Pipeline CI/CD

Github Actions

Julian F. Latorre

16 de agosto de 2024

Índice

1. Introducción

Este instructivo detalla paso a paso cómo configurar GitHub Actions para ejecutar pruebas Jest y Cypress automáticamente cuando se realiza un push a tu repositorio. Esta configuración es esencial para un flujo de trabajo de integración continua en desarrollo fullstack.

1.1. GitHub Actions

GitHub Actions es una plataforma de automatización, CI/CD integrada en GitHub. Permite a los desarrolladores automatizar flujos de trabajo directamente desde sus repositorios de GitHub.

Características clave de GitHub Actions:

- Automatización de flujos de trabajo de desarrollo de software.
- Integración nativa con repositorios de GitHub.
- Ejecución de trabajos en máquinas virtuales o contenedores.
- Soporte para múltiples lenguajes de programación y frameworks.

2. Primeros Pasos con GitHub Actions

GitHub Actions es una potente herramienta de automatización integrada directamente en GitHub. Esta sección te guiará a través de los primeros pasos para comenzar a utilizar GitHub Actions en tu proyecto.

2.1. Acceso a GitHub Actions

Para comenzar con GitHub Actions:

1. Inicia sesión en tu cuenta de GitHub.
2. Navega hasta el repositorio donde deseas implementar GitHub Actions.
3. En la barra de navegación superior del repositorio, haz clic en la pestaña "Actions".
4. Verás la página "Get started with GitHub Actions".

2.2. Configuración Inicial

Una vez en la página "Get started with GitHub Actions", tienes varias opciones:

- **Usar un flujo de trabajo preconfigurado**: GitHub ofrece una variedad de plantillas de flujo de trabajo para diferentes lenguajes de programación y frameworks.

- **Configurar un flujo de trabajo personalizado**: Puedes crear tu propio archivo de flujo de trabajo desde cero.
- **Explorar flujos de trabajo de la comunidad**: Puedes buscar y utilizar flujos de trabajo creados por la comunidad de GitHub.

3. Fujo de trabajo

3.1. Paso 1: Crear el archivo de configuración de GitHub Actions

1. En la raíz de tu repositorio, crea un directorio llamado `.github/workflows`
2. Dentro de este directorio, crea un archivo YAML, por ejemplo, `ci.yml`

3.2. Paso 2: Configurar el archivo YAML

Abre el archivo `ci.yml` y agrega el siguiente contenido:

```yaml
name: CI

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main, develop ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Use Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18.x'

    - name: Install dependencies
      run: npm ci

    - name: Run Jest tests
      run: npm run test

    - name: Run Cypress tests
      uses: cypress-io/github-action@v6
      with:
        build: npm run build
        start: npm start
        wait-on: 'http://localhost:3000'
```

- `name`: Nombre del flujo de trabajo

- `on`: Especifica cuándo se ejecutará el flujo de trabajo (en push y pull requests a las ramas main y develop)
- `jobs`: Define los trabajos a ejecutar
- `test`: Nombre del trabajo
- `runs-on`: Especifica el sistema operativo para ejecutar el trabajo
- `steps`: Lista de pasos a ejecutar

Los pasos incluyen:

1. Clonar el repositorio
2. Configurar Node.js
3. Instalar dependencias
4. Ejecutar pruebas Jest
5. Ejecutar pruebas Cypress

3.3. Paso 3: Configurar scripts en package.json

Asegúrate de que tu `package.json` tenga los siguientes scripts:

```
{
  "scripts": {
    "test": "jest",
    "build": "your-build-command",
    "start": "your-start-command"
  }
}
```

Reemplaza `your-build-command` y `your-start-command` con los comandos específicos de tu proyecto.

3.4. Paso 4: Commit y Push

Realiza un commit de los cambios y haz push a tu repositorio:

```
git add .github/workflows/ci.yml
git commit -m "Add GitHub Actions CI workflow"
git push origin main
```

4. Optimización y Buenas Prácticas

4.1. Caché de Dependencias

Para mejorar el rendimiento de las ejecuciones subsiguientes, es recomendable implementar el caché de dependencias:

```
- name: Cache Cypress binary
  uses: actions/cache@v3
  with:
    path: ~/.cache/Cypress
    key: cypress-${{ runner.os }}-cypress-${{ hashFiles('**/package
    .json') }}
    restore-keys: |
      cypress-${{ runner.os }}-cypress-
```

Esta configuración almacena en caché el binario de Cypress, reduciendo significativamente el tiempo de instalación en ejecuciones futuras.

4.2. Consideraciones de Rendimiento

- Utiliza selectores estables en tus pruebas, preferiblemente atributos `data-*`.
- Evita dependencias de datos externos; usa fixtures o mocks cuando sea posible.
- Divide las pruebas en conjuntos más pequeños para ejecuciones más rápidas.
- Utiliza la ejecución paralela para proyectos grandes.

4.3. Mantenimiento

- Mantén actualizadas tus dependencias, incluyendo Cypress y la acción de GitHub.
- Revisa periódicamente tus pruebas para asegurar que siguen siendo relevantes y efectivas.
- Configura notificaciones para fallos de pruebas para una respuesta rápida a los problemas.

