ESPECIALIZACIÓN EN MACHINE LEARNING ENGINEER

Tema: Fundamentos de MLE – parte 2

Docente: Daniel Chavez Gallo

Git y GitHub

Git es un sistema de **control de versiones distribuido** y de **código abierto**. Fue creado por Linus Torvalds en **2005**. Git facilita a los desarrolladores el **seguimiento de los cambios** en el código a lo largo del tiempo, lo que permite múltiples versiones, colaboración y resolución de conflictos entre versiones de archivos.

GitHub es una **plataforma de alojamiento de código** basada en la web que **utiliza Git** para el control de versiones. Fue lanzado en **2008** por Chris Wanstrath, PJ Hyett y Tom Preston-Werner. Aunque Git es la tecnología que impulsa GitHub, la **plataforma añade muchas características propias.**

Mientras que **Git es una herramienta de control de versiones** que permite a los usuarios manejar y controlar la evolución de sus proyectos de código, **GitHub es un servicio en la nube que aprovecha Git** y lo amplía para facilitar la colaboración y el alojamiento de código entre usuarios y equipos. **Git es la herramienta que usas en tu entorno local para hacer control de versiones**, y **GitHub es la plataforma en línea donde subes tu código para colaborar con otros.**

GitHub: Introducción

Descripción

GitHub es una plataforma de **alojamiento de código que utiliza Git**, un sistema de control de versiones distribuido, para facilitar el seguimiento y la colaboración en proyectos de software y de ingeniería de machine learning. Git fue diseñado para **manejar desde pequeños a grandes proyectos** con velocidad y eficiencia.

GitHub extiende la funcionalidad de Git al proporcionar una **interfaz web que mejora la visualización y gestión de los repositorios de código.**

Importancia

Gestión de Versiones: Permite a los equipos **mantener un historial completo de cambios**, comparar versiones y revertir a estados anteriores del código cuando es necesario.

Colaboración: Facilita la **colaboración entre desarrolladores** al permitir que múltiples personas trabajen en el mismo proyecto sin interferencias, mediante el uso de branches y pull requests.

Revisión de Código: Los **Pull Requests** proporcionan una plataforma para la **revisión de código**, permitiendo que otros colaboradores comenten, sugieran cambios y aprueben el código antes de que se fusione con la rama principal.

CI/CD: **GitHub Actions** y otros servicios integrados permiten la automatización de pipelines de **Continuous Integration (CI)** y **Continuous Deployment (CD)**, esenciales para la entrega continua y la integración de nuevas características y correcciones de forma segura y eficiente.

GitHub: Conceptos

Repositorio

Un repositorio en GitHub es un **contenedor de almacenamiento centralizado** para proyectos que almacena todos los archivos relacionados con un proyecto (código, documentación, configuraciones, etc.) y **registra cada cambio** realizado a estos archivos. Actúa como un **control de versiones** que ayuda a rastrear el progreso del proyecto a lo largo del tiempo.

Commits

Un commit en Git **representa un conjunto de cambios guardados** en el repositorio. Es como una instantánea del estado actual de un proyecto, que incluye los cambios realizados en los archivos desde el último commit.

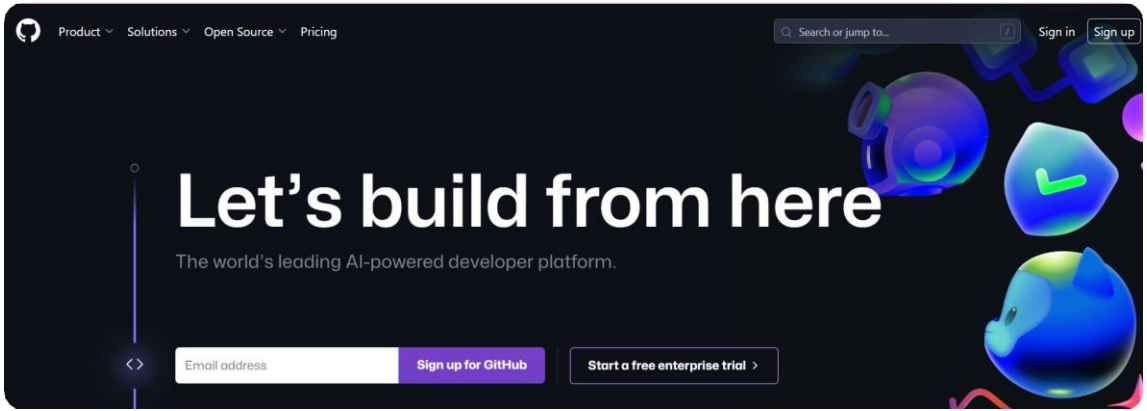
Branches

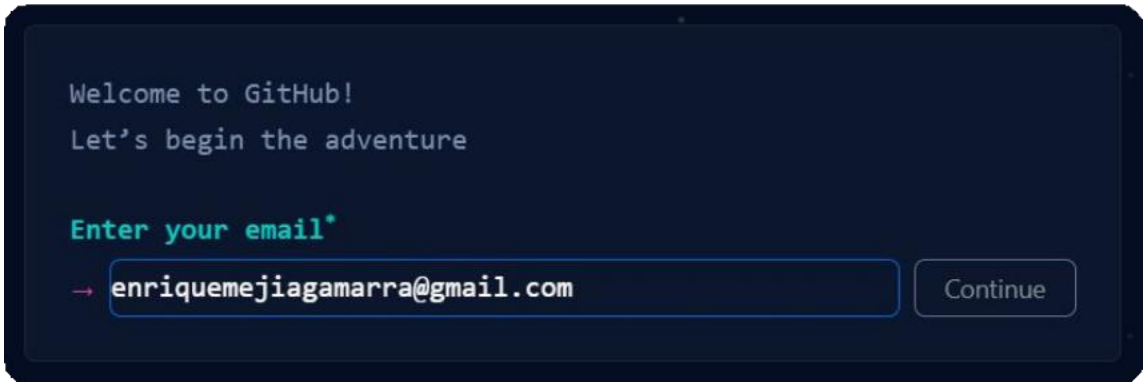
En Git, un branch o rama **permite a los desarrolladores trabajar en diferentes versiones** de un repositorio al mismo tiempo. Cada branch representa una **línea independiente de desarrollo.**

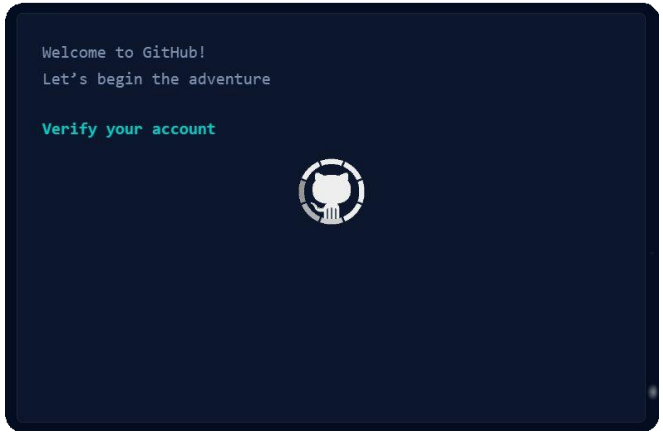
Pull Requests (PRs)

Una pull request es una **solicitud para fusionar cambios de un branch a otro** dentro de un repositorio de GitHub. Es una parte crucial del proceso de colaboración y revisión de código.

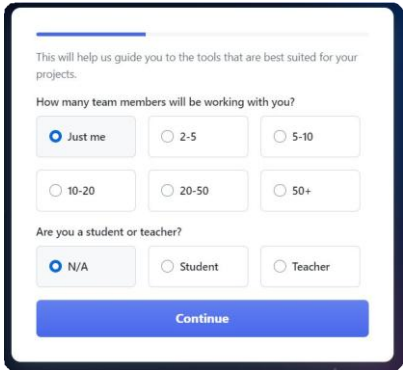
GitHub: Crear cuenta

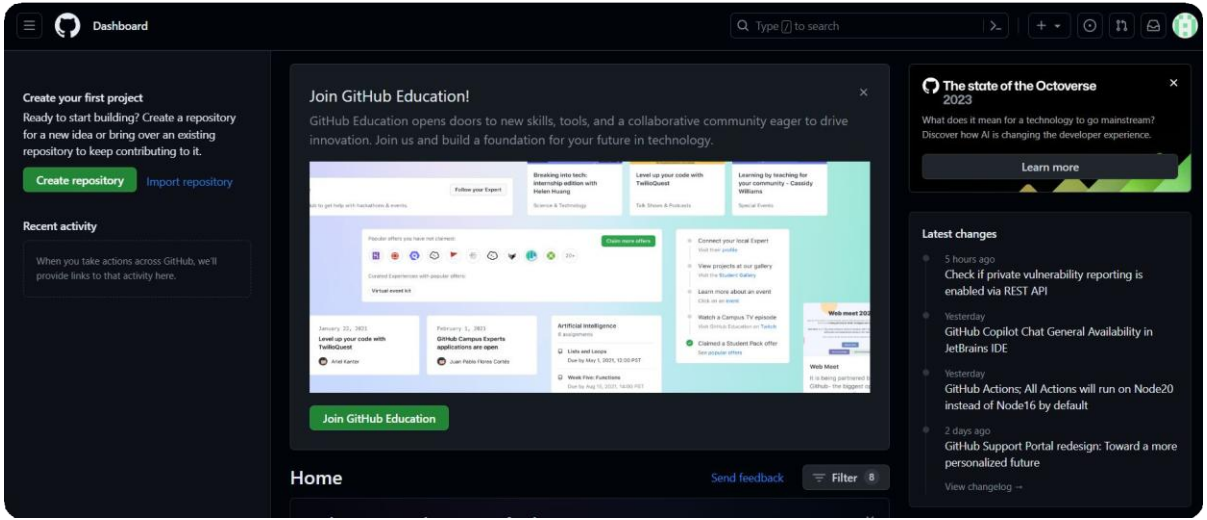






GitHub: Crear cuenta





Git: Configuración inicial

https://www.git-scm.com/downloads

```
PS D:\DMC> git --version
git version 2.39.0.windows.2
PS D:\DMC> git config --global user.name "José Mejía"
PS D:\DMC> git config --global user.email enriquemejiagamarra@gmail.com
PS D:\DMC> git config --global core.editor "code --wait"
PS D:\DMC> git config --global -e
hint: Waiting for your editor to close the file...
```

```
PS D:\DMC> git config --global credential.username "EnriqueMejia96"
PS D:\DMC> git config --global credential.useremail "enriquemejiagamarra@gmail.com"
```

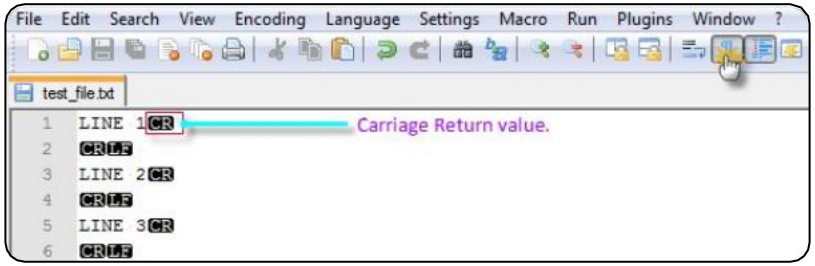
.gitconfig

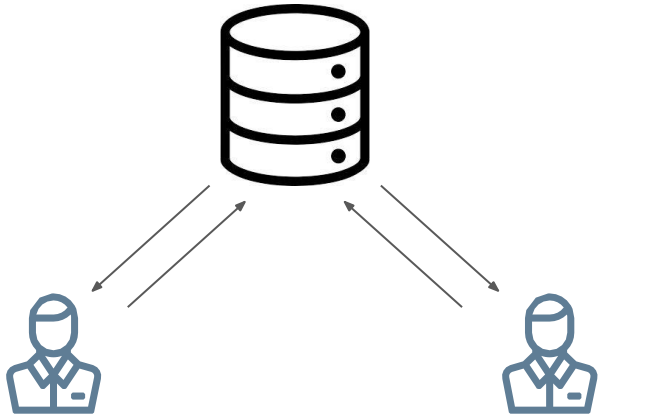
C: > Users > usuario > .gitconfig

```
1   [user]
2       name = José Mejía
3       email = enriquemejiagamarra@gmail.com
4   [credential]
5       helper = store
6   [core]
7       editor = code --wait
8
```

Git: Configuración inicial

¿Cómo git trata los saltos de línea?



