DevOps

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

Integração Contínua (CI)

O que é CI (Integração Contínua)?

- **Integração Contínua (CI)** é a prática DevOps que resolve esse problema. A ideia central é: **integrar o código de forma frequente, automática e testada.**

- Em vez de esperar semanas, os desenvolvedores integram suas pequenas mudanças ao repositório principal várias vezes ao dia.

O que é CI (Integração Contínua)?

Cada vez que isso acontece (em um push ou pull request), um "robô" (o pipeline de CI) entra em ação. Este processo automatizado faz o seguinte:

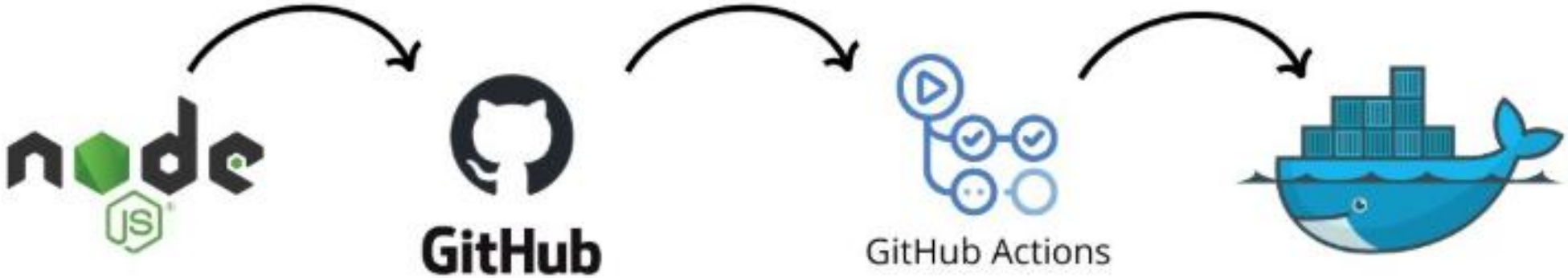
1. **Baixa o Código:** Pega a nova alteração e o restante do código.
2. **Instala Dependências:** Garante que o ambiente esteja completo.
3. **Executa Testes:** Roda automaticamente todos os testes (unitários, de integração, etc.). Se um único teste falhar, o processo para.
4. **Constrói (Build):** Compila o código e gera um "pacote" executável (no nosso caso, uma imagem Docker).
5. **Notifica (Feedback Rápido):** Se algo falhar (um teste ou o build), a equipe é notificada imediatamente.

Por que CI é Fundamental?

- **Feedback Rápido:** Os desenvolvedores descobrem se quebraram algo em minutos, não em semanas.
- **Correção Fácil de Bugs:** Como as mudanças são pequenas, é muito mais fácil encontrar e corrigir o bug.
- **Qualidade do Código:** Garante que o repositório principal (main) esteja *sempre* estável e funcionando.
- **Redução de Risco:** Automatiza tarefas manuais e repetitivas, eliminando o erro humano.
- **Base para CD:** É o primeiro passo essencial para a Entrega Contínua

Exemplo Prático: Passo a Passo

- Vamos construir um pipeline que testa e faz o build de uma aplicação Node.js simples.



Passo 1: A Aplicação Node.js (Olá Mundo DevOps!)

- Primeiro, precisamos de uma aplicação. Vamos usar o Express.
- **app.js** Esta é a lógica principal da nossa aplicação. Note que exportamos o app para que nossos testes possam usá-lo.

JS app.js > ...

```
1   // Separamos a lógica do app da inicialização do servidor
2   // para que os testes possam carregar o app sem iniciar o servidor.
3   const express = require("express");
4   const app = express();
5
6   app.get("/", (req, res) => {
7     res.status(200).send("Olá Mundo DevOps!");
8   });
9
10  module.exports = app; // Exportamos o app
```

Passo 1: A Aplicação Node.js (Olá Mundo DevOps!)

- **server.js** Este arquivo *apenas* inicia o servidor. Separar app.js de server.js é uma boa prática que facilita os testes.

```
JS server.js > ...
1   // Este arquivo importa o 'app' e inicia o servidor.
2   // O CI não vai usar este arquivo, mas o Docker sim.
3
4   const app = require("./app");
5   const port = process.env.PORT || 3000;
6
7   app.listen(port, () => {
8     console.log(`Servidor rodando na porta ${port}`);
9   });
```

Passo 1: A Aplicação Node.js (Olá Mundo DevOps!)

