Integração Contínua e Entrega Contínua (CI/CD)

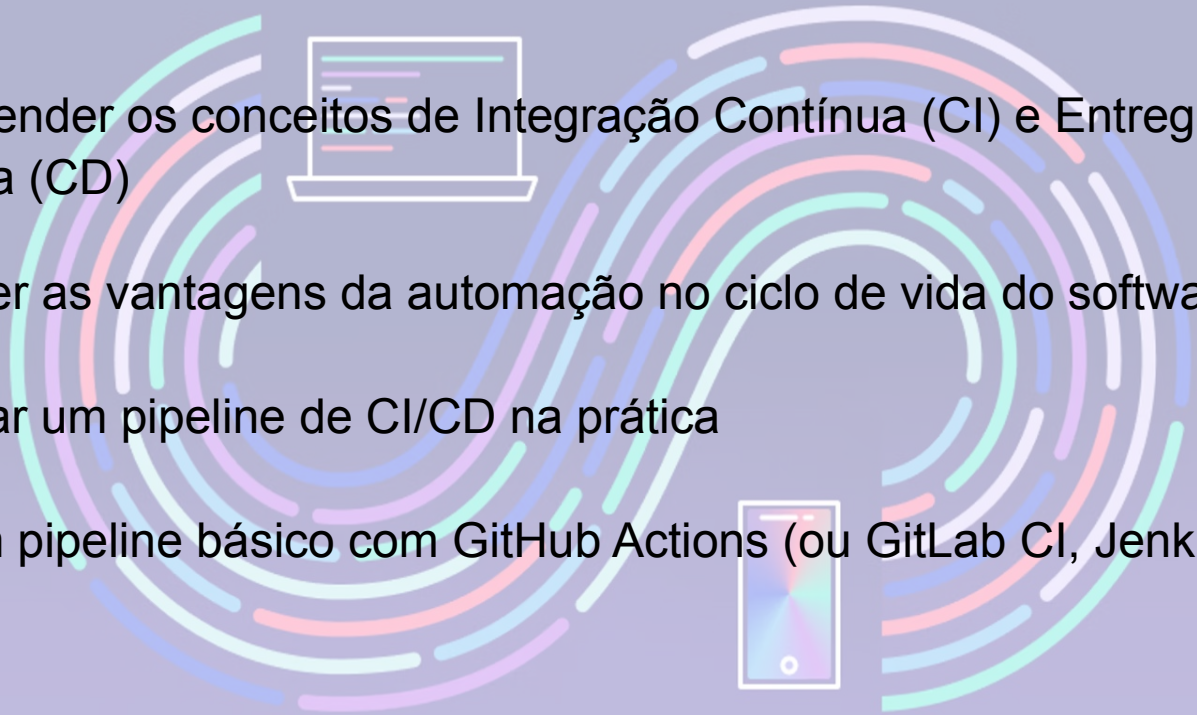
Aula 09



Prof. Msc. Francisco de Assis de Souza Rodrigues

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🎯 Objetivos



- Compreender os conceitos de Integração Contínua (CI) e Entrega Contínua (CD)
- Conhecer as vantagens da automação no ciclo de vida do software
- Visualizar um pipeline de CI/CD na prática
- Criar um pipeline básico com GitHub Actions (ou GitLab CI, Jenkins, etc.)

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

O que é CI (Integração Contínua)?

CI é uma prática do desenvolvimento de software onde os desenvolvedores integram frequentemente suas alterações de código em um repositório compartilhado. Cada integração é verificada por meio de **builds automáticos** e **testes automatizados**, permitindo detectar erros rapidamente.

Palavras-chave:

- Commits frequentes
- Automatização de build e testes
- Integração imediata



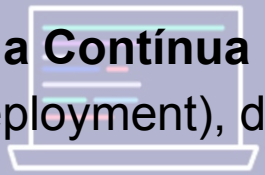
💡 *Analogia:* É como revisar um trabalho em grupo constantemente, ao invés de esperar todos terminarem suas partes para juntar no final — o que quase sempre dá errado.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

O que é CD (Entrega Contínua / Deploy Contínuo)?

CD pode significar **Entrega Contínua** (Continuous Delivery) ou **Deploy Contínuo** (Continuous Deployment), dependendo do grau de automação.

- **Entrega Contínua:** o sistema está sempre pronto para ser colocado em produção, mas o deploy é acionado manualmente.
- **Deploy Contínuo:** o deploy acontece automaticamente sempre que um novo commit passa pelos testes.





Resumo:

- CI = integrações frequentes + testes automatizados
- CD = entrega frequente (e opcionalmente automática) em produção

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

❌ Problemas comuns antes da adoção de CI/CD



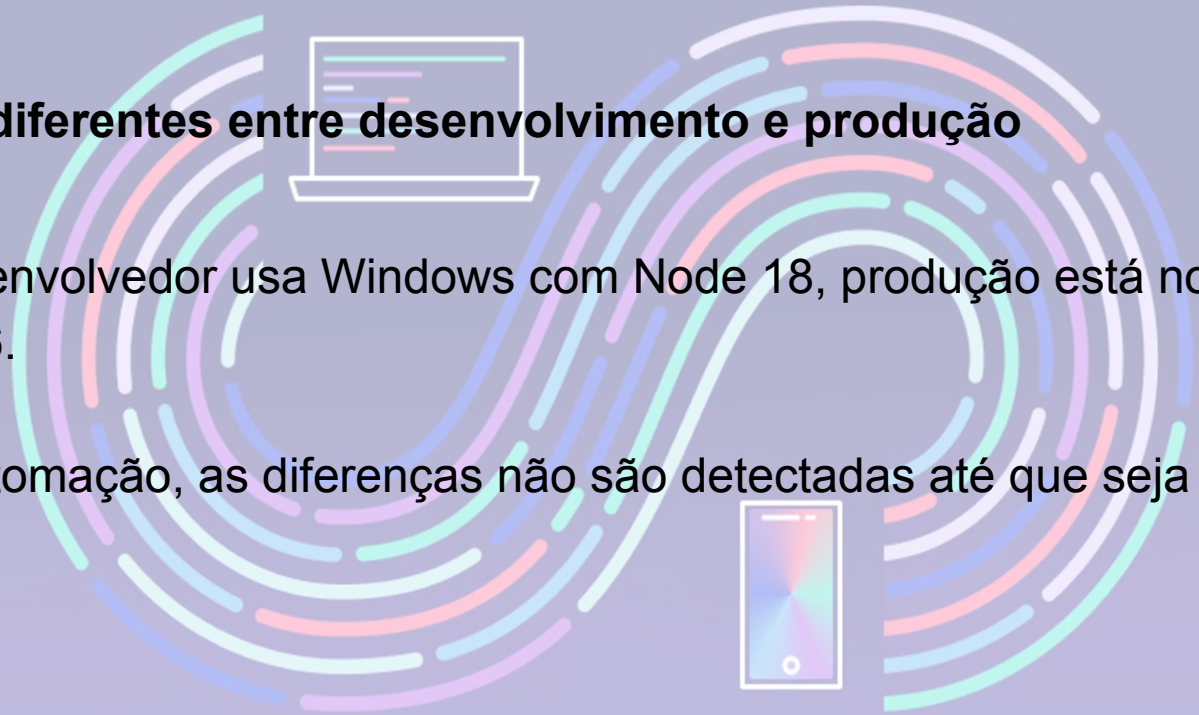
Integração tardia (Big Bang Integration)

- Equipes trabalham semanas ou meses em funcionalidades separadas e só integram tudo de uma vez no fim — gerando **conflitos complexos**, **bugs** e **re-trabalho**.

