Bootcamp Devops Engineer

Desafío 2

Profesor: Matías Anoniz

Alumno: Ing. Nelson González Escalante

Módulo: Cloud Spetialist

Entrega: 28 de agosto de 2024

Agosto 2024

Tabla de Contenido

Objetivo

Este trabajo tiene como propósito aprender a configurar un webhook en un repositorio de Github, lo cual permitirá automatizar ciertos procesos. Además, se busca crear un pipeline básico en Jenkins que pueda compilar y ejecutar pruebas en un proyecto desarrollado con Node.js. El objetivo es familiarizarnos con herramientas de integración continua (CI/CD) y entender cómo estas pueden mejorar la eficiencia en el desarrollo de software, especialmente en proyectos de aplicaciones web.

Escenario

En este proyecto, se nos ha asignado la tarea de llevar a cabo una prueba de concepto para implementar un sistema de integración y entrega continua (CI/CD) utilizando Node.js. La organización necesita comprender cómo se lleva a cabo el proceso de construcción (build) para una API desarrollada en Node.js. Para ello, nos proporcionaron una aplicación de ejemplo construida con esta tecnología, la cual incluye pruebas automatizadas utilizando Jest, una herramienta nativa del framework.

Además, se nos ha solicitado configurar un webhook en el repositorio de Github, que permita desencadenar automáticamente un build cada vez que se realice un push o un pull request. Esta automatización es esencial para optimizar la ejecución de los trabajos dentro del pipeline CI/CD, garantizando así un flujo de trabajo más eficiente y confiable.

Requisitos

1. Creación de un fork

Realizar un fork de un repositorio en GitHub es una tarea fundamental cuando se quiere trabajar con el código de otro desarrollador sin afectar el repositorio original. A continuación, detallo paso a paso cómo realicé el fork del repositorio "nodejs-helloworld-api" alojado en GitHub.

a. Acceso al repositorio original

Primero, accedí al repositorio original utilizando el enlace proporcionado:

https://github.com/yosoyfunes/nodejs-helloworld-api.

Esto me llevó a la página principal del repositorio en GitHub, donde pude ver el código fuente del proyecto, los archivos, las ramas, los issues, los pull requests, y la documentación asociada.

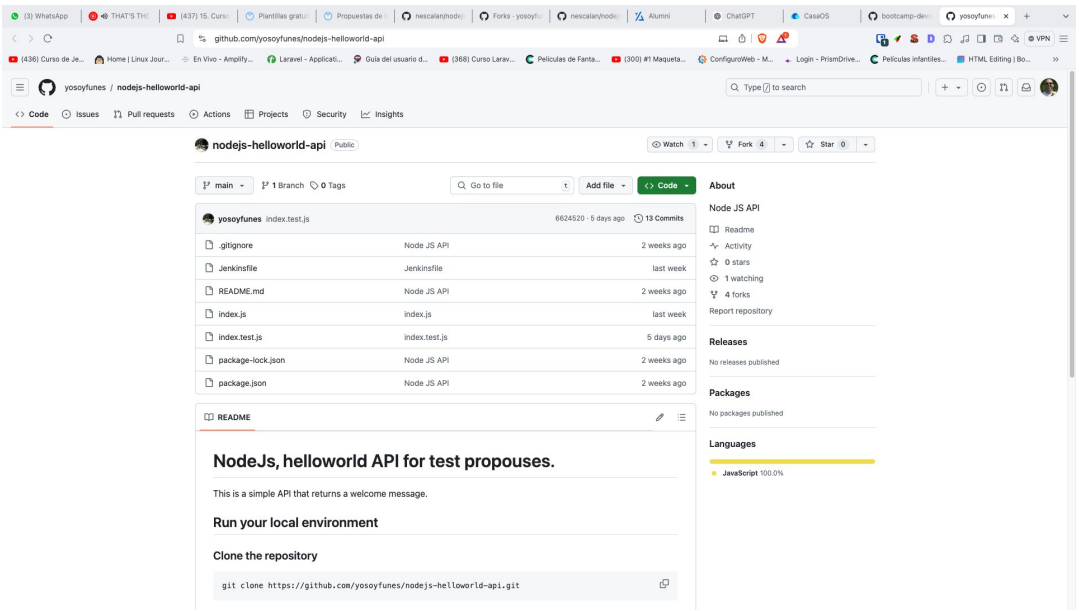

***Figura 1**: Repositorio "nodeja-helloworld-api"*

b. Realizar el fork

Una vez en la página del repositorio, observé en la esquina superior derecha de la interfaz de GitHub un botón que dice **"Fork"**. Este botón se encuentra junto a los botones de "Star" y "Watch".

