CI/CD Pipeline Setup for Automatic Deployment

This project uses GitHub Actions to automatically deploy to a test environment (Heroku) whenever changes are pushed to the `main` branch. This document explains the setup and configuration.

Prerequisites

- A GitHub repository with your Python project.
- A Heroku account and a Heroku app set up for the test environment.
- Heroku CLI installed locally for initial setup.
- A `requirements.txt` file listing project dependencies.
- A `Procfile` for Heroku (e.g., `web: gunicorn app:app` for a Flask app).
- A Python application compatible with Heroku (e.g., Flask, Django).

GitHub Actions Workflow

The CI/CD pipeline is defined in `.github/workflows/deploy.yml`. It:

1. Triggers on push to the `main` branch.
2. Runs on an Ubuntu runner.
3. Checks out the code, sets up Python, installs dependencies, and runs tests (optional).
4. Deploys to Heroku using the `akhileshns/heroku-deploy` action.

Workflow File

The workflow file is located at `.github/workflows/deploy.yml`. See the file for the full configuration.

Setup Instructions

1. **Create a Heroku App**:
   - Log in to Heroku and create a new app (e.g., `my-test-app`).

- Note the app name for configuration.

2. **Generate a Heroku API Key**:
   - In Heroku, go to Account Settings > API Key and generate a key.
   - Copy the key securely.

3. **Configure GitHub Secrets**:
   - In your GitHub repository, go to Settings > Secrets and variables > Actions > New repository secret.
   - Add the following secrets:
     - `HEROKU_API_KEY`: Your Heroku API key.
     - `HEROKU_APP_NAME`: Your Heroku app name (e.g., `my-test-app`).
     - `HEROKU_EMAIL`: Your Heroku account email.

4. **Add Workflow File**:
   - Create a `.github/workflows/` directory in your repository.
   - Add the `deploy.yml` file (see the workflow file in this repository).
   - Commit and push the changes to the `main` branch.

5. **Verify Deployment**:
   - After pushing to `main`, go to the Actions tab in your GitHub repository.
   - Check the workflow run to ensure it completes successfully.
   - Visit your Heroku app URL (e.g., `https://my-test-app.herokuapp.com`) to verify the deployment.

Optional: Running Tests

If your project includes tests (e.g., using `pytest`), the workflow runs them before deployment.

Ensure a `tests/` directory exists and update the `run-tests` step in `deploy.yml` with the appropriate test command (e.g., `pytest`).

Alternative Test Environments

Instead of Heroku, you can deploy to:

- **AWS Elastic Beanstalk**: Use the `einaregilsson/beanstalk-deploy` action. Update the workflow with AWS credentials and Elastic Beanstalk configuration.
- **Azure App Service**: Use the `azure/webapps-deploy` action with Azure credentials.
- **Docker Hub**: Build and push a Docker image using `docker/build-push-action`.

Modify the `deploy.yml` file to use the appropriate action and secrets for your chosen environment.

Troubleshooting

- **Workflow Fails**: Check the Actions tab for error logs. Common issues include missing secrets, incorrect app names, or dependency errors.
- **Heroku Deployment Fails**: Ensure your `Procfile` and `requirements.txt` are correct. Run `heroku logs --tail` for detailed logs.
- **Tests Fail**: Verify your test suite locally with `pytest` before pushing.

For further details, refer to:

- [GitHub Actions Documentation](https://docs.github.com/en/actions)
- [Heroku Deployment Guide](https://devcenter.heroku.com/articles/getting-started-with-python)