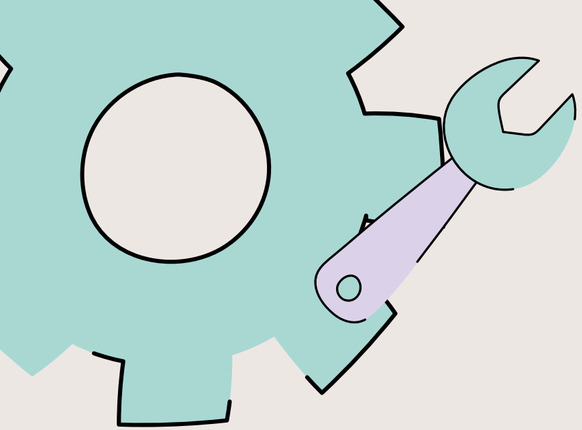
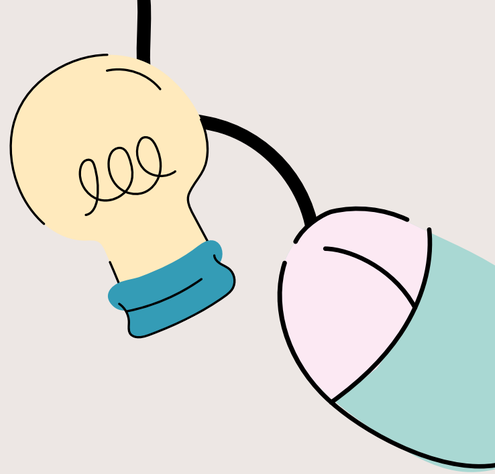
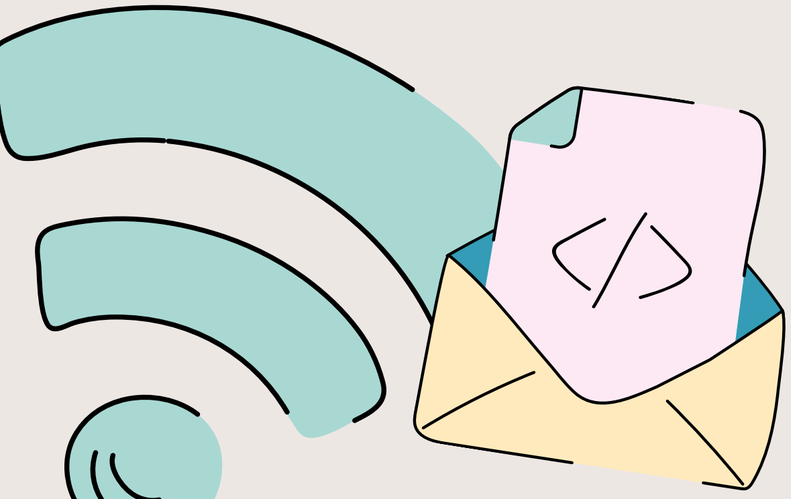


