Q . What is the use of @EnableAutoConfiguration annotation?

Ans.

* This annotation tells Spring Boot to “guess” how we will want to configure Spring, based on the jar dependencies and starters that we have added.
* Spring loads a default set of configuration based on dependencies added in pom.xml or Gradle.build.

@**EnableAutoConfiguration** annotation **auto-configures the beans that are present in the classpath**. This simplifies the developers work by guessing the required beans from the classpath and configure it to run the application. This annotation is part of the spring boot project.

For example, if we have **tomcat-embedded.jar in the classpath**, then we will need a **TomcatEmbeddedServletContainerFactory bean to configure the tomcat server**. This will be searched and configured without any manual XML configurations.

With the spring boot 1.2.0 release, the need for this annotation has been reduced because there is an alternative annotation **@SpringBootApplication** which combines the three annotations @**Configuration**,@**EnableAutoConfiguration** and code>@**ComponentScan**.

The package of the class that is annotated with @EnableAutoConfiguration has specific significance and is often used as a ‘default’. For example, it will be used when scanning for @Entity classes. It is generally recommended that we place @EnableAutoConfiguration in a root package so that all sub-packages and classes can be searched.

Auto-configuration classes are normal @Configuration annotated classes only. These are mentioned in the spring.factories file. Spring checks the spring.factories files under the folder META-INF in our project or JAR file to auto-configure the configuration classes.

**@EnableAutoConfiguration Parameters**

The following are the parameters that can be passed inside this annotation:

* 1. **exclude** – Exclude the **list of classes** from the auto configuration.
  2. **excludeNames** – Exclude the list of **fully qualified class names** from the auto configuration. This parameter added since spring boot 1.3.0.

The above parameters help we to exclude the list of configuration classes that are not required to be auto-configured.

Even though both @**SpringBootApplication** and @**EnableAutoConfiguration** can be used to enable the auto-configuration feature of Spring Boot, there is a subtle difference between them. The @**SpringBootApplication** does much more than what @**EnableAutoConfiguration** does. It's actually a combination of three annotations: @Configuration, which is used in Java-based configuration on Spring framework, @ComponentScan to enable component scanning of components we write like @Controller classes, and @EnableAutoConfgiuration itself, which is used to allow for auto-configuration in Spring Boot application.

Spring Boot designers quickly realized that these three annotations are frequently used together, so they bundled them into @**SpringBootApplicaiton**. Now, instead of three annotations, we just need to specify one annotation on our Main class.

Q. What is Auto-Configuration in Spring Boot?

If we do not know, Spring Boot aims to simplify Java development with the Spring framework. Currently, Spring does a lot for we, but in return, it also asks a lot from we in terms of configuration and dependency management, Spring Boot aims to solve that problem.

The Spring Boot auto-configuration feature tries to **automatically configure our Spring application based upon the JAR dependency we have added in the classpath**.

For example, if HSQLDB is present on our classpath and we have not configured any database manually, Spring will auto-configure an in-memory database for we.

By default, this auto-configuration feature is not enabled, and we need to opt-in for it by adding the **@EnableAutoConfiguration** or **@SpringBootApplicaiton** annotations to one of our @Configuration classes, generally the Main class which is used to run our application.

Difference between @EnableAutoConfiguration and @SpringBootApplication

**1. Availability**

The @**SpringBootApplicaiton** is relatively new than @**EnableAutoConfiguration**. It was introduced in Spring Boot 1.2 release while @EnableAutoConfiguation is present form the Spring Boot 1.0 release.

**2. Purpose**

The apparent purpose of @**EnableAutoConfiguration** is to enable automatic configuration features of the Spring Boot application, which **automatically configures things if certain classes are present in classpath** e.g., it can configure **Thymeleaf TemplateResolver** and **ViewResolver** if Thymeleaf is present in the classpath.

On the other hand, @**SpringBootApplication** does three things, it allows we to run the Main class as a JAR with an embedded container. It enables Java configuration, and it also enables Component Scanning.

**3. Uses**

It's not mandatory to put @SpringBootApplication to create a Spring Boot application, we can still use **@Configuration** and **@EnableAutoConfiguration** individually as shown in the example given in next point.

**4. Control**

The @**EnableAutoConfiguration** annotation allows we to selectively exclude certain classes from auto-configuration using exclude attribute as shown below:

@Configuration

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

public class MyConfiguration {

//.. Java code

}

If the class is not on the classpath, we can use the excludeName attribute of the @EnableAutoConfiguration annotation and specify the fully qualified class name.

**Important points**

1. We should annotate the Main class or Bootstrap class with the @**SpringBootApplication**; this will allow we to run as a JAR with embedded web server Tomcat. If we want, we can change that to Jetty or Undertow.