Hice clic en **"Fork"**, lo que inició el proceso de duplicación del repositorio en mi cuenta personal de GitHub. Durante este proceso, GitHub clona todo el contenido del repositorio original, incluyendo todos los archivos, commits, ramas, y la historia del proyecto.

c. Selección de la Cuenta de GitHub

Después de hacer clic en "Fork", GitHub me pidió seleccionar la cuenta de usuario o la organización donde quería alojar la copia del repositorio. En mi caso, seleccioné mi cuenta personal, ya que el objetivo era tener una copia en mi propio espacio de trabajo.

d. Espera y Confirmación

El proceso de fork tomó unos segundos. Una vez completado, fui redirigido automáticamente a la página del nuevo repositorio en mi cuenta de GitHub. Este nuevo repositorio tiene el mismo nombre que el original, pero ahora está bajo mi usuario, en la siguiente dirección, *https://github.com/nescalan/nodejs-helloworld-api*

e. Verificación del Fork

Para asegurarme de que el fork se realizó correctamente, verifiqué que todos los archivos y la historia del commit estuvieran presentes en mi nuevo repositorio. Además, en la parte superior de la página del repositorio, GitHub muestra un enlace que dice *"forked from yosoyfunes/nodejs-helloworld-api"*, lo que indica claramente que este repositorio es un fork de otro.

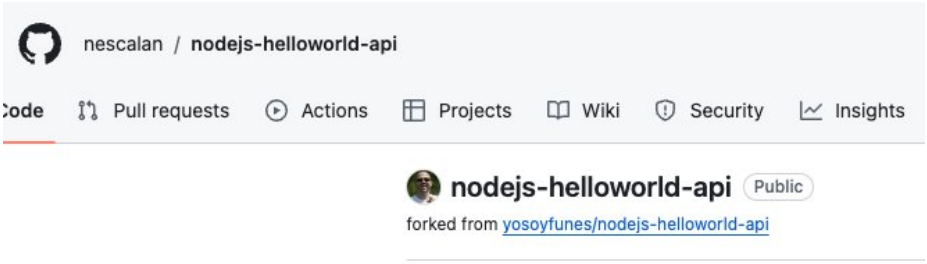

***Figura 2**: Fork del repositorio*

f. Clonación Local (Opcional)

Aunque no es un paso necesario para realizar el fork, decidí clonar el repositorio en mi máquina local para poder trabajar con el código de manera más cómoda. Para hacer esto, ejecuté el siguiente comando en mi terminal:

git clone https://github.com/miusuario/nodejs-helloworld-api.git

Este comando descargó una copia local del repositorio forked, permitiéndome realizar modificaciones, probar la aplicación y subir cambios a mi repositorio en GitHub.

2. Configuración de github webhook

La configuración de un webhook en GitHub es un paso crucial para automatizar la integración continua y entrega continua (CI/CD). En este caso, configuré un webhook para que, cada vez que se produzca un push o se cree un pull request (PR) en el repositorio, se inicialice automáticamente un job en Jenkins. A continuación, detallo los pasos que seguí para realizar esta configuración.

a. Acceso al Repositorio Forkeado

Primero, accedí al repositorio forkeado en mi cuenta de GitHub. Para ello, fui a GitHub e inicié sesión con mi usuario. Luego, busqué y seleccioné el repositorio "nodejs-helloworld-api" que había forkeado anteriormente.

b. Acceso a la Configuración del Repositorio

Una vez dentro del repositorio, hice clic en la pestaña "Settings" (Configuración), que se encuentra en la parte superior de la página, junto a las pestañas de "Code", "Issues", "Pull Requests", entre otras. La sección de configuración permite gestionar diversos aspectos del repositorio, incluyendo la configuración de webhooks.

c. Navegación a la Sección de Webhooks

Dentro de la sección de "Settings", en el menú lateral izquierdo, busqué y seleccioné la opción "Webhooks". Esta opción es donde se pueden gestionar los webhooks, que son herramientas que permiten que un sistema (en este caso, GitHub) envíe información a otro sistema (como Jenkins) automáticamente cuando ocurren ciertos eventos.

