Spring Boot

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

Spring Boot is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring framework.

Our Spring Boot Tutorial includes all topics of Spring Boot such, as features, project, maven project, starter project wizard, Spring Initializr, CLI, applications, annotations, dependency management, properties, starters, Actuator, JPA, JDBC, etc.

If you're looking for information about a specific version, or instructions about how to upgrade from an earlier release, check out the project release notes section on our wiki.

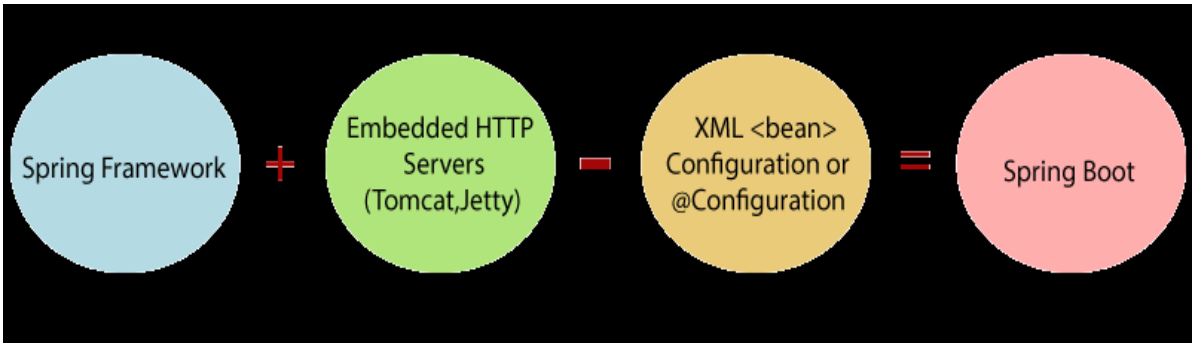
Features :

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration.

What is Spring Boot ?

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



In short, Spring Boot is the combination of Spring Framework and Embedded Servers.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

Note : We can use Spring STS IDE or Spring Initializr to develop Spring Boot Java applications.

STS Download Link : https://dist.springsource.com/release/STS/index.html

Why should we use Spring Boot Framework?

We should use Spring Boot Framework because:

The dependency injection approach is used in Spring Boot.

It contains powerful database transaction management capabilities.

It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.

It reduces the cost and development time of the application.

Along with the Spring Boot Framework, many other Spring sister projects help to build applications addressing modern business needs. There are the following Spring sister projects are as follows:

Spring Data: It simplifies data access from the relational and NoSQL databases.

Spring Batch: It provides powerful batch processing.

Spring Security: It is a security framework that provides robust security to applications.

Spring Social: It supports integration with social networking like LinkedIn.

Spring Integration: It is an implementation of Enterprise Integration Patterns. It facilitates

integration with other enterprise applications using lightweight messaging and declarative adapters.

Advantages of Spring Boot:

- It creates stand-alone Spring applications that can be started using Java -jar.
- It tests web applications easily with the help of different Embedded HTTP servers such as Tomcat, Jetty, etc. We don't need to deploy WAR files.
- It provides opinionated 'starter' POMs to simplify our Maven configuration.
- It provides production-ready features such as metrics, health checks, and externalized configuration.
- There is no requirement for XML configuration.
- It offers a CLI tool for developing and testing the Spring Boot application.
- It offers the number of plug-ins.
- It also minimizes writing multiple boilerplate codes (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
- It increases productivity and reduces development time.

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 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 −

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Spring Boot Starter Security dependency is used for Spring Security. Its code is shown below −

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Spring Boot Starter web dependency is used to write a Rest Endpoints. Its code is shown below −

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Boot Starter Thyme Leaf dependency is used to create a web application. Its code is shown below −

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Spring Boot Starter Test dependency is used for writing Test cases. Its code is shown below −

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

@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 auto configures 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.

Observe the following code for a better understanding −

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
```

}

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.

Observe the following code for a better understanding −

```
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.

Observe the following code for a better understanding −

```
import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

@ComponentScan
```

```
public class DemoApplication {

	public static void main(String[] args) {

		SpringApplication.run(DemoApplication.class, args);

	}

}
```