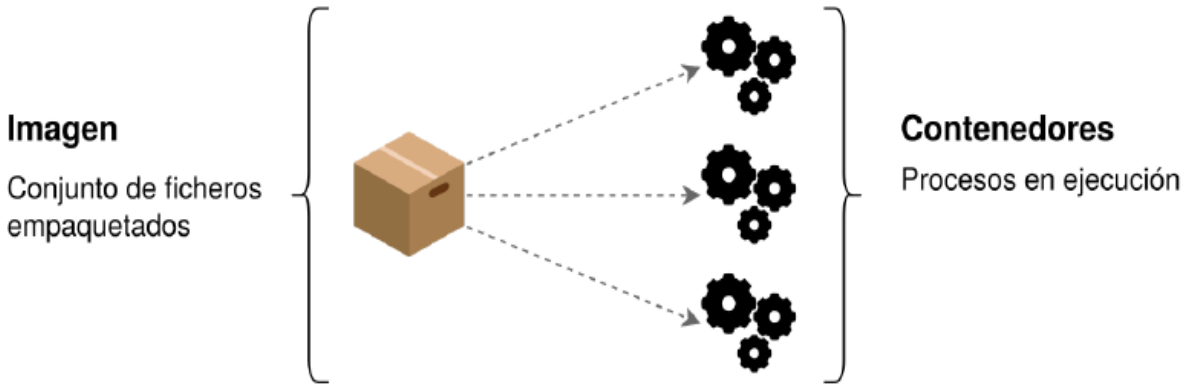
Comportamiento de una aplicación en entornos distintos

¿Por qué el código funciona bien en mi máquina, pero no en el servidor de producción?

- **Porque los entornos son distintos**

Imagen != Contenedor

- **Sin estado**



Relación entre imagen y contenedores

En la siguiente tabla se muestran las principales diferencias entre las imágenes y los contenedores.

Imagen	Contenedor
conjunto de ficheros empaquetados	es un proceso en ejecución
no tiene estado	tiene estado, p.ej iniciado, detenido, pausado
principales comandos: build, push y pull	principales comandos: run, stop, exec

¿Qué significa Docker?

- La palabra *docker* significa *estibador*. Un estibador es la persona que está en el puerto ayudando a bajar y subir mercancía de los barcos.

Ecosistema Docker

HUB

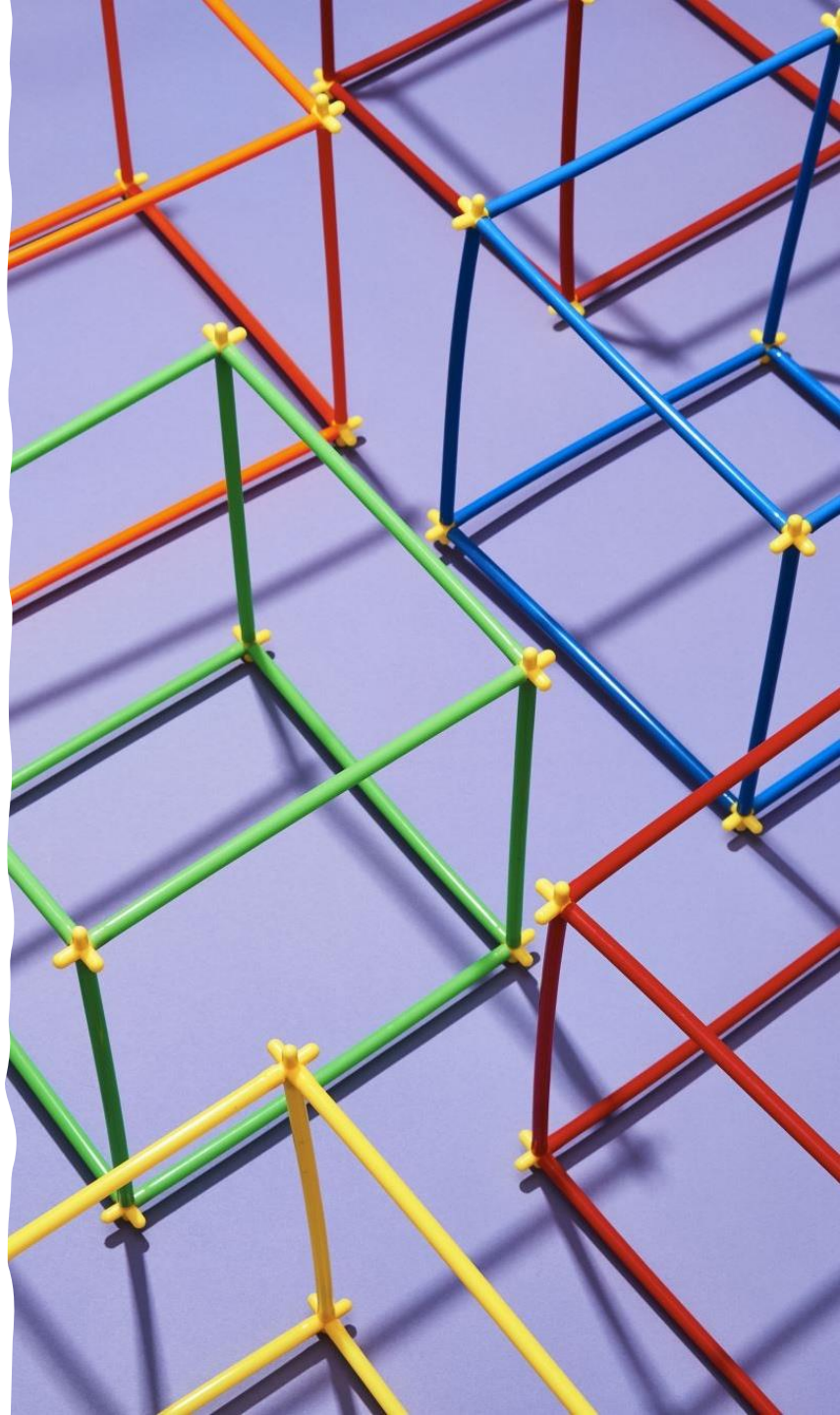
COMPOSE

REGISTRY

TRUST

Docker Hub

Plataforma donde se almacenan las imágenes de los contenedores



Docker Compose

- Permite gestionar múltiples contenedores al mismo tiempo. Es un orquestador de contenedores fácil e intuitivo de utilizar.

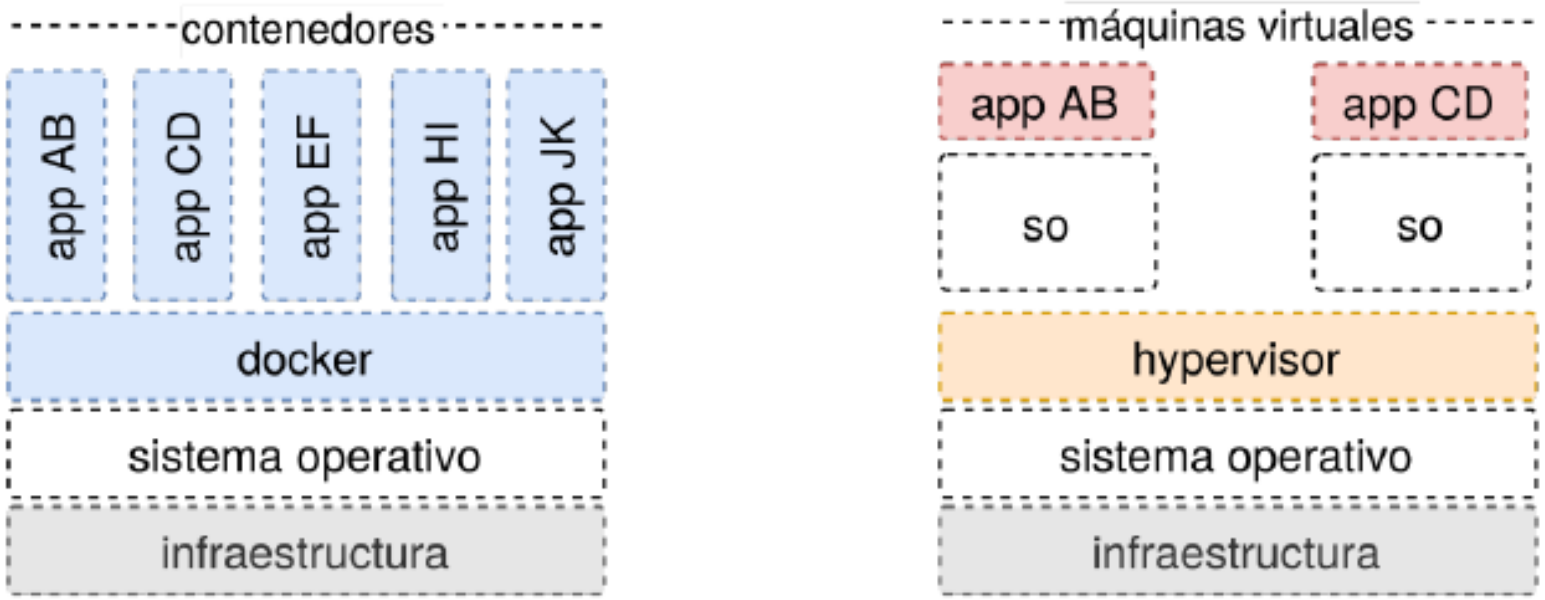
Docker Registry

- Permite crear registros privados de imágenes de contenedores. Puede instalar este istema en su plataforma y administrar directamente el almacenamiento de sus imágenes

Docker Trust

Tiene la responsabilidad de firmar digitalmente las imágenes de contenedores durante el proceso de construcción. Esta funcionalidad brinda la garantía de saber que está utilizando su imagen y que no ha sido interceptada o cambiada durante el proceso de distribución.

