

Bloque 1 -Tema 4. Docker.

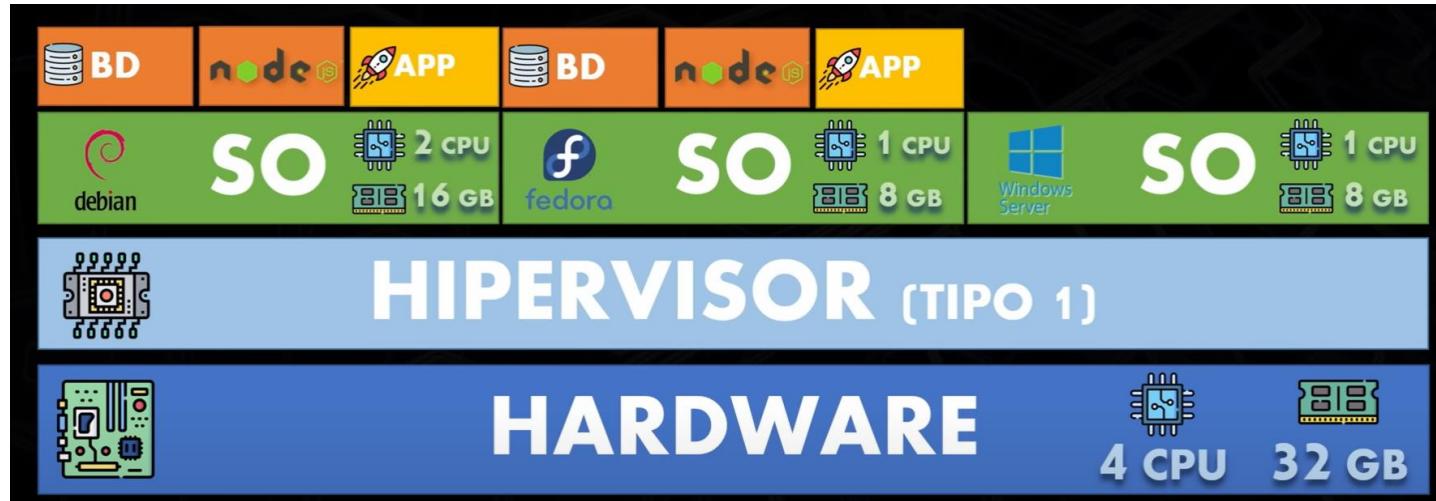
Contenedores para la Gestión, Orquestación y
Procesamiento Escalable de Datos en entornos
Big Data y Data Engineering

Servidor físico vs máquina virtual



Servidor
Físico

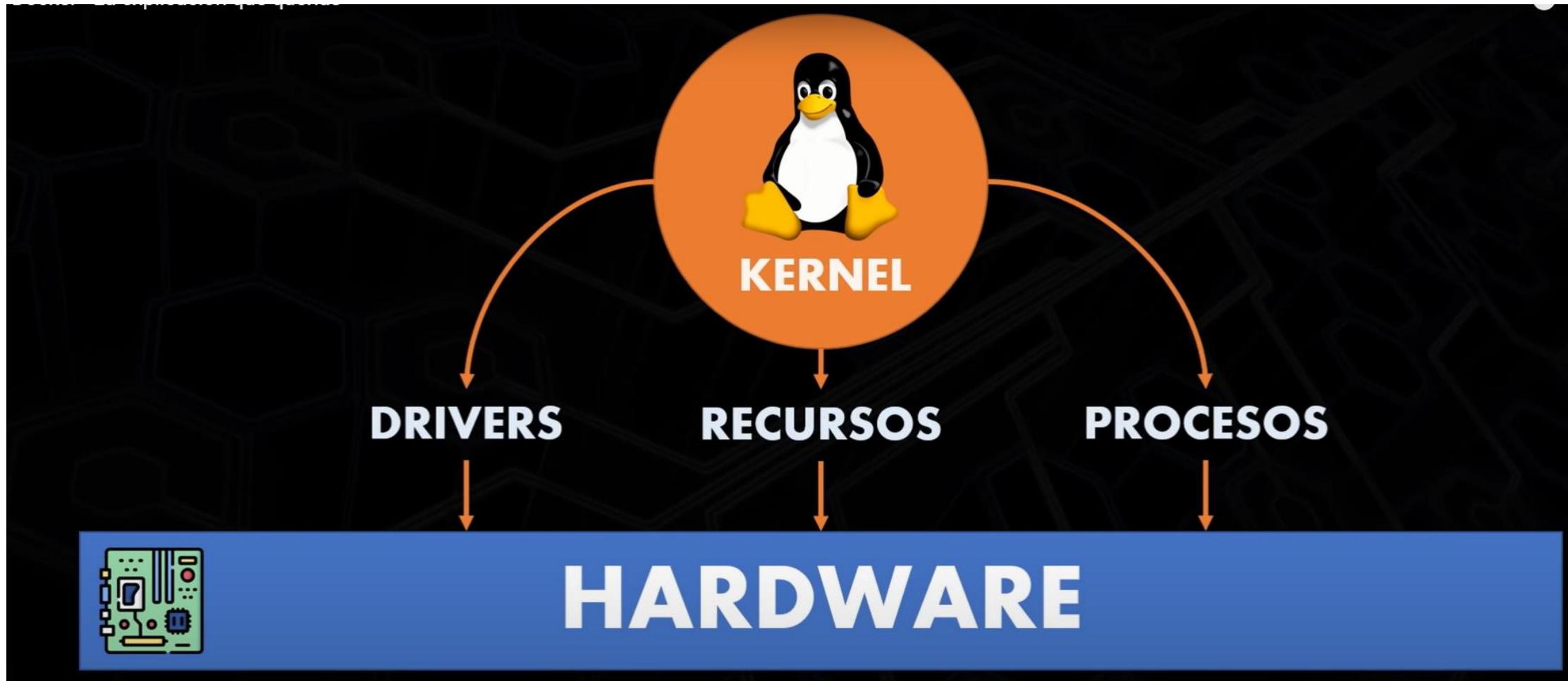
Máquina
Virtual



Al aumentar el nro de máquinas virtuales se incrementa el costo en hardware.

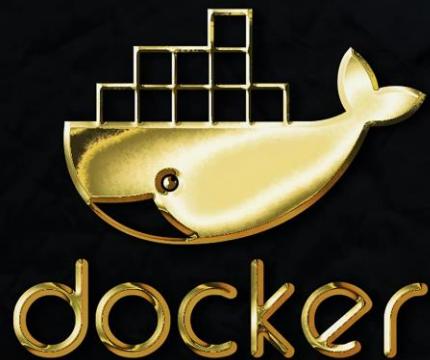


En vez de tener distintos kernels ¿Por qué no utilizar un único kernel a partir del cual se gestionen todos los servicios?



En vez de tener distintos kernels
¿Por qué no utilizar un único kernel a partir del cual se
gestionen todos los servicios?





Docker es un conjunto de herramientas que se utiliza para empaquetar aplicaciones con todas sus herramientas y bibliotecas necesarias en "contenedores" ordenados y portátiles. Estos contenedores se ejecutan en cualquier lugar, aislados unos de otros, lo que garantiza un rendimiento uniforme y eficiente.

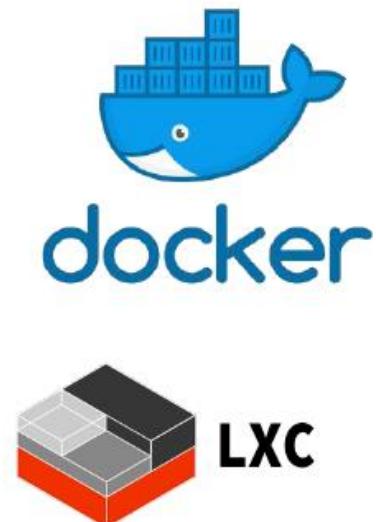
Contenedores y contenedorización (Containers and Containerization)

Un contenedor es como un conjunto de bibliotecas, dependencias y archivos de configuración. En comparación con las máquinas virtuales que emulan sistemas operativos completos, los contenedores comparten el núcleo (kernel) del host, lo que los hace más ligeros y rápidos.

Cada contenedor está aislado del otro, por lo que es independiente y sus procesos y recursos no interfieren entre sí.

¿Qué son contenedores?

Los contenedores son un paquete de elementos que permiten ejecutar una aplicación determinada en cualquier sistema operativo.



-
- Docker

 - Linux-VServer

 - LXC

 - LXD

 - OpenVZ

 - Systemd-nspawn

Contenedores y contenedorización (Containers and Containerization)

La contenedorización es el proceso de empaquetar una aplicación en un contenedor estandarizado. Docker es una popular plataforma de contenedorización que proporciona herramientas para crear, compartir y ejecutar contenedores.

La contenedorización beneficia a la ciencia de datos de muchas maneras:

Containerization

Reproducibilidad: Ejecute su análisis con el mismo entorno (versiones de software, dependencias) en cualquier lugar, garantizando la coherencia de los resultados.

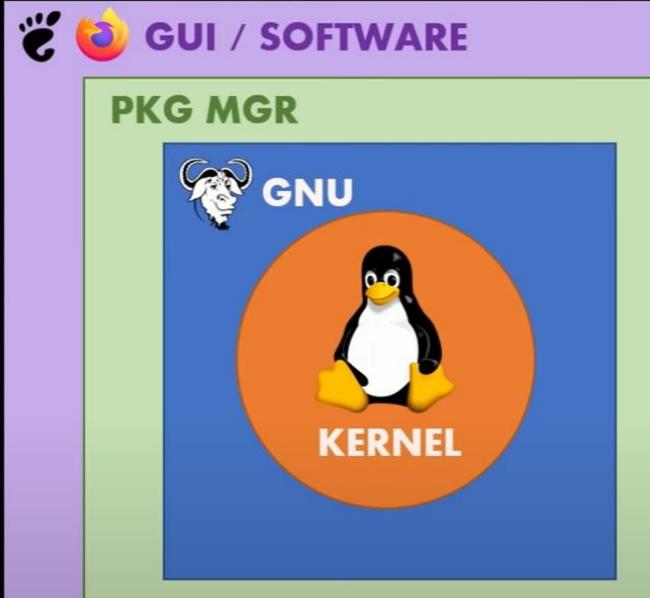
Portabilidad: Comparta y traslade fácilmente sus proyectos de ciencia de datos entre máquinas sin preocuparse por la configuración del entorno.

Eficiencia de recursos: Múltiples contenedores pueden ejecutarse en una sola máquina, maximizando los recursos informáticos para el procesamiento y análisis de datos.

Aislamiento: Aíslle tareas específicas como la ingeniería de características o la formación de modelos para una mejor colaboración y experimentación.

Máquinas virtuales frente a Docker

Máquinas virtuales



Contenedores



Máquinas virtuales frente a Docker

Las máquinas virtuales (VM) y los contenedores Docker son tecnologías utilizadas para aislar entornos de software, pero difieren significativamente en su enfoque y casos de uso. Comprender estas diferencias es crucial para elegir la herramienta adecuada para sus necesidades específicas.

Máquinas virtuales (VMs):

Emular un sistema operativo completo: Cada VM actúa como un ordenador virtual con su CPU, memoria, almacenamiento y sistema operativo.

Pros:

Aislamiento total: Las máquinas virtuales ofrecen un fuerte aislamiento entre entornos, lo que las hace ideales para ejecutar aplicaciones con dependencias conflictivas o requisitos de seguridad.