- “Funciona na minha máquina” vira frase comum.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

❌ Problemas comuns antes da adoção de CI/CD



Ambientes diferentes entre desenvolvimento e produção

- Ex: desenvolvedor usa Windows com Node 18, produção está no Linux com Node 16.
- Sem automação, as diferenças não são detectadas até que seja tarde demais.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

❌ Problemas comuns antes da adoção de CI/CD



Deploys manuais e quebradiços

- Requerem uma pessoa com conhecimento específico para fazer deploy (geralmente fora do horário de trabalho 😓).
- Comando errado ou etapa esquecida = sistema fora do ar.
- É difícil refazer um deploy ou auditar o que mudou.



UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🌟 Benefícios da automação (CI/CD)



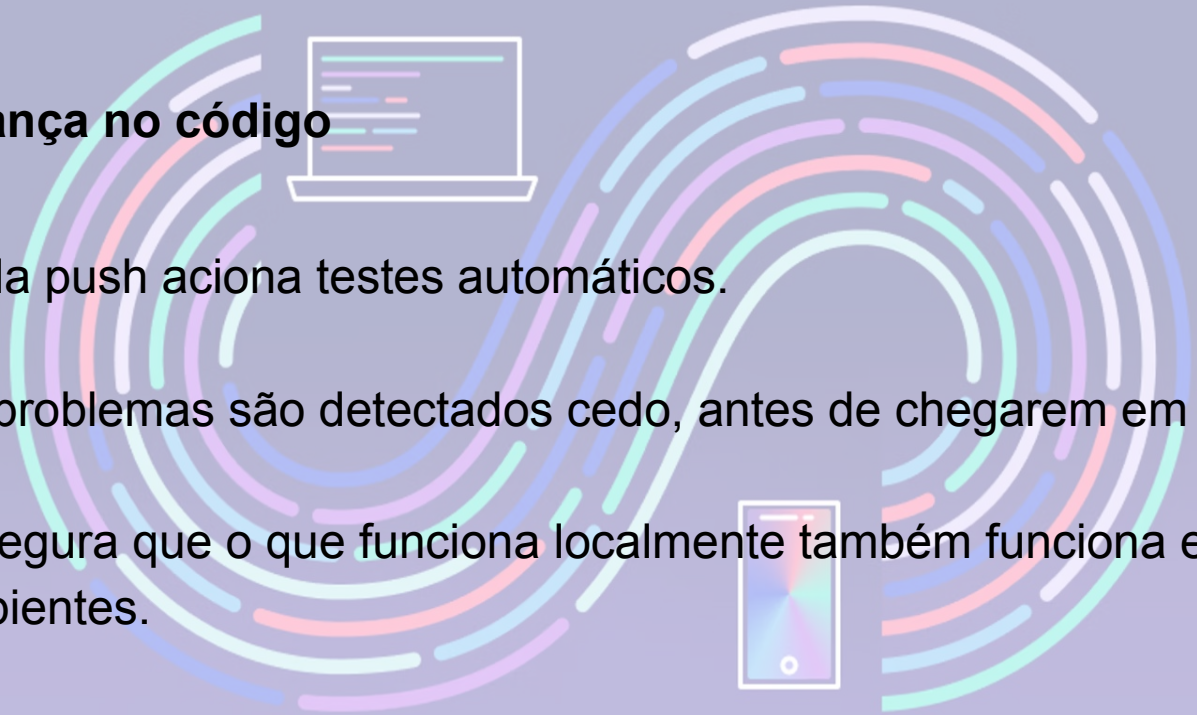
Menor tempo de entrega

- Com CI/CD, a entrega de novas funcionalidades ou correções de bugs é mais rápida e previsível.
- Diminui o tempo entre “escrever o código” e “colocar no ar”.



UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🌟 Benefícios da automação (CI/CD)



Maior confiança no código

- Cada push aciona testes automáticos.
- Os problemas são detectados cedo, antes de chegarem em produção.
- Assegura que o que funciona localmente também funciona em outros ambientes.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🌟 Benefícios da automação (CI/CD)



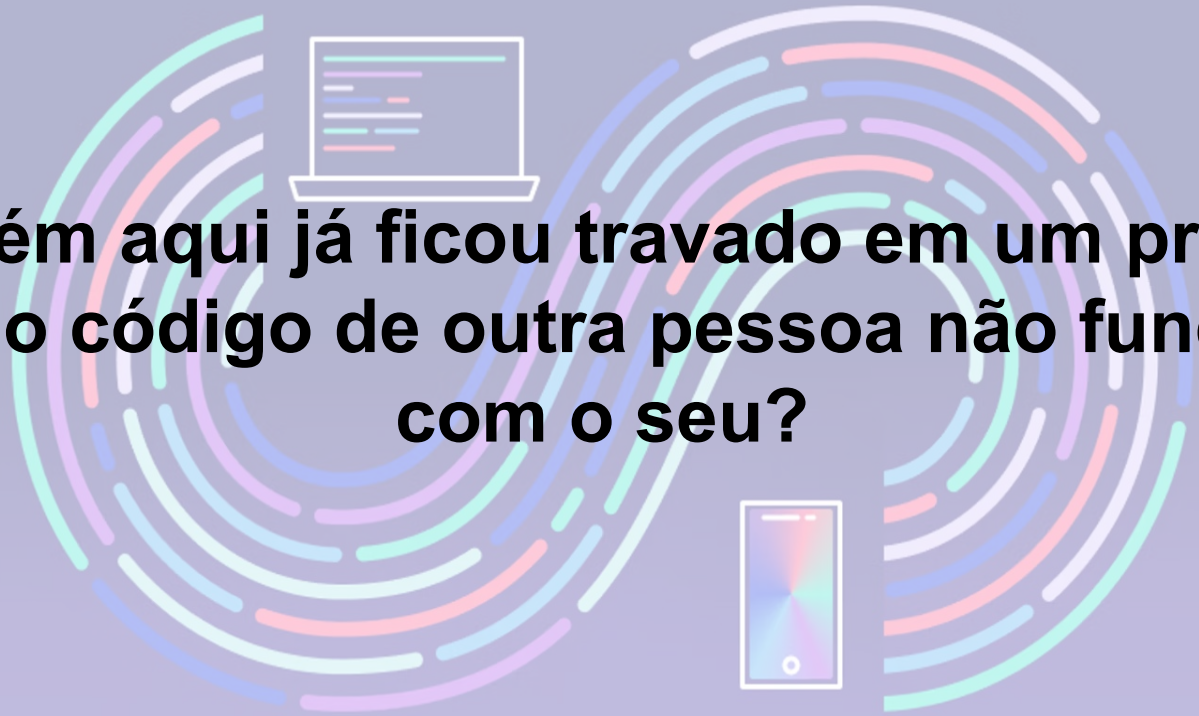
Feedback rápido

- Se algo quebra, você descobre logo após o push.
- Permite corrigir imediatamente e seguir com o desenvolvimento.



