# Mi Primera Prueba Real de CI/CD con Docker 🚀

## 🎯 ¿En qué consiste este ejercicio?

El objetivo es crear un sistema automatizado donde tú solo te encargues de programar. Al subir tu código a **GitHub**, un "robot" (GitHub Actions) se activará automáticamente, empaquetará tu aplicación en un contenedor de **Docker** y lo subirá a la nube (**Docker Hub**) y tu ordenador convertido en un servidor, realizará el despliegue automático. Esto es exactamente lo que ocurre en las empresas: el código viaja de GitHub a servidores en la nube de forma totalmente transparente.

### 🔦 Conceptos Clave

- **CI (Integración Continua):** Es la automatización de la construcción y prueba de tu código cada vez que haces un cambio. Asegura que el nuevo código "encaje" bien.
- **CD (Despliegue Continuo):** Es llevar ese código automáticamente a un entorno de producción (un servidor) para que los usuarios puedan usarlo.

## ✅ Checklist de Requisitos

Antes de tocar el código, asegúrate de tener:

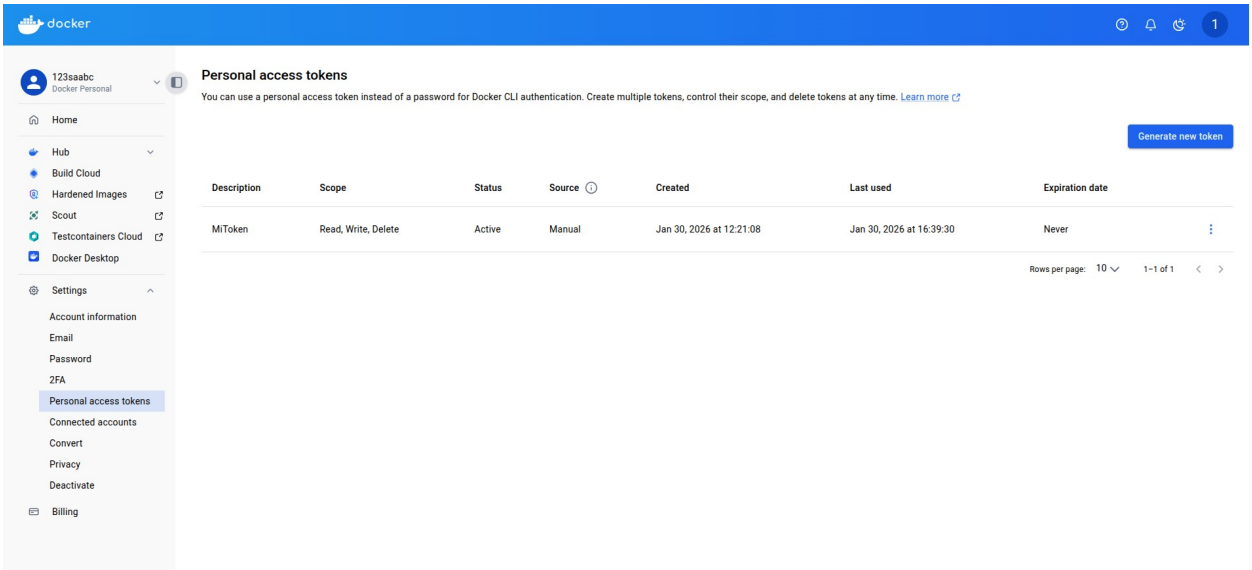
- [ ] Una cuenta en GitHub.
- [ ] Una cuenta en Docker Hub.
- [ ] Configurar el ordenador local en runner.
- [ ] Git instalado en tu ordenador.
- [ ] Un editor de código (como VS Code).

## Paso 1: Preparación en Docker Hub (Las llaves)

Para que GitHub pueda subir cosas a tu cuenta de Docker, necesitamos una "llave especial" (Token).

1. **Entra en Docker Hub** y haz clic en tu foto (arriba a la derecha) -> **Account Settings**.
2. Ve a **Security** -> **Personal Access Tokens** -> **New Access Token**.
3. **Configuración:**
   - Nombre: GitHub-Actions-CI
   - Permisos: **Read, Write, Delete**.
4. **¡Copia el Token!** Dale al botón de copiar y guárdalo en un bloc de notas temporalmente. No podrás volver a verlo.
5. **Anota tu Usuario:** Tu usuario de Docker Hub no es tu email, es el ID que aparece bajo tu

