# 

[General Questions 3](#_Toc61359540)

[1) What are the differences between Spring and Spring Boot? 3](#_Toc61359541)

[2) Advantages of Spring boot? 4](#_Toc61359542)

[3) What are spring boot components? 4](#_Toc61359543)

[5](#_Toc61359544)

[4) Can we use spring boot with non spring app? 5](#_Toc61359545)

[5) Difference between spring boot 1 and 2.x 5](#_Toc61359546)

[6) Reload changes without starting server? 6](#_Toc61359547)

[7) Embedded container in spring boot? 6](#_Toc61359548)

[8) Different ways to create spring boot application? 6](#_Toc61359549)

[9) How can we set up a Spring Boot application with Maven? 6](#_Toc61359550)

[10) How to deploy Spring Boot web applications as JAR and WAR files? 8](#_Toc61359551)

[11) How to use Spring Boot for command line applications? 9](#_Toc61359552)

[12) What are possible sources of external configuration? 10](#_Toc61359553)

[13) What does it mean that Spring Boot supports relaxed binding? 10](#_Toc61359554)

[14) Application.properties 11](#_Toc61359555)

[15) What logging support provided by Spring Boot? How can we control logging level in Spring Boot? 11](#_Toc61359556)

[16) Explain @SpringBootAnnotation? 12](#_Toc61359557)

[17) Location of Application.java file? 13](#_Toc61359558)

[18) How to configure Database in spring boot? 13](#_Toc61359559)

[19) Usage of CommandLineRunner and ApplicationRunner? 15](#_Toc61359560)

[20) Difference between CommandLineRunner and ApplicationRunner? 16](#_Toc61359561)

[21) Explain usage of spring-boot-maven-plugin? 17](#_Toc61359562)

[22) Can you control logging with Spring Boot? How? 17](#_Toc61359563)

[23) How to disable default web server? 17](#_Toc61359564)

[24) What if we don’t include the spring boot maven plugin? 18](#_Toc61359565)

[25) What is the preferable way of defining properties – prop file or yaml? 19](#_Toc61359566)

[26) Which Embedded Servers does Spring Boot Support, and How to Change the Default? 20](#_Toc61359567)

[27) How to deploy your spring boot war to wildfly or other servers? 20](#_Toc61359568)

[28) Why Do We Need Spring Profiles? 23](#_Toc61359569)

[29) Minimum requirement for spring boot? 24](#_Toc61359570)

[30) Explain what is thymeleaf and how to use thymeleaf? 24](#_Toc61359571)

[31) Steps to use JDBC template? 25](#_Toc61359572)

[32) Best way to configure properties? 27](#_Toc61359573)

[33) How to create non-web application in spring boot? 27](#_Toc61359574)

[34) Default H2 database name? 27](#_Toc61359575)

[35) How do you Add, Filter to an application? \*\*\* 27](#_Toc61359576)

[36) What is LiveReload in Spring Boot? 28](#_Toc61359577)

[37) How to set logging level to debug? 28](#_Toc61359578)

[38) Explain the difference between an embedded container and a WAR. 28](#_Toc61359579)

[39) Why spring boot is production ready? 28](#_Toc61359580)

[Integration test 29](#_Toc61359581)

[40) How to write integration tests? 29](#_Toc61359582)

[41) What is the difference between springRunner and SpringJunitRunner? 29](#_Toc61359583)

[dev tools 30](#_Toc61359584)

[42) What is Spring Boot DevTools used for? 30](#_Toc61359585)

[spring actuator 31](#_Toc61359586)

[43) What is Spring Boot Actuator used for? 31](#_Toc61359587)

[44) What is a shutdown in the actuator? 32](#_Toc61359588)

[45) How to disable Actuator endpoint security in Spring Boot? 32](#_Toc61359589)

[46) How can we create a custom endpoint in Spring Boot Actuator? 33](#_Toc61359590)

[spring initializer 33](#_Toc61359591)

[47) What is spring boot initializer? 33](#_Toc61359592)

[spring auto configuration 34](#_Toc61359593)

[48) What is auto configuration? 34](#_Toc61359594)

[49) How to disable a specific auto-configuration? 34](#_Toc61359595)

[50) How to register a custom auto-configuration? 35](#_Toc61359596)

[51) How to tell an auto-configuration to back away when a bean exists? 35](#_Toc61359597)

[spring starter 36](#_Toc61359598)

[52) What is spring boot starter? 36](#_Toc61359599)

[53) What is spring.provides? 36](#_Toc61359600)

[54) What Spring Boot starters are available out there? 37](#_Toc61359601)

[55) What does stater-parent do? 37](#_Toc61359602)

[56) How to override default dependency version? 40](#_Toc61359603)

[Spring boot CLI 41](#_Toc61359604)

[57) What is Spring boot CLI 41](#_Toc61359605)

[58) Spring boot cli advantaegs: 41](#_Toc61359606)

[59) Spring Boot admin 42](#_Toc61359607)

[Reference 42](#_Toc61359608)

# General Questions

## What are the differences between Spring and Spring Boot?

### The Spring Framework provides multiple features that make the development of web applications easier. These features include dependency injection, data binding, aspect-oriented programming, data access, and many more.

### Over the years, Spring has been growing more and more complex, and the amount of configuration such application requires can be intimidating. This is where Spring Boot comes in handy – it makes configuring a Spring application a breeze.

### Essentially, **while Spring is unopinionated, Spring Boot takes an opinionated view of the platform and libraries**, letting us get started quickly.

### Here are two of the most important benefits Spring Boot brings in:

### Auto-configure applications based on the artifacts it finds on the classpath

### Provide non-functional features common to applications in production, such as security or health checks

## Advantages of Spring boot?

### It simplifies Spring dependencies by taking the opinionated view.

### Spring Boot provides a pre-configured set of technologies/framework to reduces error-prone configuration so we as a developer focused on building our business logic and not thinking of project setup.

### It reduces development code by avoiding a lot of boilerplate code.

### Easier to integrate Spring Boot Application with Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security, etc.

### You really don’t need those big XML configurations for your project.

### **Embed Tomcat, Jetty or Undertow directly.**

### Provide opinionated Maven POM to simplify your configuration.

## What are spring boot components?

### [Boot Initializer](https://www.javadevjournal.com/spring-boot/spring-initializr/)

### [Spring Boot Starter](https://www.javadevjournal.com/spring/spring-boot-starters/)

### [Auto Configurator](https://www.javadevjournal.com/spring/spring-boot-auto-configuration/)

### [Spring Boot CLI](https://www.javadevjournal.com/spring-boot/commandlinerunner-interface-in-spring-boot/)

### [Actuator](https://www.javadevjournal.com/spring-boot/spring-boot-actuator/)

## Spring Boot Modules

## Can we use spring boot with non spring app?

NO.

## Difference between spring boot 1 and 2.x

## Reload changes without starting server?

### This is achievable by Spring Boot Dev Tools module.it’s a powerful tool for development. It helps developers to shorten the development cycle and enable easy deployment and testing during development.

