**Part VII. Spring Boot CLI**

**66. Installing the CLI**

## Installing Spring Boot

Spring Boot can be used with “classic” Java development tools or installed as a command line tool. Either way, you need [Java SDK v1.8](https://www.java.com/) or higher. Before you begin, you should check your current Java installation by using the following command:

$ java -version

If you are new to Java development or if you want to experiment with Spring Boot, you might want to try the [Spring Boot CLI](https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html#getting-started-installing-the-cli) (Command Line Interface) first. Otherwise, read on for “classic” installation instructions.

## 10.1 Installation Instructions for the Java Developer

You can use Spring Boot in the same way as any standard Java library. To do so, include the appropriate spring-boot-\*.jar files on your classpath. Spring Boot does not require any special tools integration, so you can use any IDE or text editor. Also, there is nothing special about a Spring Boot application, so you can run and debug a Spring Boot application as you would any other Java program.

Although you could copy Spring Boot jars, we generally recommend that you use a build tool that supports dependency management (such as Maven or Gradle).

### 10.1.1 Maven Installation

Spring Boot is compatible with Apache Maven 3.3 or above. If you do not already have Maven installed, you can follow the instructions at [maven.apache.org](https://maven.apache.org/).

|  |
| --- |
| [Tip] |
| On many operating systems, Maven can be installed with a package manager. If you use OSX Homebrew, try brew install maven. Ubuntu users can run sudo apt-get install maven. Windows users with [Chocolatey](https://chocolatey.org/) can run choco install maven from an elevated (administrator) prompt. |

Spring Boot dependencies use the org.springframework.boot groupId. Typically, your Maven POM file inherits from the spring-boot-starter-parent project and declares dependencies to one or more [“Starters”](https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-starter). Spring Boot also provides an optional [Maven plugin](https://docs.spring.io/spring-boot/docs/current/reference/html/build-tool-plugins-maven-plugin.html) to create executable jars.

The following listing shows a typical pom.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>myproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

*<!-- Inherit defaults from Spring Boot -->*

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.4.RELEASE</version>

</parent>

*<!-- Add typical dependencies for a web application -->*

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

*<!-- Package as an executable jar -->*

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

|  |
| --- |
| [Tip] |
| The spring-boot-starter-parent is a great way to use Spring Boot, but it might not be suitable all of the time. Sometimes you may need to inherit from a different parent POM, or you might not like our default settings. In those cases, see [Section 13.2.2, “Using Spring Boot without the Parent POM”](https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-maven-without-a-parent) for an alternative solution that uses an import scope. |

### 10.1.2 Gradle Installation

Spring Boot is compatible with Gradle 4.4 and later. If you do not already have Gradle installed, you can follow the instructions at [gradle.org](https://gradle.org/).

Spring Boot dependencies can be declared by using the org.springframework.boot group. Typically, your project declares dependencies to one or more [“Starters”](https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-starter). Spring Boot provides a useful [Gradle plugin](https://docs.spring.io/spring-boot/docs/current/reference/html/build-tool-plugins-gradle-plugin.html" \o "72. Spring Boot Gradle Plugin) that can be used to simplify dependency declarations and to create executable jars.

**Gradle Wrapper**

The Gradle Wrapper provides a nice way of “obtaining” Gradle when you need to build a project. It is a small script and library that you commit alongside your code to bootstrap the build process. See [docs.gradle.org/4.2.1/userguide/gradle\_wrapper.html](https://docs.gradle.org/4.2.1/userguide/gradle_wrapper.html) for details.

More details on getting started with Spring Boot and Gradle can be found in the [Getting Started section](https://docs.spring.io/spring-boot/docs/2.1.4.RELEASE/gradle-plugin/reference/html/#getting-started) of the Gradle plugin’s reference guide.

## 10.2 Installing the Spring Boot CLI

The Spring Boot CLI (Command Line Interface) is a command line tool that you can use to quickly prototype with Spring. It lets you run [Groovy](http://groovy-lang.org/) scripts, which means that you have a familiar Java-like syntax without so much boilerplate code.

You do not need to use the CLI to work with Spring Boot, but it is definitely the quickest way to get a Spring application off the ground.