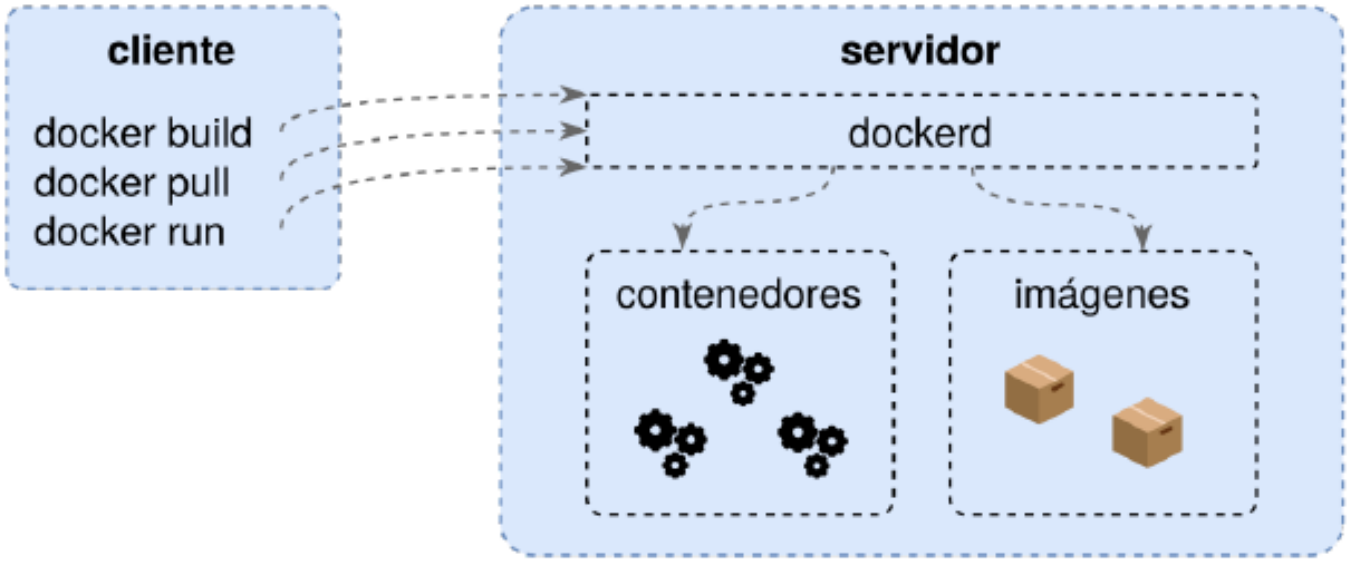
Contenedores != Máquinas Virtuales



Contenedores vs máquinas virtuales

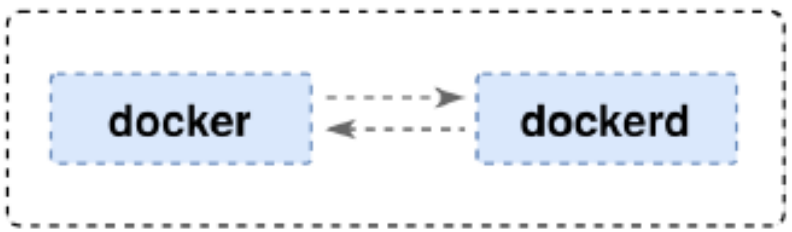
Arquitectura y componentes

- Cliente (Docker - Interfaz)
- Servidor (Dockerd – Proceso que realiza las Operaciones en la máquina)

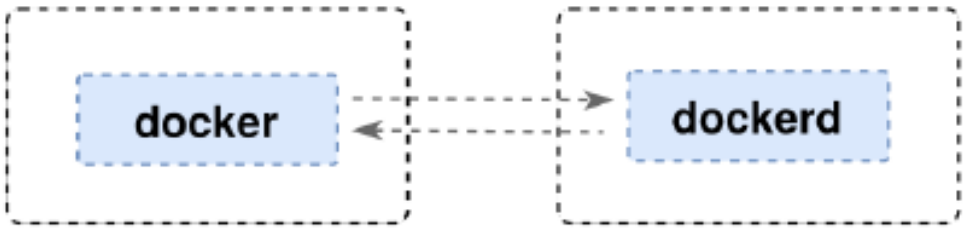


Vista de la arquitectura

Docker Engine (cliente-servidor)



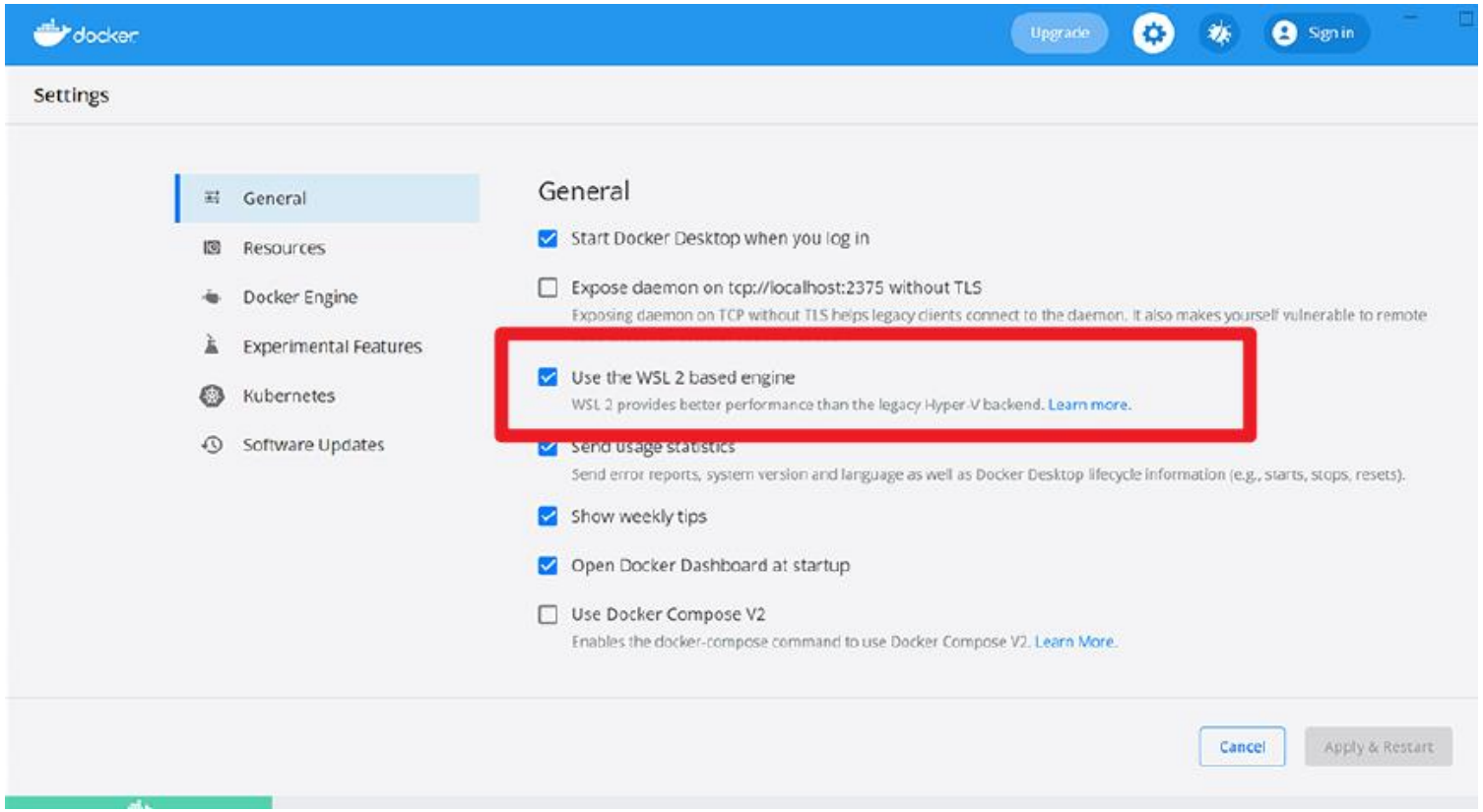
cliente y servidor
en el mismo ordenador



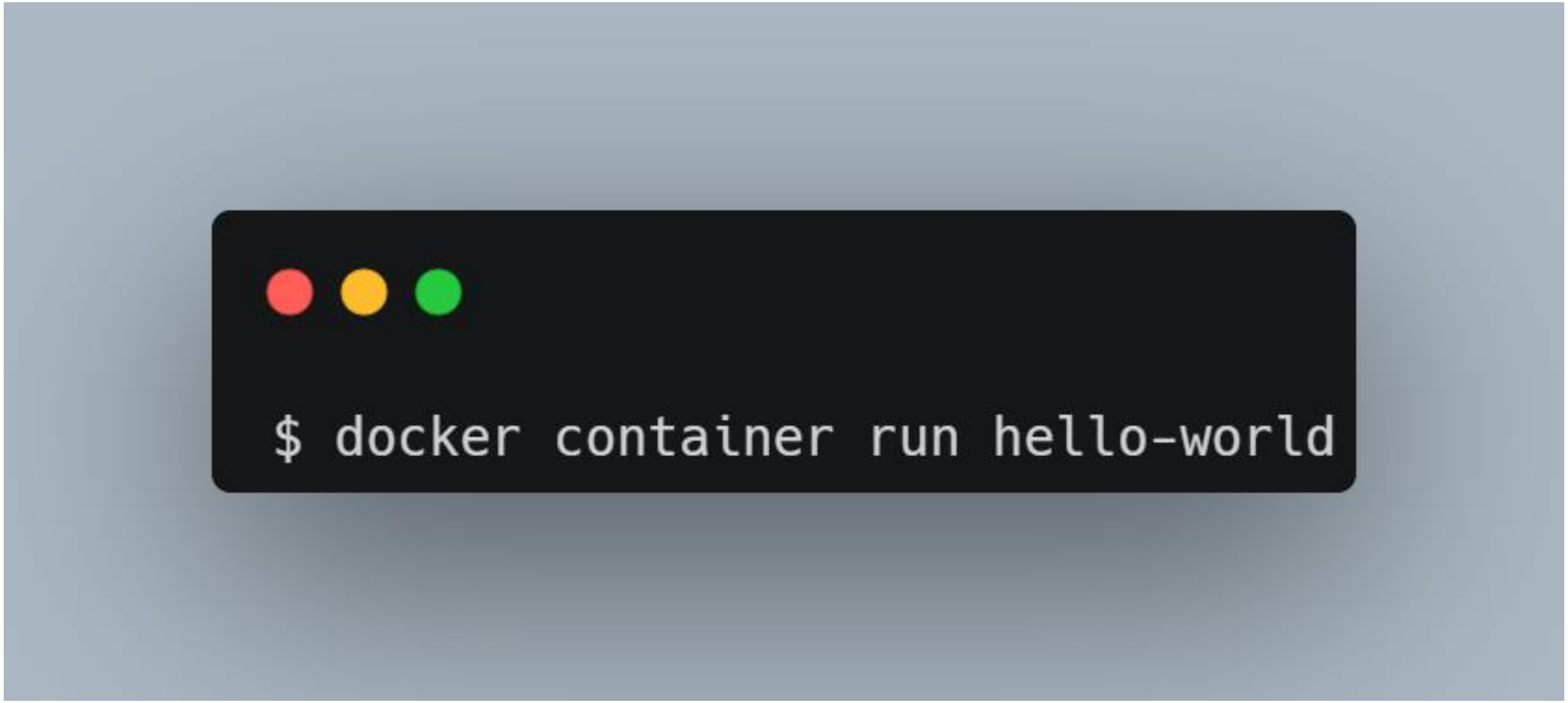
cliente y servidor
en diferentes ordenadores

Docker Desktop (Windows)

