SPRING BOOT

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.

.

**What is Micro Service?**

Micro Service is an architecture that allows the developers to develop and deploy services

independently. Each service running has its own process and this achieves the lightweight

model to support business applications.

**Advantages**

Micro services offers the following advantages to its developers:

Easy deployment

Simple scalability

Compatible with Containers

Minimum configuration

Lesser production time

**What is Spring Boot?**

Spring Boot provides a good platform for Java developers to develop a stand-alone and

production-grade spring application that you can **just run**. You can get started with

minimum configurations without the need for an entire Spring configuration setup.

**Advantages**

Spring Boot offers the following advantages to its developers:

Easy to understand and develop spring applications

Increases productivity

Reduces the development time

**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

**Why Spring Boot?**

You 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

Eases dependency management

It includes Embedded Servlet Container

**How does it work?**

Spring Boot automatically configures your application based on the dependencies you have

added to the project by using **@EnableAutoConfiguration** annotation. For example, if

MySQL database is on your classpath, but you 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.]

**Spring Boot Application**

The entry point of the Spring Boot Application is the class contains

**@SpringBootApplication** annotation. This class should have the main method to run the

Spring Boot application. **@SpringBootApplication** annotation includes Auto-

Configuration, Component Scan, and Spring Boot Configuration.

If you added **@SpringBootApplication** annotation to the class, you do not need to add

the **@EnableAutoConfiguration**, **@ComponentScan** and **@SpringBootConfiguration**

annotation. The **@SpringBootApplication** annotation includes all other annotations.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**Component Scan**

Spring Boot application scans all the beans and package declarations when the application

initializes. You need to add the **@ComponentScan** annotation for your class file to scan

your components added in your project.

**Spring Boot Starters**

Handling dependency management is a difficult task for big projects. Spring Boot resolves

this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you

include **spring-boot-starter-data-jpa** dependency in your project.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-**

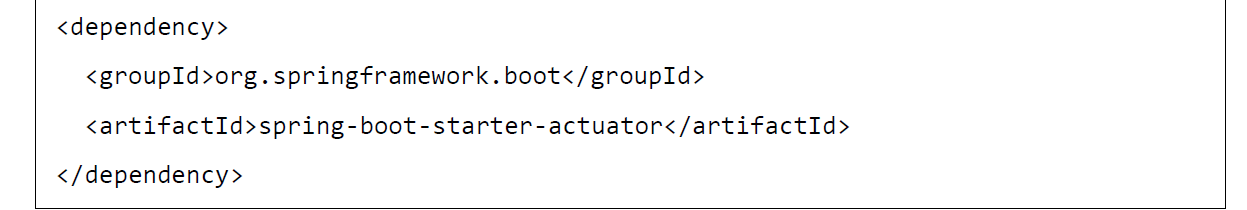
**\***, where **\*** indicates that it is a type of the application.