Máquinas virtuales (VMs):

Pros:

Flexibilidad: Las máquinas virtuales pueden ejecutar cualquier sistema operativo, lo que permite crear entornos adaptados a necesidades específicas.

Máquinas virtuales (VMs):

Contras:

Consumo intensivo de recursos: Las máquinas virtuales requieren muchos recursos, lo que limita el número de ellas que se pueden ejecutar en una sola máquina.

Arranque lento: Arrancar un sistema operativo completo lleva tiempo, por lo que las máquinas virtuales tardan más en iniciarse que los contenedores.

Contenedores Docker:

Empaqueta una aplicación con sus dependencias: Los contenedores comparten el núcleo del sistema operativo anfitrión y sólo necesitan bibliotecas y archivos específicos para ejecutarse.

Contenedores Docker:

Ligeros y portátiles: Los contenedores son mucho más pequeños y rápidos de arrancar que las máquinas virtuales, lo que los hace ideales para microservicios y aplicaciones escalables.

Entornos coherentes: Los contenedores estandarizados garantizan un comportamiento coherente de las aplicaciones en diferentes máquinas.

Uso eficiente de los recursos: Compartir el kernel permite ejecutar muchos más contenedores en una sola máquina en comparación con las VM.

Contenedores Docker:

Contras:

Aislamiento limitado: Aunque están aislados, los contenedores comparten el kernel, lo que introduce algunos riesgos potenciales de seguridad en comparación con las máquinas virtuales.

Elección de SO limitada: Los contenedores suelen utilizar el sistema operativo del host, lo que limita su flexibilidad en comparación con las máquinas virtuales.

Utilizar máquinas virtuales para:

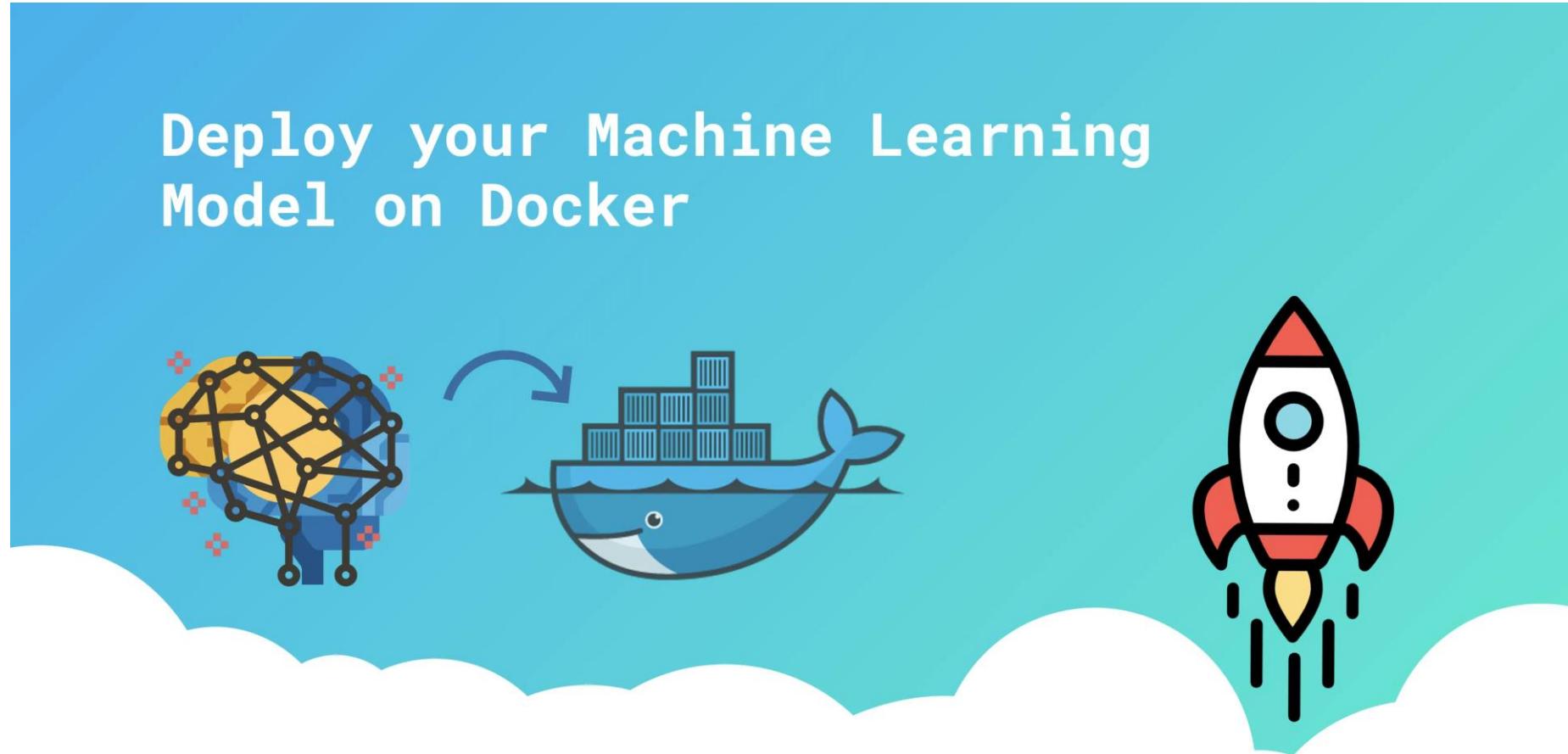
- Aplicaciones con dependencias conflictivas o estrictos requisitos de seguridad.
- Ejecución de varios sistemas operativos en una sola máquina.
- Cuando el aislamiento completo es crítico.

Utiliza contenedores Docker para:

- Microservicios y aplicaciones escalables.
- Despliegue de aplicaciones en entornos consistentes.
- Utilización eficiente de los recursos y despliegues rápidos.



¿Por qué es importante Docker en la ciencia de datos?



¿Por qué es importante Docker en la ciencia de datos?

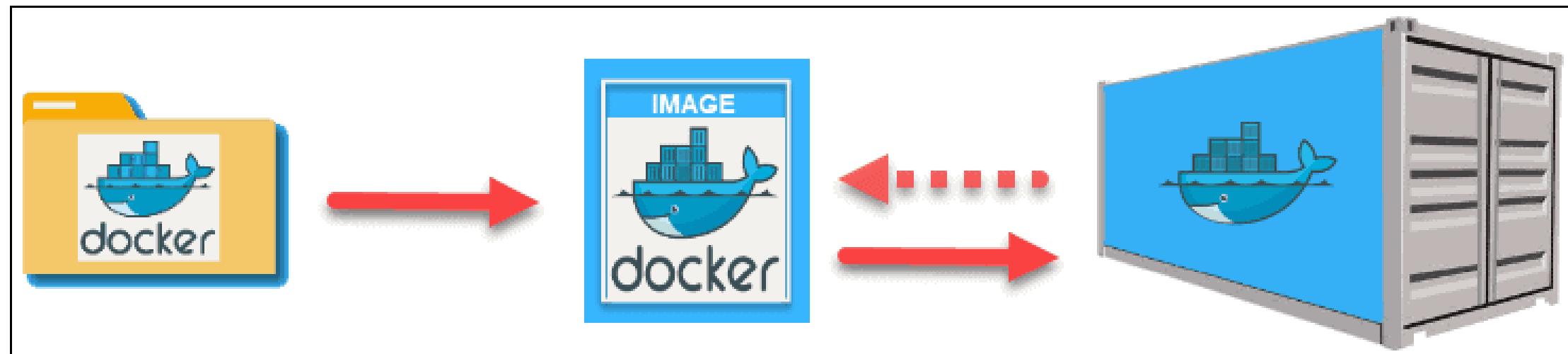
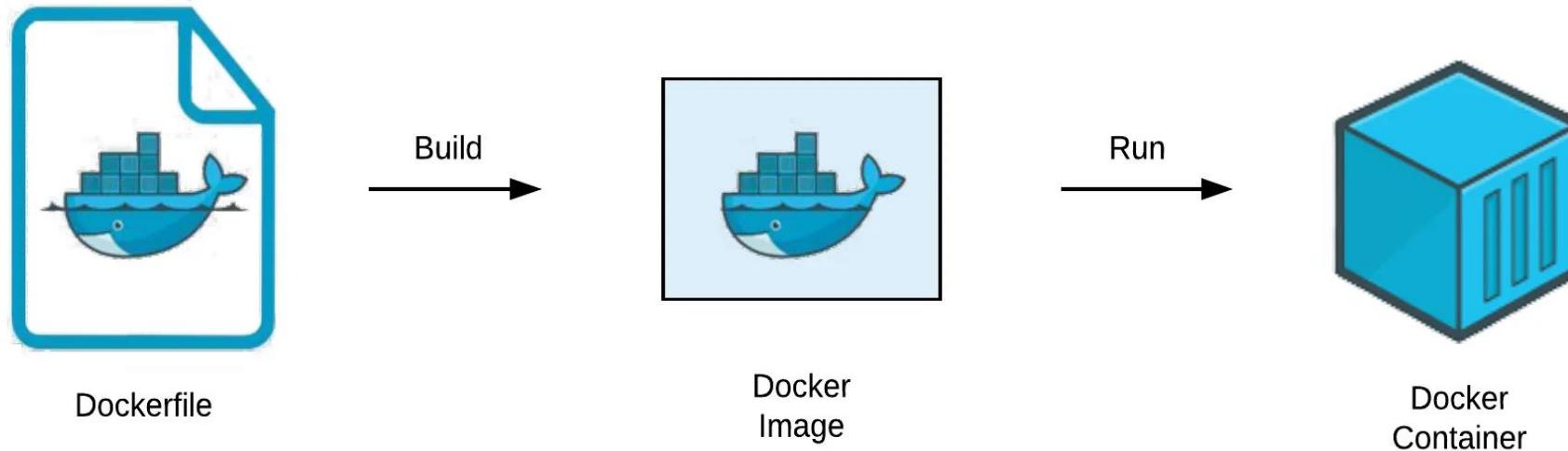
Suele pasar con mucha frecuencia que has desarrollado un modelo de aprendizaje automático y luego, cuando has cambiado de portátil, tu código se está rompiendo y te encuentras con una declaración infinita de "Error de importación" o "Módulo no encontrado". A veces, podría ser un error de versión debido a que se está tratando de ejecutar el código en una versión diferente de Python. La solución para esto es Docker.

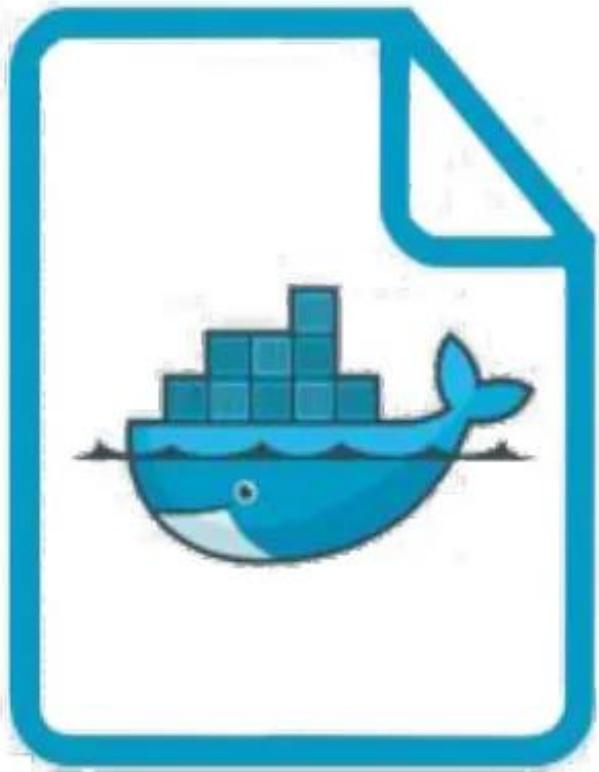
¿Por qué es importante Docker en la ciencia de datos?

