



Uni**SEN**AI

# Trilha 06

*DevOps*

## Instruções para a melhor prática de Estudo

1. **Leia atentamente todo o conteúdo:** Antes de iniciar qualquer atividade, faça uma leitura detalhada do material fornecido na trilha, compreendendo os conceitos e os exemplos apresentados.
2. **Não se limite ao material da trilha:** Utilize o material da trilha como base, mas busque outros materiais de apoio, como livros, artigos acadêmicos, vídeos, e blogs especializados. Isso enriquecerá o entendimento sobre o tema.
3. **Explore a literatura:** Consulte livros e publicações reconhecidas na área, buscando expandir seu conhecimento além do que foi apresentado. A literatura acadêmica oferece uma base sólida para a compreensão de temas complexos.
4. **Realize todas as atividades propostas:** Conclua cada uma das atividades práticas e teóricas, garantindo que você esteja aplicando o conhecimento adquirido de maneira ativa.
5. **Evite o uso de Inteligência Artificial para resolução de atividades:** Utilize suas próprias habilidades e conhecimentos para resolver os exercícios. O aprendizado vem do esforço e da prática.
6. **Participe de debates:** Discuta os conteúdos estudados com professores, colegas e profissionais da área. O debate enriquece o entendimento e permite a troca de diferentes pontos de vista.
7. **Pratique regularmente:** Não deixe as atividades para a última hora. Pratique diariamente e revise o conteúdo com frequência para consolidar o aprendizado.
8. **Peça feedback:** Solicite o retorno dos professores sobre suas atividades e participe de discussões sobre os erros e acertos, utilizando o feedback para aprimorar suas habilidades.

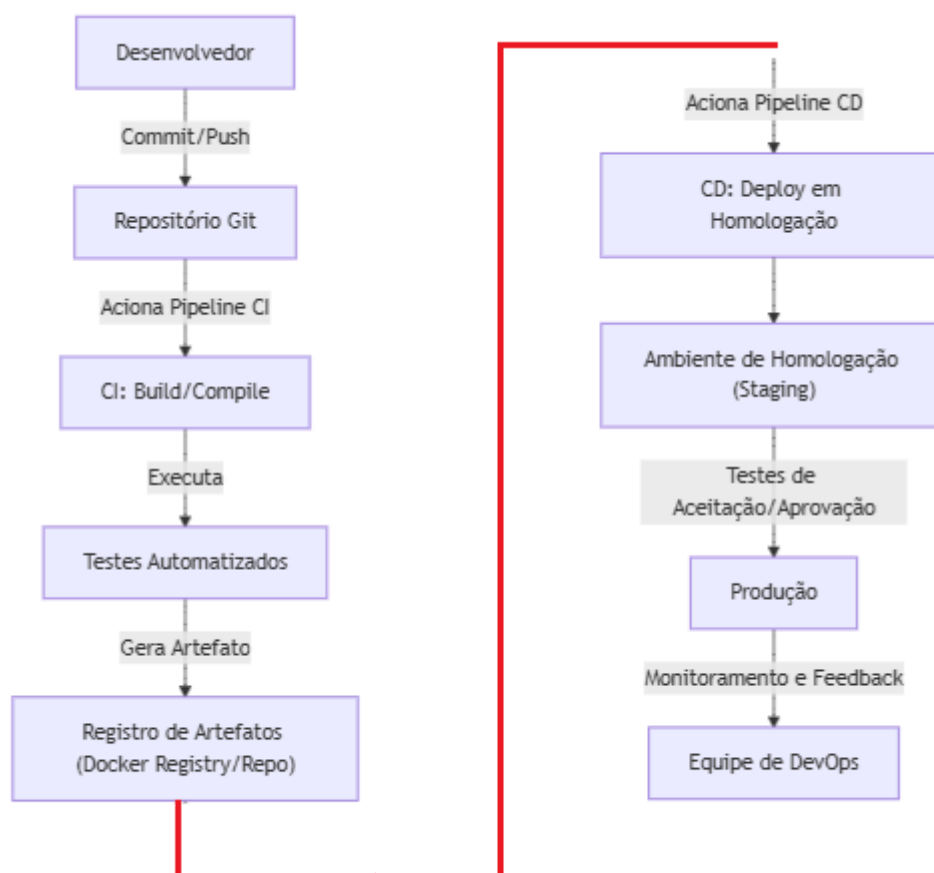
Essas instruções são fundamentais para garantir um aprendizado profundo e eficaz ao longo das trilhas.