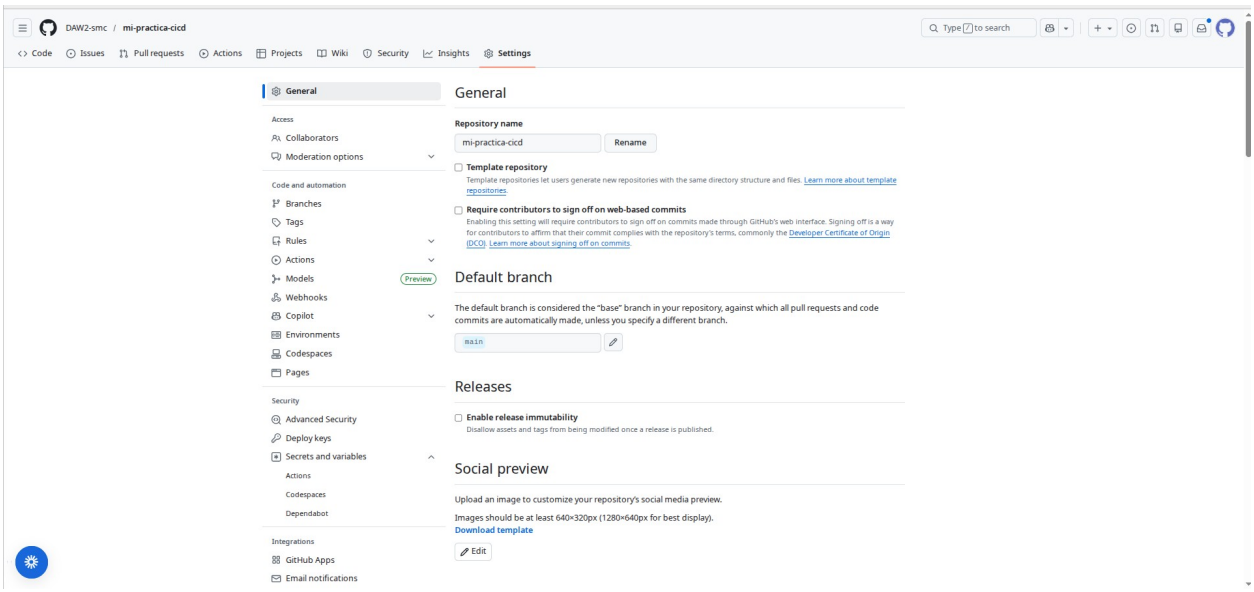
nombre en el perfil.



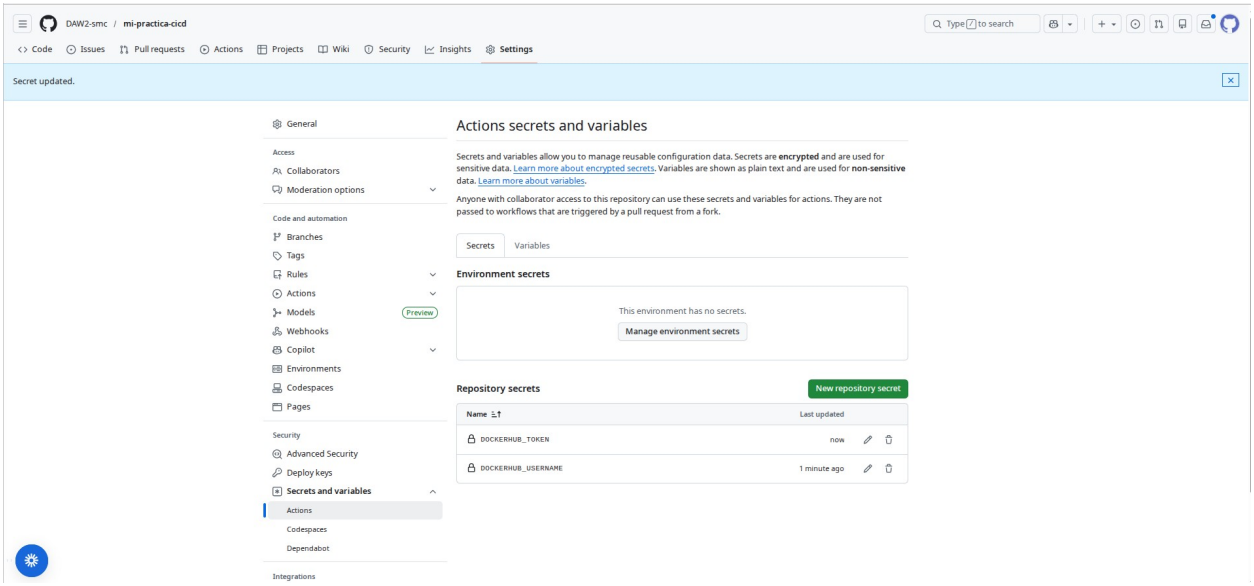

# Paso 2: Configurar los Secretos en GitHub (Donde guardar las llaves)

**IMPORTANTE:** No confundas el Settings de tu perfil con el del proyecto.

1. Crea un repositorio nuevo y público que se llame mi-practica-cicd
2. Entra en la página de tu repositorio en GitHub (ej: mi-practica-cicd).
3. En la barra superior de pestañas del repositorio, haz clic en **Settings** ⚙ (es la última a la derecha, junto a *Insights*).
4. En el menú lateral izquierdo, busca la sección **Security** -> **Secrets and variables** -> **Actions**.