- WSL (Windows Subsystem for Linux) o Hyper-V



Hello World



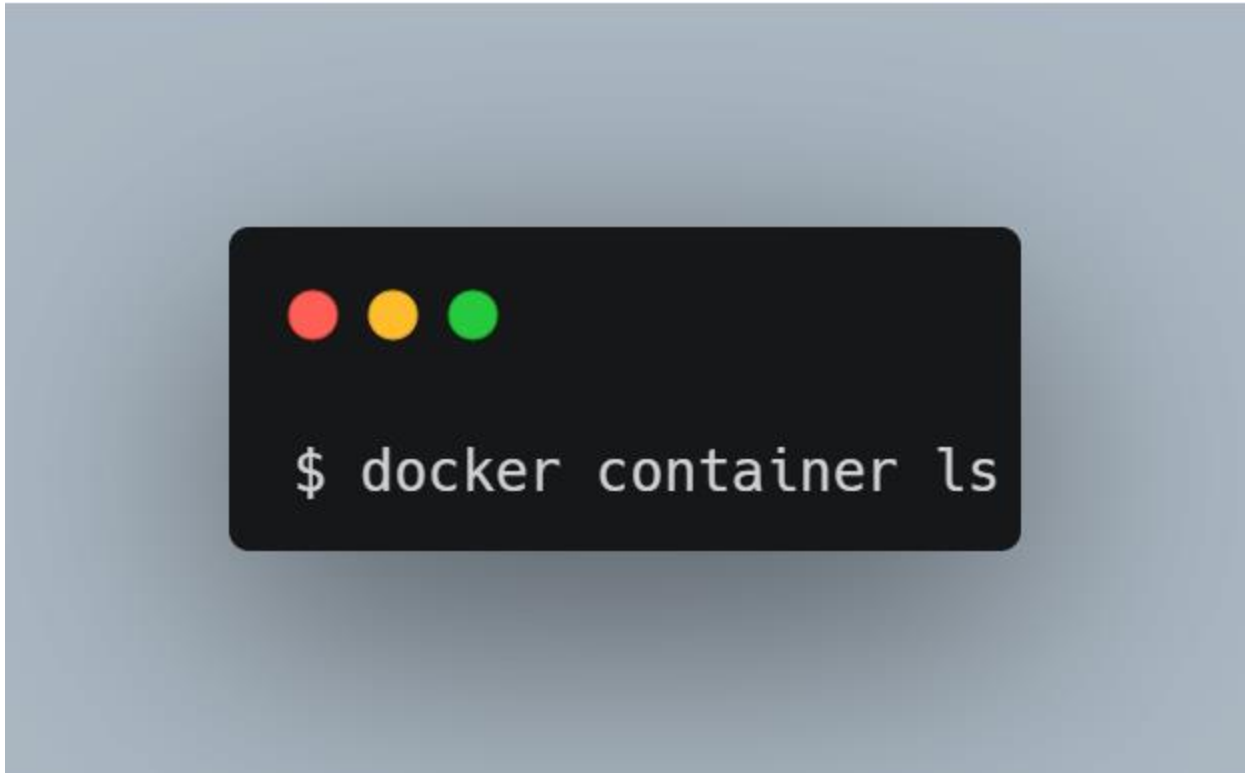
Comandos (Gestión)

- Usage: docker [OPTIONS] COMMAND

builder	Manage builds
buildx*	Docker Buildx (Docker Inc., v0.7.1)
compose*	Docker Compose (Docker Inc., v2.2.3)
config	Manage Docker configs
container	Manage containers
context	Manage contexts
image	Manage images

...

Listar contenedores en ejecución



Inspeccionar una imagen

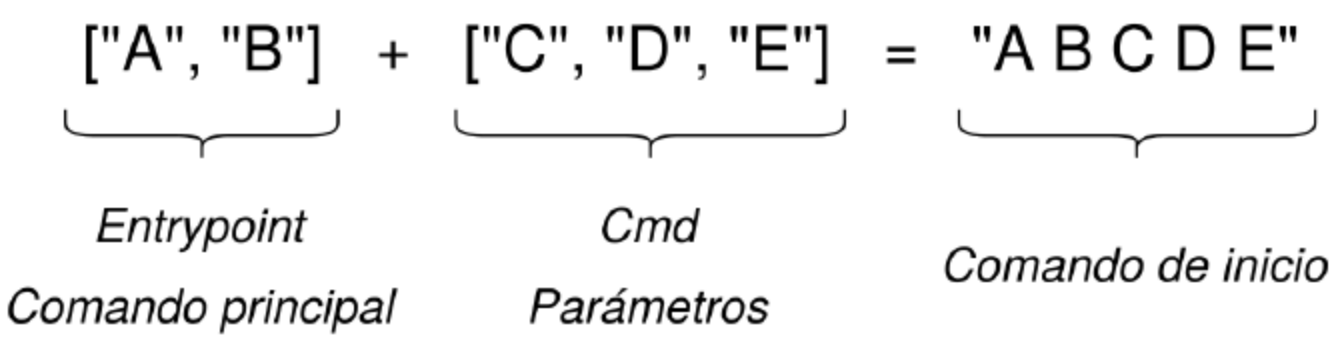
Ejemplo: Inspect (imagen Hello-World)


```
{
  "Id": "sha256:9c7a54a9a43cca047013b82af109fe963fde787f63f9e016fdc3384500c2823d",
  "RepoTags": [
    "hello-world:latest"
  ],
  "RepoDigests": [
    "hello-world@sha256:c79d06dfdfd3d3eb04cafd0dc2bacab0992ebc243e083cabe208bac4dd7759e0"
  ],
  "Parent": "",
  "Comment": "",
  "Created": "2023-05-04T17:37:03.872958712Z",
  "Container": "347ca68872ee924c4f9394b195dcadaf591d387a45d624225251efc6cb7a348e",
  "ContainerConfig": {
    "Hostname": "347ca68872ee",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
```

Ejemplo: Inspect (imagen Hello World II)


```
    "Labels": {}
},
"DockerVersion": "20.10.23",
"Author": "",
"Config": {
    "Hostname": "",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": [
        "/hello"
    ],
    "Image": "sha256:62a15619037f3c4fb4e6ba9bd224cba3540e393a55dc52f6bebe212ca7b5e1a7",
    "Volumes": null,
    "WorkingDir": "",
    "Entrypoint": null,
    "OnBuild": null,
    "Labels": null
},
```

Campo	Tipo	Descripción
Entrypoint	`array[string]`	Listado de argumentos utilizados como comando en el momento que se inicia el contenedor.
Cmd	`array[string]`	Parámetros pasados al comando de inicio *Entrypoint* del contenedor. Si el valor del *Entrypoint* está vacío, entonces el primer elemento de la lista del *Cmd* será utilizado como comando de ejecución.

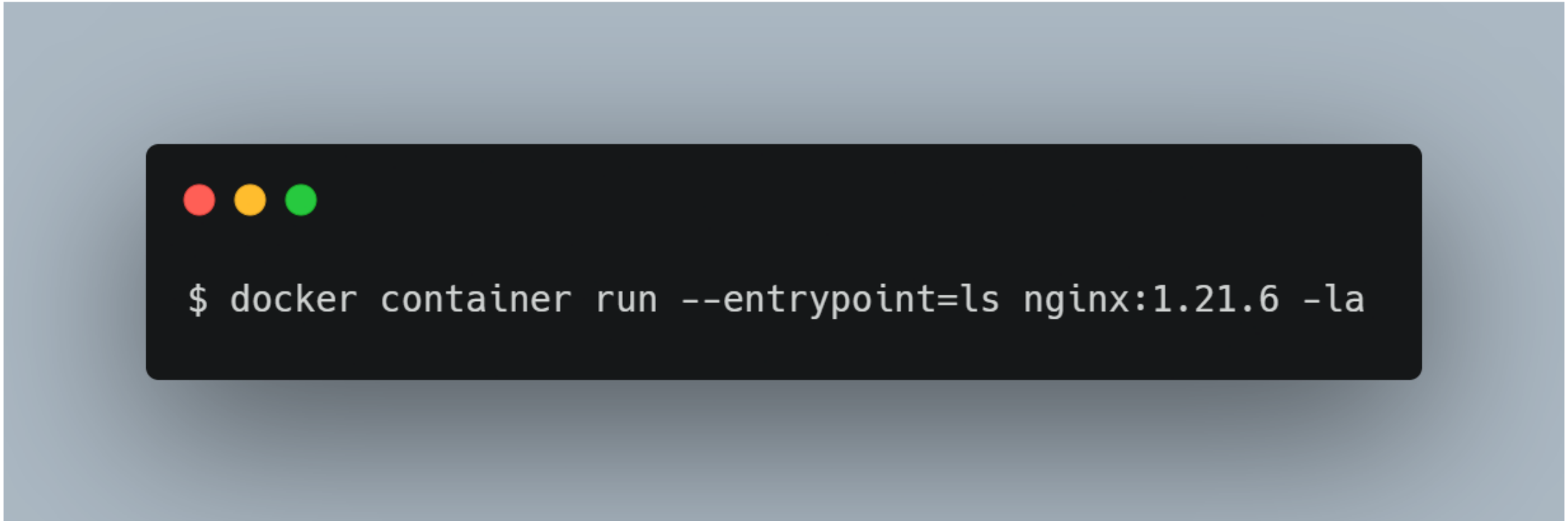


Comando de inicio

Ejemplo: NGINX

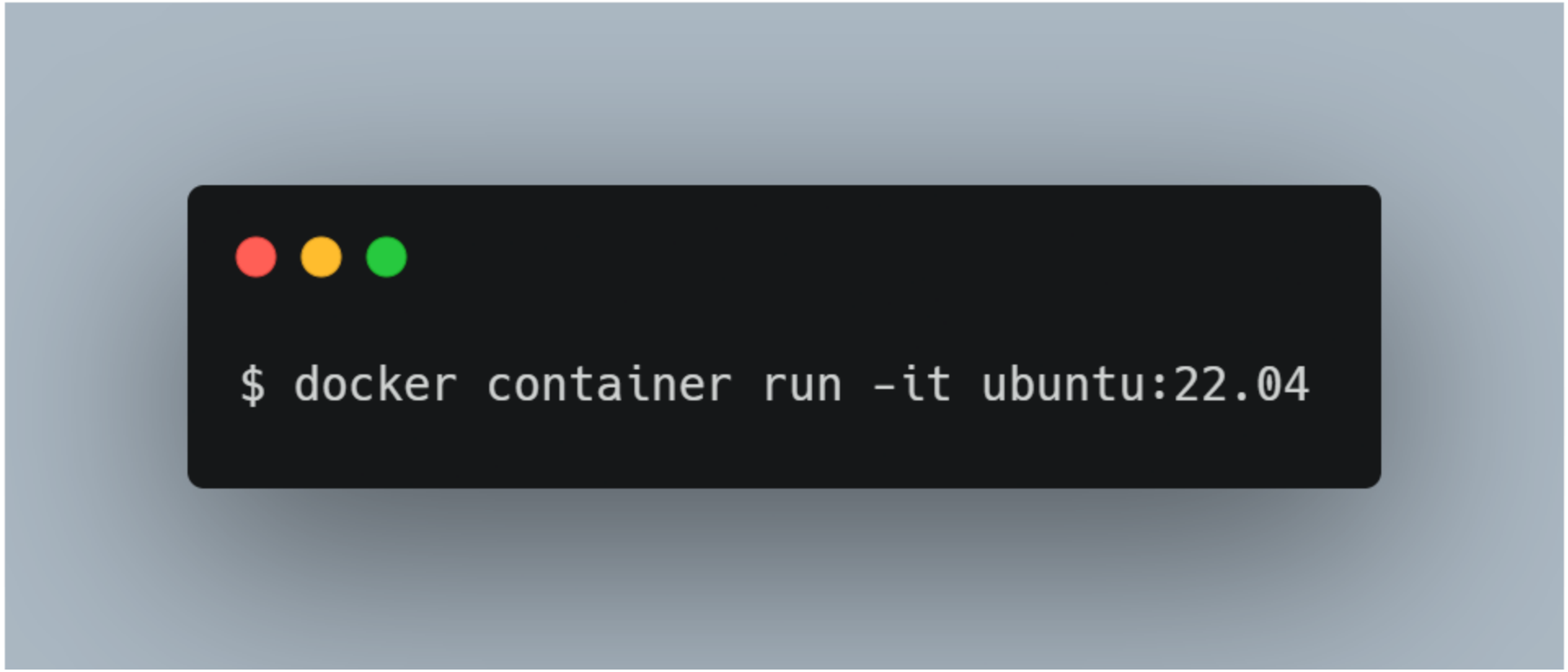

```
"ExposedPorts": {
  "80/tcp": {}
},
"Tty": false,
"OpenStdin": false,
"StdinOnce": false,
"Env": [
  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
  "NGINX_VERSION=1.25.2",
  "NJS_VERSION=0.8.0",
  "PKG_RELEASE=1~bookworm"
],
"Cmd": [
  "nginx",
  "-g",
  "daemon off;"
],
"Image": "sha256:e6ba5c3078b92a9de024ade55f756c38965b00cd1e44feff00f5b85f0c843e73",
"Volumes": null,
"WorkingDir": "",
"Entrypoint": [
  "/docker-entrypoint.sh"
],
"OnBuild": null,
"Labels": {
  "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"
},
```

