04

Faça o que eu fiz na aula

Antes de implementarmos as melhorias em nosso processo, precisamos entender como funciona o arquivo gitlab-ci. Basicamente, ele é dividido em três etapas que chamamos de " `stages` ", sendo a primeira reservada para " `build` ", a segunda para " `tests` " e por fim " `deploy` ", Em um pipeline, não precisamos implementar as três etapas, podemos implementar apenas as etapas que julgarmos necessário.

Pensando nisso, podemos certamente otimizar passo a passo nosso processo.

Em uma empresa de desenvolvimento de software, quem nunca ouviu o termo "na minha máquina funciona"? Pois é, estamos com o mesmo problema em nossa empresa, muito dos desenvolvedores alegam que as features desenvolvidas funciona em suas máquinas, mas ao subir para produção alguma coisa dá errado. Então esse será nosso primeiro desafio no processo de CI. Faremos então, com que o ambiente de produção se assemelhe ao ambiente de desenvolvimento. Sabemos que uma das maneiras de garantir tal semelhança é utilizar docker, criando uma imagem idêntica a de produção e subir um container com as configurações desejadas. Vamos então criar um Dockerfile para nossa aplicação, e nele vamos colocar tudo o que nossa imagem precisa ter. Na raiz do nosso projeto vamos criar o arquivo `Dockerfile` com o seguinte conteúdo:

```
FROM python:3.6
#Copiando os arquivos do projeto para o diretorio usr/src/app
COPY . /usr/src/app
#Definindo o diretorio onde o CMD será executado e copiando o arquivo de requerimentos
WORKDIR /usr/src/app
COPY requirements.txt ./
# Instalando os requerimentos com o PIP
RUN pip install --no-cache-dir -r requirements.txt
# Expondo a porta da APP
EXPOSE 8000
# Executando o comando para subir a aplicacao
CMD ["gunicorn", "to_do.wsgi:application", "--bind", "0.0.0.0:8000", "--workers", "3"]
```

Como nosso projeto está desenvolvido em python, criaremos uma imagem docker para atender nossa necessidade.

Com o Dockerfile criado, vamos adicionar à nossa pipeline uma imagem docker que precisaremos em todos os passos da pipeline; pois build, teste e deploy, devem ser feitos com base no ambiente de produção, evitando assim a frase "na minha máquina funciona".

Como a criação e definição da imagem é um pré requisito para nosso processo, vamos adicionar à nossa pipeline, um "job" que será responsável por realizar o build da imagem que vamos utilizar em nosso projeto. Para isso vamos editar o arquivo `.gitlab-ci` com o seguinte conteúdo:

```
build-docker:
  stage: build
  script:
  - docker build -t minha-imagem .
```

Em seguida vamos dar um push no gitlab, para dar inicio a pipeline

```
git add --all
git commit -m “realizando build da imagem”
git push gitlab master
```

Ao realizarmos um push da alteração que fizemos no arquivo gitlab-ci, o pipeline será iniciado e teremos uma falha no pipeline, pois o comando "docker" não é reconhecido pela imagem que estamos utilizando.

Para nos prevenir desse problema, vamos com a palavra-chave "image" definir uma imagem para que nosso pipeline possa utilizar comandos docker, ficando assim:

```
image: docker:stable

build-docker:
  stage: build
  script:
  - docker build -t minha-imagem .
```

Em seguida vamos dar novamente um push no gitlab:

```
git add --all
git commit -m “realizando build da imagem”
git push gitlab master
```

Realizando novamente um push de nosso projeto, veremos que o pipeline falhou mais uma vez. Isso porque agora o serviço docker está sem permissão de uso, e para resolver isso vamos conhecer mais uma palavra-chave: a "services". A palavra-chave services define uma imagem do Docker que é executada durante uma tarefa vinculada à imagem do Docker que a palavra-chave image define. Isso permite que você acesse a imagem do serviço durante o tempo de criação

Então vamos no arquivo `.gitlab-ci.yml` e vamos adicionar o seguinte conteúdo:

```
image: docker:stable

services:
  - docker:dind

before_script:
  - docker info

build-docker:
  stage: build
  script:
  - echo “hello world”
```

Adicionamos também ao pipeline a palavra reservada `before_script` , que se encarrega de executar qualquer script definido neste bloco antes da execução de cada job no pipeline.

Vamos então novamente realizar um push:

```
git add --all
git commit -m “realizando build da imagem”
git push gitlab master
```

Voltando ao pipeline, percebemos que ele agora está passando sem erros.