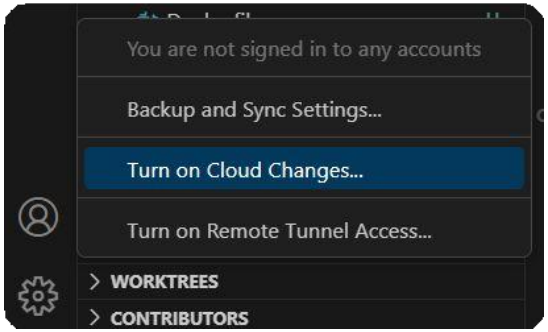


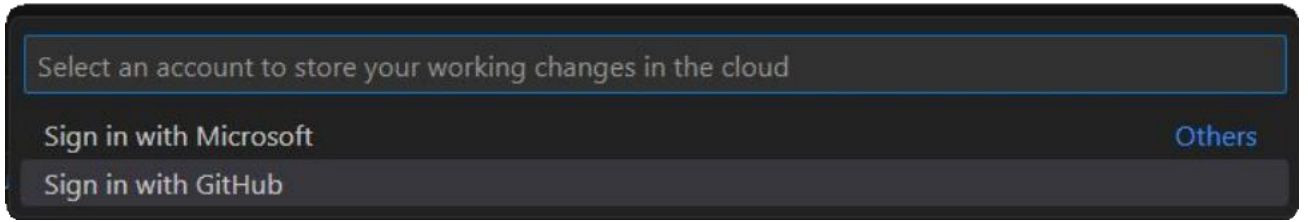

```
git config --global core.autocrlf true
```

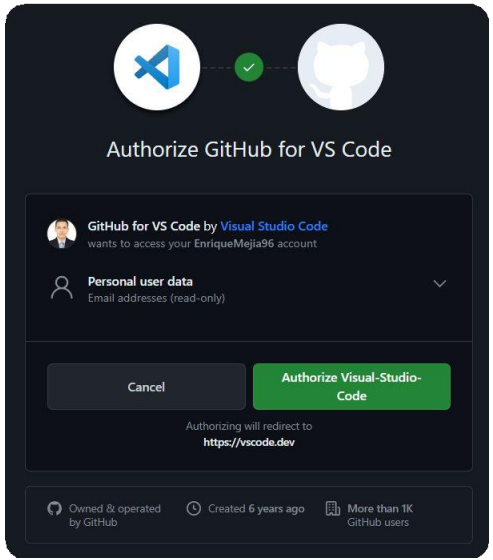
```
git config --global core.autocrlf input
```

