

Desafío 9

Comenzamos el desafío ingresando a nuestro Github y haciendo fork al repositorio <https://github.com/edgaregonzalez/nodejs-helloworld-api> seguidamente creamos un directorio en nuestro entorno local para clonar el repositorio al que hicimos fork.

```
PS C:\Users\Michael> CD .\Desktop\  
PS C:\Users\Michael\Desktop> cd '..\Desafio 9\  
PS C:\Users\Michael\Desktop\Desafio 9> git clone https://github.com/Kidbuut/nodejs-helloworld-api  
Cloning into 'nodejs-helloworld-api'...  
remote: Enumerating objects: 37, done.  
remote: Counting objects: 100% (19/19), done.  
remote: Compressing objects: 100% (9/9), done.  
remote: Total 37 (delta 12), reused 10 (delta 10), pack-reused 18 (from 1)  
Receiving objects: 100% (37/37), 45.17 KiB | 660.00 KiB/s, done.  
Resolving deltas: 100% (18/18), done.
```

Creamos un archivo Dockerfile con el código que necesitaremos

```
dockerfile U ●  
dockerfile  
1  # Usar una imagen base oficial de Node.js  
2  FROM node:14  
3  
4  # Establecer el directorio de trabajo dentro del contenedor  
5  WORKDIR /usr/src/app  
6  
7  # Copiar package.json y package-lock.json  
8  COPY package*.json ./  
9  
10 # Instalar dependencias  
11 RUN npm install  
12  
13 # Copiar el resto del código de la aplicación  
14 COPY . .  
15  
16 # Exponer el puerto en el que la app estará corriendo  
17 EXPOSE 8080  
18  
19 # Comando para iniciar la aplicación  
20 CMD ["npm", "start"]
```

Después de haber creado el Dockerfile, el próximo paso es generar una imagen Docker a partir de él y posteriormente, ejecutar un contenedor utilizando esa imagen.

Para crear nuestra imagen en Docker debemos posicionarnos en la carpeta o directorio donde está alojado nuestro código una vez allí correr el siguiente código en la terminal de