Ejemplo: NGINX (Cambiar el valor del Entrypoint)



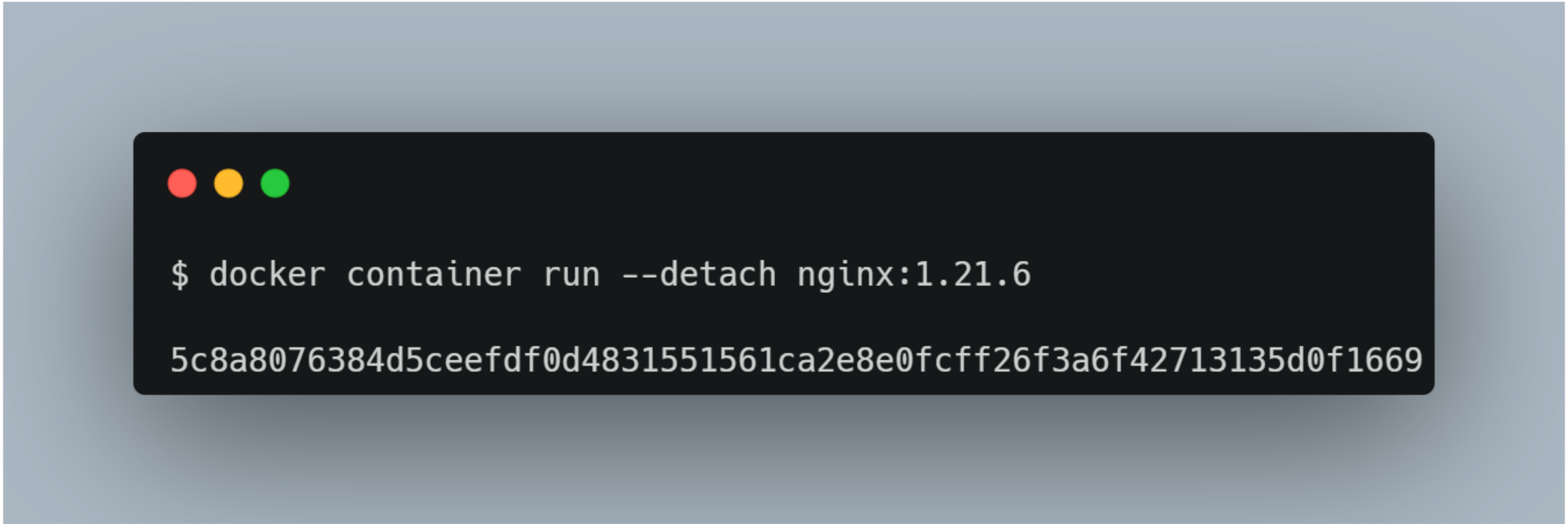
Iniciar contenedores interactivamente

- **--interactive o –i**: Permite mantener abierta la entrada de comandos
- **--tty o –t**: Permite interactuar con la consola a través de comandos

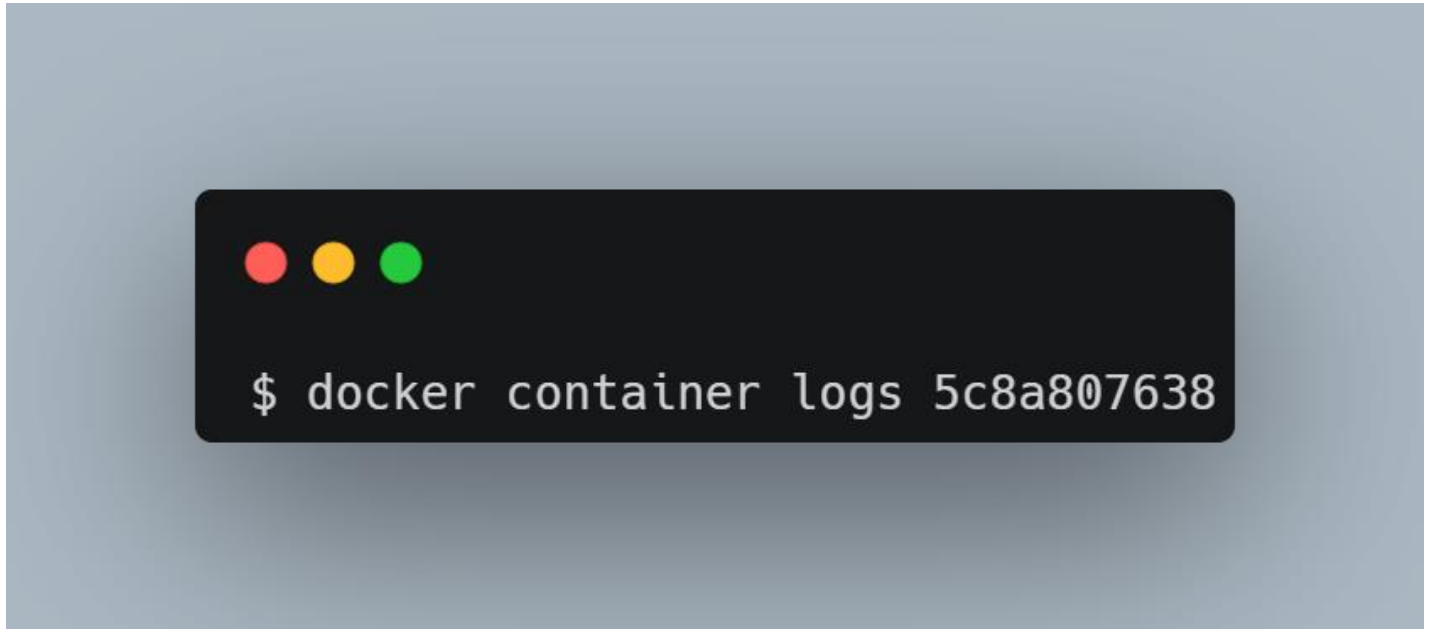


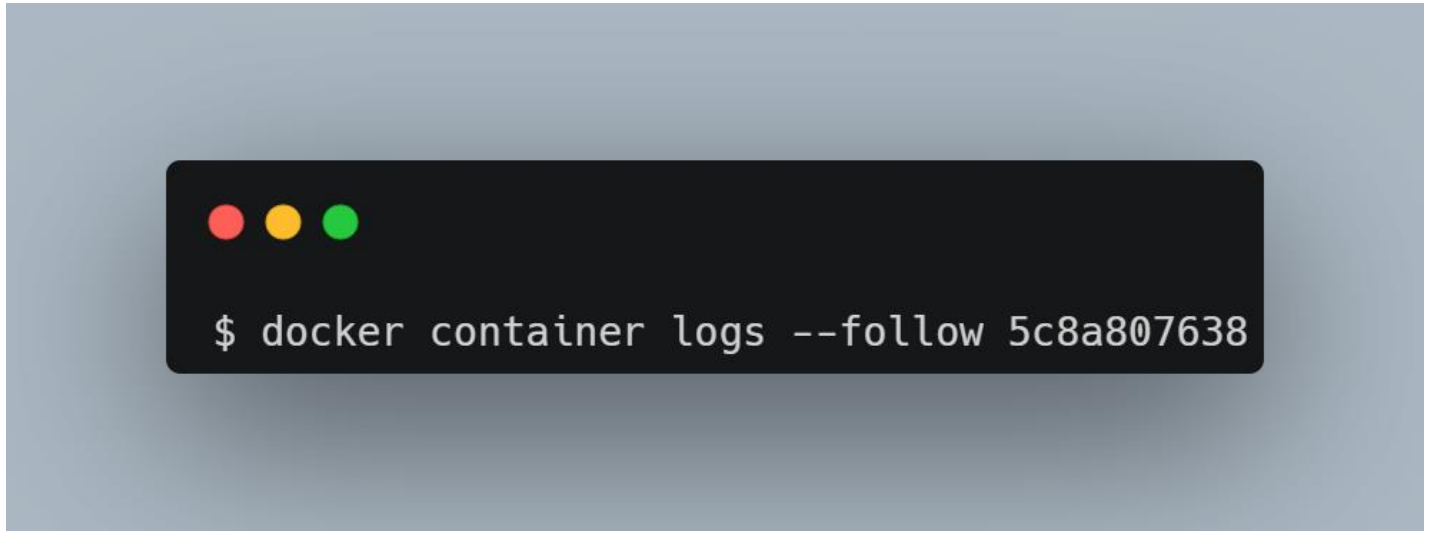
Iniciar múltiples servicios independientes