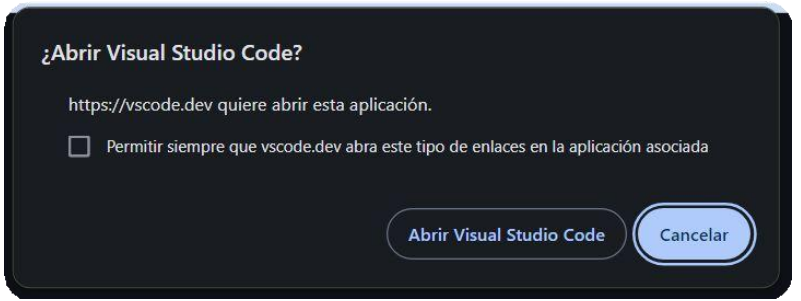
LF: Line Feed

GitHub: Iniciar sesión - VS Code

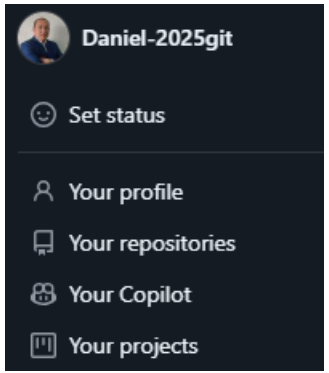


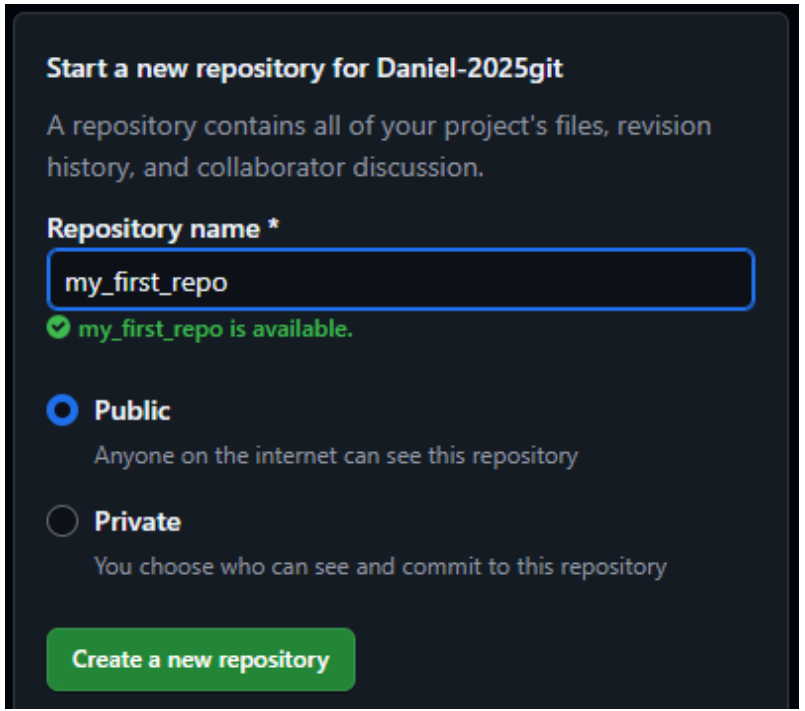




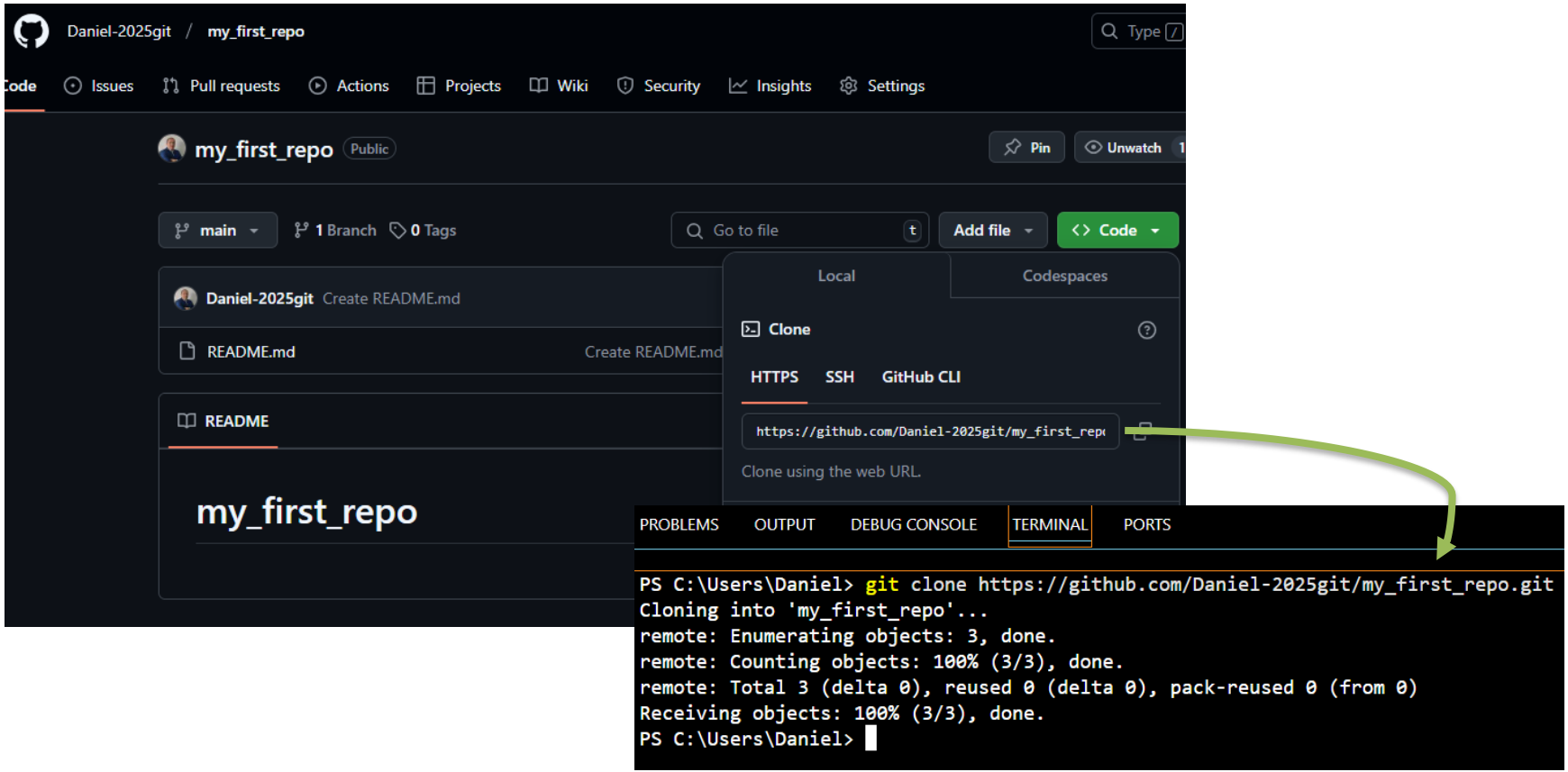


GitHub: Crear repositorio

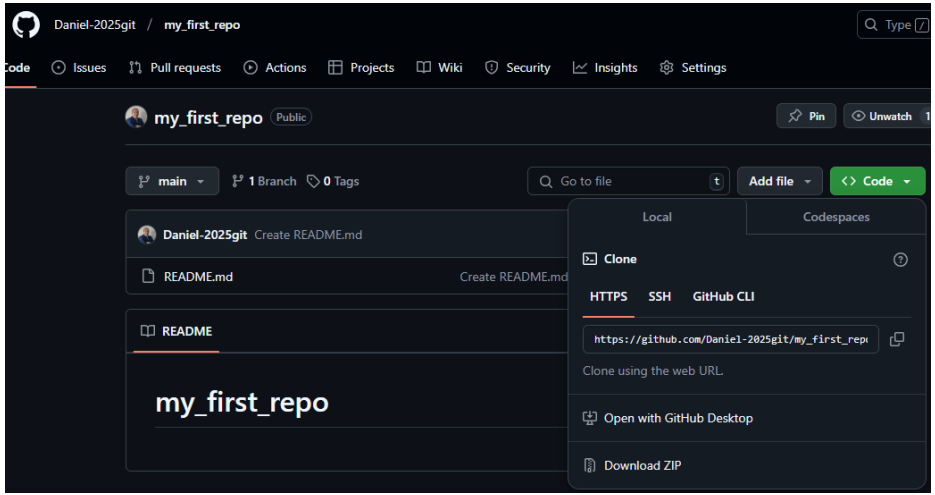


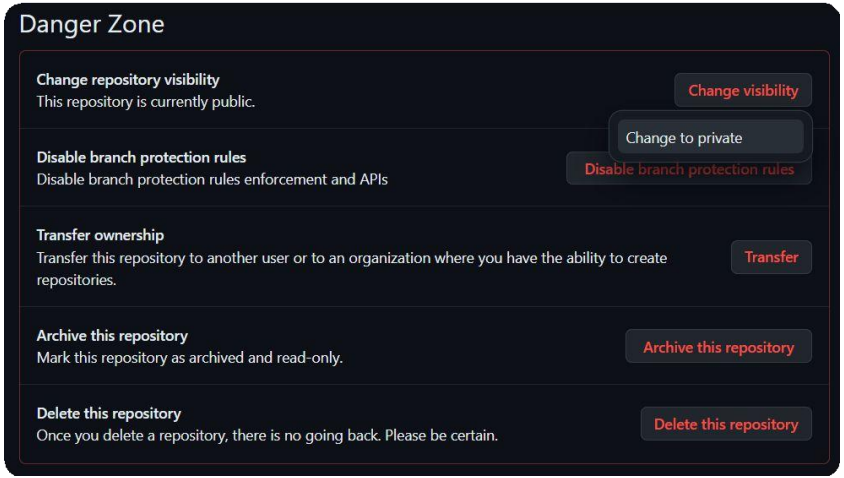


GitHub: Clonación de repositorio

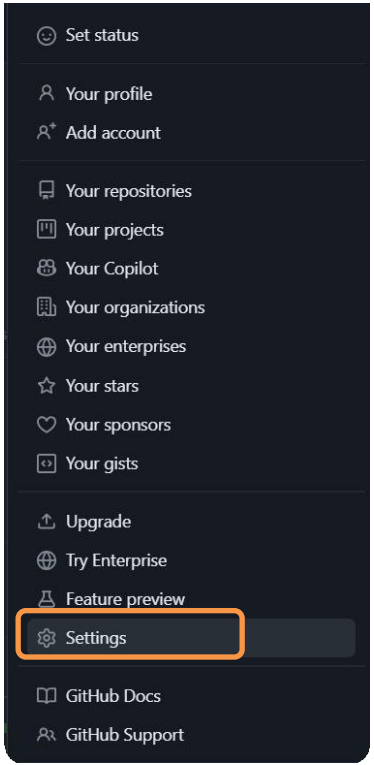


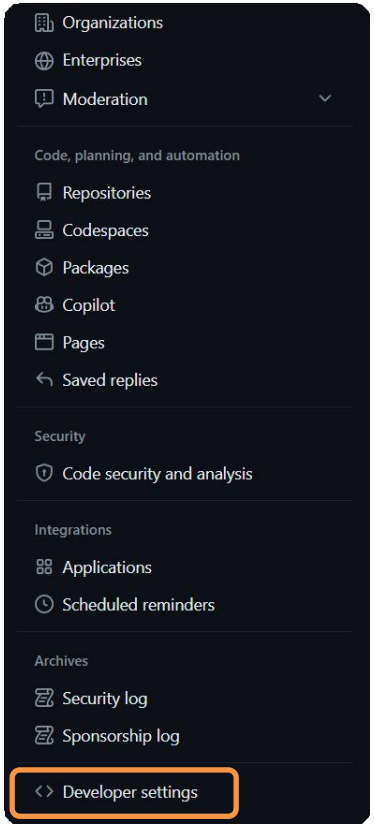
GitHub: Repositorios privados

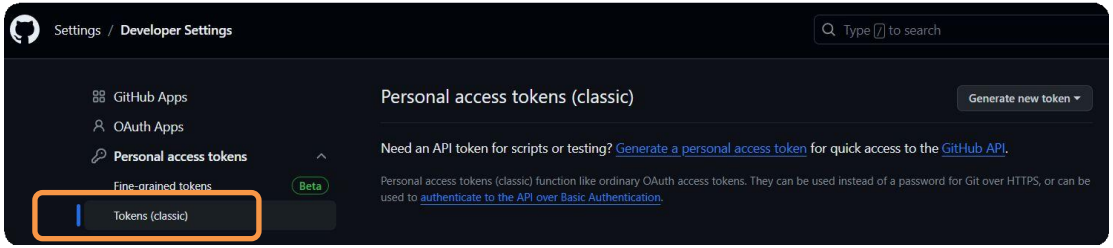


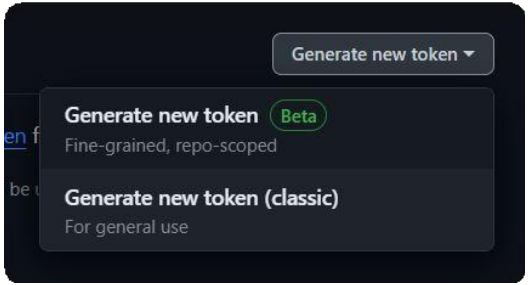


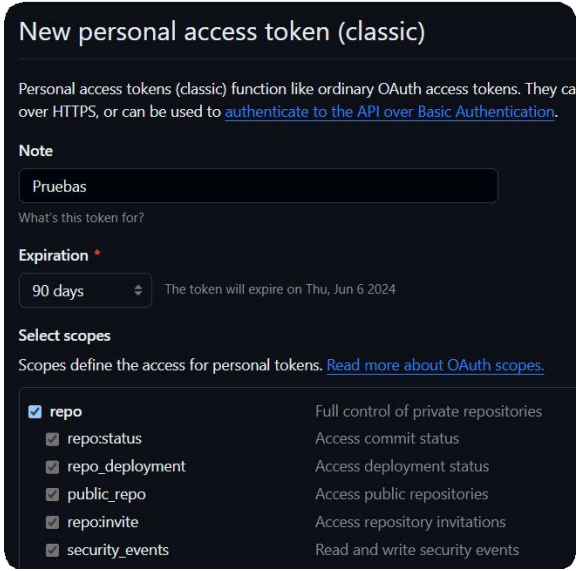
GitHub: Repositorios privados



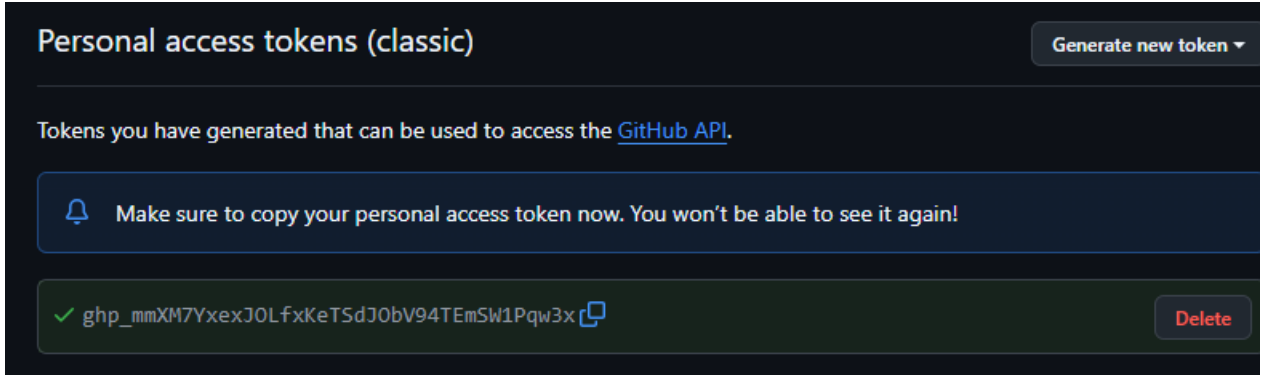








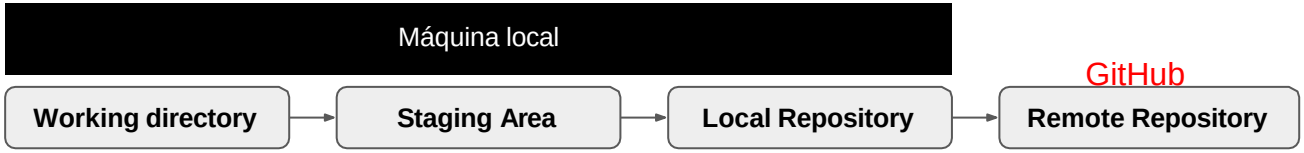
GitHub: Repositorios privados



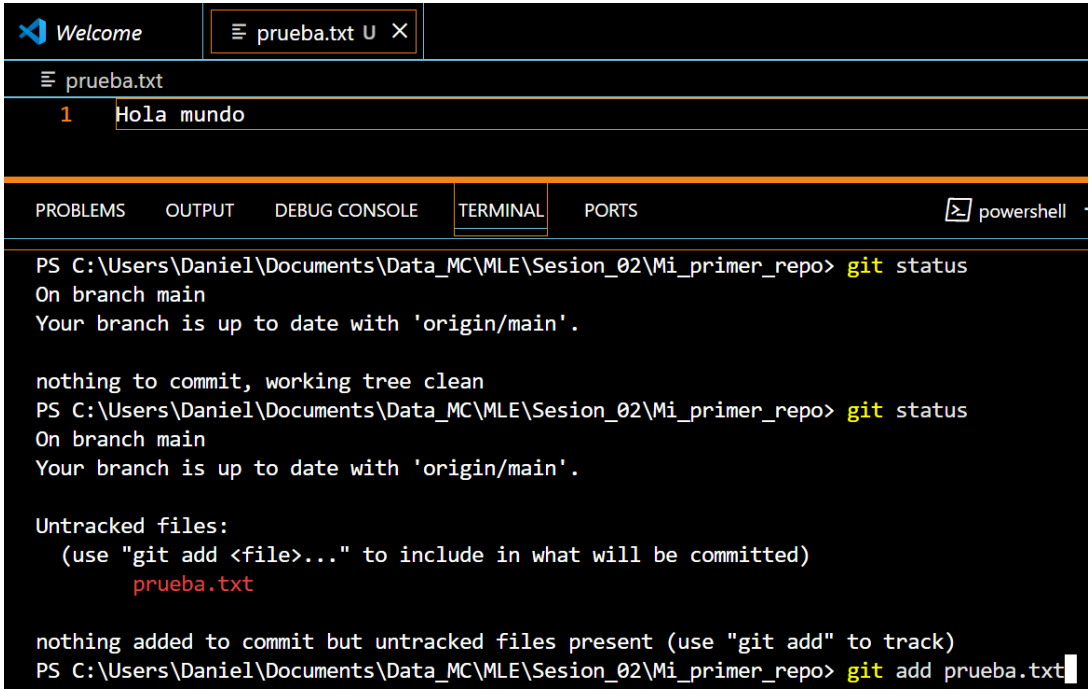

```
git clone https://<token>:x-oauth-basic@github.com/usuario/repositorio.git
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02> git clone https://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:x-oauth-basic@github.com/
Daniel-2025git/Mi_primer_repo.git
Cloning into 'Mi_primer_repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02>
```

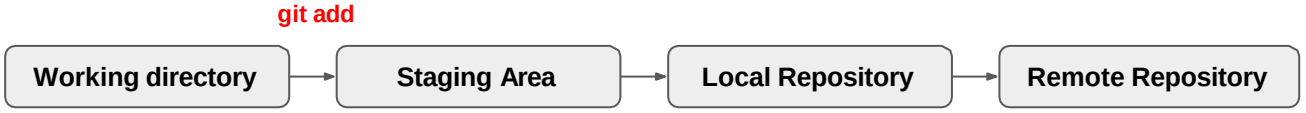
Git: Comandos



git status: Muestra el **estado de los archivos en tu directorio de trabajo** y el **área de preparación** (staging area). Esto te ayuda a ver si los **archivos han sido modificados**, si son nuevos y no están siendo rastreados por Git, o si han sido agregados al área de preparación y están listos para ser confirmados (committed).



Git: Comandos



git add: El comando git add es una parte crucial del flujo de trabajo en Git y se utiliza para **añadir cambios en el directorio de trabajo al área de preparación** (también conocida como el índice o staging area). Este proceso prepara los cambios para el **próximo commit,** permitiendo que Git sepa qué modificaciones considerar para la siguiente instantánea del proyecto.

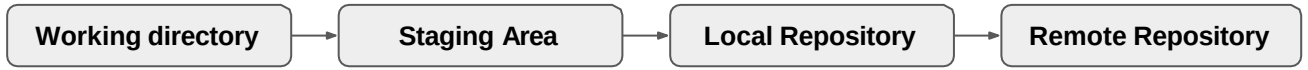
```
git add prueba.txt
```

```
git add *.txt
```

```
git add .
```

```
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git add prueba.txt
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   prueba.txt

PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo>
```

Git: Comandos




```
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   prueba.txt

PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   prueba.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        prueba2.txt

PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo>
```

Git: Comandos

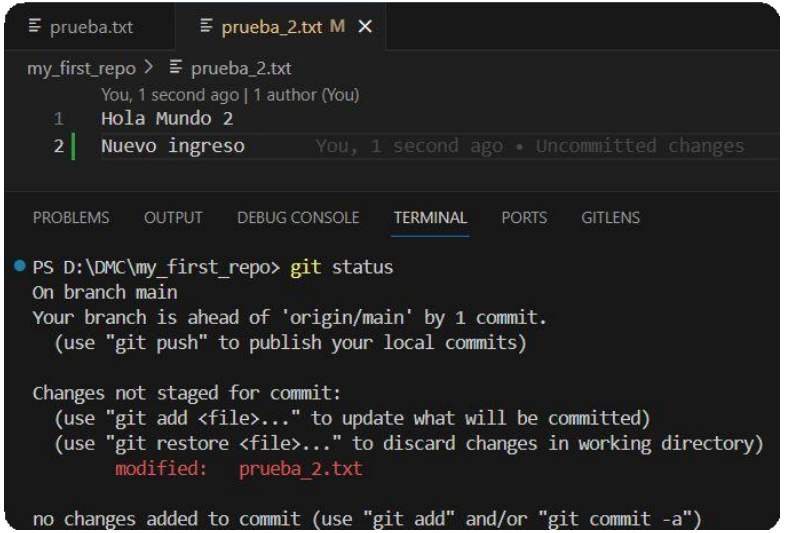
git commit

Working directory → Staging Area → Local Repository → Remote Repository

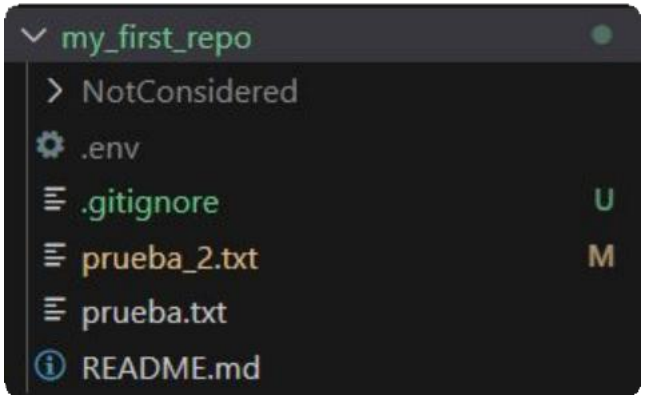
git commit: Este comando se utiliza para **guardar los cambios que ha hecho en el repositorio local**. Antes de ejecutar git commit, debe agregar los cambios con git add.

