DevOps e Integração Contínua/Entrega Contínua (CI/CD)

Disciplina: Engenharia de Software – Departamento De Telemática - IFCE

Período: 2025.1

Professor: César Olavo de Moura Filho

Objetivo Geral:

Capacitar os alunos a compreender os princípios e práticas de DevOps, bem como a aplicar os conceitos de Integração Contínua (CI) e Entrega Contínua (CD) através da implementação prática de um pipeline automatizado em uma ferramenta de CI/CD.

Modalidade:

Trabalho em Equipe (mínimo de 3, máximo de 5 integrantes).

Estrutura do Trabalho:

O trabalho será dividido em duas partes principais: Teórica e Prática.

Parte 1: Teórica - Relatório de Pesquisa e Análise (Peso: 40%)

Conteúdo:

O relatório deve abordar os seguintes tópicos de forma aprofundada, com clareza, concisão e fundamentação teórica robusta:

1. Introdução ao DevOps:
    - Definição e princípios fundamentais do DevOps.
    - Cultura DevOps: pilares (cultura, automação, lean, medição, compartilhamento).
    - Benefícios da adoção de DevOps (velocidade, qualidade, segurança, estabilidade).
    - Desafios na implementação de DevOps.
    - Ferramentas comuns no ecossistema DevOps (breve panorama).
2. Integração Contínua (CI):
    - Definição e objetivos da CI.
    - Princípios da CI (controle de versão, build automatizado, testes automatizados, feedback rápido).
    - Benefícios e desafios da CI.
    - Como a CI se integra com o controle de versão (Git/GitHub/GitLab/Bitbucket).
3. Entrega Contínua (CD) e Implantação Contínua (CD):
    - Definição de Entrega Contínua (Continuous Delivery) e Implantação Contínua (Continuous Deployment).
    - Diferenças e semelhanças entre os dois conceitos.

◦ Etapas de um pipeline de CD (build, teste, empacotamento, implantação em ambientes).

◦ Benefícios e desafios da CI/CD.

◦ Conceitos como "build once, deploy many" e "release train".

4. Ferramentas de CI/CD:

◦ Apresentação e comparação de pelo menos três (3) ferramentas populares de CI/CD, como:

▪ Jenkins

▪ GitLab CI/CD

▪ GitHub Actions

▪ Azure DevOps Pipelines

▪ CircleCI

▪ Travis CI

▪ Bitbucket Pipelines

◦ Para cada ferramenta, descrever:

▪ Suas principais características e vantagens.

▪ Sua arquitetura básica (se aplicável).

▪ Prós e contras de sua utilização em diferentes contextos.

◦ Justificativa da escolha da ferramenta a ser utilizada na parte prática do trabalho.

Formato do Relatório:

• Padrão ABNT: Formato de trabalho acadêmico (capa, folha de rosto, sumário, introdução, desenvolvimento, conclusão, referências).

• Extensão: Mínimo de 10 páginas e máximo de 15 páginas (excluindo capa, sumário, referências e anexos).

• Referências: Mínimo de 5 fontes bibliográficas (artigos científicos, livros, documentação oficial das ferramentas).

• Entrega: Documento em formato PDF.

Parte 2: Prática - Desenvolvimento de um Workflow de CI/CD (Peso: 60%)

Requisito Central:

A equipe deve desenvolver um pipeline de CI/CD para uma aplicação de software, utilizando uma das ferramentas de CI/CD pesquisadas na parte teórica.

Cenário da Aplicação (Sugestões):

• Aplicação Web Simples: Um aplicativo web em Python (Flask/Django), Node.js (Express), Java (Spring Boot), ou uma aplicação front-end (React, Angular, Vue.js).

• Aplicação de Linha de Comando: Um pequeno script ou utilitário que possa ser testado e "empacotado".

• **Observação**: O foco não é na complexidade da aplicação, mas na implementação do pipeline. A aplicação deve ser simples o suficiente para permitir que o pipeline seja o centro do trabalho.

Etapas Mínimas do Workflow de CI/CD:

O pipeline deve conter, no mínimo, as seguintes etapas automatizadas:

1. Controle de Versão:
   - ◦ O código da aplicação deve estar hospedado em um repositório Git (GitHub, GitLab, Bitbucket, Azure Repos).
   - ◦ Todas as alterações devem ser feitas via branches e pull requests (ou merge requests).
2. Integração Contínua (CI):
   - ◦ Trigger: O pipeline deve ser acionado automaticamente em cada push para o repositório (ou em pull requests).
   - ◦ Build: Compilação do código-fonte (se aplicável, ex: Java, TypeScript) ou verificação de sintaxe/linter (para linguagens interpretadas).
   - ◦ Testes Automatizados: Execução de testes unitários e/ou de integração. A equipe deve criar alguns testes simples para a aplicação.
   - ◦ Análise de Qualidade de Código (Opcional, mas altamente recomendado): Integração com ferramentas como SonarQube (auto-hospedado ou via cloud), ESLint, etc. (se for o caso).
3. Entrega Contínua (CD):
   - ◦ Empacotamento/Containerização: Criação de um artefato de deploy (ex: arquivo .jar, .zip, imagem Docker). É fortemente recomendado o uso de Docker para containerizar a aplicação. Quem não souber o que é docker, aproveite a oportunidade para aprender esse assunto essencial em um ambiente real de desenvolvimento! Esse aprendizado pode demorar. Comece hoje!
   - ◦ Implantação em Ambiente de Teste/Homologação: O pipeline deve implantar automaticamente a aplicação em um ambiente de teste simulado ou real. Sugestões de ambiente:
     - ▪ Docker Compose: Para implantação local de múltiplos contêineres.
     - ▪ Serviço de PaaS (Platform as a Service) gratuito/trial: Heroku, Vercel, Netlify, Render (para apps simples), Firebase, Supabase, etc.
     - ▪ Máquina Virtual/Servidor de baixo custo: Se a equipe tiver acesso.
     - ▪ Kubernetes (K3s/Minikube): Se a equipe tiver conhecimento prévio e/ou tempo para explorar.
   - ◦ Testes de Aceitação/Funcionais (Opcional): Execução de testes automatizados no ambiente de homologação após a implantação.

Artefatos da Parte Prática:

1. Repositório Git:
   - ◦ O repositório deve conter o código-fonte da aplicação.

◦ O arquivo de configuração do pipeline de CI/CD (ex: .gitlab-ci.yml, .github/workflows/*.yml, Jenkinsfile).

◦ README.md detalhado explicando:

▪ Como rodar a aplicação localmente.

▪ Como configurar e disparar o pipeline de CI/CD.

▪ Link para o relatório teórico.

▪ Links para o(s) ambiente(s) de deploy (se a aplicação estiver online).

2. Demonstração:

◦ Vídeo: Um vídeo de 5-10 minutos, gravado pela equipe, demonstrando o pipeline em execução. O vídeo deve mostrar:

▪ Uma alteração no código que aciona o pipeline.

▪ As etapas do pipeline sendo executadas na ferramenta de CI/CD.

▪ O resultado dos testes e da implantação.

▪ A aplicação funcionando no ambiente de destino.

◦ Apresentação oral: A equipe fará uma apresentação de 10-15 minutos em sala, explicando o trabalho teórico e demonstrando o pipeline na prática.

Avaliação:

• Parte Teórica (40%):

◦ Conteúdo e profundidade da pesquisa.

◦ Clareza, organização e coesão do texto.

◦ Conformidade com as normas ABNT.

◦ Qualidade das referências bibliográficas.

• Parte Prática (60%):

◦ Funcionalidade do pipeline (todas as etapas mínimas implementadas e funcionando).

◦ Qualidade do código da aplicação (simplicidade, legibilidade).

◦ Adequação da ferramenta de CI/CD escolhida.

◦ Qualidade da documentação no README.md.

◦ Qualidade da demonstração (vídeo e apresentação oral).

◦ Uso efetivo do controle de versão pela equipe.

◦ (Bônus) Integrações adicionais (análise de código, notificações, etc.).

Cronograma:

• 10/06: Lançamento do trabalho e formação das equipes.

• 24/06: Data limite para definição da ferramenta de CI/CD e da aplicação a ser desenvolvida.

• 01/07: Entrega parcial: Esboço do relatório teórico e arquitetura do pipeline.

• 15/07: Entrega Final (Relatório PDF + Link para o Repositório Git + Link para o Vídeo de Demonstração)

• 22/07: Apresentações orais em sala de aula.

Orientações Finais:

• Colaboração: Utilize as ferramentas de controle de versão (Git) de forma colaborativa. O histórico de commits da equipe será considerado na avaliação. Necessário pelo menos um commit de cada membro da equipe.

• Dúvidas: Não hesitem em procurar o professor para tirar dúvidas e obter orientações ao longo do desenvolvimento do trabalho.

• Recursos: Explore a documentação oficial das ferramentas, tutoriais online e exemplos de projetos de CI/CD.

Este trabalho proporcionará uma experiência prática valiosa na implementação de práticas modernas de desenvolvimento de software, preparando-os para os desafios do mercado de trabalho.

Observação importante: não caiam na tentação de “lotear” as tarefas de modo a deixar as partes teóricas com alunos que não têm costume com atividades práticas ao computador e as partes práticas com quem tem. Ao contrário! Esta é uma boa oportunidade de nivelar competências e aprender no protegido ambiente universitário esta importante e necessária habilidade que pode constituir o diferencial do aluno no ambiente de trabalho!

Boa sorte!