### To enable this feature, add the following dependency to Maven POM file.

### <dependencies>

### <dependency>

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

### <artifactId>spring-boot-devtools</artifactId>

### </dependency>

### </dependencies>

### Copy

Read [Spring Boot Dev Tools](https://www.javadevjournal.com/spring/spring-boot-devtools/) for different features of Dev Tools

## Embedded container in spring boot?

### Spring Boot includes support for the following embedded containers

### Tomcat

### Jetty

### Undertow.

## Different ways to create spring boot application?

### Spring Initializer

### Boot CLI

### Using Maven

### IDE project wizard

## How can we set up a Spring Boot application with Maven?

### We can include Spring Boot in a Maven project just like we would any other library. However, the best way is to inherit from the spring-boot-starter-parent project and declare dependencies to Spring Boot starters. Doing this lets our project reuse the default settings of Spring Boot.

### Inheriting the spring-boot-starter-parent project is straightforward – we only need to specify a parent element in pom.xml:

<parent>

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

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

<version>2.1.1.RELEASE</version>

</parent>

We can find the latest version of spring-boot-starter-parent on Maven Central.

Using the starter parent project is convenient, but not always feasible. For instance, if our company requires all projects to inherit from a standard POM, we cannot rely on the Spring Boot starter parent.

In this case, we can still get the benefits of dependency management with this POM element:

<dependencyManagement>

<dependencies>

<dependency>

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

**<artifactId>spring-boot-dependencies</artifactId>**

**<version>2.1.1.RELEASE</version>**

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Finally, we can add some dependencies to Spring Boot starters, and then we’re good to go.

## How to deploy Spring Boot web applications as JAR and WAR files?

### Traditionally, we package a web application as a WAR file, then deploy it into an external server. Doing this allows us to arrange multiple applications on the same server. During the time that CPU and memory were scarce, this was a great way to save resources.

### However, things have changed. Computer hardware is fairly cheap now, and the attention has turned to server configuration. A small mistake in configuring the server during deployment may lead to catastrophic consequences.

### **Spring tackles this problem by providing a plugin, namely spring-boot-maven-plugin, to package a web application as an executable JAR**. To include this plugin, just add a plugin element to pom.xml:

**<plugin>**

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

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

**</plugin>**

With this plugin in place, we’ll get a fat JAR after executing the package phase. This JAR contains all the necessary dependencies, including an embedded server. Thus, we no longer need to worry about configuring an external server.

We can then run the application just like we would an ordinary executable JAR.

Notice that the packaging element in the pom.xml file must be set to jar to build a JAR file:

1

<packaging>jar</packaging>

If we don’t include this element, it also defaults to jar.

In case we want to build a WAR file, change the packaging element to war:

1

**<packaging>war</packaging>**

And leave the container dependency off the packaged file:

**<dependency>**

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

**<artifactId>spring-boot-starter-tomcat</artifactId>**

**<scope>provided</scope>**

**</dependency>**

After executing the Maven package phase, we’ll have a deployable WAR file.

## How to use Spring Boot for command line applications?

### Just like any other Java program, a Spring Boot command line application must have a main method. This method serves as an entry point, which invokes the SpringApplication#run method to bootstrap the application:

### @SpringBootApplication

### public class MyApplication {

### public static void main(String[] args) {

### SpringApplication.run(MyApplication.class);

### // other statements

### }

### }

### The SpringApplication class then fires up a Spring container and auto-configures beans.

### Notice we must pass a configuration class to the run method to work as the primary configuration source. By convention, this argument is the entry class itself.

### After calling the run method, we can execute other statements as in a regular program.

## What are possible sources of external configuration?

### Spring Boot provides support for external configuration, allowing us to run the same application in various environments. We can **use properties files, YAML files, environment variables, system properties, and command-line option arguments** to specify configuration properties.

### We can then gain access to those properties using the @Value annotation, a bound object via the @ConfigurationProperties annotation, or the Environment abstraction.

### Here are the most common sources of external configuration:

### **Command-line properties:** Command-line option arguments are program arguments starting with a double hyphen, such as –server.port=8080. Spring Boot converts all the arguments to properties and adds them to the set of environment properties.

### **Application properties:** Application properties are those loaded from the application.properties file or its YAML counterpart. By default, Spring Boot searches for this file in the current directory, classpath root, or their config subdirectory.

### **Profile-specific properties:** Profile-specific properties are loaded from the application-{profile}.properties file or its YAML counterpart. The {profile} placeholder refers to an active profile. These files are in the same locations as, and take precedence over, non-specific property files.

## What does it mean that Spring Boot supports relaxed binding?

### Relaxed binding in Spring Boot is applicable to the type-safe binding of configuration properties.

### With relaxed binding, the key of an environment property doesn’t need to be an exact match of a property name. Such an environment property can be written in camelCase, kebab-case, snake\_case, or in uppercase with words separated by underscores.

### For example, if a property in a bean class with the @ConfigurationProperties annotation is named myProp, it can be bound to any of these environment properties: myProp, my-prop, my\_prop, or MY\_PROP.

## Application.properties

### Spring Boot provides various properties which can be specified inside our project's **application.properties** file. These properties have default values and you can set that inside the properties file. Properties are used to set values like: server-port number, database connection configuration etc. The file has more than 300 predefined properties to tweak the configuration.

## What logging support provided by Spring Boot? How can we control logging level in Spring Boot?

Spring Boot provides options to use all popular logging API using the relevant starter, by default Spring Boot use Commons Logging for its internal logging. If we are using [Spring Boot Starters](https://www.javadevjournal.com/spring/spring-boot-starters/) for our application, Logback will be used for logging by default unless we want to use any other logging API. To use any other logging API, we need to add the correct starter in our application. In case we like to use Log4j2 for logging configuration, all you have to add the log4j2 starter in your application (You may have to exclude logback using  pom.xml file).

Spring Boot provides an easy way to configure and set logging levels for your application. We can use application.properties file to configure the desired Logging level for our application by using ‘logging.level.\*=LEVEL’. Here is an example for the same.

logging.level.com.javadevjournal.rest=WARN

Copy

Read [Spring Boot Logging](https://www.javadevjournal.com/spring/spring-boot-logging/) for more detail.

## Explain @SpringBootAnnotation?

### This is meta Annotation, means it is annotation of annotation. It has 3 annotations-

### Many Spring Boot developers like their apps to use auto-configuration, component scan and be able to define extra configuration on their "application class". A single @SpringBootApplication annotation can be used to enable those three features, that is:

### @**EnableAutoConfiguration**: enable [Spring Boot’s auto-configuration mechanism](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-auto-configuration.html) This is also meta annotation. This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for you.

### For example, if you are writing a Spring MVC application and you have Thymeleaf JAR files on the application classpath, then Spring Boot auto-configuration can automatically configure the Thymeleaf template resolver, view resolver, and other settings automatically.

