Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Laboratorio 6: CI/CD usando GitHub Actions

Autor:
Gabriel Murillo Medina

Docente:
Ing. Enrique Calvopiña, Mgtr

Asignatura: Pruebas de Software
NRC: 22431

22 de enero de 2026

Índice

Índice de figuras

Listings

1. Introducción

La integración continua (CI) es una práctica fundamental del desarrollo de software moderno. Este laboratorio tiene como propósito familiarizar con la automatización de tareas esenciales como la instalación de dependencias, la ejecución de pruebas unitarias y la verificación de calidad del código mediante ESLint, todo ello gestionado a través de GitHub Actions. A través de una aplicación sencilla en Node.js, se experimentará el poder de los flujos automatizados y se comprenderá la importancia de detectar errores temprano en el ciclo de vida del desarrollo.

2. Objetivos

- Configurar un flujo de integración continua (CI) en GitHub Actions que se active automáticamente con cada push o pull request a la rama principal del repositorio.
- Implementar pruebas unitarias usando Jest, garantizando que la lógica del sistema funcione correctamente en cada actualización del código.
- Aplicar análisis estático de código con ESLint, reforzando buenas prácticas de programación y detección temprana de errores o inconsistencias.
- Simular un proceso de despliegue automatizado, demostrando cómo se automatizan las etapas previas al paso final de entrega continua (CD), aún sin depender de un proveedor de hosting.

3. Desarrollo

3.1. PARTE 1: Establecimiento de la estructura del proyecto base

Paso 1: Creación de la estructura básica.
Se inicializó el proyecto y se verificó la creación de la estructura de carpetas necesaria, tal como se ilustra en la Figura 1.

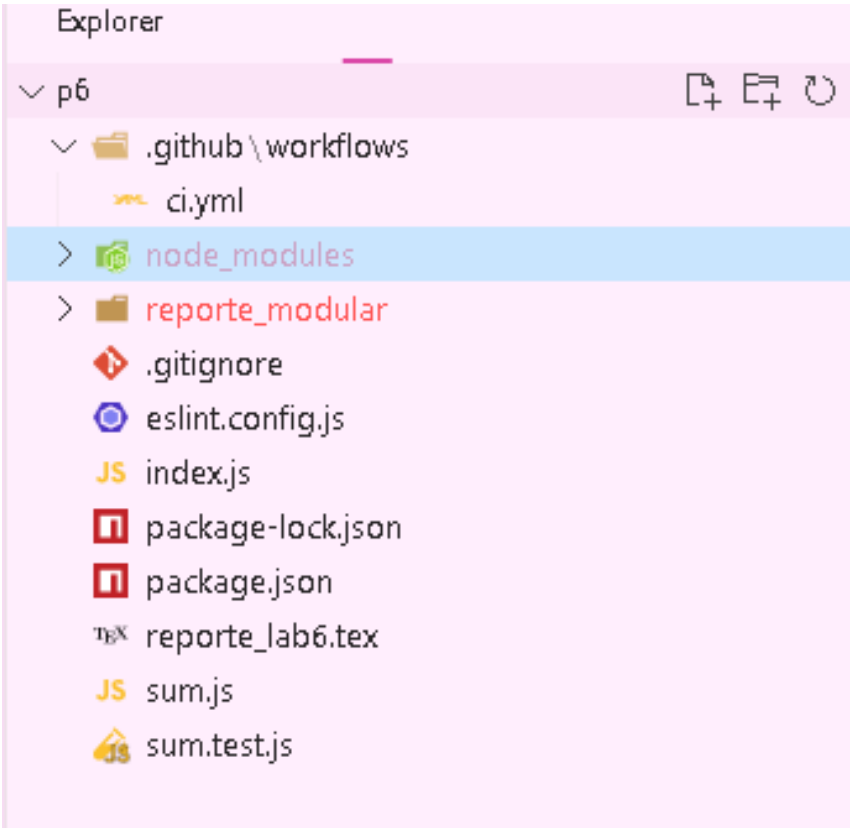

Figura 1: Estructura de archivos inicial del proyecto

Paso 2: Instalación de dependencias necesarias.

a. Se creó el archivo `package.json` mediante el comando `npm init -y`.

b. Se instaló la dependencia de Express: `npm install express`.

c. Se instalaron las dependencias de desarrollo para pruebas y calidad de código: `npm install --save-dev jest eslint`, proceso que se detalla en el Listado 1.

```
npm install express
npm install --save-dev jest eslint
```

Listing 1: Instalacion de dependencias

3.2. PARTE 2: Creación de archivos base

Paso 1: Crear archivo index.js.

a. Se configuró el servidor Express básico.

b. Se implementó un endpoint `GET /` que responde con un mensaje de bienvenida.

c. Se levantó el servidor en el puerto 3111, como se evidencia en la Figura 2. El código implementado se presenta en el Listado 2.