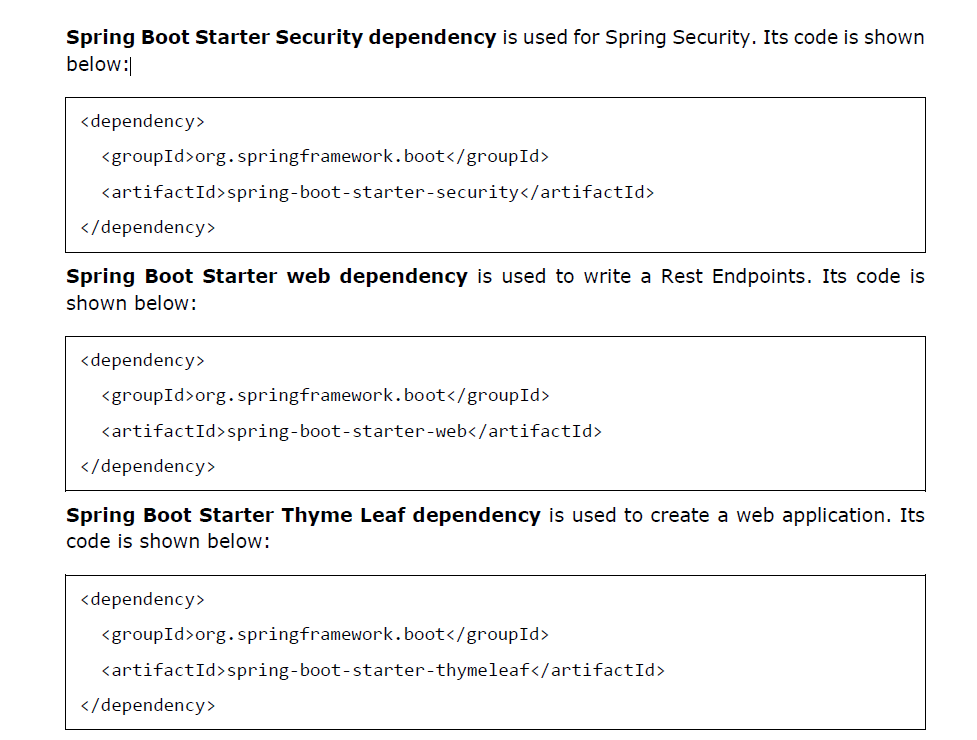
**Examples**

Look at the following Spring Boot starters explained below for a better understanding:

**Spring Boot Starter Actuator dependency** is used to monitor and manage your

application. Its code is shown below:





**Auto Configuration**

Spring Boot Auto Configuration automatically configures your Spring application based on

the JAR dependencies you added in the project. For example, if MySQL database is on your

class path, but you have not configured any database connection, then Spring Boot autoconfigures an in-memory database.

For this purpose, you need to add **@EnableAutoConfiguration** annotation or

**@SpringBootApplication** annotation to your main class file. Then, your Spring Boot

application will be automatically configured.

**Spring Boot Tomcat Deployment**

**Spring Boot Servlet Initializer:**

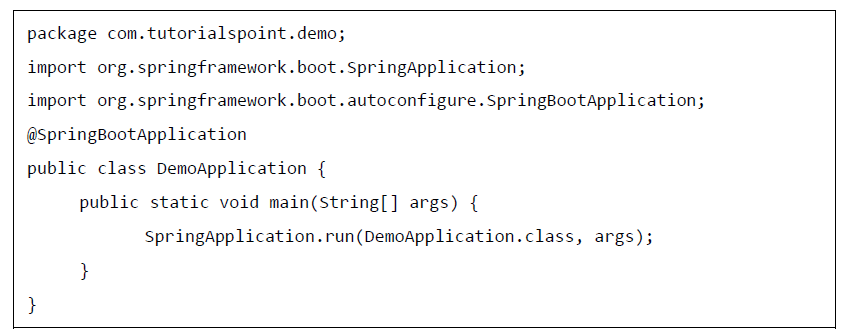
The traditional way of deployment is making the Spring Boot Application

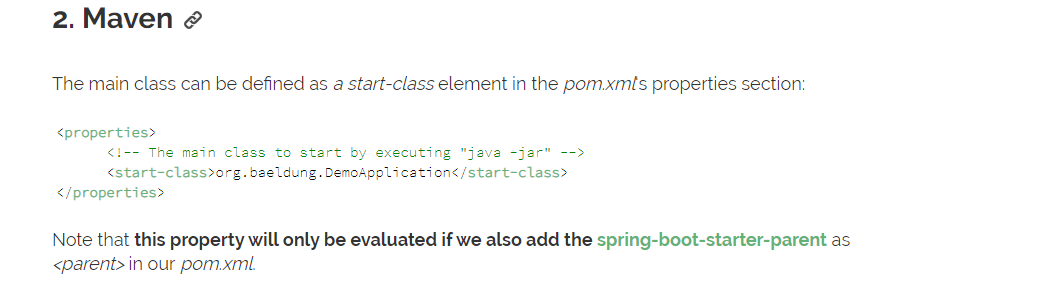
**@SpringBootApplication** class extend the **SpringBootServletInitializer** class. Spring

Boot Servlet Initializer class file allows you to configure the application when it is launched

by using Servlet Container.

The code for Spring Boot Application class file for JAR file deployment is given below:





**Packaging your Application**

Now, create a WAR file to deploy into the Tomcat server by using Maven and Gradle

commands for packaging your application as given below:

For Maven, use the command **mvn package** for packaging your application. Then, the

WAR file will be created and you can find it in the target directory as shown in the

screenshots given below:



