Embedded Servers in Spring

Estimated time needed: 4 minutes

Learning objectives:

- Describe the role of embedded servers in Spring MVC applications
- Identify different embedded servers available for Spring Boot
- Configure and modify settings for an embedded server in Spring Boot
- Run a Spring Boot application using an embedded server

Why use embedded servers?

If you're developing a Spring MVC application, embedded servers can simplify deployment and enhance portability. Instead of relying on a separate web server, your application includes its lightweight server, making development and deployment more efficient. But what exactly are embedded servers, and how do they improve the workflow?

Understanding embedded servers

Traditionally, Java web applications required standalone web servers like Tomcat, Jetty, or WildFly. Developers had to build a Web Application Archive (WAR) file and deploy it separately. Embedded servers eliminate this step by bundling the server within the application itself. This turns the Spring MVC application into a self-contained executable that includes the code and the required server.

Benefits of embedded servers

1. **Simplified development workflow:**
   Developers can run applications directly from an IDE or command line without configuring external servers or deploying WAR files. Simply running the application starts the embedded server.

2. **Consistent environments:**
   Embedded servers ensure that all developers, testers, and production environments use the same server configuration, reducing compatibility issues.

3. **Microservices compatibility:**
   Embedded servers work well with microservices architectures, allowing each service to run independently without interfering with others.

4. **Easy deployment:**
   Applications packaged with embedded servers can be deployed as JAR files, simplifying containerization with tools like Docker.

Popular embedded servers for Spring Boot

1. **Tomcat (Default in Spring Boot):**
   Tomcat serves as the default embedded server for Spring Boot applications. It is stable, widely used, and performs well in most scenarios.

2. **Jetty:**
   Jetty has a smaller footprint and is ideal for applications that prioritize memory efficiency.

3. **Undertow:**
   Undertow, developed by JBoss, is designed for high-performance applications that handle large concurrent connections efficiently.

Getting started with embedded servers

Spring Boot includes an embedded server by default when you use the ***spring-boot-starter-web*** dependency. Here's a basic example of a Spring Boot application running on an embedded Tomcat server:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

Running this class automatically starts the embedded Tomcat server, typically on port 8080.

Configuring your embedded server

Spring Boot allows easy customization of embedded servers using ***application.properties*** or ***application.yml***:

```
# Change the port
server.port=9090
# Enable HTTPS
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=password
# Customize server threads
server.tomcat.max-threads=200
```

Summary

- Embedded servers eliminate the need for standalone web servers by bundling the server within the application.
- They simplify development and deployment by enabling direct execution without external configuration.
- Popular embedded servers for Spring Boot include Tomcat, Jetty, and Undertow, each with unique benefits.
- Spring Boot includes an embedded server by default when using the spring-boot-starter-web dependency.
- Configuration is straightforward with properties files, allowing ports, SSL, and thread management to be customized.