Docker es una herramienta para crear y desplegar entornos aislados para ejecutar aplicaciones con sus dependencias. Básicamente, Docker facilita la escritura y ejecución de códigos sin problemas en otras máquinas con sistemas operativos diferentes, reuniendo el código y todas sus dependencias en un contenedor.

Este contenedor hace que el código sea autónomo e independiente del sistema operativo.

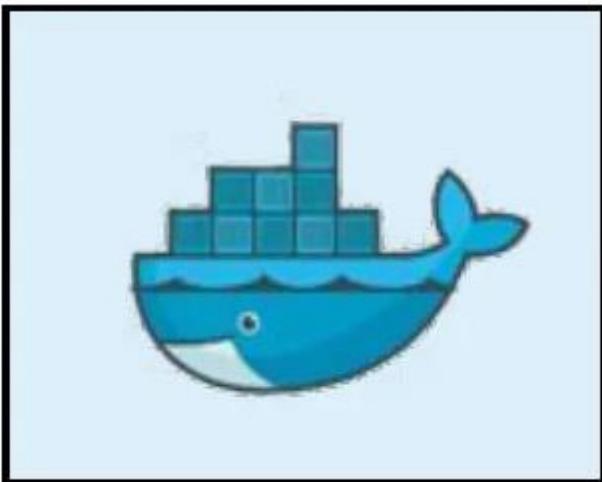
Terminología de Docker





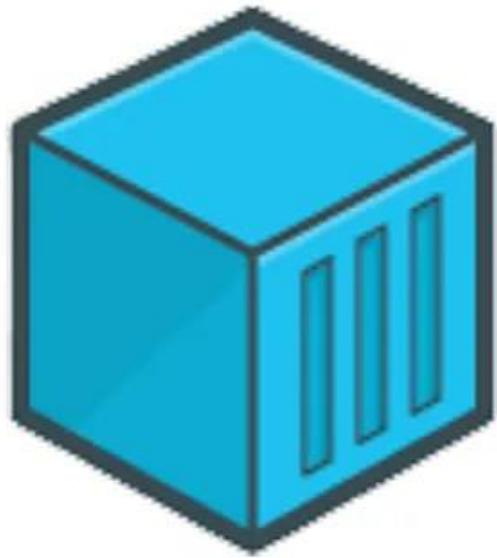
Dockerfile

Dockerfile - Un Dockerfile contiene todo el código para configurar un contenedor docker desde la descarga de la imagen docker hasta la configuración del entorno. Puedes pensar en él como si describiera la instalación completa del sistema operativo del sistema que quieras ejecutar.



Docker
Image

La imagen Docker es una plantilla de sólo lectura que contiene un conjunto de instrucciones para crear un contenedor que pueda ejecutarse en la plataforma Docker.



Docker
Container

Contenedor Docker - Un contenedor es una instancia ejecutable de una imagen. Puede crear, iniciar, detener, mover o eliminar un contenedor mediante la API o la CLI de Docker.

¿Por qué es importante Docker en la ciencia de datos?

Ejecutar modelos ML en contenedores Docker aporta una serie de ventajas:

Reproducibilidad: Una vez que hayas probado tu aplicación en contenedores, puedes desplegarla en cualquier otro sistema en el que se ejecute Docker y puedes estar seguro de que tu aplicación funcionará exactamente igual que cuando la probaste.



¿Por qué es importante Docker en la ciencia de datos?

Agilidad: La portabilidad y las ventajas de rendimiento que ofrecen los contenedores pueden ayudarle a que su proceso de desarrollo sea más ágil y receptivo. Mejora sus procesos de integración continua y entrega continua.

Portabilidad: Esto significa que pasar del desarrollo local a un clúster de supercomputación es fácil. Puede evitar problemas de configuración.



Importancia de Docker en Big Data

Consistencia en los entornos: Permite crear entornos de ejecución consistentes que se comportan de la misma manera, independientemente del sistema operativo o la infraestructura subyacente.

Los contenedores Docker encapsulan las dependencias, librerías y configuraciones necesarias para que las aplicaciones de Big Data funcionen correctamente. Esto asegura que el código funcione igual en desarrollo, prueba y producción.

Importancia de Docker en Big Data

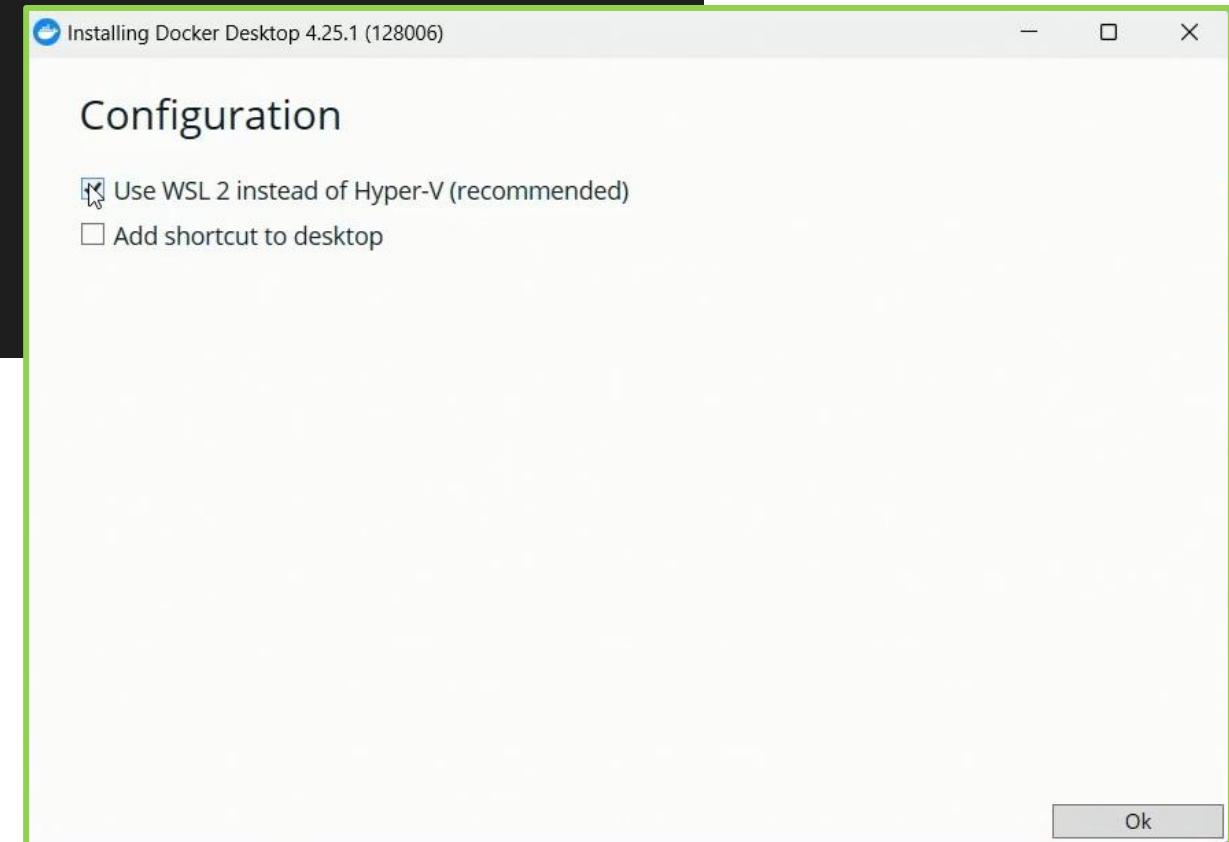
Docker facilita la escalabilidad al permitir que los nodos de procesamiento se creen y gestionen rápidamente en contenedores que pueden desplegarse en cualquier servidor o clúster.

Con Docker Swarm o Kubernetes, la orquestación de múltiples contenedores se simplifica, permitiendo escalar horizontalmente aplicaciones como Hadoop o Spark para adaptarse a cargas de trabajo cambiantes.

Verificar versión WSL en VS Code

PROBLEMS OUTPUT TERMINAL PORTS GITLENS AZURE SQL CONSOLE COMMENTS DEBUG CONSOLE

- PS C:\Users\juanj\OneDrive\Escritorio\Big-Data-2024> `wsl --version`
Versión de WSL: 2.3.24.0
Versión de kernel: 5.15.153.1-2
Versión de WSLg: 1.0.65
Versión de MSRDC: 1.2.5620
Versión de Direct3D: 1.611.1-81528511
Versión DXCore: 10.0.26100.1-240331-1435.ge-release
Versión de Windows: 10.0.22631.4317
- PS C:\Users\juanj\OneDrive\Escritorio\Big-Data-2024>



Instalar extensión en VS Code

The screenshot shows the Visual Studio Code interface with the Extensions sidebar open. The search bar at the top contains the text "Big-Data-2024". In the Extensions sidebar, the "Docker" extension by Microsoft is highlighted with a red box. The main view displays the "Docker" extension details page. The extension icon is a blue whale with a stack of squares above it. The name "Docker" is displayed in large letters, followed by the version "v1.29.3". The developer information shows "Microsoft" and "microsoft.com" with a star rating of 5 stars and 38,395,485 installs. Below this, a description states "Makes it easy to create, manage, and debug containerized applications." There are buttons for "Disable", "Uninstall", and "Auto Update". At the bottom, tabs for "DETAILS", "FEATURES", "CHANGELOG", and "DEPENDENCIES" are visible, along with sections for "Contributing" and "Code of Conduct". The status bar at the bottom shows tabs for PROBLEMS, OUTPUT, TERMINAL, PORTS, GITLENS, AZURE, SQL CONSOLE, COMMENTS, and DEBUG CONSOLE. The "TERMINAL" tab is currently selected.

Instalación de Docker en Windows

[Home](#) / [Manuals](#) / [Docker Desktop](#) / [Install](#) / [Windows](#)

Install Docker Desktop on Windows

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than \$10 million USD in annual revenue) requires a [paid subscription](#).

This page contains the download URL, information about system requirements, and instructions on how to install Docker Desktop for Windows.

[Docker Desktop for Windows - x86_64](#)

[Docker Desktop for Windows - Arm \(Beta\)](#)

For checksums, see [Release notes](#)

UI de Docker cuando se ejecuta desde Windows

Docker desktop PERSONAL

Containers [Give feedback](#)

Container CPU usage ⓘ Container memory usage ⓘ Show charts

No containers are running.

Search Only show running containers

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	proxy dcf993e5b110	nginx:<none>	Exited	8080:80	N/A	2 months ago	> ⋮ trash

Docker desktop PERSONAL

Containers [Give feedback](#)

Container CPU usage ⓘ Container memory usage ⓘ Show charts

No containers are running.

