06

Deploy tradicional com arquivo war

Transcrição

[00:00] Oi, bem-vindos de volta ao curso de Spring Boot na Alura. Já aprendemos como fazer para gerar um Build do projeto e substituir senhas que estavam no arquivo “properties” por variáveis de ambiente e simular um ambiente de produção.

[00:15] Outra coisa importante para vermos é em relação a justamente esse processo de Build e de Deploy de uma aplicação com Spring Boot. E é um pouco diferente do modelo tradicional, de aplicações web com Java de antigamente.

[00:30] Em uma aplicação web tradicional, nós gerávamos um arquivo WAR, que era o Web Archive. E esse WAR continha todo o conteúdo da pasta Web Content, Web App da aplicação. E nós pegávamos esse WAR e jogávamos dentro do diretório do servidor, Tomcat, Jetty, WildFly, Websphere, qualquer que seja o servidor de aplicações.

[00:53] No Spring Boot vocês puderam perceber que é diferente. Nós não geramos um WAR, nós não colocamos esse WAR dentro do servidor. Nós geramos um JAR. É como se fosse uma aplicação Standalone.

[01:05] E quando rodamos esse JAR pela linha de comando internamente tem o servidor, que por padrão é o Tomcat. Mas até tem como você trocar para usar o Jetty ou outros servidores.

[01:16] Porém, se o projeto em que você está trabalhando por algum motivo tem uma restrição e você não pode, não deve gerar um JAR, você deve gerar

um WAR para ele ser *deployado* dentro de um servidor, conforme era feito antigamente, é possível você fazer isso.

[01:32] Então no Spring Boot existe essa possibilidade de você fazer o Deploy do jeito tradicional, utilizando um arquivo WAR. Então nesse vídeo vou mostrar justamente como você pode fazer isso, como em vez de gerar um JAR eu gero um WAR para ser *deployado* dentro de um servidor de uma maneira tradicional, como era feito antigamente.

[01:50] Para isso precisamos fazer três mudanças no projeto. As duas primeiras mudanças são no arquivo `pom.xml`, então vou abri-lo. Em cima, no começo do arquivo, onde tem o `groupId`, `artifactId`, em algum lugar precisamos colocar mais uma *tag*, chamada `packaging`.

[02:12] E por padrão, como não tinha essa *tag*, o padrão é JAR. Por isso que quando geramos com o Maven ele gerou um arquivo JAR. Então a primeira coisa que temos que fazer é colocar a *tag* `packaging` e trocar para WAR: `<packaging>war</packaging>`.

[02:24] Eu estou dizendo para o Maven que quando for gerar o Build desse projeto é para gerar um arquivo WAR. Essa é a primeira mudança que temos que fazer.

[02:31] A segunda mudança é dentro ainda do arquivo `pom.xml`, nas dependências do projeto. Eu vou colocar mais uma dependência. Eu já estou com ela copiada, vou colar. Vou dar um "Ctrl + Shift + F" para ele formatar.

[02:50] Então temos que adicionar essa dependência: `org.springframework.boot`, `spring-boot-starter-tomcat`. É para adicionar a dependência do Tomcat.

[03:01] Só que na verdade ela já está no projeto. Só que agora vem o segredo: na *tag* `<scope>`, nós colocamos como `provided` para falar para o Maven que quando ele for gerar o Build do projeto ele não deve incluir a biblioteca do

Tomcat no WAR, porque ela vai ser *provided*, vai ser provida pelo próprio servidor.

[03:21] Como vamos jogar o WAR dentro do Tomcat, dentro do Tomcat já estão os JAR's do próprio Tomcat. Então precisamos colocar essa dependência só para dizer que é para usar o Tomcat como *provided*, não para ele ser gerado junto com o WAR, senão as dependências do Tomcat vão ficar duplicadas. Então essa é a segunda mudança, vou salvar.

[03:41] A terceira mudança é naquela classe Main, que usamos para rodar o projeto. Temos que vir nessa classe e herdar de outra classe do Spring Boot, chamada `SpringBootServletInitializer`.

[03:58] Então nossa classe `main` vai ter que herdar dessa classe do Spring Boot para inicializar, deixar ser possível configurar a Servlet 3.0 dentro do servidor, do Tomcat, no caso.

[04:09] Então precisamos dar um `extends` nessa classe e precisamos ainda sobrescrever um método. Vou colocar embaixo do método `main`. Tem um método chamado `configure`. Vou dar um "Ctrl + barra de espaço" para ele completar. Esse é o método que precisamos sobrescrever.

