📘 Documentação do Projeto - CI/CD com Docker e GitHub Actions

🗒️ Projeto: Gestão de Resíduos (.NET + Docker + GitHub Actions)

Este projeto tem como objetivo demonstrar a aplicação de práticas DevOps com foco em CI/CD e containerização de uma aplicação .NET. A API foi desenvolvida para gestão de resíduos e organizada para funcionar em ambientes controlados via Docker, com deploy automatizado em staging e produção via GitHub Actions.

🔧 Etapas do Pipeline de CI/CD

O pipeline foi configurado usando o **GitHub Actions**, com as seguintes etapas:

1. **Disparo automático**:
    - Executado a cada push nos branches `main` (produção) e `staging`.
2. **Checkout do código**:
    - Usa a ação `actions/checkout` para obter o código do repositório.
3. **Configuração do .NET**:
    - Instala o SDK .NET 8 usando `actions/setup-dotnet`.
4. **Restaurar dependências**:
    - Executa `dotnet restore` para baixar pacotes do projeto.
5. **Compilação do projeto**:
    - Usa `dotnet build` para compilar o projeto sem restaurar novamente.
6. **Build da imagem Docker**:
    - Cria a imagem da aplicação a partir do Dockerfile.

7. **Autenticação no Docker Hub**:
    - Usa variáveis secretas (`DOCKER_USERNAME`, `DOCKER_PASSWORD`) do GitHub.
8. **Push da imagem Docker**:
    - A imagem é enviada com a tag do nome do branch atual (staging ou main).
9. **Simulação de Deploy**:
    - Um passo simples com `echo` para simular deploy com base no ambiente.

🐳 Estratégia de Containerização

A aplicação foi **containerizada com Docker** para garantir portabilidade entre ambientes. O Dockerfile está localizado dentro do projeto e permite:

- Definir o ambiente de execução (ASP.NET Core)
- Copiar os arquivos da API
- Executar o build da aplicação
- Rodar a aplicação em porta exposta

Para facilitar a orquestração, foi criado um arquivo `docker-compose.yml` que:

- Sobe o container da API
- Mapeia a porta 80 para a porta 5000 no host
- Define o ambiente como `Development`

🧠 Importância para o DevOps

A combinação de CI/CD + Docker + Orquestração garante:

- **Consistência** nos ambientes de desenvolvimento, homologação e produção

- **Rapidez no deploy** e integração contínua
- **Detecção precoce de erros** com builds automatizados
- **Padronização** dos processos operacionais

📸 Prints das Configurações e Execuções

Os prints foram organizados para demonstrar:

- Configuração do workflow no GitHub
- Execução dos jobs no GitHub Actions (Build e Deploy)
- Estrutura de pastas e arquivos (`.github/workflows`)
- Execução local com `docker-compose up`
- Teste da aplicação rodando em `http://localhost:5000`

(Você pode tirar essas capturas na sua própria tela e colar no final do documento)

✅ Conclusão

Este trabalho mostra, de forma prática, como é possível estruturar uma aplicação .NET moderna com foco em boas práticas DevOps. Com o uso de Docker e GitHub Actions, o ciclo de desenvolvimento fica mais ágil, seguro e automatizado.