2. The @**SpringBootApplication** is a combination of three annotations @**Configuration** (used for Java-based configuration), @**ComponentScan** (used for component scanning), and @**EnableAutoConfiguration** (used to enable auto-configuration in Spring Boot).

3. The @EnableAutoConfiguration annotations enable auto-configuration features of Spring Boot, which **configures modules based on the presence of certain classes on the classpath**. For example, if Thymeleaf JAR is present in classpath and Spring MVC is enabled like using a spring-boot-web-starter package, then it can automatically configure template resolver and view resolver for we.

4. The @EnableAutoConfiguration annotation is based on @Conditional annotation of Spring 4.0, which enables conditional configuration.

5. In the case of auto-configuration, manually declared beans can override beans automatically created by auto-configuration feature. **This is achieved by using @ConditionalOnMissingBean of Spring 4.0**

6. If we are using @EnableAutoConfiguration classes, then we can selectively **exclude certain classes from auto-configuration by using exclude as shown below:**

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

7. The @SpringBootApplication annotation also provides aliases to customize the attributes of **@EnableAutoConfiguration and @ComponentScan annotations.**

**@EnableAutoConfiguration**

This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for we.

For example, if we are writing a Spring MVC application and we 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, we 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 we 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.

This will start the embedded server that comes along with Spring Boot and runs our web application inside it. Once we see the log without any errors, we can go to the browser and open the localhost with the server port to access our Spring Boot application.

That's all about the @SpringBootApplication annotation and a simple application to demonstrate how to use it. As I said, this nice little annotation packs quite a lot of punch. We can just write this one line of code to enable Java-based configuration, component scanning, and to enable the auto-configuration feature of Spring Boot. It makes our code more readable.

### Q. What is Spring Boot Starter?

* Spring Boot provides a number of “Starters” that make easy to manage dependencies for our project.
* When we are adding more than one starters spring **automatically detects the compatible version for integrating it.**
* Ex: if we add spring boot web and spring boot JPA starters**, spring automatically detects the right version of spring, JPA and hibernate.**
* **Spring Boot starter** makes **it easier to add jars to the classpath.** While creating a applications managing dependency is a complex task. The keen objective of spring boot starter is reducing time while managing dependency manually. Spring boot has more than 30 starters for making this complicated stuff easier.
* **Spring boot maven plugin**
* The **Spring Boot Maven Plugin** provides Spring Boot support in Maven, letting we package executable jar or war archives and run an application “in-place”. To use it, we must use Maven 3.2 (or later).

<plugin>

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

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

</plugin>

* **Spring-boot-starter-parent maven**
* The **spring-boot-starter-parent** provides Maven and spring default configuration and dependency-management section so that we can omit version tags for “blessed” dependencies.

<parent>

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

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

<version>1.5.6.RELEASE</version>

</parent>

Note : We can also declare our own parent if needed refer

Spring-boot-starter-web maven

Other “Starters” simply provide dependencies that we are likely to need when developing a specific type of application. Since we are developing a web application, we will add a spring-boot-starter-web dependency to our opm.xml file

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

* **Spring boot dependencies – Spring-boot-starter maven for Web application**

<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>

<groupId>com.candidjava.spring.boot</groupId>

<artifactId>helloworld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>helloworld</name>

<url>http://maven.apache.org</url>

<parent>

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

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

<version>1.5.6.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>



### 8. What is spring-boot-starter-parent?

* The spring-boot-starter-parent is a special starter that makes Maven or Gradle dependency-management easier by adding jars to our classpath.
* It **adds a basic set of spring jars needed** for any type of spring based applications.
* **Spring Boot starter** makes it easier to add jars to the classpath. While creating a applications managing dependency is a complex task. The keen objective of spring boot starter is reducing time while managing dependency manually. Spring boot has more than 30 starters for making this complicated stuff easier.

### Spring boot maven plugin

* The **Spring Boot Maven Plugin** provides Spring Boot support in Maven, letting we package executable jar or war archives and run an application “in-place”. To use it, we must use Maven 3.2 (or later).

<plugin>

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

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

</plugin>

### Spring-boot-starter-parent maven

* The **spring-boot-starter-parent** provides Maven and spring default configuration and dependency-management section so that we can omit version tags for “blessed” dependencies.