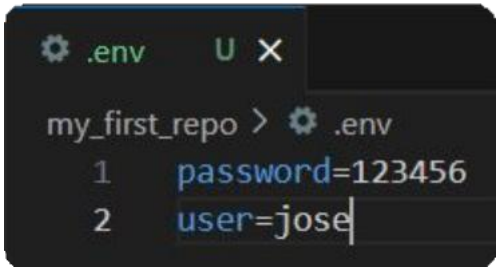
```
git commit -m "Commit inicial"  --> debe ser una descripción precisa
```

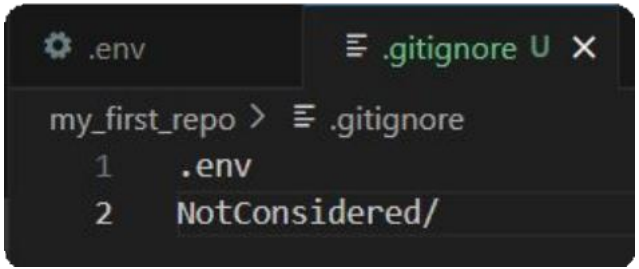
```
PS D:\DMC\my_first_repo> git commit -m 'Commit inicial'
[main 71caa4f] Commit inicial
 2 files changed, 2 insertions(+)
 create mode 100644 prueba.txt
 create mode 100644 prueba_2.txt
PS D:\DMC\my_first_repo> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```



Git: .gitignore







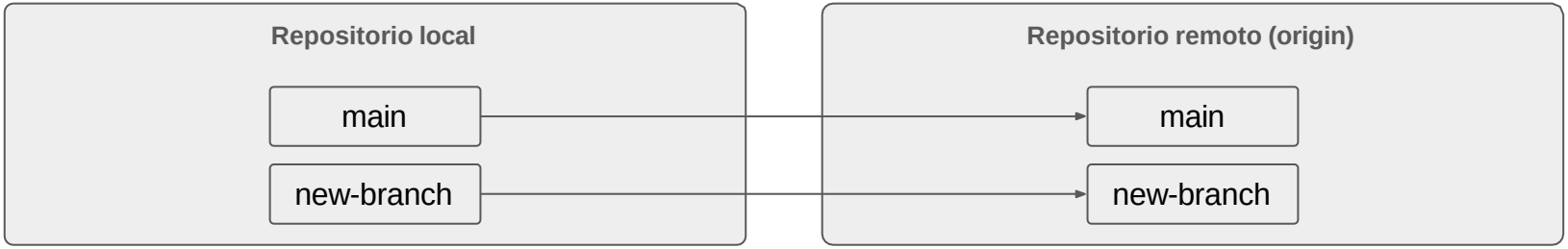
Su contenido son todos los archivos que no serán reflejados Dentro del repositorio remoto, es decir en el GitHub

```
PS D:\DMC\my_first_repo> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   prueba_2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
PS D:\DMC\my_first_repo> git add .
PS D:\DMC\my_first_repo> git commit -m '.gitignore'
[main 612ccef] .gitignore
 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 .gitignore
```

Git: Comandos

git push: Subir cambios asociando rama local con rama de repositorio

```
git push --set-upstream origin new-branch
```




```
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (7/7), 665 bytes | 166.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Daniel-2025git/Mi_primer_repo.git
   1a96e78..0fa8388  main -> main
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo>
```

Git: Comandos

git log: El comando git log --oneline es una variante del comando git log utilizado en Git para **mostrar el historial de commits de un repositorio.** Este comando proporciona una visión simplificada del historial, mostrando cada commit en una sola línea

```
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git log --oneline
0fa8388 (HEAD -> main, origin/main, origin/HEAD) add:gitignore & prueba.txt
3bd0719 add: new file prueba.txt
1a96e78 Initial commit
```

Git: Comandos

git branch: Cuando se ejecuta git branch sin argumentos adicionales, **enumera todas las ramas locales** en el repositorio actual. La rama en la que actualmente se encuentra marcada con un asterisco (*) y se muestra en un color diferente para resaltarla.

```
PS D:\DMC\my_first_repo> git branch
* main
```

Crear una rama (local)

```
git branch new-branch
```

```
PS D:\DMC\my_first_repo> git branch new-branch
PS D:\DMC\my_first_repo> git branch
* main
  new-branch
```

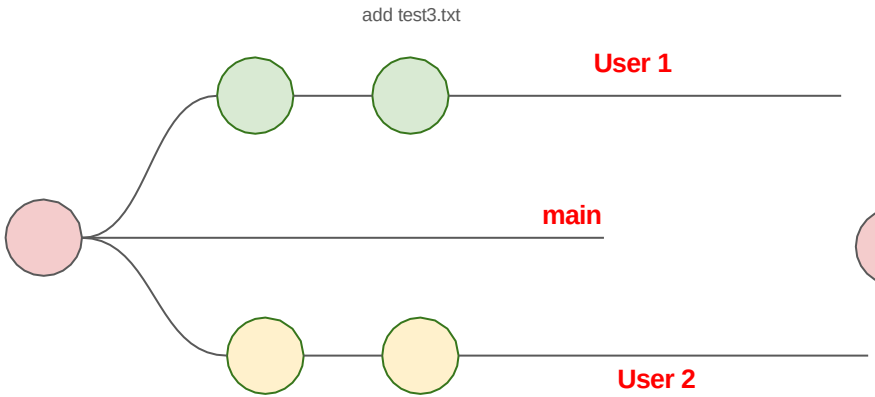
Cambiar a una rama existente

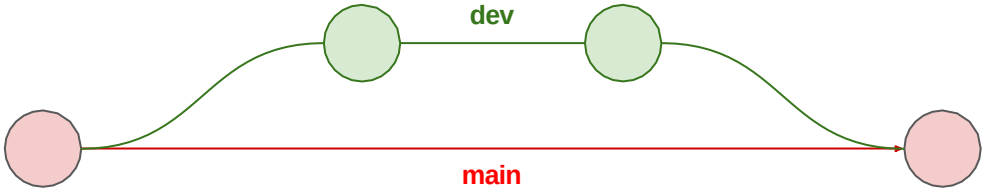
```
git switch new-branch (git checkout new-branch)
```

```
PS D:\DMC\my_first_repo> git switch new-branch
Switched to branch 'new-branch'
```

Git: Comandos

git branch: Cuando se ejecuta git branch sin argumentos adicionales, **enumera todas las ramas locales** en el repositorio actual. La rama en la que actualmente se encuentra marcada con un asterisco (*) y se muestra en un color diferente para resaltarla.





Git: Comandos

git merge: El comando git merge se utiliza en Git para **combinar los cambios de una rama (branch) en otra.** Generalmente, se utiliza para incorporar los cambios de una rama de desarrollo en la rama principal del proyecto (como puede ser main o master).

Agregar modificación en prueba.txt desde la rama “new-branch”

```
PS D:\DMC\my_first_repo> git add .
PS D:\DMC\my_first_repo> git commit -m 'modificación prueba.txt'
[new-branch 2d29b14] modificación prueba.txt
 1 file changed, 2 insertions(+), 1 deletion(-)
```

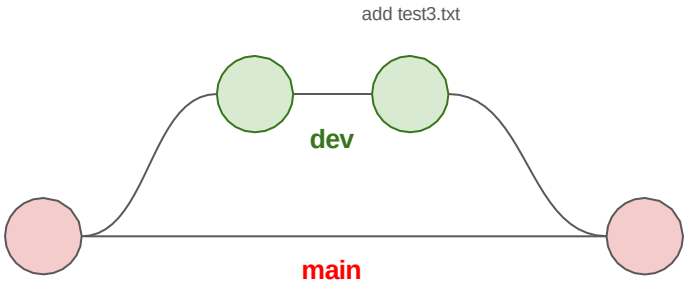
```
PS D:\DMC\my_first_repo> git branch
  main
* new-branch
PS D:\DMC\my_first_repo> cat prueba.txt
Hola mundo
ModificaciÃ³n
PS D:\DMC\my_first_repo> git switch main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)
PS D:\DMC\my_first_repo> git branch
* main
  new-branch
PS D:\DMC\my_first_repo> cat prueba.txt
Hola mundo
```

Combinar cambios de “new-branch” y “main”

```
PS D:\DMC\my_first_repo> git branch
* main
  new-branch
PS D:\DMC\my_first_repo> git merge new-branch
Updating 612ccef..2d29b14
Fast-forward
 prueba.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
PS D:\DMC\my_first_repo> cat prueba.txt
Hola mundo
ModificaciÃ³n
```

Git: Comandos

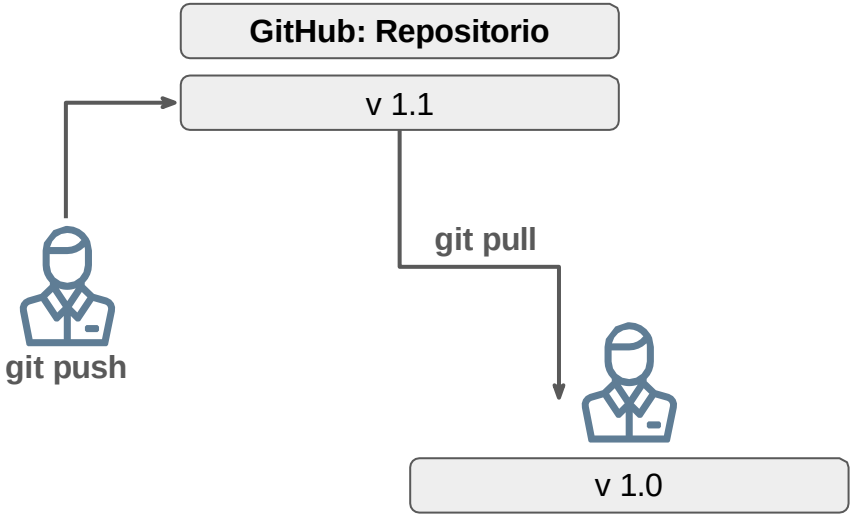
git merge: El comando git merge se utiliza en Git para **combinar los cambios de una rama (branch) en otra.** Generalmente, se utiliza para incorporar los cambios de una rama de desarrollo en la rama principal del proyecto (como puede ser main o master).

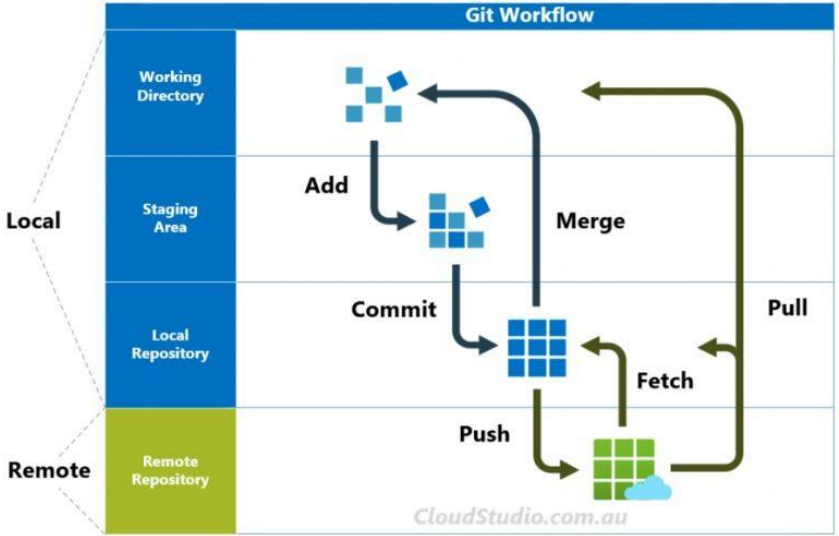


Git: Comandos

git pull: Se usa para **actualizar su repositorio local con los últimos cambios del repositorio remoto.** Esencialmente, git pull es una combinación de git fetch seguido de git merge.

