

## **Desafío número 9 - Bootcamp Devops Engineer.**

**Alumno: Ignacio Peretti**

Objetivo:

Este desafío integra una práctica que vimos en clase sobre Docker, GitHub Actions y la configuración de la registry de Docker Hub.

El objetivo es crear un pipeline completo que realice el build y delivery de nuestra aplicación como una imagen de contenedor y la publique en una registry.

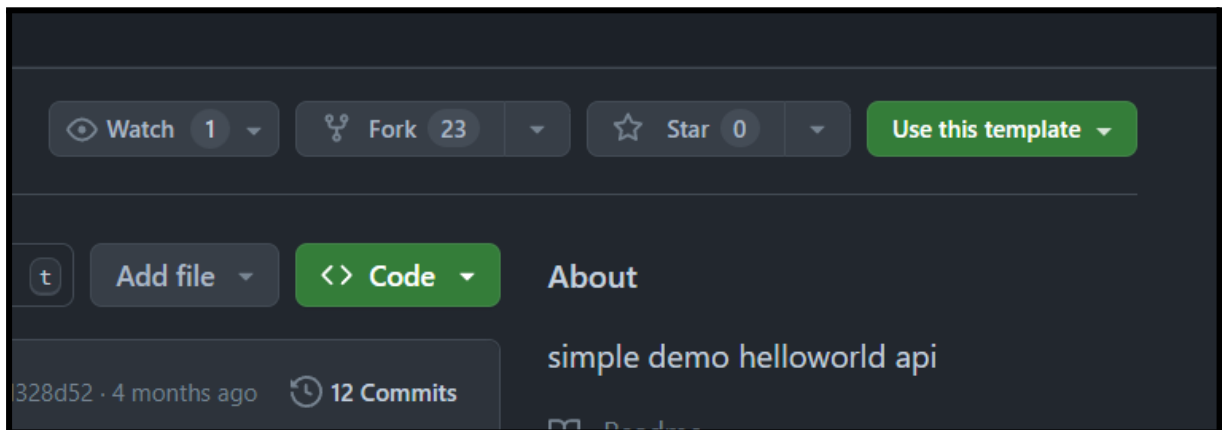
Requisitos:

1. Crear un nuevo repositorio a partir del template nodejs-helloworld-api
2. Para alojar la imagen de esta aplicación crea un repositorio privado en tu cuenta de Docker Hub.
3. Genera un token en tu cuenta de Docker Hub.
4. Configurar las el entorno de github action para autenticar con Docker Hub.
5. Agregar los archivos necesarios para construir la imagen del contenedor.
6. Implementar un github action multiplataforma.

## Creación de un repositorio en github a partir del template **nodejs-helloworld-api**.

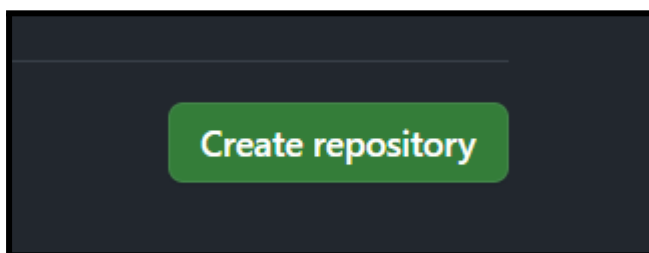
Accedemos a github y nos dirigimos a este repositorio, [nodejs-helloworld-api](#).

Daremos clic en el botón "Use this template" en la parte superior derecha.



Completamos la información necesaria, como el nombre del repositorio y la descripción.

Haz clic en "Create repository from template".



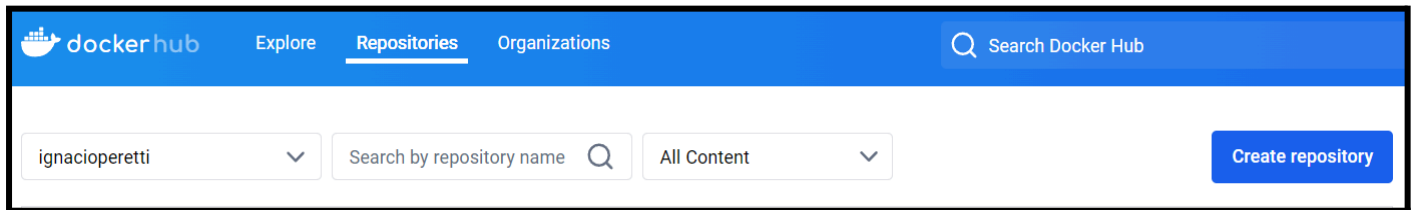
Listo, ya tenemos creado el repositorio.

Lo siguiente será crearnos un repositorio en nuestra cuenta de DockerHub para alojar nuestra imagen.

## Crear un repositorio privado en **Docker Hub**.

Accedemos a Docker Hub e iniciamos sesión con nuestra cuenta.

En la página principal hacemos click en “Create Repository”



Completa la información necesaria, como el nombre del repositorio y asegúrate de marcar la opción "Private".

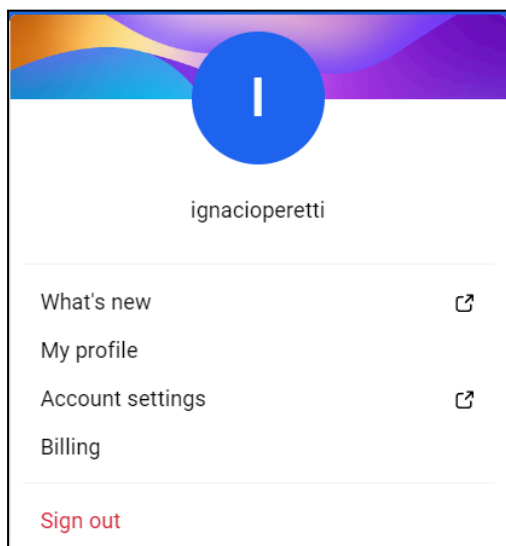
Haz clic en "Create" para finalizar la creación del repositorio.

Con esto ya tendremos nuestro repositorio en DockerHub para alojar nuestra imagen.

En el siguiente paso generamos un token en DockerHub

## Generar un token en **Docker Hub**.

En Docker Hub, ve a tu perfil y selecciona "Account Settings".



En la pestaña "Security", busca la sección "Personal Access Token"

Security

Two-factor authentication

Two factor authentication is disabled.

Personal access tokens

There are 2 personal access tokens associated with your account.

Dentro, seleccionamos "Generate New Token", añadimos un nombre y le añadimos permisos de Lectura, Escritura y Borrado. (**Read,Write,Delete**)

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

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

Cancel

Generate

**Copia el token generado, ya que lo necesitarás para autenticar GitHub Actions.**

**Ya tenemos configurado nuestro repositorio en DockerHub.**

ignacioperetti

Search by repository name

All Content

Create repository

ignacioperetti / nodejs-helloworld-api

Contains: Image • Last pushed: about 1 hour ago

☆ 0

📄 2

🌐 Public

🛡 Scout inactive

En el siguiente paso Configuraremos un entorno de github actions para autenticarnos con docker hub.

## Configurar el entorno de GitHub Actions para autenticar con Docker Hub

Agregar los secretos en GitHub:

Ve a tu repositorio en GitHub.

Haz clic en "Settings" > "Secrets and variables" > "Actions" > "New repository secret".

Agrega los siguientes secretos:

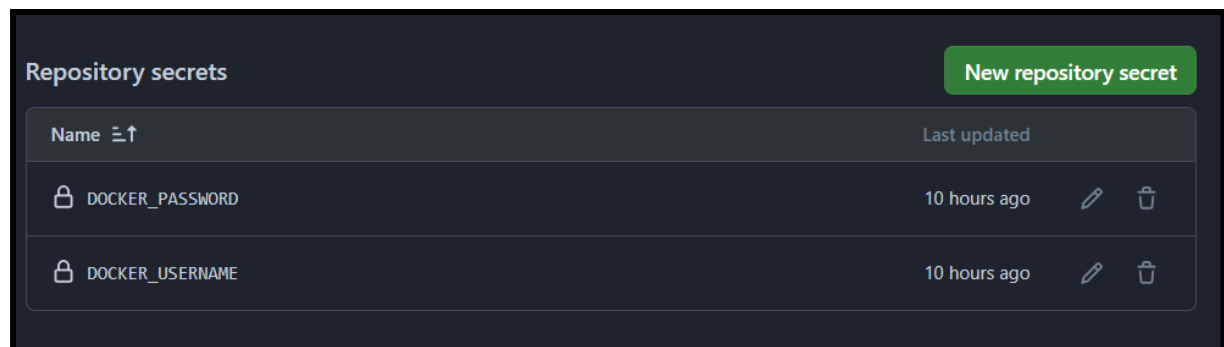
En el apartado **Name\*** pondremos DOCKER\_HUB\_USERNAME

En el apartado **Secret\*** Tu nombre de usuario de Docker Hub.

Generamos otro secreto con estos valores

**Name\*** DOCKER\_HUB\_ACCESS\_TOKEN

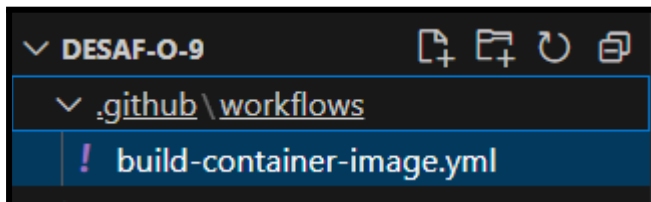
**Secret\*** El token que generaste en el paso anterior.



Debería quedarte así, con esto ya tenemos configurados nuestros secretos.

## Configurar GitHub Actions

En nuestro repositorio, tendremos que crear a nivel raíz una carpeta llamada **.github** y dentro de esta otra llamada **workflows**, dentro tendremos el archivo que se encargará de hacer el build and push.



Luego necesitamos agregar los archivos necesarios para construir la imagen del contenedor.

Mencionado anteriormente dentro de mi carpeta workflows se aloja mi pipeline de CI/CD para construir y publicar una imagen de Docker en Docker Hub.

Este archivo lo nombre como `build-container-image.yml`

**Este archivo de configuración de GitHub Actions hace lo siguiente:**

**Disparadores:** Se ejecuta en eventos de push y pull\_request a la rama main si los cambios afectan a ciertos archivos.

**Configuración del Entorno:** Usa una máquina virtual con Ubuntu.

### Pasos del Workflow:

Clona el repositorio.

Configura Docker Buildx.

Se autentica en Docker Hub.

Construye y publica una imagen de Docker.

Con esta configuración, cada vez que se realice un push a la rama main o se actualice un pull request hacia esa rama, GitHub Actions construirá una nueva imagen de Docker y la publicará en Docker Hub.

```
name: Build and Push Docker Image - test

on:
  push:
    branches: [ "main" ]
    paths:
      - "Desafio-9/**"
      - ".github/workflows/build-container-image.yml"
  pull_request:
    branches: [ "main" ]
    paths:
      - "Desafio-9/**"
      - ".github/workflows/build-container-image.yml"

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
      - name: Check out the code
        uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Login to DockerHub
        uses: docker/login-action@v1
        with:
          username: ${ secrets.DOCKER_USERNAME }
          password: ${ secrets.DOCKER_PASSWORD } # token dockerhub

      - name: Build and push
        uses: docker/build-push-action@v2
        with:
          context: ${ github.workspace }
          file: ${ github.workspace }/Dockerfile.multi
          # Asume que tu Dockerfile está en la raíz de tu repositorio
          # Si tu Dockerfile está en una ruta específica, por ejemplo, ./path/to/Dockerfile, cambia 'context' y 'file' como sigue:
          # context: ./path/to/
          # file: ./path/to/Dockerfile
          push: true
          tags: ignacioperetti/nodejs-helloworld-api:latest # Cambia esto con tu nombre de usuario y nombre de imagen
```

Luego tendremos que crear un archivo **Dockerfile** en la raíz.

Este archivo Dockerfile define un proceso de construcción de una imagen Docker para una aplicación Node.js.

Utiliza la técnica de **multistage builds** (construcción de múltiples etapas) para optimizar la imagen final.

El contenido de este es el siguiente

```
1 FROM node:17.9.0 AS base
2
3 WORKDIR /usr/src/app
4 COPY package*.json ./
5 RUN npm install
6 COPY . .
7
8 # for lint
9
10 FROM base AS linter
11
12 WORKDIR /usr/src/app
13 RUN npm run lint
14
15 # for build
16
17 FROM linter AS builder
18
19 WORKDIR /usr/src/app
20 RUN npm run build
21
22 # for production
23
24 FROM node:17.9.0-alpine3.15
25
26 WORKDIR /usr/src/app
27 COPY package*.json ./
28 RUN npm install --only=production
29 COPY --from=builder /usr/src/app/dist ./
30 EXPOSE 3000
31
32 ENTRYPOINT ["node", "./app.js"]
```



## Resumen del código.

**Eta** **Base**: Prepara el entorno de trabajo y las dependencias.

**Eta** **de Lint**: Ejecuta herramientas de linting para asegurar la calidad del código

**Eta** **de Construcción**: Construye la aplicación para producción.

**Eta** **de Producción**: Prepara una imagen ligera con solo las dependencias de producción y los archivos contruidos, optimizada para ejecutar la aplicación.

