Spring Boot

Spring Boot is an open source Java-based framework used to create a Micro Service. It is developed by Pivotal Team. 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 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.

@SpringBootApplication::--

A single `@SpringBootApplication` annotation can be used to enable those three features, that is:

- `@EnableAutoConfiguration`: enable Spring Boot's auto-configuration mechanism
- `@ComponentScan`: enable `@Component` scan on the package where the application is located
- `@Configuration`: allow to register extra beans in the context or import additional configuration classes

The `@SpringBootApplication` annotation is equivalent to using `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`

Project Creation ways:--

1. Maven project with Eclipse{ we should add all the necessary dependencies}
2. Create directly starter project with STS
3. By using SpringInitaizer (web url) very advantage spring boot provided
4. Spring CLI

SpringInitiaizer

- We can create our springboot application with SpringInitaizer

<parent> tag { Maven dependency version conflict }

- **Spring::--** if you are working with spring we should know the versions of maven dependencies before you just added the dependencies otherwise you will get version conflicts

- **Spring boot:--** here you no need to worry about versions and its conflicts bcz spring boot provided the feature like **parent**, you have to mention only spring boot version of this project, remaining versions and all you no need to mention,

whenever you will add the maven dependencies it will fetch the relate versions or supported versions of parent version.

- Here no worries of version conflicts.
- If you want specify the version of your dependencies we can specify, but it will override the default version taken by spring boot.

Starters

- Spring boot provided a very good feature like earlier versions of spring if you want to add database related dependencies, we should add lot more dependencies.
- In a spring boot introduced starters, it means collection of dependencies.

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

All the starters follow the same structure as like above :: spring-boot-starter-*

- Ex:- starters
  1.web, data, security, devtools, actuators, test

Note:-- we can add normal maven dependencies also in pom.xml

Embedded servers

- Spring boot provided the embedded servers(default servers) like
  1. Tomact
  2. Jetty
  3. Undertow
- By default spring boot will consider the **apache tomcat**.
- If you don't want use this server, if you want to run in your external servers we can deploy it
  1. Add <packaging> war </packaging>
  2. Inside <properties> tag provide a <start-class> tag to give main method class fully qualified name.
  3. Add maven dependency wherever you want deploy your application (tomcat starter or jboss)
  4. You may have to create a class like

```
public class ServeletInitializer extends SpringBootServletInitializer{
  @override
  protected SpringApplicationBuilder configure(SpringApplicationBuilder application){
     return application.sources(SpringBootApplication.class);
   }
}
```

```
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.2.4.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.example</groupId>
	<artifactId>SpringBoot_ExternalServer</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>war</packaging>
	 <name>SpringBoot_ExternalServer</name>
	<description>Demo project for Spring Boot</description>

	<properties>
		<java.version>1.8</java.version>
	<startclass>/SpringBoot_ExternalServer/src/main/java/com/example/dem
	o/SpringBootExternalServerApplication.java</start-class>
	</properties>

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

	// external dependency
	<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-startertomcat</artifactId>
	</dependency>
</project>
```

- This will generate the war file if you build, we can deploy it in any web servers.

Auto Configuration

- When the server starts spring boot will see the classes from the class path what and all available in class path(our created & available jars from external), Spring boot will do the auto configure.
  if you see the loggers spring boot auto-configuration perform like it will search in class path like below
  1. Positive Matches: available classes matched the conditions if available – auto configure
  2. Negative Matches: classes are not available and may be it not satisfied the conditions, it will not do the auto configure.
  EX:-- if you want to see these details you should enable the **logging.level.root = debug** in application.properties file

Developer Tools

- If you add devtools dependency to your application in pom.xml
- Whenever you will do any changes in the classpath the server will restart automatically..
- If you stop the server and restart the server will take more time to compare with auto restart

Actuator

Spring-boot-starter-actuator :: maven dependency

- Actuator uses for monitoring our application, this feature will give like production ready application.
- Add actuator dependency to our pom.xml

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

- Whenever you will add this dependency, we can get the information(monitor) about our application like

1. We can see the beans list
2. We can see how many calls came to our application
3. Which services call how many times
4. We can see the metrix
5. Health – server is up or not
6. Database details & memory utilization details --------- like this so many we can monitor by using this feature.

We can access through **localhost:8080/actuator**

Note:--

* now we can see the few of the urls for monitoring

* If you add below line in application.properties file, we can see all links to monitor different areas`management.endpoints.web.exposure.include=*`

Note:--

```
<dependency>
	<groupId>org.springframework.data</groupId>
	<artifactId>spring-data-rest-hal-browser</artifactId>
</dependency>
```

If you add above dependency we can get the hal browser for monitor our rest urls

NOTE:-- we should not give access to all, we should add below line in application.properties file to restrict access ..

EX: --
/auditevents: we can see the security how many people failed to connect this application.

/beans:: displays all the beans configured in this project..

/health: check the health of our application { status: up }

/metrix: we can get all information here like jvm memory ---

ENDPOINT	USAGE
`/env`	Returns list of properties in current environment
`/health`	Returns application health information.
`/auditevents`	Returns all auto-configuration candidates and the reason why they 'were' or 'were not' applied.
`/beans`	Returns a complete list of all the Spring beans in your application.
`/trace`	Returns trace logs (by default the last 100 HTTP requests).
`/dump`	It performs a thread dump.
`/metrics`	It shows several useful metrics information like JVM memory used, system CPU usage, open files, and much more.

`We may get some performance issues if you add all these features..`