Práctica: Armado de un Pipeline CI/CD en GitHub

Objetivo

Esta práctica tiene como objetivo principal comprender y aplicar los conceptos de **Continuous Integration (CI)** y **Continuous Deployment (CD)** mediante la construcción de un pipeline en **GitHub Actions**. Se trabajará con las funcionalidades nativas de GitHub para automatizar pruebas y crear releases.

Pre-requisitos

- Conocimientos básicos de Git y GitHub.
- Conocimientos básicos de Python.
- Familiaridad con el concepto de unit tests.
- Acceso a una cuenta de GitHub.

Materiales y Herramientas

- Cuenta de GitHub.
- Editor de texto/IDE (VS Code, PyCharm, etc.).
- Terminal/Línea de comandos.

---

1. Introducción a GitHub Actions y Pipelines CI/CD

GitHub Actions ofrece una plataforma flexible para automatizar flujos de trabajo directamente desde el repositorio. Los pipelines CI/CD por defecto son plantillas preconfiguradas que facilitan la implementación de la automatización.

Paso 1: Explorar Workflows Existentes

1. Crear un nuevo repositorio en GitHub (o utilizar uno existente para la práctica).
2. Navegar a la pestaña "Actions" en tu repositorio.
3. Observar las sugerencias de workflows que GitHub ofrece basadas en el tipo de código del repositorio (por ejemplo, Python, Node.js, etc.).

4. Seleccionar uno de los workflows sugeridos (por ejemplo, para Python) y revisar su estructura básica. No hay que configurarlo todavía, solo analizar el archivo `main.yml` que se genera.

Analizar y responder:

- ¿Qué elementos clave se identifican en la estructura de un archivo de workflow de GitHub Actions?
- ¿Qué eventos disparan estos workflows por defecto?
- ¿Qué tareas o "jobs" suelen incluir los workflows de CI por defecto?

Nota: Las ejecuciones de GitHub Actions en repositorios públicos y privados consumen minutos de la cuenta de GitHub. A los objetivos de la práctica alcanza con el Free Tier.

---

2. Construcción de un Pipeline de Continuous Integration (CI) con Pruebas Automatizadas

En esta sección, se detalla el procedimiento para configurar un pipeline de CI que se encargará de ejecutar pruebas unitarias cada vez que se realice un push o se abra un pull request a la rama principal.

Prepacion:
- Crear un repositorio nuevo y vacío en GitHub
- Clonar el repositorio al entorno local.

Paso 1: Preparar el Código para las Pruebas

1. En el repositorio local, crear un archivo `app.py` con el siguiente contenido:

```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b
```

2. Een el mismo directorio, crear un archivo `test_app.py` para las pruebas unitarias usando `unittest`:

```python
import unittest
from app import add, subtract, multiply

class TestApp(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(1, 2), 3)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(-1, -1), -2)

    def test_subtract(self):
        self.assertEqual(subtract(5, 3), 2)
        self.assertEqual(subtract(3, 5), -2)
        self.assertEqual(subtract(10, 0), 10)

    def test_multiply(self):
        self.assertEqual(multiply(2, 3), 6)
        self.assertEqual(multiply(-1, 5), -5)
        self.assertEqual(multiply(0, 10), 0)

if __name__ == '__main__':
    unittest.main()
```

3. Correr los tests y verificar que funcionen como corresponde: `python -m unittest test_app.py`
4. Añadir y commitear ambos archivos.

Paso 2: Configurar el Workflow de CI

1. En el repositorio local, crear el directorio `.github/workflows`.
2. Dentro de `workflows`, crear un nuevo archivo llamado `ci_pipeline.yml` con el siguiente contenido:

```
name: CI Pipeline

on:
```

```
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4
    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.x'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        # No hay dependencias específicas para este ejemplo, pero se incluiría
aquí
    - name: Run unit tests
      run: python -m unittest test_app.py
```

3. Añadir el archivo `ci_pipeline.yml` al control de versiones y hacer commit.
4. Hacer `git push` al repositorio remoto.

Paso 3: Verificar la Ejecución del CI en el push

1. Navegar a la pestaña "Actions" en el repositorio de GitHub.
2. Se debería ver una nueva ejecución del workflow "CI Pipeline".
3. Hacer clic en la ejecución para ver los detalles y verificar que todas las etapas (Checkout, Set up Python, Install dependencies, Run unit tests) se hayan completado exitosamente.