Figura 2: Servidor levantado y respondiendo en el puerto configurado

```
import express from 'express';
const app = express();
const port = 3111;

app.get('/', (req, res) => {
  res.send('CI/CD Lab - GitHub Actions');
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

Listing 2: index.js

Paso 2: Crear archivo sum.js.

a. Se implementó la función `sum` para realizar sumas aritméticas.

b. Se exportó la función para su uso en módulos externos, según se observa en el Listado 3.

```
export function sum(a, b) {
    return a + b;
}

export function multiply(a, b) {
    return a * b;
}
```

Listing 3: sum.js

Paso 3: Crear archivo sum.test.js.

a. Se importó la función de suma en el archivo de pruebas.

b. Se crearon casos de prueba unitarios utilizando el framework Jest, como se muestra en el Listado 4.

```
import { sum, multiply } from './sum.js';

test('adds 1 + 2 to equal 3', () => {
    expect(sum(1, 2)).toBe(3);
});

test('multiplies 2 * 3 to equal 6', () => {
    expect(multiply(2, 3)).toBe(6);
});
```

Listing 4: sum.test.js

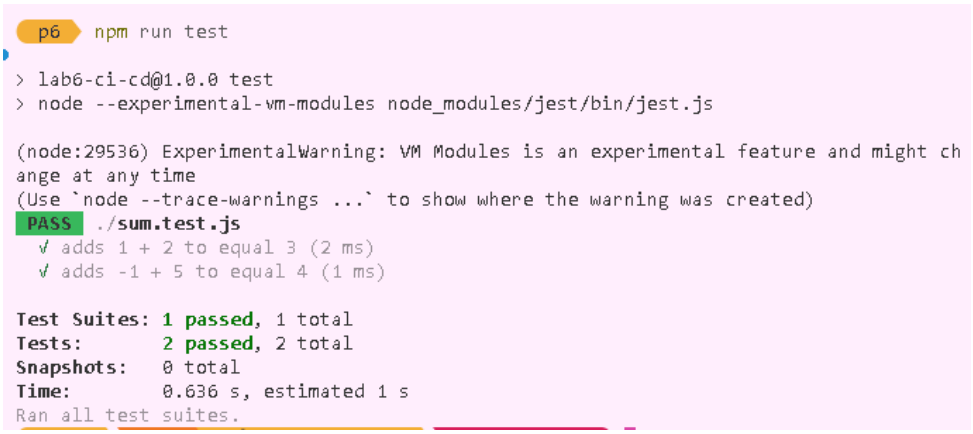

Figura 3: Ejecución de pruebas unitarias locales exitosa

En la Figura 3 se puede verificar que las pruebas locales corrieron de forma satisfactoria.

Paso 4: Configurar package.json.

a. Se agregaron los scripts para `start`, `test` y `lint`.

b. Se añadió `"type": "module"`, como se visualiza en la Figura 4.

Paso 5: Crear el archivo ESLint.
Se configuró ESLint para el análisis estático de código, cuya configuración se aprecia en la Figura 5.

Paso 6: Ignorar node_modules.
Se creó el archivo `.gitignore`, el cual se presenta en la Figura 6.

3.3. PARTE 3: Configuración de Git

Paso 1: Crear repositorio en la cuenta de Git.

a. Se accedió a GitHub y se creó un repositorio público vacío.

Paso 2: Ejecución de comandos para clonar al repositorio. Se inicializó el repositorio local y se vinculó con el remoto en GitHub.

Paso 3: Crear el workflow de GitHub Actions.

a. Se creó el archivo `.github/workflows/ci.yml`, presentado en el Listado 5.

b. Se configuraron los disparadores y jobs definidos en el manual.

Figura 4: Configuracion de scripts y tipo de modulo en package.json

