Tutorial: Integração com GitHub Actions para Deploy Automático (Windows)

Karina Casola

16 de agosto de 2025

Conteúdo

1 Introdução e Configuração Inicial

1.1 O que é GitHub Actions?

GitHub Actions é uma plataforma de integração contínua e entrega contínua (CI/CD) integrada diretamente no GitHub. Ele permite automatizar workflows de construção, teste e implantação diretamente do seu repositório GitHub.

Principais conceitos:

- **Workflows**: Arquivos YAML que definem os processos automatizados
- **Jobs**: Conjuntos de passos que são executados no mesmo executor

- **Steps**: Tarefas individuais que podem executar comandos ou ações
- **Actions**: Comandos reutilizáveis que podem ser incluídos nos seus workflows

1.2 Pré-requisitos

- **Conta no GitHub** - `https://github.com`
- **Git instalado** - `https://git-scm.com/download/win`
- **VS Code** - `https://code.visualstudio.com/download`
- **Docker Desktop** - `https://docs.docker.com/desktop/install/windows-install/`
- **Python 3.8+** - `https://www.python.org/downloads/`

1.3 Configurando Git e GitHub no VS Code

1. Abra o VS Code
2. Instale a extensão "GitHub Pull Requests and Issues"(opcional, mas útil)
3. Abra o terminal integrado (Ctrl+‘)
4. Configure seu usuário Git:

```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
```

5. Conecte-se ao GitHub:
   - Clique no ícone de conta no canto inferior esquerdo
   - Selecione "Sign in with GitHub"
   - Siga o fluxo de autenticação

2 Criando um Projeto Django

1. Crie uma nova pasta para seu projeto e abra no VS Code
2. No terminal, crie um ambiente virtual:

```
python -m venv venv
.\venv\Scripts\activate
```

3. Instale o Django:

```
pip install django
```

4. Crie um novo projeto Django:

```
django-admin startproject deploydemo
cd deploydemo
python manage.py startapp myapp
```

5. Modifique `settings.py` para incluir 'myapp' em INSTALLED_APPS

6. Crie uma view simples em `myapp/views.py`:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello from Django with GitHub Actions!")
```

7. Adicione a URL em `deploydemo/urls.py`:

```
from django.contrib import admin
from django.urls import path
from myapp.views import home

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home, name='home'),
]
```

8. Teste localmente:

```
python manage.py runserver
```

   Acesse `http://localhost:8000` para ver a mensagem

3 Configurando o Docker

3.1 Criando o Dockerfile

Crie um arquivo `Dockerfile` na raiz do projeto:

```
# Use a imagem oficial do Python
FROM python:3.9-slim

# Define o diret rio de trabalho
WORKDIR /app

# Copia os arquivos de requisitos
COPY requirements.txt .

# Instala as depend ncias
RUN pip install --no-cache-dir -r requirements.txt

# Copia todo o conte do para o diret rio de trabalho
COPY . .

# Exp e a porta 8000
EXPOSE 8000

# Comando para rodar a aplica  o
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

3.2 Criando requirements.txt

```
Django>=3.2,<4.0
```

3.3 Criando docker-compose.yml

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DJANGO_SETTINGS_MODULE=deploydemo.settings
    command: python manage.py runserver 0.0.0.0:8000
```

3.4 Testando localmente

```
docker-compose up --build
```

Acesse `http://localhost:8000` para verificar

4 Configurando o GitHub Actions

1. Crie um repositório no GitHub (não inicialize com README)
2. No terminal do seu projeto:

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/seu-usuario/nome-repo.git
git push -u origin main
```

3. Crie a pasta `.github/workflows` na raiz do projeto
4. Crie um arquivo `deploy.yml` dentro dessa pasta:

```
name: Django CI/CD Pipeline

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install dependencies
```

```
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        python manage.py test

  build-and-deploy:
    needs: test
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${{ secrets.DOCKER_HUB_USERNAME }}
        password: ${{ secrets.DOCKER_HUB_TOKEN }}

    - name: Build and push Docker image
      uses: docker/build-push-action@v4
      with:
        push: true
        tags: ${{ secrets.DOCKER_HUB_USERNAME }}/django-demo:latest

    - name: SSH and deploy
      uses: appleboy/ssh-action@master
      with:
        host: ${{ secrets.SSH_HOST }}
        username: ${{ secrets.SSH_USERNAME }}
        key: ${{ secrets.SSH_KEY }}
        script: |
          docker pull ${{ secrets.DOCKER_HUB_USERNAME }}/django-demo:latest
          docker-compose down
          docker-compose up -d
```

5 Configurando Secrets no GitHub

1. No seu repositório GitHub, vá em Settings ¿ Secrets ¿ Actions
2. Adicione os seguintes segredos:
   - `DOCKER_HUB_USERNAME`: Seu usuário Docker Hub
   - `DOCKER_HUB_TOKEN`: Token de acesso do Docker Hub
   - `SSH_HOST`: Endereço do servidor
   - `SSH_USERNAME`: Usuário SSH
   - `SSH_KEY`: Chave privada SSH

6 Simulando o Servidor com Docker (Local)

Para testar localmente sem um servidor real, modifique o passo "SSH and deploy"no workflow para:

```
- name: Simulate deployment
  run: |
    echo "Deployment would happen here in a real scenario"
    echo "For local testing, you would pull and run the container manually"
```

Ou crie um servidor local com Docker:

```
docker run -d -p 2222:22 -e USERNAME=test -e PASSWORD=test -e SUDO_ACCESS=
    true linuxserver/openssh-server
```

7 Testando o Fluxo Completo

1. Faça uma modificação no código
2. Commit e push:

```
git add .
git commit -m "Update message"
git push origin main
```

3. Vá para o GitHub ¿ Seu repositório ¿ Actions
4. Veja o workflow em execução
5. Quando completar, acesse `http://localhost:8000` para ver as mudanças

8 Dicas e Solução de Problemas

- **Erros de Build**:
  - Verifique os logs do GitHub Actions
  - Teste localmente com `docker-compose up --build`
- **Problemas com Docker Hub**:
  - Verifique se o token está correto
  - Garanta que o repositório no Docker Hub existe
- **Problemas com SSH**:
  - Teste a conexão SSH manualmente primeiro
  - Verifique permissões da chave (deve ser 600)
- **Para produção real**:
  - Use um servidor real (AWS, DigitalOcean, etc.)
  - Configure domínio e HTTPS
  - Adicione mais testes ao pipeline

9 Conclusão

Agora você tem um pipeline CI/CD completo:

1. O código é testado automaticamente em cada push
2. Uma imagem Docker é construída e enviada para o Docker Hub
3. A aplicação é implantada automaticamente no servidor

Isso permite desenvolvimento ágil com garantia de qualidade e implantação rápida.