d. Creación de un Nuevo Webhook

En la página de "Webhooks", hice clic en el botón "Add webhook" (Añadir webhook) ubicado en la esquina superior derecha. Después de confirmar el acceso, se abrió un formulario donde se configuran los detalles del webhook.

e. Configuración de la Payload URL

El primer campo que configuré fue la "Payload URL". Aquí ingresé la URL de mi instancia de Jenkins que estaba siendo expuesta a través de Ngrok, lo cual permite que Jenkins sea accesible desde internet para recibir las solicitudes de GitHub. La URL se veía algo así:

https://2777-190-113-110-91.ngrok-free.app/login?from=%2F

Este enlace apunta directamente al endpoint de Jenkins que está configurado para recibir eventos de GitHub.

f. Selección del Content Type

En el campo "Content type", seleccioné "application/json". Este formato es el más adecuado para que Jenkins procese correctamente la información enviada por GitHub.

g. Configuración de los Eventos

Luego, en la sección de "Which events would you like to trigger this webhook?" (¿Qué eventos te gustaría que activaran este webhook?), seleccioné la opción "Just the push event." (Solo el evento de push). Esto asegura que el webhook se active cada vez que se haga un push al repositorio.

h. Configuración de los Secrets (Opcional)

En el campo de "Secret", si se desea mayor seguridad, se puede configurar una cadena secreta (token) que Jenkins utilizará para verificar que las solicitudes provienen realmente de GitHub y no de un tercero no autorizado. Esta opción es recomendable en entornos de producción, aunque en este caso la dejé en blanco para simplificar el proceso.

i. Guardar y Verificar el Webhook

Finalmente, hice clic en "Add webhook" (Añadir webhook) para guardar la configuración. GitHub me mostró una lista de todos los webhooks configurados para el repositorio, donde pude ver el nuevo webhook que acababa de crear.

j. Verificación de Funcionamiento

Para asegurarme de que el webhook funcionara correctamente, realicé un pequeño push a una rama del repositorio. Inmediatamente, verifiqué en Jenkins que el job se iniciara automáticamente. Para hacerlo, fui a la interfaz de Jenkins y comprobé que el pipeline se estaba ejecutando en respuesta al push que había hecho. Esto confirmó que el webhook estaba correctamente configurado y que GitHub estaba comunicándose con Jenkins como se esperaba.

k. Ajustes y Pruebas Adicionales

Después de la configuración inicial, realicé varias pruebas adicionales, incluyendo la creación de pull requests y pushes en diferentes ramas, para asegurarme de que el webhook respondiera correctamente en todos los escenarios posibles. Esto me permitió verificar que el pipeline se activara y ejecutara los pasos necesarios cada vez que se produjera uno de estos eventos en el repositorio.

3. Exposición con ngrok

Exponer Jenkins a través de Ngrok es un paso importante cuando se trabaja en un entorno de desarrollo local y se necesita que servicios externos, como GitHub, puedan interactuar con Jenkins. Ngrok permite que un servidor local, que normalmente solo es accesible dentro de una red privada, esté disponible públicamente a través de una URL segura y temporal. A continuación, explico de manera detallada cómo realicé la exposición de mi servicio de Jenkins utilizando Ngrok.

a. Instalación de Ngrok

Antes de poder utilizar Ngrok, fue necesario instalarlo en mi sistema. Ngrok está disponible para diferentes sistemas operativos, y yo utilicé la versión para Linux.

b. Descarga de Ngrok:

Ingresé al sitio web oficial de Ngrok en (https://ngrok.com/) y descargué la versión adecuada para mi sistema operativo utilizando el siguiente comando:

```
snap install ngrok
```

c. Verificación de la Instalación:

Para asegurarme de que Ngrok estaba instalado correctamente, ejecuté el siguiente comando:

```
ngrok  -- version
```

Esto mostró la versión de Ngrok instalada, confirmando que estaba listo para ser utilizado.