### So, you can say that @SpringBootApplication is a 3-in-1 annotation that combines the functionality of @Configuration, @ComponentScan, and @EnableAutoConfiguration.

### It also marks the class as a BootStrap class, which means you can runt it as a normal Java class, e.g. by running its JAR file from the command prompt as shown here, or just right-clicking and running it as a Java program in Eclipse IDE.

### @**ComponentScan**: enable @Component scan on the package where the application is located (see [the best practices](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-structuring-your-code.html)). This annotation enables component-scanning so that the web controller classes and other components you create will be automatically discovered and registered as beans in Spring's Application Context. All the@Controller classes you write are discovered by this annotation.

### @**Configuration**: allow to register extra beans in the context or import additional configuration classes This annotation marks a class as a Configuration class for Java-based configuration. This is particularly important if you favor Java-based configuration over XML configuration.

### The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes, as shown in the following example:

### **package** com.example.myapplication;

### **import** org.springframework.boot.SpringApplication;

### **import** org.springframework.boot.autoconfigure.SpringBootApplication;

### @SpringBootApplication *// same as @Configuration @EnableAutoConfiguration @ComponentScan*

### **public** **class** Application {

### **public** **static** **void** main(String[] args) {

### SpringApplication.run(Application.**class**, args);

### }

### }

## Location of Application.java file?

### We generally recommend that you locate your main application class in a root package above other classes. The [@SpringBootApplication annotation](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html) 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 @SpringBootApplication annotated class is used to search for @Entity items. Using a root package also allows component scan to apply only on your project.

## How to configure Database in spring boot?

### Add the required db dependency to pom.

1. Define properties in application.properties. Spring boot will ick up and set up every thing for you.

# create and drop tables and sequences, loads import.sql

spring.jpa.hibernate.ddl-auto=create-drop

# Oracle settings

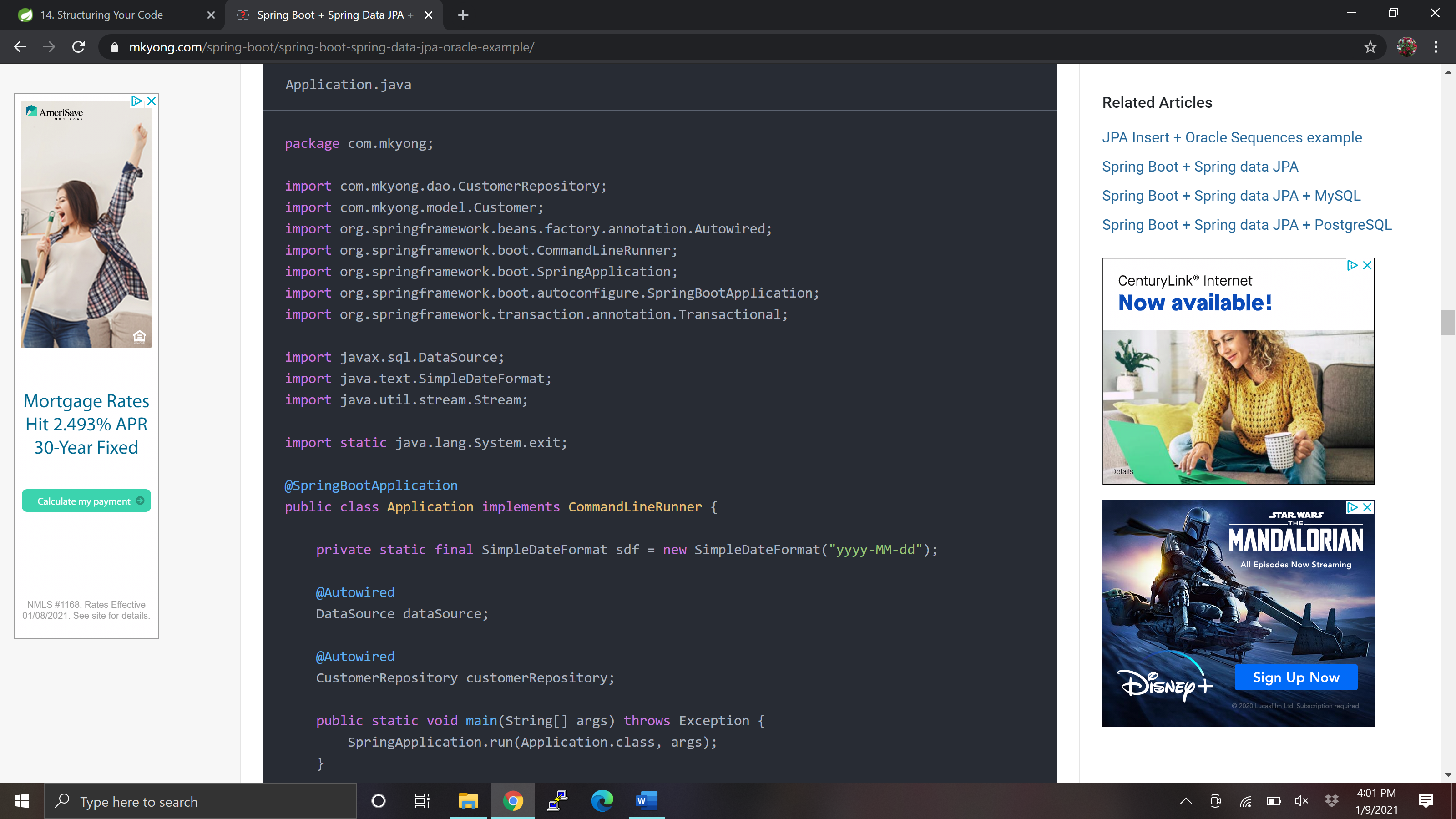
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