Search Only show running containers

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	proxy dcf993e5b110	nginx:<none>	Exited	8080:80	N/A	2 months ago	> ⋮ trash

Learning Center

Containers, volumes, extensions and more... **Ctrl+K**

Learning center

Walkthroughs

Quick hands-on guides to show you around

What is a container?
5 mins

How do I run a container?
6 mins

[View all](#)

AI/ML guides

Get started with AI/ML using Docker

GenAI Stack
Start your GenAI application using Neo4j, Langchain, Ollama, Python, and Docker Compose

Docker for beginners by language

45 min guides written for different programming languages

NodeJS

Python

Go

Java

C# (.NET)

Rust

[Request a guide ↗](#)

Docker Images

Images [Give feedback](#)

Local Hub

265.74 MB / 265.74 MB in use 2 images

Search Filter Sort

Name	Tag
nginx 5ef79149e0ec	latest
ubuntu edbfe74c41f8	latest

Images [Give feedback](#)

Local Hub

juanjorb23

Search

Tags

 juanjorb23/sitioweb	latest
---	--------

Volumes en Docker:

Volumes Give feedback						
<input type="checkbox"/>	Search	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Name	Status	Created	Size	Scheduled exports <small>BETA</small>	Actions
<input type="checkbox"/>	03fee6fec6a1fe1efefb3136c54d8a6dde3d4a618cf4458329ffc77b971b1115	-	2 months ago	0 Bytes	Inactive	<input type="button"/> <input type="button"/>
<input type="checkbox"/>	20ef28bd50e42339859896a248ee28e04b4b7b1318101a4164f08767a1e6effd	-	2 months ago	300.4 MB	Inactive	<input type="button"/> <input type="button"/>
<input type="checkbox"/>	4806a44ba1e1c549d556d99acd8fe80ea9b82804a997834ced0c8c3335fa84b8	-	2 months ago	300.2 MB	Inactive	<input type="button"/> <input type="button"/>

Es un mecanismo para persistir datos generados o utilizados por contenedores. Los volúmenes se utilizan cuando los datos necesitan sobrevivir al ciclo de vida de un contenedor (es decir, cuando un contenedor se detiene o es eliminado, los datos persisten).

Settings - Configuración

The screenshot shows the Docker Desktop settings interface. The title bar includes the Docker Desktop logo, the word "PERSONAL", a search bar ("Search for images, containers, volumes, extensions and more..."), and various icons for control and help. The main header says "Settings" and "Give feedback".

The left sidebar has a "General" tab selected, which is highlighted in blue. Other tabs include "Resources", "Docker Engine", "Builders", "Kubernetes", "Software updates", "Extensions", "Features in development", and "Notifications".

The "General" tab contains the following configuration options:

- Start Docker Desktop when you sign in to your computer
- Open Docker Dashboard when Docker Desktop starts
- Choose theme for Docker Desktop:
 Light Dark Use system settings
- Choose container terminal:
 Integrated System default
Determines which terminal is launched when opening the terminal from a container.
- Enable Docker terminal
- Enable Docker Debug by default [Learn more](#)
Active subscription required. [Upgrade](#)
- Expose daemon on tcp://localhost:2375 without TLS
Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.
- Use the WSL 2 based engine (Windows Home can only run the WSL 2 backend)
WSL 2 provides better performance than the Hyper-V backend. [Learn more](#)
- Add the *.docker.internal names to the host's /etc/hosts file (Requires password)
Lets you resolve *.docker.internal DNS names from both the host and your containers. [Learn more](#)
- Use containerd for pulling and storing images [Give feedback](#)
The containerd image store enables native support for multi-platform images, attestations, Wasm, and more.
- Send usage statistics

Docker CLI

Desde vs code abrimos una terminal de wsl y ejecutamos: **docker --version**

PROBLEMS OUTPUT TERMINAL PORTS GITLENS AZURE SQL CONSOLE COMMENTS DEBUG CONSOLE

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker --version
Docker version 27.2.0, build 3ab4256
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ █
```

Para acceder a toda la documentación (todos los comandos) se escribe: **docker**

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker
Usage: docker [OPTIONS] COMMAND
A self-sufficient runtime for containers
Common Commands:
```

El comando docker info: proporciona un resumen detallado de la configuración actual de Docker, incluyendo información sobre el estado del servidor, los recursos disponibles y la configuración del entorno de Docker.

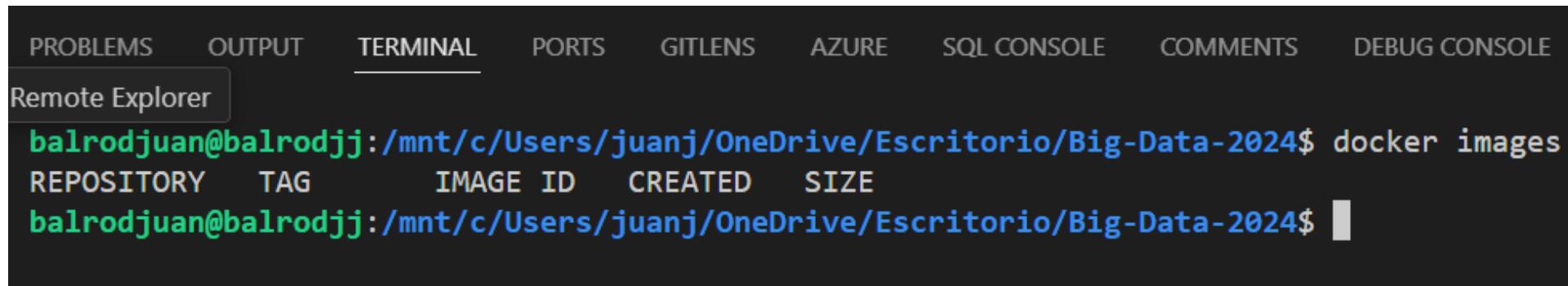
```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker info
Client:
  Version: 27.2.0
  Context: default
  Debug Mode: false
  Plugins:
    buildx: Docker Buildx (Docker Inc.)
      Version: v0.16.2-desktop.1
      Path: /usr/local/lib/docker/cli-plugins/docker-buildx
    compose: Docker Compose (Docker Inc.)
      Version: v2.29.2-desktop.2
      Path: /usr/local/lib/docker/cli-plugins/docker-compose
    debug: Get a shell into any image or container (Docker Inc.)
      Version: 0.0.34
      Path: /usr/local/lib/docker/cli-plugins/docker-debug
    desktop: Docker Desktop commands (Alpha) (Docker Inc.)
      Version: v0.0.15
      Path: /usr/local/lib/docker/cli-plugins/docker-desktop
    dev: Docker Dev Environments (Docker Inc.)
      Version: v0.1.2
      Path: /usr/local/lib/docker/cli-plugins/docker-dev
    extension: Manages Docker extensions (Docker Inc.)
      Version: v0.2.25
      Path: /usr/local/lib/docker/cli-plugins/docker-extension
    feedback: Provide feedback, right in your terminal! (Docker Inc.)
```

Actividad:

Escribe docker info y luego explica lo que contiene la salida. Investigar para que se usa. Guarda tus resultados en tus apuntes

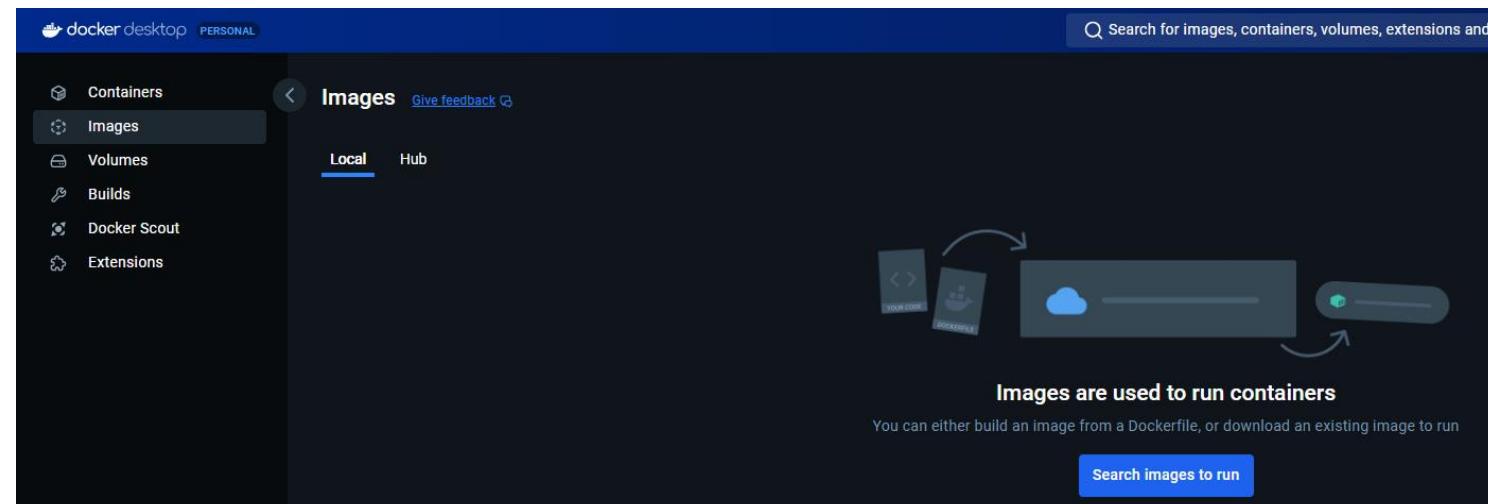
docker images

Se usa para listar las imágenes disponibles localmente en tu sistema Docker. Proporciona un resumen de todas las imágenes que están almacenadas en tu máquina, mostrando detalles clave como el nombre de la imagen, la etiqueta, el ID, la fecha de creación y el tamaño.



```
PROBLEMS OUTPUT TERMINAL PORTS GITLENS AZURE SQL CONSOLE COMMENTS DEBUG CONSOLE
Remote Explorer
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker images
REPOSITORY      TAG          IMAGE ID   CREATED     SIZE
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$
```

Desde Docker Desktop



Documentación de docker images: docker images --help

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker images --help
```

```
Usage: docker images [OPTIONS] [REPOSITORY[:TAG]]
```

```
List images
```

```
Aliases:
```

```
  docker image ls, docker image list, docker images
```

```
Options:
```

-a, --all	Show all images (default hides intermediate images)
--digests	Show digests
-f, --filter filter	Filter output based on conditions provided
--format string	Format output using a custom template: 'table': Print output in table format with column headers (default) 'table TEMPLATE': Print output in table format using the given Go template 'json': Print in JSON format 'TEMPLATE': Print output using the given Go template. Refer to https://docs.docker.com/go/formatting/ for more information about formatting output with templates
--no-trunc	Don't truncate output
-q, --quiet	Only show image IDs
--tree	List multi-platform images as a tree (EXPERIMENTAL)

docker run

Permite crear y ejecutar contenedores a partir de una imagen. Al ejecutar este comando, Docker hace lo siguiente:

Descargar la imagen (si no está disponible localmente).

Crear un contenedor basado en esa imagen.

Ejecutar un comando dentro del contenedor.

Asignar recursos (como redes, volúmenes, etc.) según las opciones proporcionadas.

Sintaxis básica:

```
docker run [opciones] imagen [comando]
```

Ejemplo, vamos a ejecutar Ubuntu desde una imagen oficial de Docker Hub

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker run -it ubuntu /bin/bash
```