5. Pulsa el botón verde **New repository secret** para crear dos secretos:
    - **Primer Secreto:**
        - Name: DOCKERHUB_USERNAME
        - Secret: (Tu usuario de Docker Hub, el ID corto).
    - **Segundo Secreto:**
        - Name: DOCKERHUB_TOKEN
        - Secret: (El token que copiaste en el Paso 1 que empieza por dckr_pat_...).




** Estamos metiendo en GitHub las **claves (usuario y token) de DockerHub**, porque el robot lo que va a hacer es que cuando haya cambios en el código en el repositorio común de desarrolladores GITHUB, va a crear una imagen y subirlo al HUB de DOCKER de forma automática.

# Paso 3: Configurar el Runner en ordenador Local

# ⚠ IMPORTANTE: Estructura de Carpetas

**NUNCA** instales el Runner dentro de la carpeta de tu repositorio de Git. Esto causará errores de subida por el peso de los archivos.

- **Carpeta de Código (Git):** `.../mi-proyecto/` (Solo archivos de texto: html, yaml).
- **Carpeta del Runner:** `~/actions-runner/` (Fuera de la carpeta de Git).

Aún incluyendo el fichero de .ignore, la carpeta de runner debe estar fuera de la carpeta que se sincroniza con gitHub.

## 3.1. Registrar tu ordenador como Runner en GitHub

- En tu repo de GitHub, ve a **Settings** -> **Actions** -> **Runners**.
- Pulsa **New self-hosted runner**.
- Selecciona tu sistema operativo (**Windows**, **macOS** o **Linux**).
- GitHub te dará una lista de comandos. **¡No cierres esa pestaña!**

## 3.2. Configurar el Runner en tu PC

- Abre una terminal en tu ordenador.
- Crea una carpeta llamada runner-github (ej: mkdir runner-github && cd runner-github).
- Copia y pega los comandos de **Download** que te dio GitHub.
- Copia y pega el comando de **Configure** (el que empieza por ./config.sh o config.cmd).
  - **Importante:** Cuando te pregunte por **Labels**, escribe: mi-pc-local.
  - A lo demás, puedes darle a **Enter** para aceptar los valores por defecto.
- Finalmente, arranca el runner con: ./run.sh (o run.cmd en Windows).
  - Deberías ver: ✓ Connected to GitHub. Listening for Jobs.


# Paso 4: El Proyecto en Local (Tu PC)

## 4.1.Crear carpeta en local

Bash
mkdir mi-proyecto-cicd-ej2 && cd mi-proyecto-cicd-ej2

## 4.2 Crear archivo index.html

- index.html: Tu página web sencilla.

index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <title>Mi Primera Web con CI/CD</title>
</head>
<body>
    <h1>¡Hola! Este sitio se actualiza solo.</h1>
    <p>Aprendiendo Docker y GitHub Actions.</p>
</body>
</html>
```

## 4.3 Crear archivo dockerfile

- Dockerfile: Las instrucciones para que Docker sepa cómo montar tu web.

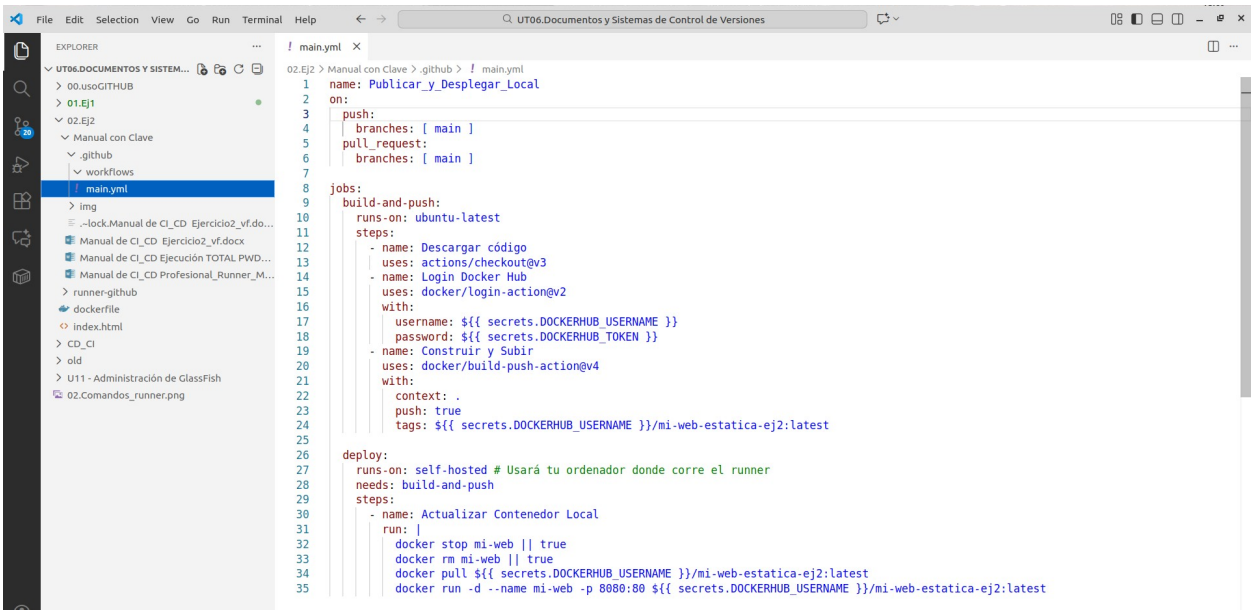
dockerfile > ...