📈 *Empresas que adotam CI/CD com maturidade conseguem fazer dezenas ou centenas de deploys por dia sem comprometer a qualidade.*

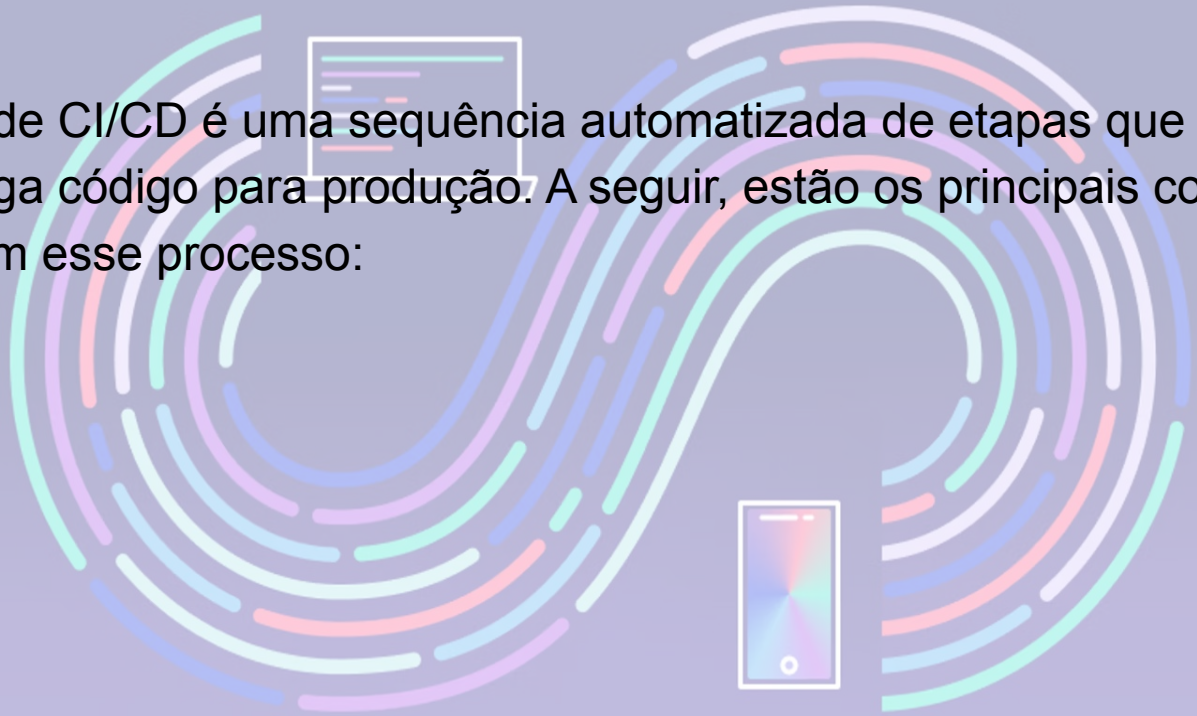
UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO



Alguém aqui já ficou travado em um projeto porque o código de outra pessoa não funcionava com o seu?

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Componentes de um Pipeline de CI/CD



Um pipeline de CI/CD é uma sequência automatizada de etapas que prepara, testa e entrega código para produção. A seguir, estão os principais componentes que compõem esse processo:

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🔁 1. Versionamento (Git)

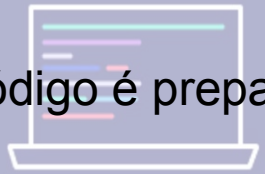
O versionamento é a base do CI/CD. Todo o código-fonte precisa estar em um sistema de controle de versões — geralmente o **Git** — para que as mudanças possam ser acompanhadas, revertidas e acionem os pipelines.

- **Branching**: prática comum de trabalhar em branches (como feature/, hotfix/, develop) e abrir pull requests para revisão.
- **Triggers**: sistemas de CI/CD são geralmente configurados para iniciar automaticamente com base em eventos Git, como:
  - push para a branch main
  - pull_request para revisão de código

📌 *Exemplo: Um push para main aciona testes automatizados no pipeline.*

UniFAP

⚙️ 2. Build (Compilação, Empacotamento)



É o momento em que o código é preparado para ser executado:

- **Compilação**: para linguagens compiladas (Java, C#, Go), essa etapa transforma o código em binários.
- **Instalação de dependências**: linguagens interpretadas (Python, JavaScript) precisam instalar bibliotecas antes de rodar.
- **Lint e formatação**: checagem de estilo de código também pode ser incluída nessa etapa.



📌 *Exemplo: npm install seguido de npm run build em projetos JavaScript.*

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

✅ 3. Testes Automatizados



Testes são essenciais para garantir a qualidade e evitar regressões. Eles podem ser:

- **Unitários**: testam partes isoladas do código (funções, métodos)
- **De integração**: testam como os componentes do sistema funcionam juntos
- **End-to-end (E2E)**: simulam o comportamento do usuário final
- **Estáticos**: análise de código para verificar padrões e falhas