```
git pull origin main
```

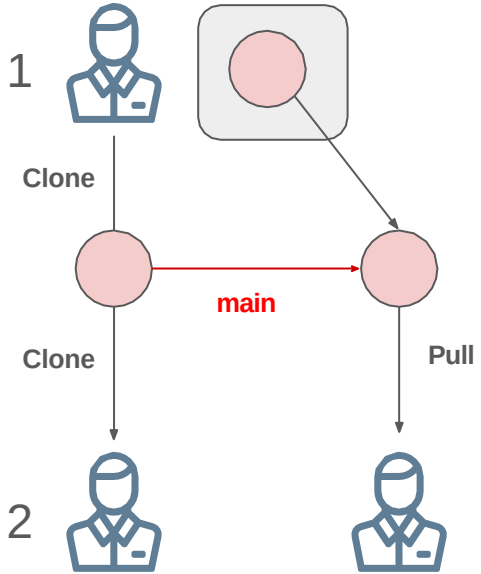




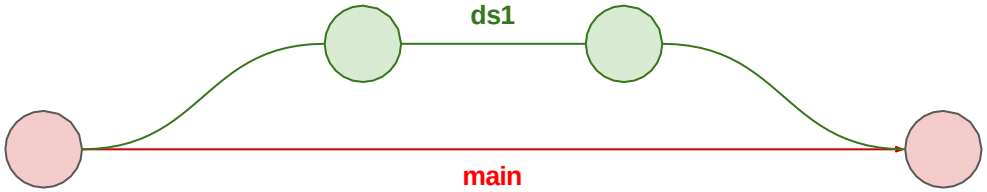
Git: Comandos

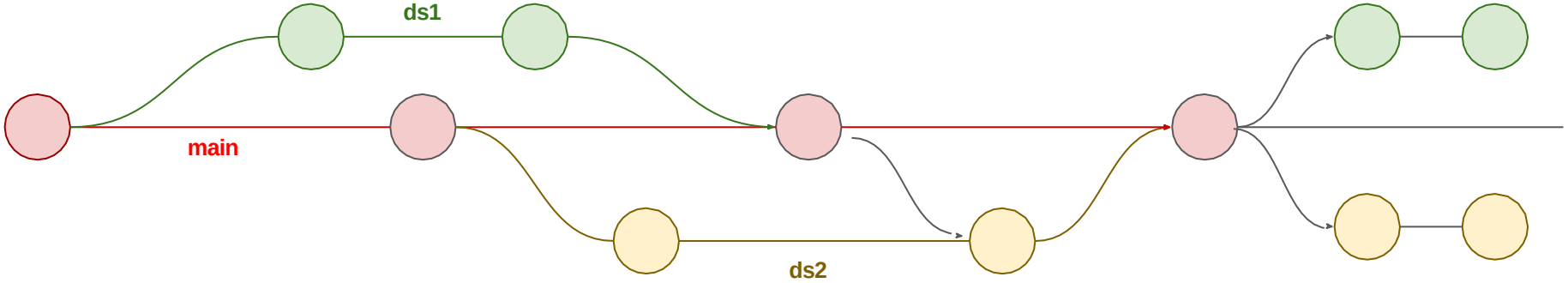
git pull: Se usa para **actualizar su repositorio local con los últimos cambios del repositorio remoto.** Esencialmente, git pull es una combinación de git fetch seguido de git merge.

```
git pull origin main
```



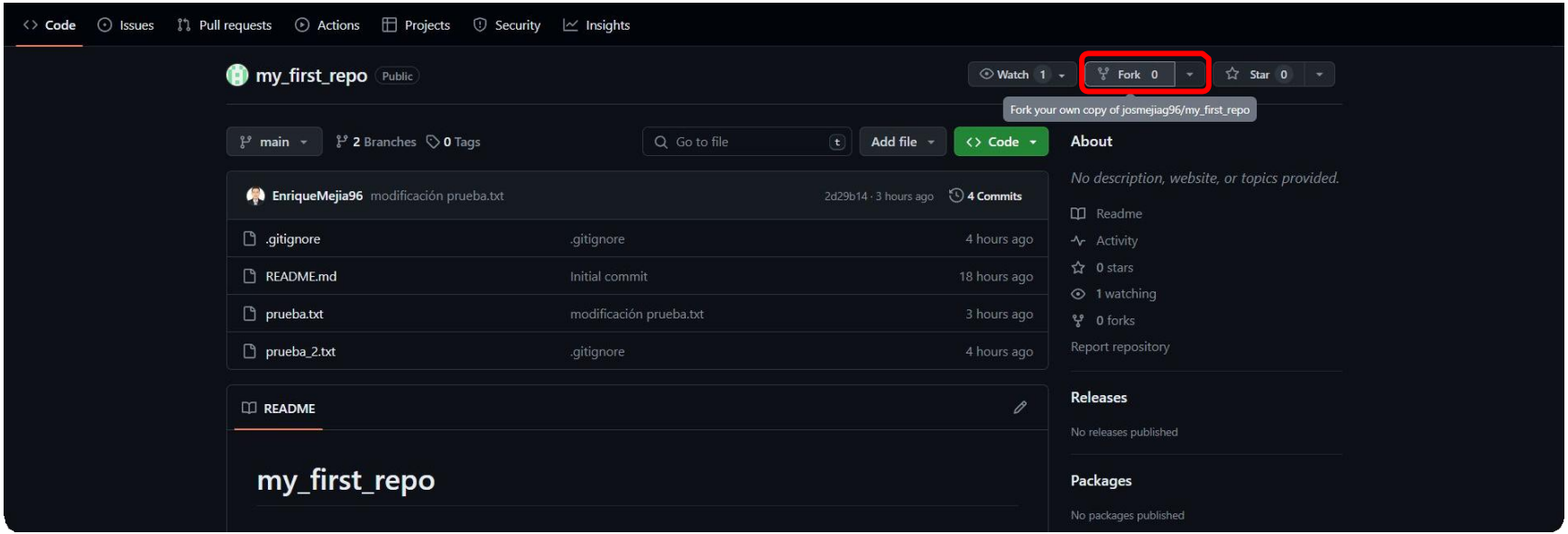
Git: Ejemplos ramas en git



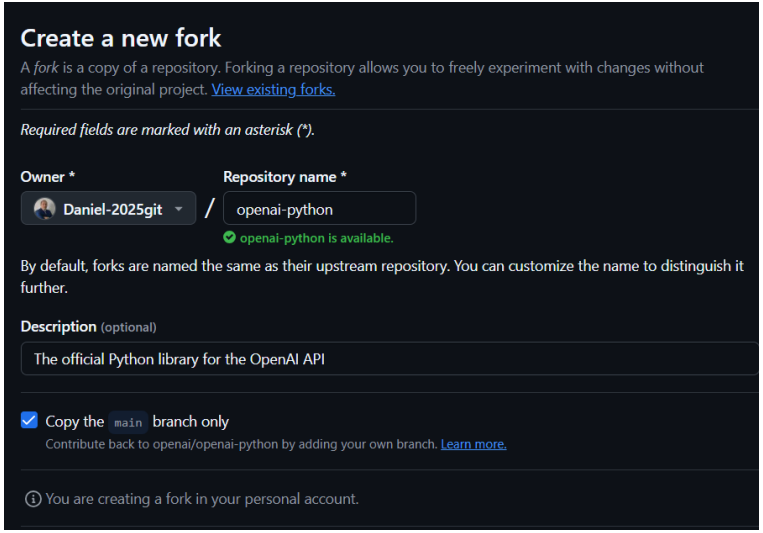


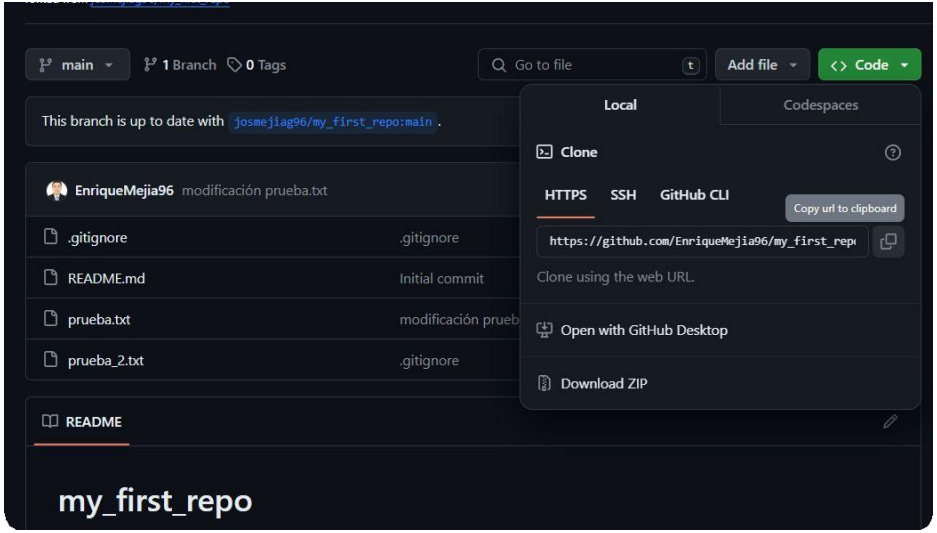
GitHub: Forks

Un “Fork” (bifurcación) es esencialmente una **copia personal del repositorio de otro usuario** que reside en su cuenta. Cuando bifurcas un repositorio, creas una instancia completamente separada de ese repositorio en tu perfil de GitHub. Este nuevo repositorio es idéntico al original en el momento de la bifurcación, pero **existe de forma independiente**, lo que significa que puedes **realizar cambios, probar nuevas ideas o experimentar** con el código sin afectar el proyecto original.



GitHub: Forks



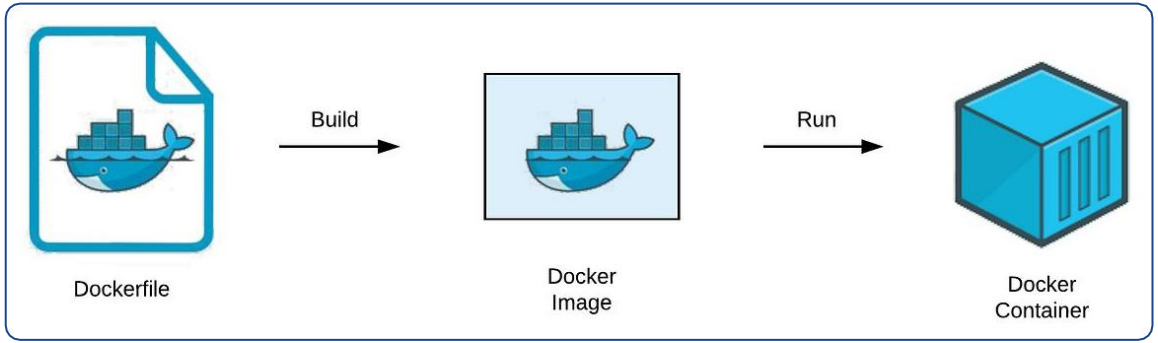



```
PS C:\Users\Daniel\Documents\Data_MC\MLE\Sesion_02\Mi_primer_repo> git clone https://github.com/Daniel-2025git/openai-python-fork.git
Cloning into 'openai-python-fork'...
remote: Enumerating objects: 8390, done.
remote: Counting objects: 100% (1498/1498), done.
remote: Compressing objects: 100% (311/311), done.
remote: Total 8390 (delta 1353), reused 1187 (delta 1187), pack-reused 6892 (from 3)
Receiving objects: 100% (8390/8390), 3.88 MiB | 4.20 MiB/s, done.
Resolving deltas: 100% (5672/5672), done.
```

Docker: Conceptos

¿Qué es Docker?

Docker es una plataforma para **desarrollar, enviar y ejecutar aplicaciones** utilizando **contenedores**.



Receta

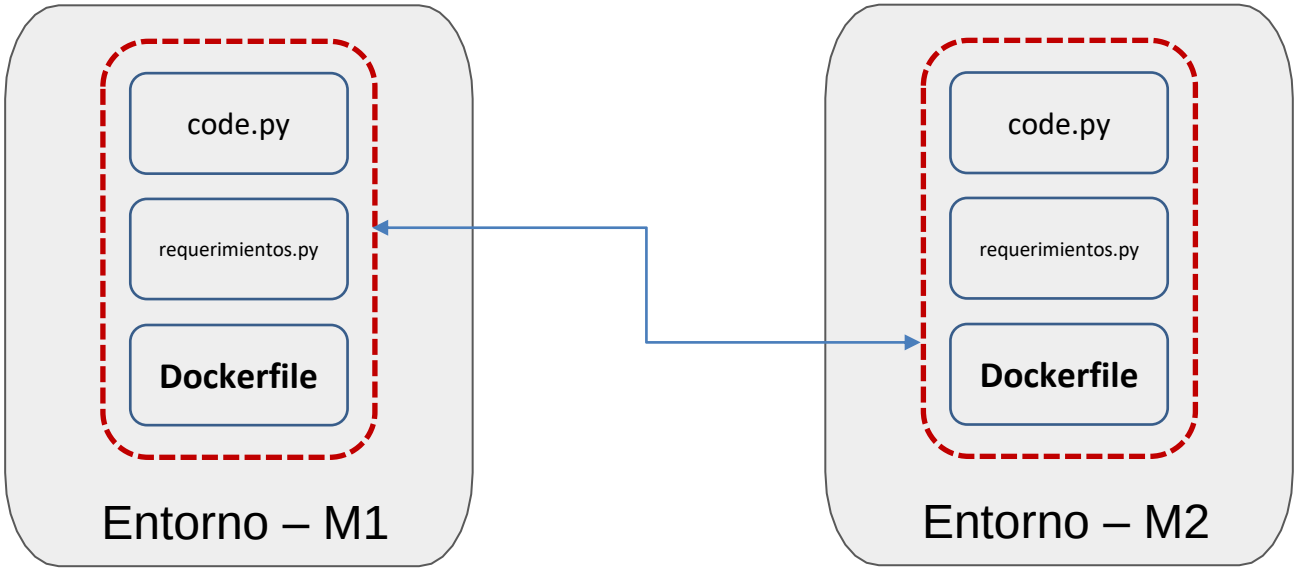
Masa + Ingrediente

Pizza horneada

Docker: Conceptos

¿Qué es Docker?

Docker es una plataforma para **desarrollar, enviar y ejecutar aplicaciones** utilizando **contenedores**.



Docker: Conceptos

Beneficios

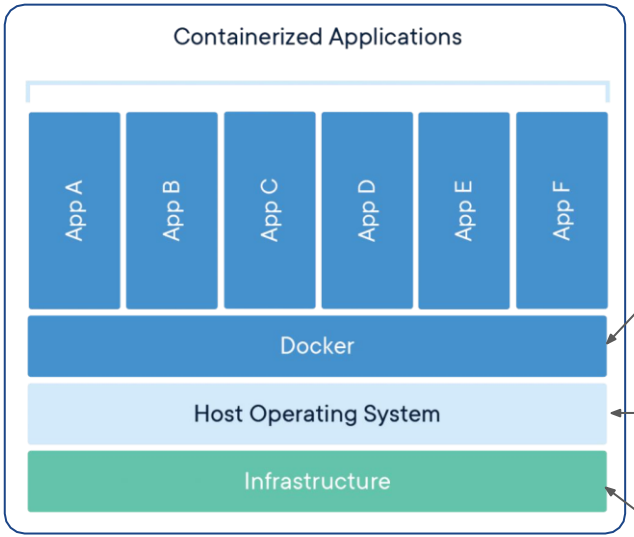
Consistencia y portabilidad: La aplicación se ejecute de la misma manera independientemente de dónde se implemente.

Aislamiento: Los cambios en un contenedor no afectan a otros contenedores.

Reproducibilidad: Los contenedores Docker se pueden configurar para que tengan versiones específicas de bibliotecas y dependencias.

Implementación rápida: Los contenedores se pueden crear, iniciar, detener y destruir rápidamente.

Arquitectura



Es una plataforma que permite empaquetar una aplicación con todas sus dependencias en una unidad estandarizada llamada contenedor

Es el sistema operativo instalado en la infraestructura. (Linux/Windows/macOS)

Este es el hardware físico o los recursos basados en la nube sobre los que se ejecuta todo.

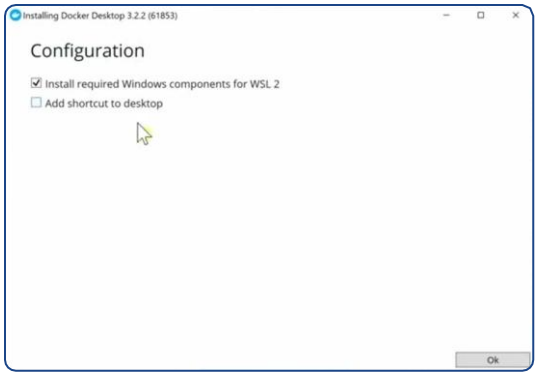
Docker: Docker Desktop

Docker Desktop es una **aplicación de escritorio** que facilita la **gestión y uso de Docker,** especialmente en entornos de desarrollo locales. Es una **interfaz gráfica de usuario (GUI)** que proporciona a los desarrolladores una manera fácil de construir, compartir y correr contenedores Docker en sistemas operativos como Windows y macOS. Docker Desktop **integra las funcionalidades esenciales de Docker, como Docker Engine, Docker CLI** (interfaz de línea de comandos), **Docker Compose**, y otras herramientas útiles en una sola instalación fácil de usar.

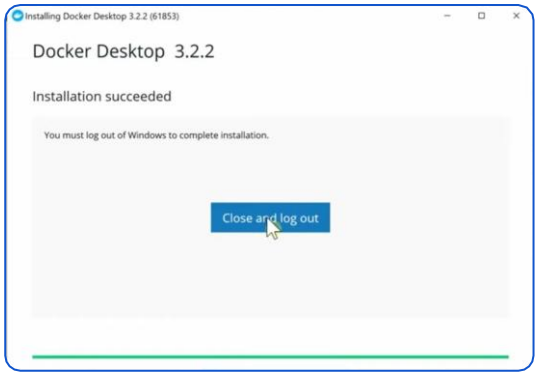
Iniciar con Docker - Desktop

1. Descargar instalador

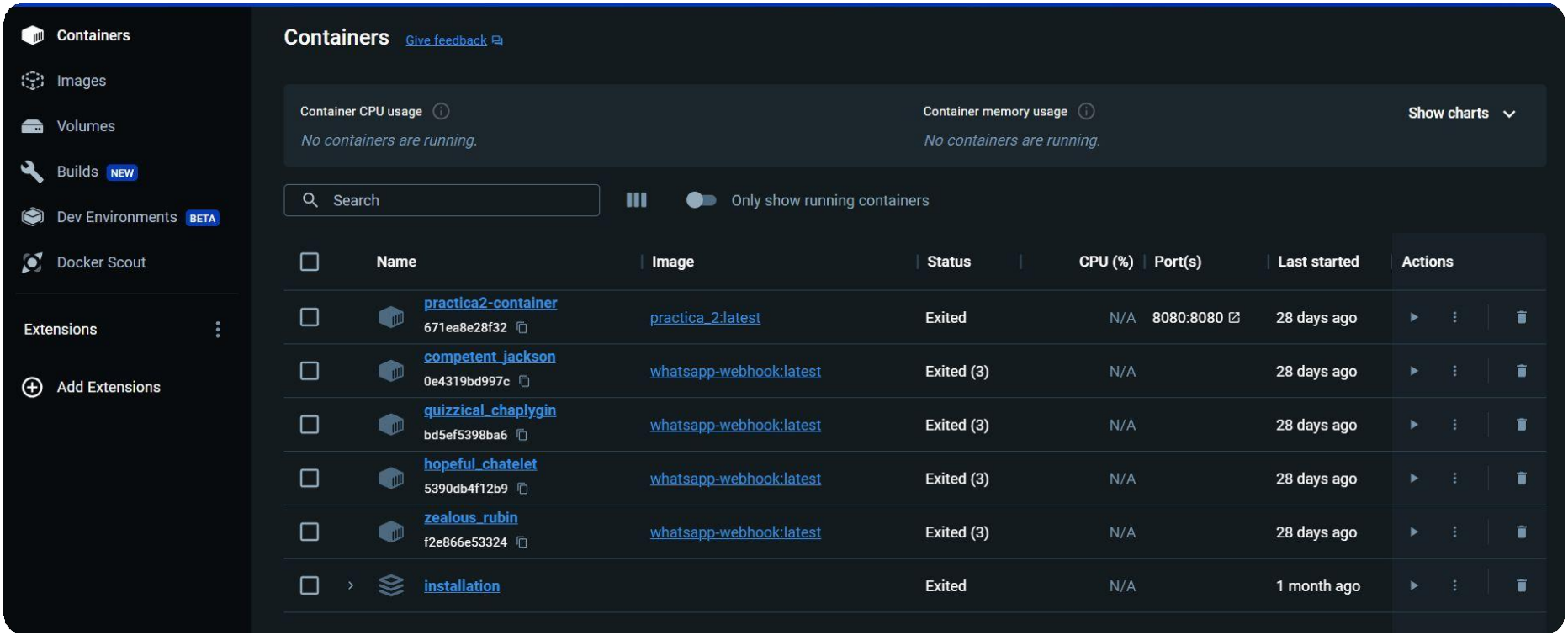
2. Agregar configuraciones



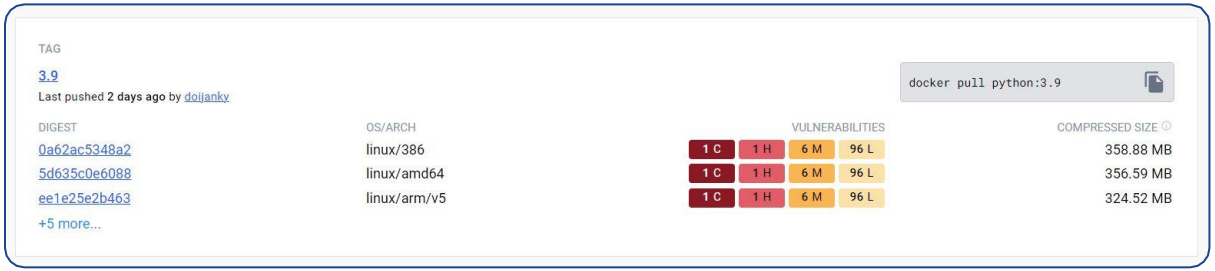
3. Reiniciar PC



Docker: Docker Desktop



Docker: Repositorio [Público]

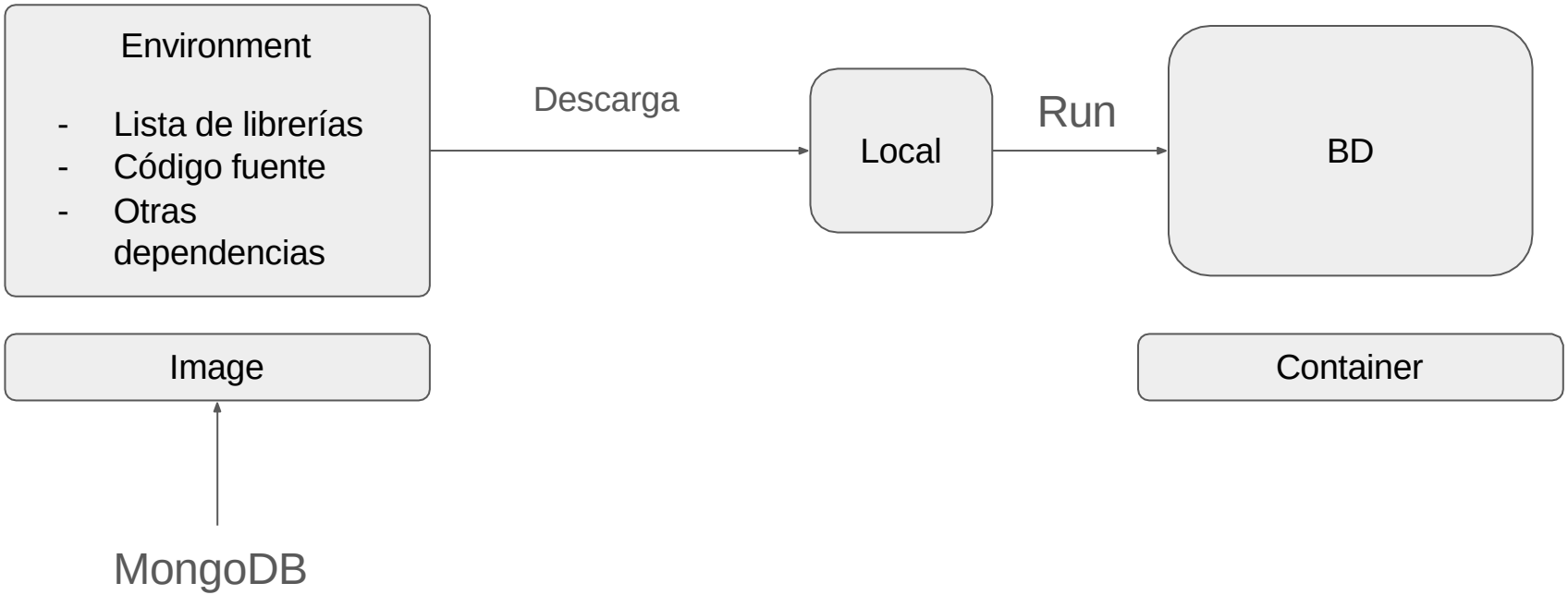


docker hub

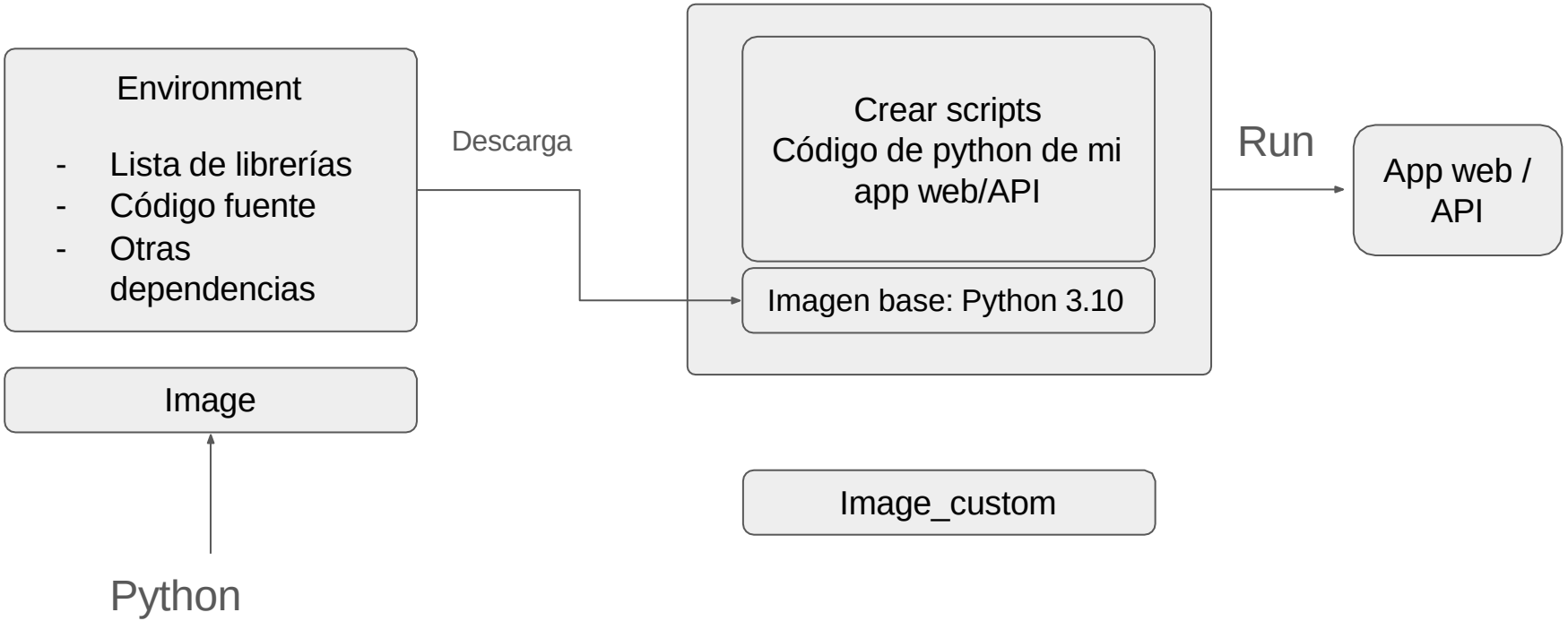
Docker Hub es un servicio de **repositorio basado en la nube** proporcionado por Docker para **buscar y compartir imágenes** de contenedores con su equipo.

Alberga repositorios de **imágenes públicos y privados**, que puede utilizar como base para sus propias aplicaciones.

Docker: Repositorio [Público]



Docker: Repositorio [Público]



Docker: Repositorio [Privado]

Amazon Elastic Container Registry (ECR): Es un servicio de registro de contenedores que permite a los usuarios almacenar, administrar e implementar imágenes de contenedores Docker en un registro privado. Se integra estrechamente con Amazon Web Services (AWS), ofreciendo seguridad de alto nivel y escalabilidad.

Google Artifact Registry (GCR): Es una solución de Google Cloud Platform para almacenar y acceder a imágenes de contenedores Docker. Ofrece integración con Google Cloud Identity & Access Management (IAM) para control de acceso y seguridad.

Azure Container Registry (ACR): Es el registro de contenedores de Microsoft Azure. Permite almacenar y gestionar imágenes de contenedores para todos los tipos de contenedores de Azure, con seguridad de nivel empresarial y compatibilidad con Azure Active Directory.

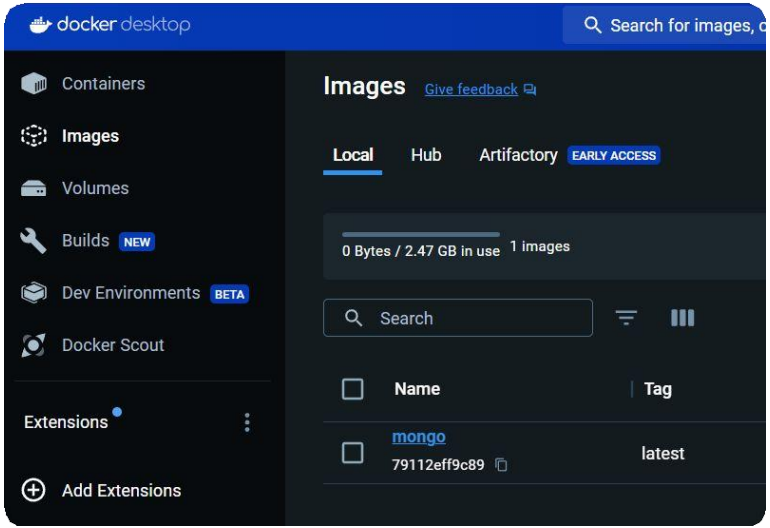
GitLab Container Registry: Integrado directamente dentro de GitLab, este registro de contenedores privado permite a los usuarios construir, almacenar y compartir imágenes de contenedores junto con su código fuente y pipelines de CI/CD.

Docker: Comandos

docker pull: El comando docker pull se utiliza en la interfaz de línea de comandos de Docker para **descargar imágenes de contenedores** desde un registro de contenedores. Un "registro de contenedores" es un repositorio (o una colección de repositorios) donde se almacenan las imágenes de contenedores. Docker Hub es el registro público de contenedores más conocido, pero también puedes descargar imágenes desde otros registros privados o públicos.