```
# Usamos un servidor web muy ligero llamado Nginx
FROM nginx:alpine

# Copiamos nuestro archivo html dentro de la carpeta del servidor
COPY index.html /usr/share/nginx/html/index.html
```

## 4.4 Crear el workflow en un archivo main.yml

- main.yml: Este archivo es el "cerebro" que le dice a GitHub: "Usa mis secretos de Docker Hub para hacer login y subir mi imagen".
- Tengo que crear una carpeta oculta .github y dentro una carpeta workflow donde creo el fichero.




```
name: Publicar_y_Desplegar_Local

on:
push:
branches: [ main ]
pull_request:
branches: [ main ]
```

```
jobs:
build-and-push:
runs-on: ubuntu-latest
steps:
- name: Descargar código
uses: actions/checkout@v3
- name: Login Docker Hub
uses: docker/login-action@v2
with:
username: ${{ secrets.DOCKERHUB_USERNAME }}
password: ${{ secrets.DOCKERHUB_TOKEN }}
- name: Construir y Subir
uses: docker/build-push-action@v4
with:
context: .
push: true
tags: ${{ secrets.DOCKERHUB_USERNAME }}/mi-web-estatica-ej2:latest

deploy:
runs-on: self-hosted # Usará tu ordenador donde corre el runner
needs: build-and-push
steps:
- name: Actualizar Contenedor Local
run: |
docker stop mi-web || true
docker rm mi-web || true
docker pull ${{ secrets.DOCKERHUB_USERNAME }}/mi-web-estatica-ej2:latest
docker run -d --name mi-web -p 8080:80 ${{ secrets.DOCKERHUB_USERNAME }}/mi-
web-estatica-ej2:latest
```

## Paso 5: Sincronizar GitLocal y GitHub

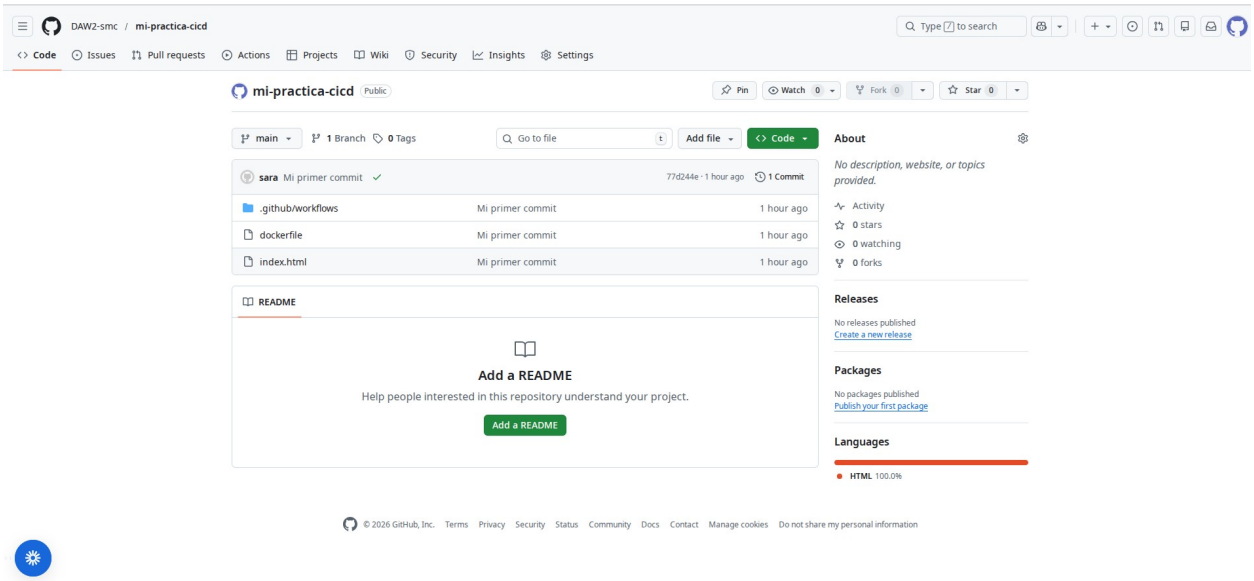
Los ficheros que hemos creado en local, ahora queremos compartirlos con nuestro equipo en GitHub, que es donde colaboramos juntos. Para eso, donde tengo los fichero de index.html [...] abro un terminal y realizo los pasos para enviar los datos al GitHub.

