TALLER PRÁCTICO

Construyendo un Pipeline CI/CD

Modalidad	Equipos de 3-4 personas
Herramientas	GitHub + GitHub Actions (o GitLab CI)

Objetivo del Taller

Implementar un pipeline CI/CD funcional que automatice el proceso de integración y despliegue de una aplicación web simple. Al finalizar, cada equipo tendrá un repositorio con:

- Integración continua con pruebas automatizadas
- Análisis de calidad de código
- Despliegue automático a un ambiente de staging

Requisitos Previos

- Cuenta de GitHub (gratuita)
- Git instalado en la máquina local
- Editor de código (VS Code recomendado)
- Node.js instalado (versión 18 o superior)

Fase 1: Configuración del Repositorio

1.1 Crear el Proyecto Base

Cada equipo debe crear un nuevo repositorio en GitHub y configurar una aplicación Node.js básica.

Pasos:

1. Un miembro del equipo crea un repositorio público en GitHub
2. Clona el repositorio localmente
3. Inicializa el proyecto con el siguiente comando:

```
npm init -y
npm install express --save
npm install jest supertest --save-dev
```

Estructura del Proyecto:

```
mi-proyecto/
├── src/
│   └── app.js
├── tests/
│   └── app.test.js
├── .github/
│   └── workflows/
│       └── ci.yml
├── package.json
└── README.md
```

1.2 Crear la Aplicación

Crear el archivo src/app.js con el siguiente contenido:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.json({ message: 'Hola, DevOps!' });
});

app.get('/health', (req, res) => {
  res.json({ status: 'OK', timestamp: new Date() });
});

module.exports = app;

if (require.main === module) {
  const PORT = process.env.PORT || 3000;
  app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
  });
}
```

1.3 Configurar el Primer Workflow de CI

Crear el archivo .github/workflows/ci.yml:

```yaml
name: CI Pipeline

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

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm ci
      - name: Build check
        run: echo 'Build successful!'
```

Tip: Haz commit y push después de cada cambio significativo para ver el pipeline ejecutarse en GitHub Actions.

Fase 2: Pruebas Automatizadas

2.1 Escribir las Pruebas

Crear el archivo tests/app.test.js:

```
const request = require('supertest');
const app = require('../src/app');

describe('API Endpoints', () => {
  test('GET / should return welcome message', async () => {
    const response = await request(app).get('/');
    expect(response.status).toBe(200);
    expect(response.body.message).toBe('Hola, DevOps!');
  });

  test('GET /health should return OK status', async () => {
    const response = await request(app).get('/health');
    expect(response.status).toBe(200);
    expect(response.body.status).toBe('OK');
  });
});
```

2.2 Configurar Scripts en package.json

Agregar los siguientes scripts:

```
"scripts": {
  "start": "node src/app.js",
  "test": "jest --coverage",
  "test:watch": "jest --watch"
}
```

2.3 Actualizar el Workflow

Modificar ci.yml para incluir las pruebas:

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm ci
      - name: Run tests
        run: npm test
      - name: Upload coverage
        uses: codecov/codecov-action@v3
        if: always()
```

Ejercicio:

Agrega un nuevo endpoint GET /version que retorne { version: '1.0.0' } y escribe la prueba correspondiente. Verifica que el pipeline pase con todos los tests.

Fase 3: Despliegue Automático

3.1 Configurar Despliegue a GitHub Pages

Para simular un despliegue, crearemos una página de status. Agregar al workflow:

```
deploy:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'
  steps:
    - uses: actions/checkout@v4
    - name: Create status page
      run: |
        mkdir -p public
        echo '<html><body>' > public/index.html
        echo '<h1>Deploy Status</h1>' >> public/index.html
        echo '<p>Last deploy: '$(date)'</p>' >> public/index.html
        echo '<p>Commit: '$GITHUB_SHA'</p>' >> public/index.html
        echo '</body></html>' >> public/index.html
    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./public
```

3.2 Habilitar GitHub Pages

1. Ir a Settings → Pages en tu repositorio
2. En Source, seleccionar 'Deploy from a branch'
3. Seleccionar la rama 'gh-pages'
4. Tu sitio estará disponible en: https://[usuario].github.io/[repo]/

Cierre: Presentación de Resultados

Cada equipo debe presentar brevemente:

1. Demostrar el pipeline funcionando (hacer un commit y ver la ejecución)
2. Mostrar las pruebas pasando en GitHub Actions
3. Mostrar el sitio desplegado en GitHub Pages
4. Compartir un desafío que enfrentaron y cómo lo resolvieron

Rúbrica de Evaluación

Criterio	Puntos	Obtenido
Pipeline CI ejecuta correctamente	20	
Pruebas automatizadas pasan (mínimo 3)	25	
Despliegue automático funciona	25	
Endpoint adicional implementado	15	
Presentación clara del equipo	15	
TOTAL	**100**	

Recursos Adicionales

- GitHub Actions Documentation: docs.github.com/en/actions
- Jest Testing Framework: jestjs.io
- Express.js Guide: expressjs.com
- DevOps Roadmap: roadmap.sh/devops