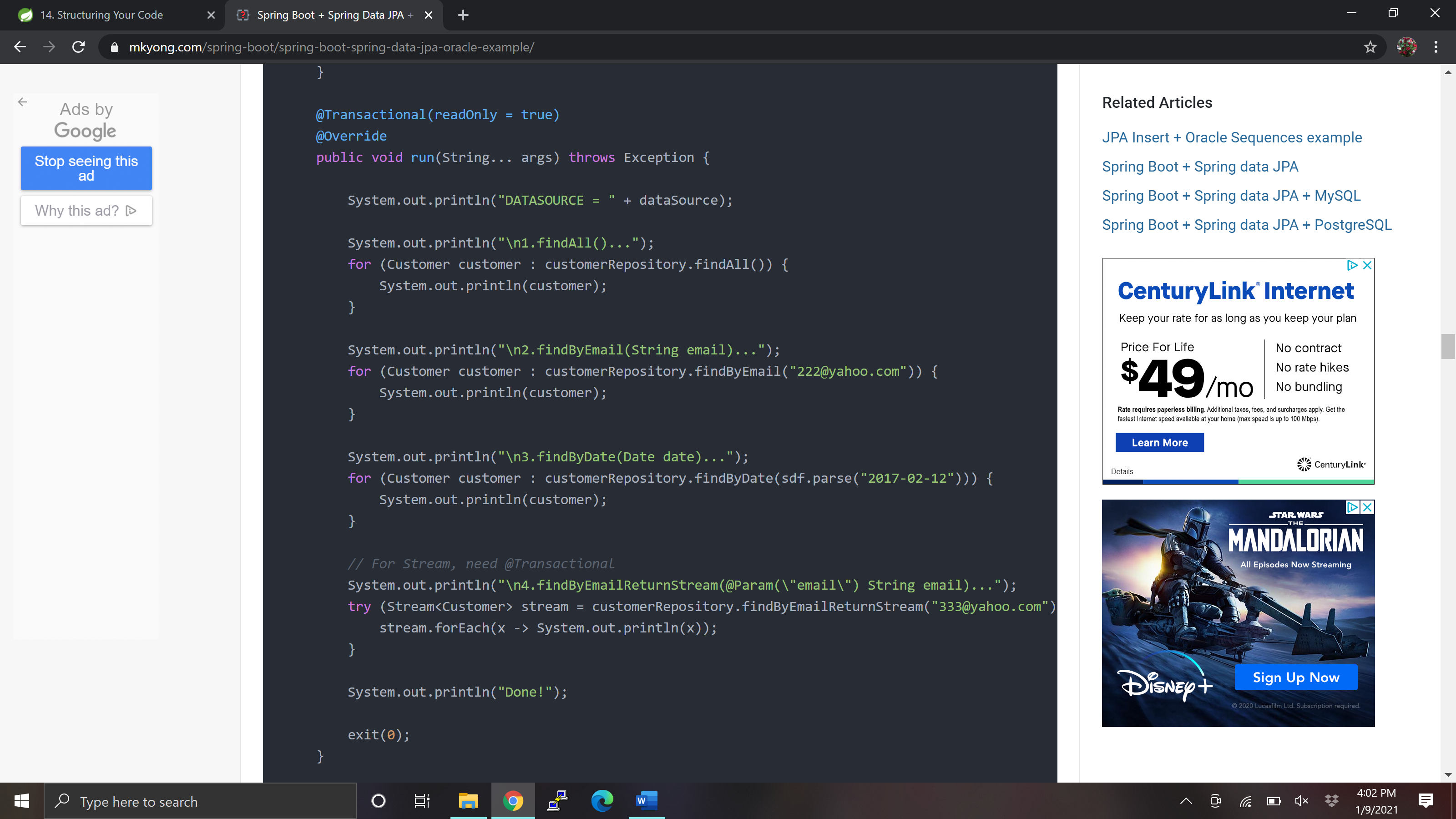
spring.datasource.username=system

spring.datasource.password=password

spring.datasource.driver-class-oracle.jdbc.driver.OracleDrive

1. Optional:  If import.sql is found in the classpath, Hibernate will load it automatically.
2. Use @Datasource or @JdbcTEmplalte OR @Repository.





## Usage of CommandLineRunner and ApplicationRunner?

### Application Runner and Command Line Runner interfaces lets you to execute the code after the Spring Boot application is started. You can use these interfaces to perform any actions immediately after the application has started. This chapter talks about them in detail.

### Spring boot’s [CommandLineRunner](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/CommandLineRunner.html) interface is used to run a code block only once in application’s lifetime – after application is initialized.

These can be used in 3 ways –

1. As component
2. In springBootApplication class
3. As seperate Bean

@Component

public class ApplicationStartupRunner implements CommandLineRunner {

    protected final Log logger = LogFactory.getLog(getClass());

    @Override

    public void run(String... args) throws Exception {

        logger.info("ApplicationStartupRunner run method Started !!");

    }

}

It is important to note that if any exceptions are thrown inside the run(String… args) method, this will cause the context to close and an application to shut down. So put risky code in try-catch block – ALWAYS.

@Order in Runner-

You may have multiple implementations of CommandLineRunner interface. By default, spring boot to scan all its run() methods and execute it. But if you want to force some ordering in them, use [@Order](https://docs.spring.io/spring-framework/docs/4.3.7.RELEASE/javadoc-api/org/springframework/core/annotation/Order.html?is-external=true) annotation.

**Usage**:

### Command line runners are a useful functionality to execute the various types of code that only have to be run once, right after application startup.

### FYI, Spring Batch relies on these runners in order to trigger the execution of the jobs.

### We can use the dependency injection to our advantage in order to wire in whatever dependencies that we need and in whatever way we want – in run() method implementation.

## Difference between CommandLineRunner and ApplicationRunner?

### These runners are used to run the logic on application startup, for example spring boot has [ApplicationRunner](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/ApplicationRunner.html)(Functional Interface) with run method

### ApplicationRunner run() will get execute, just after applicationcontext is created and before spring boot application startup.

### ApplicationRunner takes ApplicationArgument which has convenient methods like getOptionNames(), getOptionValues() and getSourceArgs().

### [CommandLineRunner](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/CommandLineRunner.html) is also a Functional interface with run method

### CommandLineRunner run() will get execute, just after applicationcontext is created and before spring boot application starts up.

### It accepts the argument, which are passed at time of server startup.

### Both of them provides the same functionality and the only difference between CommandLineRunner and ApplicationRunner is CommandLineRunner.run() accepts Stringarray[] whereas ApplicationRunner.run() accepts ApplicationArguments as argument.

## Explain usage of spring-boot-maven-plugin?

### The maven plugin will create an "executable" archive. In the case of the war packaging, you would be able to execute your app with java -jar my-app.war. If you intend to deploy your Spring Boot application in an existing Servlet container, then this plugin is, indeed, not necessary.

### The maven plugin does more things like running your app from the shell or creating build information.

### The Spring Boot Maven Plugin provides Spring Boot support in [Apache Maven](https://maven.org/). It allows you to **package executable jar or war archives**,

### **run Spring Boot applications**,

### **generate build information and start your Spring Boot application prior to running integration tests.**

## Can you control logging with Spring Boot? How?

Yes, we can control logging with Spring Boot by specifying log levels on application.properties file. Spring Boot loads this file when it exists in the [classpath](http://www.java67.com/2012/08/what-is-path-and-classpath-in-java-difference.html) and it can be used to configure both Spring Boot and application code.  
  
Spring Boot uses Commons Logging for all internal logging and you can change log levels by adding following lines in the application.properties file:  
  
logging.level.org.springframework=DEBUG  
logging.level.com.demo=INFO  
  
Read more: <https://www.java67.com/2018/06/top-15-spring-boot-interview-questions-answers-java-jee-programmers.html#ixzz6jD4NilON>

## How to disable default web server?

### **Spring Boot** automatically starts an application in web server mode if it finds the **web** starter module in the classpath. To disable web server configuration, set the **webApplicationType** to none in the **application.properties** file as in this example:

### spring.main.web-application-type=none

## What if we don’t include the spring boot maven plugin?

### If you don’t include plugin, the creaed jar/wara will not be executable. When we add plugin, there is 2 jar in targe folder. One is <name>.jar and second is <name>.jar.original.