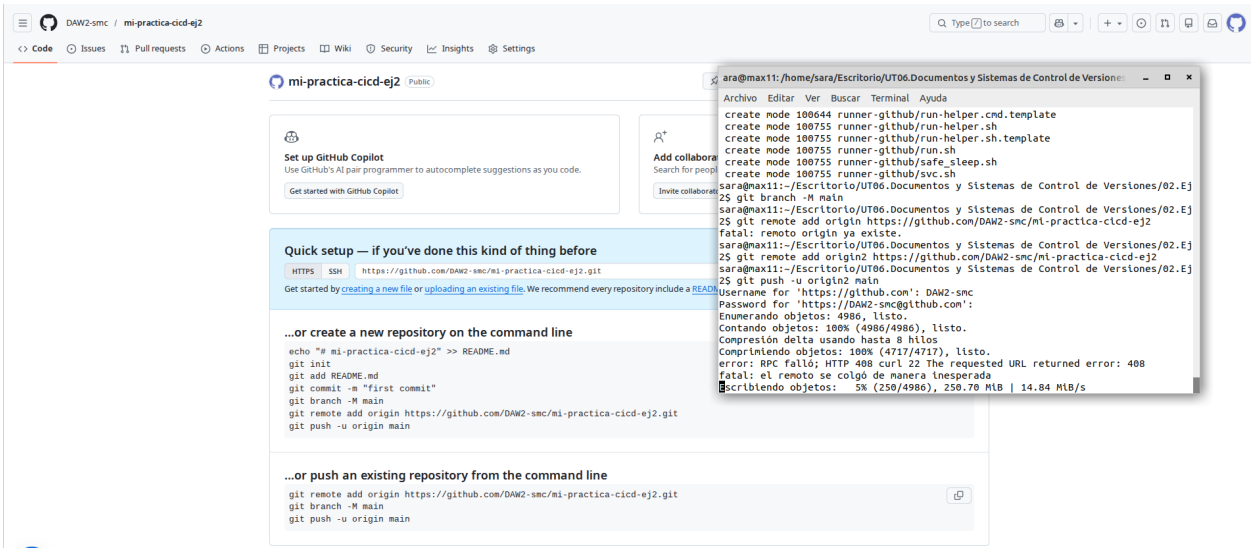
Bash

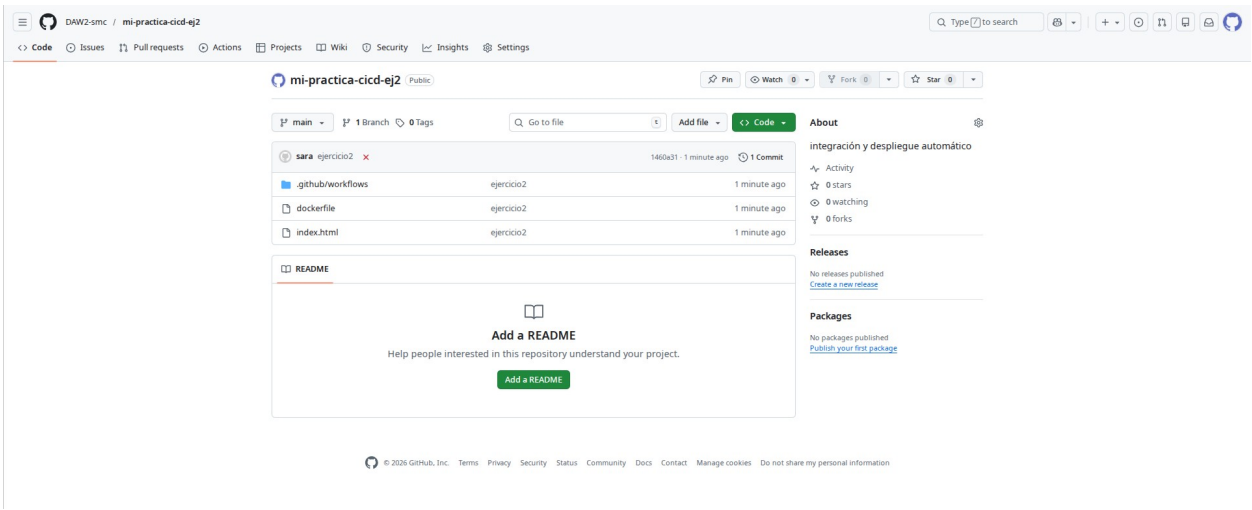
```
git init
git add .
git commit -m "Estructura inicial"
git branch -M main
git remote add origin URLRepositorio
```

git push -u origin main

Cuando lo hagamos vemos en nuestro GITHUB la información actualizada:

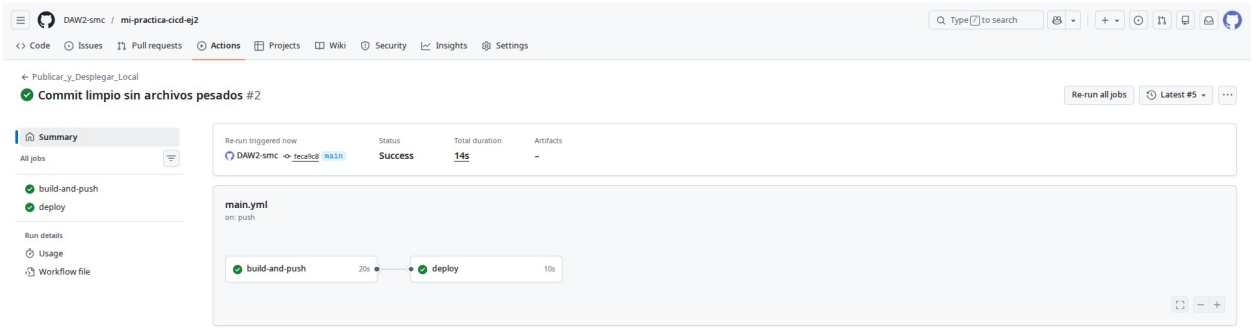





# Paso 6: Menú ACTIONS en GitHub

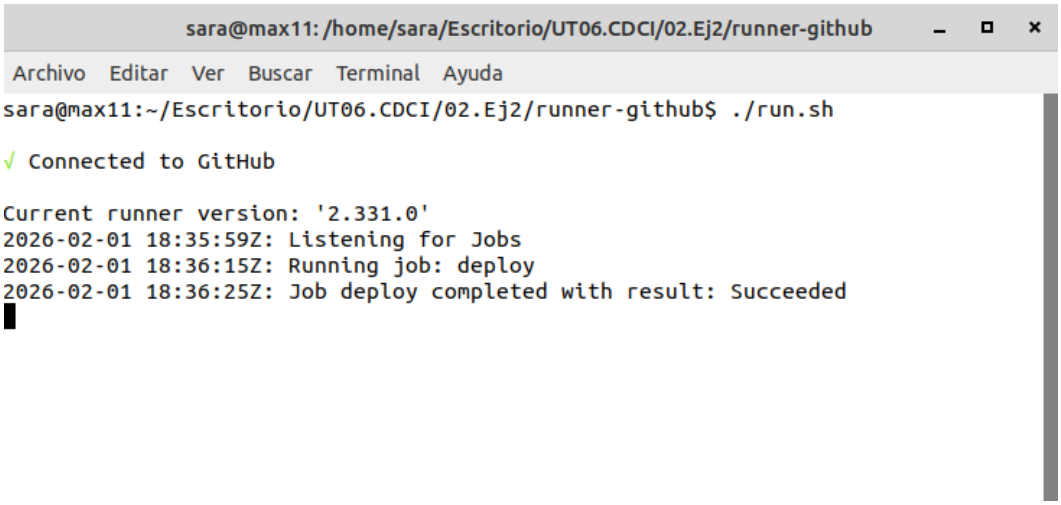
Cuando hemos hecho el push, además de subirse los ficheros, se ha ejecutado el código del main.yml creando un job, que ha podido terminar ok y se muestra en verde o terminar con errores y nos dice qué ha fallado.




Cuando se han ejecutado los dos pasos, mágicamente se publica nuestra web:

http://localhost:8080/

