INFORME TÉCNICO

# Implementación de CI/CD para Aplicación Web Flask con GitHub Actions y Render

**Proyecto:** Flask Todo Application  
**Autor:** joe-696  
**Fecha:** 06 de July, 2025  
**Repositorio:** https://github.com/joe-696/flask-todo-app  
**URL Producción:** <https://flask-todo-app-joe.onrender.com>

Dashboard render: <https://dashboard.render.com/web/srv-d1lbr9adbo4c739qh7l0/deploys/dep-d1lbr9idbo4c739qh7q0?r=2025-07-06%4018%3A16%3A11%7E2025-07-06%4018%3A19%3A10>

# ÍNDICE

1. 1. RESUMEN EJECUTIVO
2. 2. INTRODUCCIÓN
3. 3. OBJETIVOS
4. 4. MARCO TEÓRICO
5. 4.1. CI/CD (Continuous Integration/Continuous Deployment)
6. 4.2. GitHub Actions
7. 4.3. Flask Framework
8. 4.4. Testing Automatizado
9. 5. METODOLOGÍA
10. 6. DESARROLLO DEL PROYECTO
11. 6.1. Estructura de la Aplicación Flask
12. 6.2. Configuración del Repositorio Git
13. 6.3. Implementación de Tests Automáticos
14. 6.4. Configuración de GitHub Actions
15. 6.5. Despliegue en Render
16. 7. ARQUITECTURA DEL SISTEMA
17. 8. FLUJO CI/CD IMPLEMENTADO
18. 9. RESULTADOS OBTENIDOS
19. 10. CAPTURAS DE PANTALLA
20. 11. CONCLUSIONES
21. 12. RECOMENDACIONES
22. 13. BIBLIOGRAFÍA
23. 14. ANEXOS

# 1. RESUMEN EJECUTIVO

Este informe documenta la implementación exitosa de un pipeline de CI/CD (Continuous Integration/Continuous Deployment) para una aplicación web desarrollada con Flask. El proyecto abarca desde la creación de una aplicación CRUD simple hasta la implementación de un sistema automatizado de integración y despliegue continuo utilizando GitHub Actions y Render.

Los resultados obtenidos incluyen: una aplicación web funcional con operaciones CRUD completas, tests automáticos que validan la funcionalidad, un pipeline de CI/CD que ejecuta automáticamente en cada cambio de código, y despliegue automático a producción con validación previa.

# 2. INTRODUCCIÓN

En el desarrollo de software moderno, la implementación de prácticas DevOps como CI/CD es fundamental para mantener la calidad del código y acelerar los ciclos de desarrollo. Este proyecto demuestra la implementación práctica de estas metodologías en una aplicación real.

La aplicación desarrollada es un sistema de gestión de tareas (Todo List) que permite a los usuarios crear, leer, actualizar y eliminar tareas. Aunque simple en concepto, sirve como base perfecta para demostrar conceptos avanzados de CI/CD.

# 3. OBJETIVOS

## 3.1. Objetivo General

Implementar un sistema completo de CI/CD para una aplicación web Flask, incluyendo tests automáticos, validación de código y despliegue automático a producción.

## 3.2. Objetivos Específicos

• Desarrollar una aplicación web CRUD funcional utilizando Flask

• Implementar tests automáticos con pytest

• Configurar GitHub Actions para CI/CD

• Establecer despliegue automático en Render

• Documentar todo el proceso de implementación

• Validar el funcionamiento del pipeline completo

# 4. MARCO TEÓRICO

## 4.1. CI/CD (Continuous Integration/Continuous Deployment)

CI/CD es una metodología de desarrollo que permite a los equipos entregar cambios de código de forma rápida y confiable. La Integración Continua (CI) implica la fusión frecuente de cambios de código en un repositorio central, mientras que el Despliegue Continuo (CD) automatiza la entrega de aplicaciones a entornos de producción.

## 4.2. GitHub Actions

GitHub Actions es una plataforma de CI/CD que permite automatizar workflows directamente desde el repositorio de GitHub. Utiliza archivos YAML para definir pipelines que se ejecutan en respuesta a eventos como commits, pull requests, o schedules.