- **--detach o –d**
- Inicia el contenedor en un hilo independiente a la consola



Logs


$ docker container logs 5c8a807638

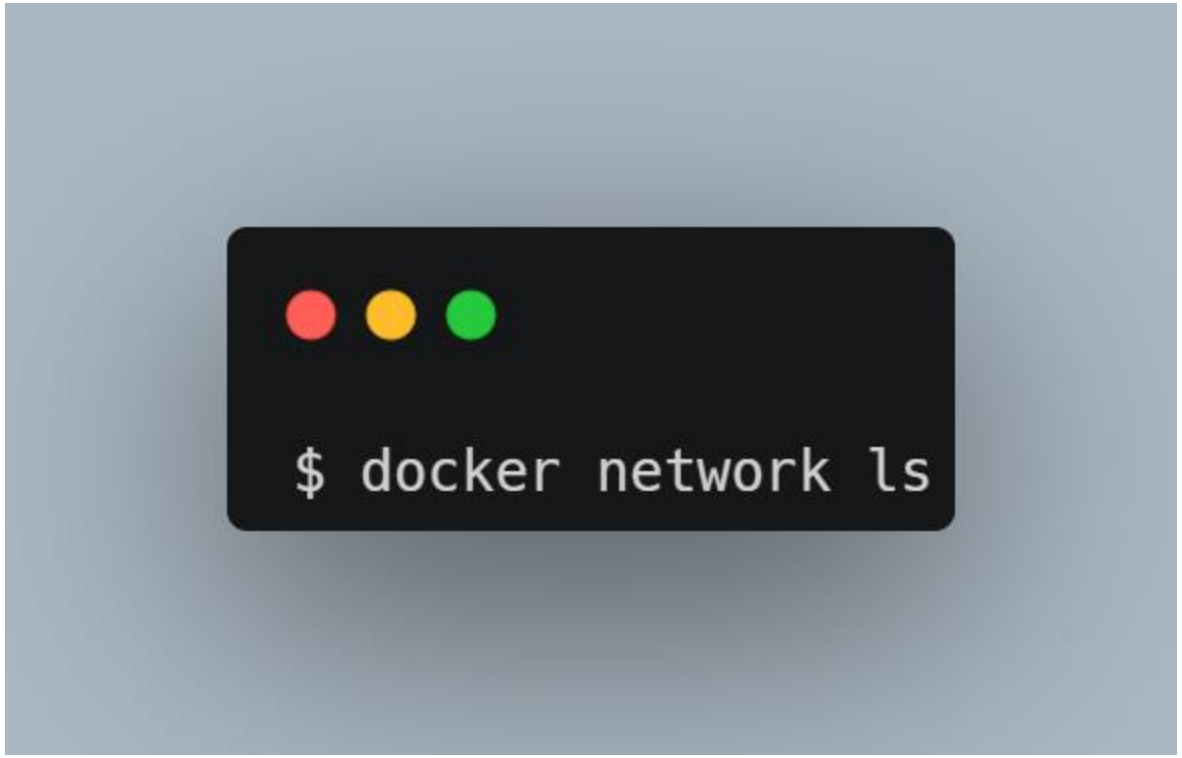

$ docker container logs --follow 5c8a807638

Consumo de recursos

```
$ docker container stats --no-stream --format 'table {{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemPerc}}'
```

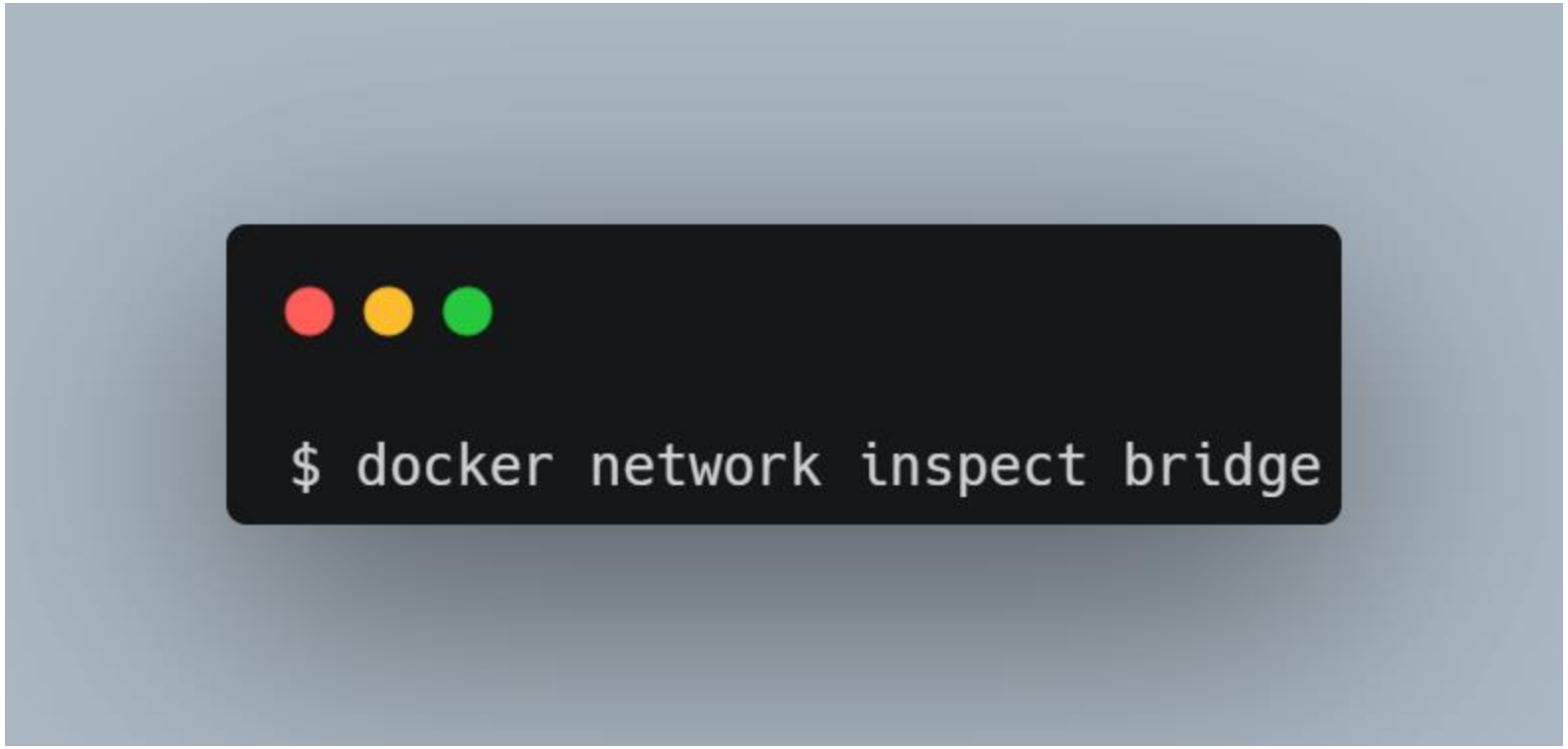
```
isuar docker container stats --no-stream --format 'table {{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemPerc}}'
CONTAINER ID   NAME           CPU %     MEM %
5c8a8076384d   zealous_bohr   0.00%     0.25%
isuar ~                                                    in pwsh at 12:12:50
```

Publicar y consumir servicios

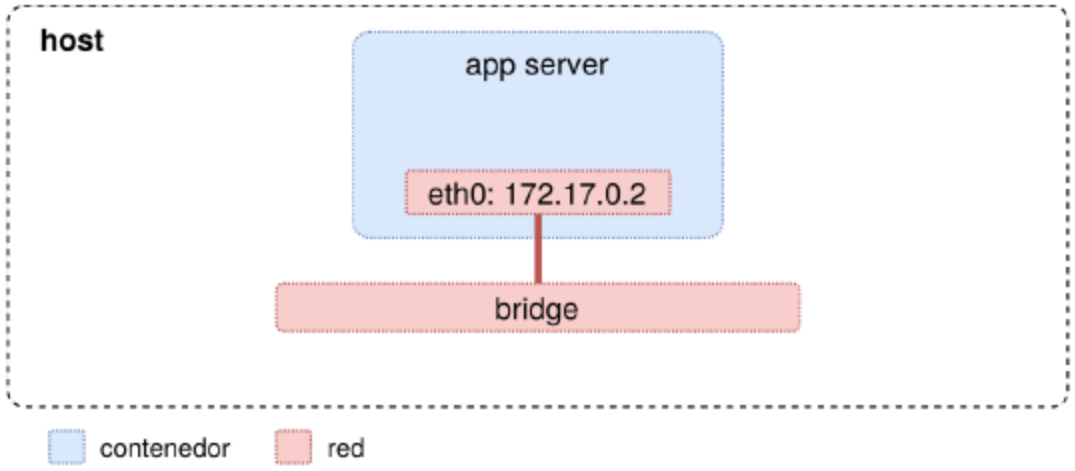


Publicar y consumir servicios (Controlador Bridge)

- Controlador por defecto
- Segmento de red para aislar los contenedores del resto



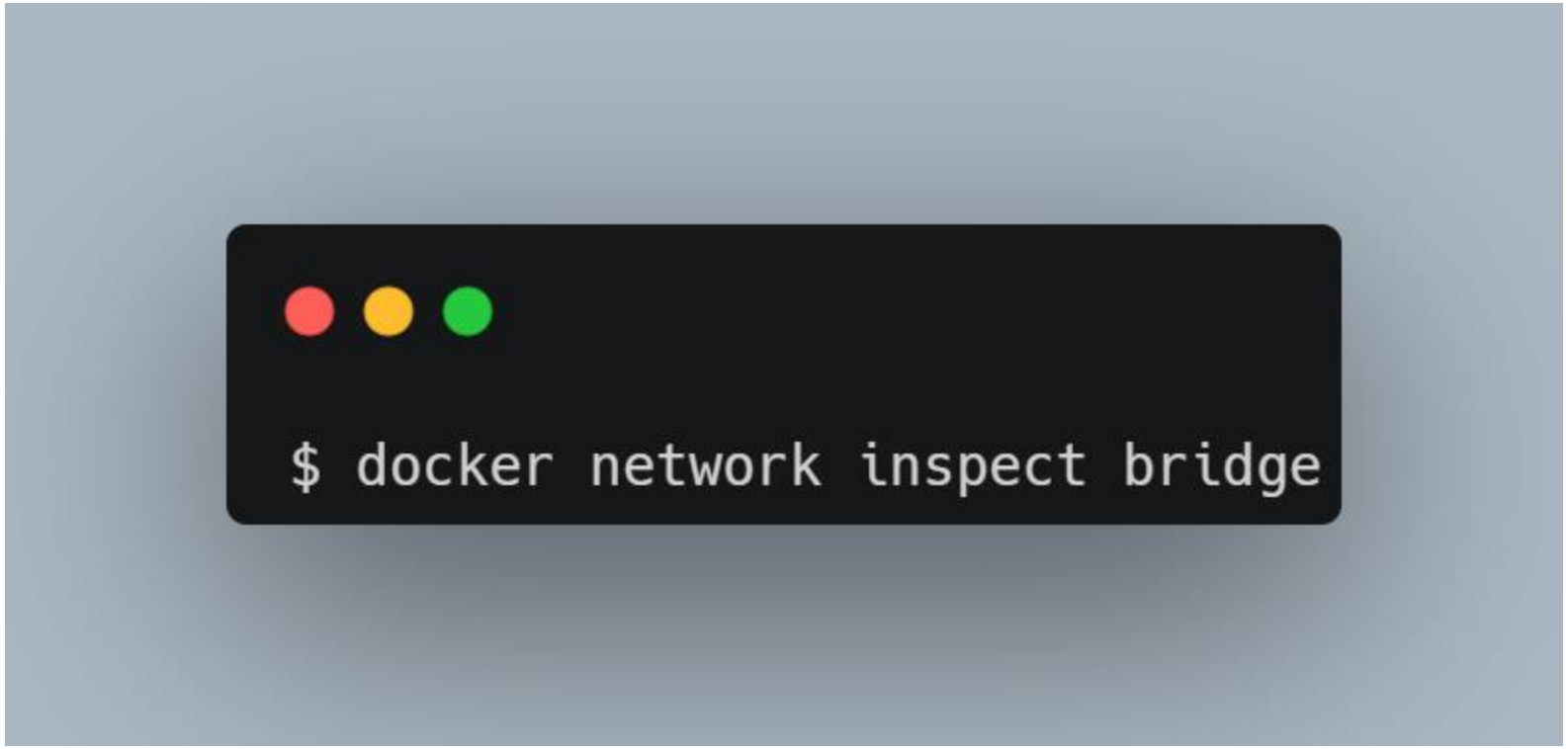
Publicar y consumir servicios (Controlador Bridge)