```
sara@max11:~$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED          STATUS          POR
TS                                    NAMES
8d7611914e17   123saabc/mi-web-estatica-ej2:latest    "/docker-entrypoint.…"   11 minutes ago   Up 11 minutes   0.0
.0.0:8080->80/tcp, [::]:8080->80/tcp   mi-web-ej2
```



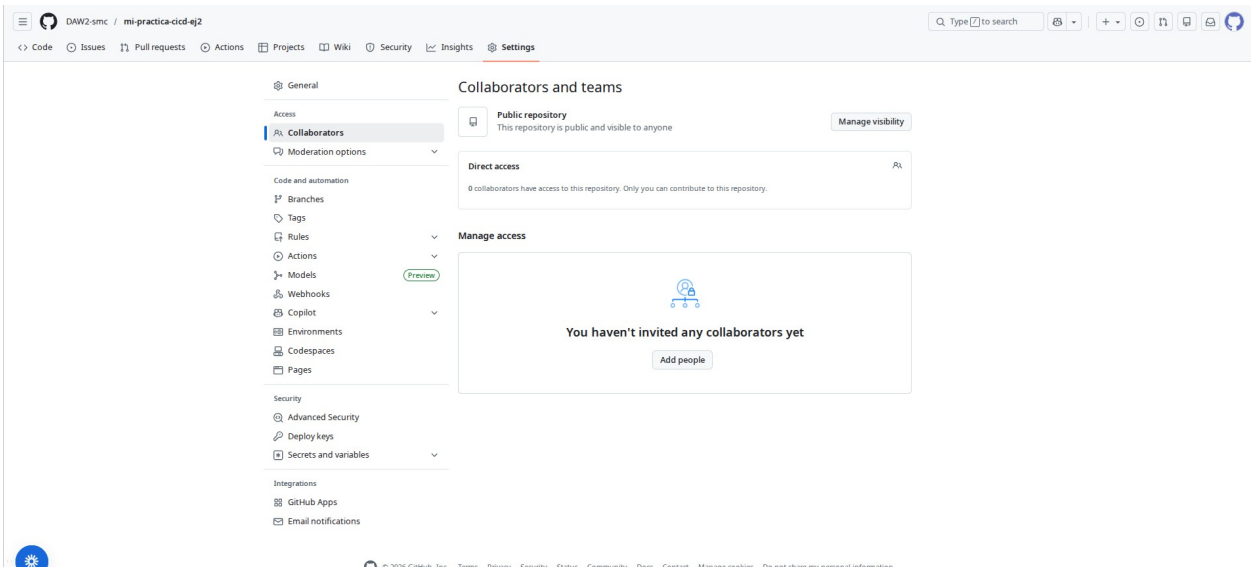

# Paso 7: Pull request de un colaborador.

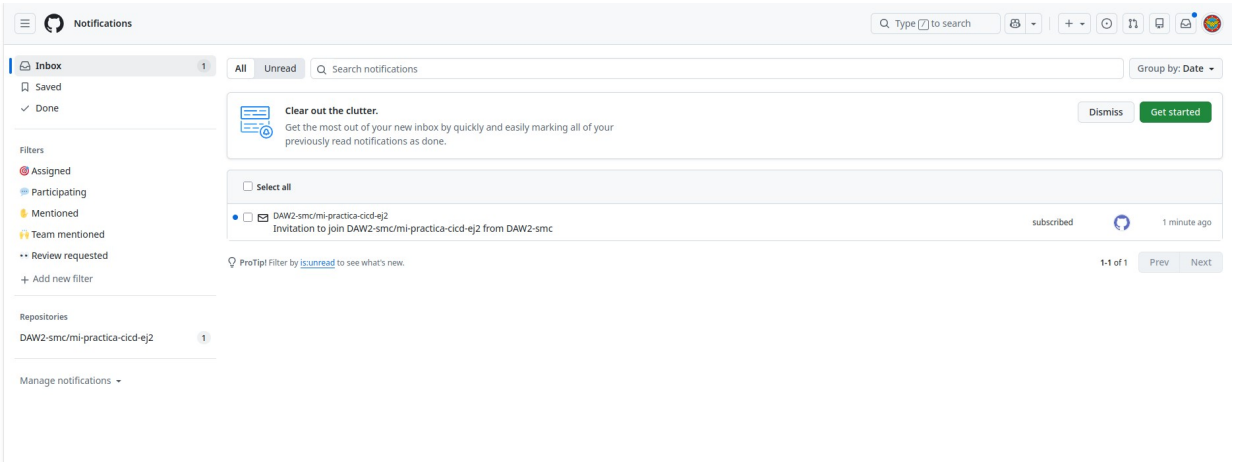
Para ver el CD en acción:

2. Asegúrate de que el Runner está en estado Listening en tu terminal.
3. Simula un cambio (o pídele a un colaborador que lo haga):
   - Cambia el fondo de la web en index.html.
   - Haz git push a la rama main.
4. Mira tu terminal: Verás que el Runner recibe un "Job", ejecuta los comandos de Docker, descarga la imagen y la arranca.
5. Abre en tu navegador localhost:8080. **¡Tu web se ha actualizado sola!**

## 7.1.Damos acceso a un colaborador

Entramos en Settings del repositorio y en Colaboradores añadimos a un colaborador. Este, entrará en su cuenta de GitHub y aceptará la invitación.



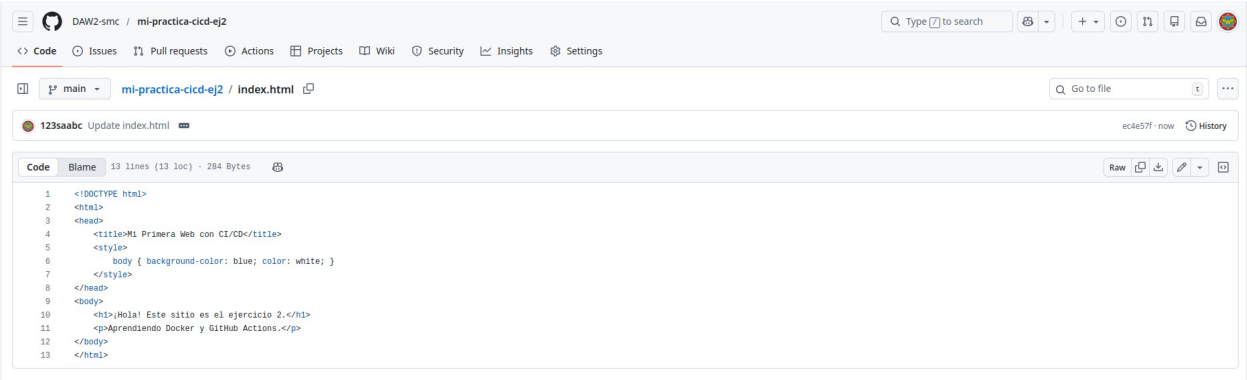




## 7.2.Colaborador hace cambios y actualiza GH.

### Opción A: Directamente desde la web de GitHub (La más rápida)

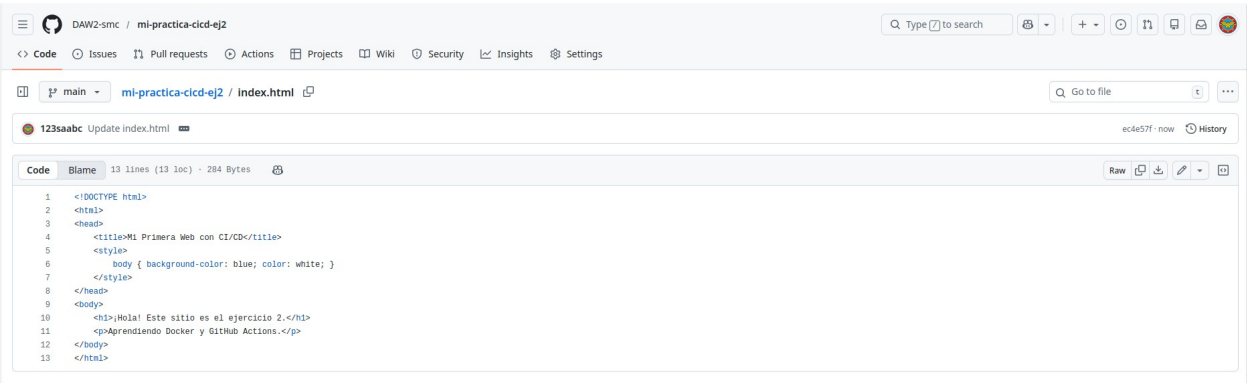
1. **Entra con la cuenta del colaborador** en el repositorio.
2. Busca el archivo `index.html` y haz clic en el icono del **lápiz (Edit)**.