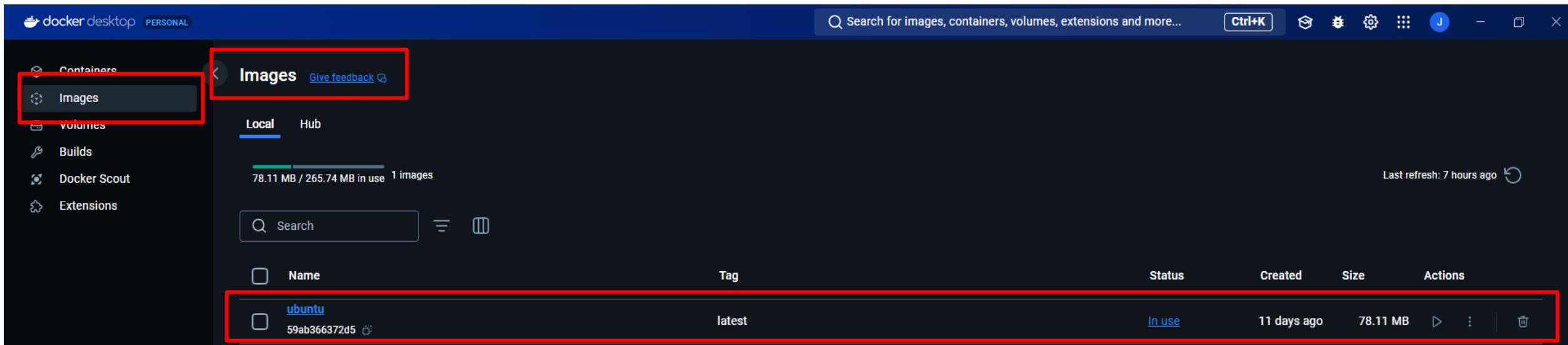
```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
ff65ddf9395b: Pull complete
Digest: sha256:99c35190e22d294cdace2783ac55effc69d32896daaa265f0bbbedbcde4fbe3e5
Status: Downloaded newer image for ubuntu:latest
root@1a428a62008c:/#
```

```
root@1a428a62008c:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@1a428a62008c:/#
```

```
docker run -it ubuntu /bin/bash
```

- -it: Ejecuta el contenedor en modo interactivo y con una terminal.
- ubuntu: Nombre de la imagen que estás utilizando.
- ./bin/bash: El comando que se ejecuta dentro del contenedor, en este caso, abre una shell de bash.

En Docker Desktop, en ‘Images’ se ve lo siguiente:



The screenshot shows the Docker Desktop interface with the 'Images' tab selected. A red box highlights the 'Images' tab in the sidebar and the 'ubuntu' image entry in the main table.

Docker Desktop PERSONAL

Search for images, containers, volumes, extensions and more... Ctrl+K

Containers Images Give feedback

volumes Builds Docker Scout Extensions

Local Hub

78.11 MB / 265.74 MB in use 1 images Last refresh: 7 hours ago

Search

Name	Tag	Status	Created	Size	Actions
ubuntu 59ab366372d5	latest	In use	11 days ago	78.11 MB	▶ ⋮ trash

Mientras que, en ‘Containers’ se ve lo siguiente:

The screenshot shows the Docker Desktop application interface. The left sidebar has options: Containers (selected and highlighted with a red box), Images, Volumes, Builds, Docker Scout, and Extensions. The main area is titled 'Containers' with a 'Give feedback' link. It displays 'Container CPU usage' and 'Container memory usage', both showing 'No containers are running.' There is a 'Show charts' link. A search bar and a 'Only show running containers' toggle are present. The table below lists one container:

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	exciting_ishizaka 1a428a62008c	ubuntu:<none>	Exited		N/A	1 hour ago	▶ ⋮ trash

Ejecutar un contenedor y eliminarlo al salir

```
docker run --rm ubuntu echo "Hola desde Docker"
```

Ejecuta un contenedor temporal que se elimina automáticamente cuando termina su ejecución.

- `--rm`: Elimina automáticamente el contenedor cuando termina.
- `echo "Hola desde Docker"`: El comando que se ejecuta dentro del contenedor, que imprime un mensaje.

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker run --rm ubuntu echo "Hola desde Docker"
Hola desde Docker
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$
```

Si no se elimina con el comando anterior hacer lo siguiente:

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    GITLENS    AZURE    SQL CONSOLE    COMMENTS    DEBUG CONSOLE

balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
1a428a62008c        ubuntu              "/bin/bash"         3 hours ago       Exited (0) 2 hours ago
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker stop 1a428a62008c
1a428a62008c
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker rm 1a428a62008c
1a428a62008c
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$
```

Si nos vamos a Docker Desktop podemos verificar que se borró el ‘container’

The screenshot shows the Docker Desktop application window. The top navigation bar includes the Docker logo, 'docker desktop' text, a 'PERSONAL' badge, a search bar with placeholder text 'Search for images, containers, volumes, extensions and more...', and keyboard shortcut 'Ctrl+K'. On the left, a sidebar menu lists 'Containers' (selected), 'Images', 'Volumes', 'Builds', 'Docker Scout', and 'Extensions'. The main content area is titled 'Containers' with a 'Give feedback' link. It features a central graphic of three blue cylinders representing containers, with a small green cube icon in the middle cylinder. Below the graphic, the text 'Your running containers show up here' is displayed, followed by the definition 'A container is an isolated environment for your code'. At the bottom, there are two cards: 'What is a container?' (5 mins) with a question mark icon, and 'How do I run a container?' (6 mins) with a code snippet icon.

docker desktop PERSONAL

Containers Images Volumes Builds Docker Scout Extensions

Containers Give feedback

Search for images, containers, volumes, extensions and more... Ctrl+K

Your running containers show up here

A container is an isolated environment for your code

What is a container? 5 mins

How do I run a container? 6 mins

View more in the Learning center

En el sitio web oficial de Docker se pueden descargar las imágenes

The screenshot shows the Docker Hub homepage with a dark blue header. The header includes the Docker Hub logo, navigation links for 'Explore', 'Repositories', 'Organizations', and 'Usage', and various user interface icons. A banner at the top reads 'Develop faster. Run anywhere.' and 'Docker Hub is the world's easiest way to create, manage, and deliver your team's container applications.' Below the banner is a search bar with the placeholder 'Search Docker Hub' and a keyboard shortcut 'ctrl+K'. On the left side, there are two expandable sections: 'Trusted content' (listing Docker Official Image, Verified Publisher, and Sponsored OSS) and 'Categories' (listing API Management, Content Management System, Data Science, Databases & Storage, Languages & Frameworks, Integration & Delivery, Internet of Things, Machine Learning & AI, Message Queues, Monitoring & Observability, Networking, Operating Systems, Security, Web Servers, and Developer Tools). The main content area features a 'Spotlight' section with three cards: 'Build up to 39x faster with Docker Build Cloud' (Cloud Development), 'LLM Everywhere: Docker and Hugging Face' (AI/ML Development), and 'Take action on prioritized insights' (Software Supply Chain). Below the spotlight is a 'Machine Learning & AI' section with cards for 'tensorflow/tensorflow', 'pytorch/pytorch', 'langchain/langchain', and 'ollama/ollama'. At the bottom, a 'Trending this week' section is partially visible.

Explore Repositories Organizations Usage

New More Docker. Easy Access. New Streamlined Plans. Learn more. →

Docker Hub

Develop faster. Run anywhere.

Docker Hub is the world's easiest way to create, manage, and deliver your team's container applications.

Search Docker Hub ctrl+K

Trusted content

- Docker Official Image
- Verified Publisher
- Sponsored OSS

Categories

- API Management
- Content Management System
- Data Science
- Databases & Storage
- Languages & Frameworks
- Integration & Delivery
- Internet of Things
- Machine Learning & AI
- Message Queues
- Monitoring & Observability
- Networking
- Operating Systems
- Security
- Web Servers
- Developer Tools

Spotlight

CLOUD DEVELOPMENT
Build up to 39x faster with Docker Build Cloud

Introducing Docker Build Cloud: A new solution to speed up build times and improve developer productivity

docker buildcloud

AI/ML DEVELOPMENT
LLM Everywhere: Docker and Hugging Face

Set up a local development environment for Hugging Face with Docker

LLM

SOFTWARE SUPPLY CHAIN
Take action on prioritized insights

Bridge the gap between development workflows and security needs

docker scout

Machine Learning & AI

tensorflow/tensorflow
Official Docker images for the machine learning framework TensorFlow...

2.6K 50M+

pytorch/pytorch
PyTorch is a deep learning framework that puts Python first.

1.2K 10M+

langchain/langchain
Building applications with LLMs through compositability

183 50K+

ollama/ollama
The easiest way to get up and running with large language models.

795 5M+

Trending this week ↗

Ejemplo básico de creación de un contenedor básico para un portfolio o web personal.

1. Creamos una carpeta en local para nuestro porfolio

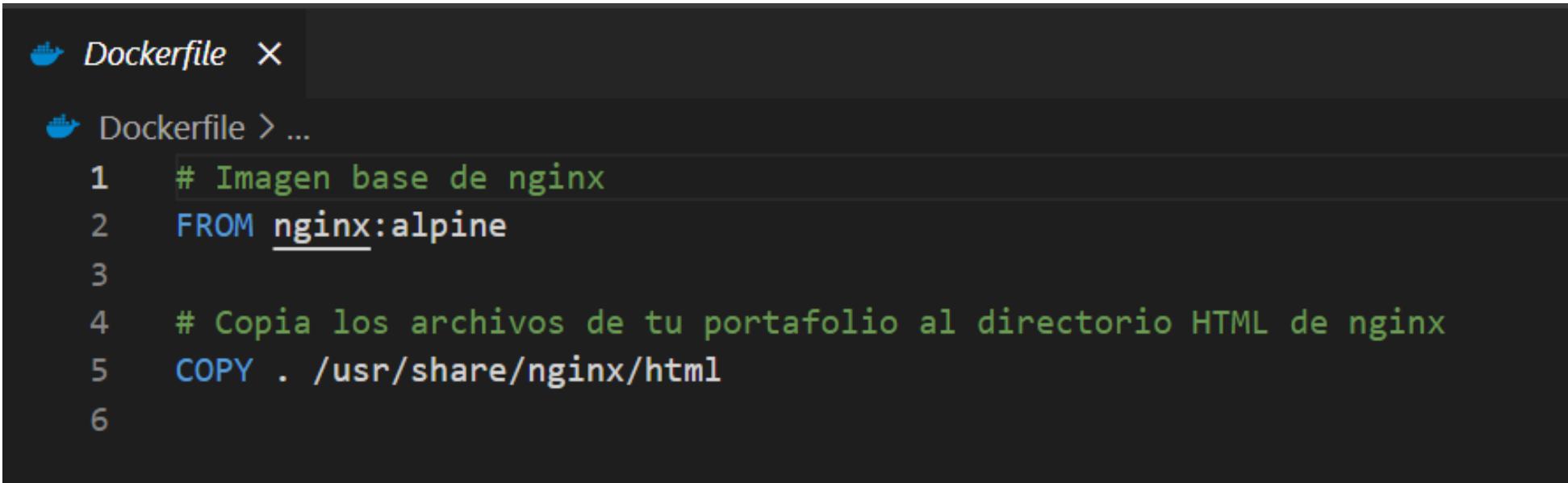
```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio$ mkdir miport  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio$ cd miport  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ █
```