Original one can not be executed but jar one can be executed. The folder structure will be different in both jars

Without plugin;

Manifest-Version: 1.0

Implementation-Title: SpringBasicSecurity

Implementation-Version: 0.0.1-SNAPSHOT

Build-Jdk-Spec: 1.8

Created-By: Maven Archiver 3.4.0

With plugin

Manifest-Version: 1.0

Implementation-Title: SpringBasicSecurity

Implementation-Version: 0.0.1-SNAPSHOT

Start-Class: com.apress.SpringBasicSecurityApplication

Spring-Boot-Classes: BOOT-INF/classes/

Spring-Boot-Lib: BOOT-INF/lib/

Build-Jdk-Spec: 1.8

Spring-Boot-Version: 2.1.7.RELEASE

Created-By: Maven Archiver 3.4.0

Main-Class: org.springframework.boot.loader.JarLauncher

## What is the preferable way of defining properties – prop file or yaml?

### YAML offers many advantages over properties files, such as:

### More clarity and better readability

### Perfect for hierarchical configuration data, which is also represented in a better, more readable format

### Support for maps, lists, and scalar types

### Can include several [profiles](https://www.baeldung.com/spring-profiles) in the same file

### However, writing it can be a little difficult and error-prone due to its indentation rules.

### For details and working samples, please refer to our [Spring YAML vs Properties](https://www.baeldung.com/spring-yaml-vs-properties) tutorial.

## Which Embedded Servers does Spring Boot Support, and How to Change the Default?

### As of date, **Spring MVC supports Tomcat, Jetty, and Undertow**. Tomcat is the default application server supported by Spring Boot's web starter.

### **Spring WebFlux supports Reactor Netty, Tomcat, Jetty, and Undertow** with Reactor Netty as default.

### In Spring MVC, to change the default, let's say to Jetty, we need to exclude Tomcat and include Jetty in the dependencies:

### <**dependency**>

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

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

### <**exclusions**>

### <**exclusion**>

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

### <**artifactId**>spring-boot-starter-tomcat</**artifactId**>

### </**exclusion**>

### </**exclusions**>

### </**dependency**>

### <**dependency**>

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

### <**artifactId**>spring-boot-starter-jetty</**artifactId**>

### </**dependency**>

### Similarly, to change the default in WebFlux to UnderTow, we need to exclude Reactor Netty and include UnderTow in the dependencies.

### [“Comparing embedded servlet contains in Spring Boot”](https://www.baeldung.com/spring-boot-servlet-containers) contains more details on the different embedded servers we can use with Spring MVC.

## How to deploy your spring boot war to wildfly or other servers?

### <https://thepracticaldeveloper.com/how-to-deploy-a-spring-boot-war-in-wildfly-jboss/>

### Go to the **Home tab** and click on the small launcher arrow in the lower right-hand corner of the font section to go to the Font dialog box. Select +Body and the size text you want, then click Set as Default in the lower left-hand corner.

### Word will ask whether you want to make this the default for this document only or for all future documents based on the Normal template. Choose the latter and click OK.

### Save everything you’ve done by clicking Set as Default on the Design tab (next to the Colors and Fonts buttons).

### First, go to your application’s main class and make it extend SpringBootServletInitializer. Note that, if you prefer to work with Spring Boot version 1.5 or earlier, this class was moved to a different package in Spring Boot 2:

### org.springframework.boot.web.support.SpringBootServletInitializer: if you use Spring Boot 1.5

### org.springframework.boot.web.servlet.support.SpringBootServletInitializer: if you use Spring Boot 2

### Override the configure method and point your sources to this same class (it’s the entry point of your application). Use the passed argument, SpringApplicationBuilder. If you don’t do that, your different components won’t be detected, won’t be injected in the context so your application won’t work.

1. @SpringBootApplication
2. public class JbossWildflyDemoApplication extends SpringBootServletInitializer {
3. @Override
4. protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
5. return application.sources(JbossWildflyDemoApplication.class);
6. }
7. public static void main(String[] args) {
8. SpringApplication.run(JbossWildflyDemoApplication.class, args);
9. }
10. }

### Now you should write some additional configuration in your Maven pom.xml file. Apart from changing the packaging method to generate a Spring Boot war package, you can exclude now the embedded tomcat server (you don’t need it) and specify a provided dependency with the javax.servlet API.

<project ...>

...

<packaging>war</packaging>

...

<build>

<plugins>

<!-- OPTIONAL: Remove this plugin to reduce the WAR file size.

It's not needed to deploy in an external application server. -->

<plugin>

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

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

</plugin>

</plugins>

</build>

<dependencies>

...

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<!-- We need to include the javax.servlet API specs, the implementation

will be provided by Wildfly / JBoss / Tomcat -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

...

</project>

### As you can see in the notes above, you can even remove the spring-boot-maven-plugin plugin , since you don’t need it when deploying on the external server. This will save you some extra megabytes in the WAR file.

### Note that you can also choose to specify the spring-boot-starter-tomcat as provided, instead of excluding it from the starter web artifact. If you do that, you don’t need the javax.servlet-api artifact to build your application. However, the alternative we follow within this post saves us some megabytes in the resulting WAR file.

### 

## Why Do We Need Spring Profiles?

### When developing applications for the enterprise, we typically deal with multiple environments such as Dev, QA, and Prod. The configuration properties for these environments are different.

### For example, we might be using an embedded H2 database for Dev, but Prod could have the proprietary Oracle or DB2. Even if the DBMS is the same across environments, the URLs would definitely be different.

### To make this easy and clean, **Spring has the provision of profiles, to help separate the configuration for each environment**. So that instead of maintaining this programmatically, the properties can be kept in separate files such as application-dev.properties and application-prod.properties. The default application.properties points to the currently active profile using spring.profiles.active so that the correct configuration is picked up.