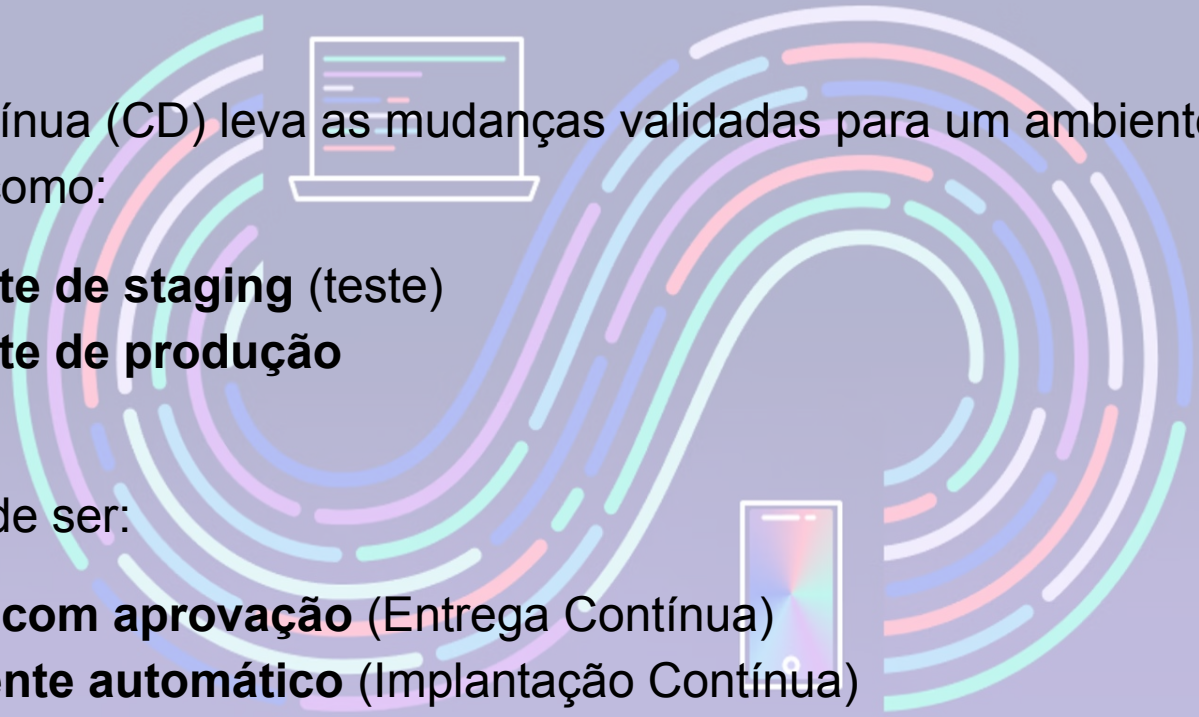


Esses testes são executados automaticamente após o build. Se algum falhar, o pipeline é interrompido.

📌 *Exemplo: pytest (Python), npm test (JavaScript), mvn test (Java)*

UniFAP
CENTRO UNIVERSITÁRIO ADVENTISTA

🚀 4. Deploy Automatizado (CD)



Entrega contínua (CD) leva as mudanças validadas para um ambiente controlado, como:

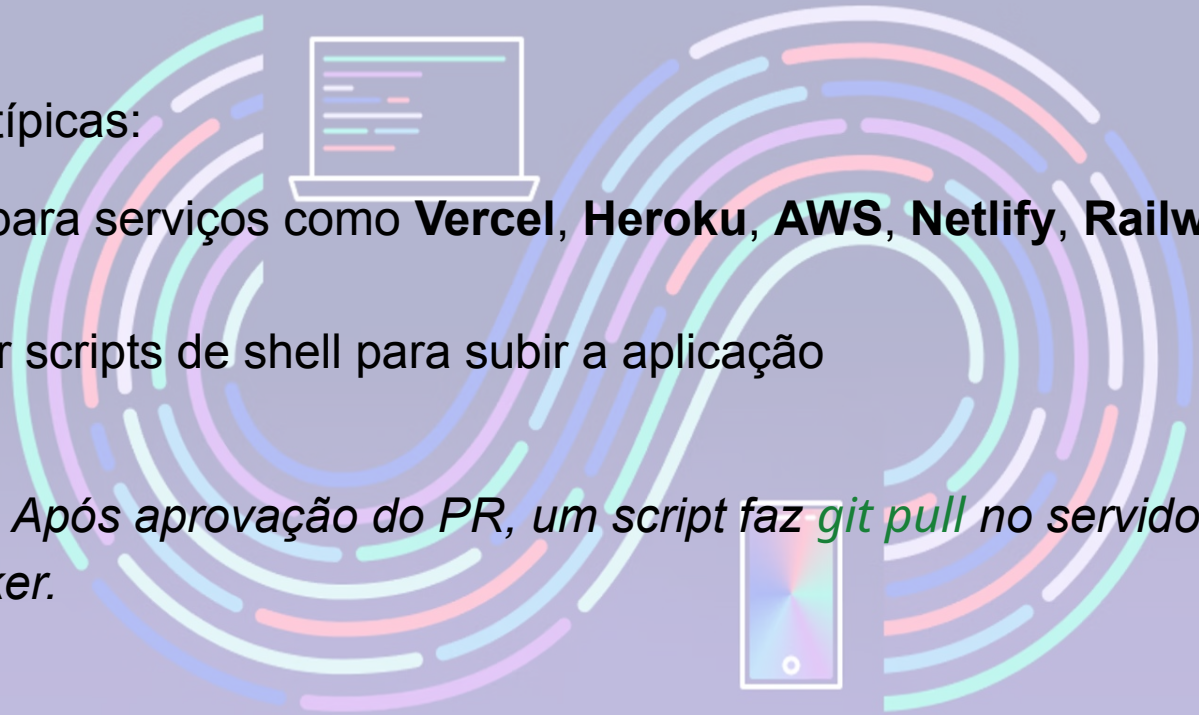
- **Ambiente de staging** (teste)
- **Ambiente de produção**

O deploy pode ser:

- **Manual com aprovação** (Entrega Contínua)
- **Totalmente automático** (Implantação Contínua)

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🚀 4. Deploy Automatizado (CD)



Integrações típicas:

- Deploy para serviços como **Vercel**, **Heroku**, **AWS**, **Netlify**, **Railway**
- Executar scripts de shell para subir a aplicação

📌 *Exemplo: Após aprovação do PR, um script faz git pull no servidor e reinicia o app via Docker.*

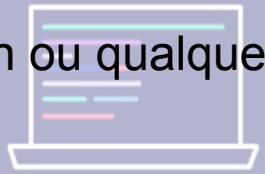
UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🛠️ 5. Ferramentas Comuns de CI/CD

Ferramenta	Descrição
GitHub Actions	Nativo do GitHub. Usa arquivos YAML no repositório para definir pipelines. Gratuito com limites.
GitLab CI	Integrado ao GitLab. Suporta runners locais e cloud. YAML também.
Jenkins	Muito flexível, open-source, mas exige configuração e manutenção própria.
Travis CI	Famoso em projetos open-source. YAML, integração com GitHub.
CircleCI	Foco em performance e integração com containers. Tem planos gratuitos.