Powershell **Docker build -t desafioapp** . “desafioapp” es el nombre que yo elegi para mi imagen en el caso del que esté leyendo la documentacion seria el que elija.

```
PS C:\Users\Michael\Desktop\Desafio 9\nodejs-helloworld-api> docker build -t desafioapp .
[+] Building 52.4s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> == transferring dockerfile: 486B
=> [internal] load metadata for docker.io/library/node:14
[auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> == transferring context: 2B
=> [1/5] FROM docker.io/library/node:14@sha256:a150d3b94de3fa813fa6c8c590b8f8a860e015ad4e59bbce5744d2f6fd8461aa
=> == resolve docker.io/library/node:14@sha256:a150d3b94de3fa813fa6c8c590b8f8a860e015ad4e59bbce5744d2f6fd8461aa
=> sha256:a158d3b94de3fa813fa6c8c590b8f8a860e015ad4e59bbce5744d2f6fd8461aa 776B / 776B
=> sha256:2cfa3fbb8b6529ee4726b4f599ec27ee557ea3dea7019182323b3779959927f 2.21kB / 2.21kB
=> sha256:24f1d7c1c74a25258bfa6f8bda0b8a727f84819f56f8ca0cabc747976c408 50.45MB / 50.45MB
=> sha256:1d1270f4e622a3c8b3984cde8ea228c17357070b010bb4fb8e2a914085457 7.51kB / 7.51kB
=> sha256:b253aea7ea97e0671bb60080df01de101a38a045f77bc6563b0fbc7c09cca5 7.86MB / 7.86MB
=> sha256:3d2201bd995ccc12851a50820de03d34a17011dcbb9ac9fd3a58c952cbb131 10.00MB / 10.00MB
=> sha256:1de76e268b183d05fa860e0f77951ff50b912b63d29c34f5d6adfd099f5f9ee2 51.88MB / 51.88MB
=> sha256:d9a8df5994511ce28a05e2925a75e8a1acbd0634c39ad734fd4fba8e23d1b1569 191.85MB / 191.85MB
=> extracting sha256:24f1d7c1c74a25258bfa6f8bda0b8a727f84819f56f8ca0cabc747976c408
=> sha256:6f51ee005deac8d99898e41b8ce6ebf250eb1a31a0b03f613aec6bbc9b83d8 4.19kB / 4.19kB
=> sha256:5f32ed3c3f27bedda4fc571c880b5277355a29a8f52b52cdf865f85837ba590 35.24MB / 35.24MB
=> extracting sha256:b253aea7ea97e0671bb60080df01de101a38a045f77bc6563b0fbc7c09cca5
=> extracting sha256:3d2201bd995ccc12851a50820de03d34a17011dcbb9ac9fd3a58c952cbb131
=> extracting sha256:1de76e268b183d05fa860e0f77951ff50b912b63d29c34f5d6adfd099f5f9ee2
=> sha256:0c8cc2f244dc6b6de02e086fc9446b8a541e3acd9ad72d2e99df3ba22f158b3 2.29MB / 2.29MB
=> sha256:0d27a8e61329097574c6766fba9464d8e28d2c8e964e873de352603f22c4ceb
=> extracting sha256:d9a8df5994511ce28a05e2925a75e8a1acbd0634c39ad734fd4fba8e23d1b1569
=> extracting sha256:6f51ee005deac8d99898e41b8ce6ebf250eb1a31a0b03f613aec6bbc9b83d8
=> extracting sha256:5f32ed3c3f27bedda4fc571c880b5277355a29a8f52b52cdf865f85837ba590
=> extracting sha256:0c8cc2f244dc6b6de02e086fc9446b8a541e3acd9ad72d2e99df3ba22f158b3
=> extracting sha256:0d27a8e61329097574c6766fba9464d8e28d2c8e964e873de352603f22c4ceb
=> [internal] load build context
=> == transferring context: 251.85kB
=> [2/5] WORKDIR /usr/src/app
=> [3/5] COPY package*.json /
=> [4/5] RUN npm install
=> [5/5] COPY .
=> exporting to image
=> exporting layers
=> writing image sha256:35556efa157f60465141def3d8662bf5b190f2eb5934fd145f8edd4bfa94244f
=> naming to docker.io/library/desafioapp

What's next:
  View a summary of image vulnerabilities and recommendations + docker scout quickview
PS C:\Users\Michael\Desktop\Desafio 9\nodejs-helloworld-api>
```

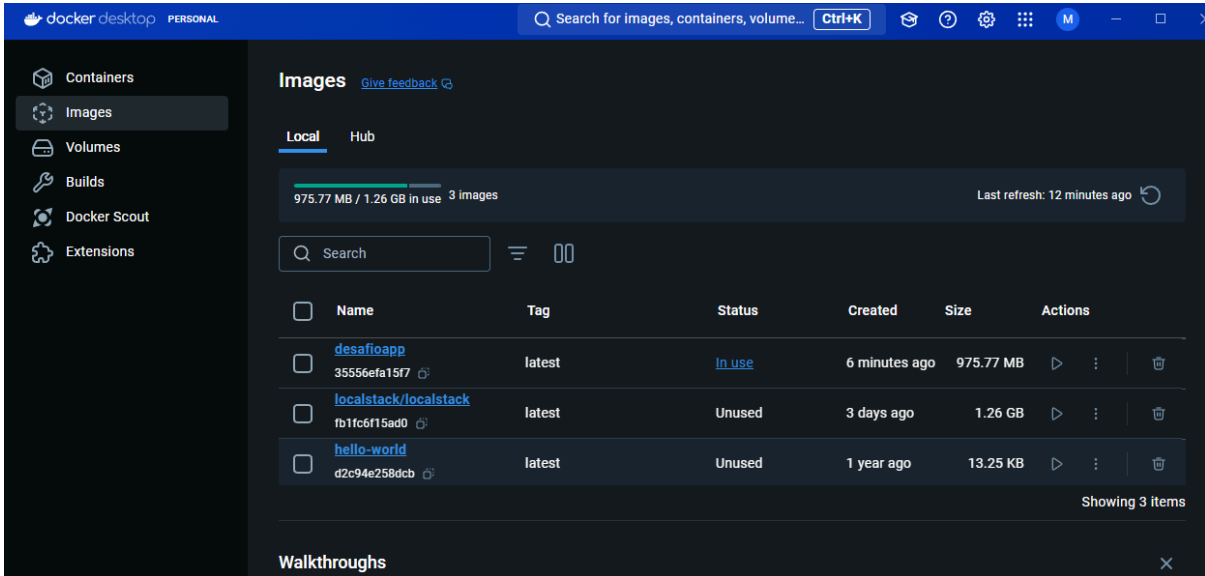
Seguimos con la ejecución del contenedor y para eso usaremos el código en nuestra terminal **docker run -d -p 3000:3000 desafioapp**

```
PS C:\Users\Michael\Desktop\Desafio 9\nodejs-helloworld-api> docker run -d -p 3000:3000 desafioapp
68f053b4d828b62b61bc02377fd6c2a8f26ef2de89212ca645ceb31b3ad77b5
PS C:\Users\Michael\Desktop\Desafio 9\nodejs-helloworld-api>
```