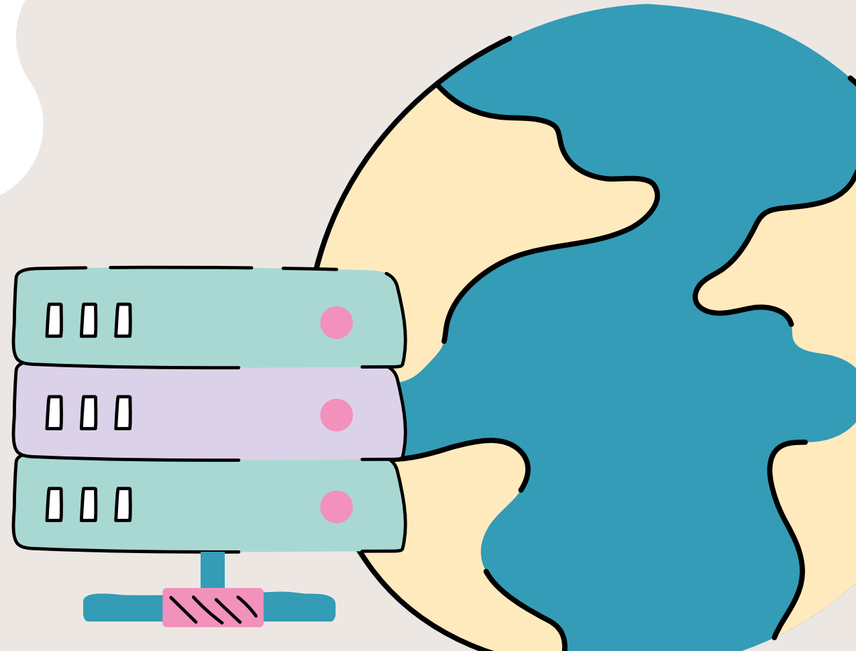


CI/CD

Automatiza tus pruebas y despliegues sin complicarte la vida

Por Juan Duran





¿Qué es CI/CD y por qué deberías usarlo?

Aunque parezcan siglas técnicas reservadas a grandes empresas o proyectos masivos, **CI/CD** es una **filosofía de trabajo** que busca que tus aplicaciones estén **siempre listas para funcionar correctamente en producción.**

- **CI** significa Integración Continua: se refiere a comprobar automáticamente que tu código funciona cada vez que haces cambios.
- **CD** puede significar Entrega Continua o Despliegue Continuo, según el nivel de automatización: el objetivo es que tus cambios lleguen a producción de forma rápida, segura y sin intervención manual.

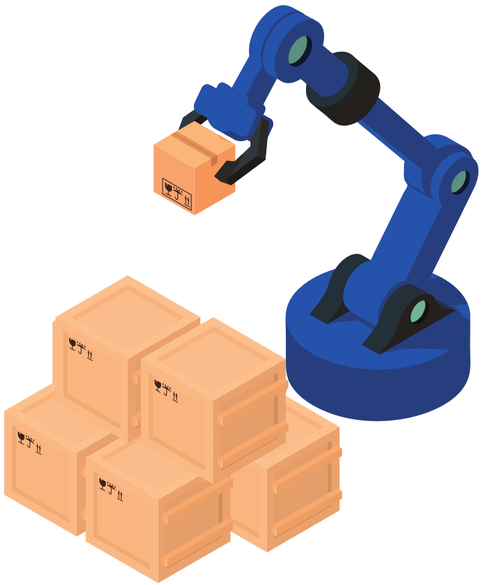
Empezar a aplicar CI/CD es dar un paso hacia un flujo de trabajo más profesional, ordenado y robusto, incluso si estás trabajando solo.

¿Qué problema soluciona CI/CD?

Todos hemos pasado por alguna de estas situaciones:

- **Subes código** nuevo que rompe algo... y te das cuenta demasiado tarde.
- Tienes que **repetir pasos manuales** cada vez que haces un cambio: probar, subir, reiniciar, verificar...
- El código **funciona en tu equipo**, pero no en otro entorno.
- No sabes exactamente qué versión está corriendo en **producción**.

CI/CD automatiza esos pasos clave: pruebas, construcción y despliegue. Así, reduces errores, ganas tiempo y mantienes un flujo de trabajo más predecible y seguro.



¿Qué es la Integración Continua (CI)?

La **Integración Continua** es la práctica de **probar tu código automáticamente** cada vez que haces un cambio. Esto se logra configurando un sistema que, al hacer push a tu repositorio:

- **Ejecuta tests** para comprobar que todo sigue funcionando.
- **Informa** si algo falla antes de que llegue a producción.
- Permite **detectar** errores **tempranamente**, cuando son más fáciles y baratos de corregir.