UniFAP

Prática



Projeto simples em Python ou qualquer app com testes

Etapas da demo:

1. Repositório no GitHub
2. Criar .github/workflows/ci.yml
3. Configurar etapas:
    - checkout
    - setup da linguagem (ex: Node.js, Python)
    - install dependencies
    - run tests
4. Push e análise do workflow em ação



UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

📌 Exemplo simples de GitHub Actions

.github

workflow

main.yaml



yaml

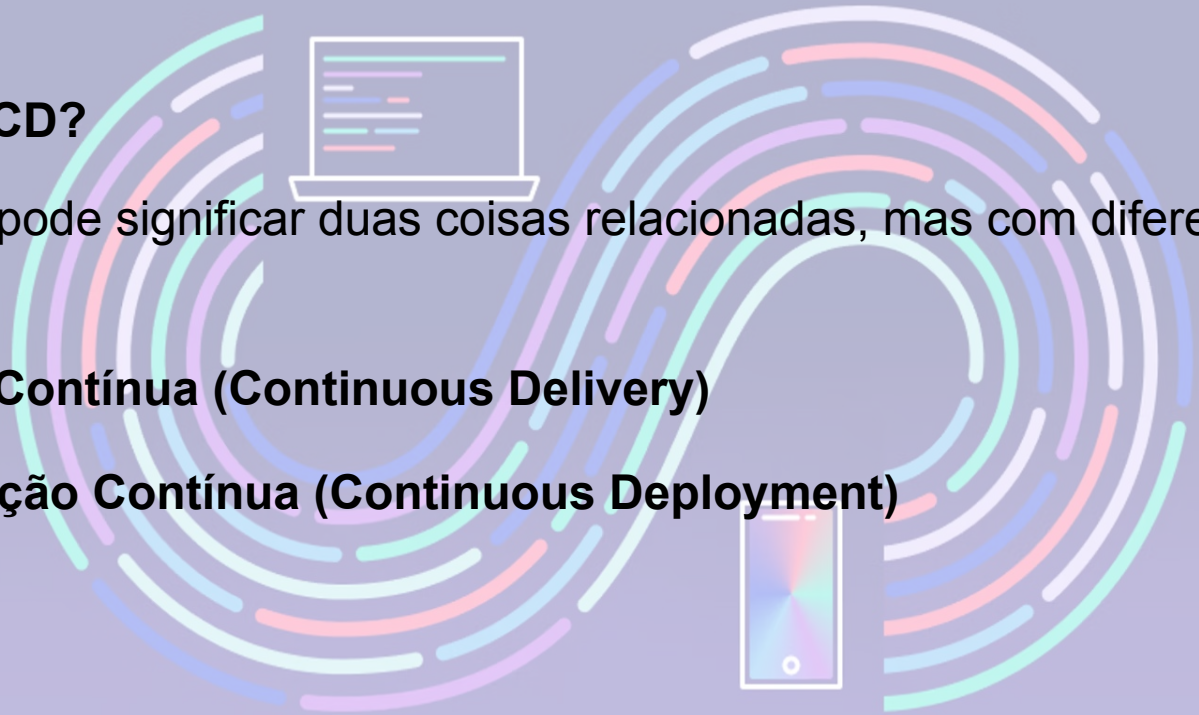
```yaml
name: CI Pipeline

on:
  push:
    branches: [ "main" ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
```

Conceito de CD – Entrega Contínua vs Implantação Contínua



🧩 O que é CD?

O termo CD pode significar duas coisas relacionadas, mas com diferenças importantes:

🚚 Entrega Contínua (Continuous Delivery)

🚀 Implantação Contínua (Continuous Deployment)

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🚚 Entrega Contínua (Continuous Delivery)

Definição:

Processo onde o software está sempre em um estado pronto para ser colocado em produção, com testes e validações automatizadas. Porém, a publicação em produção ainda exige uma ação manual (como aprovação ou clique em "Deploy").

Características:

- Código testado e validado em ambiente de staging
- Deploy automático em ambientes de teste
- Deploy em produção é **manual**, mas rápido e previsível
- Permite controle de *quando* algo é publicado

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🚚 Entrega Contínua (Continuous Delivery)



Exemplo de ferramentas usadas:

- GitHub Actions com deploy para *staging*
- GitLab CI com ambientes separados

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🚀 Implantação Contínua (Continuous Deployment)

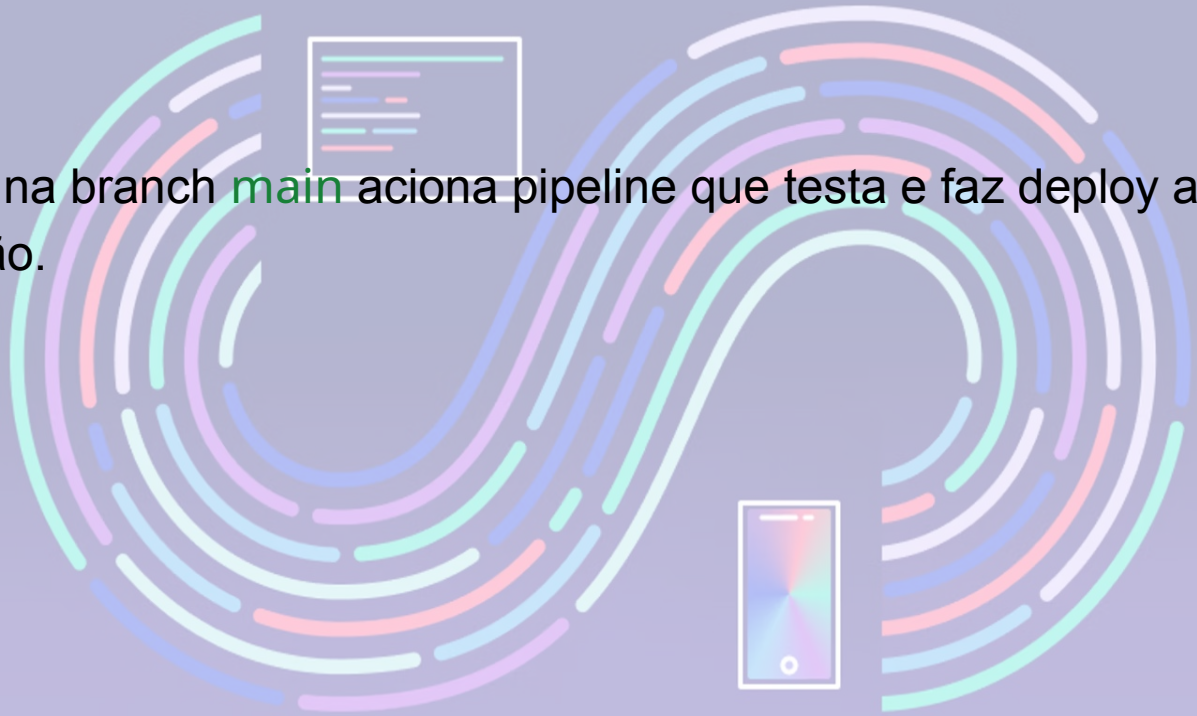
Definição:
É o próximo passo da Entrega Contínua: **todo código aprovado no repositório principal vai direto para produção automaticamente**, sem intervenção humana.

