

# Documentación Técnica: Dockerización de una API y uso de workflows

Autor: Pedro José Meixús Belsol

4 de diciembre de 2025

# Índice

<b>1. Preparativos</b>	<b>2</b>
<b>2. Dockerización de API</b>	<b>2</b>
2.1. Creación .dockerignore	2
2.2. Creación dockerfile	2
2.3. Construcción de la imagen	3
2.4. Creación docker-compose	3
2.5. Subir imagen a DockerHub	4
<b>3. Workflows</b>	<b>4</b>
3.1. Subida automática a DockerHub	4
3.2. Enviar mensaje automático a Discord	6

# 1. Preparativos

Antes de empezar a meternos con Docker, lo primero que vamos a necesitar serán dos cosas:

- Una base de datos
- Una API que se conecte a dicha base de datos

Para la base de datos utilizamos Mongo de manera local, de preferencia, ya metida en Docker previamente, por su parte la API laaremos en Express con JavaScript.

Cabe destacar que la BBDD guardará empleados y grupos (que contienen empleados).

## 2. Dockerización de API

### 2.1. Creación .dockerignore

Lo primero que haremos en nuestro proyecto, será añadir el archivo `.dockerignore` para evitar problemas de seguridad. En él, añadiremos los archivos que no queremos que se suban a DockerHub, y contendrá lo siguiente:

```
1 node_modules
2 npm-debug.log
3 .git
4 .gitignore
5 .env
6 .DS_Store
7 README.md
8 *.md
```

Código 1: Contenido de `.dockerignore`

### 2.2. Creación dockerfile

Ahora crearemos nuestro `dockerfile`, que será el encargado de indicar las instrucciones a Docker a la hora de construir una imagen.

En este archivo indicaremos:

- La imagen base que usará, en nuestro caso Node.
- Los archivos de dependencias que llevará del proyecto a la imagen mediante `COPY`.
- Los comandos de instalación para las dependencias necesarias con `RUN`.
- Llevaremos el resto de proyecto con otro `COPY`.

- El puerto que se expondrá.
- Los comandos que debe ejecutar cuando se arranque el contenedor.

### 2.3. Construcción de la imagen

Con lo anterior ya preparado, podremos construir la imagen. Lo haremos con el siguiente comando:

```
PS C:\Users\pmeibel\Desktop\laburo\MongoDocker> docker build -t ejemplo-api:v1.0.0 .
```

El comando monta la imagen con lo que hemos indicado en el dockerfile, además de tener el tag que indicamos en el comando (después del -t).

Para probar si funciona correctamente podemos ejecutar el siguiente comando:

```
PS C:\Users\pmeibel\Desktop\laburo\MongoDocker> docker run --rm -it -p 3000:3000 ejemplo-api:v1.0.0
```

Una vez ejecutado, nos debería indicar por logs si ha arrancado todo correctamente, en cualquier caso siempre podemos comprobarlo accediendo a localhost en el puerto que le hayamos indicado (3000 en nuestro caso).

### 2.4. Creación docker-compose

Teniendo ya la API y la BBDD correctamente montadas, vamos a crear nuestro docker-compose que nos permitirá levantar ambos contenedores con un solo archivo y comando.

Dentro de este indicaremos nuestros servicios que serán la API por un lado y la BBDD por otro. Ambos tendrán los siguientes parámetros:

- Image: Donde indicaremos la imagen que utilizará, la BBDD, usará la propia de mongo, la API usará la que montamos anteriormente.
- Ports: Aquí indicaremos los puertos que usará cada servicio.
- Environment: Donde guardaremos las variables de entorno necesarias.

A mayores de esto, en la API indicaremos que arranque una vez la BBDD lo haga con **"depends\_on"**, y en la BBDD indicamos donde debe guardar los datos con **"volumes"**

Con el docker-compose terminado podemos probarlo con el siguiente comando:

```
PS C:\Users\pmeibel\Desktop\laburo\MongoDocker> docker-compose up --build -d
```

## 2.5. Subir imagen a DockerHub

Ahora que lo tenemos todo montado correctamente, nos falta subir la imagen a DockerHub, esto podemos hacerlo con dos comandos, el primero para logearnos:

```
● PS C:\Users\pmeibel\Desktop\laburo\MongoDocker> docker login
Authenticating with existing credentials... [Username: starlili]

💡 Info → To Login with a different account, run 'docker Logout' followed by 'docker Login'

Login Succeeded
```

Y un segundo para pushear nuestra imagen a DockerHub:

```
PS C:\Users\pmeibel\Desktop\laburo\MongoDocker> docker push starlili/ejempli-api:v1.0.0
```

## 3. Workflows

### 3.1. Subida automática a DockerHub

Para esta última parte crearemos una la siguiente ruta dentro de nuestro proyecto ”.github/workflows/docker-push.yml”. El archivo docker-push será el encargado de automatizar la construcción y subida de imágenes a DockerHub. En el usaremos los ”secrets” de GitHub para guardar información sensible.

Dentro del mismo debemos indicar bastantes parámetros así que vamos a ir viéndolos poco a poco:

Empezamos el archivo indicando el nombre de nuestro workflow, seguido del disparador, que es nuestro caso será un push a las rama main o master.

```
1 name: Build and Push Docker Image Pedro
2
3 on:
4   push:
5     branches:
6       - main
7       - master
8   workflow_dispatch:
```

Código 2: Nombre workflow y disparador

Seguido de esto tenemos la declaración de variables de entorno, donde guardaremos el nombre de nuestra imagen (incluyendo el secret de nuestro usuario) y el tag de la misma.

```

1 env:
2   DOCKER_IMAGE_NAME: ${{ secrets.DOCKER_USERNAME }}/ejemplo-api
3   DOCKER_TAG: latest

```

Código 3: Variables de entorno

A continuación declaramos los trabajos o ”**jobs**” que ejecutará una vez el disparador se active, en nuestro caso solo ejecuta el trabajo ”**build-and-push**” que se ejecutarán en un runner (máquina virtual) Ubuntu que nos proporciona GitHub.

```

1 jobs:
2   build-and-push:
3     runs-on: ubuntu-latest

```

Código 4: Trabajos definidos

A partir de aquí indicaremos los pasos (steps) que queremos que siga, como en nuestro caso queremos subir la imagen a DockerHub haremos los pasos pertinentes.