<https://dzone.com/articles/spring-boot-profiles-1>

### [Spring Profiles](https://www.baeldung.com/spring-profiles) gives a comprehensive view of this topic.

## Minimum requirement for spring boot?

Spring Boot 2.1.7.RELEASE requires

* Java 8 +
* Spring Framework 5.1.9 +

**Explicit build support**

* Maven 3.3+
* Gradle 4.4+

**Servlet Container Support**

* Tomcat 9.0 – Servlet Version 4.0
* Jetty 9.4 –  Servlet Version 3.1
* Undertow 2.0 – Servlet Version 4.0

## Explain what is thymeleaf and how to use thymeleaf?

Thymeleaf is a server-side Java template engine used for web applications. It aims to bring natural template for your web application and can integrate well with Spring Framework and HTML5 Java web applications. To use Thymeleaf, you need to add the following code in the pom.xml file:

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

## Steps to use JDBC template?

Spring Boot starter projects provide the required libraries to connect the application with JDBC. So, for example, if you just have to create an application  and connect it with [MySQL](https://www.edureka.co/blog/mysql-tutorial/) database, you can follow the below steps:

**Step 1:** Create a database in MySQL

|  |  |
| --- | --- |
| 1 | CREATE DATABASE example; |

**Step 2:**Then you have to create a table inside this database.

|  |  |
| --- | --- |
| 1 | CREATE TABLE customers(customerid INT PRIMARY KEY NOT NULL AUTO\_INCREMENT, customername VARCHAR(255)); |

**Step 3:** Now, create a Spring Boot project and provide the required details

**Step 4:** Add the JDBC, MySQL and web dependencies.

**Step 5:** Once the project is created, you have to configure the database into application properties

|  |  |
| --- | --- |
| 1  2  3  4 | spring.datasource.url=jdbc:mysql://localhost:3306/example  spring.datasource.username=root  spring.datasource.password=edureka  spring.jpa.hibernate.ddl-auto=create-drop |

**Step 6:** The main application.java class should have the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | package com.edureka;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class SampleApplication {      public static void main(String[] args) {          SpringApplication.run(SampleApplication.class, args);      }  } |

**Step 7:** Next, you have to create a controller to handle the HTTP requests, by mentioning the following code:

package com.edureka;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class JdbcController {

@Autowired

JdbcTemplate jdbc;

@RequestMapping("/insert")

public String index(){

jdbc.execute("insert into customers(name)values('Aryya')");

return "Data Entry Successful";

}

}

**Step 8:** Finally, execute this project as a Java application.  
**Step 9:** Next, open the URL (localhost:8080/insert), and you will see the output as Data Entry Successful. You can also go forward and check if the data is entered into the table.

## Best way to configure properties?

### One way to expose the custom application configuration in Spring Boot is by using the @Value annotation. But, the only problem with this annotation is that all the configuration values will be distributed throughout the application. Instead, you can use a centralized approach.

### By centralized approach, I mean that you can define a configuration component using the @ConfigurationProperties as follows:

### @Component

### @ConfigurationProperties("example")

### public class SampleConfiguration {

### private int number;

### private boolean value;

### private String message;

### According to the above snippet, the values configured in application.properties will be as follows:

### example.number: 100

### example.value: true

### example.message: Dynamic Message

## How to create non-web application in spring boot?

### Yes, we can create a non-web application by removing the web dependencies from the classpath along with changing the way Spring Boot creates the application context.

* 1. Remove web dependencies from pom, only include parent and spring-boot-starter.
  2. Use CommandLineRunner.

<https://knpcode.com/spring-boot/spring-boot-stand-alone-non-web-application/>

## Default H2 database name?

The name of the default H2 database is **testdb.  Refer below:**

spring.datasource.name=testdb # Name of the datasource.

**Note:** Just incase if you are using H2 in-memory database, then exactly that is the name of Spring Boot which is used to setup your H2 database.

## How do you Add, Filter to an application? \*\*\*

There are three methods to add filter to Spring Boot application:

* By implementing Filter interface.
* Using FilterRegistrationBean.
* Using MVC controller.

## What is LiveReload in Spring Boot?

### LiveReload is a spring-boot-devtools module that includes LiveReload server to trigger a browser refresh when a resource is changed. LiveReload server extensions are available freeware for Firefox, Chrome, and Safari.

## How to set logging level to debug?

logging.level.root=DEBUG

## Explain the difference between an embedded container and a WAR.

### The main difference between these two is:

### Embedded containers help you to run Spring Boot application as a JAR from the command prompt without setting up any web server, while to run a WAR you need first to set up Tomcat.

## Why spring boot is production ready?

### Spring Boot aims to enable production ready applications in quick time. Spring Boot provides a few non functional features out of the box like caching, logging, monitoring and embedded servers.

### spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box

### spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container

### spring-boot-starter-logging - For Logging using logback

### spring-boot-starter-cache - Enabling Spring Framework’s caching support

# Integration test

## How to write integration tests?

### When running integration tests for a Spring application, we must have an ApplicationContext.

### To make our life easier, Spring Boot provides a special annotation for testing – @SpringBootTest. This annotation creates an ApplicationContext from configuration classes indicated by its classes attribute.

### In case the classes attribute isn’t set, Spring Boot searches for the primary configuration class. The search starts from the package containing the test up until it finds a class annotated with @SpringBootApplication or @SpringBootConfiguration.

### Notice if we use JUnit 4, we must decorate the test class with @RunWith(SpringRunner.class).

### For detailed instructions, check out our tutorial on testing in Spring Boot.

## What is the difference between springRunner and SpringJunitRunner?

### No difference. SpringRunner is new name to make it easy to read.

# dev tools

## What is Spring Boot DevTools used for?

### Spring Boot Developer Tools, or DevTools, is a set of tools making the development process easier. To include these development-time features, we just need to add a dependency to the pom.xml file:

### <dependency>

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

### <artifactId>spring-boot-devtools</artifactId>

### </dependency>

### The spring-boot-devtools module is automatically disabled if the application runs in production. The repackaging of archives also excludes this module by default. Hence, it won’t bring any overhead to our final product.

### By default, DevTools applies properties suitable to a development environment. These properties disable template caching, enable debug logging for the web group, and so on. As a result, we have this sensible development-time configuration without setting any properties.

### Applications using DevTools restart whenever a file on the classpath changes. This is a very helpful feature in development, as it gives quick feedback for modifications.

### By default, static resources, including view templates, don’t set off a restart. Instead, a resource change triggers a browser refresh. Notice this can only happen if the LiveReload extension is installed in the browser to interact with the embedded LiveReload server that DevTools contains.

### For further information on this topic, please see Overview of Spring Boot DevTools.

# spring actuator

## What is Spring Boot Actuator used for?

### Essentially, Actuator brings Spring Boot applications to life by enabling production-ready features. These features allow us to monitor and manage applications when they’re running in production.

### Integrating Spring Boot Actuator into a project is very simple. All we need to do is to include the spring-boot-starter-actuator starter in the pom.xml file:

### <dependency>

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

### <artifactId>spring-boot-starter-actuator</artifactId>

### </dependency>

### Spring Boot Actuator can expose operational information using either HTTP or JMX endpoints. Most applications go for HTTP, though, where the identity of an endpoint and the /actuator prefix form a URL path.

### Here are some of the most common built-in endpoints Actuator provides:

### auditevents: Exposes audit events information

### env: Exposes environment properties

### health: Shows application health information

### httptrace: Displays HTTP trace information

### info: Displays arbitrary application information

### metrics: Shows metrics information

### loggers: Shows and modifies the configuration of loggers in the application

### mappings: Displays a list of all @RequestMapping paths

### scheduledtasks: Displays the scheduled tasks in your application

### threaddump: Performs a thread dump

This is an interesting Spring Boot question and mostly asked on Java roles which also has some support responsibility. Spring Actuator is another cool Spring Boot feature which allows seeing inside a running application.  
  
