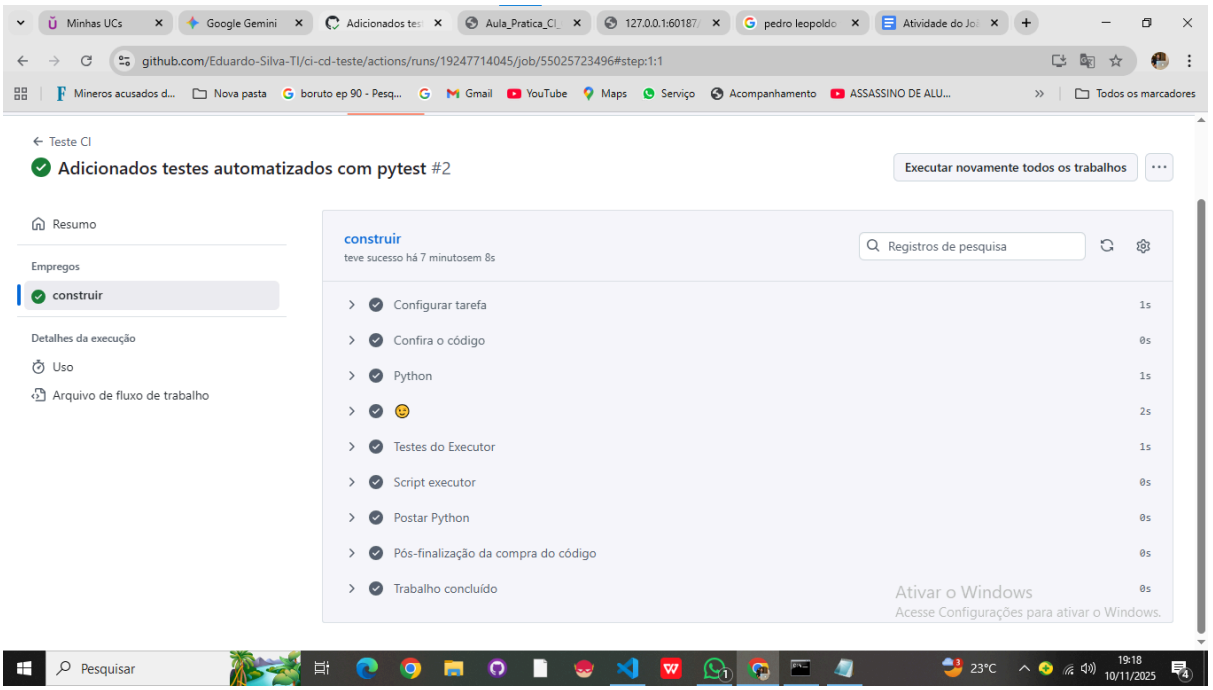
Etapa 3

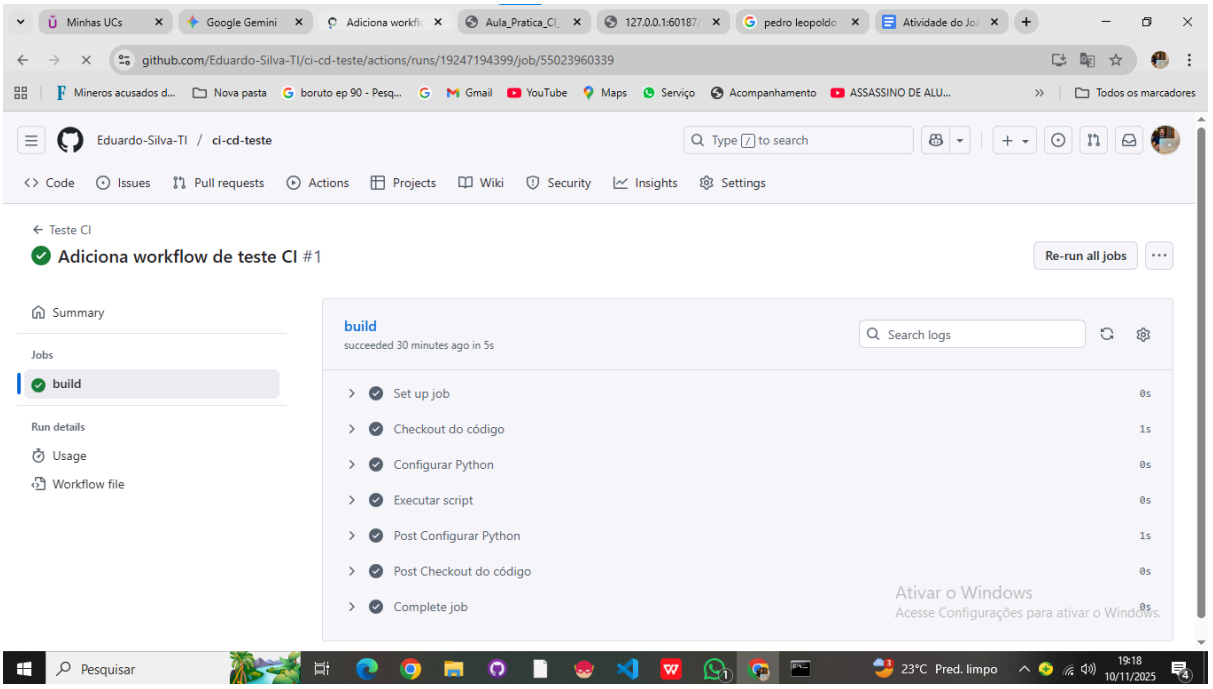
1. O log mostra cada "step" que definimos no .yml. Você verá o "Checkout", a "Configuração do Python" e, o mais importante, no "step" Executar script, aparecerá a saída Hello CI/CD!

2. O GitHub Actions detecta esse novo push automaticamente (por causa da configuração on: push: branches: ["main"]) e inicia uma nova execução de todo o workflow do zero.

Etapa 4

Pergunta extra: Se um assert falhar (ex: assert 1 + 1 == 3), o step "Executar testes" dará erro. Isso interrompe todo o workflow e o GitHub marca o "job" com um X vermelho. Isso é o objetivo da CI: impedir que código com bugs avance.





4. Para finalizar:

- Ele **executa os testes automatizados (como o `pytest`) a cada push**. Se uma nova alteração de código quebrar um teste, o desenvolvedor (ou a equipe) é notificado imediatamente pelo "build" vermelho.

- **Web:** A cada push, o pipeline poderia rodar testes, fazer o *build* do projeto (ex: `npm run build` para React) e, se passar, fazer o *deploy* automático para um servidor (como Vercel, Netlify ou AWS).
Mobile: A cada push, o pipeline poderia compilar o app (APK/IPA) e enviá-lo para um serviço de testes internos (como TestFlight da Apple ou Google Play Internal Testing).

- Adicionando um **novo "step" no final** do nosso "job" `build`. Esse "step" só rodaria se os testes passassem. Ele usaria uma *action* específica (ex: `aws-s3-sync`) ou um script (ex: `scp`/`rsync`) para copiar os arquivos do projeto para o servidor de produção.

Link do GitHub:

https://github.com/Eduardo-Silva-TI/ci-cd-teste/actions/runs/19247194399/job/55023960339