[04:27] É um método `protected SpringApplicationBuilder configure` e ele recebe como parâmetro esse `SpringApplicationBuilder`. Nós precisamos sobrescrever esse método

[04:38] Por padrão veio esse `return`, mas podemos tirar. Em vez de colocar o `super.configure`, temos que chamar o parâmetro, que é esse tal de Builder. Tem um método chamado `sources`. Nós pegamos esse método e passamos a nossa classe: `return builder.sources(ForumApplication.class);`.

[05:01] Então temos que sobrescrever esse método `configure` e pegar o parâmetro que ele recebe, chamar o método `sources` passando a própria classe `ForumApplication`.

[05:09] Feito isso, salvei. São só essas três mudanças que você precisa fazer. Na classe `main`, herdar dessa classe `SpringBootServletInitializer`, sobrescrevendo o método `configure`, chamando o método `sources` da classe `builder` e passando a própria classe `ForumApplication`.

[05:26] E no `pom.xml` trocar o `<packaging>` para WAR e dizer que a dependência do Tomcat é `provided`, que ela vai ser provida pelo próprio servidor. Pronto, só isso já basta. Agora vamos fazer o teste.

[05:37] Eu vou no terminal. Já estou no fórum. Vou rodar o `mvn clean package`, aquele mesmo comando. Não vai mudar nada, é o Maven da mesma maneira.

[05:48] Só que agora o Maven vai fazer o Build do projeto, vai compilar as nossas classes, vai compilar as classes de teste, vai rodar os testes e vai fazer o build do projeto. Só que agora ele vai ver que o `packaging` é WAR, então ele não vai mais gerar um JAR, vai gerar um arquivo `.war`.

[06:04] Ele vai ver que colocamos a dependência do Tomcat como `provided`, então ele não vai incluir as dependências do Tomcat. E a classe `main` está com aquele `extends`, com aquele método `configure` sobrescrito.

[06:15] E a partir de agora pegamos esse WAR e ele tem que ser *deployado* do jeito tradicional. Então a equipe de infraestrutura que vai cuidar da parte do Deploy vai pegar esse WAR e vai jogá-lo dentro do servidor, do Tomcat, do Jetty, do WildFly, seja lá qual for o servidor de aplicação.

[06:33] Então perceba que embora no Spring Boot o padrão seja gerar um arquivo JAR, que é muito mais simples, muito mais leve e já tem um servidor embutido, também é possível gerar o arquivo WAR tradicional, que era comum antigamente nas aplicações web com Java.

[06:53] Então o Spring deixou isso flexível. Se você por algum motivo precisa ainda gerar o WAR, quer fazer o Deploy do jeito tradicional, não tem problem O Spring te permite fazer isso.

[07:04] A única coisa que você vai ter que fazer são aquelas mudanças no código. Mas foi bem simples também, só mexemos no `pom.xml`, demos `extends` na classe `main`, sobrescrevemos o método `configure`. Foi fácil, em 2 minutos resolvemos o problema.

[07:18] E a partir de agora o Build sempre que for realizado vai gerar um arquivo `.war`. Vamos só esperar ele terminar e ver se ele gerou o arquivo WAR.

[07:30] Feito isso, acabou e você já gerou o WAR, e esse WAR vai ser *deployado* em qualquer servidor que suporte a especificação de Servlet 3.0. Os servidores atuais há um bom tempo já dão suporta à especificação de Servlet 3.0.

[07:46] Mas para rodar o projeto no Eclipse, na sua IDE, não muda nada. Você continua rodando da mesma maneira, você roda aquela classe `main` da mesma maneira. Para ambiente de desenvolvimento, do ponto de vista da pessoa que está programando, que está desenvolvendo, não muda nada. Ela roda aquele Main da mesma maneira, os recursos continuam iguais.

[08:09] A única mudança mesmo é na hora de gerar o Build com o Maven, que ele vai gerar um WAR. Teve essa mudança no código que fizemos, mas ela não impacta nada do ponto de vista da pessoa que está desenvolvendo. Só vai mudar na hora de fazer o Deploy.

[08:22] E ele já imprimiu, está gerando um WAR. Na pasta `target` ele está gerando o `forum.war`. Então ele vai chamar os processos do Maven, e apareceu `BUILD SUCCESS`. Vamos dar um `ls` na pasta `target` e está lá o `forum.war`.

[08:39] Então espero que vocês tenham gostado desse vídeo e tenham aprendido como funciona caso você queira gerar o WAR tradicional, para ser *deployado* dentro de um servidor web tradicional, como era feito antigamente. Espero vocês na próxima aula. Um abraço e até lá.