Yes, you read it correctly. It allows you to see inside an application. Since Spring Boot is all about auto-configuration it makes debugging difficult and at some point in time, you want to know which [beans](https://javarevisited.blogspot.com/2012/05/what-is-bean-scope-in-spring-mvc.html#axzz5IZi1jCsQ) are created in Spring's Application Context and how Controllers are mapped. Spring Actuator provides all that information.  
  
It provides several endpoints e.g. a REST endpoint to retrieve this kind of information over the web. It also provides a lot of insight and metrics about application health e.g. [CPU and memory usage](http://javarevisited.blogspot.sg/2013/06/find-cpu-and-memory-used-by-java-solaris-prstat-command-example.html), number of threads etc.  
  
It also comes with a remote shell which you can use to securely go inside Spring Boot application and run some command to expose the same set of data. You can even use JMX to control this behavior at runtime.  
  
Btw, it's important to secure your Spring Actuator endpoints because it exposes a lot of confidential information and a potentially dangerous one-two. For example, by using /showdown endpoint you can kill a Spring Boot application.  
  
But, don't worry. You can use Spring Security to secure Spring Actuator endpoints.  
  
Read more: <https://www.java67.com/2018/06/top-15-spring-boot-interview-questions-answers-java-jee-programmers.html#ixzz6jD2eRx00>

## What is a shutdown in the actuator?

### A shutdown is an endpoint that helps application to be shut down properly. This feature is not enabled by default.

### However, you can use it by setting command: management.endpoint.shutdown.enabled=true in your application.properties file.

## How to disable Actuator endpoint security in Spring Boot?

### By default all sensitive HTTP endpoints are secured such that only users that have an ACTUATOR role may access them. Security is enforced using the standard HttpServletRequest.isUserInRole method. We can disable security using - **management.security.enabled=false** It is suggested to disable security only if the actuator endpoints are accessed behind firewall.

## How can we create a custom endpoint in Spring Boot Actuator?

### To create a custom endpoint using Spring Boot 1.x, we should expose the instance of the custom endpoint class as a bean. We need to implement Endpoint<T> interface.

### @Component

### public class CustomEndpoint implements Endpoint {

### //method implimentation

### }

### Copy

### Spring Boot 2.x changed it by introducing @Endpoint annotation. Spring Boot expose endpoints with @Endpoint@WebEndpointor @WebEndpointExtension over HTTP using Jersey, Spring MVC, or Spring WebFlux.Read [Custom Endpoint in Spring Boot Actuator](https://www.javadevjournal.com/spring-boot/spring-boot-actuator-custom-endpoint/) for more detail.

<https://www.javadevjournal.com/spring-boot/spring-boot-actuator-custom-endpoint/>

<https://www.baeldung.com/spring-boot-actuators>

# spring initializer

## What is spring boot initializer?

### Spring Boot Initializr provides a simple interface to quickly bootstrap a Spring Boot application. Here are benefits or advantages of using Initilizr.

### Spring Initializr provides an extensible API to generate quick start projects.

### Reduce time to create an application setup. We can create application setup using a few clicks.

### It increases Productivity

### Initializr offers a configuration structure to define all the aspects related to the project to generate: list of dependencies, supported java and boot versions.

you can see our default instance at [start.spring.io](https://start.spring.io/). It provides a simple web UI to configure the project to generate and endpoints that you can use via plain HTTP.

# spring auto configuration

## What is auto configuration?

### It takes a lot of configurations and boilerplate code create a simple Spring MVC application without Spring Boot. Spring Boot Auto Configuration provides an opinionated approach to bootstrap your application. Auto-Configuration will attempt to automatically try to set up our application with default behavior based on the jars in the classpath.

### For example, if Spring Boot finds HSQLDB in out classpath, it will automatically configure an in-memory database for us. Think of the auto-configuration as an intelligent system which can provide ready to use the application to us based on the configured jars in our classpath.

## How to disable a specific auto-configuration?

### If we want to disable a specific auto-configuration, we can indicate it using the exclude attribute of the @EnableAutoConfiguration annotation. For instance, this code snippet neutralizes DataSourceAutoConfiguration:

### // other annotations

### @EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)

### public class MyConfiguration { }

### If we enabled auto-configuration with the @SpringBootApplication annotation — which has @EnableAutoConfiguration as a meta-annotation — we could disable auto-configuration with an attribute of the same name:

### 1

### 2

### 3

### // other annotations

### @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)

### public class MyConfiguration { }

### We can also disable an auto-configuration with the spring.autoconfigure.exclude environment property. This setting in the application.properties file does the same thing as before:

### **spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration**

### On the other foot, if the class is not on the classpath, you can use the excludeName attribute of the annotation and specify the fully qualified name instead.

## How to register a custom auto-configuration?

### To register an auto-configuration class, we must have its fully-qualified name listed under the EnableAutoConfiguration key in the META-INF/spring.factories file:

### **org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.baeldung.autoconfigure.CustomAutoConfiguration**

### If we build a project with Maven, that file should be placed in the resources/META-INF directory, which will end up in the mentioned location during the package phase.

## How to tell an auto-configuration to back away when a bean exists?

### To instruct an auto-configuration class to back off when a bean is already existent, we can use the @ConditionalOnMissingBean annotation. The most noticeable attributes of this annotation are:

### value: The types of beans to be checked

### name: The names of beans to be checked

### When placed on a method adorned with @Bean, the target type defaults to the method’s return type:

@Configuration

public class CustomConfiguration {

@Bean

@ConditionalOnMissingBean

public CustomService service() { ... }

}

# spring starter

## What is spring boot starter?

### *Spring Boot Starters* are the set of convenient dependency descriptors which can be easily included in any level of application. These starters work as a 4) bootstrapping process for the *Spring* related technologies, we no longer need to worry about the dependencies and they will be automatically managed by Spring Boot Starters.

### The starters contain a lot of the dependencies you need to get a project up and running quickly and with a consistent, supported a set of managed transitive dependencies. To summarize, ***Spring Boot Starters* are just JAR files used by Spring Boot for auto-dependency**.

## What is spring.provides?

### --- THIS Is deprecated now.

### Every starter jar contains a file named spring.provides under META\_INF which list all the dependency used by starter.

### The spring.provides files shipped with each starter JAR have not been widely supported by IDEs and have become a maintenance burden. We intend to remove them in Spring Boot 2.1.

## What Spring Boot starters are available out there?

### Dependency management is a crucial facet of any project. When a project is complex enough, managing dependencies may turn into a nightmare, as there will be too many artifacts involved.

### This is where Spring Boot starters come in handy. Each starter plays a role as a one-stop shop for all the Spring technologies we need. Other required dependencies are then transitively pulled in and managed in a consistent way.

### All starters are under the org.springframework.boot group, and their names start with spring-boot-starter-. This naming pattern makes it easy to find starters, especially when working with IDEs that support searching dependencies by name.