Es como tener un **asistente invisible** que revisa tu trabajo constantemente y no deja que los errores se acumulen. Esto se vuelve aún más importante si estás en equipo, pero incluso en proyectos personales, es un **salto de calidad enorme**.

¿Qué es el Despliegue Continuo (CD)?

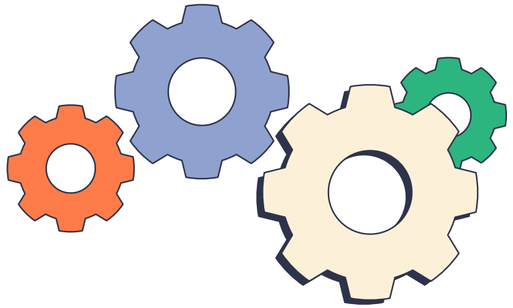
El **Despliegue Continuo** lleva la automatización un paso más allá. No solo prueba tu código, sino que lo **sube automáticamente a producción** si todo está en orden.

Esto significa que puedes hacer push a main y, sin mover un dedo más, tu app se actualiza con la nueva versión.

CD no significa que todo sea automático a ciegas: puedes configurar un **flujo de aprobación**, un entorno intermedio de testing o una publicación por etapas.

El objetivo es eliminar los cuellos de botella y los errores humanos en el proceso de despliegue, manteniendo un ritmo de entrega **rápido**, **confiable** y **controlado**.

CI/CD: de la teoría a la práctica



Una implementación básica de CI/CD incluye:

- **Repositorio de código**, como GitHub.
- **Sistema de automatización**, como GitHub Actions o GitLab CI.
- **Tests automatizados**, escritos por ti o el equipo.
- **Entorno de producción**, como Render, Railway o Heroku.

El flujo general es:

- Haces cambios y los subes al repositorio.
- Se ejecutan **tests automáticamente**.
- Si todo pasa, se **despliega la nueva versión**.

Este proceso puede tardar segundos o minutos, pero lo importante es que se hace sin **pasos manuales** ni olvidos.

Ventajas clave de aplicar CI/CD

- **Confianza en el código**: sabes que si algo rompe, te enteras al instante.
- **Menos errores en producción**: porque todo se prueba antes de desplegar.
- **Velocidad de entrega**: puedes lanzar cambios pequeños de forma continua, en lugar de grandes lanzamientos que generan estrés.
- **Documentación viva**: los flujos están codificados, no en tu cabeza ni en notas sueltas.
- **Preparación para escalar**: aunque empieces solo, ya trabajas como lo harías en un equipo profesional.

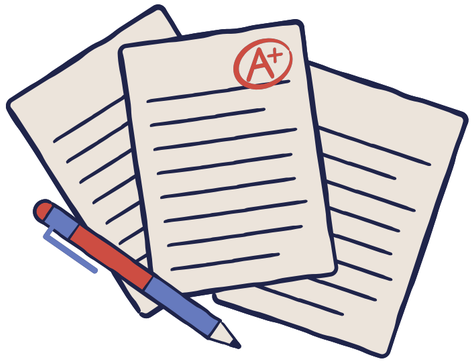
¿Cómo empiezo con CI/CD en GitHub?

Si usas **GitHub**, puedes aprovechar **GitHub Actions**, que te permite definir **workflows automatizados** mediante archivos YAML que viven dentro de tu repo.
Para empezar, solo necesitas:

- Crear una carpeta .github/workflows/.
- Crear un archivo .yml con las instrucciones que quieras automatizar.
- Definir eventos (push, pull_request) y los pasos que quieres ejecutar.

Ejemplo: instalar dependencias, correr tests, construir tu app, desplegarla.

Ejemplo de CI con tests en Python

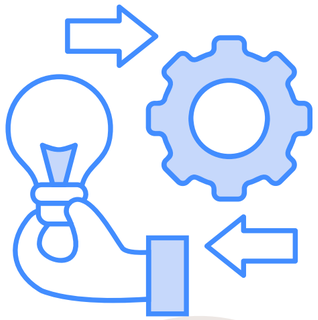


```
name: Test Python App

on: [push]

jobs:
 test:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3
   - name: Set up Python
    uses: actions/setup-python@v4
    with:
     python-version: '3.11'
   - name: Install dependencies
    run: pip install -r requirements.txt
   - name: Run tests
    run: pytest
```

Este archivo:

- Se activa al hacer push.
- Instala Python y tus dependencias.
- Lanza pytest para ejecutar tus tests.
- Es un ejemplo mínimo pero poderoso: ya tienes CI funcionando en segundos.

Ejemplo de CD: despliegue automático a Render



Puedes agregar un paso más en el mismo workflow o crear uno separado para **desplegar a una plataforma como Render.**

Normalmente se hace enviando una llamada a su API, usando una clave secreta almacenada en GitHub Secrets.

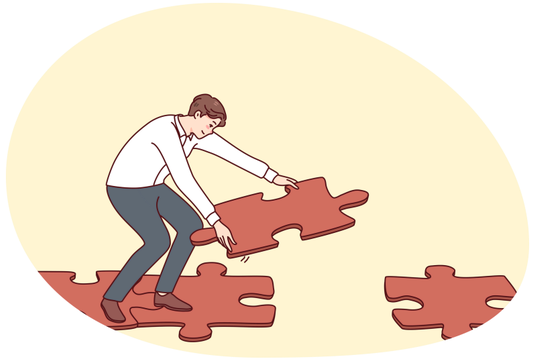
También puedes usar git push a un repositorio conectado con Heroku, Railway o PythonAnywhere.

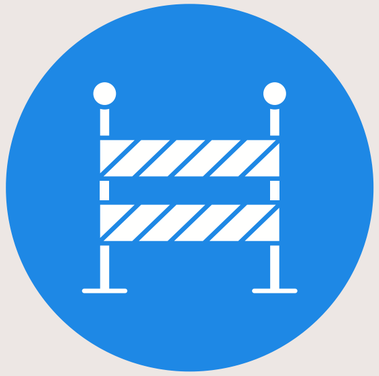
La clave es que el despliegue ocurra solo si todo lo anterior salió bien.

Testing y CI/CD: mejores prácticas

- **Empieza con tests simples**: no necesitas cobertura completa al inicio.
- **Usa ambientes separados**: uno para desarrollo, otro para producción.
- **Configura tu CI** para ejecutarse solo en ramas importantes.
- **Usa variables de entorno y secretos** para manejar claves, tokens y configuraciones sensibles.
- **Integra notificaciones** en Slack, Discord o correo para saber cuándo algo falla.

Desafíos comunes al implementar CI/CD





- **Falsos negativos** en tests mal escritos pueden bloquearte.
- El entorno de CI puede no coincidir con el tuyo local: usa contenedores si es posible.
- Los despliegues automáticos mal configurados pueden sobrescribir versiones estables.
- Puede haber una curva de aprendizaje con YAML y flujos complejos.

Pero lo importante es empezar pequeño: un test, un flujo de CI básico, y luego ir mejorando.

Conclusiones

Automatización

Incorporar CI/CD temprano evita errores y te ahorra tiempo más adelante.

Impacto

Con un solo test automatizado ya mejoras tu flujo de trabajo.

Validación

Cada push es revisado automáticamente. Confianza sin esfuerzo.

De local a producción

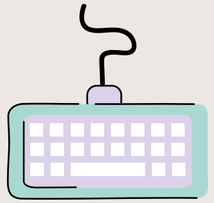
Despliega sin miedo ni pasos manuales. Todo queda controlado.

Profesionalización

Aunque trabajes solo, CI/CD te hace trabajar como un equipo serio.

Menos errores

Te olvidas de tareas repetitivas y te concentras en mejorar tu app.



Gracias

Por Juan Duran

“Coding, Gaming and Leveling Up”