## 1. DevOps

DevOps é uma cultura, um conjunto de práticas e ferramentas que visa integrar equipes de desenvolvimento (Dev) e operações (Ops) para otimizar todo o ciclo de entrega de software.

Em um fluxo tradicional, o time de desenvolvimento cria o software e, depois, repassa para a equipe de operações implantar e gerenciar o ambiente em produção. Esse modelo pode gerar lentidão, gargalos e conflitos de responsabilidade. O DevOps, por outro lado, promove maior colaboração entre esses dois times, automatizando processos de integração, entrega, monitoramento e correção, resultando em entregas de software mais rápidas e confiáveis.

**Exemplo:** Abaixo está um exemplo simplificado de como pode funcionar um pipeline DevOps em um projeto de software. Este diagrama exemplifica o fluxo de trabalho do desenvolvedor até a aplicação rodando em produção.



## 2. GitHub Actions

O GitHub Actions é uma ferramenta de automação integrada ao GitHub, que permite criar workflows para CI/CD (Continuous Integration e Continuous Deployment). Com ele, podemos automatizar tarefas como execução de testes, build de aplicações, deploys, verificações de segurança e muito mais.

## Principais usos do GitHub Actions:

- Execução automática de testes sempre que houver um novo commit ou pull request.
- Deploy automatizado para servidores ou serviços como AWS, Vercel, Netlify e Firebase.
- Verificações de código (Linting, Segurança, Code Coverage) antes de aceitar novas alterações.
- Compilação e empacotamento de aplicações automaticamente.

**Exemplo:** Criação de um workflow para rodar testes automatizados em Node.js toda vez que um commit for enviado ao repositório, ao final será enviado uma notificação ao usuário através do Discord.

No arquivo YAML definimos todo o script necessário, desde checkout do projeto, instalação no node, testes do projeto e por fim o envio da notificação.

## Exemplo Prático: Automatizando Testes com GitHub Actions

Vamos criar um workflow para rodar testes automatizados em Node.js toda vez que um commit for enviado ao repositório.

### Criando o Arquivo de Configuração

Dentro da pasta do seu projeto, crie a pasta **.github/workflows** e adicione um arquivo chamado **test.yml**.

### Adicionando um Workflow no GitHub Actions

No arquivo **test.yml**, insira o seguinte código YAML:

```

name: CI - Rodar Testes Automatizados

on:
  push:    # Executa quando um commit é enviado
  pull_request: # Executa em pull requests

jobs:
  test:
    runs-on: ubuntu-latest # Define o ambiente do workflow (Linux)

    steps:
      - name: 🚀 Checkout do repositório
        uses: actions/checkout@v4

      - name: 🛠 Configurar Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18

      - name: 📦 Instalar dependências
        run: npm install

      - name: 🏃 Rodar os testes
        run: npm test # Executa os testes unitários

```

### Explicação do Workflow:

- Gatilhos (**on**): O workflow roda sempre que houver um push ou pull request.
- **runs-on**: Define que o workflow será executado em um ambiente Ubuntu (Linux).
- **steps**: Cada passo executa uma ação específica, como baixar o código, instalar dependências e rodar os testes.
- **actions/checkout@v4**: Baixa o código do repositório.
- **actions/setup-node@v4**: Configura o ambiente Node.js.
- **npm install**: Instala as dependências do projeto.
- **npm test**: Executa os testes automatizados.

### Testando e Verificando

```

git add .
git commit -m "Adicionando GitHub Actions para testes"
git push origin main

```

### Acesse o GitHub e verifique a execução do workflow:

- Vá até a aba **"Actions"** no repositório.
- Selecione o workflow **"CI - Rodar Testes Automatizados"**.
- Verifique os logs para conferir se os testes rodaram corretamente.

## 2.1. Envio de Notificações

Podemos configurar o GitHub Actions para enviar notificações para um canal do Discord sempre que um evento ocorrer no repositório, como um commit, pull request ou falha nos testes. Isso ajuda a manter a equipe informada sobre o status do projeto.

## 2.2. Criar um Webhook no Discord

Antes de configurar o GitHub Actions, precisamos criar um Webhook no Discord para receber as notificações.