### At the time of this writing, there are more than 50 starters at our disposal. The most commonly used are:

### spring-boot-starter: core starter, including auto-configuration support, logging, and YAML

### spring-boot-starter-aop: starter for aspect-oriented programming with Spring AOP and AspectJ

### spring-boot-starter-data-jpa: starter for using Spring Data JPA with Hibernate

### spring-boot-starter-jdbc: starter for using JDBC with the HikariCP connection pool

### spring-boot-starter-security: starter for using Spring Security

### spring-boot-starter-test: starter for testing Spring Boot applications

### spring-boot-starter-web: starter for building web, including RESTful, applications using Spring MVC

### For a complete list of starters, please see this repository. <https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters>

## What does stater-parent do?

### It is a special starter which makes Gradle or Maven dependency-management easy by adding jars to your classpath.

### Spring Boot provides a number of “Starters” that let you add jars to your classpath. For ex. spring-boot-starter-security,spring-boot-starter-web etc. The "spring-boot-starter-parent" is a special starter that provides useful Maven defaults i.e it adds all the required jars and other things automatically. It also provides a dependency-management section so that you can omit version tags for dependencies you are using in pom.xml

### boot: run runs your Spring Boot application.

### spring-boot:repackage repackages your jar/war to be executable.

### spring-boot:start and spring-boot:stop to manage the lifecycle of your Spring Boot application (i.e. for integration tests).

### spring-boot:build-info generates build information that can be used by the Actuator.

The spring-boot-starter-parent dependency is the parent POM providing dependency and plugin management for Spring Boot-based applications. It contains the default versions of Java to use, the default versions of dependencies that Spring Boot uses, and the default configuration of the Maven plugins.

Few important configurations provided by this file are as below. Please refer to this [link](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-starters/spring-boot-starter-parent/pom.xml) to read the complete configuration.

|  |
| --- |
| pom.xml |
| <?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> <http://maven.apache.org/xsd/maven-4.0.0.xsd;>      <modelVersion>4.0.0</modelVersion>      <parent>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-dependencies</artifactId>          <version>${revision}</version>          <relativePath>../../spring-boot-dependencies</relativePath>      </parent>      <artifactId>spring-boot-starter-parent</artifactId>      <packaging>pom</packaging>      <name>Spring Boot Starter Parent</name>      <description>Parent pom providing dependency and plugin management for applications          built with Maven</description>      <properties>          <java.version>1.8</java.version>          <resource.delimiter>@</resource.delimiter> <!-- delimiter that doesn't clash with Spring ${} placeholders -->          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>          <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>          <maven.compiler.source>${java.version}</maven.compiler.source>          <maven.compiler.target>${java.version}</maven.compiler.target>      </properties>        ...        <resource>          <directory>${basedir}/src/main/resources</directory>          <filtering>true</filtering>          <includes>              <include>\*\*/application\*.yml</include>              <include>\*\*/application\*.yaml</include>              <include>\*\*/application\*.properties</include>          </includes>      </resource>    </project> |

The spring-boot-starter-parent dependency further inherits from spring-boot-dependencies, which is defined at the top of above POM file at line number : 9.

This file is the actual file which contains the information of default version to use for all libraries. The following code shows the different versions of various dependencies that are configured in spring-boot-dependencies:

|  |
| --- |
| pom.xml |
| <properties>      <!-- Dependency versions -->      <activemq.version>5.15.3</activemq.version>      <antlr2.version>2.7.7</antlr2.version>      <appengine-sdk.version>1.9.63</appengine-sdk.version>      <artemis.version>2.4.0</artemis.version>      <aspectj.version>1.8.13</aspectj.version>      <assertj.version>3.9.1</assertj.version>      <atomikos.version>4.0.6</atomikos.version>      <bitronix.version>2.1.4</bitronix.version>      <byte-buddy.version>1.7.11</byte-buddy.version>      <caffeine.version>2.6.2</caffeine.version>      <cassandra-driver.version>3.4.0</cassandra-driver.version>      <classmate.version>1.3.4</classmate.version>        ...      ...  </properties> |

Above list is very long and you can read complete list in this [link](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-dependencies/pom.xml).

## How to override default dependency version?

As you see, spring boot has default version to use for most of dependencies. You can override the version of your choice or project need, in properties tag in your project’s pom.xml file.

e.g. Spring boot used default version of google GSON library as 2.8.2.

|  |
| --- |
| <groovy.version>2.4.14</groovy.version>  <gson.version>2.8.2</gson.version>  <h2.version>1.4.197</h2.version> |

I want to use 2.7 of gson dependency. So I will give this information in properties tag like this.

|  |
| --- |
| <properties>      <java.version>1.8</java.version>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>      <gson.version>2.7</gson.version>  </properties> |

Now in your eclipse editor, you can see the message as : The managed version is 2.7 The artifact is managed in org.springframework.boot:spring-boot-dependencies:2.0.0.RELEASE.

# Spring boot CLI

## What is Spring boot CLI

### Spring Boot CLI is a command line interface which allows you to create Spring-based Java application using Groovy. Since it's used Groovy, it allows you to create Spring Boot application from the command line without ceremony e.g. you don't need to define getter and setter method, or access modifiers, return statements etc. It's also very powerful and can auto-include a lot of library in Groovy's default package if you happen to use it. For example, if you use JdbcTempalte, it can automatically load that for you. Read more: <https://www.java67.com/2018/06/top-15-spring-boot-interview-questions-answers-java-jee-programmers.html#ixzz6jD3b8L88>

## Spring boot cli advantaegs:

With Spring Boot CLI:

### No Semicolons

### No Public and private access modifiers

### No Imports(Most)

### No “return” statement

### No setters and getters

### No Application class with main() method(It takes care by SpringApplication class).

### No Gradle/Maven builds.

### No separate HTTP Servers.

## Spring Boot admin

<https://www.baeldung.com/spring-boot-admin#:~:text=Spring%20Boot%20Admin%20is%20a,the%20Spring%20Boot%20Actuator%20endpoints>.

<https://www.javainuse.com/spring/boot-admin>

### Spring Boot provides actuator endpoints to monitor metrics of individual microservices. These endpoints are very helpful for getting information about applications like if they are up, if their components like database etc are working good. But a major drawback or difficulty about using actuator enpoints is that we have to individually hit the enpoints for applications to know their status or health. Imagine microservices involving 50 applications, the admin will have to hit the actuator endpoints of all 50 applications. To help us deal with this situation, we will be using open source project located at <https://github.com/codecentric/spring-boot-admin>. Built on top of Spring Boot Actuator, it provides a web UI to enable us visualize the metrics of multiple applications.

# Reference

<https://www.guru99.com/spring-boot-interview-questions.html>

<https://www.javainuse.com/spring/SpringBootInterviewQuestions>

<https://www.javadevjournal.com/spring-boot/spring-boot-interview-questions/>