```yaml
name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [18.x, 20.x, 22.x]
    steps:
    - uses: actions/checkout@v4
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm install
    - run: npm run lint
    - run: npm test
```

Listing 5: .github/workflows/ci.yml

La correcta disposición del archivo de flujo de trabajo se ilustra en la Figura 7.

Paso 4: Probar la CI.

a. Se realizó un cambio intencional en la lógica del servidor para validar el disparo del *workflow*.

Figura 5: Archivo de configuracion de ESLint

b. La plataforma GitHub Actions detectó automáticamente el evento de *push*, iniciando los contenedores de ejecución para las versiones de Node.js 18, 20 y 22.

c. Los resultados confirmaron la correcta configuración de los disparadores (*triggers*) y la ejecución secuencial de los *scripts* de instalación, linting y pruebas unitarias, como se verifica en la Figura 8.

3.4. SECCIÓN DE PREGUNTAS/ACTIVIDADES

- **Agregar más pruebas unitarias**
  - Agregar al menos 2 funciones nuevas (por ejemplo, factorial, fibonacci) en un archivo `math.js`.
  - Crear su correspondiente archivo `math.test.js` con pruebas Jest.

Figura 6: Archivo .gitignore para excluir dependencias

- Asegurarse de que GitHub Actions ejecute todas las pruebas con éxito.

```
export function factorial(n) {
    if (n < 0) return undefined;
    if (n === 0 || n === 1) return 1;
    let result = 1;
    for (let i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}

export function fibonacci(n) {
    if (n < 0) return undefined;
    if (n === 0) return 0;
    if (n === 1) return 1;
    let a = 0, b = 1;
    for (let i = 2; i <= n; i++) {
        let temp = a + b;
        a = b;
        b = temp;
    }
    return b;
}
```

Listing 6: math.js

La implementación de las funciones mencionadas se detalla en el Listado 6. Posteriormente, se generó el archivo de pruebas mostrado en el Listado 7.

```
import { factorial, fibonacci } from './math.js';

describe('Math functions', () => {
    test('factorial of 5 should be 120', () => {
        expect(factorial(5)).toBe(120);
    });