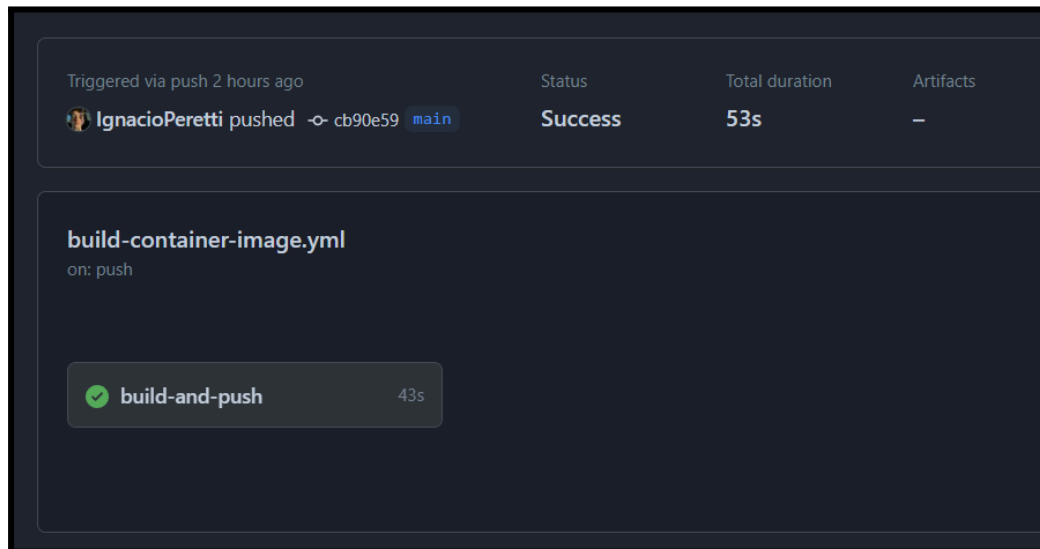
## Instrucciones para ejecutar el pipeline de GitHub Actions.

Una vez que realicemos un push o un pull request se activará y GitHub Actions construirá una nueva imagen de Docker y la publicará en Docker Hub.

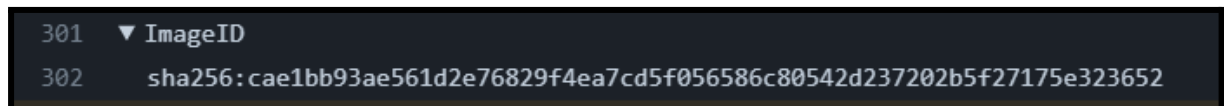
Esto podemos verlo si nos dirigimos a la sección Actions en GitHub.

The screenshot shows the GitHub Actions interface for the repository 'Desafio-9' by user 'IgnacioPeretti'. The top navigation bar includes links to Code, Issues, Pull requests, Actions (selected), Projects, Wiki, Security, Insights, and Settings. A search bar is located on the right. The left sidebar shows a list of workflows, with 'Build and Push Docker Image - test' highlighted. The main area displays 'All workflows' with a search bar and a list of workflow runs. One workflow run is shown for the 'push' event on the 'main' branch, with a status of 'succeeded' and a duration of 53s. The interface is in dark mode.

Si damos click en la salida “build-and-push” podremos ver más a detalle.



En la sección build and push - IMAGEID podemos ver la imagen creada.




Al igual que en la sección de Metadata

```
"buildx.build.ref": "builder-926f506f-f2fe-4573-92ba-9367d04f27cf/builder-926f506f-f2fe-4573-92ba-9367d04f27cf0/mg4yp5524rpmjyfwssgo5d85d",
"containerimage.descriptor": {
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "digest": "sha256:cae1bb93ae561d2e76829f4ea7cd5f056586c80542d237202b5f27175e323652",
  "size": 856
},
"containerimage.digest": "sha256:cae1bb93ae561d2e76829f4ea7cd5f056586c80542d237202b5f27175e323652",
"image.name": "***-helloworld-api:latest"
```


Si seguimos recorriendo el total de la salida podemos ver que todo se ha ejecutado correctamente.



Luego de comprobar la salida en Github, nos dirigimos a nuestro repositorio en DockerHub para comprobar que este alojada allí la imagen en nuestro repositorio.

En repositorios podremos ver que tenemos agregado un nuevo tag.

ignacioperetti/nodejs-helloworld-api 



Updated about 2 hours ago

Desafio-9 

This repository does not have a category   INCOMPLETE

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 latest		Image	an hour ago	2 hours ago

[See all](#)

Si podemos visualizar la imagen en nuestro repositorio es porque pudimos subir bien la imagen.

Con esto finalizamos el desafío 9.