IP del contenedor en la red bridge

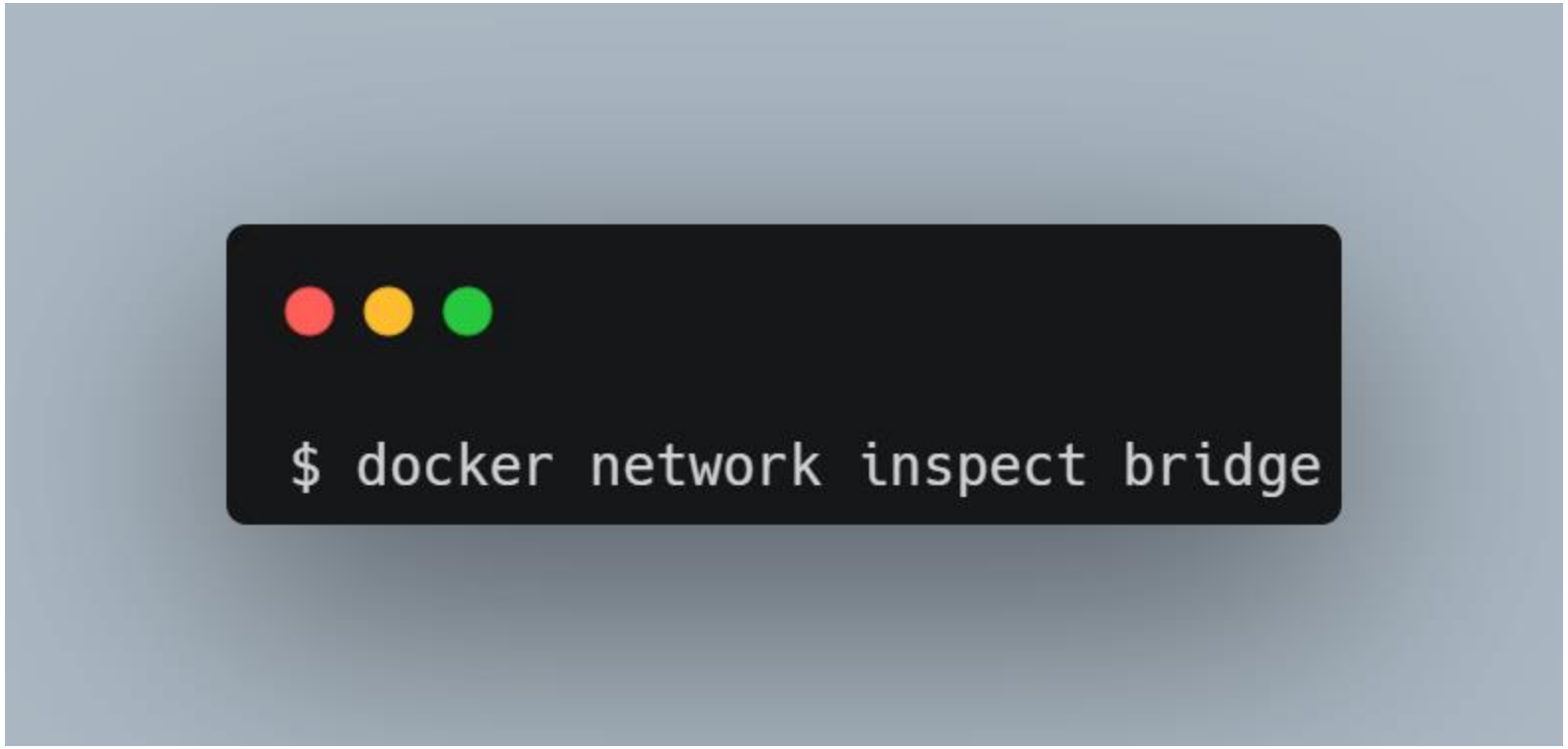
Publicar y consumir servicios (Controlador host)

- No aisla al contenedor y no asigna direccionamiento IP
- Comparte espacio de red de la máquina que lo inicia



Publicar y consumir servicios (Controlador null)

- Se usa para eliminar todas las configuraciones de red del contenedor
- Comparte espacio de red de la máquina que lo inicia

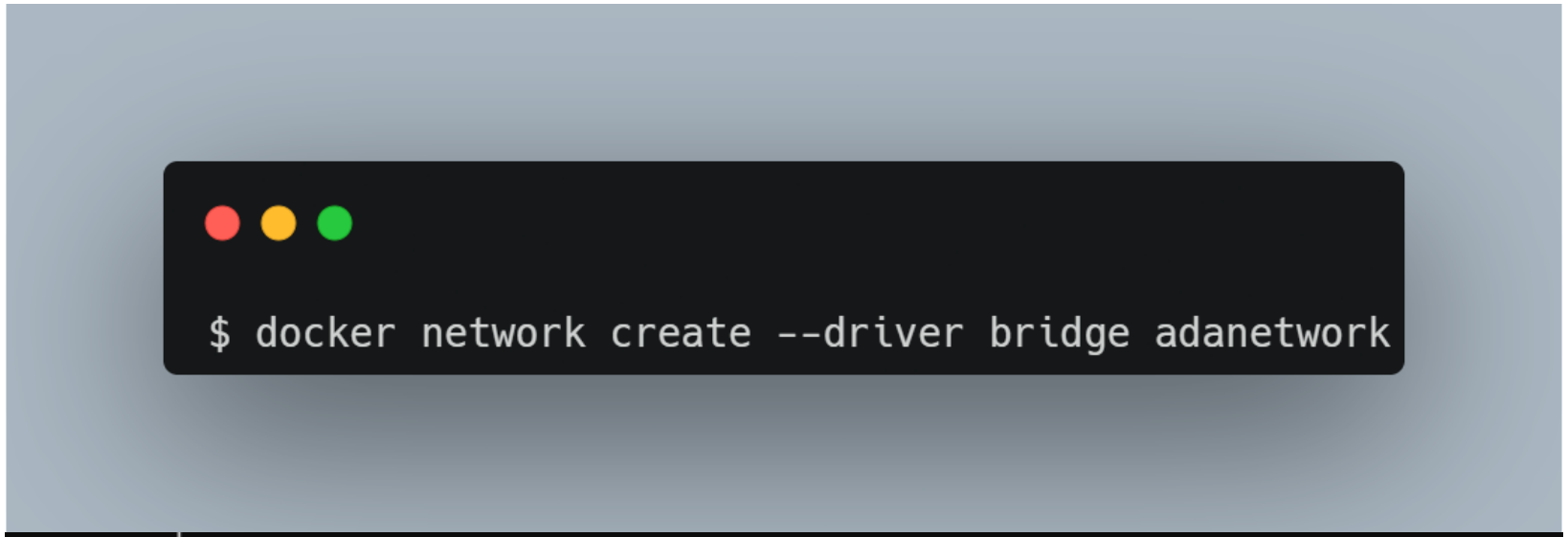


Conocer la IP de un contenedor

```
$ docker container inspect 5c8a807638 --format '{{ json .NetworkSettings.Networks.bridge.IPAddress }}'
```

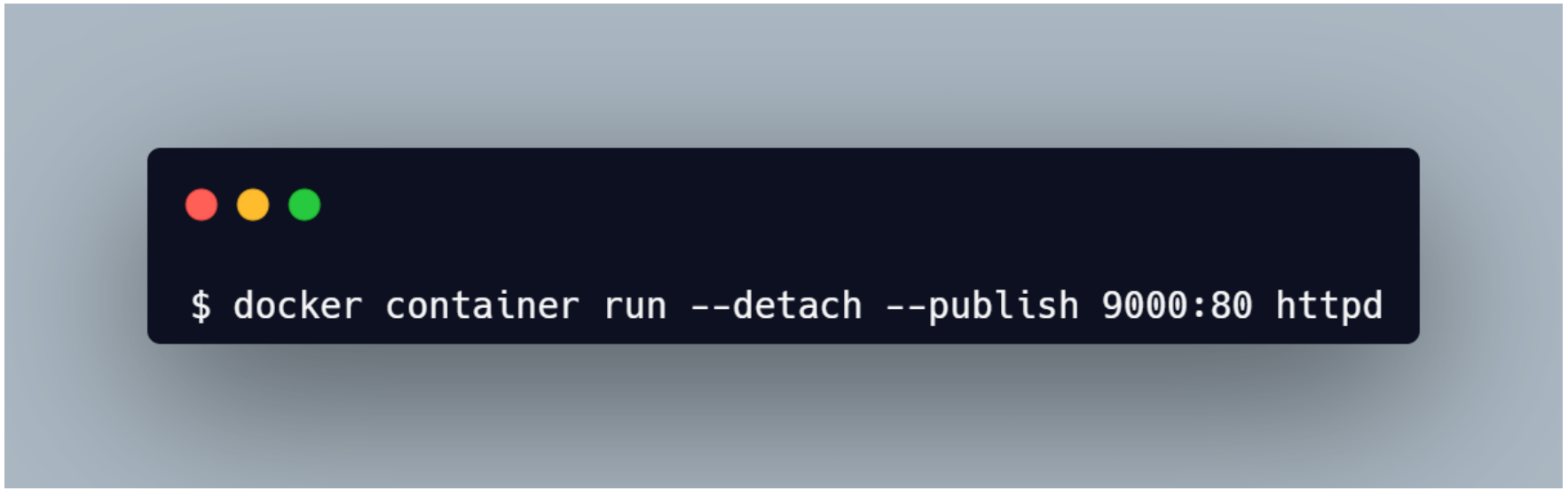
```
isuar docker container inspect 5c8a807638 --format '{{ json .NetworkSettings.Networks.bridge.IPAddress }}'
"172.17.0.2"
isuar                                                                    in pwsh at 1
```

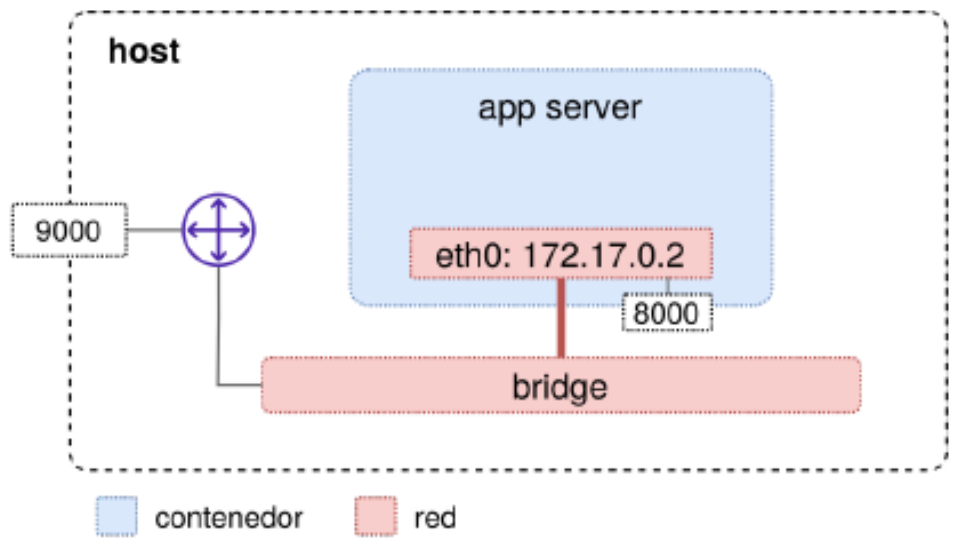
Crear una red




```
isuar docker network inspect adanetwork --format='{{ json .IPAM }}' | jq
{
  "Driver": "default",
  "Options": {},
  "Config": [
    {
      "Subnet": "172.21.0.0/16",
      "Gateway": "172.21.0.1"
    }
  ]
}
```

Publicación de puertos




host
app server
9000
eth0: 172.17.0.2
8000
bridge
contenedor
red

Persistencia de datos

¿Por qué pierdo los datos?

