Spring Boot: ¿war o jar? Ambos

Índice de contenidos

* [1. Entorno](https://www.adictosaltrabajo.com/2018/12/13/spring-boot-war-o-jar-ambos/#01)
* [2. Introducción](https://www.adictosaltrabajo.com/2018/12/13/spring-boot-war-o-jar-ambos/#02)
* [3. Vamos al lío](https://www.adictosaltrabajo.com/2018/12/13/spring-boot-war-o-jar-ambos/#03)
* [4. Conclusiones](https://www.adictosaltrabajo.com/2018/12/13/spring-boot-war-o-jar-ambos/#04)

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

* Hardware: Slimbook Pro 2 13.3″ (Intel Core i7, 32GB RAM)
* Sistema Operativo: LUbuntu 18.04
* Spring Boot 1.5.8.RELEASE

2. Introducción

Cuando en el back estamos trabajando con Spring Boot para un producto que tiene que ser potencialmente desplegable en distintos entornos nos podemos encontrar estas situaciones: entornos 100% Docker con Kubernetes, máquinas Linux donde poder ejecutar nuestros .jar de Spring Boot y despliegues en clásicos servidores de aplicaciones tipo WebLogic, Wildfly, JBoss, etc…

Las dos primeras situaciones se resuelven con el jar de Spring Boot, en Docker se embebe dentro de un contenedor y se ejecuta con Java y en una máquina Linux lo podemos ejecutar directamente con Java (o incluso configurarlo como servicio de la máquina).

Pero para el tercer escenario no nos vale con un jar, ya que estos pesados servidores de aplicaciones requieren de al menos un war para el despliegue de la aplicación.

En este tutorial vamos a ver cómo tenemos que configurar nuestro proyecto de Spring Boot para soportar estos escenarios sin necesidad de cambiar código, solo cambiando el perfil de Maven.

3. Vamos al lío

Partimos de que tenemos un proyecto de Spring Boot con la configuración por defecto, es decir, que se despliega como un jar y que este proyecto es un proyecto web.

Lo primero que tenemos que hacer es editar la clase principal del proyecto para que herede de la clase SpringBootServletInitializer y añadirle el método configure que vemos a continuación:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package com.autentia.training.devops;    import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.builder.SpringApplicationBuilder;  import org.springframework.boot.web.support.SpringBootServletInitializer;    @SpringBootApplication  public class DevopsAppApplication extends SpringBootServletInitializer{        @Override      protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {          return application.sources(DevopsAppApplication.class);      }        public static void main(String[] args) {           SpringApplication.run(DevopsAppApplication.class, args);      }    } |

Ahora vamos a configurar el pom.xml para establecer la configuración para los dos perfiles: uno jar que permita el despliegue con el servidor embebido y otro war que permita su despliegue en los servidores de aplicaciones.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | ...  <packaging>${packaging.type}</packaging>  ...  <profiles>     <profile>         <id>jar</id>         <properties>             <packaging.type>jar</packaging.type>             <spring.profiles.active>dev</spring.profiles.active>         </properties>     </profile>     <profile>         <id>war</id>         <properties>              <packaging.type>war</packaging.type>              <spring.profiles.active>ds</spring.profiles.active>         </properties>         <dependencies>              <dependency>                  <groupId>org.springframework.boot</groupId>                  <artifactId>spring-boot-starter-tomcat</artifactId>                  <scope>provided</scope>              </dependency>         </dependencies>     </profile>  </profiles> |

Como ves, hacemos el cálculo de la propiedad del tipo de packaging del proyecto de forma que si el perfil es «jar» entonces generamos un jar que desplegamos en el tomcat embebido de Spring Boot y si el perfil es «war» generamos un war y le decimos que el servidor de Tomcat debe ser proporcionado, de esta forma no intentará arrancar la aplicación en el servidor de Tomcat embebido.

Lo más probable es que ambos perfiles lleven perfiles de Spring Boot distintos para en el caso de «jar» hacer uso de un datasource local, y para el caso de «war» hacer uso del datasource que previamente tiene que estar definido en el servidor de aplicaciones donde se vaya a desplegar el war.

Para aplicar uno u otro perfil de Spring Boot en función del perfil de Maven, tenemos que añadir lo siguiente al fichero application.properties de nuestro proyecto:

|  |  |
| --- | --- |
| 1 | spring.profiles.active=@spring.profiles.active@ |

Si todo es correcto cuando quieras un jar solo tendrás que ejecutar:

|  |  |
| --- | --- |
| 1 | $> mvn clean package -Pjar |

Comprobarás que en la carpeta «target» se ha generado efectivamente un jar.

Y si lo que quieres es desplegar en un servidor de aplicaciones:

|  |  |
| --- | --- |
| 1 | $> mvn clean package -Pwar |

Comprobarás que en la carpeta «target» se ha generado en esta ocasión un war, que poder desplegar dependiendo del servidor de aplicaciones objetivo.

4. Conclusiones

Con esta simple configuración podemos dar servicio a todos los escenarios de despliegue sin tener que cambiar una y otra vez la configuración del proyecto.