📘 Aula - Teoria de CI/CD

CI/CD - O que é?

CI/CD é uma combinação de práticas de integração contínua e entrega contínua. O CI/CD automatiza grande parte da intervenção humana manual tradicionalmente necessária para obter novo código de um commit em produção, como compilação, teste e deploy.

CI - Integração Contínua

Integração contínua é a prática de integrar todas as suas alterações de código na branch principal de um repositório, testando automaticamente cada alteração quando é realizado um commit ou merge, iniciando em seguida também de forma automática uma compilação. Com CI, erros e problemas de segurança podem ser identificados e corrigidos com mais facilidade e muito mais cedo no ciclo de vida de desenvolvimento de um software.

Ao realizar merges de alterações com frequência e acionar processos automáticos de teste e validação, é possível minimizar a possibilidade de conflito de código, mesmo com vários desenvolvedores trabalhando no mesmo aplicativo. Uma vantagem secundária é que você não precisa esperar muito tempo por respostas e pode, se necessário, corrigir bugs e problemas de segurança enquanto o código ainda está fresco em sua mente.

CD - Entrega Contínua

Entrega contínua é uma prática de desenvolvimento de software que funciona em conjunto com CI para automatizar o provisionamento de infraestrutura e o processo de lançamento de aplicações.

Depois que o código é testado e construído como parte do processo de CI, a etapa de CD assume o controle durante os estágios finais para garantir que possa ser feito o deploy da aplicação com todos os recursos e processos necessários. A entrega contínua pode abranger tudo, desde o provisionamento da infraestrutura até o deploy da aplicação no ambiente de teste ou produção.

CI/CD - Pipeline

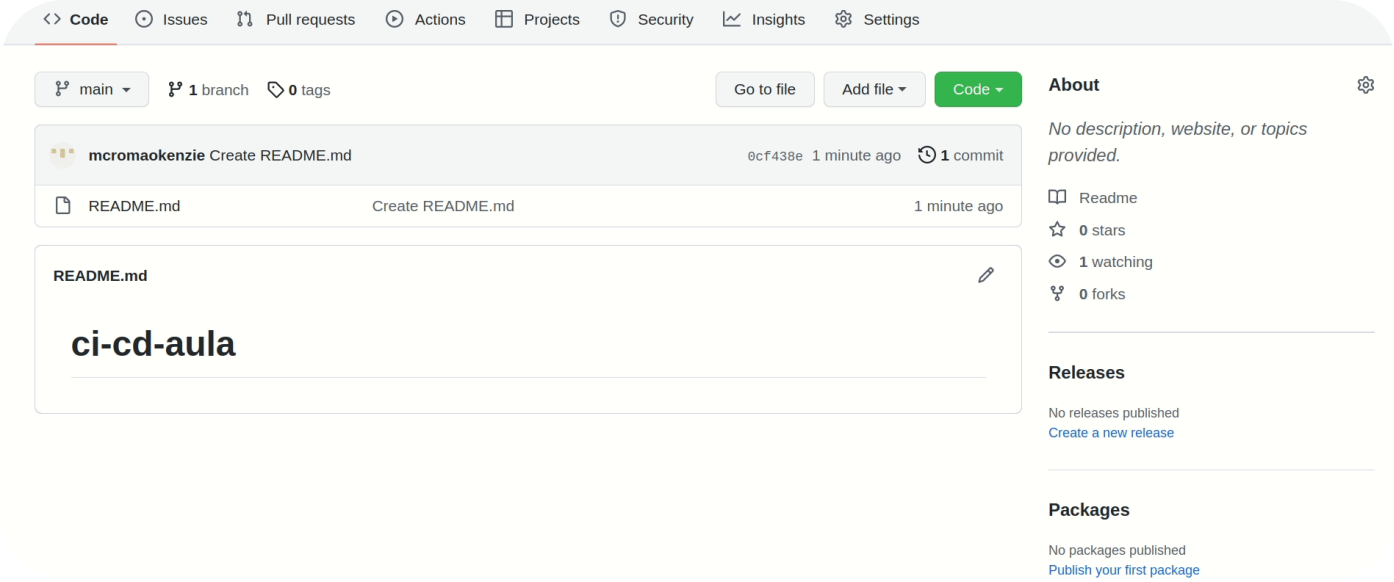
Um pipeline de CI/CD consiste em uma série de etapas a serem realizadas para a disponibilização de uma nova versão de um código. Na prática, declaramos todas essas etapas em um arquivo.

Existem diversas ferramentas para criarmos e executarmos esses pipelines, por exemplo: Jenkins, TravisCI, Azure Pipelines, entre outos. Nós usaremos o Github Actions. O deploy será feito no Heroku. Entretanto, em vez de usarmos scripts para realizar o processo de CD, utilizaremos uma ferramenta do próprio Heroku, que possibilita uma conexão automática com o repositório do projeto.

Entendendo o arquivo usado pelo GitHub Actions

O arquivo usado pelo GitHub Actions possui a extensão yml. Isso significa que ele possui uma estrutura muito parecida com a dos arquivos de docker compose.

Vamos analisar um template fornecido pelo próprio Github Actions. Você pode notar que há diversos templates para CI e CI/CD disponibilizados pelo Actions. Vamos iniciar com um workflow simplificado, que possui uma **estrutura básica**.



Note que o arquivo com o pipeline de CI/CD foi criado em uma pasta `.github/workflows` que fica na raiz do projeto. Essa é a configuração padrão para o uso do GitHub Actions.

Agora vamos entender um pouco melhor cada parte do arquivo.

Parte 1 - "name"

Copiar para àrea de transferência

```yaml
#file.yml

name: CI
```

Iniciando o arquivo, colocamos o nome da Action que estamos criando.

Parte 2 - "on"

Copiar para área de transferência

```yaml
#file.yml
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
```

Nesta parte informamos **quando** a Action deve ser executada (é o trigger do Action). No exemplo acima, a Action será executada sempre que for feito um push ou um pull_request na branch main.

Parte 3 - "jobs"

Copiar para área de transferência

```
#file.yml
jobs:
  build:
    runs-on: ubuntu-latest
```

É uma lista de "steps". Nossa aplicação possui apenas um job, chamado build. Esse job será executado em uma máquina virtual com a última versão do ubuntu.

Parte 4 - "steps"

Copiar para área de transferência

```
#file.yml
steps:
  - uses: actions/checkout@v3
  - name: Run a one-line script
    run: echo Hello, world!
```

```
  - name: Run a multi-line script
    run: |
      echo Add other actions to build,
      echo test, and deploy your project.
```

Este workflow possui 3 steps bem simples. No primeiro step, é feito um "check-out" do repositório na máquina virtual que está rodando o ubuntu. Isso permitirá rodar scripts e realizar outras ações com base no código.

O segundo step executa um comando simples. E o terceiro step executa um conjunto de comandos. É importante destacar que um step começa a ser executado apenas quando o anterior termina de ser executado.

Na próxima aula veremos como aplicar essas conceitos para criar um pipeline de CI/CD de um projeto em Django. Serão necessárias pequenas alterações no settings.py e no heroku.yml.

Referências

What is CI/CD?

GitHub Actions