```
docker pull mongo:latest
```

```
PS D:\DMC> docker pull mongo:latest
latest: Pulling from library/mongo
bccd10f490ab: Pull complete
93f07c832c0f: Pull complete
dab2da22865c: Pull complete
1f7ceac81d5b: Pull complete
3e507bb35e33: Pull complete
bf53d03cb40e: Pull complete
cc32baa6c25a: Pull complete
15cbd2791748: Pull complete
Digest: sha256:5a6889e9f5e0c71ad8d1cf94b6d83d38162ba198e6083371e5141fc38137a01a
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```



Docker: Comandos

docker images: El comando docker images en Docker se utiliza para **listar todas las imágenes** de Docker que están **localmente almacenadas en tu máquina**. Las imágenes de Docker son plantillas usadas para crear contenedores de Docker y contienen el sistema operativo, software, archivos de aplicación y configuraciones necesarias para ejecutar una aplicación o un servicio.

```
docker images
```

Docker: Comandos

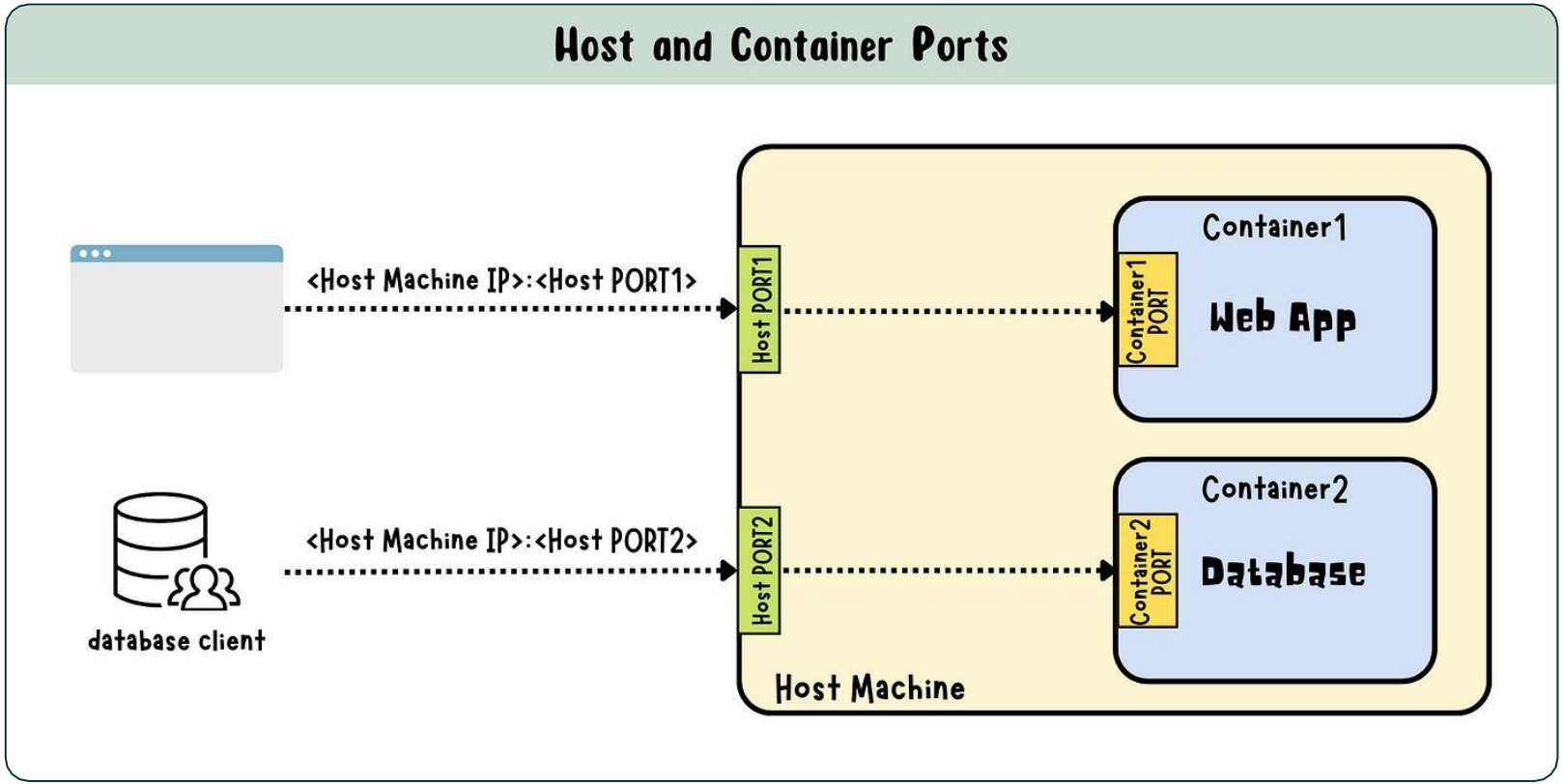
docker image rm: El comando docker image rm es utilizado en Docker para **eliminar una o más imágenes del sistema local.** Se utiliza para liberar espacio de almacenamiento o para limpiar imágenes que ya no necesitas.