Empezaremos clonando nuestro código/proyecto a la máquina virtual, después configuraremos el Docker BuildX para que le permita construir imágenes. Después haremos login en nuestra cuenta (indicando los secrets) y ya le indicamos que construya la imagen.

```

1 steps:
2   - name: Checkout codigo
3     uses: actions/checkout@v4
4
5   - name: Configurar Docker Buildx
6     uses: docker/setup-buildx-action@v3
7
8   - name: Login a Docker Hub
9     uses: docker/login-action@v3
10    with:
11      username: ${{ secrets.DOCKER_USERNAME }}
12      password: ${{ secrets.DOCKER_PASSWORD }}
13
14   - name: Construir y subir imagen Docker
15     uses: docker/build-push-action@v5
16    with:
17      context: .
18      file: ./dockerfile
19      push: true
20      tags: ${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}
21      cache-from: type=registry,ref=${{ env.DOCKER_IMAGE_NAME }}:
22        buildcache
23      cache-to: type=inline

```

---

### Código 5: Pasos definidos

Ahora podemos hacer que nos muestre si la imagen se ha construido simplemente añadiendo un step para ello.

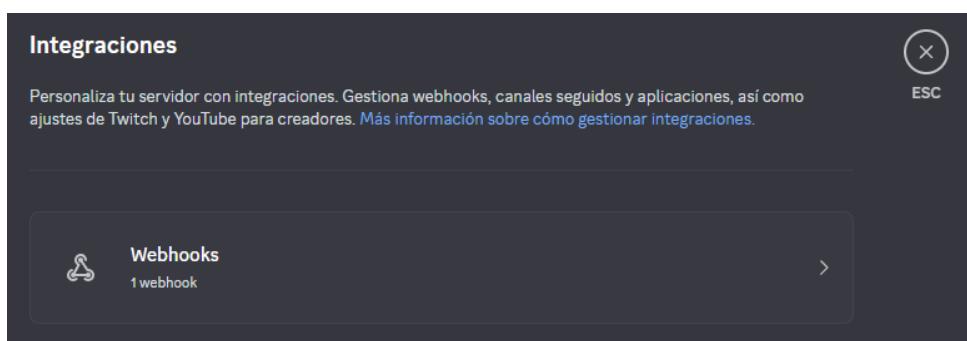
```
1  - name: Mostrar informacion de la imagen
2    run: |
3      echo "Imagen construida y subida exitosamente:"
4      echo "  - Imagen: ${{ env.DOCKER_IMAGE_NAME }}:${{ env.
5        DOCKER_TAG }}"
6      echo "  - Docker Hub: https://hub.docker.com/r/${{ secrets.
7        DOCKER_USERNAME }}/ejemplo-api"
```

Código 6: Log para mostrar información

## 3.2. Enviar mensaje automático a Discord

En mi caso he decidido hacer que el mensaje se envie a Discord.

Para hacer esto, primero iremos a un servidor nuestro y accederemos a su configuración, luego iremos a la sección de **Integraciones**.



Aquí crearemos un nuevo **Webhook**, donde le daremos un nombre y el canal donde queremos que se envíe el mensaje.

The screenshot shows the 'Integraciones > Webhooks' section in the Discord developer portal. It features a blue button labeled 'Nuevo webhook'. Below it, a section titled 'Publicando en #cosas' shows a webhook named 'Captain Hook' created by 'starlili' at 4 dic 2025. The webhook icon is orange with a white Discord logo.

Ahora copiaremos la URL del webhook que no da y la guardaremos en los secrets de GitHub con el nombre **DISCORD\_WEBHOOK**.

The screenshot shows the 'Repository secrets' page in GitHub. It lists three secrets: 'DISCORD\_WEBHOOK' (last updated 6 hours ago), 'DOCKER\_PASSWORD' (last updated last week), and 'DOCKER\_USERNAME' (last updated last week). Each secret has edit and delete icons next to it. A green button labeled 'New repository secret' is visible in the top right corner.

Por último, añadiremos un nuevo step al workflow anterior para que envíe el mensaje a Discord una vez la imagen se haya subido correctamente.

```
1 - name: Discord Webhook
2   run: |
3     curl -H "Content-Type: application/json" \
4       -d "{\"content\": \"Imagen Docker actualizada y subida \
5         correctamente a Docker Hub!\"}" \
      ${secrets.DISCORD_WEBHOOK}
```

Código 7: Step para enviar mensaje a Discord

Ahora, cada vez que hagamos un push a main o master, se construirá y subirá la imagen automáticamente, y además se enviará un mensaje a nuestro canal de Discord indicándonos que se ha subido correctamente.

The screenshot shows a message from the 'Captain Hook APP' channel at 18:47. The message content is 'Imagen Docker actualizada y subida correctamente a Docker Hub!'. The message is displayed in a dark-themed interface with a blue circular icon containing a white Discord logo.

Con esto finalizamos la documentación técnica sobre la dockerización de una API y el uso de workflows para automatizar la subida a DockerHub.