    test('fibonacci of 6 should be 8', () => {
        expect(fibonacci(6)).toBe(8);
    });
});
```

Listing 7: math.test.js

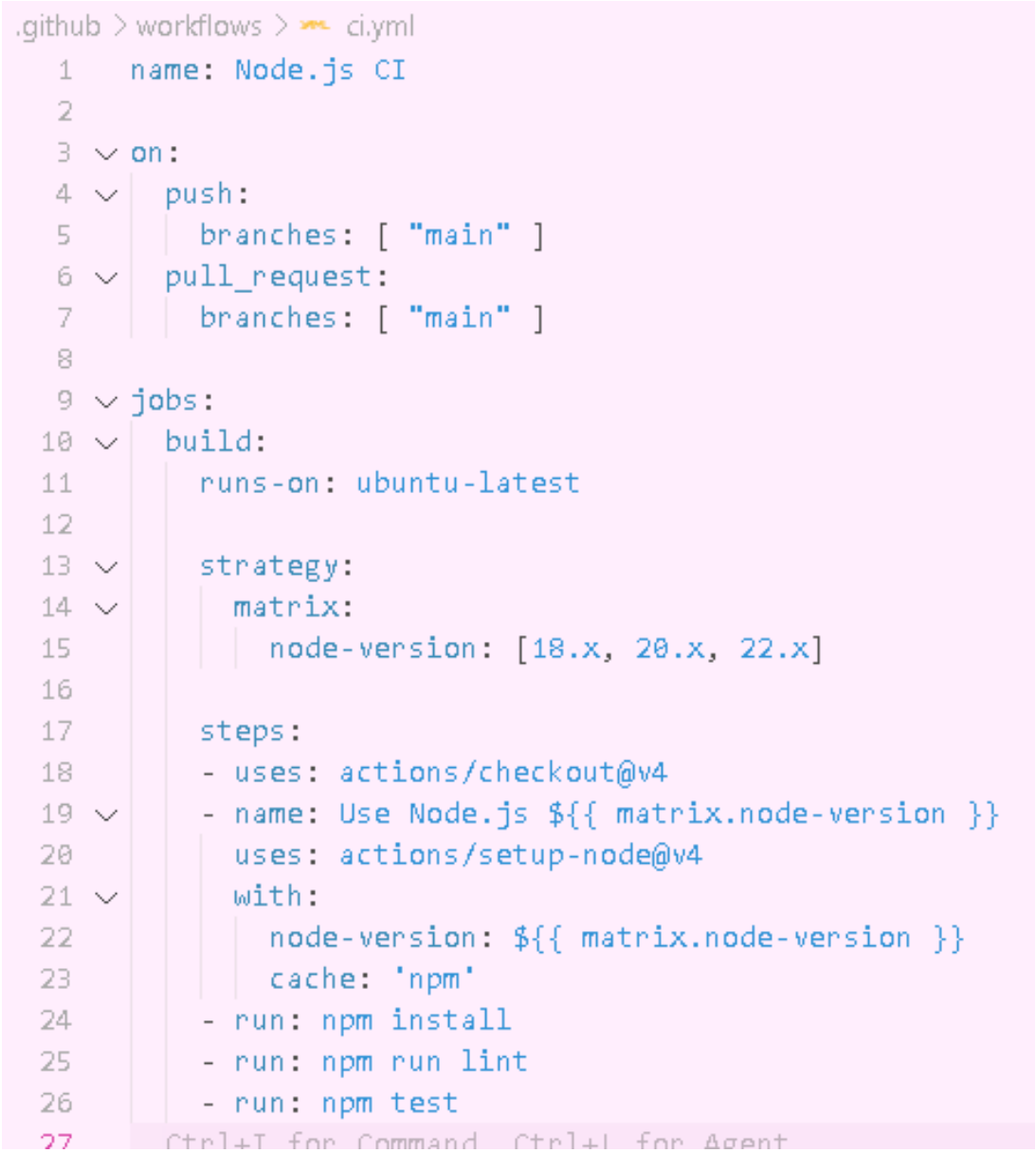

Figura 7: Configuracion del Workflow de GitHub Actions

- **Provocar un error intencional y corregirlo**
  - Modificar cualquier función o el test de alguna de ellas para que falle intencionalmente.
  - Subir los cambios y verificar que el flujo CI falla.

Como se visualiza en la Figura 9, el flujo de CI se detuvo al detectar el error provocado.

  - Corregir el error y volver a subir.

Finalmente, tras la corrección, la Figura 10 muestra el historial de ejecuciones exitosas, confirmando la resolución satisfactoria del inconveniente.

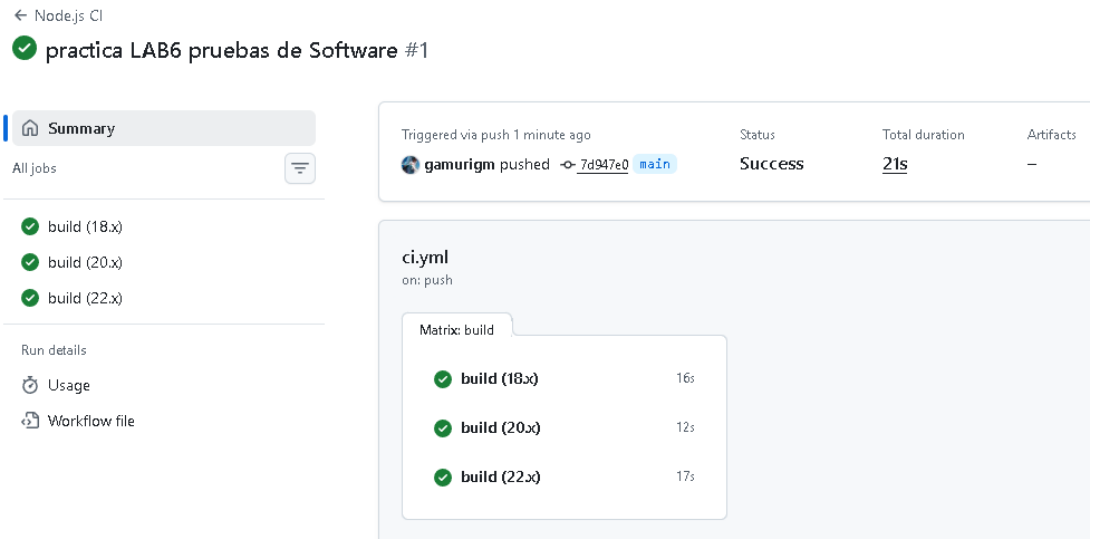

Figura 8: Verificación de la ejecución de la Integración Continua

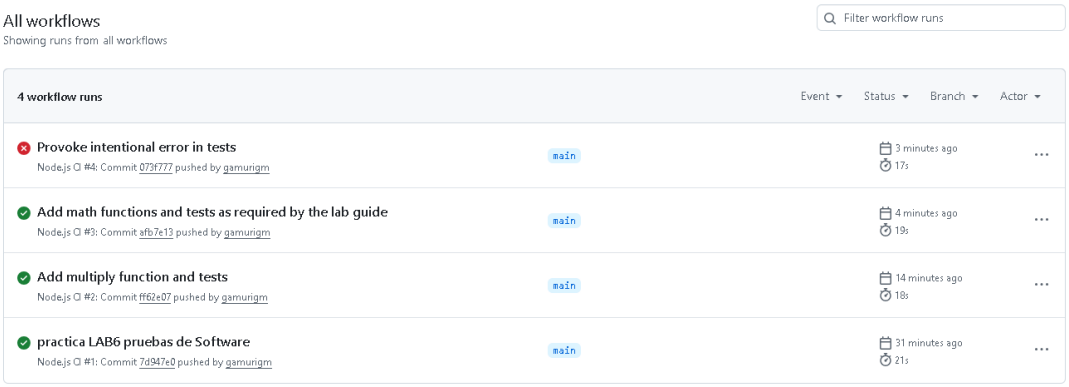

Figura 9: Fallo intencional detectado por GitHub Actions

4. Resultados y Evidencias

Los resultados obtenidos demuestran la efectividad de la implementación de un pipeline de Integración Continua. Se logró mitigar el riesgo de integrar código con errores de lógica o de estilo mediante la automatización de Jest y ESLint. Las evidencias presentadas en la sección de Desarrollo confirman que el sistema es capaz de detectar fallos en tiempo real, garantizando la estabilidad de la rama principal (*main*) ante cualquier modificación.

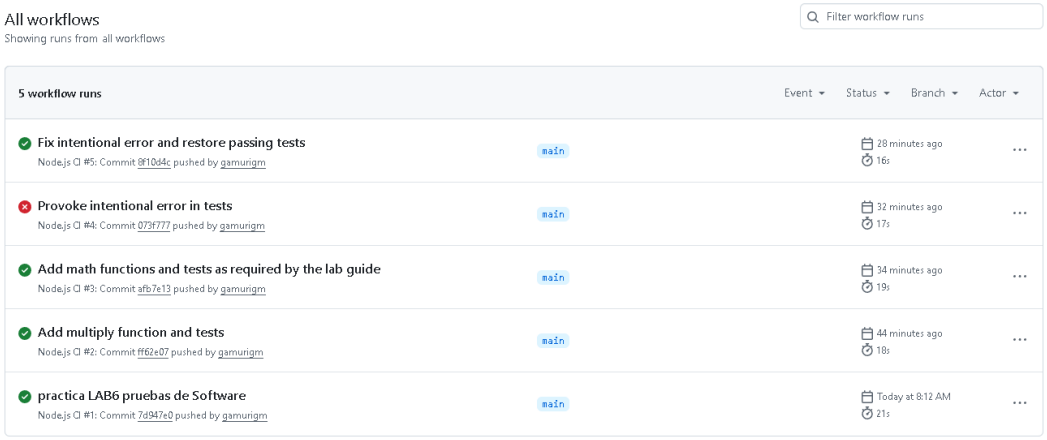

Figura 10: Corrección del error y ejecución exitosa de todas las pruebas (Estado Final)

5. Conclusiones

