CI/CD Pipeline Process for Python with GitHub and Render

Overview

This document outlines a CI/CD pipeline for deploying a Python-based web application (e.g., Flask) using **GitHub** for version control and **Render** for hosting. Render continuously deploys from a GitHub repository, automating the build and deployment process with minimal configuration.

CI/CD Pipeline Process Deployment Proposal

1. Version Control and Repository Setup

- Use **GitHub** to host and manage your source code.
- Create branches for:
  - Feature development: `feature/branch-name`
  - Bug fixes: `fix/branch-name`
  - Releases: `release/v1.0`
- Protect your main branch with pull request review rules and branch policies.

---

2. Continuous Integration (CI) - Build & Test on GitHub

- GitHub automatically builds and tests code on push via optional GitHub Actions (or locally before pushing).
- Use a `requirements.txt` or `pyproject.toml` to manage Python dependencies.
- Run unit tests using `pytest` or `unittest`.
- Perform static analysis using `flake8`, `black`, or `pylint` locally or as GitHub Actions if needed.

---

3. Continuous Deployment (CD) - Hosting on Render

- Deploy directly from GitHub by connecting your repo to Render.
- Render will:
  - Auto-detect your Python app
  - Install dependencies from `requirements.txt`
  - Run your custom start command (e.g., `gunicorn app.app:app`)
- Add environment variables from `.env` via Render's **Environment tab**.
- Changes pushed to the connected GitHub branch (e.g. `main` or `dev`) trigger automatic deployments.

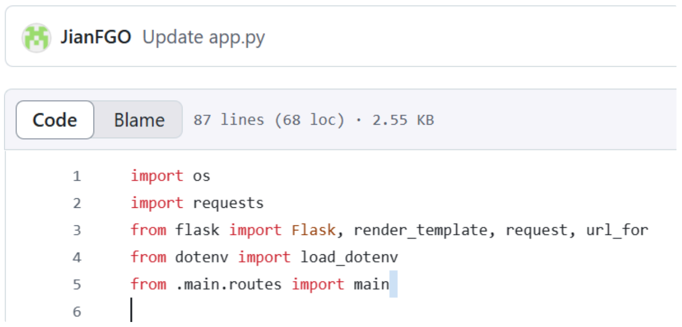
Start Command Example (for Flask in /app folder):

```
gunicorn app.app:app
```

Example requirements.txt (partial):

```
Flask
requests
gunicorn
python-dotenv
```

app.py main.route (May need to change for deployment:


JianFGO Update app.py

Code Blame 87 lines (68 loc) · 2.55 KB

```
import os
import requests
from flask import Flask, render_template, request, url_for
from dotenv import load_dotenv
from .main.routes import main
```

Comparison of CI/CD Tools

Feature	GitHub Actions	Azure DevOps Pipelines	GitLab CI/CD	Render + GitHub *(Chosen)*
Integration	Deep integration with GitHub	Best for Azure services and repos	Best for GitLab repositories	Direct integration with GitHub via webhook or OAuth
Pipeline Definition	YAML-based workflows	YAML-based or Classic UI	YAML-based	No pipeline needed – deploy auto on push
Runner/Agent	GitHub-hosted or self-hosted runners	Microsoft-hosted or self-hosted	GitLab-hosted or self-hosted	Fully managed by Render
Cost	Free for public repos; limited for private	Free tier available, but form approval takes 4–5 days ❌	Free for public projects, limited free minutes on private repos	Free tier includes 750 build minutes/month + shared hosting
Security & Compliance	Supports OIDC, secrets management	Strong enterprise compliance support	Supports SAST, dependency scanning	Secrets managed via dashboard; .env support built-in
Ease of Use	Easy to set up, good for GitHub users	Setup requires approval & more steps ❌	More complex; good for power users, but has a learning curve ❌	Extremely easy – connect repo and hit deploy ✅
Deployment Support	Supports AWS, Azure, GCP, Docker, Kubernetes	Deep Azure DevOps integration only	Supports Kubernetes, Docker, Terraform	Ideal for dynamic apps like Flask, FastAPI ✅
Extensibility	Huge marketplace of Actions	Tight Microsoft service integration	Great with 3rd-party integrations	Simpler stack, but supports webhooks and environment configs ✅

❌ Why the Others Weren’t Chosen

- **Azure DevOps Pipelines**: Required a **manual approval form** that takes **4–5 business days** just to access basic features — far too slow for fast iteration.
- **GitLab CI/CD**: Great for CI, but **GitLab Pages only hosts static sites** — not suitable for our **Python-based web app** (Flask), which needs a dynamic server.
- **GitHub Actions**: Powerful, but we don't need full custom pipeline logic — **Render handles builds & deploys automatically** from GitHub, so Actions weren't necessary here.

Conclusion

Using **GitHub + Render** is a fast and developer-friendly CI/CD setup for Python apps:

- **No YAML pipelines or runners needed**
- **Easy GitHub integration**
- **Automatic redeployment on push**
- **Free hosting for public and small-scale projects**