



# Despliegue Automático de Aplicación Web con Jenkins y Netlify

TRABAJO FIN DE GRADO - TFG

Miguel Ángel Barea Ortega | ASIX

## Índice

Introducción de la idea del proyecto .....	2
Palabras clave del proyecto.....	2
Objeto / finalidad.....	3
Metodología .....	4
Contenido (paso a paso) .....	5
Conclusiones .....	29
Glosario .....	29
Referencias bibliográficas utilizadas .....	29
Anexos .....	29
Presentación.....	29

## Introducción de la idea del proyecto

Este proyecto tiene como objetivo implementar un sistema de despliegue automático de una aplicación web sencilla usando herramientas modernas de integración continua (CI) y despliegue continuo (CD). Para ello, utilizaré Jenkins como servidor de automatización y Netlify como plataformas de despliegue en la nube.

## Palabras clave del proyecto

### CI/CD (Integración Continua / Entrega o Despliegue Continuo)

Es un conjunto de prácticas de desarrollo que permiten automatizar y mejorar el proceso de integración de código (CI) y su entrega o despliegue (CD).

- **CI (Continuous Integration):** Integrar los cambios de código frecuentemente, asegurando que funcionen correctamente mediante pruebas automáticas.
- **CD (Continuous Delivery o Deployment):** Automatizar la entrega o el despliegue del software a entornos de pruebas o producción.

---

### Jenkins

Herramienta de automatización de código abierto utilizada para implementar procesos de CI/CD. Permite ejecutar tareas como compilaciones, pruebas y despliegues automáticamente cada vez que se actualiza el código.



---

### GitHub

Plataforma de alojamiento de código que utiliza Git como sistema de control de versiones. Permite colaborar, almacenar y gestionar versiones del código fuente, además de integrarse fácilmente con herramientas de CI/CD.



---

### Netlify

Plataforma para desplegar sitios web y aplicaciones web estáticas o JAMstack. Automatiza procesos como el build, testeo y despliegue, integrándose con repositorios como GitHub.



## Automatización

Proceso de ejecutar tareas sin intervención manual, utilizando scripts o herramientas como Jenkins, Docker o GitHub Actions. Permite ahorrar tiempo y reducir errores humanos.

---

## Despliegue web

Proceso de mover una aplicación o sitio web desde un entorno de desarrollo local hasta un servidor o servicio donde los usuarios puedan acceder a ella a través de Internet.

---

## Docker

Plataforma que permite crear, ejecutar y administrar contenedores. Un contenedor es una unidad de software que incluye todo lo necesario para ejecutar una aplicación, asegurando que funcione igual en cualquier entorno.



## Pipeline

Secuencia automatizada de pasos que sigue una aplicación desde que se escribe el código hasta que se despliega. En un pipeline CI/CD, estos pasos pueden incluir compilación, pruebas, empaquetado y despliegue.

---

## Cloudflare (Cloudflare Tunnel)

Cloudflare Tunnel permite establecer una conexión segura entre un servicio local (como un servidor Jenkins en desarrollo) y una URL accesible públicamente en Internet, sin necesidad de abrir puertos en el router ni modificar configuraciones de red. Esta solución expone el servicio local a través de un subdominio personalizado, protegido por la red de Cloudflare, lo que facilita la recepción de peticiones externas (como webhooks) en entornos de desarrollo o pruebas, manteniendo un alto nivel de seguridad y disponibilidad.



## Objeto / finalidad

- Automatizar el proceso de despliegue de una aplicación web.
- Configurar Jenkins para detectar cambios en un repositorio GitHub.
- Desplegar automáticamente la aplicación en Netlify.
- Aprender el flujo completo de CI/CD en un entorno real.

## Metodología

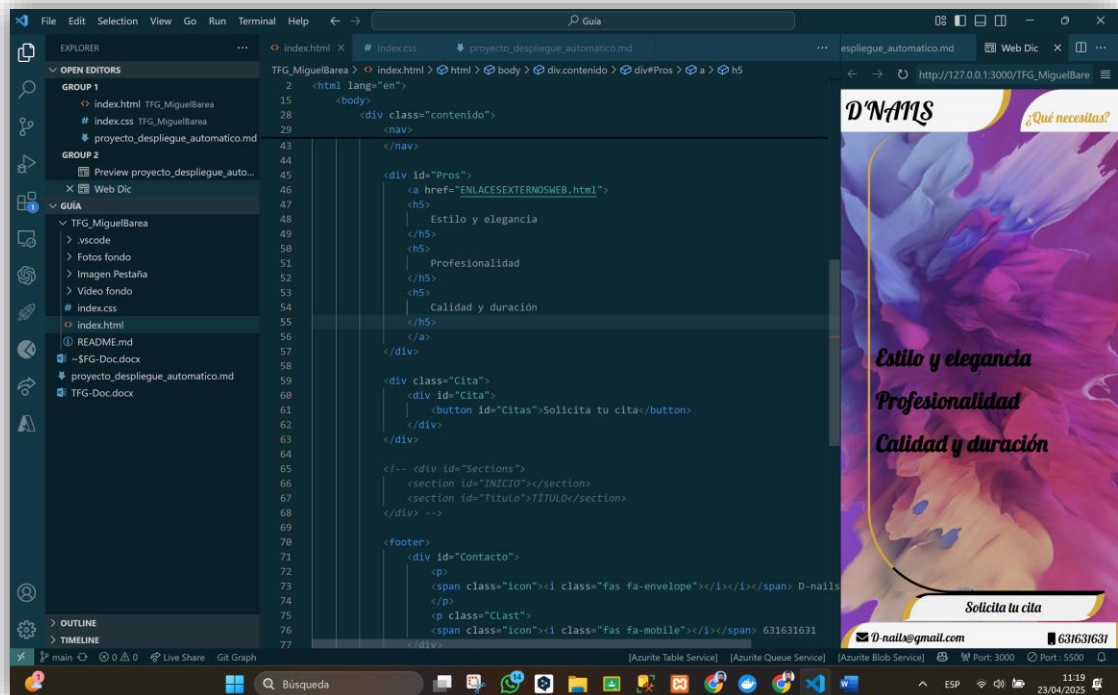
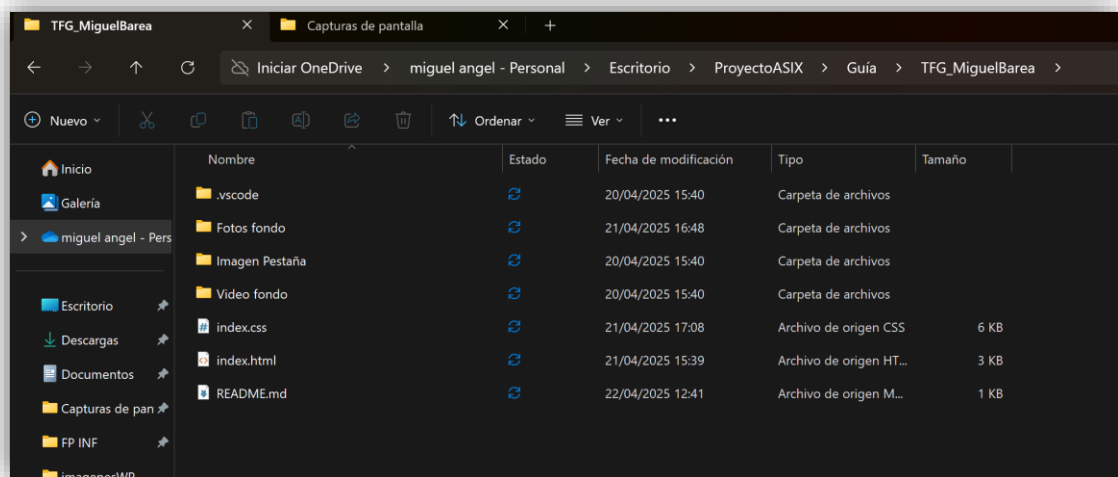
- Origen de la idea: Durante mis prácticas en el departamento de DevOps, estuve expuesto a flujos de integración y despliegue continuo (CI/CD) en proyectos reales. Dado que estoy desarrollando una web para una futura empresa familiar, decidí aprovechar ese aprendizaje para montar un sistema de despliegue profesional y automatizado, que además me sirviera como trabajo de fin de ciclo.
- Hipótesis: Es posible automatizar todo el ciclo de despliegue de una aplicación estática utilizando herramientas gratuitas y modernas como Jenkins y Netlify.
- Herramientas utilizadas:
  - GitHub (repositorio de código)
  - Jenkins (automatización de procesos)
  - Docker (contenedorización de Jenkins y Cloudflare Tunnel)
  - Netlify (hosting y despliegue continuo de la web)
  - Cloudflare Tunnel (exposición de Jenkins local de forma segura)
  - Cloudflare (gestión de dominio y DNS)
  - Dominio personalizado (adquirido e integrado para una URL estable y segura)
  - Ngrok (utilizado temporalmente durante las primeras pruebas de webhook)
- Estrategia: El desarrollo se organizó por etapas a lo largo de 2 meses y medio durante las prácticas. Se adoptó un enfoque iterativo y práctico, validando cada paso con pruebas reales. El proceso fue el siguiente:
  1. Desarrollo inicial de la web estática y despliegue manual en Netlify.
  2. Subida del proyecto a GitHub para gestión de versiones.
  3. Instalación y configuración de Jenkins en un contenedor Docker.
  4. Generación y gestión de credenciales seguras para Netlify y GitHub.
  5. Creación del Jenkinsfile para definir el pipeline de CI/CD.
  6. Implementación de la automatización del despliegue con Netlify CLI.
  7. Configuración inicial del webhook con Ngrok (problemas con URL temporal).
  8. Migración a Cloudflare Tunnel para tener una URL permanente y segura.
  9. Compra y configuración de un dominio propio gestionado con Cloudflare.
  10. Configuración final del webhook usando jenkins.d-nails.es y validación completa del flujo.

Todos los pasos fueron documentados con capturas, pruebas de funcionamiento y resolución de problemas como la exposición del entorno local y la autenticación segura entre servicios.

