Dockerización de API y BBDD

Programación de Servicios y Procesos

Adrián Condines Celada

Aula Estudio

2º Ciclo Superior - Desarrollo de Aplicaciones Multiplataforma

Curso 2025 - 2026

Índice

1. INTRODUCCIÓN Y OBJETIVOS

En esta práctica se documenta la API utilizada en esta práctica, así como el proceso de creación y subida de una imagen de la API a Docker Hub y el proceso de automatización de nuevas versiones utilizando GitHub Actions.

2. API

Está sección detalla los endpoints disponibles en la API para gestionar los usuarios y los grupos.

2.1. Listar todos los usuarios

2.1.1. Descripción

Obtiene una lista de todos los registros almacenados en la tabla `users`.

- **Método:** GET
- **URL:** `/api/usuarios`

2.1.2. Ejemplo de Input

No requiere ningún input (ni parámetros en la URL ni body).

2.1.3. Ejemplo de Output (Éxito)

Responde con un array de objetos JSON, cada uno representando un rapero.

```
[
    {
        "id": 1,
        "name": "Bruno"
    },
    {
        "id": 2,
        "name": "Adriano"
    }
]
```

2.1.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al devolver los datos de los usuarios: (error)"
}
```

2.2. Listar un usuario en concreto

2.2.1. Descripción

Obtiene un solo registro en la tabla `users`.

- **Método:** GET
- **URL:** `/api/usuarios/:id`

2.2.2. Ejemplo de Input

Parámetro `id` en la URL.

2.2.3. Ejemplo de Output (Éxito)

Responde con un JSON en el que se incluye el usuario a buscar.

```
[
    {
        "id": 1,
        "name": "Bruno"
    }
]
```

2.2.4. Ejemplo de Output (Error)

```
\justify
Responde con un JSON incluyendo un mensaje de error.

{
  "error": "Error al devolver los datos del usuario: (error)"
}
```

2.3. Añadir un usuario

2.3.1. Descripción

Añade un registro a la tabla `users`.

- **Método:** POST
- **URL:** `/api/usuarios`

2.3.2. Ejemplo de Input

Un objeto JSON en el body de la petición con el nombre del nuevo usuario.

```
{
    "name": "Adriano"
}
```

2.3.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 201.

```
{
    "success": "Los datos se han insertado correctamente"
}
```

2.3.4. Ejemplo de Output (Error)

```
\justify
Responde con un JSON incluyendo un mensaje de error.

{
  "error": "Error al añadir el usuario: (error)"
}
```

2.4. Eliminar un usuario

2.4.1. Descripción

Elimina un registro de la tabla `users`.

- **Método:** DELETE
- **URL:** `/api/usuarios/:id`

2.4.2. Ejemplo de Input

Parámetro `id` en la URL.

2.4.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 201.

```
{
    "success": "Los datos se han insertado correctamente"
}
```

2.4.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al añadir el usuario: (error)"
}
```

2.5. Cambiar el nombre de un usuario

2.5.1. Descripción

Modifica el campo **name** de un registro de la tabla `users`.

- **Método:** PUT
- **URL:** `/api/usuarios/:id`

2.5.2. Ejemplo de Input

Parámetro `id` en la URL y un objeto JSON en el body de la petición con el nombre del nuevo usuario.

```
{
    "name": "Adriano"
}
```

2.5.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 200.

```
{
    "success": "Se ha modificado registro correctamente"
}
```

2.5.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al modificar el nombre del usuario: (error)"
}
```

2.6. Listar todos los grupos

2.6.1. Descripción

Obtiene una lista de todos los registros almacenados en la tabla `groups`.

- **Método:** GET
- **URL:** `/api/grupos`

2.6.2. Ejemplo de Input

No requiere ningún input (ni parámetros en la URL ni body).

2.6.3. Ejemplo de Output (Éxito)

Responde con un array de objetos JSON, cada uno representando un rapero.

```
[
    {
        "id": 1,
        "name": "Aulaestudienses"
    },
    {
        "id": 2,
        "name": "Antijavanianos"
    }
]
```

2.6.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al devolver los datos de los grupos: (error)"
}
```

2.7. Listar un grupo en concreto

2.7.1. Descripción

Obtiene un solo registro en la tabla `groups`.

- **Método:** GET
- **URL:** `/api/grupos/:id`

2.7.2. Ejemplo de Input

Parámetro `id` en la URL.