- Acesse seu servidor Discord e clique no nome dele no topo esquerdo.
- Vá para "**Configurações do Servidor**" → "**Integrações**".
- Clique em "**Criar Webhook**" e dê um nome para ele (exemplo: "**Notificação GitHub**").
- Escolha o canal onde as notificações serão enviadas.
- Copie a URL do Webhook e guarde-a, pois será usada no GitHub Actions.

### Criar um Secret no GitHub para o Webhook

Para não expor a URL do webhook no código, armazenamos no GitHub.

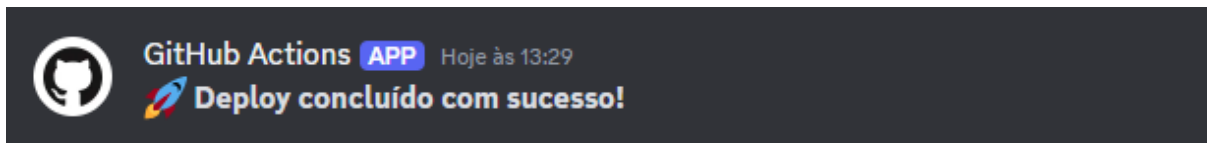
- No repositório do GitHub, vá para "**Settings**" → "**Secrets and variables**" → "**Actions**".
- Clique em "**New repository secret**".
- No campo Name, insira: **DISCORD\_WEBHOOK**.
- No campo Value, cole a URL do Webhook copiada no Discord.
- Clique em "**Add secret**".

### Configuração adicional no arquivo YAML:

```
- name: Enviar notificação para Discord
  if: always()
  env:
    DISCORD_WEBHOOK: ${ secrets.DISCORD_WEBHOOK }
  run: |
    curl -H "Content-Type: application/json" \
    -X POST \
    -d '{
      "username": "GitHub Actions",
      "avatar_url": "https://github.githubassets.com/images/modules/logos_page/GitHub-Mark.png",
      "content": "🚀 **Deploy concluído com sucesso!**"
    }' \
    $DISCORD_WEBHOOK
```

### Explicação do Workflow:

- **curl**: Envia uma requisição HTTP para o Webhook do Discord.
- **\${ secrets.DISCORD\_WEBHOOK }**: Usa a URL secreta armazenada no GitHub Secrets.
- **Mensagem**: Mensagem que irá aparecer no Discord.

**Resultado final no Discrod:****2.3. Cron**

Cron é um serviço (ou daemon) presente em sistemas Unix/Linux que executa tarefas de forma automática em horários ou intervalos pré-definidos.

Com o Cron, você pode programar scripts ou comandos para rodar diariamente, semanalmente, mensalmente etc., sem precisar fazer isso manualmente.

**Exemplo Simples de Cron no Linux**

Supondo que você queira executar uma ação (por exemplo, rodar testes ou simplesmente mostrar a data/hora) todo dia às 3h da manhã, você cria um arquivo YAML (por exemplo, `.github/workflows/cron-example.yml`) no seu repositório com o seguinte conteúdo:

```
name: Exemplo de cron no GitHub Actions

on:
  schedule:
    - cron: "0 3 * * *" # Executa diariamente às 3h (UTC)

jobs:
  mostrar-data:
    runs-on: ubuntu-latest
    steps:
      - name: Exibir data e hora
        run: date
```

**name** é o nome do workflow (você escolhe).

**on.schedule.cron** define a expressão de agendamento (no formato cron). Aqui, `"0 3 * * *"` significa às 3h UTC de todos os dias.

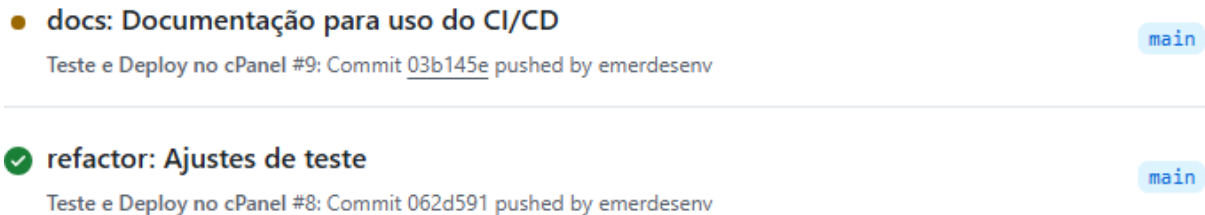
Em **jobs**, definimos o que o GitHub Actions fará nessa execução programada. Nesse caso, o job **"mostrar-data"** roda em **ubuntu-latest** e executa o comando `date`, apenas para exibir a data e hora no log.