2. Configuración del Servicio de Jenkins

Antes de exponer Jenkins, me aseguré de que el servicio de Jenkins estuviera funcionando correctamente en mi máquina local. Jenkins generalmente corre en el puerto `8080` por defecto.

a. Verificación de Jenkins:

Abrí el navegador web y navegué a `http://localhost:8080` para confirmar que Jenkins estaba corriendo y accesible localmente.

b. Ajustes Opcionales en Jenkins:

Aunque no fue necesario en mi caso, es posible que se deba configurar un nombre de usuario y una contraseña en Jenkins para limitar el acceso, especialmente cuando se expone a internet.

c. Exponiendo Jenkins con Ngrok

Con Ngrok instalado y Jenkins corriendo localmente, procedí a exponer Jenkins a través de Ngrok:

1. Iniciar Ngrok en el Puerto 8080:

Ejecuté Ngrok utilizando el siguiente comando en la terminal:

```
ngrok http http://localhost:8080
```

Este comando le indicó a Ngrok que creara un túnel desde internet hacia el puerto `8080` de mi máquina local, donde Jenkins estaba corriendo. Obtuve un error de comunicación entre ngrok debido a que yo no había actualizado la políticas del firewall de Linux, y lo resolví de la siguiente manera:

```
sudo ufw allow 8080

sudo ufw enable

sudo ufw status
```

2. Recepción de la URL Pública:

Tras ejecutar el comando, Ngrok generó una URL pública temporal que redirige hacia el puerto `8080` de mi máquina. Esta URL suele tener el siguiente formato:

https://cad8-190-113-110-91.ngrok-free.app

Cabe indicqr que se generó una versión HTTPS de la URL, lo que es importante para trabajar con servicios que requieren conexiones seguras.

3. Verificación del Acceso Externo a Jenkins:

Probé la URL generada por Ngrok en mi navegador para asegurarme de que Jenkins era accesible desde fuera de mi red local. Ingresando la URL pública proporcionada por Ngrok en el navegador, pude acceder a la interfaz de Jenkins, lo que confirmó que Jenkins estaba efectivamente expuesto a través de internet.

4. Integración con GitHub Webhooks

Con Jenkins expuesto públicamente a través de Ngrok, el siguiente paso fue utilizar la URL pública generada en la configuración del webhook de GitHub. Esto permitió que GitHub enviara solicitudes a Jenkins cada vez que ocurría un evento relevante (como un push o un pull request).

5. Configuración del Webhook:

Usé la URL pública proporcionada por Ngrok en la sección "Payload URL" del webhook en GitHub, asegurándome de agregar `/github-webhook/` al final de la URL para que Jenkins pudiera recibir y procesar las solicitudes correctamente.

Prueba del Webhook:

Después de configurar el webhook con la URL de Ngrok, realicé un push en el repositorio para verificar que Jenkins recibiera la solicitud y ejecutara el pipeline automáticamente.

5. Manteniendo Ngrok Activo

Es importante notar que la URL generada por Ngrok es temporal y se cierra una vez que se detiene Ngrok. En entornos de desarrollo, esto es suficiente, pero si se requiere una URL persistente, Ngrok ofrece opciones de pago para reservar subdominios específicos.

1. Mantener el Túnel Activo:

Mientras estaba trabajando en el proyecto, mantuve la terminal abierta con Ngrok corriendo para asegurarme de que el servicio de Jenkins permaneciera accesible.

2. Reiniciar Ngrok (si es necesario):

Si Ngrok se detiene o si se necesita una nueva URL, simplemente se puede ejecutar el comando `ngrok http 8080` nuevamente para generar una nueva URL.

4. Pipeline de jenkins

Para configurar un pipeline en Jenkins que ejecute una serie de pasos necesarios para el desarrollo de un proyecto NodeJS, es esencial entender tanto la instalación de herramientas como la configuración adecuada de Jenkins y GitHub. A continuación, detallo el proceso paso a paso, desde la instalación de NodeJS hasta la definición de herramientas en el Jenkinsfile.

a. Instalación de NodeJS en el Servidor

El primer paso fue asegurarme de que NodeJS estuviera instalado en el servidor, ya que es esencial para ejecutar proyectos basados en esta tecnología.

1. **Acceder al Servidor:**

   Ingresé al servidor a través de una terminal o conexión SSH.

2. **Verificación de la Instalación de NodeJS:**

   Para verificar si NodeJS ya estaba instalado, ejecuté el siguiente comando en la terminal:

```
nodejs -- version
```

El comando devolvió la versión de NodeJS instalada, lo que confirmó que NodeJS ya estaba presente en el servidor y no fue necesario realizar una nueva instalación.