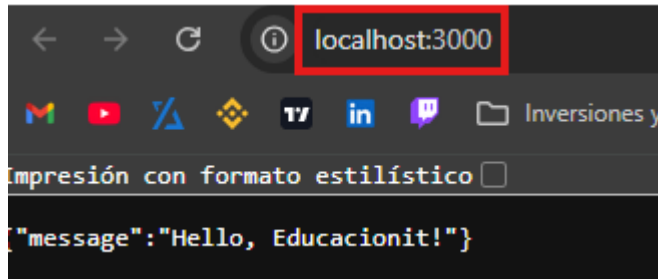
Verificamos que nuestro contenedor esté en ejecución con el comando **docker ps**

```
PS C:\Users\Michael\Desktop\Desafio 9\nodejs-helloworld-api> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
68f053b4d828   desafioapp "docker-entrypoint.s..." About a minute ago Up About a minute   0.0.0.0:3000->3000/tcp, 8080/tcp    hungry_wilson
```

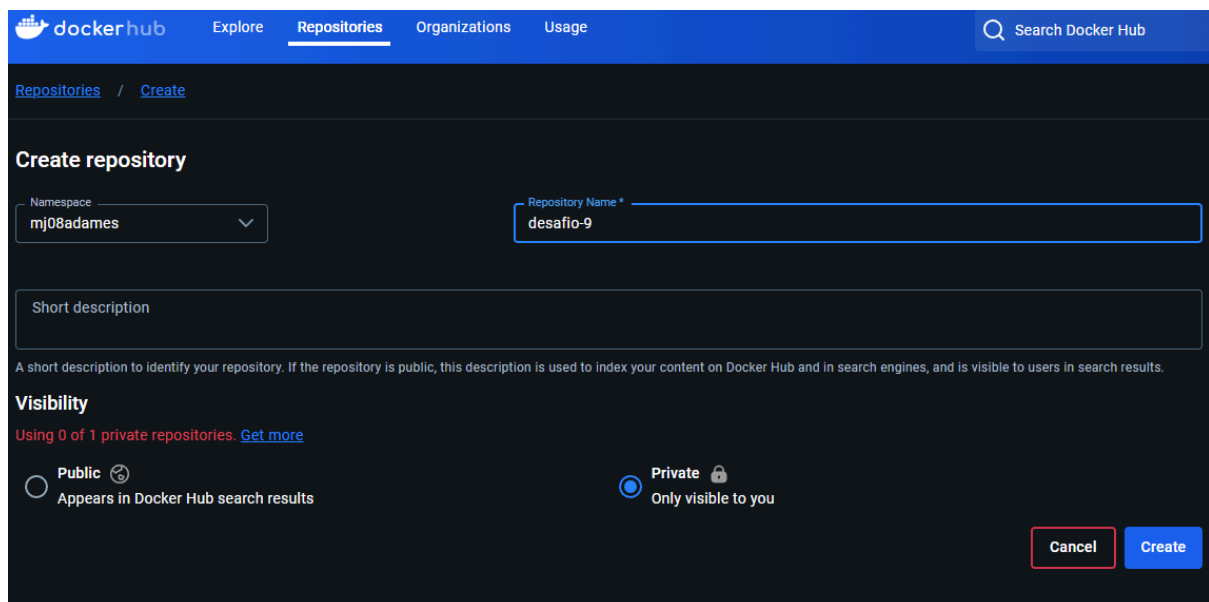
Observamos que en docker este todo lo creado via Powershell



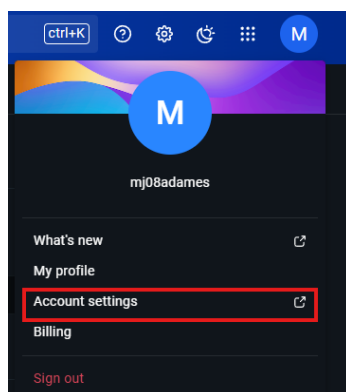
En nuestro navegador colocaremos nuestra direccion ip seguido del puerto <http://localhost:3030> para verificar que nuestra imagen esta corriendo.