Assim, você tem uma rotina agendada na nuvem sem precisar configurar nada no seu próprio servidor.

Aqui segue um site de referência para calcular melhor o tempo: [Cron Guru](#).

## Exemplo de um fluxo simples com CI/CD

na aba Actions temos a lista de todos os nossos workflow, com isso conseguimos ter uma visão de quais os processo estão rodando, qual usuário submeteu e sua branch de origem, abaixo um exemplo:



The screenshot shows two workflow entries in the GitHub Actions interface. The first entry is 'docs: Documentação para uso do CI/CD' with a yellow dot icon, indicating it is currently running. Below it, the text reads 'Teste e Deploy no cPanel #9: Commit 03b145e pushed by emerdesenv'. To the right, a blue pill indicates the branch is 'main'. The second entry is 'refactor: Ajustes de teste' with a green checkmark icon, indicating it has completed successfully. Below it, the text reads 'Teste e Deploy no cPanel #8: Commit 062d591 pushed by emerdesenv'. To the right, a blue pill indicates the branch is 'main'.

Acima são listados dois workflow, o primeiro da lista está sendo processado já o último já foi e o mesmo concluído com êxito. Ao clicar sobre algum workflow conseguimos ter uma visão mais detalhada de todo o processo sendo realizado e o que foi, exemplo abaixo:

Aqui realiza os primeiros processos, que definem os testes web e conferência das variáveis de ambiente:



Aqui realiza as etapas finais que são, deploy no cPanel e Envio da Notificação ao Discord:



**Testes com Cypress:** É justamente a etapa onde são realizados todos os testes, esse processo se dá quando um desenvolvedor realiza algum envio de alteração em código para o GitHub, neste exemplo definir que esse workflow será executado quando identificar alguma alteração que foi enviada na branch main.

**Variáveis de Ambiente:** Aqui faço um teste simples para averiguar se todas as variáveis de ambiente estão disponíveis para realizar o processo de deploy.







**Deploy no cPanel:** Efetivamente realizada todo o processo de realizar as transferências dos arquivos que foram alterados ao servidor.

**Envio Notificação:** Aqui é feito um envio de uma mensagem para um grupo de trabalho que fica no Discord, isso facilita e muito para realizar avisos a equipe de que um deploy foi implementado em produção.

## Observações

Ainda na mesma aba tenho as evidências dos testes realizados e algumas métricas:



Resultado	Aprovado 	Falhou 	Pendente 	Ignorado 	Duração 
Passando 	1	0	0	0	9.936s

Nome

Tamanho



Evidências dos Testes

1,1 MB

### 3. GitHub Pages

O GitHub Pages é um serviço gratuito oferecido pelo GitHub que permite hospedar sites estáticos diretamente de um repositório GitHub. Ele é ideal para portfólios, documentação de projetos, blogs e páginas simples que não precisam de backend (como PHP, Node.js etc.).

#### Usando GitHub Pages

O GitHub Pages é um recurso gratuito do GitHub que permite hospedar sites diretamente a partir de um repositório. Ele é muito usado para:

- Portfólios pessoais
- Documentação de projetos
- Blogs
- Sites simples de projetos open source

#### Como funciona?

Você cria um repositório no GitHub com os arquivos do seu site (HTML, CSS, JS etc.). A partir dele, o GitHub gera e hospeda automaticamente uma página web acessível por um link do tipo:

```
https://seunome.github.io/nomedorepositorio/
```

#### Principais características:

- Gratuito para projetos públicos
- Suporta HTML estático ou sites gerados com Jekyll (um gerador de site estático)
- Sem necessidade de configurar servidor, hospedagem ou domínio (mas dá pra usar domínio personalizado também)

#### Como publicar?

- Crie um repositório no GitHub
- Adicione os arquivos do seu site
- Vá em Settings > Pages

- Escolha a branch e a pasta onde estão os arquivos (geralmente main e /root ou /docs)
- Pronto! O GitHub vai gerar um link com seu site publicado

### Onde ficam os arquivos

- Você pode configurar o GitHub Pages para buscar os arquivos em:
- a branch main ou master
- a pasta /docs dentro do repositório
- a branch gh-pages (comum em projetos que geram os arquivos automaticamente, como Jekyll, React etc.)