1. A pesar de la transparencia del flujo de CI, se identificaron desafíos técnicos críticos en la configuración inicial de las reglas de *ESLint*, los cuales generaban falsos negativos. La resolución de estos conflictos mediante la calibración de archivos de exclusión (*.eslintignore*) evidenció la importancia de alinear el análisis estático con la arquitectura del proyecto para evitar cuellos de botella en la integración.

2. La gestión de versiones de Node.js en ambientes heterogéneos planteó problemas de paridad entre el entorno local y los contenedores de GitHub Actions. Sin embargo, la implementación de una matriz de estrategia (*matrix strategy*) en el flujo de trabajo permitió subsanar estas discrepancias, garantizando que el comportamiento del software sea consistente y resiliente a través de múltiples *run-times*.

6. Recomendaciones

1. Para solventar posibles inconsistencias en la validación de dependencias, se recomienda migrar hacia el uso de comandos de instalación estrictos como `npm ci`, lo cual garantiza que el entorno de CI replique de forma idéntica el árbol de dependencias definido en el archivo `package-lock.json`, evitando errores por actualizaciones imprevistas de bibliotecas externas.

2. Ante el riesgo de fugas de credenciales en proyectos escalables, se sugiere la integración de herramientas de escaneo de secretos (*Secret Scanning*) adicionales al flujo de CI. Esto permitiría actuar no solo sobre los problemas de lógica detectados por Jest, sino también sobre vulnerabilidades de infraestructura, robusteciendo la seguridad integral del ciclo de desarrollo.