Después de tener nuestra imagen y contenedor ejecutándose localmente, el siguiente paso es registrarla en **Docker Hub**. Para hacerlo, primero iniciamos sesión en **Docker Hub** y creamos un repositorio privado. Luego, asignamos un nombre al repositorio; el nombre que elegi es **Desafío-9**



Como creamos nuestro repositorio privado necesitaremos crear un token de seguridad para que nos permita la conexión externa



M

mj08adames
Joined October 19, 2022

General

Account information

Add your account information.

>

Email

m.j.adames08@gmail.com ✓ VERIFIED

>

Password

You can change your password by initiating a reset via email. [Reset password](#)

Security

Two-factor authentication

Two factor authentication is disabled.

>

Personal access tokens

There are 2 personal access tokens associated with your account.


>

Connected accounts

Connect your Docker account to Google or GitHub to sign in using this account.

>

Crear el token y agregarle permisos de lectura

 **docker** EARLY ACCESS

[Settings](#) / [Personal access tokens](#) / New access token

Create access token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access token description

Token Desafio-9

Access permissions

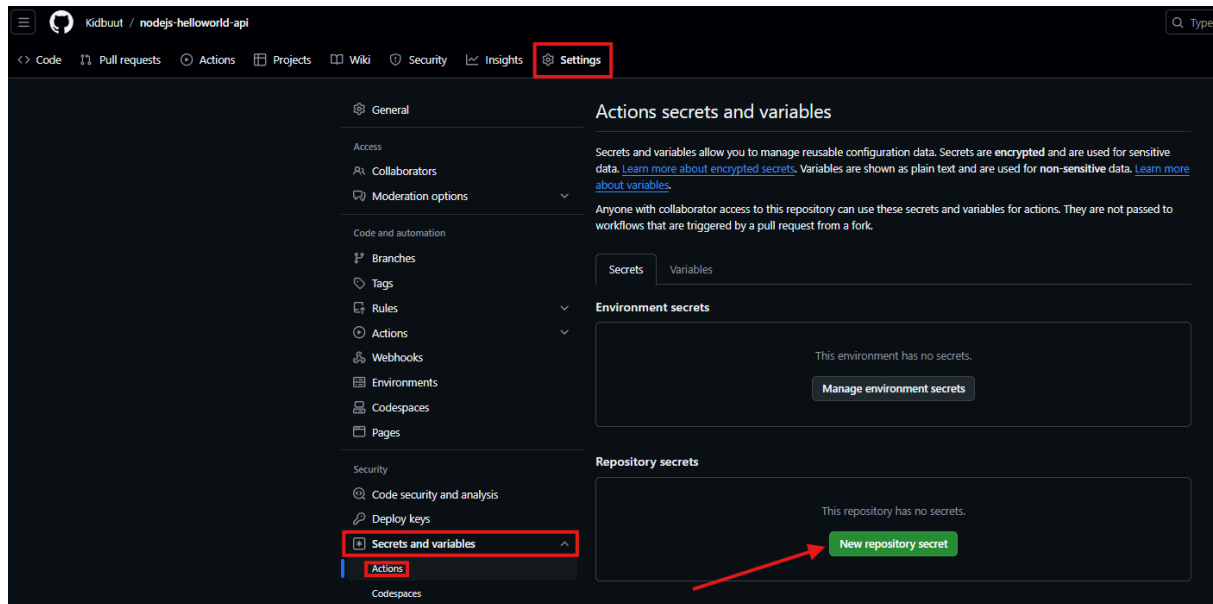
Read-only

Read-only tokens allow you to view, search, and pull images from any public repositories and any private repositories that you have access to.