### Limitações

- Apenas para sites estáticos (sem backend dinâmico)
- Limite de uso para fins pessoais/projetos (não para serviços comerciais pesados)
- Sem suporte direto a banco de dados ou servidores dinâmicos

### Exemplo

Tendo nosso projeto criado poderemos seguir com as demais configurações.




### Segue o padrão de configuração - Pages

- Selecione a opção **Deploy from a branch**
- Escolha a branch de origem e o diretório que por padrão é o root

## GitHub Pages

[GitHub Pages](#) is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at <https://emerdesenv.github.io/inicial-site-exemplo/>  
Last [deployed](#) by  [emerdesenv](#) 1 minute ago

 Visit site



## Build and deployment


Source

Deploy from a branch ▾

Branch

Your GitHub Pages site is currently being built from the `main` branch. [Learn more about configuring the publishing source for your site.](#)

 main ▾

 / (root) ▾

Save

A partir que uma alteração subir para a branch especificada o mesmo irá acionar um processo para atualização do Page no menu Actions:

### pages build and deployment

pages-build-deployment #10: by emerdesenv

## 4. SonarQube

O SonarQube é uma ferramenta de análise estática de código que identifica vulnerabilidades, problemas de qualidade e padrões ruins de codificação em projetos de software. Ele é usado para garantir que o código esteja limpo, seguro e seguindo boas práticas antes de ser implantado em produção.

### Como o SonarQube funciona?

- Escaneia o código-fonte para identificar problemas.
- Avalia critérios como segurança, manutenibilidade e complexidade do código.
- Gera um relatório detalhado com sugestões de correções.
- Classifica a qualidade do código com base em métricas como bugs, vulnerabilidades e cobertura de testes.

**Exemplo:** Imaginemos que entramos em uma empresa onde tenha um projeto legado e o gestor decidiu que iremos realizar várias mudanças nele, mas a equipe de desenvolvimento sugeriu que talvez seja necessário criar um projeto novo, nessa situação o SonarQube irá ajudar e muito, pois com ele conseguimos escanear todo o sistema em busca de possíveis problemas no software e assim apresentar um relatório condizente para tais mudanças no escopo.

## Exemplo Prático: Analisando Código com o SonarQube

Para rodar o SonarQube localmente, podemos usar Docker, caso ainda não tenha o Docker, consulte a seção [Docker Setup](#):

```
docker run -d --name sonar -p 9000:9000 sonarqube:lts
```

Isso iniciará um servidor SonarQube na porta **9000**.

### Rodando Sonar

Caso já tenha o container, basta subir o container com o comando: **docker start sonar**

### Configurar o SonarQube em um Projeto

Em um projeto Node.js ou Java, podemos usar o SonarScanner para analisar o código.

### Baixe o SonarScanner:

```
npm install -g sonarqube-scanner
```

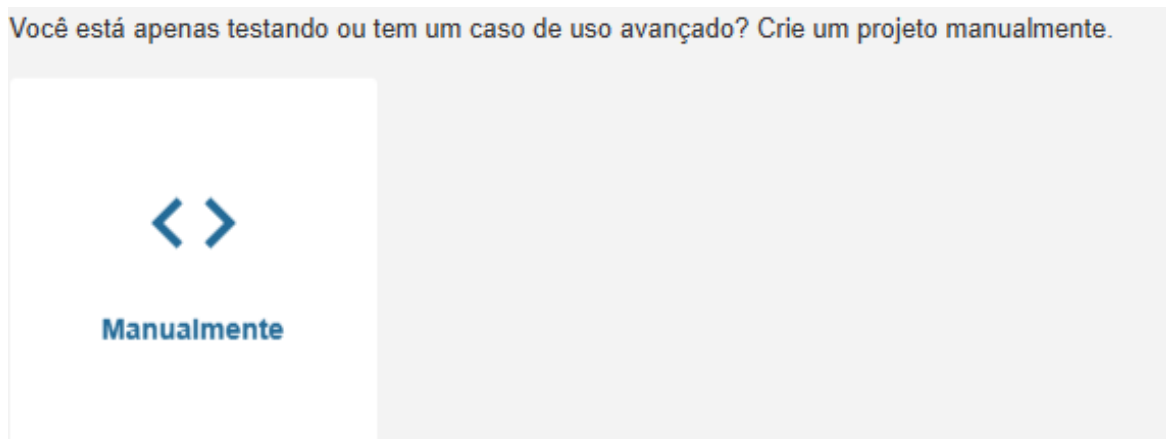
### Logando no Sonar:

URL: <http://localhost:9000>

Na primeira vez irá pedir usuário e senha, basta colocar as credenciais abaixo. Após isso você deverá criar uma senha:

- Usuário: **admin**
- Senha: **admin**


**Passo 1** - Clique sobre a opção abaixo:



**Passo 2** - Preencha as informações abaixo:


Todos os campos marcados com \* são obrigatórios

**Nome de exibição do projeto \***

test\_inicial 

Até 255 caracteres. Alguns scanners podem substituir o valor fornecido.

**Chave do projeto \***

test\_inicial 

A chave do projeto é um identificador exclusivo para o seu projeto. Ela pode conter até 400 caracteres. Os caracteres permitidos são alfanuméricos, '-' (traço), '\_' (sublinhado), '.' (ponto final) e ':' (dois pontos), com pelo menos um caractere que não seja um dígito.

**Nome da filial principal \***


main

O nome do branch padrão do seu projeto [Saiba mais](#)

**Configurar**

**Passo 3** - Clique sobre a opção abaixo:

Você está apenas testando ou tem um caso de uso avançado? Analise seu projeto localmente.



**Localmente**

#### Passo 4 - Gere a chave de acesso:

Forneça um token

Gerar um token de projeto

Nome do token <span>?</span>	Expira em	
<input type="text" value="Analyze 'test_inicial'"/>	<input type="text" value="30 dias"/>	<input type="button" value="Gerar"/>



Observe que este token permitirá que você analise apenas o projeto atual. Se quiser usar o mesmo token para analisar vários projetos, será necessário gerar um token global na sua [conta de usuário](#). Consulte a [documentação](#) para obter mais informações.

O token é usado para identificá-lo quando uma análise é realizada. Se ele tiver sido comprometido, você pode revogá-lo a qualquer momento na sua [conta de usuário](#).

#### Passo 5 - Finalização das configurações:

Após esses processo o mesmo será gerado uma chave de testes, copie e cole no seu arquivo **sonar-project.properties**, caso não tenha crie na raiz do seu projeto:

Aqui é um exemplo usando em um projeto simples com HTML e JS:

```
sonar.projectKey=test_inicial
sonar.host.url=http://localhost:9000
sonar.language=js
sonar.sources=src
sonar.login=sqp_0d74af481be73236a4552ccd4ac098ea371e171c
```

#### Execute a análise:

Rodando o comando no terminal: **npx sonar-scanner**

#### Verificando os Resultados

Acesse o dashboard do SonarQube no navegador: <http://localhost:9000>

#### Tipos de Métricas:

- Erros no código
- Cobertura de testes
- Vulnerabilidades detectadas

#### Resultados de um Projeto Escaneado

Aqui podemos identificar que existem vários problemas e nessa situação seria urgente a revisão da estrutura do projeto.

Issues in new code			
▼ Type			
🐛 Bug			690
🔒 Vulnerability			0
💩 Code Smell			6.4k
▼ Severity			
🔴 Blocker	68	🟢 Minor	4.6k
🔴 Critical	729	📘 Info	0
🔴 Major	1.7k		

## Lista de Exercícios de Fixação

### Questões Dissertativas

1. Explique como a cultura DevOps melhora o ciclo de desenvolvimento de software. Quais são os principais benefícios da adoção dessa abordagem em empresas que possuem um modelo tradicional de desenvolvimento e operações separadas?
2. O GitHub Actions permite a automação de workflows dentro do GitHub. Descreva como ele pode ser utilizado para implementar um pipeline de CI/CD, incluindo etapas como testes, build e deploy.
3. O SonarQube é uma ferramenta essencial para garantir a qualidade do código. Explique como ele pode ser utilizado para melhorar a manutenibilidade de um projeto e evitar problemas técnicos a longo prazo.
4. Você foi encarregado de configurar uma notificação automatizada no Discord para avisar a equipe sempre que um commit for realizado no repositório. Explique quais são os passos necessários para configurar essa integração usando GitHub Actions e Webhooks do Discord.
5. Imagine que sua equipe precisa decidir entre refatorar um projeto legado ou criar um novo do zero. Explique como o SonarQube pode ajudar nessa decisão e quais métricas podem ser analisadas para avaliar a qualidade do código existente.