<parent>

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

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

<version>1.5.6.RELEASE</version>

</parent>

* ***Note : We can also declare our own parent if needed refer***

### Spring-boot-starter-web maven

* Other “Starters” simply provide dependencies that we are likely to need when developing a specific type of application. Since we are developing a web application, we will add a **spring-boot-starter-web** dependency to our opm.xml file

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

### Spring boot dependencies – Spring-boot-starter maven for Web application

<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>

<groupId>com.candidjava.spring.boot</groupId>

<artifactId>helloworld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>helloworld</name>

<url>http://maven.apache.org</url>

<parent>

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

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

<version>1.5.6.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

### 9. What is spring-boot-starter-web?

This starter will add Tomcat and Spring MVC dependency to our application and its default configuration.

**Spring Boot starter** makes it easier to add jars to the classpath. While creating a applications managing dependency is a complex task. The keen objective of spring boot starter is reducing time while managing dependency manually. Spring boot has more than 30 starters for making this complicated stuff easier.

### Spring boot maven plugin

The **Spring Boot Maven Plugin** provides Spring Boot support in Maven, letting we package executable jar or war archives and run an application “in-place”. To use it, we must use Maven 3.2 (or later).

<plugin>

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

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

</plugin>

### Spring-boot-starter-parent maven

The **spring-boot-starter-parent** provides Maven and spring default configuration and dependency-management section so that we can omit version tags for “blessed” dependencies.

**<parent>**

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

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

**<version>1.5.6.RELEASE</version>**

**</parent>**

### Spring-boot-starter-web maven

Other “Starters” simply provide dependencies that we are likely to need when developing a specific type of application. Since we are developing a web application, we will add a **spring-boot-starter-web** dependency to our opm.xml file

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

Spring boot dependencies – Spring-boot-starter maven for Web application

<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>

<groupId>com.candidjava.spring.boot</groupId>

<artifactId>helloworld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>helloworld</name>

<url>http://maven.apache.org</url>

<parent>

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

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

<version>1.5.6.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

### **Q. How to create an executable jar using spring boot?**

Add this below plugin to

pom.xml

**<build>**

<plugins>

<plugin>

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

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

**</plugin>**

</plugins>

</build>

**Use mvn clean package to create the executable jar file.**

### **Q. How do we run and stop spring boot executable jar?**

* Open cmd or shell window and use java -jar as shown below
* $ java -jar my project-0.0.1-SNAPSHOT.jar
* To stop use ctrl+C
* To create an executable jar we need to add the spring-boot-maven-plugin to our pom.xml. Insert the following lines just below the dependencies

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

Use maven goal as below to create a executable jar

mvn clean package



### **Q. How do we add Add a Servlet, Filter or Listener to an application?**

There are two ways to add Servlet, Filter, ServletContextListener and the other listeners supported by the Servlet spec to our application. We can either provide Spring beans for them or enable scanning for Servlet components.

**Q. What is the use of @EnableAutoConfiguration annotation?**

This annotation tells Spring Boot to “guess” how we will want to configure Spring, based on the jar dependencies and starters that we have added.

Spring loads a default set of configuration based on dependencies added in pom.xml or Gradle.build.

**Q. How do we change JDK version in spring boot?**

Java 1.6 as the default compiler level.

We can overwrite it by adding Java.version property tag as shown below

<properties>

<**java.version**>1.8</java.version>

</properties>

**Q. What is spring-boot-dev tools?**

Applications that use spring-boot-dev-tools 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.

**Q. What are Profiles in spring boot?**

Spring Profiles provide a way to segregate parts of our application configuration and make it only available in certain environments. **Any @Component or @Configuration** can be marked with **@Profile** to limit when it is loaded.

**Q. Spring Boot Actuator?**

**Spring Boot Actuator help we monitor and manage our application health when we push it to production.**

We can choose to manage and monitor our application by using HTTP endpoints.

**Q. How do we Change tomcat or jetty HTTP port?**

We can change the Tomcat HTTP port by changing default HTTP property in the application.properties file.

**Q. How to disable specific auto-configuration in spring boot?**

We can use exclude property as shown below to disable specific auto-configuration.

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

## Q. What is Spring Boot? Why Spring Boot? How does it work? What are advantage of it?

Spring Boot is a rapid application development platform built on top of the popular Spring Framework.

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that we can **just run**. we can get started with minimum configurations without the need for an entire Spring configuration setup.

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

Spring typical requires a lot of configuration. Spring Boot simplifies this setup by providing defaults for many features. We can still adjust the defaults according to our needs.

Spring Boot is basically an extension of the Spring framework which eliminated the boilerplate configurations required for setting up a Spring application.

It takes an opinionated view of the Spring platform which paved the way for a faster and more efficient development eco-system.

