🧠 Perguntas rápidas

1. O que é CI/CD e por que é importante?
   💡 *Resposta:*

   CI/CD é um conjunto de práticas e ferramentas que automatizam integração, testes e entrega/deploy de software.

   CI (Continuous Integration) executa builds e testes automáticos a cada alteração; CD (Continuous Delivery/Deployment) automatiza a entrega ou deploy para ambientes.

   Importância: reduz erros, acelera feedback e releases, aumenta confiabilidade e produtividade da equipe

2. Em qual pasta os workflows do GitHub ficam armazenados?
   💡 *Resposta:*

   Os workflows ficam armazenados na pasta "/.github/workflows" na raiz do repositório. Cada workflow é um arquivo YAML com extensão de arquivo .yml ou .yaml. As ações do github leem

🧩 *Etapa 3 – Verificando o pipeline*

No GitHub:

- *Vá em Actions → Teste CI*
- *Veja o workflow em execução após o push*

Perguntas:

1. *O que aparece no log do GitHub Actions após a execução?*
   💡 *Resposta:*

   Lista de passos do workflow com status (started/completed).
   Saída do passo Checkout (actions/checkout@v4).
   Saída do passo de setup do Python (actions/setup-python@v5), mostrando a versão selecionada.
   Saída do passo "Executar script" com o stdout do script, por exemplo: Hello CI/CD!

Códigos de saída, timestamps, versão das actions e o status final do job (Succeeded se tudo terminou com exit code 0).

Se algo falhar, o log inclui o erro e qual passo falhou.

2. *O que acontece se alterar o código e fizer novo push?*
   💡 Resposta:
   Todas as ações são executadas novamente, sendo assim o script será executado também, porém a saída pode mudar caso o código do script se altere;

🧩 Etapa 4 – Introduzindo um teste automatizado (extra)

Crie um arquivo `test_sample.py`:

```
def test_soma():

    assert 1 + 1 == 2
```

Atualize o workflow para rodar testes com `pytest`:

```
- name: Instalar dependências

  run: pip install pytest



- name: Executar testes

  run: pytest -v
```

Faça commit e push novamente.
O GitHub Actions agora executará testes automatizados em cada push.

Pergunta extra:
O que acontece se um teste falhar?
💡 Resposta:

O job e o workflow ficam marcados como "failed" (verá um X/vermelho em Actions e na aba Checks do PR).

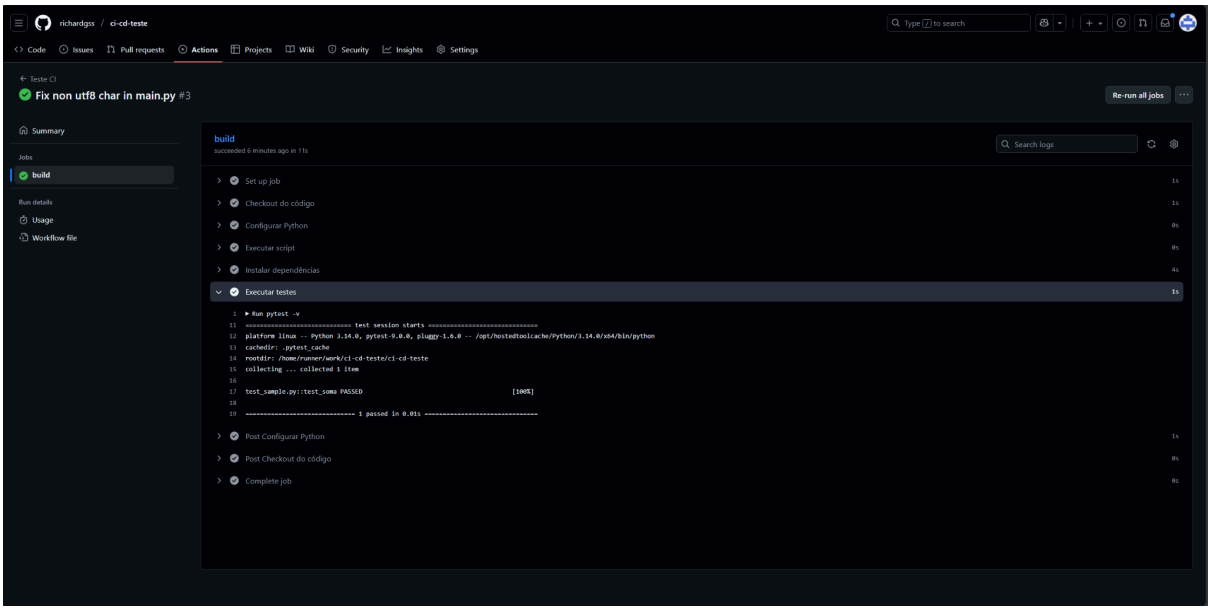
Os logs mostram o passo que falhou com a saída do pytest: nome do teste, a asserção que falhou, traceback e o código de saída não-zero.