## 4.3. Flask Framework

Flask es un micro-framework web para Python que proporciona las herramientas básicas para construir aplicaciones web. Es minimalista pero extensible, lo que lo hace ideal para proyectos que requieren flexibilidad y control granular.

## 4.4. Testing Automatizado

Los tests automáticos son scripts que verifican el comportamiento esperado del código. En Python, pytest es una de las herramientas más populares para escribir y ejecutar tests, proporcionando funcionalidades avanzadas como fixtures y parametrización.

# 5. METODOLOGÍA

El desarrollo del proyecto siguió una metodología incremental, implementando funcionalidades en el siguiente orden:

1. Análisis de requerimientos y diseño de la aplicación

2. Desarrollo de la aplicación Flask básica

3. Implementación de operaciones CRUD

4. Creación de templates HTML con Bootstrap

5. Configuración del entorno de desarrollo

6. Implementación de tests automáticos

7. Configuración de GitHub Actions

8. Despliegue en plataforma Render

9. Validación del pipeline CI/CD completo

10. Documentación y análisis de resultados

# 6. DESARROLLO DEL PROYECTO

## 6.1. Estructura de la Aplicación Flask

La aplicación fue estructurada siguiendo las mejores prácticas de Flask:

***proyecto/  
├── app.py # Aplicación principal  
├── requirements.txt # Dependencias  
├── Procfile # Configuración Render  
├── .gitignore # Archivos ignorados por Git  
├── templates/ # Plantillas HTML  
│ ├── base.html # Template base  
│ ├── index.html # Lista de tareas  
│ ├── add\_todo.html # Agregar tarea  
│ └── edit\_todo.html # Editar tarea  
├── tests/ # Tests automáticos  
│ ├── \_\_init\_\_.py   
│ └── test\_app.py # Tests principales  
└── .github/ # GitHub Actions  
 └── workflows/  
 └── ci-cd.yml # Pipeline CI/CD***

## 6.2. Configuración del Repositorio Git

Se configuró un repositorio Git con las siguientes características:

• Inicialización del repositorio local

• Configuración de credenciales de usuario

• Creación de .gitignore apropiado

• Conexión con repositorio remoto en GitHub

• Configuración de token de acceso personal con permisos de workflow

## 6.3. Implementación de Tests Automáticos

Se implementaron tests comprehensivos que cubren:

• Test de carga de página principal

• Test de funcionalidad de creación de tareas

• Test de validación de campos obligatorios

• Test de edición de tareas existentes

• Test de eliminación de tareas

• Test de cambio de estado (completado/pendiente)

• Test del modelo de datos Todo

## 6.4. Configuración de GitHub Actions

El workflow de GitHub Actions incluye dos jobs principales:

**Job de Testing:**

• Checkout del código fuente

• Configuración de Python 3.11

• Instalación de dependencias

• Ejecución de tests con pytest

• Validación de startup de la aplicación

**Job de Deploy:**

• Ejecución condicional (solo en branch main)

• Dependencia del job de testing

• Notificación de despliegue automático por Render

## 6.5. Despliegue en Render

La configuración de Render incluye:

• Conexión automática con repositorio GitHub

• Detección automática de aplicación Flask

• Build command: pip install -r requirements.txt

• Start command: gunicorn app:app

• Variables de entorno para producción

• Despliegue automático en cada push a main

# 7. ARQUITECTURA DEL SISTEMA

La arquitectura implementada sigue un patrón de tres capas:

**Capa de Presentación:**

Templates HTML con Bootstrap para interfaz responsive y moderna.

**Capa de Lógica de Negocio:**

Aplicación Flask con rutas para operaciones CRUD y manejo de formularios.

**Capa de Datos:**

SQLAlchemy ORM con base de datos SQLite para desarrollo y PostgreSQL disponible para producción.