2.7.3. Ejemplo de Output (Éxito)

Responde con un JSON en el que se incluye el grupo a buscar.

```
[
    {
        "id": 1,
        "name": "Aulaestudienses"
    }
]
```

2.7.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al devolver los datos del usuario: (error)"
}
```

2.8. Añadir un grupo

2.8.1. Descripción

Añade un registro a la tabla `users`.

- **Método:** POST
- **URL:** `/api/grupos`

2.8.2. Ejemplo de Input

Un objeto JSON en el body de la petición con el nombre del nuevo grupo.

```
{
    "name": "Aulaestudienses"
}
```

2.8.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 200.

```
{
    "success": "Los datos se han insertado correctamente"
}
```

2.8.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al añadir el usuario: (error)"
}
```

2.9. Eliminar un grupo

2.9.1. Descripción

Elimina un registro de la tabla `groups`.

- **Método:** DELETE
- **URL:** `/api/grupos/:id`

2.9.2. Ejemplo de Input

Parámetro `id` en la URL.

2.9.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 201.

```
{
    "success": "Grupo eliminado correctamente"
}
```

2.9.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al eliminar el grupo: (error)"
}
```

2.10. Añadir un usuario a un grupo

2.10.1. Descripción

Añade un registro a la tabla `users_groups`.

- **Método:** POST
- **URL:** `/api/grupos/:id_grupo/:id_usuario`

2.10.2. Ejemplo de Input

Parámetros `id_grupo` y `id_usuario` en la URL.

2.10.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 200.

```
{
    "success": "Los datos se han insertado correctamente"
}
```

2.10.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al añadir el usuario al grupo: (error)"
}
```

2.11. Eliminar un usuario de un grupo

2.11.1. Descripción

Elimina un registro de la tabla `users_groups`.

- **Método:** DELETE
- **URL:** `/api/grupos/:id_grupo/:id_usuario`

2.11.2. Ejemplo de Input

Parámetros `id_grupo` y `id_usuario` en la URL.

2.11.3. Ejemplo de Output (Éxito)

Responde con un mensaje de confirmación y un estado 200.

```
{
    "success": "Usuario eliminado del grupo correctamente"
}
```

2.11.4. Ejemplo de Output (Error)

Responde con un JSON incluyendo un mensaje de error.

```
{
  "error": "Error al añadir el usuario al grupo: (error)"
}
```

3. ESTRUCTURA DEL PROYECTO

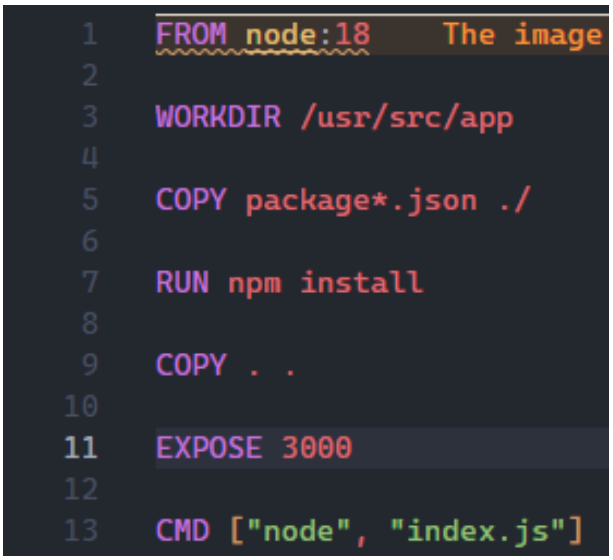
En esta sección se define la estructura del proyecto:

```
api_usuarios_grupos/
|-- .github/
|-- db/
|-- DOCUMENTACIÓN/
|   |-- images/
|   |-- DOCUMENTACIÓN API Y BBDD - ADRIANO.tex
|   |-- DOCUMENTACIÓN API Y BBDD - ADRIANO.pdf
|-- .dockerignore
|-- .env.example
|-- docker-compose.yaml
|-- Dockerfile
|-- ejemplo ci-cd.txt
|-- index.js
|-- package-lock.json
|-- package.json
```

4. DOCKERIZACIÓN Y SUBIDA DE IMAGEN

El primer requisito para esta sección de la práctica es crear una cuenta en Docker Hub y una vez creada, se debe inicar sesión con esa misma cuenta en la aplicación de Docker Desktop.

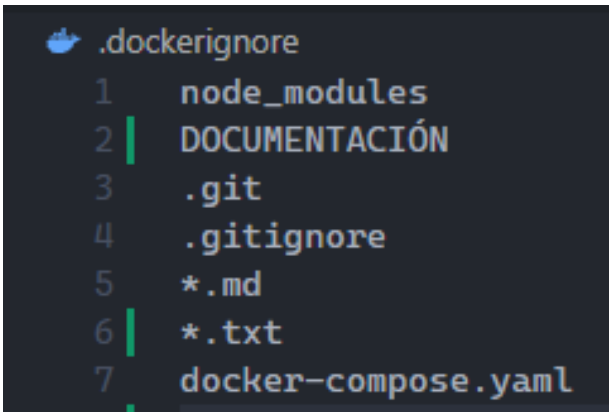
Una vez creada la cuenta, se creará el archivo Dockerfile con la siguiente estrcutura:



En el archivo Dockerfile se incluyen las siguientes sentencias:

- **FROM**: Indica una imagen base necesaria para la creación de la imagen de la API.
- **WORKDIR**: Indica el directorio de trabajo.
- **COPY**: Copia el archivo `package.json` y `package-lock.json` al directorio de trabajo.
- **RUN**: Ejecuta el comando `npm install` para instalar las dependencias.
- **COPY**: Copia el resto de archivos al directorio de trabajo.
- **EXPOSE**: Indica el puerto en el que se ejecutará la API.
- **CMD**: Indica el comando que se ejecutará al iniciar el contenedor.

También se debe crear un archivo `.dockerignore` para evitar que se suban archivos innecesarios a la imagen que se subirá a Docker Hub. En este caso tiene la siguiente estructura:

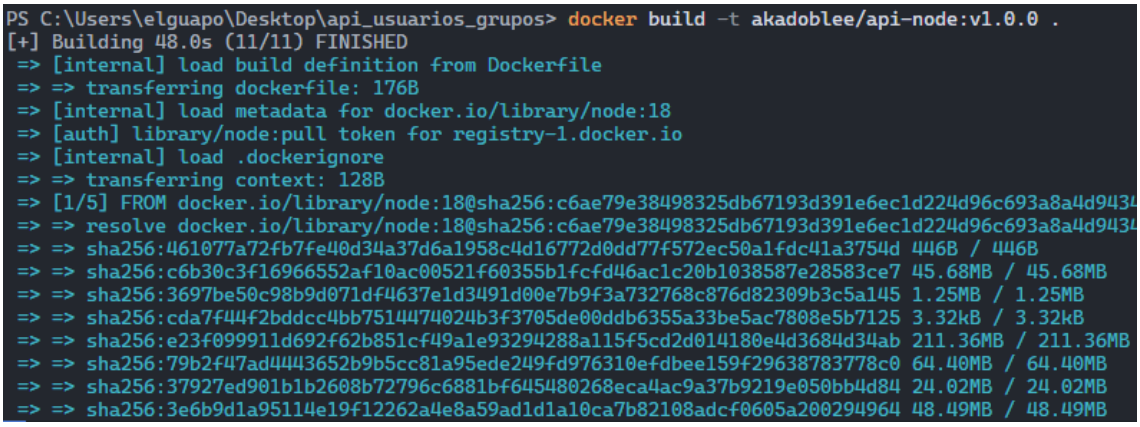


En este caso se ignoran los siguientes archivos o directorios:

- **node_modules/**: Directorio de dependencias de Node.js.
- **DOCUMENTACIÓN/**: Directorio en el que se encuentra la documentación.
- **.git/**: Directorio que almacena la información de control de versiones de Git.
- **.gitignore**: Archivo que indica los archivos y directorios cuyo contenido no se subirá al repositorio remoto.
- ***.txt y *.md**: Archivos de texto y markdown.
- **docker-compose.yaml**: Archivo para la creación posterior de un stack utilizando la imagen que se subirá a Docker Hub.

Una vez preparados los dos archivos, se debe abrir una terminal en el directorio del proyecto y ejecutar el siguiente comando para realizar una `build` de la imagen:

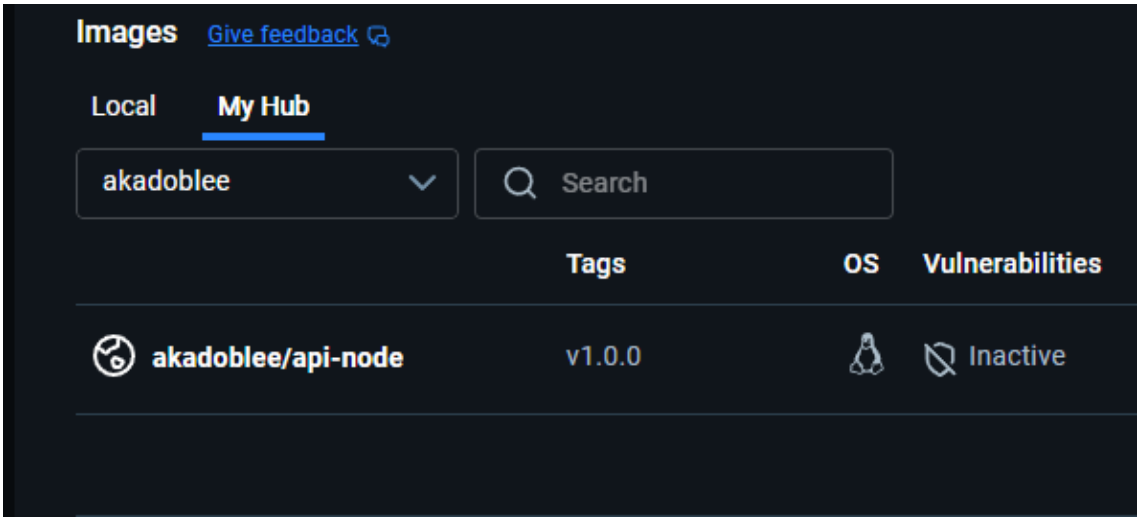
```
docker build -t <usuario>/<imagen>:<tag><ruta_archivo_dockerfile>
```



Una vez creada la imagen, se debe subir a Docker Hub utilizando el siguiente comando:

```
docker push <usuario>/<imagen>:<tag>
```

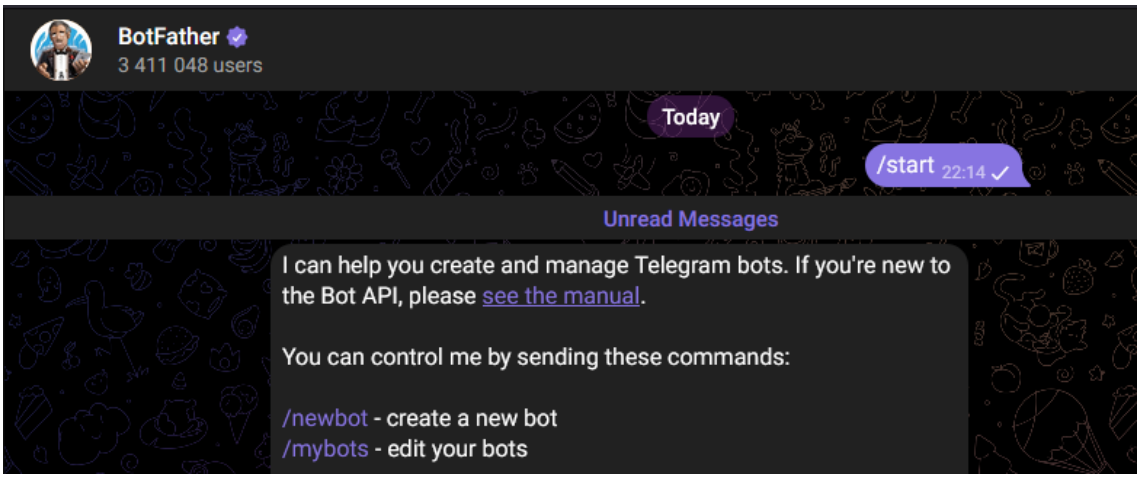
Para verificar que la imagen se haya subido correctamente a Docker Hub, se debe ejecutar la aplicación Docker Desktop con la sesión iniciada y en el panel de la izquierda se debe seleccionar la pestaña llamada **Images**. Una vez en ella se debe seleccionar la pestaña llamada **My Hub** y en ella ya debería de aparecer la imagen subida.



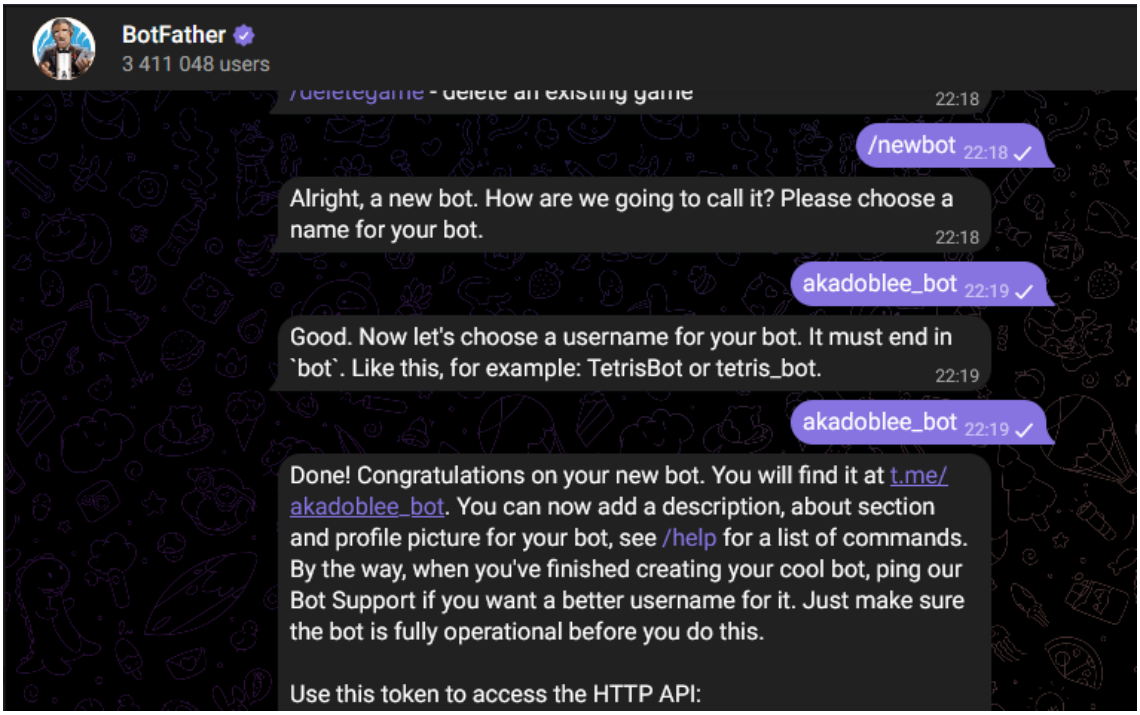
5. CI/CD CON GITHUB ACTIONS

En esta sección se explicarán los pasos para crear un workflow con GitHub Actions que permita subir la imagen a Docker Hub cuando se realice un push a la rama main del repositorio remoto y se enviará un mensaje de aviso por Telegram.

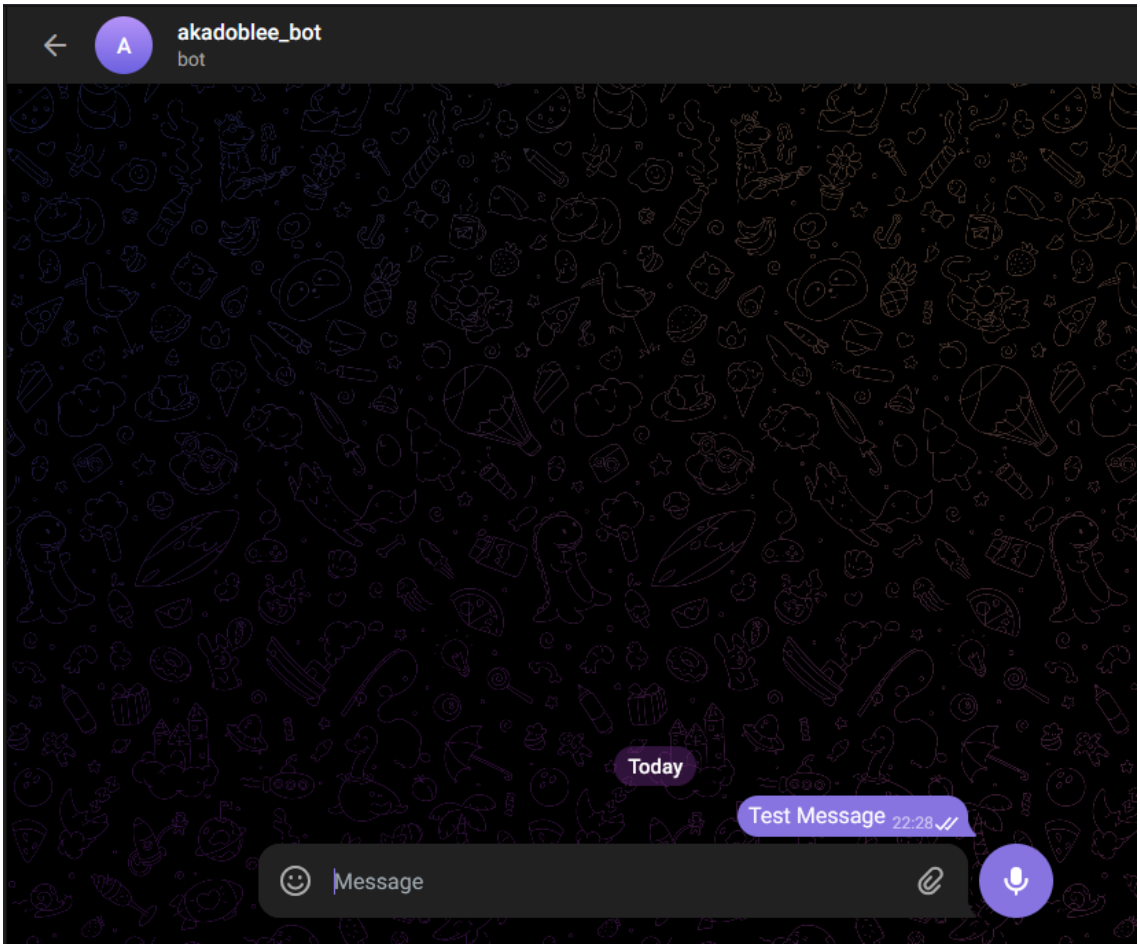
El primer paso será crear el bot de Telegram en el que se verán reflejados los avisos de nuevas subidas de la imagen en Docker Hub y para ello el primer paso será iniciar un nuevo chat con el bot **@BotFather** con el comando `/start`.



Una vez iniciada la conversación, se debe utilizar el comando `/newbot` para comenzar el proceso de creación del bot. Una vez ejecutado el comando anterior, se deberá indicar el nombre y el usuario del bot siguiendo las instrucciones recibidas. Posteriormente se podrá acceder al chat con el bot recién creado con el enlace proporcionado y el token que se indica en el mensaje se debe de guardar para su posterior uso.

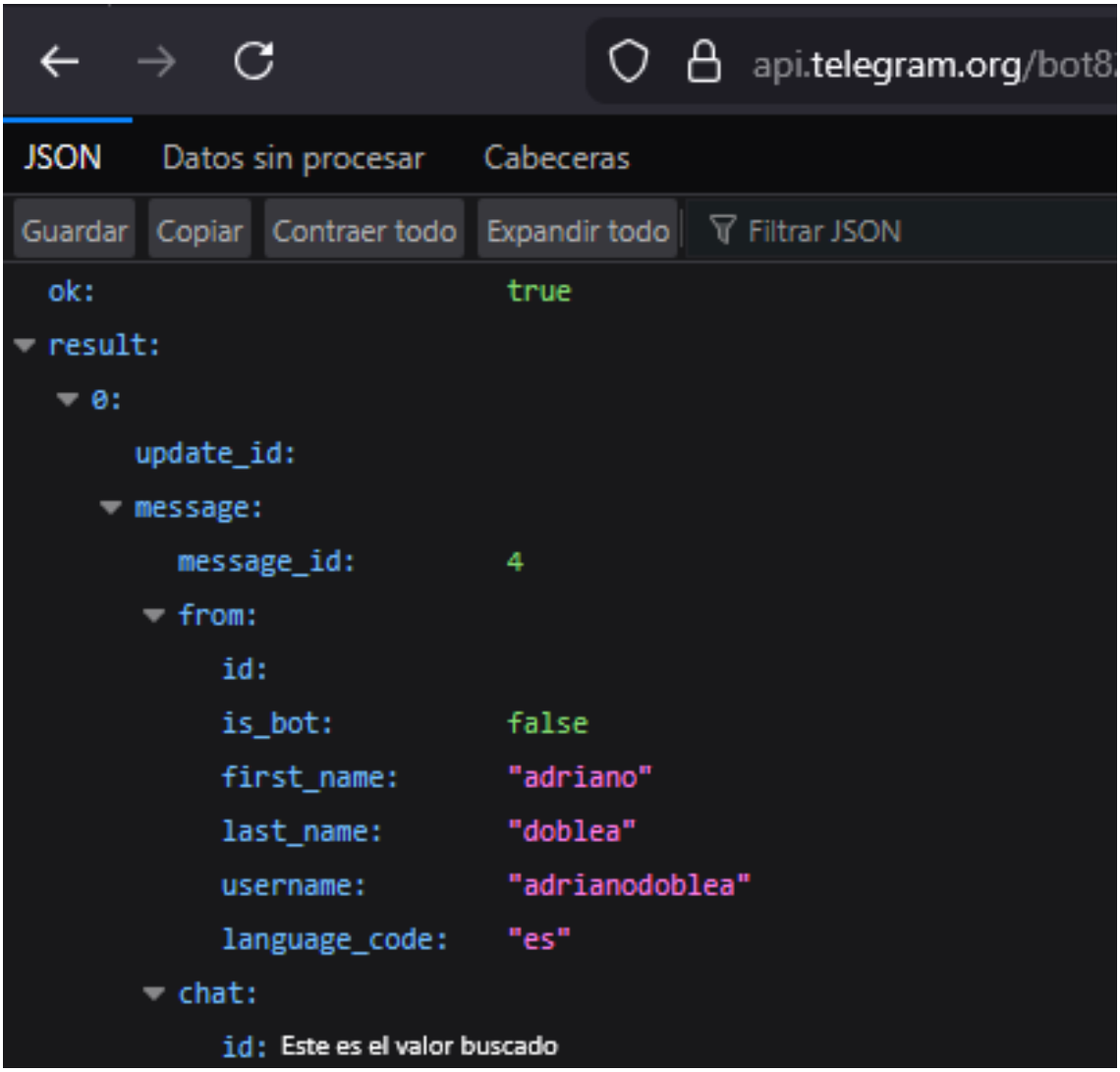


El siguiente paso es obtener el **CHAT_ID** de la conversación con el bot creado en el paso anterior. Para ello lo primero que se debe hacer es acceder al chat mediante el enlace proporcionado en la conversación con @BotFather y escribir cualquier mensaje.

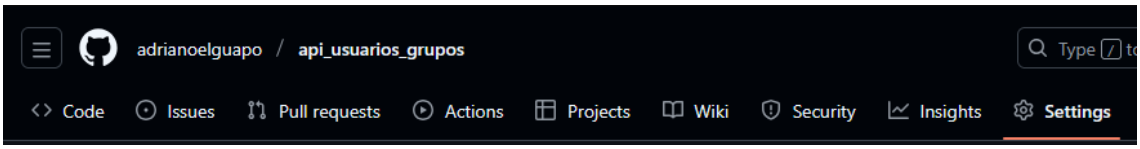


Una vez escrito el mensaje, se enviará una solicitud a la API de Telegram para obtener el **CHAT_ID**. La URL a utilizar es la siguiente (se debe sustituir la etiqueta `token_bot` por el token proporcionado por @BotFather). El valor buscado en la respuesta es el `id` del objeto `chat` que forma parte del objeto `message`:

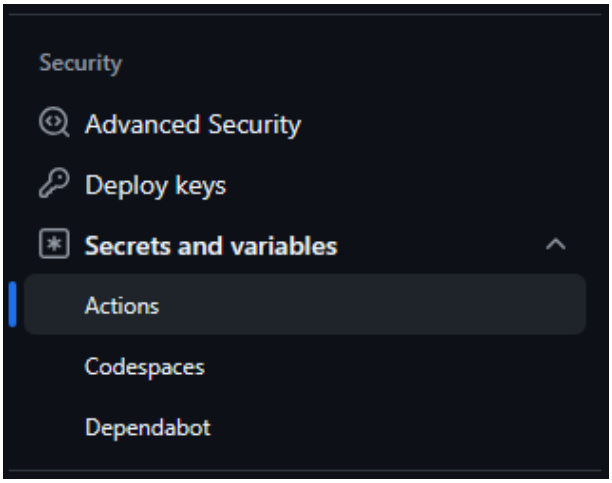
`https://api.telegram.org/bot<token_bot>/getUpdates`



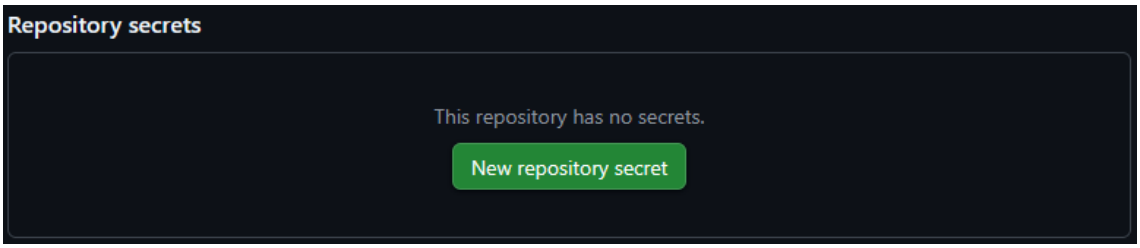
Una vez obtenido el **CHAT_ID**, se debe acceder al repositorio del proyecto en GitHub y dirigirse a la pestaña llamada **Settings**.



Una vez en la pestaña de configuración del repositorio, se debe pulsar sobre la opción llamada **Secrets and variables** y en las opciones que se muestran se debe pulsar sobre la opción llamada **Actions**.

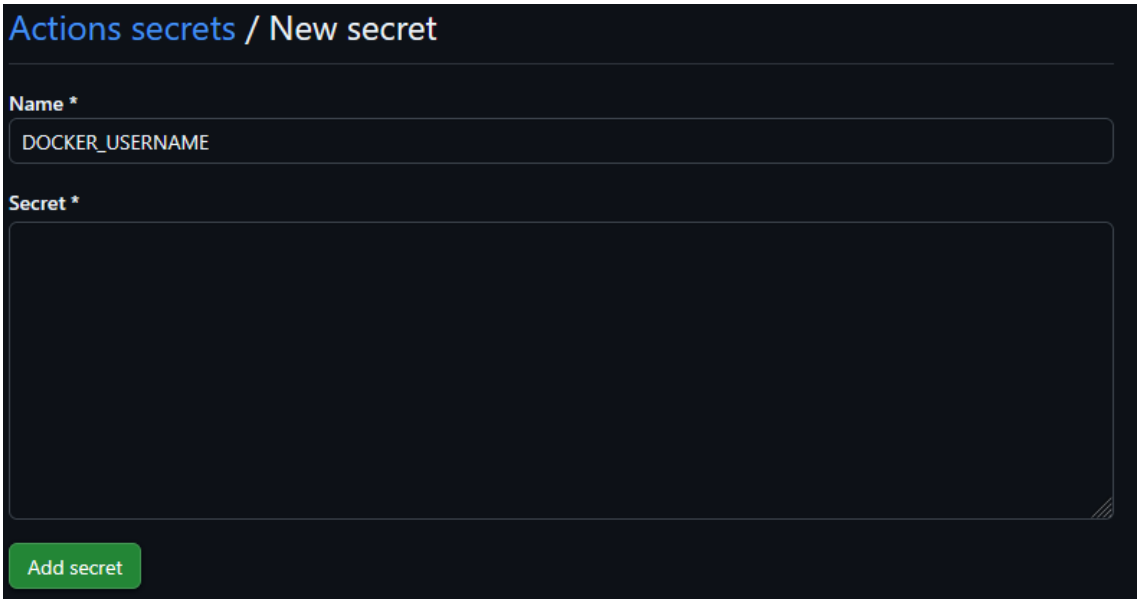


Una vez realizado el paso anterior, se debe pulsar sobre el botón verde de **New repository secret**.



A continuación se debe de proporcionar el nombre de la variable de entorno y su valor. De esta manera se crearán las siguiente variables de entorno en el repositorio:

- **DOCKER_USERNAME**: Nombre de usuario de la cuenta de Docker Hub.
- **DOCKER_PASSWORD**: Contraseña de la cuenta de Docker Hub.
- **TELEGRAM_CHAT_ID**: ID del chat con el bot de Telegram.
- **TELEGRAM_TOKEN**: Token del bot de Telegram.



Una vez añadidas las variables de entorno en el repositorio, se debe de crear una archivo con extensión `.yml` en la siguiente ruta:

`.github/workflows/<nombre_archivo>.yml`

```
C:\Users\elguapo\Desktop\api_usuarios_grupos>mkdir .github

C:\Users\elguapo\Desktop\api_usuarios_grupos>cd .github

C:\Users\elguapo\Desktop\api_usuarios_grupos\.github>mkdir workflows
```

El archivo creado para el workflow incluye los siguientes pasos:

- **Check**: Comprueba si se ha realizado un push a la rama main o master y si se ha creado una nueva etiqueta.
- **Login**: Inicia sesión en Docker Hub.
- **Set Tag**: Establece la tag de la imagen.
- **Build & Push**: Construye y sube la imagen a Docker Hub.
- **Telegram Message**: Envía un mensaje al chat del bot de Telegram.