2. Creamos un archivo **Dockerfile** dentro de la carpeta miport

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ touch Dockerfile█
```

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ code Dockerfile█
```

Contenido del Dockerfile



A screenshot of a code editor window titled "Dockerfile". The file contains the following Dockerfile code:

```
1 # Imagen base de nginx
2 FROM nginx:alpine
3
4 # Copia los archivos de tu portafolio al directorio HTML de nginx
5 COPY . /usr/share/nginx/html
6
```

3. Crear la plantilla html para el portfolio

3. Crear plantilla html para nuestro portfolio

3.1 Crear el archivo index.html en la misma carpeta del Dockerfile

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ touch index.html  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ code index.html  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ █
```

El contenido del archivo index.html lo tienes en el bloc de notas

4. Construir la imagen de Docker

Desde la terminal y estando ubicados en el mismo directorio del Dockerfile escribimos:

docker build -t miport .

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ docker build -t miport .
[+] Building 7.2s (8/8) FINISHED                                            docker:default
=> [internal] load build definition from Dockerfile                      0.1s
=> => transferring dockerfile: 176B                                         0.1s
=> [internal] load metadata for docker.io/library/nginx:alpine           3.5s
=> [auth] library/nginx:pull token for registry-1.docker.io               0.0s
=> [internal] load .dockerignore                                         0.0s
=> => transferring context: 2B                                           0.0s
=> [1/2] FROM docker.io/library/nginx@sha256:2140dad235c130ac861018a4e13a6bc8aea3a35f3a40e20c1b060d51a7efd250 3.2s
=> => resolve docker.io/library/nginx@sha256:2140dad235c130ac861018a4e13a6bc8aea3a35f3a40e20c1b060d51a7efd250 0.0s
=> => sha256:2140dad235c130ac861018a4e13a6bc8aea3a35f3a40e20c1b060d51a7efd250 9.07kB / 9.07kB 0.0s
=> => sha256:43c4264eed91be63b206e17d93e75256a6097070ce643c5e8f0379998b44f170 3.62MB / 3.62MB 0.9s
=> => sha256:596d53a7de8832c0963cd374bf19a0a1ca2284c80329e1a1462c4f51035ae0c8 629B / 629B 0.5s
=> => sha256:ae136e431e76e12e5d84979ea5e2ffff4dd9589c2435c8bb9e33e6c3960111d3 2.50kB / 2.50kB 0.0s
```

`docker build -t mi_portafolio .`

Cuando ejecutas este comando en el mismo directorio donde está el Dockerfile, Docker hará lo siguiente:

- Buscará el archivo Dockerfile en el directorio actual (representado por el .).
- Leerá las instrucciones en el Dockerfile.
- Descargará la imagen base (`nginx:alpine`) si no está disponible localmente.
- Ejecutará los comandos en el Dockerfile, como copiar archivos o instalar dependencias.
- Construirá una nueva imagen con los cambios realizados.
- Asignará el nombre y la etiqueta `miport:latest` a la nueva imagen.

Resultado

Después de ejecutar el comando, tendrás una nueva imagen de Docker llamada miport en tu sistema. Puedes verificar que se ha creado con:
docker images

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
miport          latest        8d737369c9fd   8 minutes ago  47MB
ubuntu           latest        59ab366372d5   11 days ago   78.1MB
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ █
```

5. Ejecutar el Contenedor Docker

5.1. Utilizando la imagen que has creado ejecuta el contenedor Docker.

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ docker run -d -p 8080:80 miport  
22265504da35af47ab5bcd970931da5a60c09463a9823c19abaa74fde6fe60f2  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/miport$ █
```

5.2. Accede a tu portfolio en el navegador web.

<http://localhost:8080>

Vista del portafolio desde el navegador:

Bienvenido a Mi Portafolio

Proyecto 1

Descripción del proyecto 1.

Proyecto 2

Descripción del proyecto 2.

6. Publicar en GitHub.

6.1 Iniciar en local:

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/miport
● $ git init
Initialized empty Git repository in C:/Users/juanj/OneDrive/Escritorio/miport/.git/
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/miport (master)
● $ git add .
warning: in the working copy of 'Dockerfile', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/miport (master)
● $ git commit -m "Inicial commit de mi portafolio"
[master (root-commit) 3e1bea5] Inicial commit de mi portafolio
 2 files changed, 36 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 index.html
```

6.2 Crear un nuevo repositorio en GitHub

The screenshot shows a GitHub repository page for a repository named "miport". The repository is public. At the top, there are buttons for "Pin" and "Unwatch" with a count of 1. Below the header, there are buttons for "main" (with a dropdown arrow), "1 Branch" (with a dropdown arrow), and "Tags". A search bar with a magnifying glass icon and the placeholder "Go to file" is followed by a "t" button and a "+" button. A green "Code" button with a dropdown arrow is also present. The main content area shows a single commit by "rodbalza" titled "Initial commit" with hash "51a9393 · now" and "1 Commit". Below the commit, there is a file listing for "README.md" with the status "Initial commit" and "now". At the bottom, there is a preview of the "README" file content, which contains the word "miport".

miport Public

Pin Unwatch 1

main ▾ 1 Branch Tags

Go to file t + Code ▾

rodbalza Initial commit 51a9393 · now 1 Commit

README.md Initial commit now

README

miport

6.2 Agrega el repositorio remoto

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/miport (master)
$ git remote add origin https://github.com/rodbalza/miport.git
```

6.3 Subir los archivos al repositorio remoto

```
juanj@balrodjj MINGW64 ~/OneDrive/Escritorio/miport (master)
● $ git push -u origin master
  Enumerating objects: 4, done.
  Counting objects: 100% (4/4), done.
  Delta compression using up to 8 threads
  Compressing objects: 100% (4/4), done.
  Writing objects: 100% (4/4), 766 bytes | 766.00 KiB/s, done.
  Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:     https://github.com/rodbalza/miport/pull/new/master
remote:
To https://github.com/rodbalza/miport.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Practica .

Visita el sitio web de Docker Hub busca imágenes de algún video juego muy ligero que puedas configurar y ejecutar usando Docker run.

Bind mounts

Cuando usas un bind mount, estás diciendo a Docker que monte un directorio o archivo específico del host en un directorio específico dentro del contenedor. Cualquier cambio realizado en los archivos del directorio del host se refleja inmediatamente dentro del contenedor, y viceversa.

Vamos a crear un nuevo directorio

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/Big-Data-2024$ cd ..  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio$ mkdir dockerdata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio$ cd dockerdata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ █
```

Ahora vamos a ejecutar un nuevo contenedor de MongoDB

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name db mongo  
Unable to find image 'mongo:latest' locally  
latest: Pulling from library/mongo  
ff65ddf9395b: Already exists  
382d4b58543b: Pull complete  
fdc78058942b: Pull complete  
12e5d6c58a07: Pull complete  
b21d1c4a4df0: Pull complete  
e5f63e34d357: Pull complete  
cca7bf1bde66: Pull complete  
6582a7dd6d6f: Pull complete  
Digest: sha256:9342a9279a9841fc5f8192e49dcaaf9404e6bb3a90e8cf134eb96074830dd448  
Status: Downloaded newer image for mongo:latest  
5c9bc93bd64d2fd089510a4979f1185c26043928049c6ba4f4b49596d309e378
```

docker run -d --name db mongo

- -d: indica que el contenedor se debe ejecutar en segundo plano y no bloqueará la terminal como sucede con la opción –it.
- --name: Se utiliza para asignar un nombre específico al contenedor, en nuestro caso hemos llamado a nuestro contenedor db. Darle nombre a tu contenedor lo hace más fácil para referenciarlo en lugar de usar el ID.
- mongo: Este es el nombre de la imagen de MongoDB que Docker usará para crear el contenedor.

Verificamos que el contenedor está corriendo:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS     NAMES
5c9bc93bd64d        mongo      "docker-entrypoint.s..."   9 minutes ago      Up 9 minutes   27017/tcp   db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Vamos a entrar a este contenedor para crear algunos datos

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it db bash
root@5c9bc93bd64d:/#
```

Este comando abre una terminal interactiva de bash dentro del contenedor que ya está en ejecución. Docker exec se utiliza para ejecutar un nuevo proceso o comando dentro de un contenedor en ejecución.

Entramos escribiendo: mongosh

```
root@5c9bc93bd64d:/# mongosh
```

```
-----  
test> |
```

¿Qué bases de datos trae Mongo?

```
test> show dbs  
admin   40.00 KiB  
config  12.00 KiB  
local   40.00 KiB  
test> |
```

Vamos a crear una base de datos simple

```
test> use prueba
switched to db prueba
prueba>
```

Insertamos algunos datos

```
prueba> db.users.insert({"nombre":"Nadie"})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('67194d63caa8b53f9ffe6911') }
}
prueba> db.users.insertOne({"nombre":"Alguien"})
{
  acknowledged: true,
  insertedId: ObjectId('67194dc9caa8b53f9ffe6912')
}
```

Verificamos si se hizo el registro

```
prueba> db.users.find()
[
  { _id: ObjectId('67194d63caa8b53f9ffe6911'), nombre: 'Nadie' },
  { _id: ObjectId('67194dc9caa8b53f9ffe6912'), nombre: 'Alguien' }
]
prueba>
```

Salimos del mongosh

```
prueba> exit
root@5c9bc93bd64d:/#
```

Borramos el contenedor:

```
root@5c9bc93bd64d:/# exit
exit
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker rm -f db
db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS          NAMES
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

docker rm -f db

- La opción -f es la abreviatura de force, lo que significa que el contenedor será forzadamente detenido y eliminado.
- Si el contenedor aún está en ejecución, Docker primero detendrá el contenedor de manera forzada y luego lo eliminará. Es útil si quieres eliminar un contenedor en ejecución sin tener que detenerlo manualmente antes.

Bind mounts

Al borrar un contenedor de la forma anterior perdemos también los datos

¿Qué se podría hacer para mantener los datos una vez que ya no queremos usar el contenedor? Una opción es usar Bind mounts.

Vamos a crear un nuevo directorio para el ejemplo: mkdir mongodata

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ mkdir mongodata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ █
```

Obtenemos la ruta de nuestro directorio y copiamos:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ pwd  
/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ █
```

Creamos un volumen en docker

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker volume create --name mongodata  
mongodata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

docker run -d --name db -v mongodata:/data/db mongo

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name db -v mongodata:/data/db mongo  
2ea4ecb6a46f5413afe34a6d05685b9e6e14378faf7da2858d427f6ac0ddc7ca  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
2ea4ecb6a46f mongo "docker-entrypoint.s..." 54 seconds ago Up 53 seconds 27017/tcp db  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

docker exec –it db bash

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it db bash
```

Ejecutamos mongodb escribiendo mongosh

```
root@2ea4ecb6a46f:/# mongosh
```

```
test> use prueba
switched to db prueba
```

Vamos a introducir un registro cualquiera:

```
prueba> db.users.insertOne({"nombre": "Alguien"})
{
  acknowledged: true,
  insertedId: ObjectId('6719c0217026b6a829fe6911')
}
prueba>
```

Verificamos:

```
prueba> db.users.find()
[ { _id: ObjectId('6719c0217026b6a829fe6911'), nombre: 'Alguien' } ]
prueba>
```

