UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01120 – TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS

**Trabalho Prático - Etapa 3 - Grupo 5**

Carolina Magagnin Wajner (00134101)
Gabriel Souza Lima (00584520)
Luísa Righi Bolzan (00578954)
Nicolas Chin Lee (00579322)
Rodrigo Salvadori Feldens (00578803)

Porto Alegre, Rio Grande do Sul

## 1. Implementação de um pipeline de Continuous Integration/Continuous Delivery-Deployment (CI/CD)

### 1.1. Ferramenta Utilizada

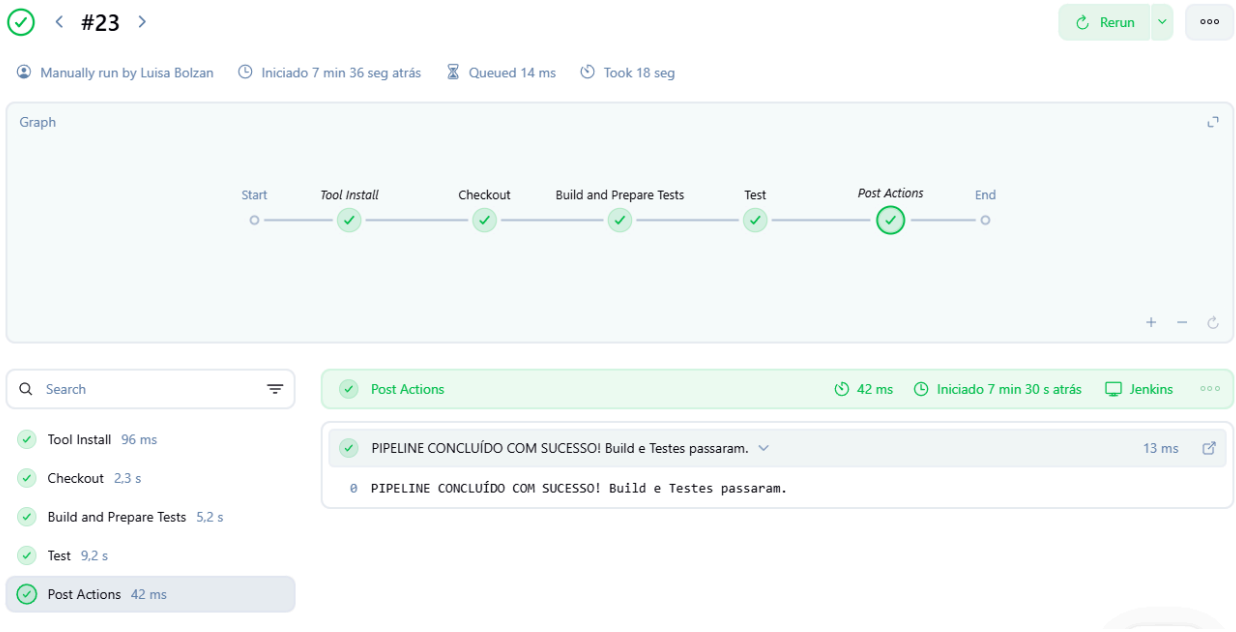
A ferramenta escolhida para a implementação do pipeline foi o **Jenkins**. A escolha se deu pelo objetivo de aprender a configurar e gerenciar uma ferramenta de CI/CD *self-hosted* (hospedada localmente).

### 1.2. Processo de Instalação e Configuração

1. **Instalação de Pré-requisitos:** Foi identificado que a versão LTS do Jenkins exigia Java JDK 17 ou 21 para sua execução. Foi instalado o JDK 21.
2. **Instalação do Servidor Jenkins:** O Jenkins foi baixado de seu site oficial e instalado utilizando o instalador *.msi* para Windows, que configurou a ferramenta como um serviço, garantindo sua execução automática em segundo plano.
3. **Configuração Inicial:** Após a instalação, o Jenkins foi acessado via navegador (http://localhost:8080). O processo inicial incluiu:
    - Desbloqueio da instância utilizando a senha forte em “initialAdminPassword”.
    - Instalação dos plugins sugeridos pela plataforma para garantir as funcionalidades essenciais.
    - Criação de um usuário administrador para o gerenciamento do servidor, o que evita de usar a senha forte inicial do Jenkins.
4. **Configuração de Ferramentas (JDK):** Para que o pipeline pudesse utilizar a versão correta do Java, foi necessário configurar o JDK 21 na seção "Manage Jenkins" > "Tools". Foi criada uma ferramenta do tipo JDK com o nome `JDK-21`, apontando para o seu diretório de instalação.
5. **Criação do Pipeline:** Foi criado um novo item do tipo "Pipeline". A lógica do pipeline foi definida diretamente na interface do Jenkins através de um script em Groovy, que automatiza todo o processo.

### 1.3. Demonstração de Sucesso do Pipeline

Conforme a imagem em anexo, o pipeline foi executado com sucesso. Isso indica que todas as etapas configuradas foram concluídas sem erros:




**Estágio "Checkout":** O Jenkins conectou-se com sucesso ao repositório no GitHub e baixou a versão mais recente do código-fonte do branch “main”.

