05

Job para scan do código

Transcrição

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

```
# Criar um job para Coverage com o nome: todo-list-sonarqube
    # Gerenciamento de código fonte > Git
        git: git@github.com:alura-cursos/jenkins-todo-list.git (Selecione as mesmas credenciais)
        branch: master
        Pool SCM: * * * * *
        Delete workspace before build starts
        Execute Script:

    # Build > Adicionar passo no build > Executar Shell

        #!/bin/bash
        # Baixando o Sonarqube
        wget https://s3.amazonaws.com/caelum-online-public/1110-jenkins/05/sonar-scanner-cli-3.3.0.1

        # Descompactando o scanner
        unzip sonar-scanner-cli-3.3.0.1492-linux.zip

        # Rodando o Scanner
        ./sonar-scanner-3.3.0.1492-linux/bin/sonar-scanner   -X \
          -Dsonar.projectKey=jenkins-todolist \
          -Dsonar.sources=. \
          -Dsonar.host.url=http://192.168.33.10:9000 \
          -Dsonar.login=<seu token>
```

[00:00] Bom pessoal, agora tá na hora da gente criar o nosso job pra fazer o scanner do código, pra fazer a cobertura de qualidade. Então a gente vai criar um novo job com o nome todo-list-sonarqube e do tipo freestyle.

[00:17] Agora que a gente criou o job, quais são as opções que a gente vai marcar? A primeira opção é que é um código git que está nesse repositório aqui, que é o nosso repositório desde o começo. Ou seja, tanto o nosso pipeline quantos o SonarQube vão escanear o mesmo código e vão usar a chave SSH que a gente criou.

[00:39] Sempre olhando a Branch Master, isso é sempre importante da gente colocar dependendo da branch que vocês forem trabalhar, nesse nosso caso é a Branch Master.

[00:46] Qual que é o trigger build dele? Vai ser a cada minuto ele vai observar se existe algum tipo de alteração dentro do nosso código fonte. Se existir, ele dispara. Ele vai também ter que deletar o projeto antes de começar um novo, ele vai apagar o workspace e criar um workspace novo dentro aqui do Jenkins pra evitar sujeira no código. E aí o build step dele, que é o scanner em si, vai ser um shell que contém isso aqui. É essa parte aqui que tá selecionada. O que isso aqui corresponde?

[01:25] Primeiro a gente baixa o SonarScanner CLI, que tá disponível também no site do SonarQube mas a gente disponibilizou pra vocês nesse armazenamento aqui pra sempre estar disponível.

[01:38] Depois, a gente vai descompactar esse scanner e vai executar o SonarScanner, que tá dentro de bin desse diretório, com esses parâmetros aqui que são os parâmetros que nós copiamos na instalação do SonarQube, que é: a chave do projeto; qual é o source dele ou seja, ele vai analisar qual diretório, é daqui pra baixo, ele vai fazer o scanner em todos os diretórios dentro do meu repositório, vai registrar o resultado nesse servidor aqui.

[02:08] Esse resultado é consequência do scanner. Então o scanner funciona, gera uma massa de dados e registra isso no servidor usando essa chave de autenticação. Então a gente copia esse script, cola e salva.

[02:26] Então, daqui um minuto, ele vai executar esse build e a gente vai ver o resultado disso daí. Enquanto ele não executa, tem uma observação legal pra ser feita. Se, eventualmente, vocês não tiverem espaço de memória na máquina de vocês, vocês podem disponibilizar mais recurso só alterando o Vagrant file, porque vocês eventualmente vão encontrar dificuldades dentro da quantidade de aplicações que estão subindo. A gente já volta.

[02:55] Bom, o scanner terminou. Vamos analisar nosso job aqui e vamos dar uma olhadinha no console rapidinho dentro do Jenkins. Olha, ele fez o download do arquivo .zip, depois ele deu unzip no arquivo, e aí ele chamou a execução do SonarQube, passando os parâmetros que a gente pediu.

[03:22] Voltando aqui no SonarQube, a gente vai clicar no nosso projeto e ele vai carregar a interface com a cobertura que ele fez. Então ele encontrou zero bugs, zero vulnerabilidades, ele encontrou um Code Smell.

[03:40] Esse Code Smell, se a gente clicar aqui no Code Smell que ele encontrou, ele vai falar pra gente o seguinte: nesse arquivo, nessa classe de teste, ele não gostou muito do jeito que a gente nomeou a função pra fazer o setup da classe de teste.

[03:58] Se nós corrigirmos isso daí pra sugestão que ele deu, isso é uma sugestão, que é separar com underscore que é o padrão de nomenclatura que ele pede, a nossa classe de teste para de funcionar e a gente quebra o nosso pipeline.

[04:13] Então fica aí como tarefa pra vocês testarem. Se quiserem corrigir essa classe e fazer o push vocês vão perceber que o pipeline de vocês vai parar de funcionar.

[04:24] Pessoal, voltando aqui no overview, só alguns resultados que ele passou. Ele passou no teste de qualidade, é um teste bem básico. O SonarQube tem várias outras maneiras de fazer essa cobertura mais profundas, mas é uma introdução bem legal pra vocês. Ele encontrou 314 linhas de código, qual que é a linguagem predominante, o que ele achou de XML, nesse caso é HTML que eventualmente é a mesma coisa.

[04:50] Então essa aqui é uma maneira de se integrar o Jenkins com o SonarQube. Então, pra essa aula, a gente acabou.