Cuando ejecutas este comando, Docker eliminará la imagen especificada de tu máquina local. Si una imagen está siendo utilizada por algún contenedor, ya sea en ejecución o detenido, no podrás eliminarla a menos que uses opciones adicionales o elimines esos contenedores.

```
docker image rm mongo:latest
```

```
PS D:\DMC> docker image rm mongo:latest
Untagged: mongo:latest
Untagged: mongo@sha256:5a6889e9f5e0c71ad8d1cf94b6d83d38162ba198e6083371e5141fc38137a01a
Deleted: sha256:79112eff9c89624d06e0506e6ae01b76c68ec9fdbe2eda0c6f7ae56dae725f41
Deleted: sha256:6b390ae83ba538b2a48b2d76e6142a61081ea327415b33c9d2596ed9d48afeb9
Deleted: sha256:dd74d6aecd2a6ebb0cd0d595dcf258d7bb80b8162ad2010a6788af09aa98cc78
Deleted: sha256:6f8658f4f3a4d4d7565dfee4b7141b04662137ad17d90211bedf59abc071a72a
Deleted: sha256:926d3d7b4fbc75d42cb1a79cdfa7c09fdf2dc03d3466a6f7d4c3752e575a9782
Deleted: sha256:cb5732edf9aa5a75a217c70c5cde17d5870d26094cfda457cc216467ab0a1e8a
Deleted: sha256:63e0ede228e6ee673b9aa7f4efabddf054d5e428cdfb555097a4f9b634862463
Deleted: sha256:579b72c8f5f843cb2a4d31ed680475da3b2266b0e388577497325d01d79d5dd9
Deleted: sha256:5498e8c22f6996f25ef193ee58617d5b37e2a96decf22e72de13c3b34e147591
```

Docker: Port Mapping

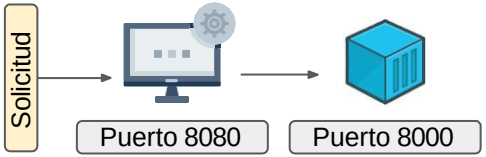


Docker: Comandos

docker create: El comando docker create se utiliza para **crear un nuevo contenedor a partir de una imagen de Docker,** pero sin iniciarlo inmediatamente. Este comando es útil cuando necesitas configurar un contenedor de manera específica antes de arrancarlo o si deseas preparar varios contenedores para iniciarlos más tarde.

```
docker create -p 8080:8000 --name contenedor_mongo mongo:latest
```

```
PS D:\DMC> docker create -p 8080:8000 --name mi_contenedor python:latest
0d935e0abb196b9f1205c549489d4bb8ddacad01f8c916efbb774c3bac916048
```



docker ps -a: El comando docker ps -a es utilizado en Docker para **listar todos los contenedores disponibles en tu sistema**, incluyendo tanto los contenedores en ejecución como los detenidos.
La opción -a o --all es lo que hace que el comando muestre todos los contenedores. Sin esta opción, docker ps por defecto muestra solo los contenedores activos (en ejecución).

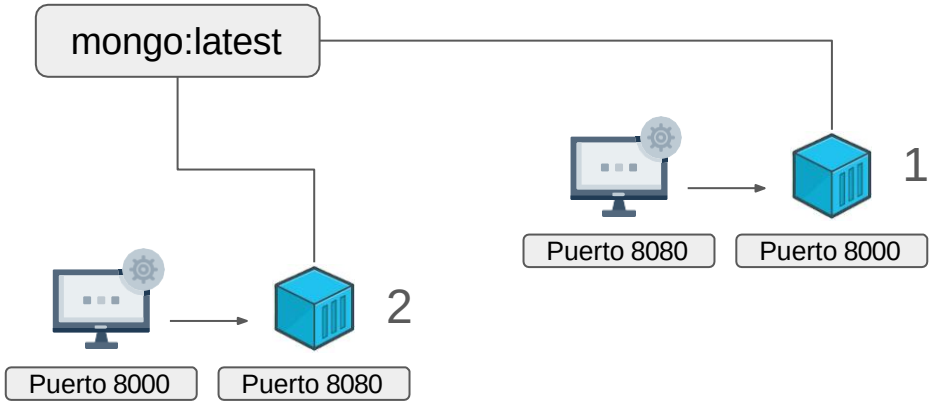
```
docker ps -a
```

```
PS D:\DMC> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS    PORTS     NAMES
a9a035df76fb   mongo:latest   "docker-entrypoint.s…"   9 seconds ago   Created             contenedor_mongo
```

Docker: Comandos

docker create: El comando docker create se utiliza para **crear un nuevo contenedor a partir de una imagen de Docker,** pero sin iniciarlo inmediatamente. Este comando es útil cuando necesitas configurar un contenedor de manera específica antes de arrancarlo o si deseas preparar varios contenedores para iniciarlos más tarde.

```
docker create -p 8080:8000 --name contenedor_mongo mongo:latest
```

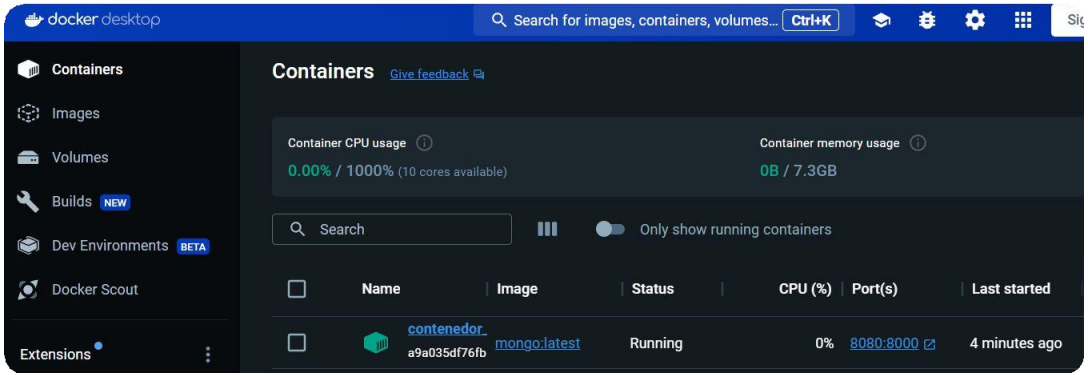


Docker: Comandos

docker start: El comando docker start se utiliza en Docker para **iniciar uno o más contenedores que ya han sido creados pero que están detenidos.** Esto es útil cuando tienes un contenedor que fue previamente detenido (ya sea manualmente o porque el proceso dentro del contenedor se completó) y necesitas volver a arrancarlo sin cambiar ninguna de su configuración o datos existentes.

```
docker start contenedor_mongo
```

```
PS D:\DMC> docker start contenedor_mongo
contenedor_mongo
PS D:\DMC> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                NAMES
a9a035df76fb   mongo:latest   "docker-entrypoint.s…"   5 minutes ago   Up About a minute   27017/tcp, 0.0.0.0:8080->8000/tcp   contenedor_mongo
```

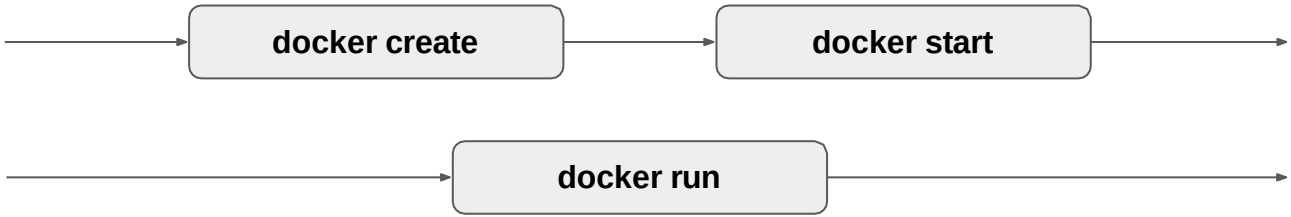


Docker: Comandos

docker run: El comando docker run es uno de los comandos más fundamentales en Docker y se utiliza para **crear y arrancar un contenedor a partir de una imagen específica.** Este comando combina varias acciones en una sola: descargar la imagen (si no está localmente disponible), crear un contenedor a partir de esa imagen, y luego ejecutar el contenedor.

```
docker run -d --name contenedor_mongo_2 -p 7070:7000 mongo
```

```
PS D:\DMC> docker run -d --name contenedor_mongo_2 -p 7070:7000 mongo
c95bc7c312c163709a642cf1dfc37ca24196c9be7e49825d3496cdbb784d44ed
PS D:\DMC> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                   NAMES
c95bc7c312c1   mongo          "docker-entrypoint.s…"   3 seconds ago    Up 2 seconds    27017/tcp, 0.0.0.0:7070->7000/tcp       contenedor_mongo_2
a9a035df76fb   mongo:latest   "docker-entrypoint.s…"   33 minutes ago   Up 29 minutes   27017/tcp, 0.0.0.0:8080->8000/tcp       contenedor_mongo
```



Docker: Comandos

docker stop: El comando docker stop se utiliza en Docker para **detener uno o más contenedores en ejecución.** Este comando envía una señal SIGTERM al proceso principal dentro del contenedor, lo que le pide cortésmente que se detenga. Si el proceso dentro del contenedor no se detiene después de un periodo de gracia (por defecto son 10 segundos), entonces Docker envía una señal SIGKILL para forzar la detención del contenedor.

```
docker stop contenedor_mongo_2
```

docker rm: El comando docker rm se utiliza en Docker para **eliminar uno o más contenedores de tu sistema.** Este comando solo eliminará contenedores que están detenidos; si intentas eliminar un contenedor que está en ejecución, recibirás un error a menos que utilices la opción -f o --force para forzar la eliminación.

```
docker rm contenedor_mongo_2
```

Docker : Imagen personalizada de Docker

Documentos necesarios

- app.py
- Dockerfile
- requirements.txt

app.py
Este archivo es el punto de entrada de la aplicación Python. Contiene el **código fuente de la aplicación** que desea ejecutar dentro del contenedor Docker. Por ejemplo, si se está desarrollando una aplicación web con Flask, app.py podría verse así:

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, Docker!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

En este ejemplo, app.py inicia un servidor web Flask que **escucha en todas las interfaces disponibles en el puerto 8080.** Cuando se accede a la ruta raíz ('/'), devuelve el mensaje "Hello, Docker!".

Docker : Imagen personalizada de Docker

Documentos necesarios

- app.py
- Dockerfile
- requirements.txt

Dockerfile
Este archivo contiene las **instrucciones para construir la imagen** Docker de su aplicación. Define el entorno y los **comandos necesarios** para ejecutar su aplicación. Aquí hay un ejemplo de Dockerfile para una aplicación Flask:

```
# Usa una imagen base oficial de Python como punto de partida
FROM python:3.8-slim

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copia los archivos 'requirements.txt' y 'app.py' al directorio de
trabajo
COPY requirements.txt app.py ./

# Instala las dependencias de Python especificadas en 'requirements.txt'
RUN pip install --no-cache-dir -r requirements.txt

# Informa a Docker que el contenedor escuchará en el puerto 8080
EXPOSE 8080

# Define el comando que se ejecutará cuando se inicie el contenedor
CMD ["python", "app.py"]
```

Este Dockerfile comienza con una **imagen base de Python**, establece un **directorio de trabajo**, **copia los archivos** necesarios del proyecto, **instala las dependencias** y especifica el **comando para ejecutar** la aplicación.

Docker : Imagen personalizada de Docker

Documentos necesarios

- app.py
- Dockerfile
- requirements.txt

requirements.txt
Este archivo lista todas las **dependencias de Python** necesarias para su aplicación. Docker las instalará en el contenedor basándose en este archivo. Si su aplicación utiliza Flask, el contenido de requirements.txt podría ser tan simple como:

```
Flask==2.3.2
```

Cada línea en requirements.txt especifica un **paquete** y, opcionalmente, **una versión exacta**. Al construir la imagen Docker, el comando RUN pip install --no-cache-dir -r requirements.txt instalará estas dependencias en el contenedor.

Docker : Imagen personalizada de Docker

Documentos necesarios

- app.py
- Dockerfile
- requirements.txt

Crear imagen personalizada de docker

```
docker build -t my-first-app:latest .
```

Iniciar contenedor de docker

```
docker run -p 8000:8080 my-first-app
```

¡GRACIAS!