Deploying Spring Boot App to JBoss Wildfly

by Tapas Joshi · Oct. 02, 19 · Java Zone · Tutorial

Learn how to deploy Spring Boot applications to JBoss Wildfly

We know that when we create any Spring Boot application, it comes with an embedded Tomcat server, and we have no need to set up the server additionally. This happens because, in the Spring Boot parent project, it has the dependency for Tomcat, and when we start the Spring Boot application, it bootstraps the application with the Tomcat server itself.

You may also like: Welcome to Spring Boot (With Embedded Tomcat Features)

But sometimes, our requirements might include creating a war file and then deploying it to any web application server, like JBoss. In this article, we will see how to package a Spring Boot application and deploy it to any external server. Here, we will be using JBoss to perform the following tasks:

- Create a Spring Boot application
- Remove the embedded Tomcat server from the application
- Generate a war file that can be deployed to the JBoss server

Creating a Spring Boot Application

- You can create a Spring Boot application by generating a project from **start.spring.io.**
- Once downloaded, you can import the Maven project to your IDE (Spring Tool Suite is recommended) as an existing Maven project.
- Then, implement a Maven build to fetch the required dependencies.

Separate the Embedded Tomcat Server From the Spring Boot App

- First of all, open the pom.xml and add the Tomcat dependency. Next, you can change the scope to the provided. When we make the scope as provided that means this dependency will be available during compile time only and will not be available at run-time. This is because we need this dependency in compile time to compile the web-component-related files.
- Adding the Tomcat dependency and changing the scope to provided will give us the following:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

```
<!-- Make sure that you changed the packaging to war -->
 <packaging>war</packaging>
```

- Then, extend the `SpringBootServletInitializer` class in your Spring Boot main class and override the configure method. We need to do this to initialize the application in any other servlet container like JBoss.

- You can see the code below that extends the `SpringBootServletInitializer` and overrides the configure method.

```
@SpringBootApplication
@EnableSwagger2
public class SpringBootDemoApplication extends SpringBootServletInitializer {

  @Override
  protected SpringApplicationBuilder configure(SpringApplicationBuilder builder){
    return builder.sources(SpringBootDemoApplication.class);
  }

  public static void main(String[] args){
    SpringApplication.run(SpringBootDemoApplication.class, args);
  }
}
```

- With this configuration, the application is ready to deploy to any external application server.

- Once this is done, you can package your Spring Boot application as a war and deploy it to the JBoss server. To do that, you need to run `mvn clean package`, which will generate the war file.

That's all for now. We hope you enjoyed this short demonstration. Be sure to leave thoughts and questions in the comments section.

Further Reading

Welcome to Spring Boot (With Embedded Tomcat Features)

Introducing Spring Boot

[DZone Refcard] Getting Started With JBoss