03

Integração dos jobs

Transcrição

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

```
# Post build actions para os 3 jobs

Job: jenkins-todo-list-principal > Ações de pós-build > Trigger parameterized buld on other projects
    Projects to build: todo-list-desenvolvimento
    # Add parameters > Predefined parameters
        image=${image}

Job: todo-list-desenvolvimento

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

        stages {
            stage('Carregando o ENV de desenvolvimento') {
                steps {
                    configFileProvider([configFile(fileId: '2ed9697c-45fc-4713-a131-53bdbeea2ae6', v
                        sh 'cat $env > .env'
                    }
                }
            }
            stage('Derrubando o container antigo') {
                steps {
                    script {
                        try {
                            sh 'docker rm -f django-todolist-dev'
                        } 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:
                }
            }
            stage ('Fazer o deploy em producao?') {
                steps {
                    script {
                        slackSend (color: 'warning', message: "Para aplicar a mudança em produção, a
                        timeout(time: 10, unit: 'MINUTES') {
                            input(id: "Deploy Gate", message: "Deploy em produção?", ok: 'Deploy')
                        }
                    }
                }
            }
            stage (deploy) {
                steps {
                    script {
                        try {
                            build job: 'todo-list-producao', parameters: [[$class: 'StringParameter\
                        } catch (Exception e) {
                            slackSend (color: 'error', message: "[ FALHA ] Não foi possivel subir o
                            sh "echo $e"
                            currentBuild.result = 'ABORTED'
                            error('Erro')
                        }
                    }
                }
            }
        }
    }
```

[00:00] Tudo bem pessoal? Vamos pra mais uma aula? Agora a gente vai aprender a integrar os jobs que a gente criou, passando parâmetros de um pro outro.

[00:09] Então, primeiro job que a gente vai fazer a configuração é no principal. A gente vem aqui em configurar, aqui embaixo nas opções de pós-build a gente vai escolher a opção de fazer o trigger num build parametrizado pra outros projetos. Clicamos aqui, aí ele vai perguntar o seguinte, então agora vamos escolher qual job que o principal vai disparar. Clicando nele aqui a gente vai colocar aqui todo-list-desenvolvimento, aí ele vai ser o job que vai passar o parâmetro.

[00:48] E a gente vai fazer o seguinte: vai adicionar um parâmetro que vai ser passado pro próximo job, e ele vai ser pré-definido. Qual vai ser esse parâmetro? Vai ser o image que a gente sabe que ele espera no próximo lado.

[01:07] E qual vai ser o valor desse parâmetro? Image também só que agora a gente coloca em dólar, por quê? Esse image vai referenciar desse parâmetro aqui. Esse vai ser o inicial, daqui pra frente todos os jobs que forem acionados na cadeia, que nós definimos, vai receber essa mesma imagem como parâmetro.

[1:32] A gente dá um salvar. Então agora a gente vai configurar o segundo job que é o job de desenvolvimento que vai subir os nossos containers de dev. Então a gente volta aqui, dentro do job de desenvolvimento a gente vai em Configurar e nós faremos duas alterações aqui.

[01:52] A primeira alteração, assim como no job passado, o próximo job que vai ser chamado pra fazer a configuração de produção. Só que pra que isso aconteça, como esse job ele é escrito em Groovy, eu preciso trocar o arquivo Groovy que a gente tá usando até agora.

[02:11] O que tem de diferente nesse arquivo novo? Deixa eu abrir aqui pra mostrar pra vocês. Bom, aqui no arquivo, no começo é igual ao que a gente já tinha usado, a gente só tem alguns estágios novos.

[02:23] O primeiro estágio é o seguinte: eu vou fazer uma pergunta pro usuário "Olha, você quer fazer o deploy em produção?" Isso vai aparecer na interface do Jenkins. Quais são os steps desse estágio aqui? Eu vou mandar uma mensagem de warning lá no slack e vou avisar o usuário: "Olha, você quer aplicar a mudança em produção? Você tem uma janela de 10 minutos pra fazer isso". Isso aqui é um exemplo somente. E aí ele dá a URL, a qual o usuário acessa pra aceitar ou não aceitar o deploy, passando lá nossas credenciais do slack token do mesmo jeito.

[02:57] Como é que eu defino esse tempo e como é que eu ponho essa mensagem na tela pro usuário? Eu uso uma função de time-out com tempo de 10 minutos, é o tempo e a unidade que eu escolhi, vocês podem mudar de vocês no projeto de vocês não tem problema nenhum, e aí eu coloco um input.

[03:13] O input, o que que ele vai fazer? Ele vai mostrar um pop-up dentro daquele job perguntando pro cara: "Você quer ou não quer fazer o deploy?". E aí ele vai fazer o seguinte: qual é a mensagem? "Você quer fazer o deploy em produção?". Se clicar em ok, o que acontece? Ele continua o pipeline dentro do Groovy.

[03:31] E qual que é o próximo passo? É o deploy. Se você não clicar pra fazer o deploy em produção, ele vai abortar o seu processo, ele já fez o deploy em desenvolvimento, seu job vai ficar marcado como aborted, não como erro.

[03:44] E aí dentro desse estágio de deploy, mais um stage, eu tenho uma função nova que vocês não viram ainda, que é como eu chamo o próximo job via Groovy.

