To set up a CI/CD pipeline for your FastAPI project using GitHub Actions, here is a clear step-by-step guide including the necessary code, configuration, and commands based on best practices for Dockerized FastAPI apps:

Step 1: Prepare Your GitHub Repository

Make sure your FastAPI project directory contains:

- `main.py` (your FastAPI app)
- `Dockerfile` (to build the image)
- `requirements.txt` (dependencies)
- Your GitHub repository initialized and pushed with these files

Step 2: Create GitHub Actions Workflow File

Inside your project repo, create the directory `.github/workflows` if it doesn't exist. Then create a file named `ci-cd.yml` with the following content:

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main   # or 'master' depending on your default branch

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

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1
```

```
      - name: Log in to Docker Hub
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v2
        with:
          context: .
          push: true
          tags: your-dockerhub-username/fastapi-app:latest

      # Optional: Add deployment step if you have a server with SSH access
      # - name: Deploy to Server
      #   run: |
      #     ssh user@your-server-ip "docker pull your-dockerhub-username/fastapi-app:latest && docker stop fastapi-container || true && docker rm fastapi-container || true && docker run -d -p 80:80 --name fastapi-container your-dockerhub-username/fastapi-app:latest"
```

Notes:

- Replace `your-dockerhub-username` with your actual Docker Hub username.
- The workflow triggers on pushes to the `main` branch.
- It logs into Docker Hub using secrets (see next step).
- Builds and pushes your Docker image to Docker Hub.
- Optionally, you can add a deployment step that SSHs into your server and restarts the container.

Step 3: Add Secrets to GitHub Repository

Go to your GitHub repository → Settings → Secrets and variables → Actions → New repository secret, add:

- `DOCKER_HUB_USERNAME` - your Docker Hub username
- `DOCKER_HUB_ACCESS_TOKEN` - a Docker Hub access token or password (token recommended)

This allows GitHub Actions to authenticate with Docker Hub securely.

Step 4: Commit and Push Workflow

Commit your new workflow file and push it to GitHub:

```
git add .github/workflows/ci-cd.yml
git commit -m "Add GitHub Actions CI/CD workflow"
git push origin main
```

This will trigger the pipeline automatically on GitHub.

Step 5: Monitor GitHub Actions

- Go to your GitHub repo → Actions tab.
- You should see your workflow running.
- It will checkout code, build the Docker image, log in to Docker Hub, and push the image.
- If you added deployment, it will deploy to your server.

Additional Tips

- **Testing:** You can extend the workflow to run tests before building the image.
- **Taskfile Automation:** For complex tasks, consider using a Taskfile to automate commands and integrate with GitHub Actions pre-jobs for efficiency[7].
- **Branch Strategy:** Adjust branches in the workflow triggers as per your development flow.
- **Deployment:** For deployment, ensure your server is SSH-accessible and Docker is installed.