01

Job de produção

Transcrição

O script que o instrutor segue durante a aula é o seguinte:

```
# Criar o job para colocar a app em producao:
    Nome: todo-list-producao
    Tipo: Freestyle
    # Este build é parametrizado com 2 Builds de Strings:
        Nome: image
        Valor padrão: - Vazio, pois o valor sera recebido do job anterior.

        Nome: DOCKER_HOST
        Valor padrão: tcp://127.0.0.1:2376

    # Ambiente de build > Provide configuration files
        File: .env-prod
        Target: .env

    # Build > Executar shell
        #Execute shell
        #!/bin/sh
        {
            docker run -d -p 80:8000 -v /var/run/mysqld/mysqld.sock:/var/run/mysqld/mysqld.sock -v /

        } || { # catch
            docker rm -f django-todolist-prod
            docker run -d -p 80:8000 -v /var/run/mysqld/mysqld.sock:/var/run/mysqld/mysqld.sock -v /
        }

Ações de pós-build > Slack Notifications: Notify Success e Notify Every Failure
```

[00:00] Tudo bem pessoal? Vamos continuar agora com o nosso pipeline, só pra gente recapitular aonde a gente parou.

[00:07] Primeiro a gente fez a parte do build da imagem, nós já fizemos a parte de deployar a imagem pro ambiente de desenvolvimento. Agora é a etapa da gente deployar em produção.

[00:22] Qual que é a diferença de desenvolvimento pra produção? Desenvolvimento muda o arquivo .env pra você conectar no banco de desenvolvimento. Produção é a mesma coisa. Então, o que nós vamos fazer agora? Nós vamos criar um job pra fazer o deploy em produção, utilizando o arquivo de configuração pra produção.

[00:43] Vale ressaltar que quando a gente fala de ambiente de produção cada empresa tem a sua política, cada projeto. Normalmente esse deploy não é feito automaticamente, a não ser que você tenha uma política muito bem estruturada de responsabilidades, entre times de desenvolvimento e produção. Nesse nosso caso a gente vai criar o job primeiro, depois nós vamos colocar uma decisão pro usuário. "Depois de deployar em desenvolvimento você quer colocar em produção, sim ou não?"

[01:11] Então vamos lá. Vindo pro Jenkins aqui, a gente vai criar um novo job com o nome todo-list-producao e esse job vai ser free-style, a gente vai criar novamente um free-style. Então que que a gente precisa fazer aqui para que esse job funcione? Primeiro a gente tem que avisar que ele é parametrizado. E quais são os parâmetros que ele vai receber? O DOCKER_HOST, que a gente já tem o valor aqui, e também a image que vai estar vazia porque quem vai passar essa informação é o job anterior.

[01:59] Como eu falei para vocês, depois do deploy de desenvolvimento vai ter opção pro usuário "Você quer ou não quer pôr em produção?", se ele quiser o job vai passar o nome da imagem nova pra ele. Então o valor padrão vai estar vazio.

[02:12] Depois disso a gente vai ter que configurar como que as variáveis de ambiente vão ser trabalhadas nesse job. É muito simples, a gente vem aqui embaixo e seleciona "Provide Configuration files". Quando vocês selecionarem ele vai dar aquele dropdown, que a gente já viu uma vez, que mostra qual dos arquivos a gente vai ter que escolher. A gente vai escolher o de prod agora. E qual que é o target dele? Ele vai ficar aqui dentro desse diretório como .env.

[02:53] Lembrando que cada job tem o seu diretório específico dentro do Jenkins. Então, dessa maneira a gente tem o .env dentro de /var/jenkins e dentro desse diretório todo-list-producao.

[03:08] Feito isso, agora a gente vai colocar um shell script pra subir o container de produção. Lembrando que muda a porta de conexão então, nesse caso, a porta de conexão vai ser a 80. A de produção é 80, e a desenvolvimento é 81.

[03:24] Antes de subir que que a gente tenta fazer? Derrubar o container de produção pra subir o container novo. Existem outras maneiras de fazer isso, a gente vai ver mais pra frente em outros cursos da Alura como que a gente trabalha com Kubernetes pra fazer a orquestração. Essa é uma maneira mais simples pra gente terminar a pipeline nosso de uma maneira saudável.

[03:44] Então o que ele tenta fazer? Primeiro ele tenta derrubar e subir o container. O que ele tenta fazer? Primeiro ele tenta subir o container na porta 80 passando tanto o sock de conexão do MySQL, quanto as variáveis de ambiente dentro do arquivo .env.

[04:03] Eu também dou um nome pra ele que é o django-todolist-prod, que é o nome do container, e essa imagem vem lá do parâmetro que a gente vai receber do job anterior. Se ele não conseguir subir por algum motivo, ou porque o container já tá rodando ou porque a porta tá ocupada, ele vai derrubar o container com o mesmo nome, ele vai procurar um container com mesmo nome, e vai tentar subir novamente. Se mesmo assim ele não conseguir, aí o nosso job falha.

[04:31] Então agora o que que a gente faz? Copia esse shell script, adiciona o Executar shell, passa o script aqui pra ele, e, nas ações de pós-build a gente vai marcar, no Slack Notifications, duas opções: se for com sucesso ou se for com falha. Por quê? Nós aprendemos a usar o Slack Notifications dentro do Groovy, essa é uma outra maneira da gente utilizar via interface usando o free-style.

[05:05] Vou dar um salvar e agora eu vou mandar construir com parâmetros. Como ainda ele não tá recebendo do job anterior, ele vai receber manualmente esse valor, por enquanto. Eu vou dar um construir.

[05:24] Ele tá rodando, rodou com sucesso. E agora a gente acessa a URL da aplicação sem a porta 81, que é a porta normal que é a porta 80 de conexão. A gente já tem aquele usuário que é alura, senha mestre123. Conectamos aqui, ele vai carregar os nossos "todos" pra produção. Pronto, carregou. Essa é a nossa interface de produção sem usar a porta 81.

[05:58] Na próxima aula a gente vai, agora, configurar os três jobs pra interagirem com os parâmetros, pra que a gente consiga passar valores de um job pro outro. Até a próxima.