04

Job para o ambiente de desenvolvimento

Transcrição

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

```
# Novo Job: todo-list-desenvolvimento:
    # Tipo: Pipeline
    # 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

    pipeline {

        agent any

        stages {
            stage('Oi Mundo Pipeline como Codigo') {
                steps {
                    sh 'echo "Oi Mundo"'
                }
            }
        }
    }

    pipeline {
        environment {
            dockerImage = "${image}"
        }
        agent any

        stages {
            stage('Carregando o ENV de desenvolvimento') {
                steps {
                    configFileProvider([configFile(fileId: '<id do seu arquivo de desenvolvimento>',
                        sh 'cat $env > .env'
                    }
                }
            }
            stage('Derrubando o container antigo') {
                steps {
                    script {
                        try {
                            sh 'docker rm -f django-todolist-dev1'
                        } catch (Exception e) {
                            sh "echo $e"
                        }
                    }
                }
```

```
            }
            stage('Subindo o container novo') {
                steps {
                    script {
                        try {
                            sh 'docker run -d -p 81:8000 -v /var/run/mysqld/mysqld.sock:/var/run/mys
                        } catch (Exception e) {
                            slackSend (color: 'error', message: "[ FALHA ] Não foi possivel subir o
                            sh "echo $e"
                            currentBuild.result = 'ABORTED'
                            error('Erro')
                        }
                    }
                }
            }
            stage('Notificando o usuario') {
                steps {
                    slackSend (color: 'good', message: '[ Sucesso ] O novo build esta disponivel em:
                }
            }
        }
    }

# todo-list-principal
    # Definir post build: image=$image
```

[00:00] Oi pessoal, tudo bem? Nessa aula a gente vai começar o nosso job pra fazer o deploy da aplicação no ambiente de desenvolvimento. A gente já configurou as notificações, agora tá na hora de montar o job.

[00:14] Então a gente vem aqui em Novo job e a gente vai dar um nome pra esse job. Vamos padronizar o nome da seguinte maneira: todo-list-desenvolvimento. A diferença é que agora a gente vai usar um pipeline ao invés de construir um projeto free-style. A gente clica aqui em Pipeline e clica no ok. Pronto o job tá aqui.

[00:38] Pra que ele funcione de uma maneira legal, sem que eu tenha que configurar toda vez a imagem e o docker host, a gente vai falar que esse job, esse build no caso, vai ser parametrizado com seguintes parâmetros de string: o DOCKER_HOST que vai receber o valor de "tcp://127.0.0.1:2376".

[01:18] A gente vai adicionar uma segunda string chamada image e, nesse caso, a gente vai deixar o valor padrão vazio, por quê? Ele vai receber do job anterior esse dado, é por isso que a gente instalou parametrized plugin.

[01:36] Bom, nesse job ele é um pouquinho diferente do anterior porque a gente vai usar uma linguagem interpretativa chamada Groovy. O Groovy é uma linguagem muito simples de se entender, vamos criar um exemplo aqui primeiro. Eu vou copiar esse pipeline aqui e eu vou explicar pra vocês aqui na interface mesmo.

[01:56] Então aqui, como ele funciona? Eu tô avisando pra ele que é um pipeline, ele vai rodar em qualquer agente do Jenkins. Nesse caso a gente tá trabalhando com o Jenkins na mesma máquina com uma instalação só, mas é possível trabalhar com o Jenkins com máquinas slaves e aí aqui a gente definiria em qual máquina esse meu job rodaria, se é um slave, qual grupo que eu posso rodar.

[02:23] Depois de eu colocar qual que é o agente, eu vou definir os stages que ele vai executar. Eu só tenho uma declaração de stages porém, dentro dela, eu tenho vários stages que ele vai passar. Nesse exemplo, nós criamos um stage chamado "Oi Mundo Pipeline com Código", a gente só vai printar uma mensagem no log.

[02:43] E dentro desse stage eu tenho steps, eu também posso ter múltiplos steps dentro dos stages, então eu posso ter 10 stages, dentro de cada um deles 10 steps. E aí a gente vai salvar, é bem simples.

[03:02] Quando a gente mandar construir ele vai pedir alguns parâmetros, a gente não vai passar agora o image porque a gente não tá utilizando por enquanto, e deu um ok. Ele começou a executar, vocês vão perceber uma mudança na interface nossa aqui.

[03:16] Vocês estão reparando que agora, além dos nossos logs, ele mostra os stages pra gente aqui, e ele falou que o stage "Oi Mundo Pipeline com Código" foi executado em 337 milissegundos e com sucesso, ele tá verde aqui. Se passar só o mouse em cima e depois clicar em logs, olha que legal, ele mostrou o "Oi Mundo" pra gente.

[03:41] Bom, agora a gente já executou nosso "Oi Mundo", vamos deixar isso aqui um pouquinho melhor. Vamos voltar aqui na configuração do nosso job, no nosso pipeline a gente vai colar um código novo agora. Esse código, primeiro vamos executar pra ver o que que ele vai fazer. Colamos aqui, salvamos.

[04:08] Então agora vamos ver o comportamento do job. Construir com parâmetros, muda um pouquinho. Por enquanto, a gente vai passar a nossa imagem manualmente, depois a gente vai entender como passar do job anterior pra esse, só pra gente visualizar o nosso build agora.

[04:27] Reparem que eu coloquei o repositório do Git Hub, então ele sempre vai buscar a imagem mais atualizada no repositório. E eu vou dar um construir, eu vou esperar ele terminar de construir e a gente volta.

[04:41] Olha lá, terminou. E olha agora como mudou a nossa interface, eu tenho cada stage descrito, quanto tempo cada stage levou pra executar. Vamos dar uma olhadinha no nosso canal do Slack? Olha aqui, sucesso, seu novo build está disponível nesse IP. Vamos clicar aqui, olha lá a nossa aplicação subiu. Então o dev recebeu a notificação e ela tá funcionando. Pra gente ter certeza mesmo: alura, mestre123. Legal, tá funcional no ambiente de desenvolvimento que tá na porta distinta.

[05:20] Então vamos entender e o que que tem nesse pipeline agora. Primeira coisa, o agent continua igual, ele vai executar em todos os que tiver dentro desse cluster e agora é nesse aqui. O stages tem algumas definições diferentes.

[05:37] A primeira definição é: "Carregando o ENV de desenvolvimento", como que isso funciona? Ele vai ter um step aqui dentro que vai fazer o seguinte: ele vai usar o configFileProvider, aquele plugin que a gente instalou, e vai buscar o ID deste file. Esse ID aqui, se nós olharmos aqui no nosso Jenkins, Gerenciar Jenkins, Managed files, é esse ID aqui olha, se eu clicar aqui é esse ID. Aliás, esse é o de prod, vamos abrir o de dev. E é esse ID aqui, é o f00.

[06:13] Então eu tô falando pra ele "pega esse arquivo aqui, dá o nome de env", e agora eu tô executando uma variável env, eu tô dando um cat nessa variável e jogando pra dentro de .env que é o arquivo local meu, que vai subir o meu ambiente de desenvolvimento.

[06:31] Eu tento derrubar o container antigo, mesmo que ele não exista. Vocês lembram, a gente não criou container nenhum, não tinha container nenhum rodando. E se eu olhar o output do meu job aqui olha, container rodando. Se eu der um Logs aqui, ele até deu exception, ele falou "Olha, eu não consegui derrubar, até porque não tem container", mas ele continuou porque a ideia é que ele continue mesmo, ele só válida que a gente não vai subir um container com mesmo nome.

[06:56] E aí eu tenho outro step que é subindo container novo. Então, só pra sintaxe pra vocês entenderem, eu dei um try e um catch. Então, dentro de steps, eu criei script e criei uma entrada de try. Ou seja, eu posso ter vários scripts, vários passos dentro desse script, pra cada step, e dentro de um stage eu posso ter vários steps.

[07:22] Então é em cascata: eu tenho stages; dentro de stages, stages eu vou ter um só, eu tenho 'n' steps, dependendo do que eu quero fazer, nesse caso a gente já tem dois aqui; e dentro de steps eu tenho as entradas que eu preciso. E inclusive eu usei o try catch aqui pra ver se eu pego alguma exceção.

[07:41] Aí o meu próximo step é subindo container novo. O que que ele faz? Dentro desse stage eu tenho o step que executa um script, que vai tentar subir o container na porta 81 com o daemon passando dois volumes, um mysql que a gente já viu no job de teste e o env que a gente acabou de criar. De novo, como eu sempre falo pra vocês cada workspace, cada job tem o seu diretório e dentro desse diretório tem um .env, que veio de onde? Ele veio desse passo aqui que nós pegamos o configFileProvider.