Passos seguintes no mesmo job normalmente são pulados após a falha, salvo se você usar continue-on-error: true ou condições como if: always() (úteis para upload de artefatos/limpeza).

Em pull requests, checks de CI que falham bloqueiam o merge se houver regras de proteção de branch exigindo checks aprovados.

Dependendo da integração, podem ser enviadas notificações (e-mail, Slack, etc.).

O que fazer: abrir os logs, reproduzir o teste localmente (pytest), corrigir o código/tests, commitar e dar push — isso dispara um novo run; ou usar "Re-run jobs" na UI do Actions para reexecutar.



Como o GitHub Actions ajuda a detectar erros cedo?

Automatiza builds, linting, testes e scans em cada push/PR, evitando que código quebrado chegue à branch principal.

Fornece logs imediatos com falhas (compile, testes, dependências), permitindo correção antes do merge.

Integra-se com requisitos de proteções de branch: PRs só podem ser mesclados se checks passarem.

Permite rodar testes em múltiplas versões/ambientes (matrix) para detectar regressões específicas.

Quais seriam exemplos reais de CI/CD em projetos web ou mobile?

Web (frontend): rodar ESLint, unit/integration tests, build otimizado, deploy automático para Netlify/Vercel/GitHub Pages ou upload de assets para S3 + CloudFront; ou criar e enviar imagem Docker para Docker Hub / ECR e atualizar Kubernetes.

Web (backend): rodar linters, testes, security scans, build da imagem Docker, publicar imagem num registry e atualizar serviço em ECS/GKE/AKS com rollout.

Mobile (Android/iOS): rodar lint e testes, compilar APK/AAB/IPA, assinar builds usando chaves armazenadas em Secrets, distribuir para testers via Firebase App Distribution ou TestFlight, e publicar automaticamente no Google Play/App Store com Fastlane.

Exemplos adicionais: executar SCA (Dependabot, Snyk), testes end-to-end em CI com Cypress/Playwright, e pipelines de release automatizados com ambientes staging → produção.

Como o deploy automático poderia ser feito a partir deste pipeline?

Adicionar um job de deploy que roda após build+test (usando needs: [build]) e que só roda em branches/ tags específicos (por ex. main ou semver tags).

Autenticar com o provedor usando Secrets (AWS/AZURE/GCP/Netlify/Vercel/Google Play/Apple) guardados no repositório.

Passos comuns: gerar artefato (build), armazenar como artifact ou publicar imagem Docker, então executar ação de deploy (aws s3 sync / aws ecs update-service, azure/webapps-deploy, peaceiris/actions-gh-pages, vercel-action, firebase-tools, fastlane).

Proteções: usar ambientes com aprovações manuais para produção, checks obrigatórios, e health checks/rollbacks automáticos se deploy degradar.

Segurança: não colocar credenciais no código; usar secrets, scopes mínimos e rotacionar chaves.