[INSERTAR DIAGRAMA DE ARQUITECTURA]

# 8. FLUJO CI/CD IMPLEMENTADO

El flujo de CI/CD implementado sigue el siguiente proceso:

1. Developer hace git push al repositorio

2. GitHub detecta cambios y activa workflow

3. GitHub Actions inicia job de testing

4. Se ejecutan todos los tests automáticos

5. Si tests pasan: continúa al deploy

6. Si tests fallan: detiene el proceso

7. Render detecta cambios en GitHub

8. Render ejecuta build automático

9. Render despliega nueva versión

10. Aplicación actualizada en producción

[INSERTAR DIAGRAMA DE FLUJO CI/CD]

# 9. RESULTADOS OBTENIDOS

**Aplicación Funcional:**

✅ Aplicación web completamente funcional  
✅ Operaciones CRUD implementadas  
✅ Interfaz responsive con Bootstrap  
✅ Validación de formularios  
✅ Mensajes de feedback al usuario

**CI/CD Automatizado:**

✅ Pipeline de GitHub Actions funcionando  
✅ Tests automáticos ejecutándose en cada push  
✅ Deploy automático a producción  
✅ Validación previa antes del despliegue  
✅ Tiempo total de pipeline: ~21 segundos

**Calidad del Código:**

✅ Cobertura de tests implementada  
✅ Validación automática de funcionalidades  
✅ Detección temprana de errores  
✅ Código versionado y documentado

# 10. CAPTURAS DE PANTALLA

Figura 1: Aplicación en funcionamiento - Página principal

[INSERTAR CAPTURA AQUÍ]

Figura 2: GitHub Actions - Workflow ejecutándose

[INSERTAR CAPTURA AQUÍ]

Figura 3: GitHub Actions - Tests exitosos

[INSERTAR CAPTURA AQUÍ]

Figura 4: Render - Dashboard de despliegue

[INSERTAR CAPTURA AQUÍ]

Figura 5: Aplicación en producción - URL live

[INSERTAR CAPTURA AQUÍ]

# 11. CONCLUSIONES

• La implementación de CI/CD mejora significativamente la calidad y confiabilidad del software desarrollado.

• GitHub Actions proporciona una plataforma robusta y accesible para implementar pipelines de CI/CD sin costo adicional.

• Los tests automáticos son fundamentales para detectar errores antes de que lleguen a producción.

• El despliegue automático reduce errores humanos y acelera el tiempo de entrega de nuevas funcionalidades.

• La combinación Flask + GitHub Actions + Render proporciona un stack completo para desarrollo web moderno.

• La documentación y versionado del código facilita el mantenimiento y colaboración en equipo.

# 12. RECOMENDACIONES

• Implementar métricas de cobertura de código para mejorar la calidad de tests

• Agregar múltiples entornos (desarrollo, staging, producción)

• Implementar notificaciones automáticas en caso de fallos

• Considerar implementar análisis estático de código (linting)

• Evaluar la migración a PostgreSQL para mayor robustez en producción

• Implementar monitoring y logging avanzado

• Considerar la implementación de blue-green deployment

# 13. BIBLIOGRAFÍA

Flask Documentation. (2025). Flask Web Development Framework. https://flask.palletsprojects.com/

GitHub Actions Documentation. (2025). Automating workflows with GitHub Actions. https://docs.github.com/actions

Render Documentation. (2025). Deploy and scale applications. https://render.com/docs

pytest Documentation. (2025). Testing framework for Python. https://docs.pytest.org/

SQLAlchemy Documentation. (2025). Python SQL toolkit. https://www.sqlalchemy.org/

Bootstrap Documentation. (2025). Frontend framework. https://getbootstrap.com/

# 14. ANEXOS

## Anexo A: Código Fuente Principal (app.py)

[Incluir código fuente completo de app.py]

## Anexo B: Archivo de Workflow GitHub Actions

[Incluir contenido completo de ci-cd.yml]

## Anexo C: Tests Implementados

[Incluir código completo de test\_app.py]

## Anexo D: Configuración de Despliegue

[Incluir Procfile y requirements.txt]