Spring Boot

Spring Boot is an open source Java-based framework used to create a Micro Service. It is easy to create a stand-alone and production ready spring applications using Spring Boot. Spring Boot contains a comprehensive infrastructure support for developing a micro service and enables you to develop enterprise-ready applications that you can **“just run”**.

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.

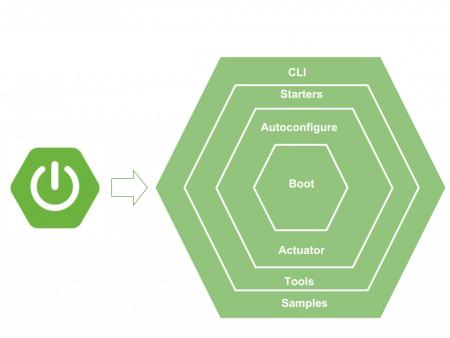
Advantages

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** is a Spring framework module which provides RAD (**Rapid Application Development**) feature to the Spring framework. It is highly dependent on the **starter templates** feature which is very powerful and works flawlessly.



## 1. What is starter template?

Spring Boot starters are templates that contain a **collection of all the relevant transitive dependencies** that are needed to start a particular functionality. For example, If you want to create a Spring WebMVC application then in a traditional setup, you would have included all required dependencies yourself. It leaves the chances of **version conflict** which ultimately result in more **runtime exceptions**.

### 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>

## Why Spring Boot?

* Spring boot helps in **resolving dependency conflict**. It identifies required dependencies and import them for you.
* It has information of **compitable version** for all dependencies. It minimizes the runtime **classloader** issues.
* It’s “opinionated defaults configuration” approach helps you in configuring most important pieces behind the scene. Override them only when you need. Otherwise everything just works, perfectly. It helps in avoiding **boilerplate code**, annotations and XML configurations.
* It provides embedded HTTP server Tomcat so that you can develop and test quickly.
* It has excellent integration with IDEs like eclipse and **intelliJ idea**.

# Spring Boot Features

## 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.

## 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 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.

## Embedded server

Spring boot applications always include **tomcat** as **embedded server** dependency. It means you can run the Spring boot applications from the command prompt without needling complex server infrastructure.

You can exclude tomcat and include any other embedded server if you want. Or you can make exclude server environment altogether. It’s all configuration based.

<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>

## Externalized Configuration

Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments. Application use YAML files to externalize configuration.

## Properties Files

Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: **server-port = 8082** and many others. It helps to organize application properties.

## YAML Support

It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.

## Type-safe Configuration

Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

## Logging

Spring Boot uses Common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies, if there is no required customization is needed.

## Security

Spring Boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of Endpoints are available for develop a secure Spring Boot application.

## Spring Initializer

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page [www.start.spring.io](https://start.spring.io/) and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.

# References

<https://www.tutorialspoint.com/spring_boot/index.htm>

<https://www.javatpoint.com/spring-boot-tutorial>

<https://www.udemy.com/spring-hibernate-tutorial/>