Paso 4: Verificar la Ejecución del CI en un Pull Request

1. Crear un nuevo branch.
2. Introducir un error intencional: Hacer que subtract retorne el resultado de una suma en lugar de una resta
3. Commitear los cambios y pushear el nuevo branch.
4. Crear un PR para mergear el nuevo branch a main.
5. Revisar que en el PR aparezca el estado del CI corriendo los tests. Debería reportar la falla de los mismos.

6. Corregir el error introducido, commitear los cambios y pushear el nuevo branch.
7. En la pantalla del PR se debería ver que el pipeline corre de vuelta, esta vez sin reportar fallas.

---

3. Construcción de un Pipeline de Continuous Deployment (CD) para Creación de Releases

En esta sección, se describen los pasos para configurar un pipeline de CD que cree automáticamente un nuevo "Release" en GitHub cada vez que se cree un tag en el repositorio. Esto simula la entrega de una nueva versión de una aplicación.

Paso 1: Configurar el Workflow de CD

1. Dentro del directorio `.github/workflows`, crear un nuevo archivo llamado `cd_release.yml` con el siguiente contenido:

```
None
name: CD Release Pipeline

on:
  create:
    tags:
      - 'v*.*.*' # Trigger cuando se crea un tag que comienza con 'v' (ej.
v1.0.0)

jobs:
  create-github-release:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Create Release
      id: create_release
      uses: actions/create-release@v1
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Token automático para
interactuar con la API de GitHub
      with:
        tag_name: ${{ github.ref_name }}
```

```
        release_name: Release ${{ github.ref_name }}
        body: |
          Nueva versión liberada: ${{ github.ref }}
          Este release fue generado automáticamente por el pipeline de CD.
        draft: false
        prerelease: false
```

2. Añadir el archivo `cd_release.yml` al control de versiones y hacer commit.
3. Hacer `git push` al repositorio remoto.

Paso 2: Probar la Creación del Release

1. Para disparar el workflow de CD, es necesario crear un tag en el repositorio y luego pushearlo:

```
None
git tag -a v1.0.0 -m "Primera versión estable"
git push origin v1.0.0
```

2. Navegar a la pestaña "Actions" en el repositorio de GitHub. Se debería ver una nueva ejecución del workflow "CD Release Pipeline".
3. Verificar que el workflow se haya completado exitosamente.
4. Navegar a la pestaña "Releases" en el repositorio de GitHub. Se debería ver el nuevo release "Release v1.0.0" creado automáticamente.

Analizar y responder:

- ¿Qué diferencias se observan entre los eventos que disparan el pipeline de CI y el de CD?
- ¿Qué ventajas ofrece la automatización de la creación de releases?
- ¿Cómo se podría extender este pipeline de CD para desplegar la aplicación en un entorno de producción?
- ¿Cuál es la función del GITHUB_TOKEN?
- ¿Cómo sería una forma segura de pasar configuraciones y contraseñas al pipeline?

Conclusiones y Próximos Pasos

En esta práctica se realizó un pipeline CI/CD básico usando GitHub Actions. Se automatizó la ejecución de pruebas y la creación de releases, sentando las bases para flujos de trabajo más complejos.

Posibles Extensiones (Opcional):

- **Notificaciones:** Integrar notificaciones (Slack, Email) al finalizar los workflows. El marketplace puede tener algo que sirva para esto.
- **Cobertura de Código:** Añadir un paso para calcular y reportar la cobertura de código de las pruebas.
- **Deployment a PaaS/IaaS:** Investigar cómo sería un despliegue de la misma aplicación (aunque sea simple) en un servicio como Heroku, Python anywhere, o un servidor básico usando un workflow de CD más avanzado.
- **Containers:** Empaquetar la aplicación en un contenedor Docker y modificar los pipelines para construir y desplegar la imagen.

Referencias

- **GitHub Actions Documentation:** https://docs.github.com/en/actions
- **Quickstart de GitHub Actions**: https://docs.github.com/en/actions/get-started/quickstart
- **Understanding GitHub Actions:** https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
- **Workflow syntax for GitHub Actions:** https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions
- **Building and testing Python:** https://docs.github.com/en/actions/tutorials/build-and-test-code/python
- **About GitHub-hosted runners:** https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners (Para entender los entornos de ejecución)
- **Eventos que disparan workflows:** https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows
- **Información del free tier de GitHub**
    - https://docs.github.com/get-started/learning-about-github/githubs-products
    - https://docs.github.com/en/billing/concepts/product-billing/github-actions#free-use-of-github-actions