Características:

- Cada mudança é publicada em produção
- Reduz tempo de feedback de usuários reais
- Requer altíssimo nível de confiança na automação

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🚀 Implantação Contínua (Continuous Deployment)

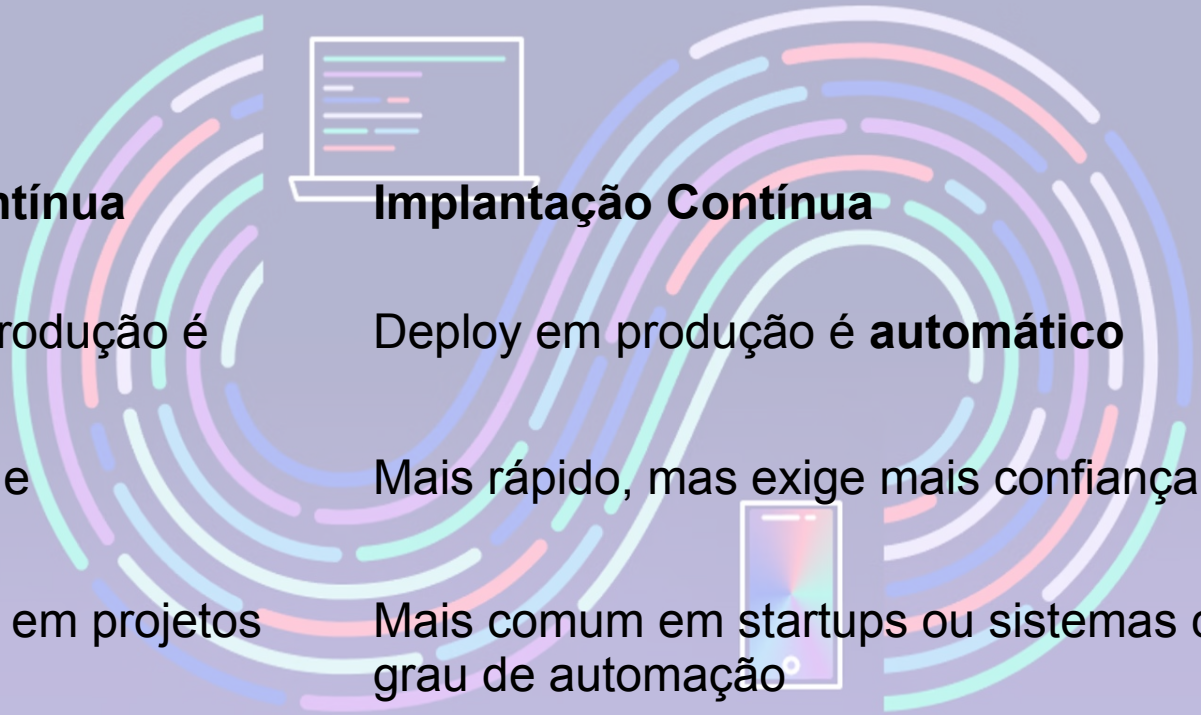


Exemplo:

Toda *merge* na branch main aciona pipeline que testa e faz deploy automático para produção.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

⚖️ Diferença principal



Entrega Contínua	Implantação Contínua
Deploy em produção é **manual**	Deploy em produção é **automático**
Maior controle	Mais rápido, mas exige mais confiança
Mais comum em projetos tradicionais	Mais comum em startups ou sistemas com alto grau de automação

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

🧪 Exemplo com Deploy Automático (Implantação Contínua)



Front-end:

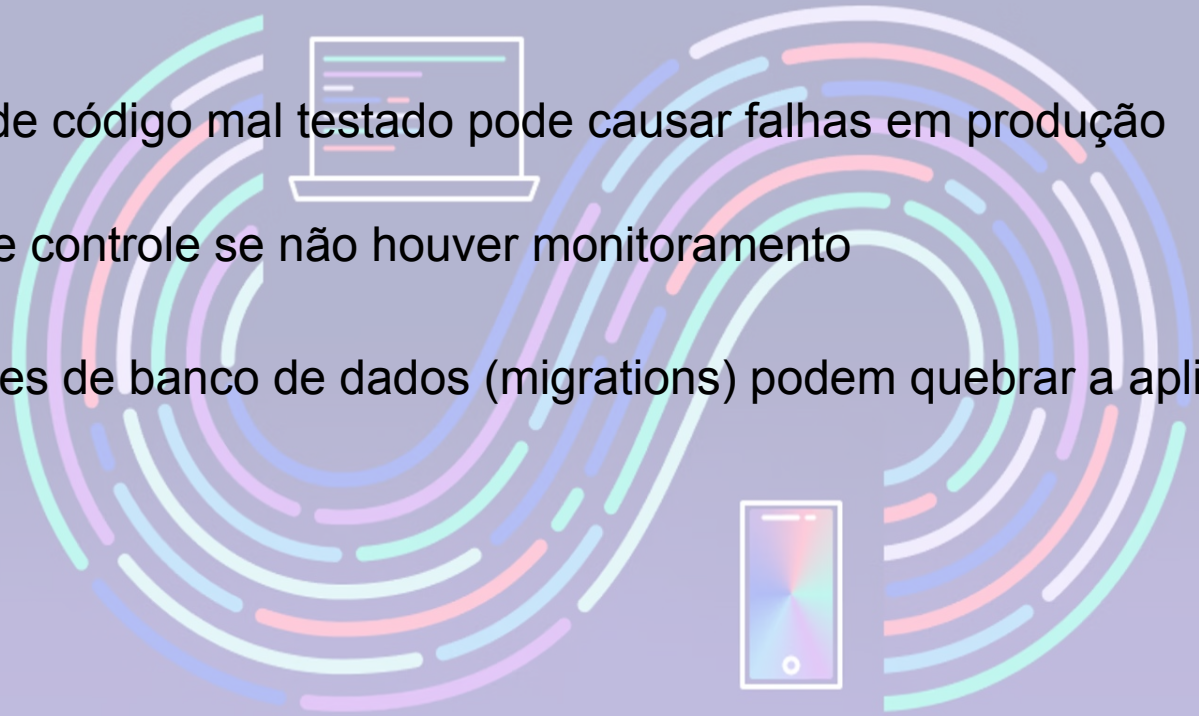
- **Vercel ou Netlify**
  - Deploy automático em cada push na main
  - Preview para pull requests
  - Ideal para apps React, Next.js, Vue etc.