[03:56] Lá no primeiro job a gente fez via interface gráfica, aqui a gente chama a função build job, passa o nome do job que é o todo-list-producao, passa o parâmetro que é o nome da imagem, então aqui a gente passa como parâmetros do tipo string, o nome do parâmetro é imagem, e o valor dele é o dockerImage que a gente já sabe, porque lá em cima em ambientes a gente já recebeu do primeiro job a mesma imagem.

[04:25] É aquela ideia que a gente conversou, você constrói uma vez só e vai até o final com o seu artefato.

[04:32] E agora que a gente fez o deploy em produção, a gente vai ter as duas aplicações rodando paralelamente. Vamos ver como isso funciona na prática?

[04:40] Bom, pra isso tudo funcionar pegamos todo o nosso script aqui, substituímos o script que a gente tinha que era um pouquinho menor e salvamos. Visualizando aqui nosso pipeline completo.

[04:59] Nós vamos agora fazer o commit no código, o código vai ser atualizado aqui no Jenkins, o Jenkins vai observar o meu GitHub, ele vai fazer o build da imagem, vai registrar a imagem no Docker Hub, vai escutar os testes da imagem, vai notificar o resultado no Slack, se for com sucesso ele vai dar o trigger no job de Dev, o job de Dev vai subir, vai notificar qual que é a URL e vai perguntar pro usuário "Você quer ou não colocar em produção?".

[05:30] Na interface do Jenkins a gente vai escolher sim ou não, e aí vai determinar se esse job vai ser executado ou não vai. Então vamos lá.

[05:41] Bom, voltamos aqui então no nosso ambiente que é dentro do nosso Vagrant da nossa máquina virtual, a gente acessou o diretório Vagrant e o diretório do código-fonte da aplicação, que foi aquele primeiro que a gente acessou lá nas primeiras aulas. E agora a gente vai fazer uma alteração nesse código. Lembrando que tá tudo configurado e conectado com GitHub.

[06:02] Então a gente vai editar dentro de templates/registration/login.html e, pra exemplificar, vamos trocar aqui de Login pra Entrar, é só um exemplo. Salvamos. Então agora a gente vai fazer o commit desse código no GitHub.

[06:25] Primeiro a gente adiciona o código. Limpar a tela aqui pra vocês. Agora a gente adicionou o código, a gente vai gerar a mensagem de commit, git commit -m "Alterando o valor do botao de entrar", é só um exemplo. E agora a gente vai fazer o push pro nosso repositório. Então a gente vai digitar "git push origin master". Feito isso, a gente vai esperar mensagem de confirmação, a gente volta lá pro Jenkins pra observar que todos os três jobs vão ser executados automaticamente.

[07:14] A gente teve um probleminha na rede aqui, então a gente vai reexecutar o comando "git push origin master", ele vai fazer o push do código pro GitHub, assim que terminar a gente volta.

[07:32] Legal, acabou o nosso commit aqui, então agora a gente vai aqui pros nossos pipelines. Na visão geral do Jenkins pra visualizar como é que ele tá fazendo esse fluxo inteiro. Automaticamente ele startou o build do job principal. A gente não tá alterando nada, isso aqui é tudo automático.

[07:58] Reparem o seguinte, ele terminou de executar o job principal e automaticamente ele chamou o job de desenvolvimento, só que ele parou no step aqui: "Fazer o deploy em produção?"

[08:11] Se a gente clicar aqui no nosso job, a gente vai perceber que nesse step de deploy em produção, se eu parar com o mouse em cima ele vai perguntar: "Você quer ou não colocar essa aplicação em produção?". A gente vai clicar em deploy e aí ele vai chamar o próximo job que é o job todo-list-producao que vai fazer o que a gente já tinha configurado pra ele fazer.

[08:33] Terminou, executou com sucesso. Se a gente olhar aqui no nosso Slack, ele começou o build disponível de dev pra gente, avisou que a mudança em produção tinha 10 minutos de janela e deu a URL pra fazer o acesso de aprovação ou não, que é a mesma URL que abre essa área pra gente. E aí ele avisou: "Olha, nosso job tá em produção".

[09:00] Vamos validar agora às duas URL que a gente produziu. A primeira é a porta 81 da URL de desenvolvimento. Então a gente acessa a aplicação aqui, a senha nossa é mestre123. Entrei, esse aqui é o meu ambiente de desenvolvimento, meu banco de dados inclusive de desenvolvimento.

[09:22] E o de produção, qual que é? O de produção é a mesma URL só que sem a porta 81, só a URL mesmo. A gente acessa com mestre123 e pronto, são duas aplicações rodando na mesma máquina que foram buildadas automaticamente, foram deployadas automaticamente, só que em portas diferentes pelo Jenkins.

[09:48] Vamos dar uma olhada no Docker Hub só pra gente ver a nossa imagem registrada lá, "hub.docker.com/u/aluracursos". A gente vai ver a nossa imagem. A nossa imagem foi deployada há 12 minutos atrás, que foi quando a gente executou o último job na gravação aqui. Eu já tenho cinco pulls na minha imagem, ou seja, cinco vezes ela foi executada pra rodar os meus containers.

[10:25] Bom, na próxima aula a gente vai trabalhar agora com qualidade de código. Nós vamos criar um job pra fazer a análise e a cobertura do nosso código, e gerar relatórios pra que a gente possa melhorar ou corrigir eventuais mau usos da linguagem, no caso do Django ou de qualquer linguagem que vocês trabalharem.

[10:45] Nós trabalharemos com aplicação SonarQube integrado com o Jenkins. A gente se vê na próxima aula.