**Why it is used :**

We can choose Spring Boot because of the features and benefits it offers as given here −

* It provides a **flexible way to configure Java Beans**, **XML configurations**, and **Database Transactions**.
* It provides a **powerful batch processing and manages REST endpoints**.
* In Spring Boot, **everything is auto configured; no manual configurations are needed.**
* **It offers annotation-based spring application**
* **Ease’s dependency management**
* **It includes Embedded Servlet Container**

**How it woks: -**

Spring Boot automatically configures our application based on the dependencies we have added to the project by using **@EnableAutoConfiguration** annotation. For example, **if MySQL database is on our classpath**, but we have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains @**SpringBootApplication** annotation and the main method.

Spring Boot **automatically scans all the components** included in the project by using **@ComponentScan** annotation.

### **Advantages and** **Features**

**Advantages**

Spring Boot offers the following advantages to its developers −

* **Easy to understand** and develop spring applications
* **Increases productivity**
* **Reduces the development time**
* Provide a range of non-functional features that are common to large classes of projects.
* **Auto-configuration** (loads set of default configuration for a quick start any application).
* Spring boot starters (**Dependency management**).
* **Embedded servlet containers** (easy to build micro-services architectures).
* **Actuators** (wide range of build-in API for **health checks and monitoring**).
* Externalized configuration (helps to centralizing it and shares between multiple servers).
* Profiling (managing application **mode like dev, production, testing**, etc.).
* Security, Metrics and lot more.
* Absolutely no code generation and **no requirement for XML configuration.**Provide a radically **faster** and widely accessible getting started experience for all Spring development.
* Reduces the time spent on development and **increases the overall efficiency** of the development team.
* Helps to **autoconfigure** all components for a production-grade Spring app.
* Facilitates the creation and **testing of Java-based applications** by providing a default setup for unit and integration tests.
* Helps to **avoid all the manual work** of writing boilerplate code, annotations, and complex XML configurations.
* Comes with **embedded HTTP servers** like **Jetty** and **Tomcat** to test web applications.
* The integration of Spring Boot with the Spring ecosystem which includes **Spring Data**, **Spring Security**, **Spring ORM**, and **Spring JDBC** is easy.
* Provides many plugins that developers can use to work with embedded and in-memory databases smoothly and readily.
* Allows for easily connecting with database and queue services like Oracle, PostgreSQL, MySQL, MongoDB, Redis, Solr, ElasticSearch, Rabbit MQ, ActiveMQ, and many more.
* Provides admin support – we can manage via remote access to the application.
* Eases the dependency and comes with **Embedded Servlet Container**.
* Offers flexibility in configuring XML configurations, Java Beans, and Database Transaction.
* Offers easy access to Command Line Interface which makes the development and testing of Spring Boot apps built with Java or Groovy agile.
* No need of creating boilerplate configuration
* **DevTools to auto restart server on code/config updates**
* Embedded **Tomcat/Jetty/Undertow** support
* Easier customization of **application properties**
* Easy management of profile specific properties
* Easier **dependency management** using platform
* The primary advantage is spring boot offers an effortless way to create spring-based applications using JAVA or Groovy.
* Alongside, it also provides a lot of plugins which aids in effortless development and testing of Spring Boot application build with the help of tools like Gradle and Maven.
* The **Spring Initializer provides** a project generator to make we productive with the certain technology stack from the beginning. We can create a skeleton project with web, data access (relational and NoSQL datastores), cloud, or messaging support.
* That means Spring Boot is nothing but existing **Spring Framework + Some Embedded HTTP Servers** (Tomcat/Jetty etc.) – XML or Annotations Configurations.
* It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
* It provides CLI (**Command Line Interface) tool to develop and test** Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using **Build Tools like Maven and Gradle**
* It provides lots of plugins to work with **embedded and in-memory Databases** very easily.
* No XML based configurations at all. Very much simplified properties. The beans are initialized, configured and wired automatically.
* The **Spring Initializer** provides a project generator to make we productive with the certain technology stack from the beginning. We can create a skeleton project with web, data access (relational and NoSQL datastores), cloud, or messaging support.

In Simple Terminology, What Spring Boot means

[What Is Spring Boot, Spring Boot Tutorial](https://cdn.journaldev.com/wp-content/uploads/2015/05/WhatIsSpringBoot1.png)

That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) – XML or Annotations Configurations.  
Here minus means we don’t need to write any XML Configuration and few Annotations only.

Goals

Spring Boot is designed with the following goals −

* To avoid complex XML configuration in Spring
* To develop a production ready Spring applications in an easier way
* To reduce the development time and run the application independently
* Offer an easier way of getting started with the application

**How does it work?**

Here are just a few of the features in Spring Boot:

* Opinionated ‘starter' dependencies to simplify build and application configuration
* Embedded server to avoid complexity in application deployment
* Metrics, Health check, and externalized configuration
* Automatic config for Spring functionality – whenever possible