Con esta verificación, pude continuar con la configuración de Jenkins para utilizar la instalación existente de NodeJS en el servidor.

b. Instalación de Plugins de Jenkins Necesarios para NodeJS

Jenkins necesita ciertos plugins para interactuar con proyectos NodeJS y GitHub, así que procedí a instalarlos.

1. **Acceder al Administrador de Plugins:**

   Desde el menú principal, seleccioné "Administrar Jenkins".

   Luego, hice clic en "Plugins".

2. **Buscar e Instalar los Plugins Requeridos:**

   En la pestaña "Disponible", busqué los siguientes plugins:

   NodeJS Plugin: Este plugin es crucial para trabajar con proyectos NodeJS en Jenkins.

GitHub Plugin: Permite la integración con GitHub, esencial para usar webhooks y SCM.

Pipeline Plugin: Necesario para crear y gestionar pipelines en Jenkins.

Stage View: Permite una representación gráfica de los “Stages” y los “Steps” de un Pipeline en Jenkins.

Seleccioné los plugins mencionados y los instalé. Hice clic en "Instalar sin reiniciar" para que Jenkins aplicara los cambios sin necesidad de reiniciar el servicio.

c. Creación de un Job Tipo Pipeline en Jenkins

Con NodeJS y los plugins instalados, procedí a crear un job en Jenkins que ejecutara los pasos de desarrollo del proyecto.

1. Crear un Nuevo Job:

- Desde el panel principal de Jenkins, seleccioné "Nuevo Item" (New Item).
- Ingresé un nombre para el job, como "NodeJS-CICD-Pipeline".
- Seleccioné "Pipeline" como tipo de proyecto y hice clic en "OK" para proceder a la configuración.

2. Configurar el Pipeline:

- En la sección de configuración del pipeline, especifiqué la opción "Pipeline script from SCM" (script de Pipeline desde el sistema de control de versiones).
- Elegí "Git" como el sistema de control de versiones.
- Ingresé la URL del repositorio de GitHub donde se alojaba el código del proyecto.

d. Configuración del Access Token en GitHub

Para permitir que Jenkins accediera al repositorio de GitHub, creé y configuré un access token en GitHub.

1. **Crear un Access Token:**
   - Ingresé a mi cuenta de GitHub y fui a la sección "Settings".
   - Seleccioné "Developer settings" y luego la opción "Tokens (classic)".
   - Hice clic en "Generate new token", luego a “Generate new token (classic)”, después de confirmar el acceso a github le di un nombre al token, como "Jenkins-Access-Token" con una validez de 30 días.
   - Marqué los permisos necesarios, como lo es el acceso al repositorio (repo), y generé el token.
2. **Guardar el Access Token:**

   Guardé el token generado en un lugar seguro, ya que solo se muestra una vez en GitHub.

e. Configuración del Token en Jenkins SCM

Con el token de GitHub, procedí a configurar el acceso en Jenkins.

1. **Configurar Credenciales en Jenkins:**
   - En la sección de configuración del pipeline en Jenkins, seleccioné "Add" para añadir nuevas credenciales.
   - Elegí "Username with password" como tipo de credencial.
   - Ingresé mi nombre de usuario de GitHub y en el campo de la contraseña, pegué el access token generado anteriormente.

- Guardé las credenciales y las asocié al repositorio configurado en el pipeline.
- Verifiqué la rama en la que voy a trabajar pero estaba seleccionada como “master” por lo que procedí a cambiarla a “main”.

f. Configuración del Path Relativo al Jenkinsfile

Para que Jenkins pueda ejecutar el pipeline, es necesario definir el path relativo al Jenkinsfile dentro del repositorio.

1. **Definir el Jenkinsfile:**
   - En la configuración del job, en el campo "Script Path" dentro de la sección del pipeline, ingresé la ruta relativa al Jenkinsfile. El nombre de Jenkinsfile estaba ya en la raíz del repositorio, simplemente verifiqué que estuviera bien escrito.
   - Esto indicó a Jenkins dónde encontrar el archivo que contiene la definición del pipeline dentro del repositorio de Gihub.

g. Definición de Herramientas en el Jenkinsfile

Finalmente, configuré el Jenkinsfile para definir las herramientas necesarias y los pasos del pipeline.

2. Creación del Jenkinsfile:

- El Jenkinsfile es un archivo que contiene la definición del pipeline en lenguaje Groovy. Con la configuración anterior, se descarga del repositorio con el siguiente contenido básico:

```
pipeline {
    agent any

    tools {
        nodejs "NodeJS-14"
    }

    stages {
        stage('Install Dependencies') {
            steps {
                sh 'npm install'
            }
        }

        stage('Run Tests') {
            steps {
                sh 'npm test'
            }
        }
    }
}
```

3. Definición de Herramientas:

En la sección tools, definí que el pipeline debía usar la versión de NodeJS instalada anteriormente en Jenkins (en este caso selecioné "NodeJS-21.0.0").

Esto aseguró que el entorno de ejecución del pipeline estuviera correctamente configurado con NodeJS.

4. Definición de Stages:

El Jenkinsfile contiene dos etapas (stages) en el pipeline:

1. **Builе**: Esta etapa ejecutó el comando npm install para instalar las dependencias del proyecto.
2. **Tests**: En esta etapa, ejecuté las pruebas del proyecto utilizando npm test, asegurando que todo el código estuviera funcionando correctamente.

5. Guardado y Commit:

Guardé el Jenkinsfile en el repositorio y realicé un commit para que Jenkins pudiera acceder a este archivo y ejecutar el pipeline.

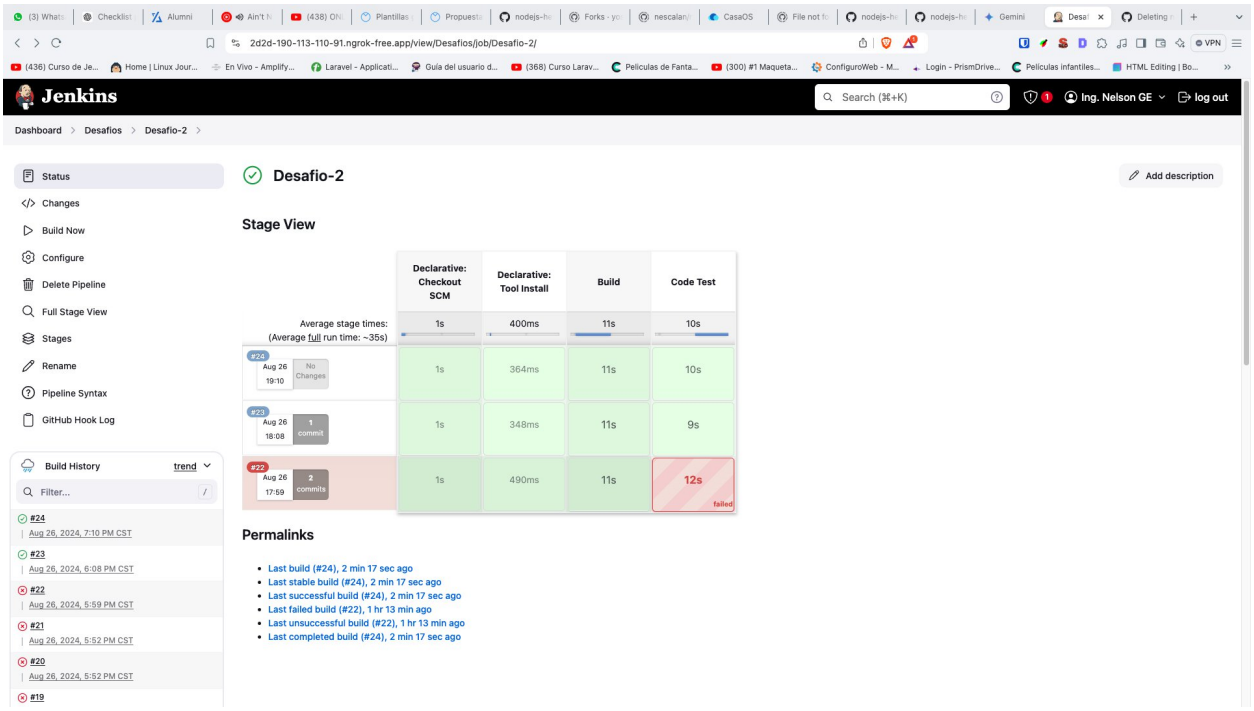

***Figura 3**: Resultado del Pipeline*

Indice de Imágenes