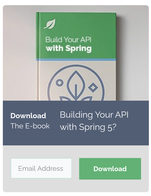
# Deploy a Spring Boot WAR into a Tomcat Server

Last modified: March 21, 2020

by baeldung

DevOps | Spring Boot | Tomcat










Spring Boot is a convention over configuration framework that allows us to set up a production-ready setup of a Spring project, and Tomcat is one of the most popular Java Servlet Containers.

By default, Spring Boot builds a standalone Java application that can run as a desktop application or be configured as a system service, but there are environments where we can't install a new service or run the application manually.

Opposite to standalone applications, Tomcat is installed as a service that can manage multiple applications within the same application process, avoiding the need for a specific setup for each application.

In this guide, we're going to create a simple Spring Boot application and adapt it to work within Tomcat.

## 2. Setting up a Spring Boot Application

We're going to setup a simple Spring Boot web application using one of the available starter templates:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.2.RELEASE</version>
    <relativePath/>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

**There's no need for additional configurations** beyond the standard *@SpringBootApplication* since Spring Boot takes care of the default setup.

We add a simple REST EndPoint to return some valid content for us:

```
@RestController
public class TomcatController {

    @GetMapping("/hello")
    public Collection<String> sayHello() {
        return IntStream.range(0, 10)
          .mapToObj(i -> "Hello number " + i)
          .collect(Collectors.toList());
    }
}
```

Now let's execute the application with *mvn spring-boot:run* and start a browser at *http://localhost:8080/hello* to check the results.

## 3. Creating a Spring Boot WAR

Servlet containers expect the applications to meet some contracts to be deployed. For Tomcat the contract is the Servlet API 3.0.

To have our application meeting this contract, we have to perform some small modifications in the source code.

First, we need to package a WAR application instead of a JAR. For this, we change *pom.xml* with the following content:

```
<packaging>war</packaging>
```

Now, let's modify the final *WAR* file name to avoid including version numbers:

```
<build>
    <finalName>${artifactId}</finalName>
    ...
</build>
```

Then, we're going to add the Tomcat dependency:

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

Finally, we initialize the Servlet context required by Tomcat by implementing the *SpringBootServletInitializer* interface:

```
@SpringBootApplication
public class SpringBootTomcatApplication extends SpringBootServletInitializer {
}
```

To build our Tomcat-deployable WAR application, we execute *the mvn clean package*. After that, our WAR file is generated at *target/spring-boot-tomcat.war* (assuming the Maven *artifactId* is "spring-boot-tomcat").

We should consider that this new setup makes our Spring Boot application a non-standalone application (if you would like to have it working in standalone mode again, remove the *provided* scope from the tomcat dependency).

## 4. Deploying the WAR to Tomcat

To have our WAR file deployed and running in Tomcat, we need to complete the following steps:

1. Download Apache Tomcat and unpackage it into a *tomcat* folder
2. Copy our WAR file from *target/spring-boot-tomcat.war* to the *tomcat/webapps/* folder
3. From a terminal navigate to *tomcat/bin* folder and execute
    3.1 *catalina.bat run* (on Windows)
    3.2 *catalina.sh run* (on Unix-based systems)
4. Go to *http://localhost:8080/spring-boot-tomcat/hello*

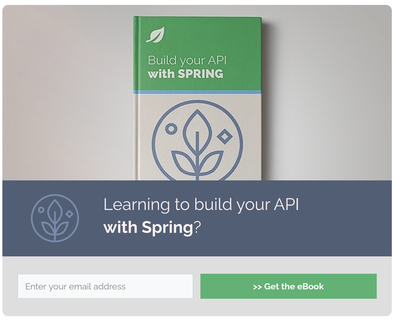
This has been a quick Tomcat setup, please check the guide on Tomcat Installation for a complete setup guide. There are also additional ways of deploying a WAR file to Tomcat.

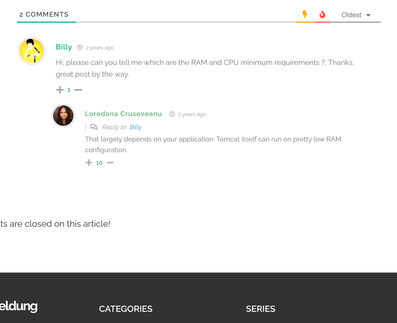

## 5. Conclusion

In this short guide, we created a simple Spring Boot application and turned it into a valid WAR application deployable on a Tomcat server.

As always, the full source code of the examples is available over on GitHub.






Comments are closed on this article!