- No diretório da aplicação:

```
npm init -y
npm install express –save
npm install jest –D
npm install supertest –D
```

Passo 1: A Aplicação Node.js (Olá Mundo DevOps!)

- Inclua no package.json os scripts de teste:

```
"scripts": {
  "start": "node server.js",
  "test": "jest"
},
```

Passo 1: A Aplicação Node.js (Olá Mundo DevOps!)

- Para testar localmente:
  - Rode `npm install` no seu terminal.
  - Rode `npm start`.
  - Acesse `http://localhost:3000` no seu navegador. Você deve ver "Olá Mundo DevOps!".

Passo 2: O Teste Automatizado

- Nosso pipeline de CI precisa de testes para validar o código.
- **app.test.js** Vamos usar jest e supertest para fazer uma requisição à nossa aplicação e verificar a resposta.

app.test.js > ...

```
1   const request = require("supertest");
2   const app = require("./app"); // Importamos nosso app
3
4   describe("API Olá Mundo", () => {
5     it('Deve retornar "Olá Mundo DevOps!" na rota /', async () => {
6       // Faz uma requisição GET para a rota /
7       const response = await request(app).get("/");
8
9       // Verifica se o status code é 200 (OK)
10      expect(response.statusCode).toBe(200);
11
12      // Verifica se o corpo da resposta é o esperado
13      expect(response.text).toBe("Olá Mundo DevOps!");
14    });
15  });
```

Passo 2: O Teste Automatizado

Para testar localmente:

- Rode `npm test` no seu terminal.
- Você deve ver uma saída indicando que o teste passou (PASS).

Passo 3: Docker

Agora, vamos criar um "pacote" da nossa aplicação usando o Docker. O Dockerfile é a receita para construir uma imagem Docker.

Dockerfile > ...

```dockerfile
# Estágio 1: Build - Instala dependências
# Usamos uma imagem "builder" que contém todas as ferramentas do Node
FROM node:22-alpine AS builder
WORKDIR /usr/src/app
# Copia o package.json e package-lock.json
COPY package*.json ./
# Instala SOMENTE as dependências de produção
RUN npm ci --omit=dev
# Copia o restante do código da aplicação
COPY . .
# ---
# Estágio 2: Produção - Imagem final
# Usamos uma imagem "slim" mais leve para produção
FROM node:22-alpine
WORKDIR /usr/src/app
# Copia as dependências instaladas do estágio "builder"
COPY --from=builder /usr/src/app/node_modules ./node_modules
# Copia o código da aplicação do estágio "builder"
COPY --from=builder /usr/src/app ./
# Expõe a porta que a aplicação usa
EXPOSE 3000
# Comando para iniciar a aplicação
CMD [ "node", "server.js" ]
```

Passo 3: Docker

.dockerignore Este arquivo diz ao Docker quais arquivos e pastas ignorar. É crucial para não copiar node_modules locais para dentro da imagem.

.dockerignore

```
1    node_modules
2    npm-debug.log
3    .git
4    .gitignore
```

Passo 3: Docker

Para testar localmente:

1. Certifique-se de ter o Docker Desktop instalado e rodando.
2. Rode `docker build -t ola-devops .`
3. Rode `docker run -p 3000:3000 -d ola-devops`
4. Acesse `http://localhost:3000`. A aplicação agora está rodando de dentro de um contêiner!

Passo 4: GitHub Actions (O Pipeline de CI)

Este é o cérebro da nossa CI. Este arquivo YAML diz ao GitHub o que fazer quando enviarmos código.

Crie a pasta .github/workflows/ e, dentro dela, o arquivo ci.yml: .github/workflows/ci.yml

.github > workflows > ! ci.yml

```yaml
# .github/workflows/ci.yml
name: Pipeline de CI - Olá Mundo DevOps
# Define quando o pipeline deve rodar
on:
  push: # Rodar em todo push
    branches: ["main"] # Apenas na branch main
  pull_request: # Rodar em todo pull request
    branches: ["main"] # Que tenha como alvo a branch main
# Define os "trabalhos" (jobs) que o pipeline executará
jobs:
  # O nome do nosso job é "test-and-build"
  test-and-build:
    # A máquina virtual que o job usará
    runs-on: ubuntu-latest
    # Os passos (steps) que o job executará em sequência
    steps:
      # 1. Baixa o código do repositório para a máquina virtual
      - name: Checkout do Código
        uses: actions/checkout@v4
      # 2. Configura o ambiente Node.js
      - name: Configurar Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "22" # Usamos a mesma versão do Dockerfile
          cache: "npm" # Habilita o cache do npm para builds mais rápidos
      # 3. Instala todas as dependências (incluindo devDependencies)
      - name: Instalar Dependências
        run: npm install
      # 4. Executa os testes
      - name: Rodar Testes
        run: npm test
      # 5. Configura o Docker Buildx (necessário para build)
      - name: Configurar Docker Buildx
        uses: docker/setup-buildx-action@v3
      # 6. Constrói (build) a imagem Docker
      # Não vamos fazer "push" para um registro por enquanto, apenas construir.
      - name: Build da Imagem Docker
        uses: docker/build-push-action@v5
        with:
          context: . # O contexto é o diretório atual
          file: ./Dockerfile # O caminho para o Dockerfile
          push: false # "false" significa apenas construir, não enviar
          tags: ola-devops:latest # Um nome para a imagem
```

Passo 4: GitHub Actions (O Pipeline de CI)

Este é o cérebro da nossa CI. Este arquivo YAML diz ao GitHub o que fazer quando enviarmos código.

Crie a pasta .github/workflows/ e, dentro dela, o arquivo ci.yml: .github/workflows/ci.yml

.github > workflows > ! ci.yml

```yaml
# .github/workflows/ci.yml
name: Pipeline de CI - Olá Mundo DevOps
# Define quando o pipeline deve rodar
on:
  push: # Rodar em todo push
    branches: ["main"] # Apenas na branch main
  pull_request: # Rodar em todo pull request
    branches: ["main"] # Que tenha como alvo a branch main
# Define os "trabalhos" (jobs) que o pipeline executará
jobs:
  # O nome do nosso job é "test-and-build"
  test-and-build:
    # A máquina virtual que o job usará
    runs-on: ubuntu-latest
    # Os passos (steps) que o job executará em sequência
    steps:
      # 1. Baixa o código do repositório para a máquina virtual
      - name: Checkout do Código
        uses: actions/checkout@v4
      # 2. Configura o ambiente Node.js
      - name: Configurar Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "22" # Usamos a mesma versão do Dockerfile
          cache: "npm" # Habilita o cache do npm para builds mais rápidos
      # 3. Instala todas as dependências (incluindo devDependencies)
      - name: Instalar Dependências
        run: npm install
      # 4. Executa os testes
      - name: Rodar Testes
        run: npm test
      # 5. Configura o Docker Buildx (necessário para build)
      - name: Configurar Docker Buildx
        uses: docker/setup-buildx-action@v3
      # 6. Constrói (build) a imagem Docker
      # Não vamos fazer "push" para um registro por enquanto, apenas construir.
      - name: Build da Imagem Docker
        uses: docker/build-push-action@v5
        with:
          context: . # O contexto é o diretório atual
          file: ./Dockerfile # O caminho para o Dockerfile
          push: false # "false" significa apenas construir, não enviar
          tags: ola-devops:latest # Um nome para a imagem
```

Passo 5: Execução do pipeline de CI

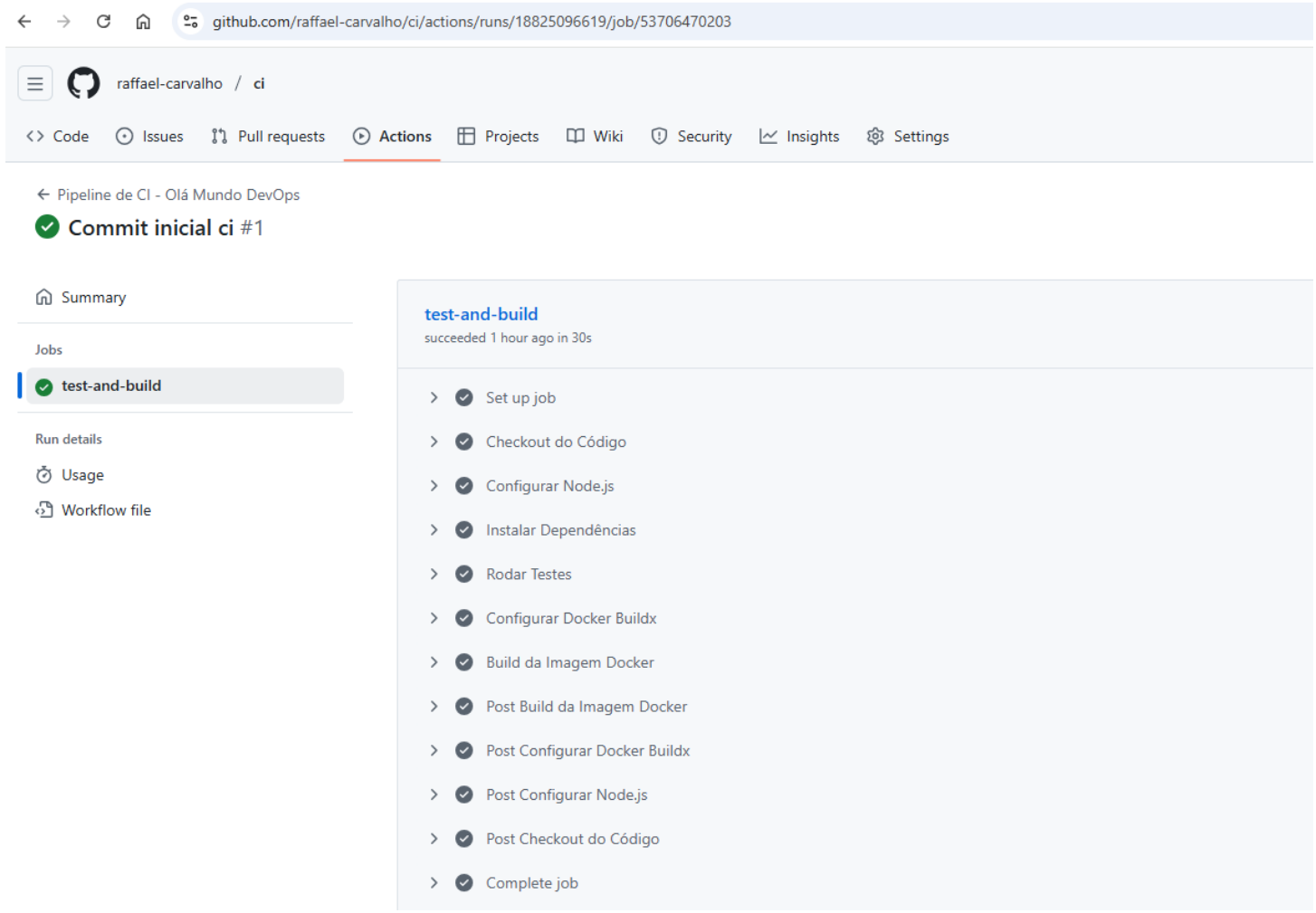
Crie um Repositório no GitHub: Vá ao GitHub e crie um novo repositório (pode ser público ou privado).

Adicione os Arquivos: Crie todos os arquivos que listamos acima no seu computador, na estrutura de pastas correta (especialmente o .github/workflows/ci.yml).

Envie o Código para o GitHub:

```
git init
git add .
git commit -m "Commit inicial do projeto Olá Mundo DevOps"
git remote add origin <URL_DO_SEU_REPOSITORIO.git>
git branch -M main
git push -u origin main
```

Passo 5: Execução do pipeline de CI



Exercício em grupo

1. Conforme a prática desta aula, replique os passos em seu computador. Envie o link do repositório criado no github, onde será possível conferir a execução do pipeline CI, juntamente com um pequeno parágrafo explicando em quais situações essa abordagem com CI pode ser útil dentro de uma empresa.

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional.8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.