[08:24] Passei o nome do container, porque que é importante passar o nome? Porque o job anterior tenta derrubar pelo nome, por isso que eu tenho que dar um nome para ele. E eu passo a imagem.

[08:35] Da onde é que veio o dockerImage? Ele veio daqui olha, tem um step novo no pipeline aqui que é o ambiente. Eu defino as variáveis de ambiente e eu tô falando pra ele que o dockerImage vem de image, que é uma variável que vai receber via parâmetro. Então a gente vai entrar aqui no Configurar de novo, via parâmetro. Eu passei manualmente, mas a gente vai configurar o próximo job pra passar automaticamente.

[09:03] E aí ele tenta subir e ele notifica se der algum erro. Então no exception aqui ele dá um slackSend, que é o nosso amiguinho aqui que tá na documentação, passando a cor de erro, que é vermelho no caso, tá como padrão, e a mensagem que não foi possível subir.

[09:24] Bom, ele notifica o usuário, caso ele falhe aqui eu tenho result = 'ABORTED', ou seja, ao invés de ficar verdinho ele vai ficar vermelho, "Opa, falhou nesse step aqui". Eu forcei o erro nesse step. Senão, eu mando uma mensagem.

[09:41] Da onde veio aquela URL? Eu coloquei ela hardcoded aqui, porque? Eu sei que o meu servidor de produção tá rodando nessa URL, passando o tokenCredentialId, esse é o slack-token quando a gente criou lá na configuração do Jenkins e aí ele exibiu a notificação pra gente aqui e a gente consegue acessar a nossa aplicação sem maiores problemas.

[10:03] Bom, antes de terminar essa aula, vamos sabotar o nosso projeto pra ver como é que o erro aparece. Tá vendo aqui que a gente derruba o container antigo? O que a gente vai fazer? Vamos mudar isso aqui, vamos tentar derrubar um container django-todolist-dev1. O que vai acontecer agora? Nosso Job já executou o container, ele tá em produção, eu consigo acessar. Se eu vier aqui e clicar, tá funcionando minha aplicação.

[10:32] Se eu tentar subir um container de novo, na mesma porta, o que será que vai acontecer? Vamos mudar isso aqui. Vamos abrir nosso pipeline aqui e aqui nesse step a gente vai derrubar o container 1, que não existe. Vamos salvar e vamos construir agora.

[10:53] Então qual que é o parâmetro da imagem? A gente já passou naquele anterior, vamos construir. Vamos ver o que que vai acontecer. Ele tá executando, tá carregando o ambiente.

[11:04] Opa, falhou, olha o que aconteceu. Esses aqui, se vocês olharem, eles estão meio laranjinha, significa o seguinte: esse step funcionou, mas porque que ele não tá verde? Porque na hora de eu subir o container novo, falhou.

[11:18] Primeiro vamos olhar se a gente realmente recebeu a mensagem de que ele falhou. Olha lá, "FALHA, Não foi possível subir o container". Ele até dá pra gente qual que é o job que ele falhou. Se eu clicar aqui, ele vai pedir pra eu autenticar no Jenkins, a gente vai autenticar. E aqui ele falou que falhou, a gente vai clicar aqui e vai ver o output do console.

[11:45] Aqui embaixo ele tá como aborted, olha o erro aqui: "Conflito, o container já existe com esse nome". Ele nem chegou a validar a porta porque o container tem o mesmo nome.

[11:56] Então aqui nós conseguimos ver exatamente o erro. Eu não precisaria ir no Slack pra ver o erro, aqui mesmo no meu painel, se eu clicar em logs, ele vai falar pra mim o seguinte: "Eu já tenho o mesmo nome de container", porque o nosso passo anterior falhou. Pra corrigir isso aqui a gente simplesmente remove aquele caractere especial que a gente adicionou na hora de remover o container, ele vai conseguir deployar o código novo.

[12:23] Bom, é isso. Na próxima aula nós vamos, agora, integrar os dois jobs, o primeiro que é o principal, que faz o build, com esse job de desenvolvimento. Ou seja, processo anterior vai passar o nome da imagem pra esse job agora. E também a gente vai trabalhar com o job de produção.

[12:42] A gente se vê na próxima aula.