4.4. Variables de entorno

Si tu aplicación requiere variables de entorno, puedes configurarlas en GitHub:

1. Ve a tu repositorio en GitHub
2. Haz clic en "Settings» "Secrets"
3. Agrega tus secretos

Luego, en tu archivo YAML:

```
env:
  MY_SECRET: ${{ secrets.MY_SECRET }}
```

4.5. Matriz de pruebas

Para probar en múltiples versiones de Node.js:

```
strategy:
  matrix:
    node-version: [12.x, 14.x, 16.x, 18.x]
```

5. Configuración de Notificaciones

La configuración de notificaciones es una parte importante del proceso de integración continua, ya que permite a los desarrolladores y equipos mantenerse informados sobre el estado de las pruebas de forma proactiva. GitHub Actions ofrece varias opciones para configurar estas notificaciones. A continuación, se detalla cómo configurar notificaciones por email para estar informado sobre el estado de las pruebas.

5.1. Notificaciones Nativas de GitHub

GitHub proporciona notificaciones por defecto para los eventos en los repositorios, incluyendo los resultados de GitHub Actions. Para configurar estas notificaciones:

1. Inicia sesión en tu cuenta de GitHub.
2. Ve a "Settings"(Configuración) >"Notifications"(Notificaciones).
3. En la sección .Email notifications"(Notificaciones por email), asegúrate de que la opción "Send notifications for the repositories I'm watching"(Enviar notificaciones para los repositorios que estoy observando) esté marcada.
4. Configura la frecuencia de las notificaciones según tus preferencias.

Sin embargo, estas notificaciones pueden ser generales y no específicas para las pruebas de Cypress.

5.2. Configuración de Notificaciones Personalizadas

Para obtener notificaciones más específicas sobre el estado de las pruebas de Cypress, puedes agregar un paso adicional en tu flujo de trabajo de GitHub Actions para enviar emails personalizados. Esto se puede lograr utilizando una acción de terceros o un script personalizado.

5.2.1. Usando una Acción de Terceros

Una opción popular es utilizar la acción .Email on Failure"de Pozetron. Aquí tienes un ejemplo de cómo integrarla en tu flujo de trabajo:

```yaml
- name: Send email on failure
  if: failure()
  uses: pozetroninc/github-action-send-mail@master
  with:
    server_address: smtp.gmail.com
    server_port: 465
    username: ${{ secrets.EMAIL_USER }}
    password: ${{ secrets.EMAIL_PASS }}
    subject: Falló la ejecución de pruebas Cypress
    body: Las pruebas Cypress fallaron en la última ejecución. Por
    favor, revisa los resultados.
    to: tu-email@ejemplo.com
    from: GitHub Actions
```

- `if: failure()`: Esta condición asegura que el email se envíe solo si las pruebas fallan.
- `server_address` y `server_port`: Configuración del servidor SMTP.
- `username` y `password`: Credenciales de la cuenta de email, almacenadas como secretos en GitHub.
- `subject` y `body`: Asunto y cuerpo del email.
- `to`: Dirección de email del destinatario.
- `from`: Nombre del remitente que aparecerá en el email.

5.2.2. Usando un Script Personalizado

Para un control más granular, puedes crear un script personalizado que envíe emails. El siguiente es un ejemplo utilizando Node.js y la librería `nodemailer`:

```javascript
// send-email.js
const nodemailer = require('nodemailer');

async function sendEmail() {
  let transporter = nodemailer.createTransport({
    host: "smtp.gmail.com",
    port: 587,
    secure: false,
    auth: {
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS
    }
  });

  let info = await transporter.sendMail({
    from: '"GitHub Actions" <github-actions@ejemplo.com>',
    to: "tu-email@ejemplo.com",
    subject: "Resultado de pruebas Cypress",
    text: "Las pruebas Cypress han finalizado. Por favor, revisa
    los resultados.",
    html: "<b>Las pruebas Cypress han finalizado. Por favor, revisa
     los resultados.</b>"
```

```
  });

  console.log("Message sent: %s", info.messageId);
}

sendEmail().catch(console.error);
```

Luego, puedes agregar un paso en tu flujo de trabajo para ejecutar este script:

```
- name: Send email notification
  if: always()
  run: |
    npm install nodemailer
    node send-email.js
  env:
    EMAIL_USER: ${{ secrets.EMAIL_USER }}
    EMAIL_PASS: ${{ secrets.EMAIL_PASS }}
```

5.3. Consideraciones de Seguridad

Al configurar notificaciones por email, hay que tener en cuenta las siguientes consideraciones de seguridad:

- Nunca expongas directamente tus credenciales de email en el código o en los archivos de configuración.
- Utiliza siempre secretos de GitHub para almacenar información sensible como nombres de usuario y contraseñas.
- Si estás utilizando Gmail, considera crear una contraseña de aplicación específica en lugar de usar tu contraseña principal.
- Revisa regularmente los permisos y accesos de las aplicaciones de terceros que utilices para enviar emails.

5.4. Personalización de Contenido

Para hacer que las notificaciones sean más útiles, considera incluir información detallada en el cuerpo del email:

- Enlace directo a los resultados de la ejecución en GitHub Actions.
- Resumen de las pruebas que fallaron.
- Información sobre el commit que desencadenó la ejecución.
- Enlaces a los artefactos generados, como capturas de pantalla o videos de las pruebas fallidas.

5.5. Recursos Adicionales

Para profundizar en GitHub Actions:

- Consulta la documentación oficial de GitHub Actions: `https://docs.github.com/en/actions`
- Explora el marketplace de GitHub Actions para encontrar acciones útiles: `https://github.com/marketplace?type=actions`
- Únete a la comunidad de GitHub para hacer preguntas y compartir conocimientos: `https://github.community/`