### 10.2.1 Manual Installation

You can download the Spring CLI distribution from the Spring software repository:

* [spring-boot-cli-2.1.4.RELEASE-bin.zip](https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.1.4.RELEASE/spring-boot-cli-2.1.4.RELEASE-bin.zip)
* [spring-boot-cli-2.1.4.RELEASE-bin.tar.gz](https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.1.4.RELEASE/spring-boot-cli-2.1.4.RELEASE-bin.tar.gz)

Cutting edge [snapshot distributions](https://repo.spring.io/snapshot/org/springframework/boot/spring-boot-cli/) are also available.

Once downloaded, follow the [INSTALL.txt](https://raw.github.com/spring-projects/spring-boot/v2.1.4.RELEASE/spring-boot-project/spring-boot-cli/src/main/content/INSTALL.txt) instructions from the unpacked archive. In summary, there is a spring script (spring.bat for Windows) in a bin/ directory in the .zip file. Alternatively, you can use java -jar with the .jar file (the script helps you to be sure that the classpath is set correctly).

### 10.2.2 Installation with SDKMAN!

SDKMAN! (The Software Development Kit Manager) can be used for managing multiple versions of various binary SDKs, including Groovy and the Spring Boot CLI. Get SDKMAN! from [sdkman.io](https://sdkman.io/) and install Spring Boot by using the following commands:

$ sdk install springboot

$ spring --version

Spring Boot v2.1.4.RELEASE

If you develop features for the CLI and want easy access to the version you built, use the following commands:

$ sdk install springboot dev /path/to/spring-boot/spring-boot-cli/target/spring-boot-cli-2.1.4.RELEASE-bin/spring-2.1.4.RELEASE/

$ sdk default springboot dev

$ spring --version

Spring CLI v2.1.4.RELEASE

The preceding instructions install a local instance of spring called the dev instance. It points at your target build location, so every time you rebuild Spring Boot,spring is up-to-date.

You can see it by running the following command:

$ sdk ls springboot

================================================================================

Available Springboot Versions

================================================================================

> + dev

\* 2.1.4.RELEASE

================================================================================

+ - local version

\* - installed

> - currently in use

================================================================================

### 10.2.3 OSX Homebrew Installation

If you are on a Mac and use [Homebrew](https://brew.sh/), you can install the Spring Boot CLI by using the following commands:

$ brew tap pivotal/tap

$ brew install springboot

Homebrew installs spring to /usr/local/bin.

|  |
| --- |
| [Note] |
| If you do not see the formula, your installation of brew might be out-of-date. In that case, run brew update and try again. |

### 10.2.4 MacPorts Installation

If you are on a Mac and use [MacPorts](https://www.macports.org/" \t "_top), you can install the Spring Boot CLI by using the following command:

$ sudo port install spring-boot-cli

### 10.2.5 Command-line Completion

The Spring Boot CLI includes scripts that provide command completion for the [BASH](https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29) and [zsh](https://en.wikipedia.org/wiki/Z_shell" \t "_top) shells. You can source the script (also named spring) in any shell or put it in your personal or system-wide bash completion initialization. On a Debian system, the system-wide scripts are in /shell-completion/bash and all scripts in that directory are executed when a new shell starts. For example, to run the script manually if you have installed by using SDKMAN!, use the following commands:

$ . ~/.sdkman/candidates/springboot/current/shell-completion/bash/spring

$ spring <HIT TAB HERE>

grab help jar run test version

|  |
| --- |
| [Note] |
| If you install the Spring Boot CLI by using Homebrew or MacPorts, the command-line completion scripts are automatically registered with your shell. |

### 10.2.6 Windows Scoop Installation

If you are on a Windows and use [Scoop](https://scoop.sh/), you can install the Spring Boot CLI by using the following commands:

> scoop bucket add extras

> scoop install springboot

Scoop installs spring to ~/scoop/apps/springboot/current/bin.

|  |
| --- |
| [Note] |
| If you do not see the app manifest, your installation of scoop might be out-of-date. In that case, run scoop update and try again. |

### 10.2.7 Quick-start Spring CLI Example

You can use the following web application to test your installation. To start, create a file called app.groovy, as follows:

@RestController

**class** ThisWillActuallyRun {

@RequestMapping("/")

String home() {

"Hello World!"

}

}

Then run it from a shell, as follows:

$ spring run app.groovy

|  |
| --- |
| [Note] |
| The first run of your application is slow, as dependencies are downloaded. Subsequent runs are much quicker. |

Open [localhost:8080](http://localhost:8080/) in your favorite web browser. You should see the following output:

Hello World!

## 10.3 Upgrading from an Earlier Version of Spring Boot

If you are upgrading from an earlier release of Spring Boot, check the [“migration guide” on the project wiki](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide) that provides detailed upgrade instructions. Check also the[“release notes”](https://github.com/spring-projects/spring-boot/wiki) for a list of “new and noteworthy” features for each release.

When upgrading to a new feature release, some properties may have been renamed or removed. Spring Boot provides a way to analyze your application’s environment and print diagnostics at startup, but also temporarily migrate properties at runtime for you. To enable that feature, add the following dependency to your project:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-properties-migrator</artifactId>

<scope>runtime</scope>

</dependency>

|  |
| --- |
| [Warning] |
| Properties that are added late to the environment, such as when using @PropertySource, will not be taken into account. |
| [Note] |
| Once you’re done with the migration, please make sure to remove this module from your project’s dependencies. | |

To upgrade an existing CLI installation, use the appropriate package manager command (for example, brew upgrade) or, if you manually installed the CLI, follow the[standard instructions](https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html#getting-started-manual-cli-installation), remembering to update your PATH environment variable to remove any older references.

## 67. Using the CLI

Once you have installed the CLI, you can run it by typing spring and pressing Enter at the command line. If you run spring without any arguments, a simple help screen is displayed, as follows:

$ spring

usage: spring [--help] [--version]

<command> [<args>]

Available commands are:

run [options] <files> [--] [args]

Run a spring groovy script

... more command help is shown here

You can type spring help to get more details about any of the supported commands, as shown in the following example:

$ spring help run

spring run - Run a spring groovy script

usage: spring run [options] <files> [--] [args]

Option Description

------ -----------

--autoconfigure [Boolean] Add autoconfigure compiler

transformations (default: true)

--classpath, -cp Additional classpath entries

-e, --edit Open the file with the default system

editor

--no-guess-dependencies Do not attempt to guess dependencies

--no-guess-imports Do not attempt to guess imports

-q, --quiet Quiet logging

-v, --verbose Verbose logging of dependency

resolution

--watch Watch the specified file for changes

The version command provides a quick way to check which version of Spring Boot you are using, as follows:

$ spring version

Spring CLI v2.1.4.RELEASE

## 67.1 Running Applications with the CLI

You can compile and run Groovy source code by using the run command. The Spring Boot CLI is completely self-contained, so you do not need any external Groovy installation.

The following example shows a “hello world” web application written in Groovy:

**hello.groovy.**

@RestController

**class** WebApplication {

@RequestMapping("/")

String home() {

"Hello World!"

}

}

To compile and run the application, type the following command:

$ spring run hello.groovy

To pass command-line arguments to the application, use -- to separate the commands from the “spring” command arguments, as shown in the following example:

$ spring run hello.groovy -- --server.port=9000

To set JVM command line arguments, you can use the JAVA\_OPTS environment variable, as shown in the following example:

$ JAVA\_OPTS=-Xmx1024m spring run hello.groovy

|  |
| --- |
| [Note] |
| When setting JAVA\_OPTS on Microsoft Windows, make sure to quote the entire instruction, such as set "JAVA\_OPTS=-Xms256m -Xmx2048m". Doing so ensures the values are properly passed to the process. |

### 67.1.1 Deduced “grab” Dependencies

Standard Groovy includes a @Grab annotation, which lets you declare dependencies on third-party libraries. This useful technique lets Groovy download jars in the same way as Maven or Gradle would but without requiring you to use a build tool.

Spring Boot extends this technique further and tries to deduce which libraries to “grab” based on your code. For example, since the WebApplication code shown previously uses @RestController annotations, Spring Boot grabs "Tomcat" and "Spring MVC".

The following items are used as “grab hints”:

| **Items** | | **Grabs** |
| --- | --- | --- |
| JdbcTemplate, NamedParameterJdbcTemplate, DataSource | | JDBC Application. |
| @EnableJms | | JMS Application. |
| @EnableCaching | | Caching abstraction. |
| @Test | | JUnit. |
| @EnableRabbit | | RabbitMQ. |
| extends Specification | | Spock test. |
| @EnableBatchProcessing | | Spring Batch. |
| @MessageEndpoint @EnableIntegration | | Spring Integration. |
| @Controller @RestController @EnableWebMvc | | Spring MVC + Embedded Tomcat. |
| @EnableWebSecurity | | Spring Security. |
| @EnableTransactionManagement | | Spring Transaction Management. |
| [Tip] | |
| See subclasses of [CompilerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.1.4.RELEASE/spring-boot-project/spring-boot-cli/src/main/java/org/springframework/boot/cli/compiler/CompilerAutoConfiguration.java" \t "_top) in the Spring Boot CLI source code to understand exactly how customizations are applied. |

### 67.1.2 Deduced “grab” Coordinates

Spring Boot extends Groovy’s standard @Grab support by letting you specify a dependency without a group or version (for example, @Grab('freemarker')). Doing so consults Spring Boot’s default dependency metadata to deduce the artifact’s group and version.

|  |
| --- |
| [Note] |
| The default metadata is tied to the version of the CLI that you use. it changes only when you move to a new version of the CLI, putting you in control of when the versions of your dependencies may change. A table showing the dependencies and their versions that are included in the default metadata can be found in the [appendix](https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-dependency-versions.html). |

### 67.1.3 Default Import Statements

To help reduce the size of your Groovy code, several import statements are automatically included. Notice how the preceding example refers to @Component, @RestController, and @RequestMapping without needing to use fully-qualified names or import statements.

|  |
| --- |
| [Tip] |
| Many Spring annotations work without using import statements. Try running your application to see what fails before adding imports. |

### 67.1.4 Automatic Main Method

Unlike the equivalent Java application, you do not need to include a public static void main(String[] args) method with your Groovy scripts. ASpringApplication is automatically created, with your compiled code acting as the source.

### 67.1.5 Custom Dependency Management

By default, the CLI uses the dependency management declared in spring-boot-dependencies when resolving @Grab dependencies. Additional dependency management, which overrides the default dependency management, can be configured by using the @DependencyManagementBom annotation. The annotation’s value should specify the coordinates (groupId:artifactId:version) of one or more Maven BOMs.

For example, consider the following declaration:

@DependencyManagementBom("com.example.custom-bom:1.0.0")

The preceding declaration picks up custom-bom-1.0.0.pom in a Maven repository under com/example/custom-versions/1.0.0/.

When you specify multiple BOMs, they are applied in the order in which you declare them, as shown in the following example:

@DependencyManagementBom(["com.example.custom-bom:1.0.0",

"com.example.another-bom:1.0.0"])

The preceding example indicates that the dependency management in another-bom overrides the dependency management in custom-bom.

You can use @DependencyManagementBom anywhere that you can use @Grab. However, to ensure consistent ordering of the dependency management, you can use@DependencyManagementBom at most once in your application. A useful source of dependency management (which is a superset of Spring Boot’s dependency management) is the [Spring IO Platform](https://platform.spring.io/), which you might include with the following line:

@DependencyManagementBom('io.spring.platform:platform-bom:1.1.2.RELEASE')

## 67.2 Applications with Multiple Source Files

You can use “shell globbing” with all commands that accept file input. Doing so lets you use multiple files from a single directory, as shown in the following example:

$ spring run \*.groovy

## 67.3 Packaging Your Application

You can use the jar command to package your application into a self-contained executable jar file, as shown in the following example:

$ spring jar my-app.jar \*.groovy

The resulting jar contains the classes produced by compiling the application and all of the application’s dependencies so that it can then be run by using java -jar. The jar file also contains entries from the application’s classpath. You can add and remove explicit paths to the jar by using --include and --exclude. Both are comma-separated, and both accept prefixes, in the form of “+” and “-”, to signify that they should be removed from the defaults. The default includes are as follows:

public/\*\*, resources/\*\*, static/\*\*, templates/\*\*, META-INF/\*\*, \*

The default excludes are as follows:

.\*, repository/\*\*, build/\*\*, target/\*\*, \*\*/\*.jar, \*\*/\*.groovy

Type spring help jar on the command line for more information.

## 67.4 Initialize a New Project

The init command lets you create a new project by using [start.spring.io](https://start.spring.io/) without leaving the shell, as shown in the following example:

$ spring init --dependencies=web,data-jpa my-project

Using service at https://start.spring.io

Project extracted to '/Users/developer/example/my-project'

The preceding example creates a my-project directory with a Maven-based project that uses spring-boot-starter-web and spring-boot-starter-data-jpa. You can list the capabilities of the service by using the --list flag, as shown in the following example:

$ spring init --list

=======================================

Capabilities of https://start.spring.io

=======================================

Available dependencies:

-----------------------

actuator - Actuator: Production ready features to help you monitor and manage your application

...

web - Web: Support for full-stack web development, including Tomcat and spring-webmvc

websocket - Websocket: Support for WebSocket development

ws - WS: Support for Spring Web Services

Available project types:

------------------------

gradle-build - Gradle Config [format:build, build:gradle]

gradle-project - Gradle Project [format:project, build:gradle]

maven-build - Maven POM [format:build, build:maven]

maven-project - Maven Project [format:project, build:maven] (default)

...

The init command supports many options. See the help output for more details. For instance, the following command creates a Gradle project that uses Java 8 and war packaging:

$ spring init --build=gradle --java-version=1.8 --dependencies=websocket --packaging=war sample-app.zip

Using service at https://start.spring.io

Content saved to 'sample-app.zip'

## 67.5 Using the Embedded Shell

Spring Boot includes command-line completion scripts for the BASH and zsh shells. If you do not use either of these shells (perhaps you are a Windows user), you can use the shell command to launch an integrated shell, as shown in the following example:

$ spring shell

**Spring Boot** (v2.1.4.RELEASE)

Hit TAB to complete. Type \'help' and hit RETURN for help, and \'exit' to quit.

From inside the embedded shell, you can run other commands directly:

$ version

Spring CLI v2.1.4.RELEASE

The embedded shell supports ANSI color output as well as tab completion. If you need to run a native command, you can use the ! prefix. To exit the embedded shell, press ctrl-c.

## 67.6 Adding Extensions to the CLI

You can add extensions to the CLI by using the install command. The command takes one or more sets of artifact coordinates in the format group:artifact:version, as shown in the following example:

$ spring install com.example:spring-boot-cli-extension:1.0.0.RELEASE

In addition to installing the artifacts identified by the coordinates you supply, all of the artifacts' dependencies are also installed.

To uninstall a dependency, use the uninstall command. As with the install command, it takes one or more sets of artifact coordinates in the format of group:artifact:version, as shown in the following example:

$ spring uninstall com.example:spring-boot-cli-extension:1.0.0.RELEASE

It uninstalls the artifacts identified by the coordinates you supply and their dependencies.

To uninstall all additional dependencies, you can use the --all option, as shown in the following example:

$ spring uninstall --all

## 68. Developing Applications with the Groovy Beans DSL

Spring Framework 4.0 has native support for a beans{} “DSL” (borrowed from [Grails](https://grails.org/)), and you can embed bean definitions in your Groovy application scripts by using the same format. This is sometimes a good way to include external features like middleware declarations, as shown in the following example:

@Configuration

**class** Application **implements** CommandLineRunner {

@Autowired

SharedService service

@Override

**void** run(String... args) {

println service.message

}

}

**import** my.company.SharedService

beans {

service(SharedService) {

message = "Hello World"

}

}

You can mix class declarations with beans{} in the same file as long as they stay at the top level, or, if you prefer, you can put the beans DSL in a separate file

**69. Configuring the CLI with settings.xml**

The Spring Boot CLI uses Aether, Maven’s dependency resolution engine, to resolve dependencies. The CLI makes use of the Maven configuration found in ~/.m2/settings.xml to configure Aether. The following configuration settings are honored by the CLI:

* Offline
* Mirrors
* Servers
* Proxies
* Profiles
  + Activation
  + Repositories
* Active profiles

See [Maven’s settings documentation](https://maven.apache.org/settings.html) for further information.