Back-end:

- **Heroku ou Railway**
  - Integração com GitHub
  - Push no main → build automático → deploy
  - Pode incluir variáveis de ambiente, banco de dados etc.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

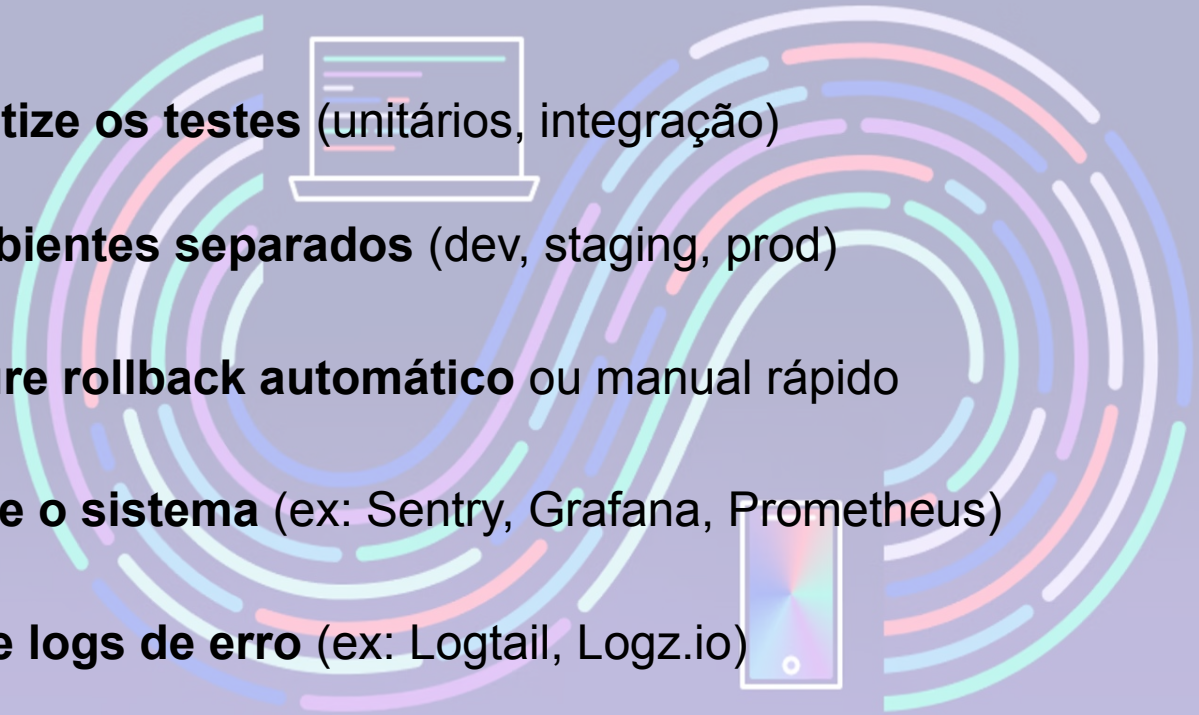
⚠️ Riscos do Deploy Automático



- Deploy de código mal testado pode causar falhas em produção
- Perda de controle se não houver monitoramento
- Alterações de banco de dados (migrations) podem quebrar a aplicação

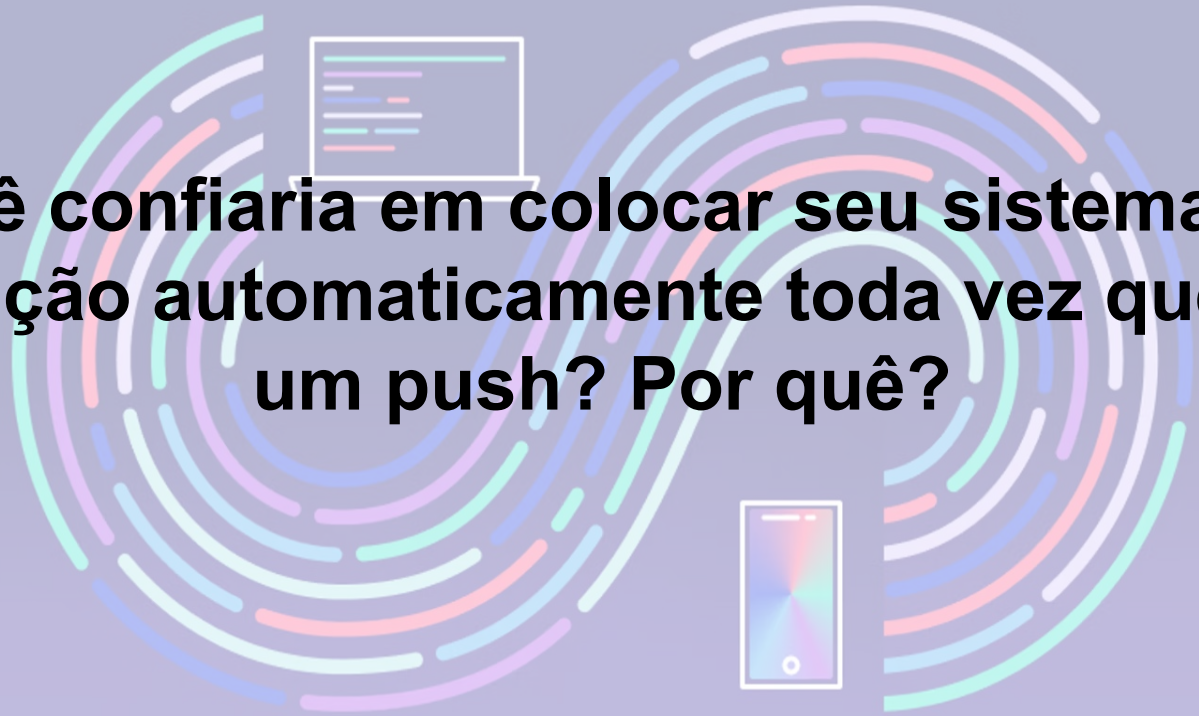
UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

✅ Boas práticas no deploy automático



- **Automatize os testes** (unitários, integração)
- **Use ambientes separados** (dev, staging, prod)
- **Configure rollback automático** ou manual rápido
- **Monitore o sistema** (ex: Sentry, Grafana, Prometheus)
- **Registre logs de erro** (ex: Logtail, Logz.io)
- **Use feature flags** para liberar recursos gradualmente

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO



Você confiaria em colocar seu sistema em produção automaticamente toda vez que fizer um push? Por quê?

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

Bibliografia

MORAES, Gleicon. Caixa de ferramentas devops: um guia para construção, administração e arquitetura de sistemas modernos. São Paulo, SP: Casa do Código, 2015. E-book. Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/212656 . Acesso em: 09 maio 2025.



Bibliografia

SATO, Danilo. Devops na prática: entrega de software confiável e automatizada. São Paulo, SP: Casa do Código, 2014. E-book. Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/212888. Acesso em: 09 maio 2025.