3. Modifica algo visible (por ejemplo, cambia el título <h1> o el color de fondo en el CSS).
4. Haz clic en el botón verde **Commit changes...**.
5. Aquí tienes dos opciones:
    - **Commit directly to the main branch:** Esto disparará el CI/CD inmediatamente.
    - **Create a new branch and start a pull request:** Esto es lo más profesional. Tú (como dueño) tendrás que ir a la pestaña **Pull Requests**, revisar el cambio y

darle a **Merge**. En cuanto le des a "Merge", se disparará el proceso hacia tu ordenador.




Hemos hecho un commit directamente desde el colaborador.

y desde el repositorio y el usuario maestro vemos ese ok al cambio del colaborador.




## Opción B: Desde la terminal del colaborador (La profesional)

Si tu colaborador quiere trabajar en su propio PC:

1. **Clonar:** `git clone https://github.com/DAW2-smc/tu-repo.git`
2. **Modificar:** Abrir el `index.html` y hacer los cambios.
3. **Subir:**

   Bash

   ```
   git add .
   git commit -m "Cambio realizado por colaborador 123saabc"
   git push origin main
   ```

---

## 💡 Lo que debes vigilar en TU ordenador (El Servidor)

Mientras el colaborador hace eso, estamos atentos a dos sitios:

1. **El terminal:** Donde tenemos escuchando los Jobs con `./run.sh`, aparecerá `job: deploy`.
2. **El navegador:** En cuanto el job termine, refrescamos `localhost:8080`. ¡Se verán los cambios que ha hecho el colaborador sin que tú hayas bajado nada de código!