Salimos de mongodb y del root

```
prueba> exit
root@2ea4ecb6a46f:/# exit
exit
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Veamos el estado del contenedor y luego lo eliminamos:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
2ea4ecb6a46f mongo "docker-entrypoint.s..." 17 minutes ago Up 17 minutes 27017/tcp db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker rm -f db
db
```

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
22265504da35 miport "/docker-entrypoint...." 30 hours ago Exited (0) 29 hours ago eloquent_nightingale
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Una vez eliminado el contenedor vamos a entrar en la carpeta mongodata y listamos archivos

```
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ cd mongodata
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata/mongodata$ ls
journal  mongod.lock  WiredTiger  WiredTiger.lock  WiredTiger.wt  WiredTiger.wt.1  WiredTiger.wt.2  WiredTiger.wt.3  WiredTiger.wt.4  WiredTiger.wt.5
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata/mongodata$
```

Si ahora corremos otro contenedor mongodb y le montamos el directorio mongodata, los datos que están guardados allí van estar disponibles en mongodb:

```
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name db -v mongodata:/data/db mongo
c8326849556c626364b5381310f6d89cf5425fef467b8931a7e7cd7e250b6ad1
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS          NAMES
c8326849556c        mongo       "docker-entrypoint.s..."   4 seconds ago   Up 3 seconds   27017/tcp      db
balrodrjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Entramos a mongo Shell otra vez (mongosh)

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it db bash
root@c8326849556c:/# mongosh
```

```
test> |
```

Al efectuar la consulta confirmamos que los datos antiguos de la base de datos anterior no se perdieron:

```
test> use prueba
switched to db prueba
prueba> db.users.find()
[ { _id: ObjectId('6719c0217026b6a829fe6911'), nombre: 'Alguien' } ]
prueba> |
```

Volúmenes

Eliminamos el contenedor anteriormente usado:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c8326849556c mongo "docker-entrypoint.s..." 24 minutes ago Up 24 minutes 27017/tcp db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker rm -f db
db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Ahora vamos a crear un nuevo volumen:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker volume create dbdata
dbdata
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Para listar los volúmenes escribimos: docker volume ls

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker volume ls
DRIVER      VOLUME NAME
local      09bc4f8562d898447be49112393baf9dc6733d3f7c96759aad953a01f34d99e
local      044f5d4b31a70047797f93e2f990a19d1ecddc3b83c4d6a2831d120ce3467535
local      266cf91b2a69e73c5d5ae6aded4ae1175b1a70190bf1a394a15a103c5779f833
local      99202e038700c105d3d19845bb3f2c81138cc7ac758aad9f48668d6abec15c83
local      dbdata
local      e41b528671a605801c0a1cafb25e672edd5b953f115a78696f86e9396c8519ba
local      ef4bb2aecde70c04c27061851b68adc83f26b5513a3db7db6fbaab09362f036f
local      f4a5a8c5388c7398d68e78b71d00beb7cc6211dc31680417a8876de15e40cfde
local      f10044a633b4a9994fbca4cfcb6ce5d4e7b66c429b1b9fbdf593ba91c5160738
local      mongodata
```

Una vez creado el volumen, se monta dicho volumen asignando los directorios fuente (src) y destino (dst) al mismo tiempo que se ejecuta el contenedor :

docker run -d --name db --mount src=dbdata,dst=/data/db mongo

```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name db --mount src=dbdata,dst=/data/db mongo
ede7bb69615a461b93dd901817eb9fdd5d2fadd560f0edad328a1bbe0d9329b8
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          PORTS          NAMES
ede7bb69615a   mongo      "docker-entrypoint.s..."   17 seconds ago   Up 16 seconds   27017/tcp   db
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Podemos ver más detalles de nuestro contenedor escribiendo: docker inspect db

```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker inspect db
[
  {
    "Id": "ede7bb69615a461b93dd901817eb9fdd5d2fadd560f0edad328a1bbe0d9329b8",
    "Created": "2024-10-24T04:34:24.781499521Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "mongod"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 1165,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-10-24T04:34:24.947994958Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "NetworkSettings": {
      "Bridge": "bridge",
      "ContainerID": "ede7bb69615a461b93dd901817eb9fdd5d2fadd560f0edad328a1bbe0d9329b8",
      "Gateway": "172.17.0.1",
      "GlobalIPv6": false,
      "GlobalIPv6Address": null,
      "GlobalIPv6PrefixLen": null,
      "HostBridge": false,
      "HostIP": null,
      "MacAddress": "02:42:AC:12:00:02",
      "NetworkMode": "bridge",
      "Ports": {},
      "SecondaryIPAddresses": [],
      "SecondaryIPv6Addresses": [],
      "StartIP": null,
      "IPv6": false
    }
  }
]
```

Detalles en el inspect

```
"Mounts": [
  {
    "Type": "volume",
    "Name": "dbdata",
    "Source": "/var/lib/docker/volumes/dbdata/_data",
    "Destination": "/data/db",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  },
  {
    "Type": "volume",
    "Name": "e70b5a1c7c26a4e8cd37d2b684c3557320a10abfe9a9df997fb7d1498e4db5ce",
    "Source": "/var/lib/docker/volumes/e70b5a1c7c26a4e8cd37d2b684c3557320a10abfe9a9df997fb7d1498e4db5ce/_data",
    "Destination": "/data/configdb",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
],
```

Entramos a mongodb usando mongosh después de usar: docker exec

```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it db bash
root@ede7bb69615a:/# mongosh
Current Mongosh Log ID: 6719cff73a8511ebb0fe6910
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=5000
Using MongoDB:          8.0.1
Using Mongosh:          2.3.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2024-10-24T04:34:25.688+00:00: Using the XFS filesystem is strongly recommended with the Wired
-filesystem
2024-10-24T04:34:26.316+00:00: Access control is not enabled for the database. Read and write
2024-10-24T04:34:26.316+00:00: For customers running the current memory allocator, we suggest
2024-10-24T04:34:26.316+00:00: We suggest setting the contents of sysfsFile to 0.
2024-10-24T04:34:26.316+00:00: Your system has glibc support for rseq built in, which is not y
cations. Please set the environment variable GLIBC_TUNABLES=glibc.pthread.rseq=0
2024-10-24T04:34:26.316+00:00: vm.max_map_count is too low
2024-10-24T04:34:26.316+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause

-----
```

test>

Añadimos algunos datos:

```
test> use prueba
switched to db prueba
prueba> db.users.insertOne({"nombre": "Nadie"})
{
  acknowledged: true,
  insertedId: ObjectId('6719d1263a8511ebb0fe6911')
}
prueba> db.users.find()
[ { _id: ObjectId('6719d1263a8511ebb0fe6911'), nombre: 'Nadie' } ]
prueba>
```

De nuevo, eliminamos el contenedor luego volvemos a ejecutar un nuevo contenedor sobre el mismo volumen creado previamente.

```
root@ede7bb69615a:/# exit
exit
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker rm -f db
db
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS      NAMES
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name db --mount src=dbdata,dst=/data/db mongo
3bba505937cd79eb4d85995ca24304b4383fb6753222cf3f0cd37caebe56e677
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it db bash
root@3bba505937cd:/# mongosh
```

```
test> use prueba
switched to db prueba
prueba> db.users.find()
[ { _id: ObjectId('6719d1263a8511ebb0fe6911'), nombre: 'Nadie' } ]
prueba> █
```

Insertar y extraer archivos de un contenedor

Vamos a crear un nuevo contenedor, al cual nosotros queremos copiar un archivo desde fuera de él

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ touch prueba.txt
```

Correr un ubuntu y le agregamos el tail para que quede activo:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker run -d --name copytest ubuntu tail -f /dev/null  
a4fbea94de4029c1a86fd558f74b58ba503f72059fe37e44dbcabebe0b9370936
```

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a4fbea94de40	ubuntu	"tail -f /dev/null"	About a minute ago	Up About a minute		copytest
3bba505937cd	mongo	"docker-entrypoint.s..."	2 hours ago	Up 2 hours	27017/tcp	db

docker exec -it copytest bash

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker exec -it copytest bash  
root@a4fbae94de40:/#
```

Creamos un directorio

```
root@a4fbae94de40:/# mkdir testing  
root@a4fbae94de40:/# exit  
exit
```

Se quiere crear una copia del archivo prueba.txt y pasarlo al contenedor.

```
docker cp prueba.txt copytest:/testing/test.txt
```

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker cp prueba.txt copytest:/testing/test.txt  
Successfully copied 1.54kB to copytest:/testing/test.txt  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Se copia el archivo dentro del contenedor

docker cp copytest:/testing localtesting

Copio el directorio de un contenedor a mi máquina, con "docker cp" no hace falta que el contenedor esté corriendo.