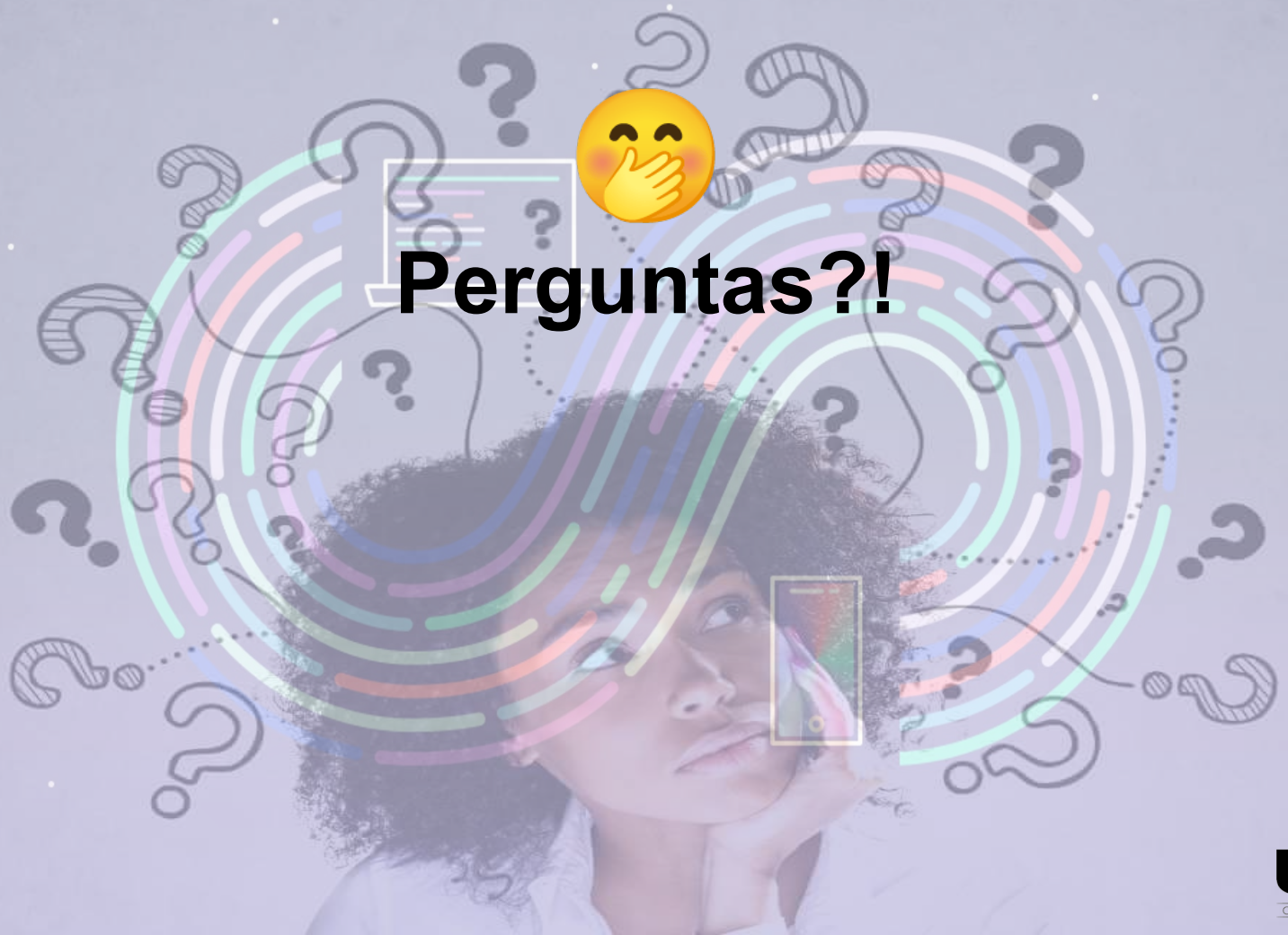


Referências

CHATGPT. **Fundamentos de Inteligência Artificial**. OpenAI, 2025. Disponível em: https://chat.openai.com. Acesso em: *20/03/2025*.

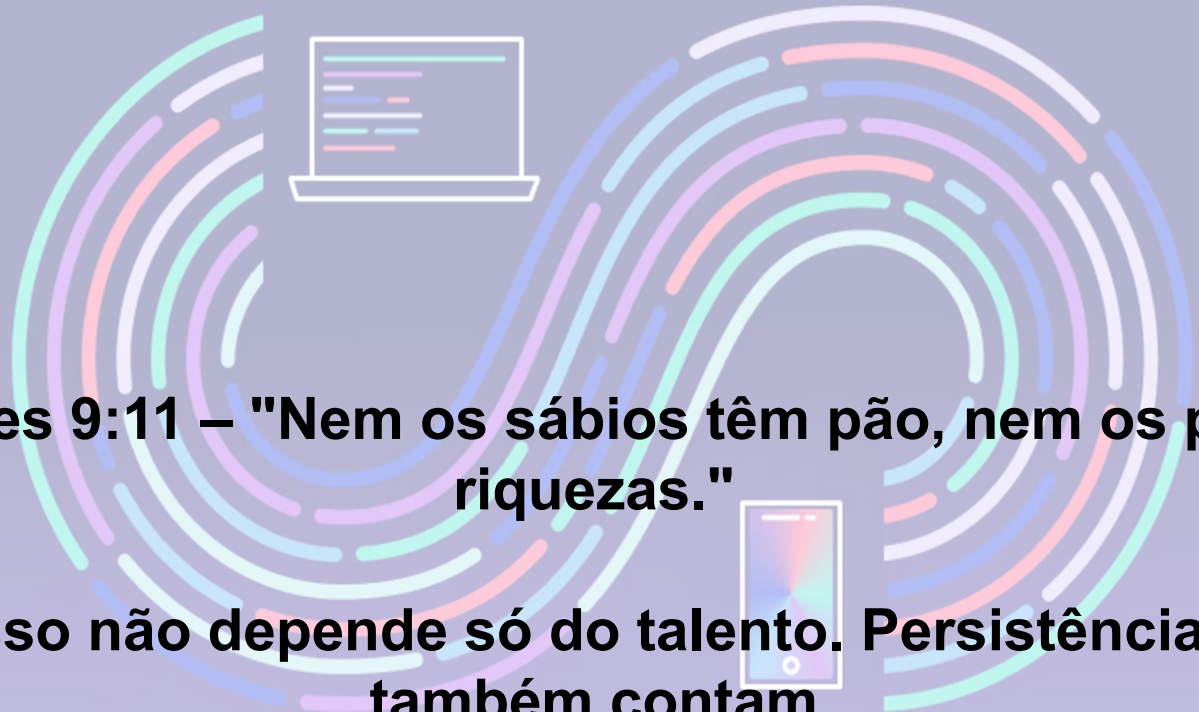


UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO



Perguntas?!

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO



Eclesiastes 9:11 – "Nem os sábios têm pão, nem os prudentes riquezas."

O sucesso não depende só do talento. Persistência e graça também contam.

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO