**SPRING BOOT Quick Look By Ashok**

**10.2 Installing the Spring Boot CLI**

The Spring Boot CLI is a command line tool that can be used if you want to quickly prototype with Spring. It allows you to run [Groovy](http://groovy.codehaus.org/) scripts, which means that you have a familiar Java-like syntax, without so much boilerplate code.

You don’t need to use the CLI to work with Spring Boot but it’s definitely the quickest way to get a Spring application off the ground.

**Servlet containers**

|  |  |
| --- | --- |
| **Name** | **Servlet Version** |
| Tomcat 8.5 | 3.1 |
| Jetty 9.4 | 3.1 |
| Undertow 1.3 | 3.1 |

The following embedded servlet containers are supported out of the box:

You can also deploy Spring Boot applications to any Servlet 3.0+ compatible container.

**11.3.2 The @EnableAutoConfiguration annotation**

The second class-level annotation is @EnableAutoConfiguration. This annotation tells Spring Boot to “guess” how you will want to configure Spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration will assume that you are developing a web application and setup Spring accordingly.

**13.2.3 Using the Spring Boot Maven plugin**

Spring Boot includes a [Maven plugin](https://docs.spring.io/spring-boot/docs/2.0.0.M4/reference/htmlsingle/#build-tool-plugins-maven-plugin) that can package the project as an executable jar. Add the plugin to your <plugins> section if you want to use it:

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

/build>

**13.5 Starters**

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors.

**2. The Web Starter**

First, let’s look at developing the REST service; we can use libraries like Spring MVC, Tomcat and Jackson – a lot of dependencies for a single application.

Spring Boot starters can help to reduce the number of manually added dependencies just by adding one dependency. So instead of manually specifying the dependencies just add one starter as in the following example:

|  |  |
| --- | --- |
| 1  2  3  4 | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency> |

spring-boot-starter-data-jpa - Starter for using Spring Data JPA with Hibernate.

spring-boot-starter-jersey Starter for building RESTful web applications using JAX-RS and Jersey. An alternative to [spring-boot-starter-web](https://docs.spring.io/spring-boot/docs/2.0.0.M4/reference/htmlsingle/#spring-boot-starter-web)

spring-boot-starter-actuator- Starter for using Spring Boot’s Actuator which provides production ready features to help you monitor and manage your application

spring-boot-starter-tomcat -Starter for using Tomcat as the embedded servlet container. Default servlet container starter used by [spring-boot-starter-web](https://docs.spring.io/spring-boot/docs/2.0.0.M4/reference/htmlsingle/#spring-boot-starter-web)

spring-boot-starter-jetty - Starter for using Jetty as the embedded servlet container. An alternative to [spring-boot-starter-tomcat](https://docs.spring.io/spring-boot/docs/2.0.0.M4/reference/htmlsingle/#spring-boot-starter-tomcat)

**Locating the main application class**

We generally recommend that you locate your main application class in a root package above other classes. The @EnableAutoConfiguration annotation is often placed on your main class, and it implicitly defines a base “search package” for certain items. For example, if you are writing a JPA application, the package of the@EnableAutoConfiguration annotated class will be used to search for @Entity items.

Using a root package also allows the @ComponentScan annotation to be used without needing to specify a basePackage attribute.

**15. Configuration classes**

Spring Boot favors Java-based configuration. Although it is possible to use SpringApplication with an XML sources, we generally recommend that your primary source is a single @Configuration class. Usually the class that defines the main method is also a good candidate as the primary @Configuration.

**15.1 Importing additional configuration classes**

You don’t need to put all your @Configuration into a single class. The @Import annotation can be used to import additional configuration classes. Alternatively, you can use @ComponentScan to automatically pick up all Spring components, including @Configuration classes.

**15.2 Importing XML configuration**

If you absolutely must use XML based configuration, we recommend that you still start with a @Configuration class. You can then use an additional @ImportResource annotation to load XML configuration files.

**16.2 Disabling specific auto-configuration**

If you find that specific auto-configure classes are being applied that you don’t want, you can use the exclude attribute of @EnableAutoConfiguration to disable them.

**import** org.springframework.boot.autoconfigure.\*;

**import** org.springframework.boot.autoconfigure.jdbc.\*;

**import** org.springframework.context.annotation.\*;

*@Configuration*

*@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})*

**public** **class** MyConfiguration {

}

If the class is not on the classpath, you can use the excludeName attribute of the annotation and specify the fully qualified name instead. Finally, you can also control the list of auto-configuration classes to exclude via the spring.autoconfigure.exclude property.

**18. Using the @SpringBootApplication annotation**

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes:

**19.2 Running as a packaged application**

If you use the Spring Boot Maven or Gradle plugins to create an executable jar you can run your application using java -jar. For example:

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

It is also possible to run a packaged application with remote debugging support enabled. This allows you to attach a debugger to your packaged application:

$java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myproject-0.0.1-SNAPSHOT.jar

**19.3 Using the Maven plugin**

The Spring Boot Maven plugin includes a run goal which can be used to quickly compile and run your application. Applications run in an exploded form just like in your IDE.

$ mvn spring-boot:run

**20.2 Automatic restart**

Applications that use spring-boot-devtools will automatically restart whenever files on the classpath change. This can be a useful feature when working in an IDE as it gives a very fast feedback loop for code changes. By default, any entry on the classpath that points to a folder will be monitored for changes. Note that certain resources such as static assets and view templates [do not need to restart the application](https://docs.spring.io/spring-boot/docs/2.0.0.M4/reference/htmlsingle/#using-boot-devtools-restart-exclude).