Persistencia de datos

- Todos los datos y configuraciones generados por el contenedor se almacenan en su estructura de ficheros.
- Al ser generados por el contenedor significa que no forman parte de los ficheros que existen en la imagen, por ende, si reinicia el contenedor pierde todos estos nuevos ficheros.

Ejemplo REDIS

1. Crear una red bridge (la podemos llamar “redis”)
   - docker network create --driver bridge redis

2. Crear el contenedor servidor
   - docker container run --detach --network redis redis:6.2.6

3. Crear contenedor cliente redis
   - docker container run --network redis -it redis:6.2.6 redis-cli -h“$CONTAINER_ID DEL SERVIDOR”
     - Comando del último container: *docker container ls --latest –quiet*

4. *set myKey myValue*
   get myKey

Diagrama de REDIS

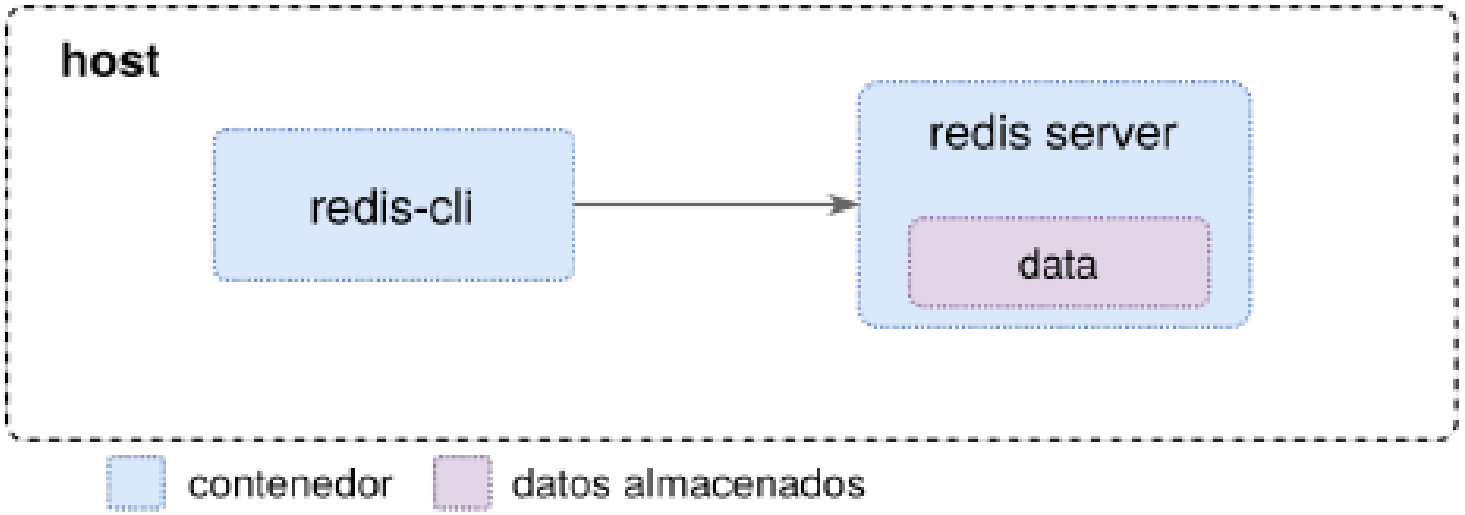


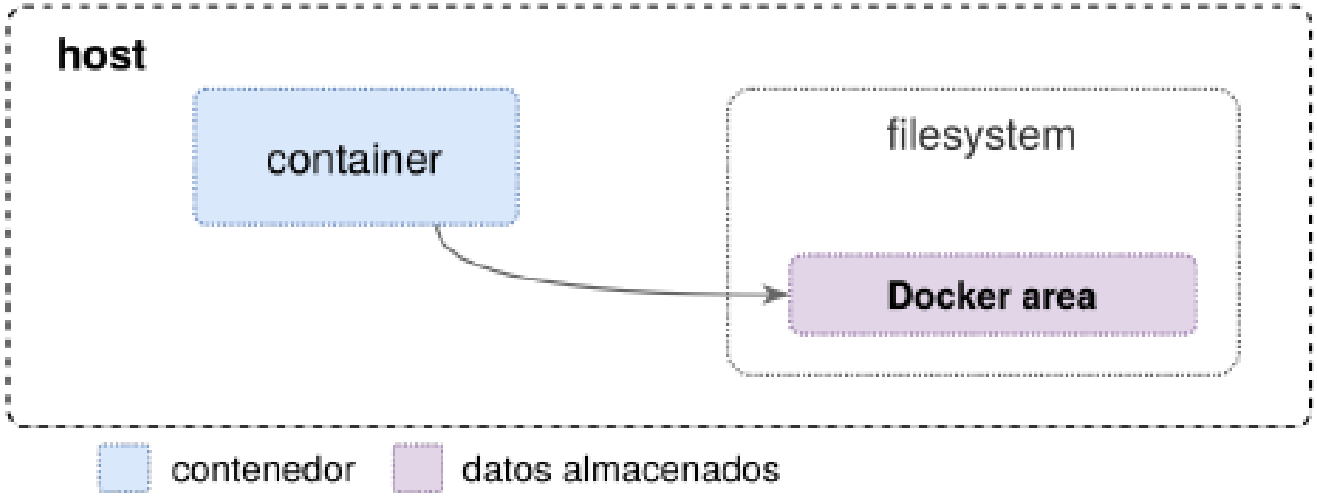
Diagrama del servicio de Redis

Persistencia de datos en volúmenes

- Los volúmenes están fuera del ciclo de vida de los contenedores, lo que significa que se puede detener y eliminar los contenedores sin preocuparse de la pérdida de la información. De hecho, se puede crear el volumen antes de iniciar el contenedor, e incorporar los ficheros necesarios.

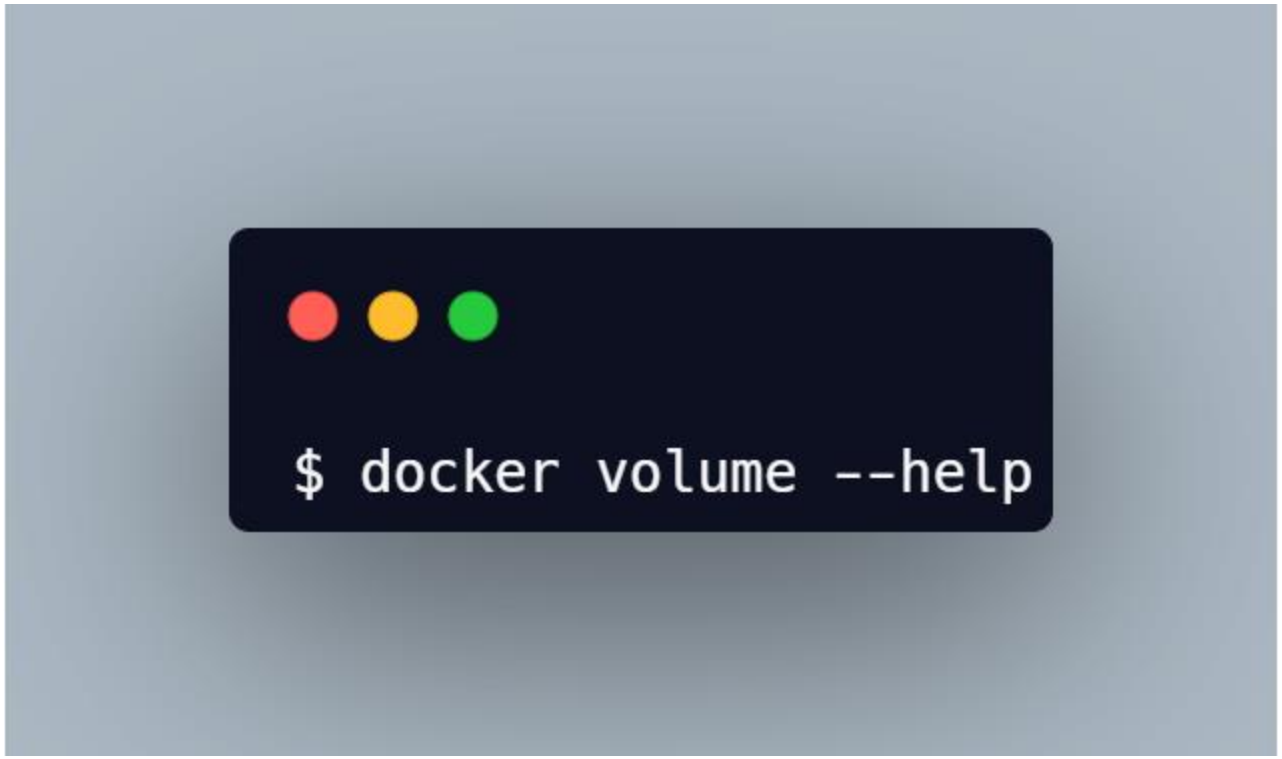
- Los beneficios de contar volúmenes son:
  - Poder detener el contenedor sin perder la información.
  - Gestionar el espacio en disco directamente desde los comandos de Docker.
  - Realizar copias y restauraciones de la información de forma fácil y rápida.
  - Compartir un mismo volumen con múltiples contenedores.

Persistencia de datos en volúmenes

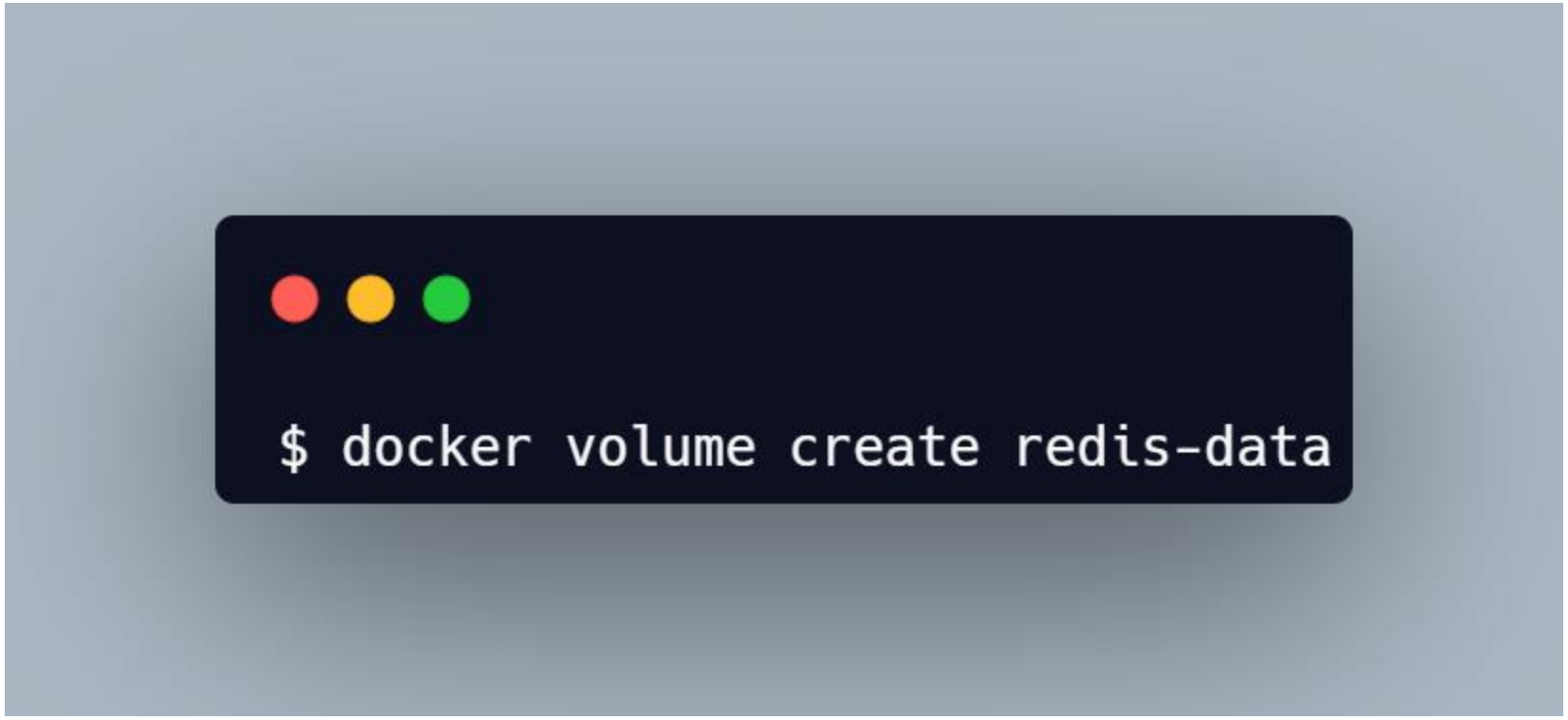


Representación de los volúmenes en Docker

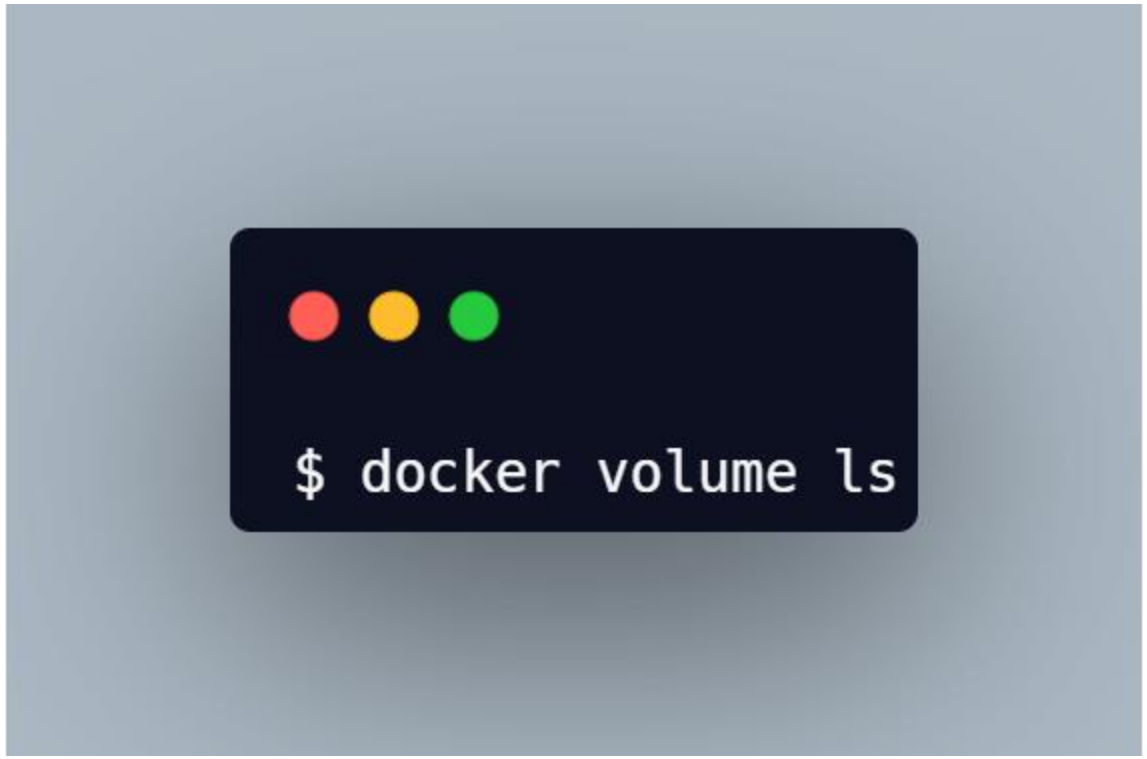
Comandos para gestionar los volúmenes



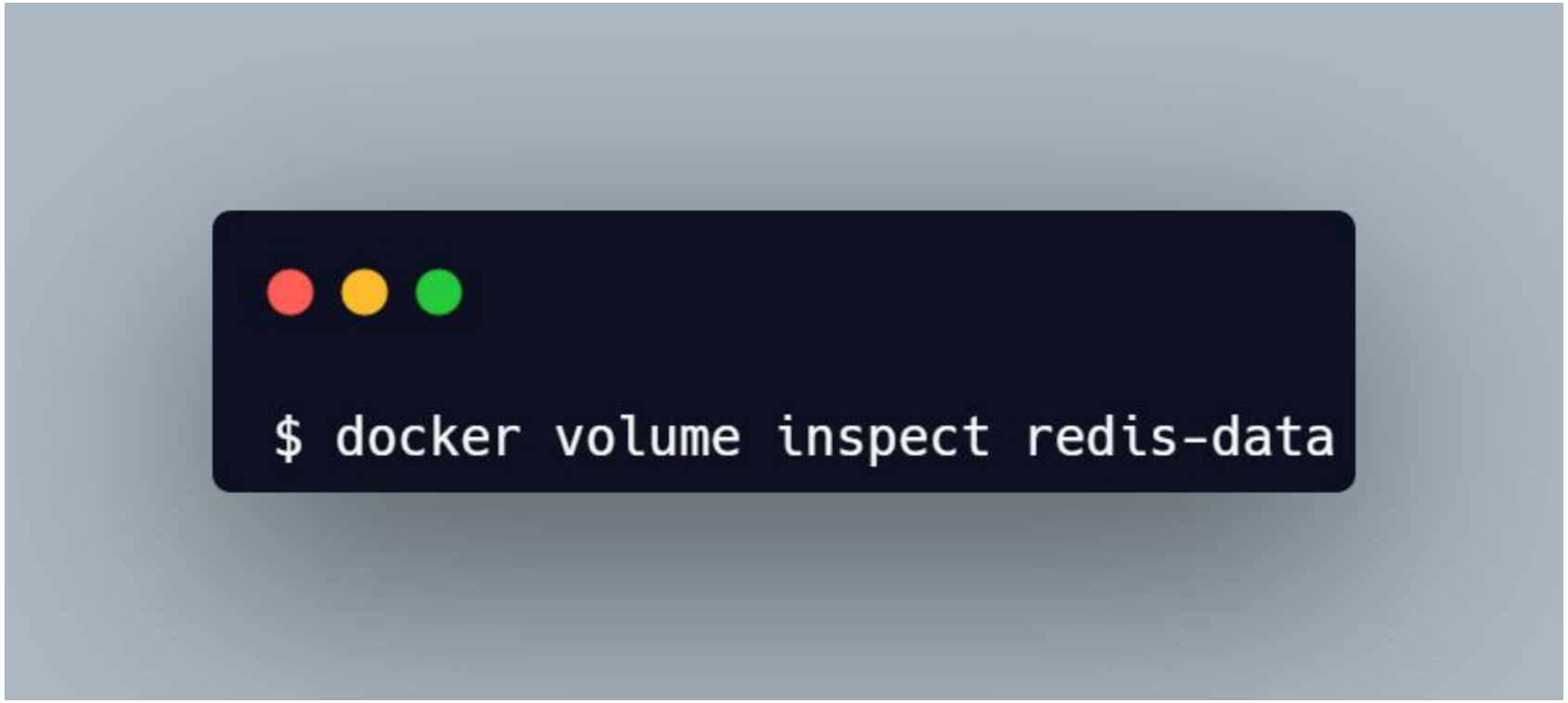
Crear volumen

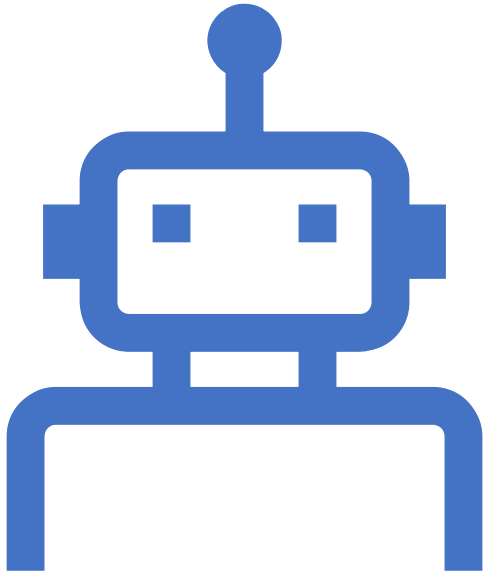


Listar los volúmenes



Inspeccionar un volumen





Conclusiones

- Docker es una herramienta esencial en el desarrollo y despliegue de aplicaciones modernas.
- Ofrece portabilidad, eficiencia, aislamiento y escalabilidad.
- Los contenedores Docker facilitan la creación de entornos coherentes desde el desarrollo hasta la producción.
- Su adopción puede mejorar la eficiencia del desarrollo y la operación de aplicaciones.