# Contenido (paso a paso)

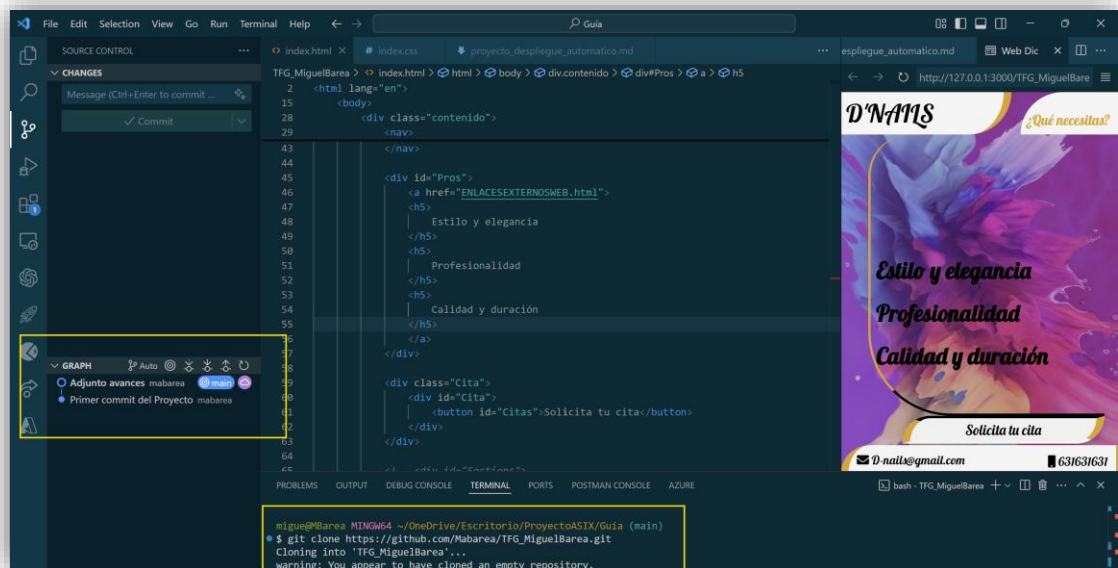
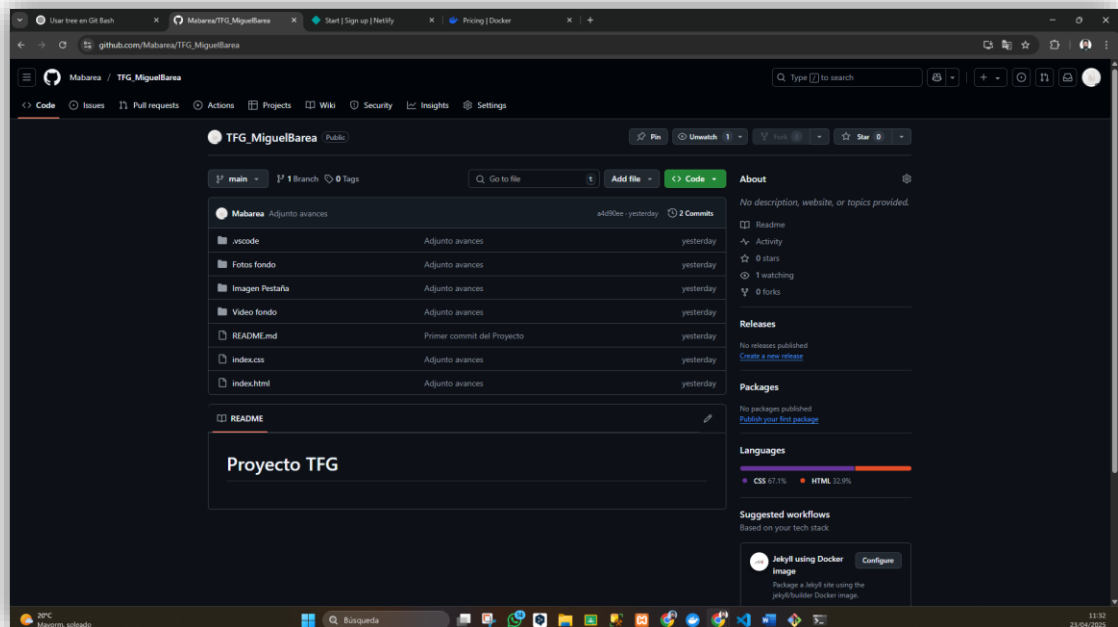
## PASO 1: Crear la aplicación web

Crear una carpeta e incluir un archivo index.html con contenido básico HTML. Aquí se puede ver la estructura de directorios y Web estática que he hecho y usado para incluir en el proyecto:



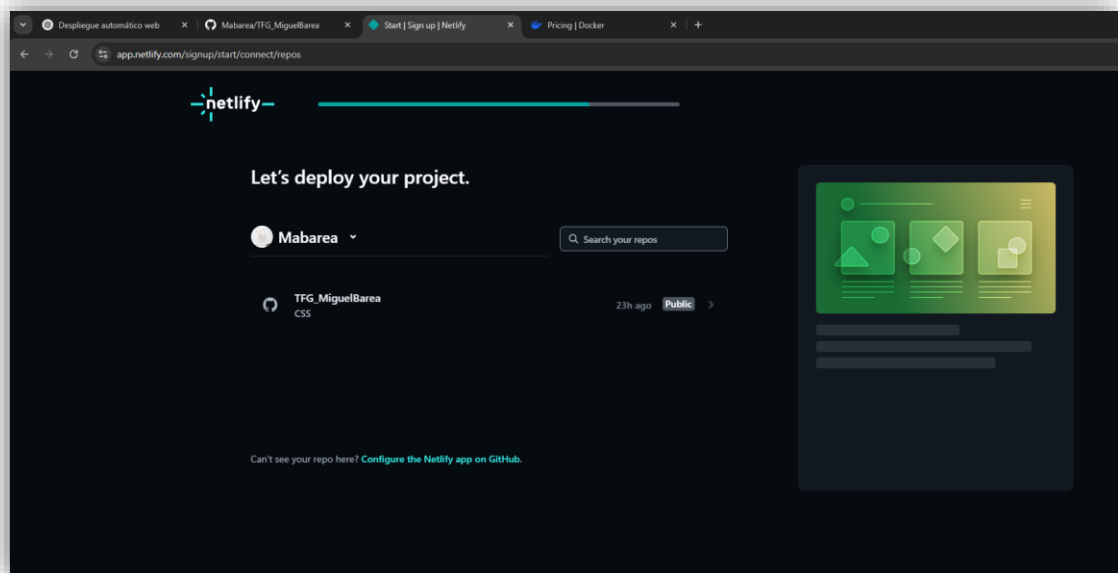
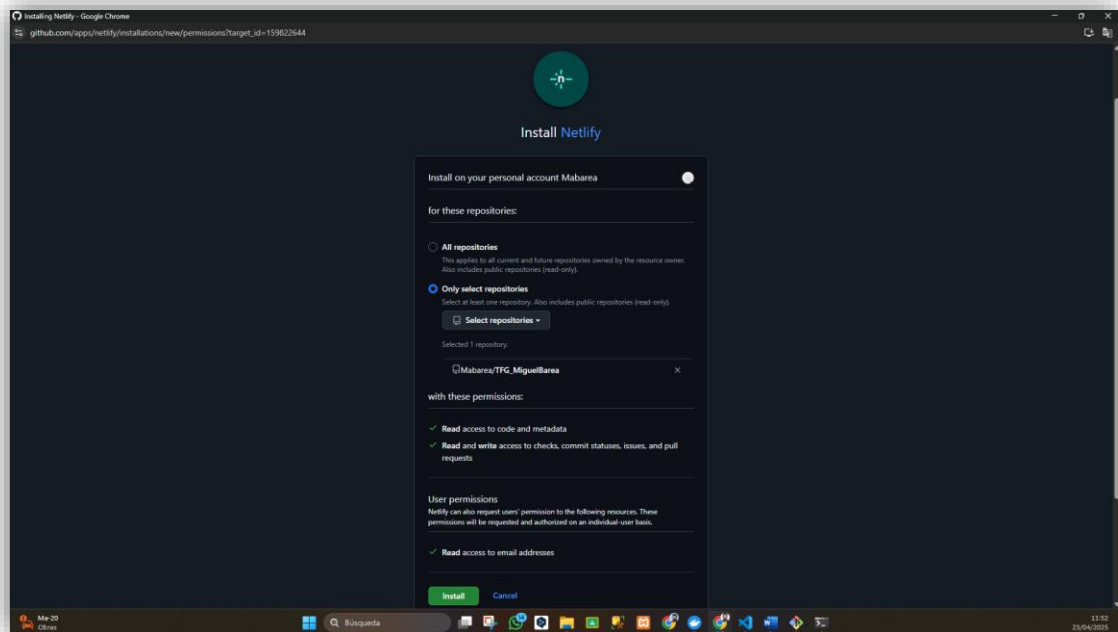
## PASO 2: Subir el proyecto a GitHub

Crear un repositorio en GitHub y clonar en Visual Studio Code:



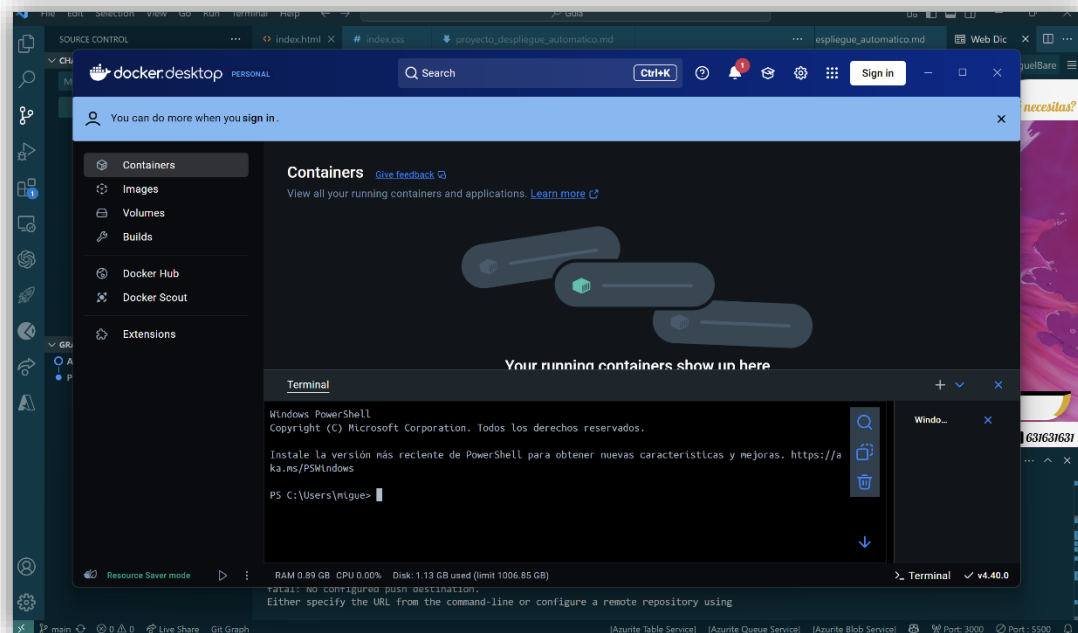
### PASO 3: Crear cuenta en Netlify

Registrar una cuenta, importar el proyecto desde GitHub y desplegarlo automáticamente. Guardar la URL pública del sitio.





## PASO 4: Instalar y configurar Jenkins con Docker

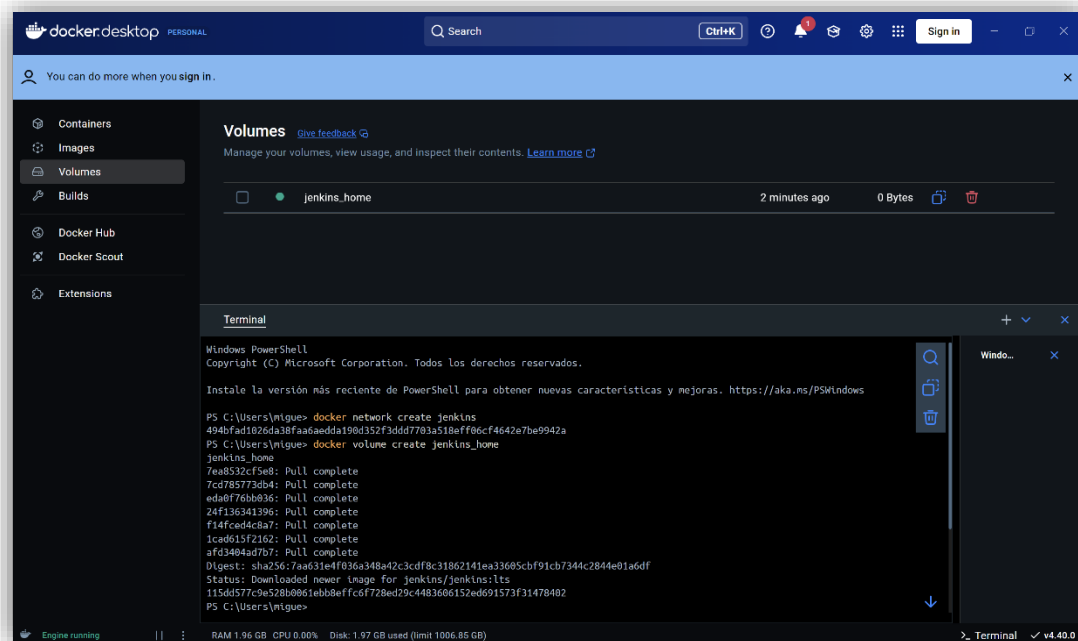


1. Instalar Docker si no está instalado.
2. Crear una red Docker:

```
docker network create jenkins
```

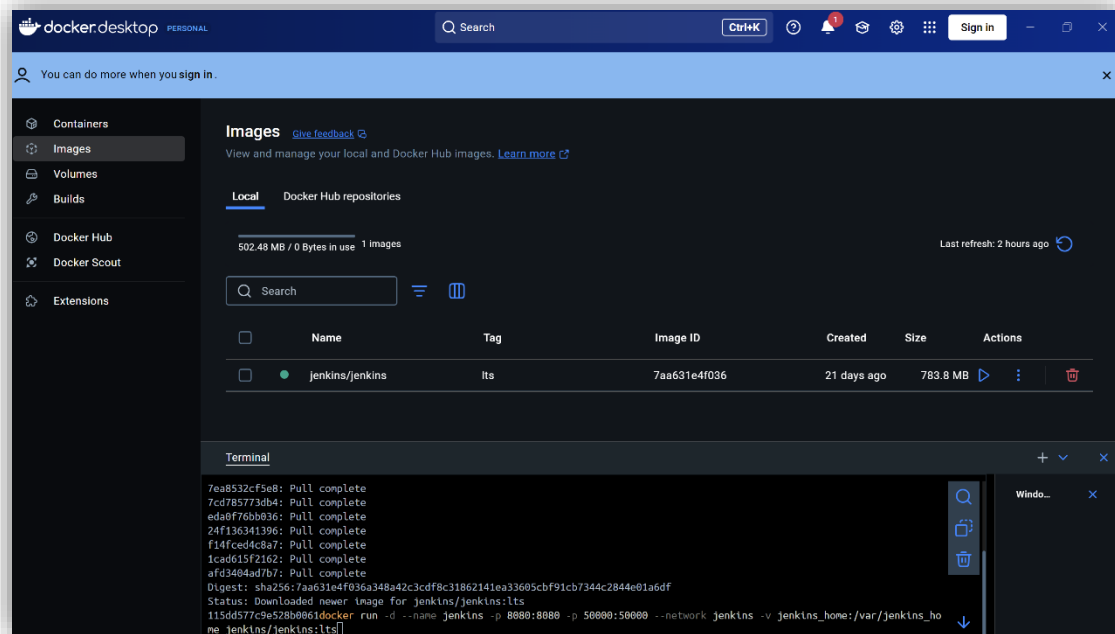
3. Crear un volumen para Jenkins:

```
docker volume create jenkins_home
```

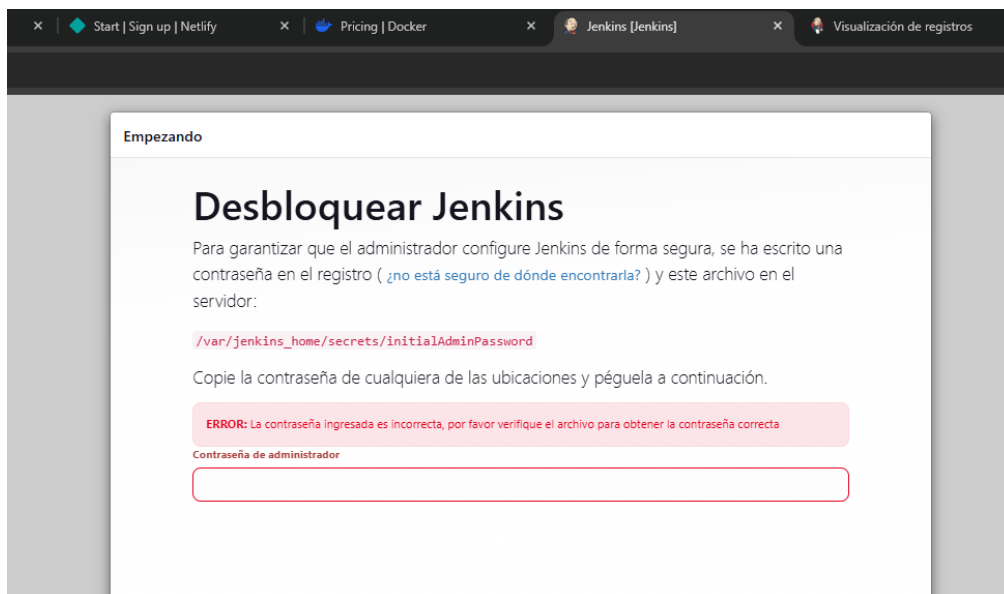


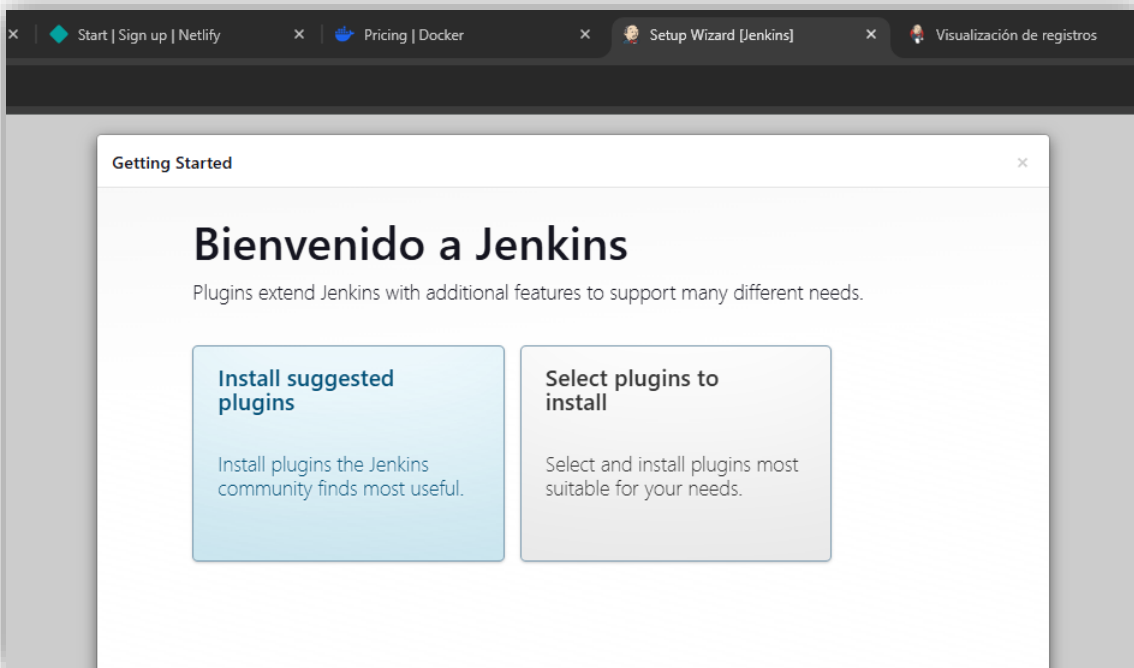
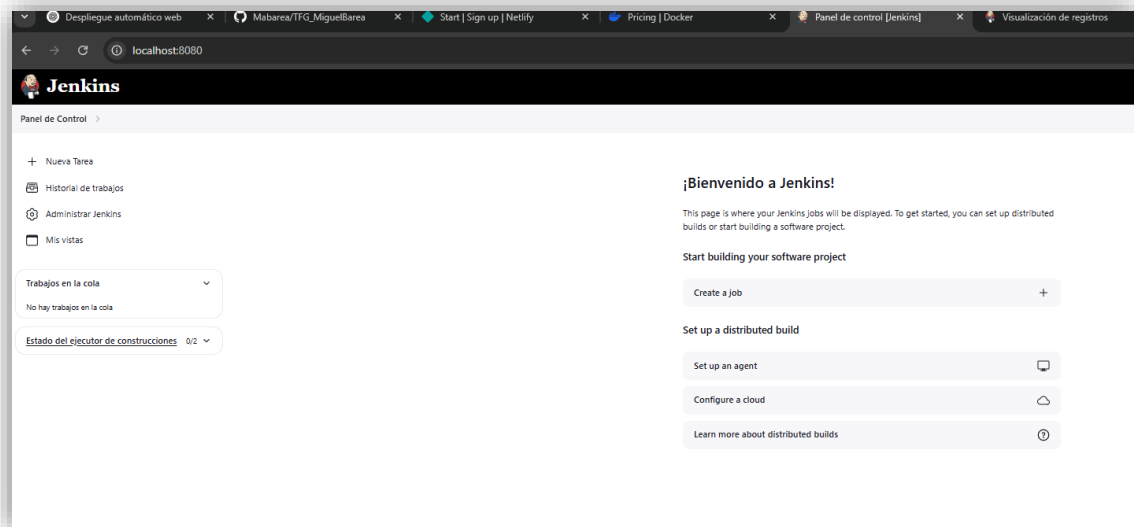
#### 4. Ejecutar Jenkins:

```
docker run -d --name jenkins -p 8080:8080 -p 50000:50000 --network jenkins -v jenkins_home:/var/jenkins_home jenkins/jenkins:latest
```



#### 5. Acceder a Jenkins desde el navegador en ***http://localhost:8080*** y seguir las instrucciones para completar la configuración inicial.

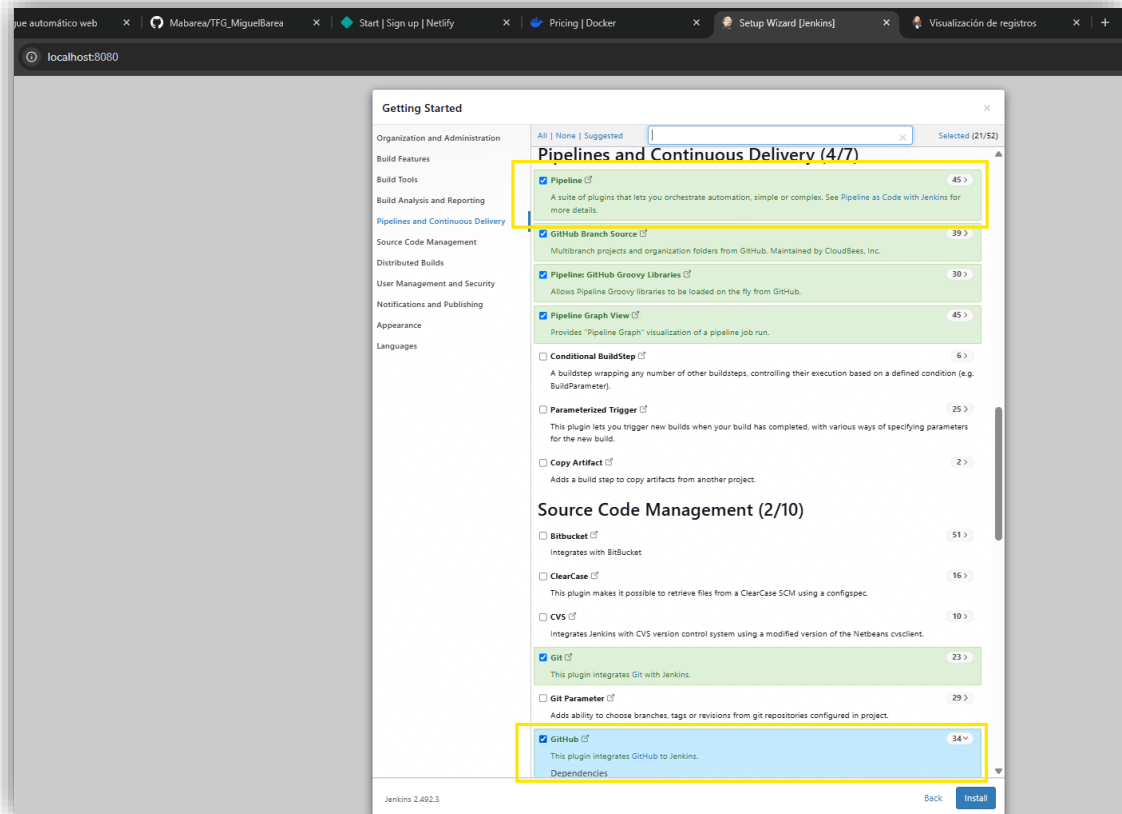




## PASO 5: Instalar plugins necesarios

Instalar los siguientes plugins desde "Manage Plugins":

- GitHub Integration
- Pipeline

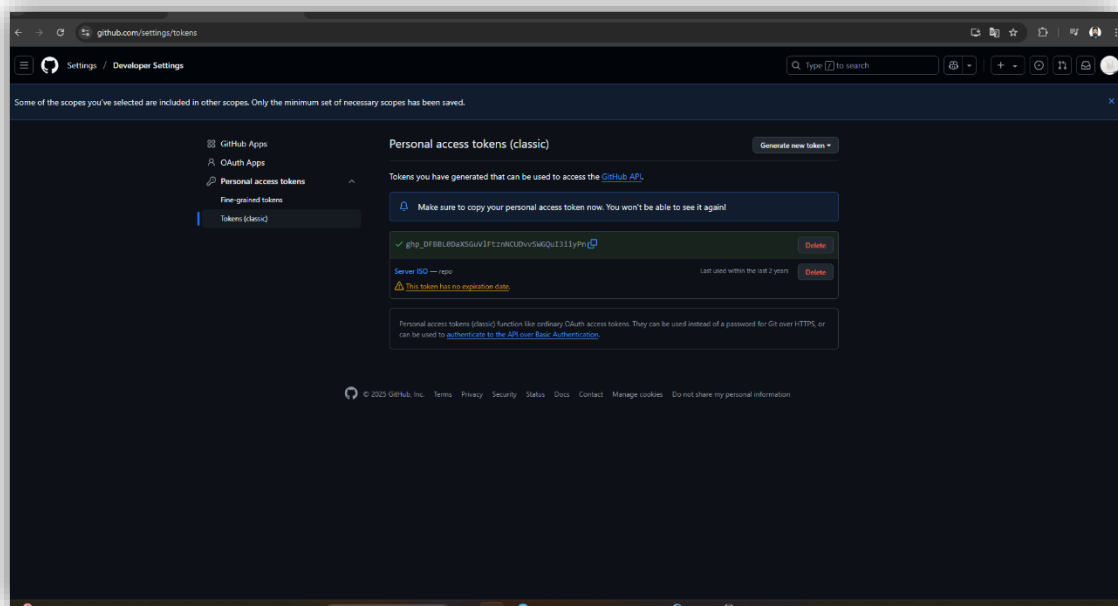


## PASO 6: Configurar credenciales de GitHub y Netlify

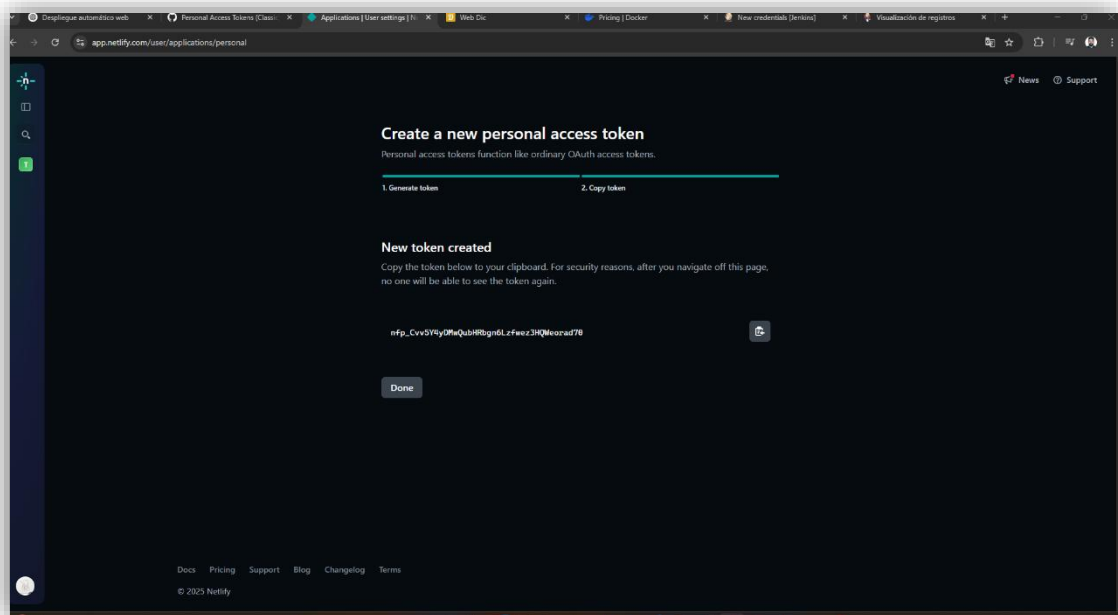
Este paso permite que Jenkins pueda autenticar y conectarse con los servicios externos de GitHub y Netlify para realizar acciones como clonar repositorios o desplegar sitios.

### 1. Obtener credenciales

- **GitHub:** Crear un token de acceso personal (PAT) desde <https://github.com/settings/tokens>, con acceso de lectura al repositorio.



- **Netlify:** Crear un token desde <https://app.netlify.com/user/applications>.

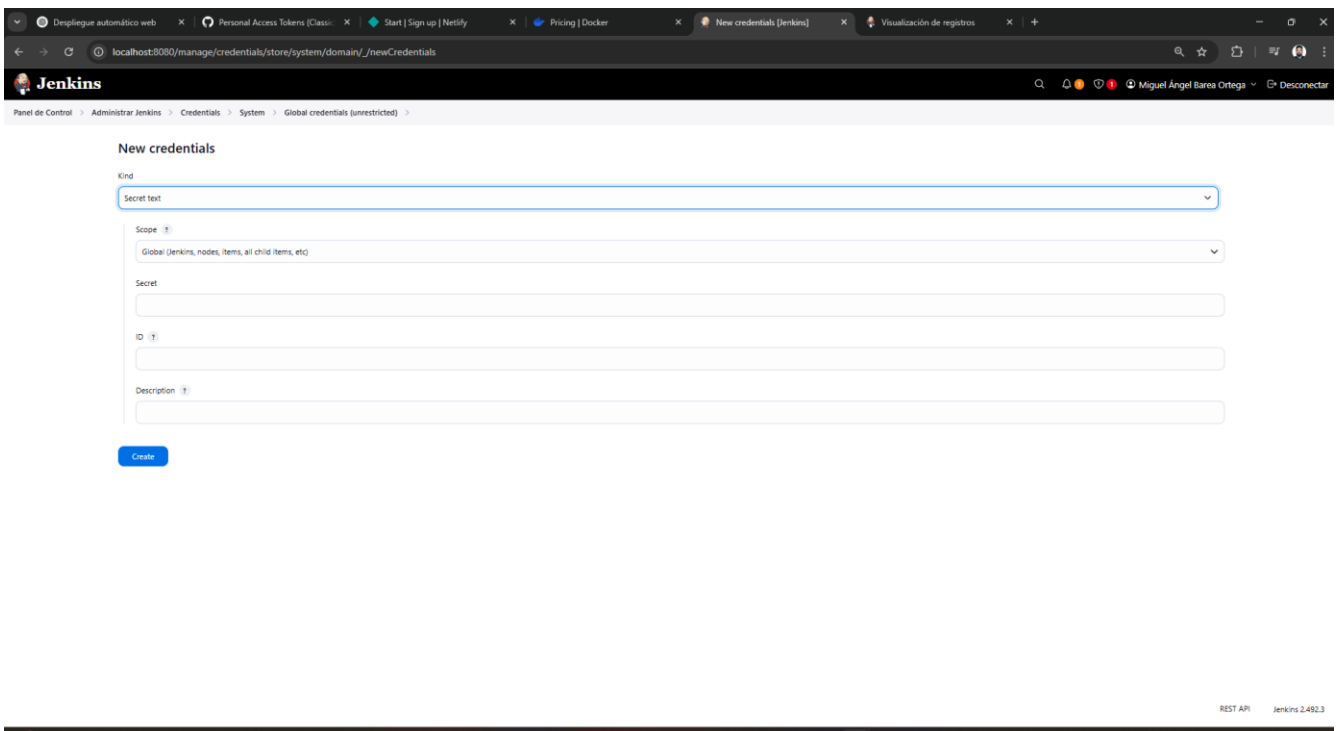


## 2. Añadir a Jenkins

Desde "Manage Jenkins → Credentials → Global → Add Credentials":

- **GitHub:**
  - Kind: Secret text
  - Secret: [token GitHub]
  - ID: github-token
  - Description: Token GitHub para Jenkins
- **Netlify:**
  - Kind: Secret text
  - Secret: [token Netlify]
  - ID: netlify-token
  - Description: Token Netlify para despliegue

En ambos casos encontraremos esto:



The screenshot shows the Jenkins web interface at the URL `localhost:8080/manage/credentials/store/system/domain/_/newCredentials`. The page title is "New credentials". The form contains the following fields:

- Kind:** A dropdown menu with "Secret text" selected.
- Scope:** A dropdown menu with "Global (Jenkins, nodes, items, all child items, etc)" selected.
- Secret:** A text input field.
- ID:** A text input field.
- Description:** A text input field.

At the bottom left of the form is a blue "Create" button. The top of the browser window shows several tabs, including "Despliegue automático web", "Personal Access Tokens (Classi...", "Start | Sign up | Netlify", "Pricing | Docker", "New credentials [Jenkins]", and "Visualización de registros". The bottom right corner of the page displays "REST API" and "Jenkins 2.482.3".

### 3. Uso en Jenkinsfile

Dentro del pipeline, se puede usar:

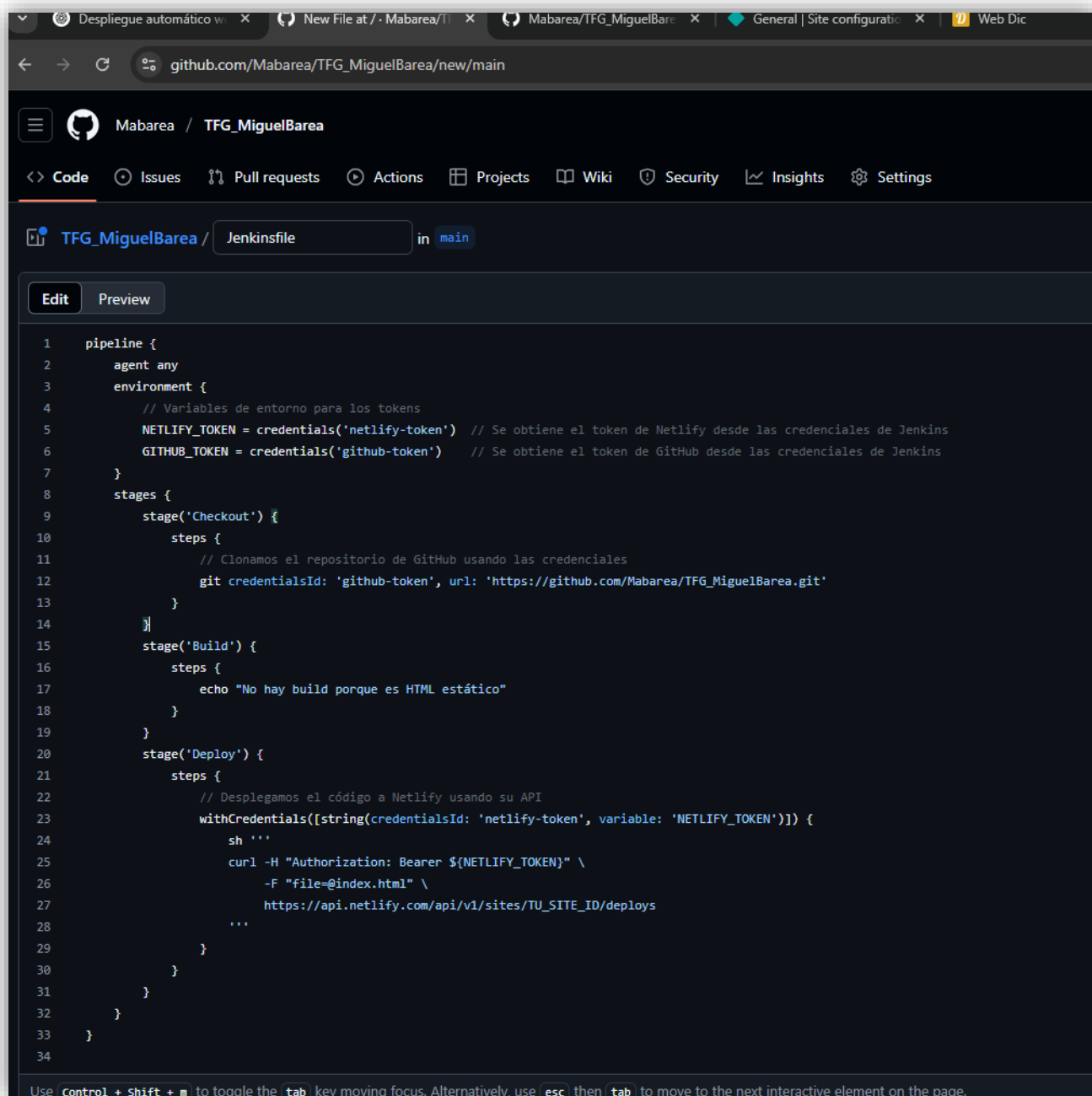
```
withCredentials([string(credentialsId: 'netlify-token',  
variable: 'NETLIFY_TOKEN')]) {  
    sh 'echo "Usando token: $NETLIFY_TOKEN"'  
}
```

## PASO 7: Crear archivo Jenkinsfile

Crear un archivo Jenkinsfile en el repositorio con la siguiente estructura:

```
pipeline {
    agent any
    environment {
        // Variables de entorno para los tokens
        NETLIFY_TOKEN = credentials('netlify-token') // Se obtiene el token de
        Netlify desde las credenciales de Jenkins
        GITHUB_TOKEN = credentials('github-token') // Se obtiene el token de
        GitHub desde las credenciales de Jenkins
    }
    stages {
        stage('Checkout') {
            steps {
                // Clonamos el repositorio de GitHub usando las credenciales
                git credentialsId: 'github-token', url:
                'https://github.com/Mabarea/TFG_MiguelBarea.git'
            }
        }
        stage('Build') {
            steps {
                echo "No hay build porque es HTML estático"
            }
        }
        stage('Deploy') {
            steps {
                // Desplegamos el código a Netlify usando su API
                withCredentials([string(credentialsId: 'netlify-token', variable:
                'NETLIFY_TOKEN')]) {
                    sh '''
                        curl -H "Authorization: Bearer ${NETLIFY_TOKEN}" \
                            -F "file=@index.html" \
                                https://api.netlify.com/api/v1/sites/TU_SITE_ID/deloys
                    '''
                }
            }
        }
    }
}
```





The screenshot shows a web browser with multiple tabs. The active tab is 'github.com/Mabarea/TFG\_MiguelBarea/new/main'. The page displays the GitHub interface for the repository 'Mabarea / TFG\_MiguelBarea'. The 'Code' tab is selected, and the 'Jenkinsfile' is chosen from the dropdown menu. The file is located in the 'main' branch. Below the file name, there are 'Edit' and 'Preview' buttons. The main content area shows the Jenkinsfile code, which is a pipeline script. The code includes stages for 'Checkout', 'Build', and 'Deploy'. The 'Checkout' stage clones the repository. The 'Build' stage echoes a message. The 'Deploy' stage uses 'withCredentials' to deploy the code to Netlify. The code is numbered from 1 to 34. At the bottom of the page, there is a footer with instructions: 'Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.'

```
1 pipeline {
2   agent any
3   environment {
4     // Variables de entorno para los tokens
5     NETLIFY_TOKEN = credentials('netlify-token') // Se obtiene el token de Netlify desde las credenciales de Jenkins
6     GITHUB_TOKEN = credentials('github-token') // Se obtiene el token de GitHub desde las credenciales de Jenkins
7   }
8   stages {
9     stage('Checkout') {
10      steps {
11        // Clonamos el repositorio de GitHub usando las credenciales
12        git credentialsId: 'github-token', url: 'https://github.com/Mabarea/TFG_MiguelBarea.git'
13      }
14    }
15    stage('Build') {
16      steps {
17        echo "No hay build porque es HTML estático"
18      }
19    }
20    stage('Deploy') {
21      steps {
22        // Desplegamos el código a Netlify usando su API
23        withCredentials([string(credentialsId: 'netlify-token', variable: 'NETLIFY_TOKEN')]) {
24          sh '''
25            curl -H "Authorization: Bearer ${NETLIFY_TOKEN}" \
26              -F "file=@index.html" \
27              https://api.netlify.com/api/v1/sites/TU_SITE_ID/deloys
28          '''
29        }
30      }
31    }
32  }
33 }
34
```

Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.

## PASO 8: Crear Pipeline en Jenkins

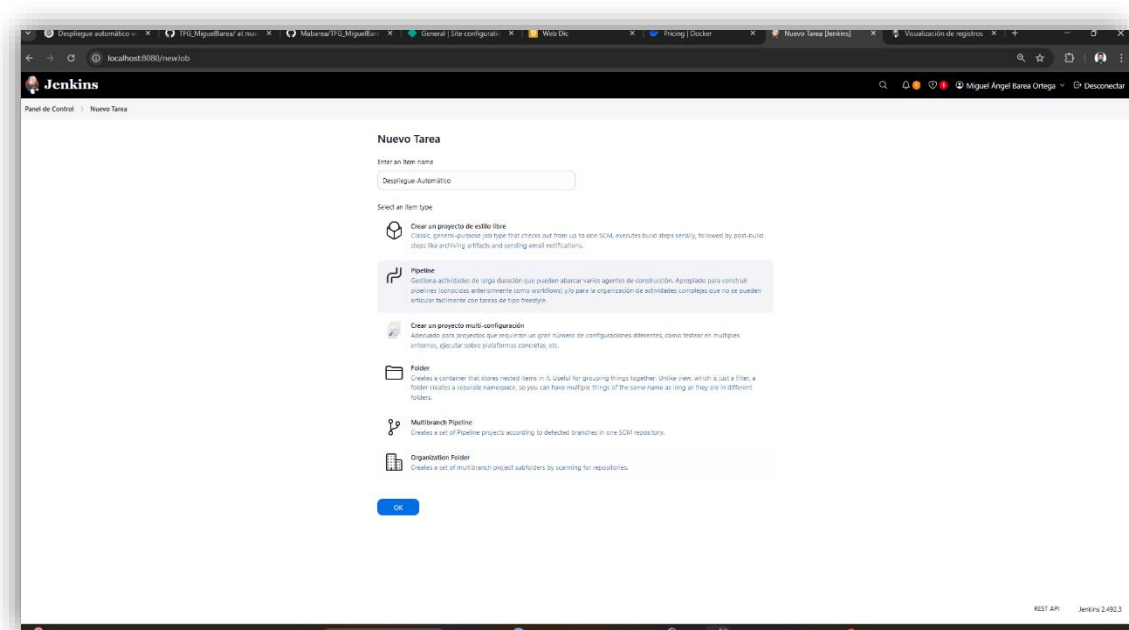
Crear un nuevo Pipeline en Jenkins que apunte al repositorio GitHub y utilice el Jenkinsfile para la configuración.

### 1. Acceder a Jenkins.

Entrar en la interfaz web de Jenkins en <http://localhost:8080>.

### 2. Crear un nuevo proyecto de tipo "Pipeline":

- Hacer clic en "Nuevo elemento" o "New Item" en la pantalla principal de Jenkins.
- Ingresar un nombre para el proyecto (por ejemplo, "Despliegue-Automático").
- Seleccionar "Pipeline" como tipo de proyecto.
- Hacer clic en "OK".

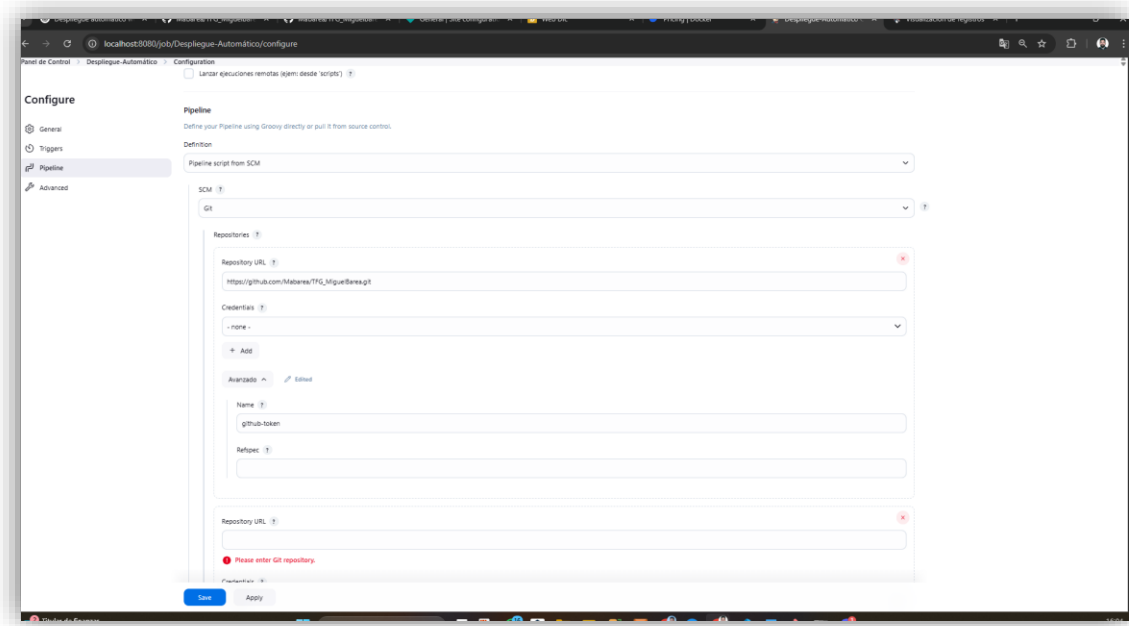


### 3. Configurar el Pipeline:

En la pantalla de configuración del nuevo Pipeline, se deben realizar las siguientes configuraciones:

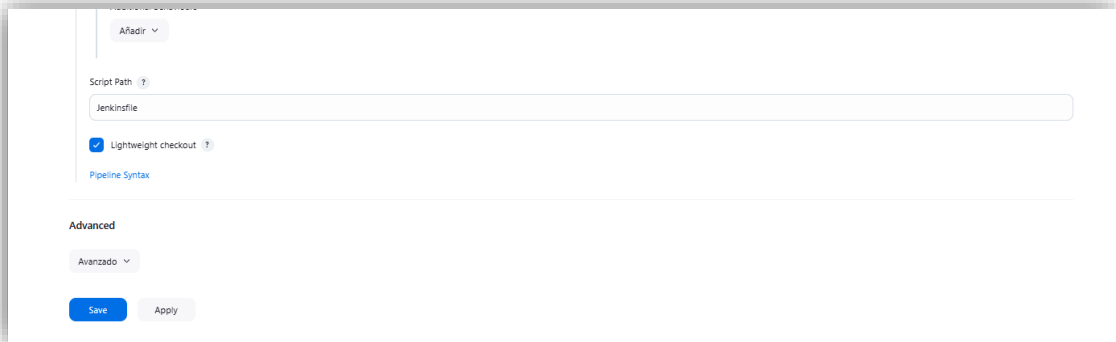
- Descripción (es opcional): Se puede incluir una breve descripción del pipeline.
- Fuente del código: Se debe conectar el repositorio de GitHub.
  - En el apartado "Pipeline", en la sección "Definition", se debe seleccionar Pipeline script from SCM (esto indica que Jenkins obtendrá el Jenkinsfile desde el repositorio).
  - En "SCM", se debe seleccionar Git.
  - En "Repository URL", se debe ingresar la URL del repositorio de GitHub, en mi caso:

[https://github.com/Mabarea/TFG\\_MiguelBarea.git](https://github.com/Mabarea/TFG_MiguelBarea.git)
  - En "Credentials", no seleccionar nada.
  - En "Avanzado", añadimos en "Name" el ID *github-token*, que incluye el token de GitHub previamente configurado.
  - En "Refspec" dejar en blanco si no se necesita configurar ramas específicas. Si solo se necesita una rama (por ejemplo, master), se puede usar el formato:
    - +refs/heads/master:refs/remotes/origin/master



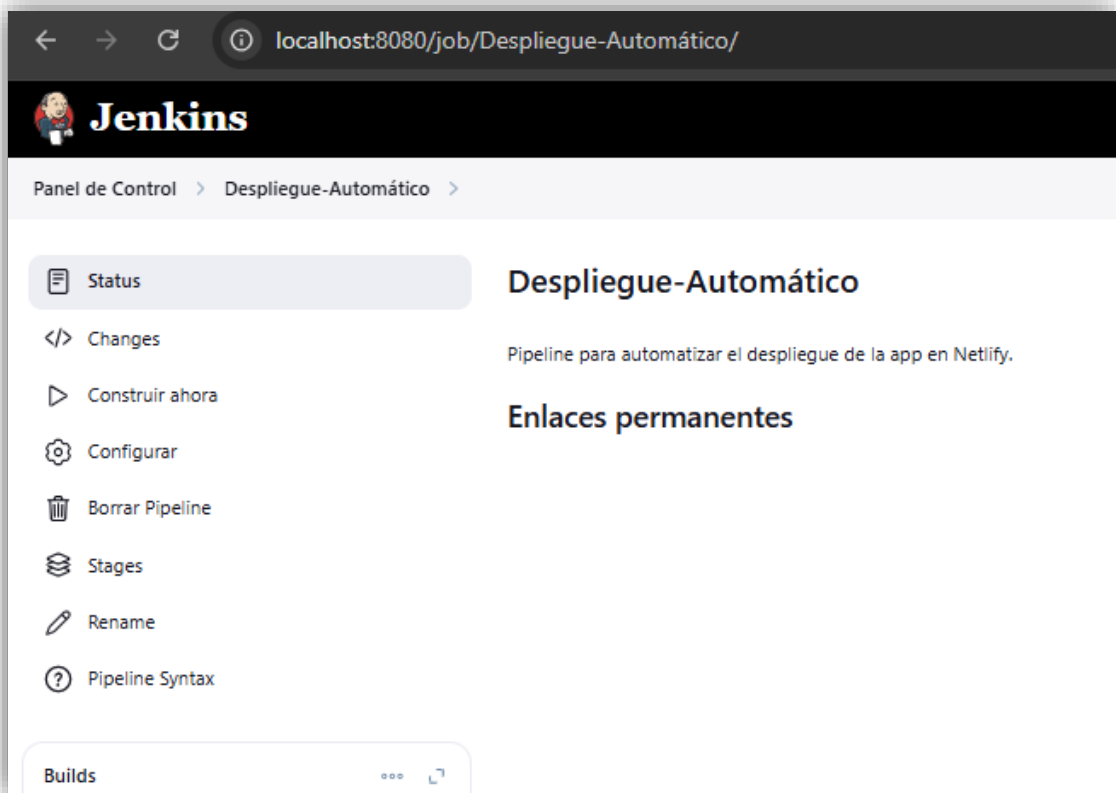
- **Configuración del Jenkinsfile:**

- En el campo "Script Path", se debe asegurar que esté configurado como Jenkinsfile, que es donde se encuentra el código del pipeline (el que se ha editado previamente).



#### 4. Guardar y ejecutar el Pipeline:

- Después de guardar los cambios, se podrá ejecutar el Pipeline de inmediato haciendo clic en "Construir ahora" o "Build Now". Jenkins debería clonar el repositorio, ejecutar el pipeline definido en el Jenkinsfile y desplegar la aplicación en Netlify.

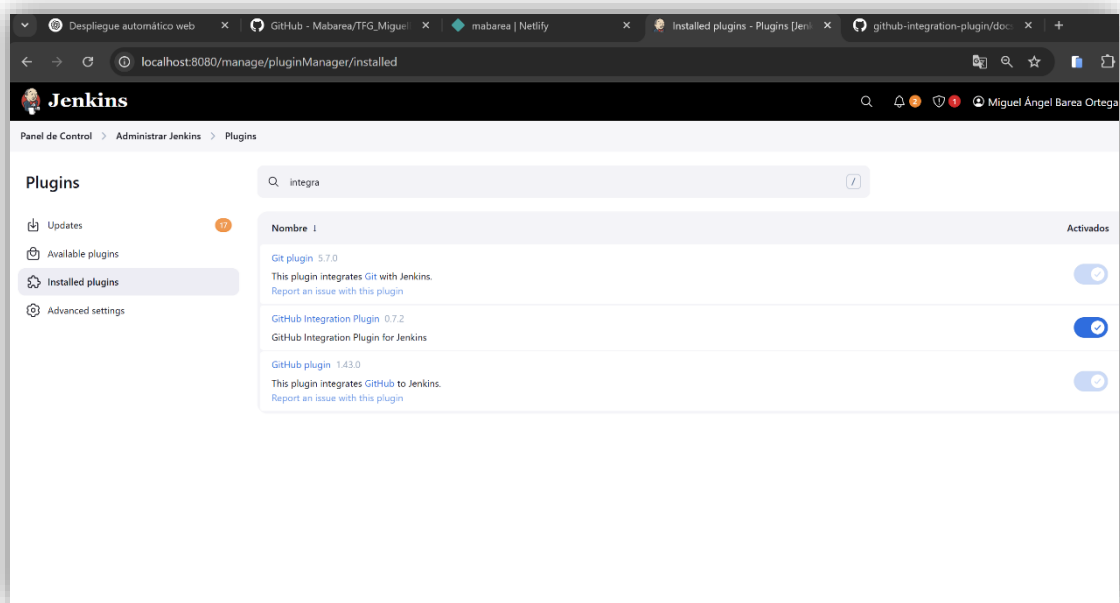


## PASO 9: Automatizar con Webhooks (opcional pero recomendable)

En GitHub, configurar un webhook que apunte a `http://TU_IP_PUBLICA:8080/github-webhook/` para que Jenkins se ejecute al hacer push.

### 1. Instalar el plugin necesario en Jenkins

- Acceder a Jenkins con usuario administrador.
- Ir a **Manage Jenkins > Manage Plugins > Available**.
- Buscar e instalar **GitHub Integration Plugin** (o similar, que incluya webhook support).

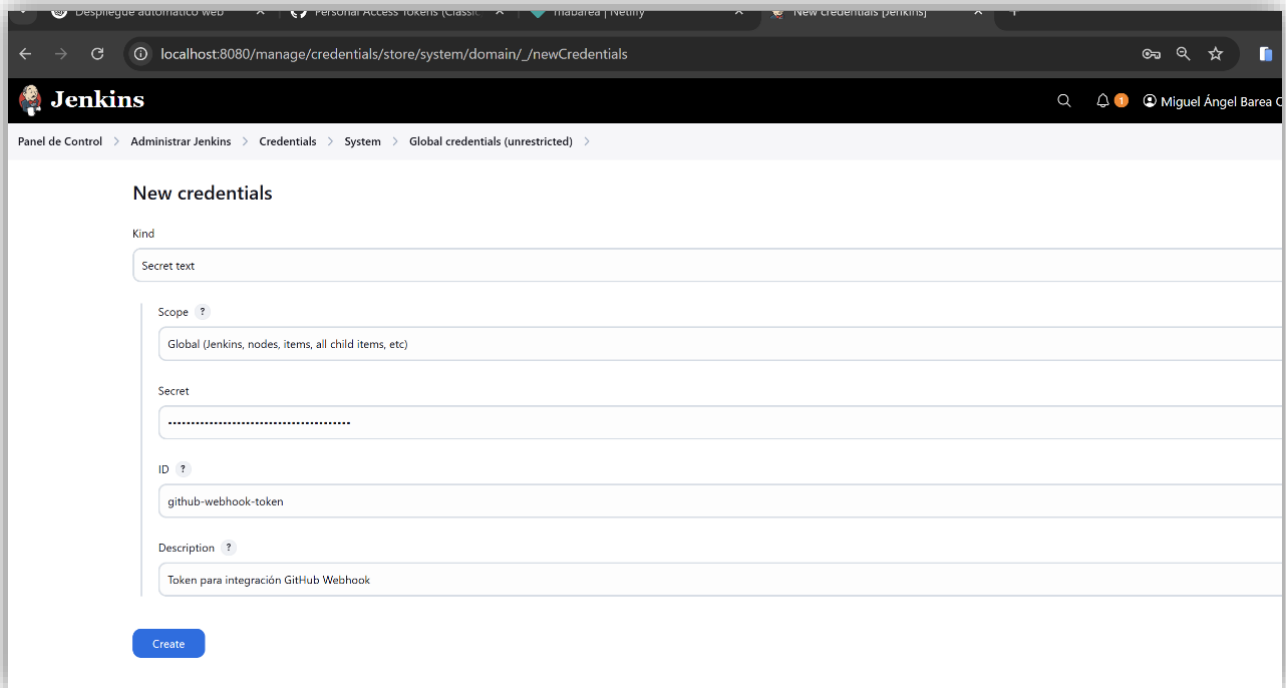


- Reiniciar Jenkins si es necesario.

---

## 2. Configurar las credenciales de GitHub en Jenkins

- Ir a **Manage Jenkins > Manage Credentials > (global)**.
- Añadir nuevas credenciales:
  - Tipo: **Secret text** o **Username with password** (según token o user/pass).
  - Introducir el token personal de GitHub o credenciales.



- Guardar y anotar el ID de credencial para usar en el pipeline.
- 

## 3. Configurar el repositorio Git en Jenkinsfile o pipeline

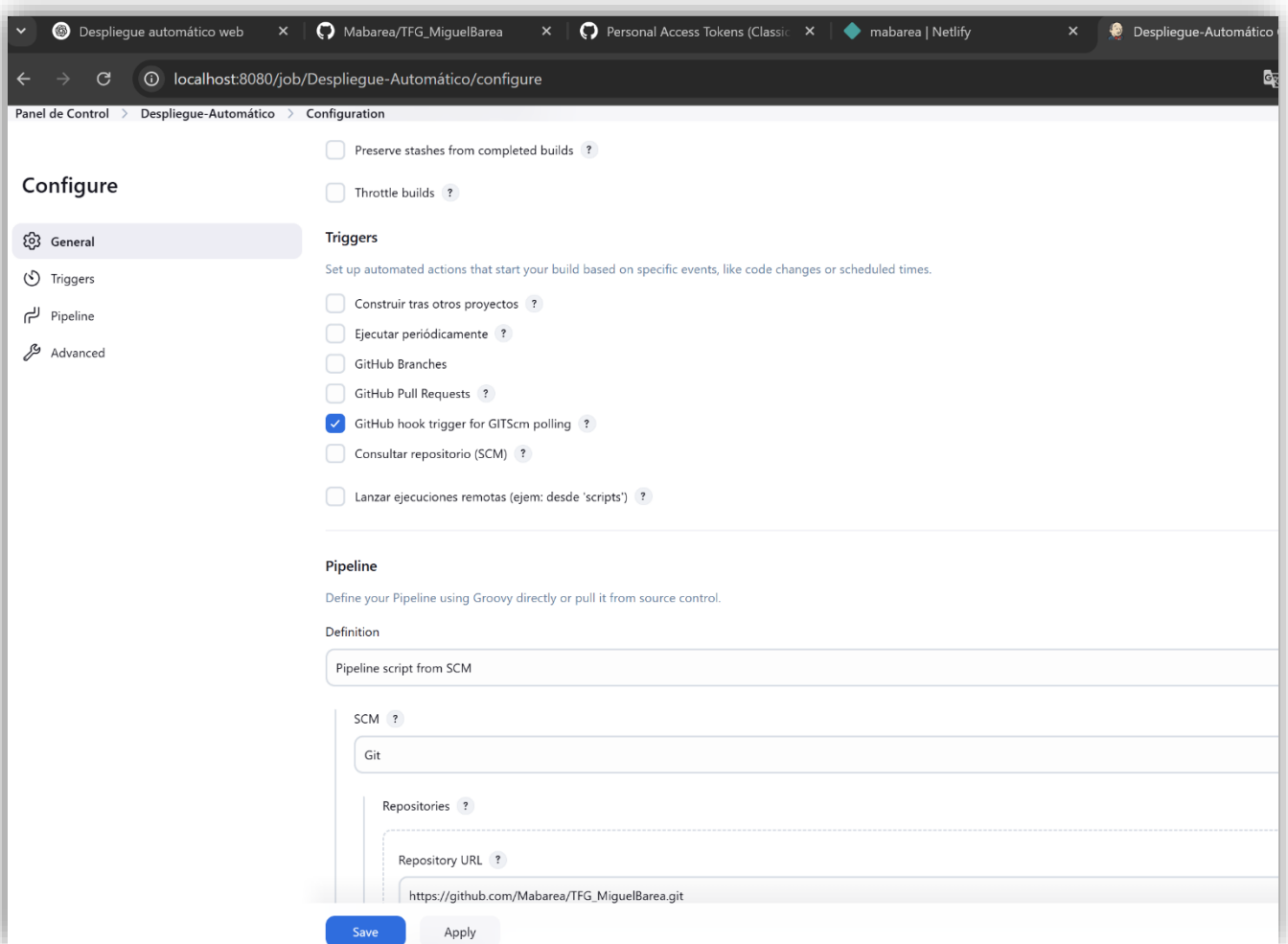
- En el Jenkinsfile, usar:

```
git credentialsId: 'github-token', url:
'https://github.com/usuario/repositorio.git', branch: 'main'
```

- Reemplazar github-token por el ID real de la credencial creada.
- Confirmar que el pipeline clona correctamente el repositorio.

#### 4. Configurar el Job en Jenkins para usar webhook

- Crear un nuevo job o editar uno existente.
- En la sección **Build Triggers** activar **GitHub hook trigger for GITScm polling**.
- Guardar los cambios.



## 5. Configurar el Webhook en GitHub

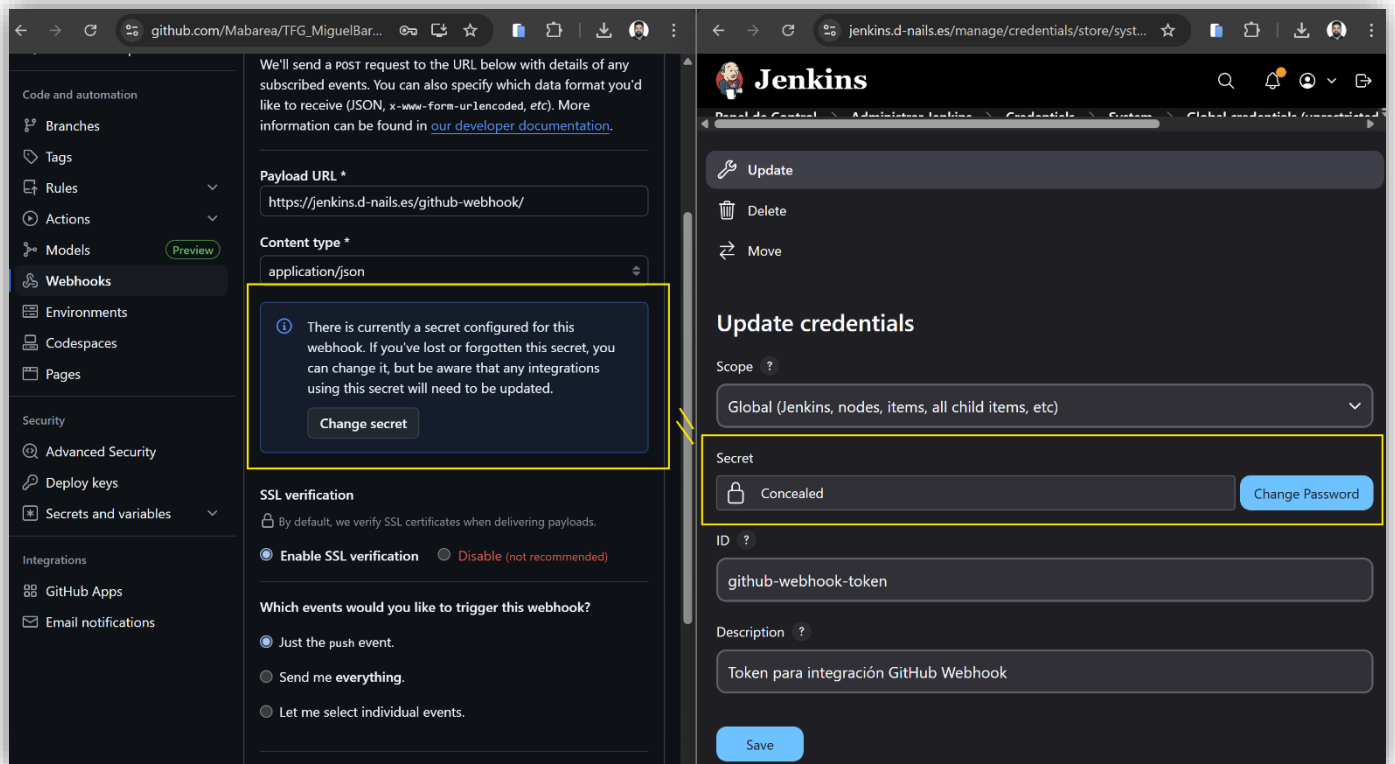
En lugar de usar una IP pública o una URL de ngrok (como se haría en entornos locales temporales), se ha configurado un dominio propio (en mi caso **d-nails.es**) gestionado con Cloudflare para apuntar de forma permanente y segura al Jenkins local expuesto mediante Cloudflare Tunnel.

Pasos:

- Ir al repositorio en GitHub → *Settings* → *Webhooks*.
- Hacer clic en "Add webhook".
- En el campo Payload URL, introducir la ruta de Jenkins para los webhooks, usando el dominio personalizado:

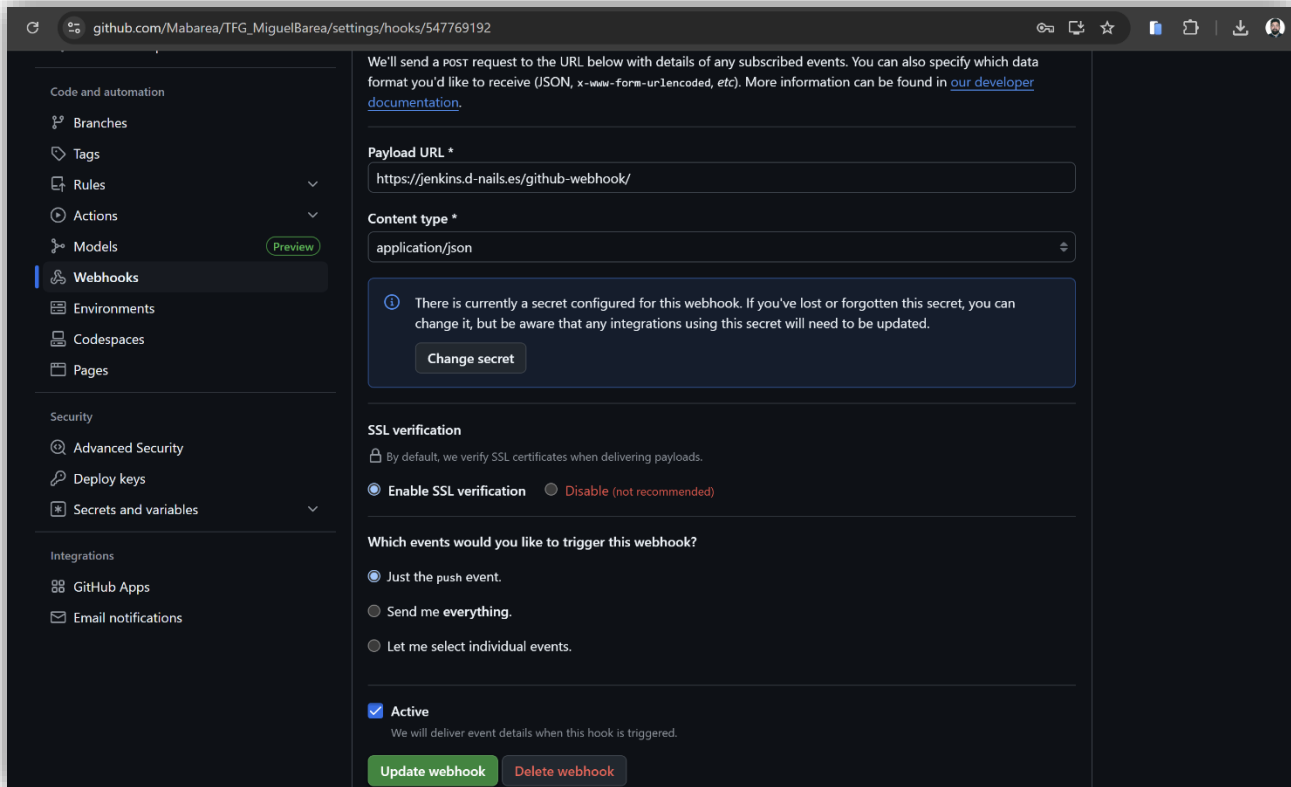
`https://jenkins.d-nails.es/github-webhook/`

- En **Content type**, seleccionar: *application/json*.
- En **Secret**, se puede dejar vacío si Jenkins no lo requiere (aunque recomendable usar el token asociado).





- En **Which events would you like to trigger this webhook?**, marcar:
  - Just the push event.
- Confirmar que **Active** está marcado.
- Hacer clic en **Add webhook**.



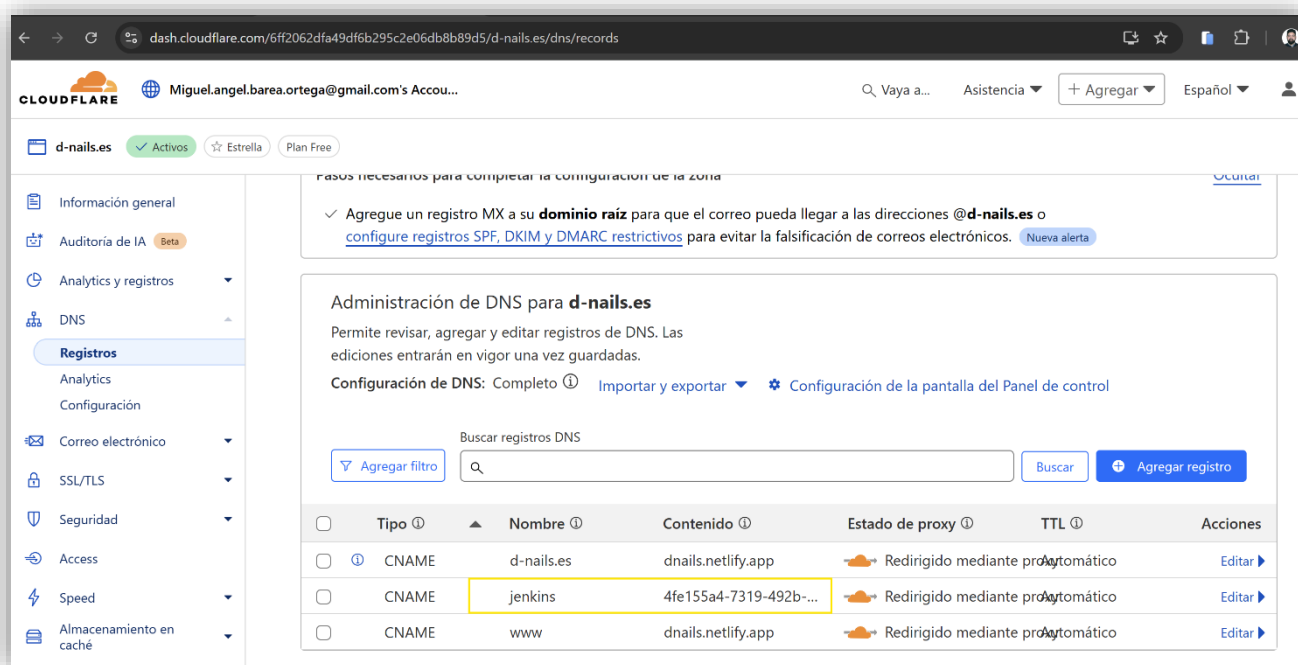
## 6. Exponer Jenkins local mediante dominio personalizado y Cloudflare Tunnel

Para permitir que GitHub se comunice con Jenkins (ubicado en un entorno local), se configuró un túnel seguro y permanente usando Cloudflare Tunnel, como alternativa más robusta y cómoda a herramientas temporales como Ngrok.

Pasos realizados:

### a) Compra y gestión de dominio personalizado:

- Se registró un dominio personalizado (**d-nails.es**) y se gestionó desde Cloudflare.
- Se configuró un subdominio (**jenkins.d-nails.es**) apuntando al túnel de Jenkins.



- Se gestionó un túnel con cloudflared desde un contenedor Docker, expuesto al puerto 8080 del Jenkins local.

b) Instalación de Cloudflare Tunnel (cloudflared):

Se decidió **gestionar el túnel desde un contenedor Docker** para mayor comodidad y persistencia.

- i) Crear un directorio para contener la configuración del túnel (en mi caso cloudflare-tunnel)
- ii) Crear el archivo config.yml en un directorio accesible desde el contenedor:

```
1 tunnel: 4fe155a4-7319-492b-bf27-208209fa2dac
2 credentials-file: /etc/cloudflared/4fe155a4-7319-492b-bf27-208209fa2dac.json
3 origincert: /etc/cloudflared/cert.pem
4
5 ingress:
6   - hostname: jenkins.d-nails.es
7     service: http://host.docker.internal:8080
8   - service: http_status:404
```

El archivo **jenkins-tunnel.json** es generado automáticamente al crear el túnel con el comando anterior y debe montarse en el contenedor.

- iii) Generar el archivo docker-compose.yml con la siguiente configuración:

```
version: '3.8'

services:
  cloudflared:
    image: cloudflare/cloudflared:latest
    container_name: cloudflared-tunnel
    restart: always
    command: ["--config", "/etc/cloudflared/config.yml",
    "tunnel", "run", "4fe155a4-7319-492b-bf27-208209fa2dac"]
    volumes:
      - ../cloudflared:/etc/cloudflared
```

El token se obtiene al crear el túnel desde el panel de Cloudflare → Zero Trust > Access > Tunnels.

iv) Levantar el contenedor:

The screenshot shows the Docker Desktop application window. The top bar includes the Docker logo, 'docker.desktop PERSONAL', a search bar, 'Ctrl+K', and a 'Sign in' button. Below the top bar, a blue banner says 'You can do more when you sign in.' The left sidebar contains navigation links: Containers, Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' and shows a table of running containers. The table has columns for Name, Container ID, Image, Port(s), CPU (%), and Actions. Two containers are listed: 'jenkins-custom' and 'cloudflare-tunnel'. Below the table is a 'Terminal' section showing the output of a 'docker-compose up -d' command. The terminal output indicates that the network 'cloudflare-tunnel\_default' was removed and then created, and the container 'cloudflared-tunnel' was started. A yellow box highlights the terminal output.

**Containers** [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Search  Only show running containers

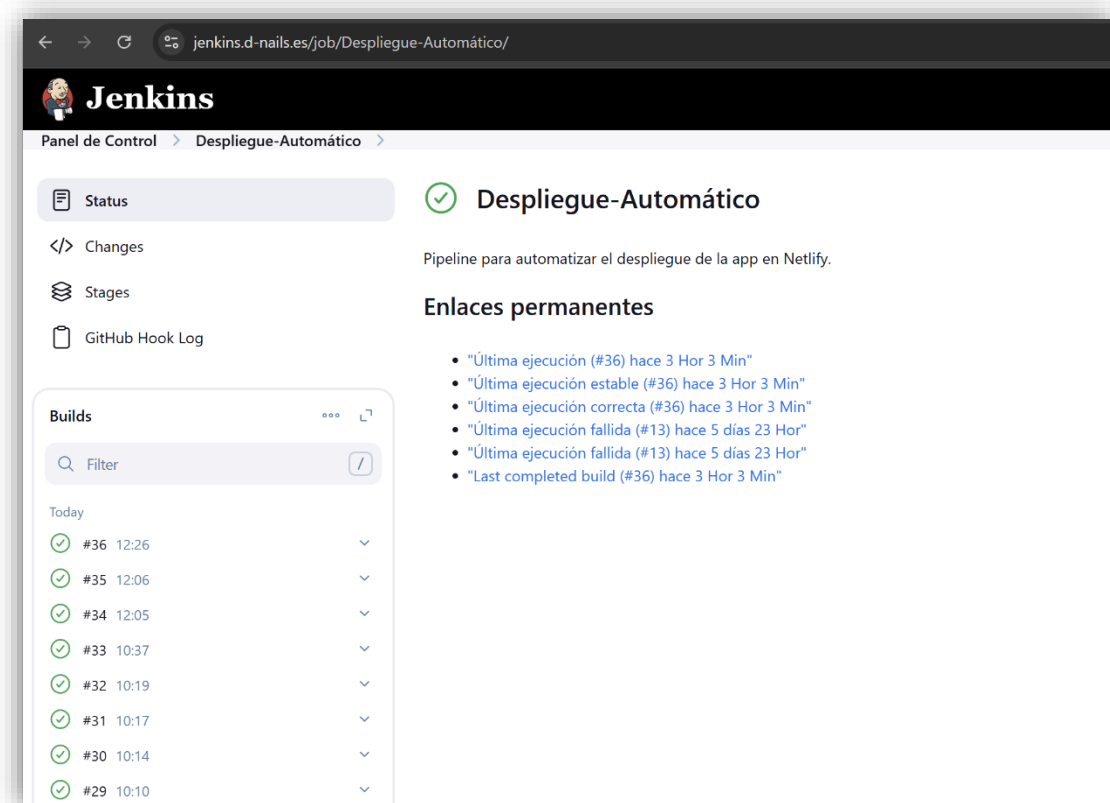
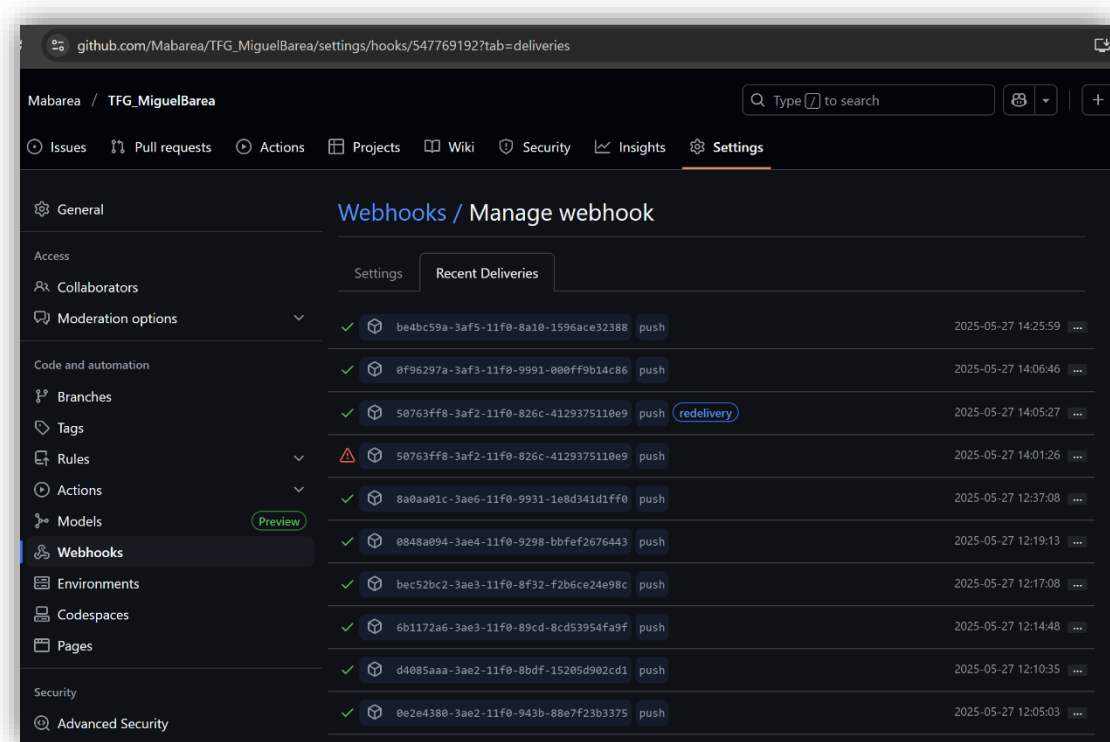
	Name	Container ID	Image	Port(s)	CPU (%)	Actions
<input type="checkbox"/>	jenkins-custom	9c43ec18f4fe	jenkins-cus	50000.50000 <a href="#">Show all ports (2)</a>	0.17%	
<input type="checkbox"/>	cloudflare-tunnel	-	-	-	0.4%	

**Terminal**

```
[+] Running 2/2
✓ Container cloudflared-tunnel Removed 10.7s
✓ Network cloudflare-tunnel_default Removed 0.7s
PS C:\Users\nigue\cloudflare-tunnel> docker-compose up -d
time="2025-05-27T14:04:51+02:00" level=warning msg="C:\\Users\\nigue\\cloudflare-tunnel\\docker-compos
e.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential co
nfusion"
[+] Running 2/2
✓ Network cloudflare-tunnel_default Created 0.0s
✓ Container cloudflared-tunnel Started 0.5s
PS C:\Users\nigue\cloudflare-tunnel>
```

## PASO 10: Prueba final

- Se ha hecho un push de prueba al repositorio de GitHub. Se ha verificado el historial del webhook en GitHub (Webhooks > Recent Deliveries) y los logs del job en Jenkins.



## Conclusiones

- Se ha implementado un flujo CI/CD completo y funcional.
- Jenkins permite automatizar tareas de despliegue eficientemente.
- Netlify simplifica el proceso de publicación web.
- El sistema puede escalarse a proyectos más grandes y complejos.

## Glosario

**CI/CD:** Integración continua / Despliegue continuo.

**Pipeline:** Flujo automatizado de tareas en un proceso de desarrollo.

**Docker:** Plataforma para ejecutar aplicaciones en contenedores.

**Webhook:** Notificación automática de eventos, como un push a un repositorio.

## Referencias bibliográficas utilizadas

- <https://www.jenkins.io/>
- <https://www.netlify.com/>
- Documentación oficial de Docker y GitHub

## Anexos

*(Ver capturas de pantalla del proceso, configuraciones de Jenkins, salidas de consola, pruebas, etc.)*

## Presentación

[https://www.canva.com/design/DAGpMDeSwvo/eOeLE6UPI3Fq9vKCoS17wA/view?utm\\_content=DAGpMDeSwvo&utm\\_campaign=designshare&utm\\_medium=link2&utm\\_source=uniquelinks&utlId=hdbcd191739](https://www.canva.com/design/DAGpMDeSwvo/eOeLE6UPI3Fq9vKCoS17wA/view?utm_content=DAGpMDeSwvo&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=hdbcd191739)