```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ docker cp copytest:/testing localtesting
Successfully copied 2.05kB to /mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata/localtesting
```

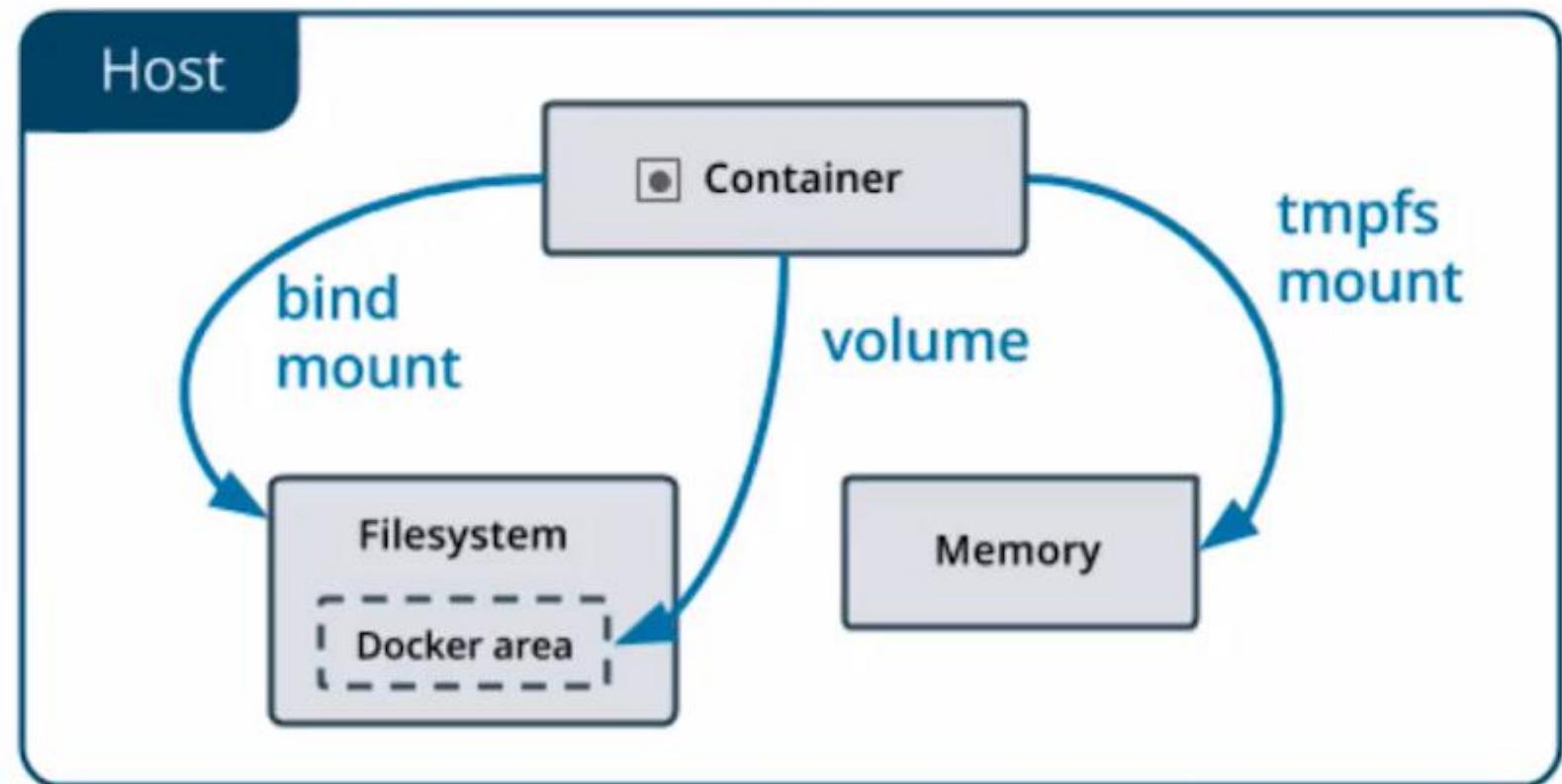
```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$ ls
localtesting  mongodata  prueba.txt
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/dockerdata$
```

Host: Donde Docker está instalado.

Bind Mount: Guarda los archivos en la maquina local persistiendo y visualizando estos datos (No seguro).

Volume: Guarda los archivos en el area de Docker donde Docker los administra (Seguro).

TMPFS Mount: Guarda los archivos temporalmente y persiste los datos en la memoria del contenedor, cuando muera sus datos mueren con el contenedor.



Ejemplo 3

1. Crear un contenedor de Docker que ejecute Apache Spark.
2. Cargar un archivo de datos sencillo (como un CSV).
3. Ejecutar un trabajo de procesamiento de datos dentro del contenedor.

1. Creación del Dockerfile

- Crea una carpeta para el proyecto:

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker$ mkdir docker_spark_bigdata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker$ cd docker_spark_bigdata  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/docker_spark_bigdata$ █
```

```
# Usar una imagen oficial de OpenJDK como base, necesario para Spark  
FROM openjdk:8-jdk-alpine  
  
# Instalar wget para descargar Spark  
RUN apk add --no-cache wget bash  
  
# Descargar Apache Spark (versión 3.5.0)  
RUN wget https://archive.apache.org/dist/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz  
  
# Descomprimir Spark  
RUN tar -xzf spark-3.5.0-bin-hadoop3.tgz  
  
# Mover Spark al directorio adecuado  
RUN mv spark-3.5.0-bin-hadoop3 /usr/local/spark  
  
# Establecer variables de entorno  
ENV SPARK_HOME=/usr/local/spark  
ENV PATH=$SPARK_HOME/bin:$PATH  
  
# Definir el comando para iniciar Spark Shell  
CMD ["spark-shell"]
```

Dentro de la carpeta creamos el Dockerfile y añadimos el respectivo contenido. El código lo tienes en tu bloc de notas.

Este Dockerfile realiza los siguientes pasos:

- Usa una imagen de OpenJDK porque Apache Spark requiere Java.
- Instala Apache Spark 3.3.1 (una versión reciente compatible con Hadoop).
- Configura las variables de entorno necesarias para que Spark funcione.
- El comando por defecto (CMD) es abrir el Spark Shell, un entorno interactivo donde puedes escribir y ejecutar comandos en Spark.

2. Crear un archivo de datos (CSV)

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker$ mkdir data  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker$ touch data/data.csv  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker$ █
```

- Dentro del archivo data.csv añadimos el siguiente contenido:

```
data > x data.csv > data  
1 id,nombre,edad  
2 1,Juan,28  
3 2,Ana,35  
4 3,Luis,40  
5 4,Carlos,22  
6 █
```

3. Construir la imagen de Docker

docker build -t spark_bigdata .

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/docker_spark_bigdata$ docker build -t spark_bigdata .
```

Verificar la imagen en tu Docker local

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/docker_spark_bigdata$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
spark_bigdata	latest	7b29f635462c	About a minute ago	1.4GB
miport	latest	8d737369c9fd	34 hours ago	47MB
ubuntu	latest	59ab366372d5	13 days ago	78.1MB
mongo	latest	d3295cd2d11f	2 weeks ago	854MB

4. Ejecutar el contenedor y procesar los datos

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/docker_spark_bigdata$ docker run -it -v $(pwd)/data:/data spark_bigdata
```

Si el contenedor está en funcionamiento, debería abrirse el Spark Shell. Desde allí, puedes escribir comandos en Scala para procesar los datos.

```
Welcome to
      _/\_
     /  \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \
    / \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \
   / \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \
  / \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \
 / \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \_/\_ \
version 3.5.0

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 1.8.0_212)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

5. Procesamiento de datos en Spark

5.1-Cargar el archivo CSV en un DataFrame de Spark: `val data = spark.read.option("header", "true").csv("/data/data.csv")`

```
scala> val data = spark.read.option("header", "true").csv("/data/data.csv")
data: org.apache.spark.sql.DataFrame = [id: string, nombre: string ... 1 more field]

scala>
```

5.2- Mostrar el contenido del DataFrame: `data.show()`

```
scala>

scala> data.show()
+---+-----+---+
| id|nombre|edad|
+---+-----+---+
|  1|   Juan|  28|
|  2|    Ana|  35|
|  3|   Luis|  40|
|  4|Carlos|  22|
+---+-----+---+
```

5.3 - Filtrar los datos donde la edad sea mayor a 30:

```
scala> val mayoresDe30 = data.filter(data("edad") > 30)
mayoresDe30: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: string, nombre: string ... 1 more field]

scala> mayoresDe30.show()
+---+-----+
| id|nombre|edad|
+---+-----+
|  2|  Ana|  35|
|  3| Luis|  40|
+---+-----+
```

Mismo ejemplo, pero en modo Clúster...

Vamos a:

- Desplegar un clúster de Apache Spark distribuido utilizando Docker.
- Ejecutar un trabajo de procesamiento de datos en el clúster con varios nodos de Spark (usando varios contenedores)

Definimos la arquitectura del clúster de Spark:

Vamos a desplegar:

1. Un contenedor **master** de Spark, que coordina el procesamiento.
2. Varios contenedores **workers** de Spark, que realizan el procesamiento de los datos.

1. Crear el archivo docker-compose.yml

- Crear una carpeta para el proyecto
- Vamos a crear el archivo docker-compose.yml para definir un maestro y varios trabajadores (workers).

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ touch docker-composer.yml  
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ █
```

El contenido del archivo docker-compose.yml lo tienes en tu bloc de notas.

Este archivo define un clúster con:

1 maestro (master) que coordina el procesamiento.

2 trabajadores (workers) que procesarán los datos.

2. Levantar el clúster de Spark

2.1 - Ejecutar el comando docker-compose. Desde el directorio donde está el archivo docker-compose.yml ejecuta el siguiente comando en la terminal para levantar el clúster:

```
docker-compose up -d
```

```
balrodrju@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ docker-compose up -d
Creating spark-master ... done
Creating spark-worker-1 ... done
Creating spark-worker-2 ... done
balrodrju@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ █
```

- docker-compose: Permite definir y ejecutar aplicaciones multicontenedor utilizando un archivo Docker-compose.yml
- up: construye, crea, inicia y adjunta contenedores para un servicio definido en el archivo docker-compose.yml
- -d : ‘Detached mode’ ejecuta los contenedores en segundo plano, permitiendo seguir utilizando la terminal para otros comandos

2.2 Verificar los contenedores en ejecución

docker ps

```
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS        PORTS          NAMES
95764d464a03   bitnami/spark:latest "/opt/bitnami/script..."  11 minutes ago  Up 11 minutes  0.0.0.0:8082->8082/tcp  spark-worker-2
109ac6781c41   bitnami/spark:latest "/opt/bitnami/script..."  11 minutes ago  Up 11 minutes  0.0.0.0:8081->8081/tcp  spark-worker-1
9676e0e88745   bitnami/spark:latest "/opt/bitnami/script..."  11 minutes ago  Up 11 minutes  0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp  spark-master
balrodrjuan@balrodrjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ █
```

3 Crear un script Python con múltiples operaciones de procesamiento:

El script de Python debe realizar varias operaciones en un clúster de Spark:

- Sumar los elementos de una lista.
- Filtrar los elementos.
- Calcular el promedio.
- Agrupar elementos por valores.

3.1 Crear el archivo Python

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ touch distributed_operations.py
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ ls
distributed_operations.py  docker-compose.yml
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ █
```

El contenido del fichero lo encuentras en tu bloc de notas

4. - Copiar el script al contenedor maestro.

Para ejecutar el script en el clúster de Spark, necesitamos copiar el archivo distributed_operations.py al contenedor del maestro.

4.1 - Usar docker cp para copiar el archivo:

```
docker cp /ruta/a/tu/archivo/distributed_operations.py spark-master:/opt/bitnami/spark/
```

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ docker cp /mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark/distributed_operations.py spark-master:/opt/bitnami/spark/
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ █
```

5.- Ejecutar el script en el clúster de Spark

Una vez que el archivo esté copiado al maestro, puedes ejecutarlo utilizando spark-submit.

5.1 – Conectarse al contenedor maestro: docker exec -it spark-master bash

```
balrodjuan@balrodjj:/mnt/c/Users/juanj/OneDrive/Escritorio/ejs-docker/ej2_bigdata_spark$ docker exec -it spark-master bash
I have no name!@9676e0e88745:/opt/bitnami/spark$ █
```

Esto te dará acceso a la terminal dentro del contenedor maestro.

5.2 Ejecutar el script con spark-submit

Ahora que estás dentro del contenedor maestro, navega al directorio donde copiaste el archivo y ejecuta el script:

```
cd /opt/bitnami/spark/
spark-submit --master spark://spark-master:7077 distributed_operations.py
```

```
spark-submit --master spark://spark-master:7077 distributed_operations.py
```

Este comando enviará el trabajo al maestro, quien distribuirá las tareas entre los trabajadores.

```
I have no name!@9676e0e88745:/opt/bitnami/spark/ej2_bigdata_spark$ spark-submit --master spark://spark-master:7077 /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py
```

6.- Ver los resultados

```
24/10/24 20:17:00 INFO DAGScheduler: Job 0 finished: reduce at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:11, took 10.120214 s
Suma de los elementos: 55
24/10/24 20:17:00 INFO SparkContext: Starting job: collect at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:16
24/10/24 20:17:02 INFO DAGScheduler: Job 1 finished: collect at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:16, took 1.964528 s
Elementos mayores que 5: [6, 7, 8, 9, 10]
24/10/24 20:17:02 INFO SparkContext: Starting job: count at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:19

24/10/24 20:17:03 INFO DAGScheduler: Job 2 finished: count at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:19, took 0.395423 s
Promedio de los elementos: 5.5
24/10/24 20:17:03 INFO SparkContext: Starting job: collect at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:25

24/10/24 20:17:05 INFO DAGScheduler: Job 3 finished: collect at /opt/bitnami/spark/ej2_bigdata_spark/distributed_operations.py:25, took 1.636516 s
Grupo False: [1, 3, 5, 7, 9]
Grupo True: [6, 8, 10, 2, 4]
24/10/24 20:17:05 INFO SparkContext: SparkContext is stopping with exitCode 0.
```













