5954024 - Introdução ao Desenvolvimento Web

CI/CD no Desenvolvimento Web

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Explicar o que são CI, CD e pipeline
- Diferenciar integração contínua, entrega contínua e implantação contínua
- Compreender o papel do GitHub Actions em projetos Web
- Criar um workflow simples para build, teste e deploy
- Identificar boas práticas e erros comuns em pipelines

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Motivação

Considere uma equipe desenvolvendo um site institucional.
Cada pessoa altera uma parte:

- Uma pessoa muda o CSS
- Outra altera componentes JavaScript
- Outra mexe no formulário de contato
- Outra atualiza dependências

Pergunta: Como garantir que tudo continua funcionando antes de publicar?

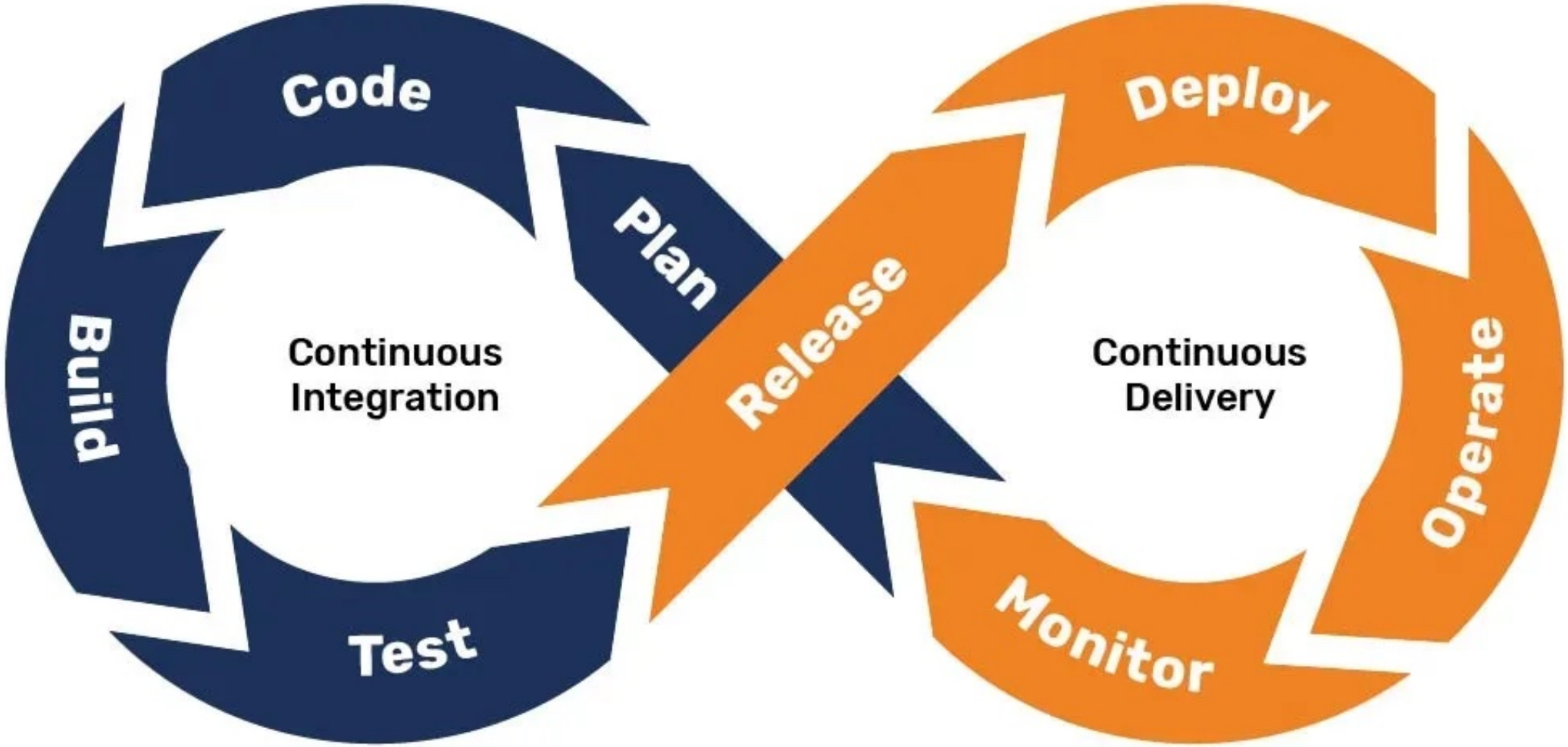
Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

O Problema do Desenvolvimento Tradicional

- **Integração Tardia:** Mesclar código apenas no final do projeto causa conflitos massivos ("Integration Hell").
- **Testes Manuais:** Processo lento e propenso a erros humanos, onde bugs são descobertos tarde demais.
- **Deploy Arriscado:** Atualizações grandes e raras que frequentemente quebram o ambiente de produção.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

CI/CD PROCESS



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

O que é CI/CD?

CI/CD é um conjunto de práticas para automatizar etapas do desenvolvimento de software.

Continuous Integration (CI), Integração Contínua:

- Desenvolvedores integram código frequentemente (ex: várias vezes ao dia).
- Cada integração é verificada automaticamente por um servidor de build.
- Objetivo: Detectar erros cedo ("fail fast").

Continuous Deployment/Delivery (CD), Entrega Contínua:

- **Delivery:** O código está pronto para ser liberado com um clique.
- **Deployment:** O código é liberado automaticamente para produção após aprovação.

O objetivo é **reduzir erros manuais** e aumentar a confiabilidade das entregas.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Benefícios no Desenvolvimento Web Moderno

Principais Vantagens:

- **Velocidade de Entrega:** Reduz o tempo entre a ideia e o produto final (Time-to-Market).
- **Qualidade do Código:** Testes automatizados garantem que novas funcionalidades não quebraram as antigas (Regression Testing).
- **Redução de Erros Humanos:** Elimina comandos manuais como **git push** ou **npm install** no servidor de produção.
- **Colaboração:** Facilita o trabalho em equipe, pois todos sabem quando uma nova versão foi publicada.

> Em um projeto web com 5 desenvolvedores, se um quebra a página ao fazer uma atualização manual, todos os usuários sofrem. Com CI/CD, o erro é detectado antes de chegar aos usuários.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Integração Contínua (CI)

CI é a prática de integrar alterações de código ao repositório principal com frequência, disparando automaticamente testes e verificações.

- Cada **commit** dispara automaticamente um processo de build e execução de testes unitários.
- A cada **push** ou **pull request**, o sistema executa:
  - Instalação de dependências
  - Verificação de qualidade do código
  - Build da aplicação
  - Testes automatizados

> Veja também a ferramenta pre-commit

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Integração Contínua (CI): Princípios

- Commits pequenos e frequentes (ao invés de grandes e raros)
- Testes automatizados executam a cada push
- O time recebe feedback imediato sobre falhas
- Código sempre em estado "integrável"

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Entrega Contínua (CD)

Entrega Contínua significa que o sistema fica sempre pronto para ser publicado.

O pipeline prepara a aplicação, mas a publicação em produção pode depender de aprovação humana.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

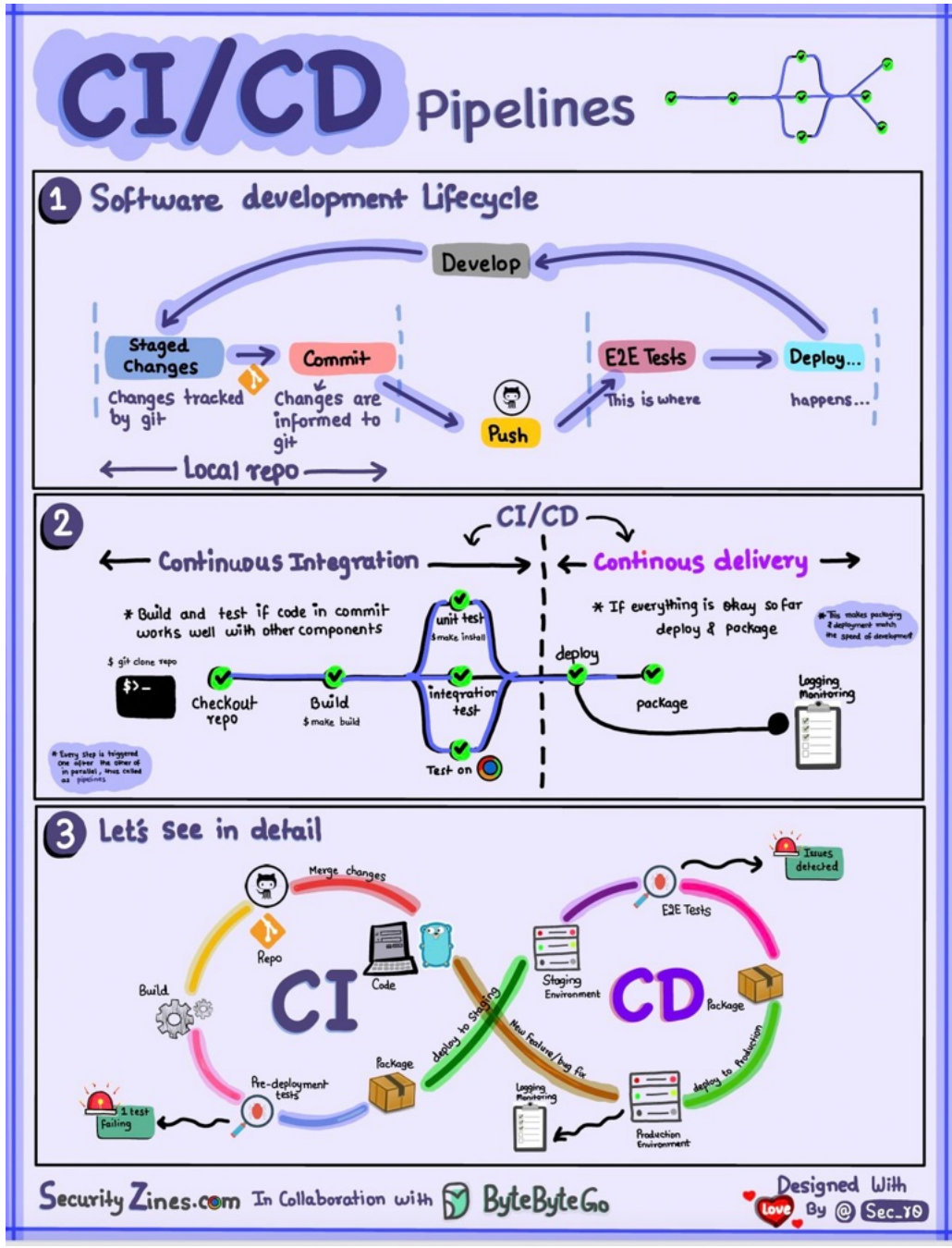
O que é um pipeline?

Pipeline é uma sequência automatizada de etapas.

Exemplo:

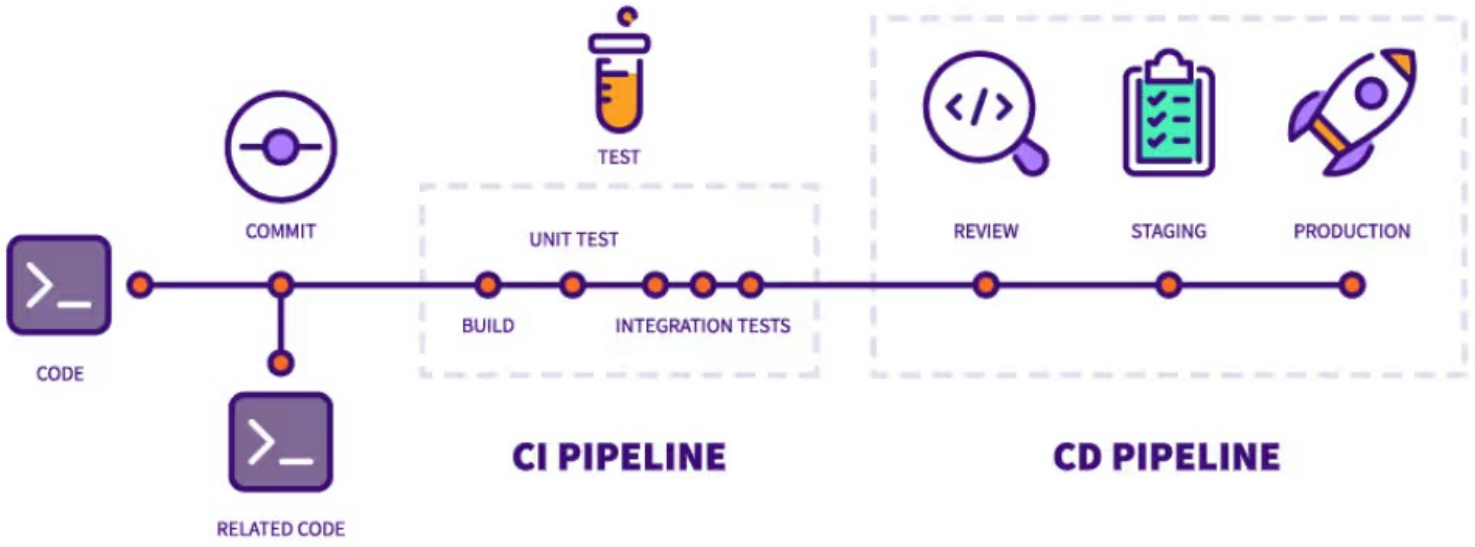
```
1. Checkout do código
2. Instalar dependências
3. Rodar lint
4. Rodar testes
5. Gerar build
6. Publicar aplicação
```

> Fonte da imagem: GTS-inc



Pipeline CI/CD

Um pipeline é uma sequência automatizada de etapas:



Se qualquer etapa falhar, o pipeline para e o time é notificado.

> Fonte da imagem: Gitlab

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Testes no Pipeline

- **Testes Unitários:** Validam os menores componentes individuais do código em isolamento.
- **Testes de Integração:** Verificam se diferentes módulos da aplicação funcionam corretamente quando combinados.
- **Testes de Fumaça (Smoke Tests):** Realizados logo após o build para verificar se as funcionalidades críticas estão funcionando minimamente.
- **Testes de Carga e Estresse:** Garantem que a aplicação suporte altos volumes de tráfego antes de lançamentos importantes.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

GitHub Actions

Plataforma de automação integrada diretamente no repositório do GitHub. Permite criar "pipelines" (fluxos de trabalho) sem instalar software complexo localmente.

Componentes Principais:

1. **Workflow:** Um arquivo YAML que define o fluxo de trabalho.
2. **Runner:** O computador/servidor onde o código é executado.
3. **Job:** Uma unidade de trabalho dentro do workflow (ex: Testar, Construir, Deploy).
4. **Step:** Uma tarefa individual dentro de um job (ex: Instalar dependências, rodar testes).

> Arquivo geralmente encontrado em `.github/workflows/nome-do-workflow.yml`.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Componentes do GitHub Actions

```
Workflow
  └── Job (executa em um runner)
        └── Step (unidade de trabalho)
              ├── action  (usa uma action pronta)
              └── run     (executa comandos shell)
```

Componente	Descrição
Workflow	O arquivo YAML completo de automação
Event (trigger)	O que dispara o workflow (push, PR, schedule...)
Job	Conjunto de steps que rodam em um mesmo runner
Step	Comando individual ou action reutilizável
Runner	Servidor virtual onde o job executa

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Anatomia de um Workflow

```yaml
name: Nome do Workflow           # ← nome exibido na interface

on:                              # ← evento que dispara o workflow
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:                            # ← lista de jobs
  nome-do-job:                   # ← identificador do job
    runs-on: ubuntu-latest       # ← sistema operacional do runner

    steps:                       # ← lista de passos
      - name: Descrição do passo
        uses: actions/checkout@v4   # ← usa uma action pronta

      - name: Outro passo
        run: echo "Hello, CI/CD!"   # ← executa comando shell
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Workflow

Um **workflow** é um processo automatizado.

Exemplo:

```
name: CI
```

Ele pode ser acionado por eventos como:

```
on:
  push:
  pull_request:
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Event

Um **event** é o gatilho que inicia o workflow.

Exemplos:

```yaml
on:
  push:
    branches: [main]

  pull_request:
    branches: [main]
```

Significa: Execute este workflow quando houver **push** ou **pull request** para a branch **main**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Job

Um **job** é um conjunto de etapas executadas em um ambiente.

Exemplo:

```
jobs:
  build:
    runs-on: ubuntu-latest
```

Neste caso:

- O job se chama **build**
- Ele roda em uma máquina virtual Ubuntu

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Step

Um **step** é uma etapa dentro de um job.

Exemplo:

```
steps:
  - name: Mostrar mensagem
    run: echo "Executando pipeline"
```

Cada step pode executar comandos ou usar uma action pronta.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Action

Uma **action** é uma tarefa reutilizável.

Exemplo:

```
- uses: actions/checkout@v4
```

Essa action baixa o código do repositório para o ambiente do workflow.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Runner

O **runner** é a máquina que executa o workflow.

Pode ser:

- Hospedado pelo GitHub
- Configurado pela própria organização

Exemplo:

```
runs-on: ubuntu-latest
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Variáveis e Secrets

Senhas, tokens de API e chaves nunca devem estar no código-fonte!

GitHub Secrets ficam em: Settings → Secrets and variables → Actions

```yaml
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Deploy para servidor
        env:
          API_KEY: ${{ secrets.MINHA_API_KEY }}    # ← secret seguro
          NODE_ENV: production                      # ← variável comum
        run: |
          echo "Fazendo deploy..."
          curl -H "Authorization: $API_KEY" https://meu-servidor.com/deploy
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

GitHub Marketplace: Actions Prontas

Não reinvente a roda! Use actions da comunidade:

Action	Para quê serve
`actions/checkout@v4`	Baixa o código do repositório
`actions/setup-node@v4`	Configura Node.js
`actions/cache@v4`	Cacheia dependências para acelerar CI
`codecov/codecov-action@v4`	Envia relatório de cobertura de testes
`actions/deploy-pages@v4`	Deploy no GitHub Pages
`docker/build-push-action@v5`	Build e push de imagem Docker

Como encontrar: github.com/marketplace?type=actions

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Monitoramento e Visualização (GitHub Actions)

- **Workflow Visualizer:** Interface gráfica no GitHub que permite ver cada etapa do pipeline em tempo real e identificar falhas com ícones de status.
- **Live Logs:** Acesso detalhado aos logs de execução de cada comando, essencial para depuração de erros sensíveis ao tempo.
- **Feedback Contínuo:** Notificações automáticas para a equipe de desenvolvimento sempre que um "build" ou teste falha.
- **Observabilidade:** Capacidade de entender o estado do sistema através de logs, métricas e performance após o deploy.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Segurança no Pipeline (DevSecOps)

- **Segurança "Shift Left":** Integrar verificações de segurança o mais cedo possível no ciclo de desenvolvimento.
- **Escaneamento de Vulnerabilidades:** Uso de ferramentas automatizadas para detectar falhas no código e em dependências de terceiros.
- **Gestão de Segredos:** Proteção de informações sensíveis (senhas, chaves de API) para que nunca sejam expostas no código-fonte.
- **Compliance Automatizado:** Garantia de que o código segue normas regulatórias e políticas da organização antes do deploy.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Boas Práticas de CI/CD

- **Use versões fixas** de actions: `actions/checkout@v4` (não `@latest`)
- **Faça commits pequenos e frequentes** — facilita rastrear erros
- **Nunca commite secrets** no código-fonte
- **Mantenha o CI rápido** — pipelines lentos são ignorados
- **Teste em paralelo** quando possível
- **Proteja a branch** `main` com regras obrigatórias
- **Documente seu pipeline** — um comentário `# por quê` vale ouro
- **Falhe rápido** — coloque validações simples no início do pipeline

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão

- CI/CD automatiza etapas importantes do desenvolvimento Web
- CI verifica continuamente se o código funciona
- Continuous Delivery prepara a aplicação para publicação
- Continuous Deployment publica automaticamente
- GitHub Actions permite criar workflows com YAML
- Um pipeline pode incluir instalação, lint, testes, build e deploy
- Secrets e environments são importantes para segurança e controle

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício e Questões

Questão 1

Discuta os seguintes pontos:

- Todo projeto precisa de CI/CD?
- Quando deploy automático pode ser perigoso?
- Quais testes deveriam ser obrigatórios antes de publicar?
- Quem deve aprovar o deploy em produção?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br