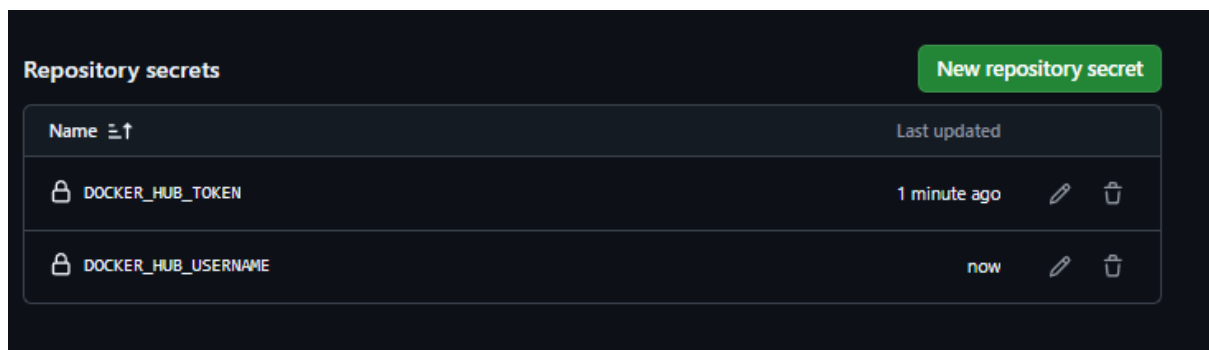
Cancel

Generate

Una vez creado y guardado nuestro token nos dirigimos hasta Github a configurar nuestras credenciales de Docker Hub.



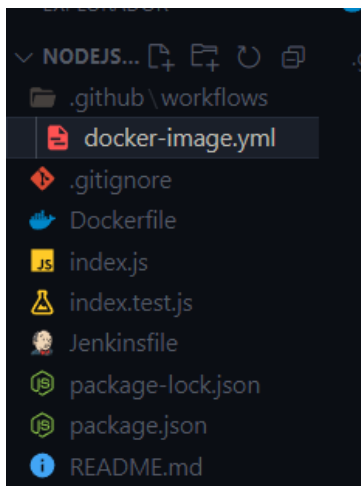
agregamos nuestro Usuario y token generado en docker hub



El siguiente paso consiste en establecer un workflow en GitHub Actions que emplee estos secrets para cargar la imagen. Para hacerlo, sigue los siguientes pasos:

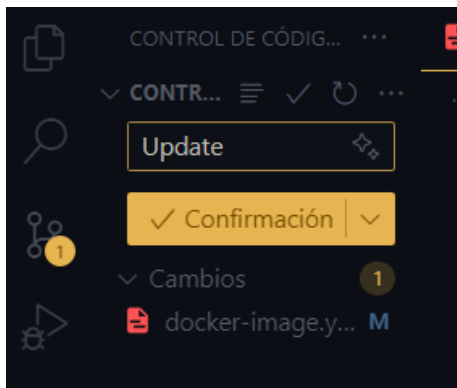
Creamos el directorio para los Workflows: En tu repositorio, genera una carpeta llamada `.github/workflows/`.

Añadimos al archivo de Configuración: En esa carpeta, crea un archivo con el nombre `docker-image.yml`. Este archivo especificará el proceso para compilar y subir la imagen de Docker utilizando los secrets que ya has configurado.

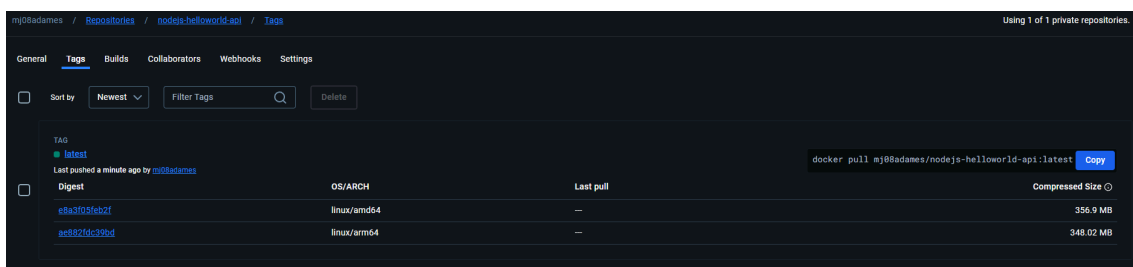
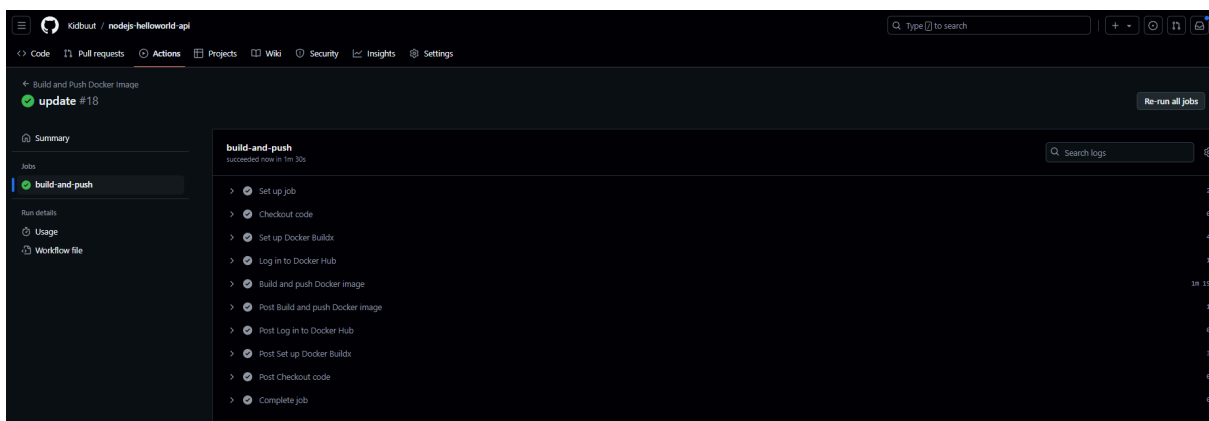
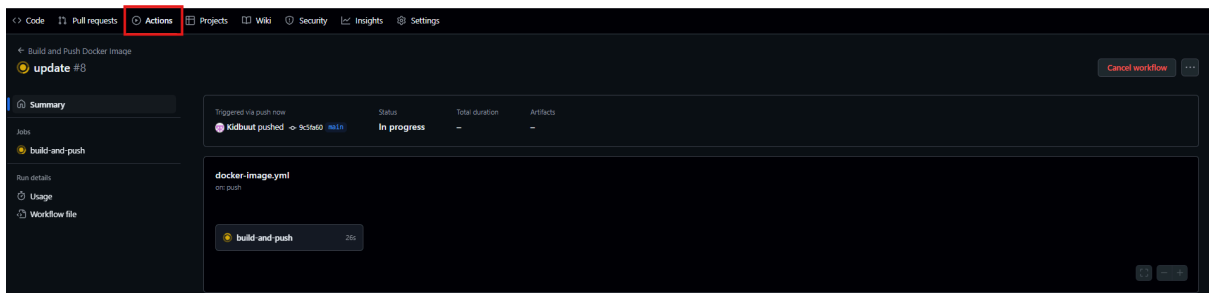


```
docker-image.yml X
.github > workflows > docker-image.yml
1  name: Build and Push Docker Image
2
3  on:
4    push:
5      branches:
6        - main
7
8  jobs:
9    build-and-push:
10     runs-on: ubuntu-latest
11     steps:
12       - name: Checkout code
13         uses: actions/checkout@v2
14
15       - name: Set up Docker Buildx
16         uses: docker/setup-buildx-action@v2
17
18       - name: Log in to Docker Hub
19         uses: docker/login-action@v2
20         with:
21           username: ${ secrets.DOCKER_HUB_USERNAME }
22           password: ${ secrets.DOCKER_HUB_TOKEN }
23
24       - name: Build and push Docker image
25         uses: docker/build-push-action@v4
26         with:
27           context: .
28           push: true
29           platforms: linux/amd64,linux/arm64
30           tags: ${ secrets.DOCKER_HUB_USERNAME }/nodejs-helloworld-api:latest
```

hacemos push para modificar y guardar los cambios de nuestro repositorio forkeado



Al hacer push, nos vamos hacia github y clickeamos en el apartado de **Actions** y veremos cómo se construye nuestra imagen en docker.



Al culminar la carga vamos a **Docker Hub** y vemos que están creadas nuestras imagenes.