**Estágio "Build":** O Jenkins executou o script “compilar.bat”, que é o método de compilação do projeto. A conclusão bem-sucedida deste estágio significa que todo o código-fonte está sintaticamente correto e foi compilado sem falhas.

**Estágio “Test”:** Todos os testes foram executados com êxito.

### 1.4. Demonstração de Falha do Pipeline

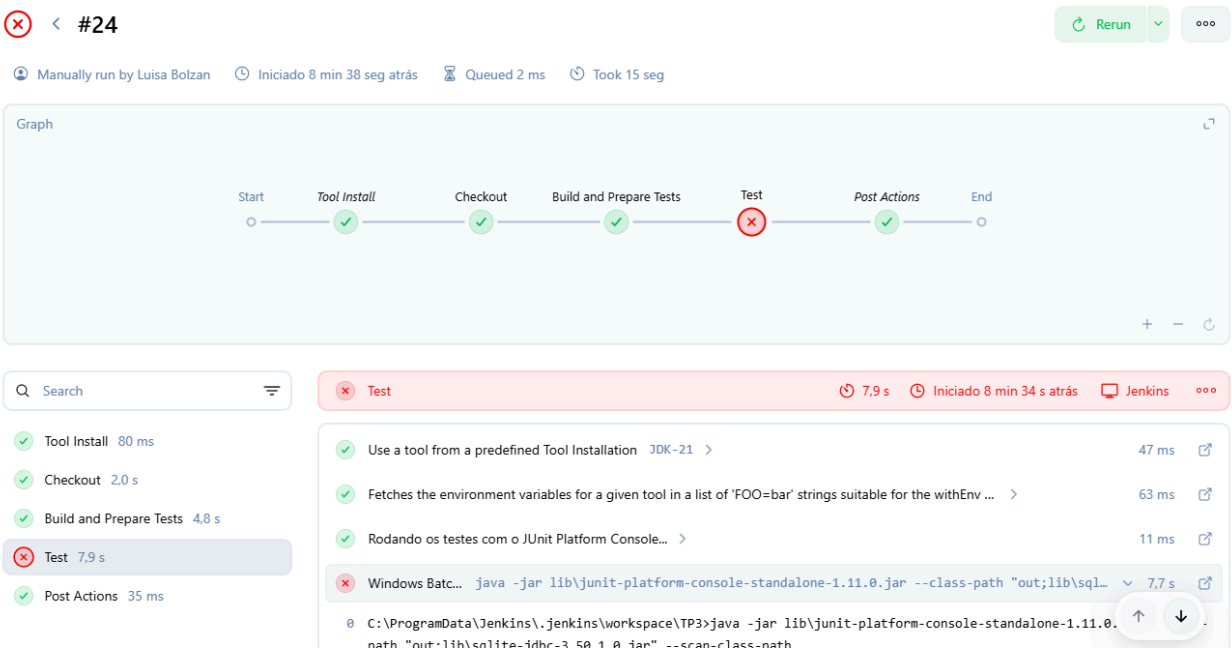
Para demonstrar a capacidade do pipeline de detectar regressões e bugs de lógica, foi executada a suíte de testes do projeto. O pipeline falhou precisamente no estágio "Test".

O log do JUnit Platform Console revelou que, dos 188 testes executados , 187 foram bem-sucedidos e exatamente **1 falhou**. A análise do log apontou o erro específico:

- **Teste com Falha:** PokemonDBTest:testUpdatePokemon()
- **Tipo de Erro:** org.opentest4j.AssertionFailedError: expected: <100.0> but was: <0.0>

Esta falha indica um bug de lógica na funcionalidade de atualização do banco de dados de Pokémons, onde o código retornou um valor (`0.0`) diferente do esperado pelo teste (`100.0`). O Jenkins imediatamente marcou a build como **"FAILURE"**, interrompendo o processo. Isso demonstra a principal função da Integração Contínua: garantir a qualidade e a corretude do código, prevenindo que bugs de lógica prossigam no ciclo de desenvolvimento.

Entretanto,  na prática esse erro não afeta o código. Tendo em vista que ao compilar o código para teste, são criado dois arquivos PokemonDBTest.class. Um dentro da pasta esperada e correto e outro fora da pasta e incorreto (esse arquivo deve ser excluído).




### 1.5. Dificuldades Encontradas

1. **Conflito de Dependências (Java):** A primeira dificuldade foi alinhar a versão do Java exigida pelo Jenkins (17 ou 21) com a versão utilizada pelo projeto, o que demandou a configuração de múltiplos JDKs na seção "Global Tool Configuration".
2. **Erros de Configuração do Jenkins:** Ocorreu um erro de "Tool 'JDK-21' not found", que foi solucionado garantindo que o nome da ferramenta no script era idêntico ao nome configurado na interface de administração do Jenkins, evidenciando a importância da precisão nas configurações.