Spring Boot is a Java framework that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run production-ready applications.

**Need for Spring Boot**

1. Using plain Spring Framework requires lot of configurations. If we want to use Spring MVC we need to use @ComponentScan annotation, Dispatcher Servlet, view resolver configuration and other jars. This kind of bootstrapping of applications will be time taking when building microservices.
2. A lot of manual configuration needs to be done to achieve production-ready application.
3. **Dependency Management:** Configuration includes dependency management which developer must manage the required dependencies and their version compatibility.
4. Configuration needed on web.xml
5. Configuration like component scan, view resolver etc. in context.xml
6. Take care of Non-Functional Requirements (NFR) like logging, Error Handling, Monitoring etc.

Spring Boot is all about bootstrapping applications in an opinionated way with some default setting that can be overridden through configuration. Which means Spring Boot is a pre-configured spring platform.

Spring Boot is a convention over configuration extension for Spring Framework.

|  |
| --- |
| **Convention over Configuration**  Convention Over Configuration is a software design paradigm used by frameworks. It is used to reduce the configuration needed by the framework. Frameworks will follow a convention to achieve the tasks and any deviation from the convention will need a configuration.  For example, Maven uses this design paradigm, Maven follows the convention of having generic project directory structure of where to place the source files, resources, test files etc. Any deviation from this requires extra configuration. |

**Spring Boot Components**

1. **Starter projects:**

Spring boot has built-in starter projects which makes development easier and faster. Starter projects are dependencies that can be included in the application.

Starter projects have pre-defined dependencies and configurations for reach use-case.

To build spring REST service, we must take care of appropriate dependencies, JSON conversion, Tomcat integration, Unit tests.

It would be great if we can group all these under one single dependency. That’s what starter projects do. They provide convenient descriptors for different use-cases.

A Starter project can have other starter projects as dependencies.

|  |  |
| --- | --- |
| **Starter Project** | **Description** |
| spring-boot-starter | It is used for core starter, including auto-configuration support, logging, and YAML. |
| spring-boot-starter-web | It is used for building the web application, including RESTful applications using Spring MVC. It uses Tomcat as the default embedded container. |
| spring-boot-starter-test | It is used to test Spring Boot applications with libraries, including JUnit, Hamcrest, and Mockito |
| spring-boot-starter-jdbc | It is used for JDBC with the Tomcat JDBC connection pool. |
| spring-boot-starter-data-jpa | It is used for Spring Data JPA with Hibernate. |
| spring-boot-starter-actuator | It is used for Spring Boot's Actuator that provides production-ready features to help you monitor and manage your application. |
| spring-boot-starter-web-services | It is used for development of SOAP web services using Spring Web Services (Spring-WS). |

|  |  |
| --- | --- |
| **Use Case** | **Starter Project needed** |
| REST and web apps | spring-boot-starter-web  spring-boot-starter-test  spring-boot-starter-web  internally has dependencies for spring context, JSON conversion, Tomcat support,  Spring Web, Spring MVC  spring-web,  spring-webmvc,  spring-boot-starter-tomcat,  spring boot-starter-json |

**Spring Auto Configuration**

Spring Boot's auto-configuration is a powerful feature that automatically configures your Spring application based on the jar dependencies present in the classpath.

This eliminates the need for manual configuration of common components, leading to faster development and less boilerplate code.

**Example:**

Let's say you add the **spring-boot-starter-web** dependency to your project. Spring Boot will automatically: Configure a Tomcat web server, Setup a DispatcherServlet, Enable Spring MVC, JSON conversion and configure static resource handling.

You can override default auto-configuration behavior by setting properties in your application.properties or application.yml file.

If you have spring-boot-starter-tomcat on the classpath spring will automatically configures embedded tomcat with predefined configuration like 8080 for port etc. If you want to customize, define the property server.port on application.properties or application.yml file.

**Dev Tools**

Increase developer productivity.

Unit testing becomes easy.

Why to restart the server when code is changed.

Starter project for this is spring-starter-devtools. Adding this starter project will allow to modify code without restarting the server.

**Embedded Servers**

Traditional war deployment approach is

1. Install Java
2. Install server (Tomcat/ JBoss/WebSphere etc.)
3. Deploy the war on to the server.

A diagram of a computer program

Description automatically generated with medium confidence

In Embedded server model, server is part of the Jar. We just need to install java and run it.

1. Install Java
2. Run Jar file which has tomcat embedded.

**Actuator:**

Actuator is helpful in application monitoring and management in production.

Actuator provides number of end points for application monitoring

1. End point for list of spring beans in your app.
2. Application health information
3. Application metrics
4. Request mapping details.

spring-boot-starter-actuator is the starter project to be included to include the actuator in your project.

<http://localhost:8080/actuator>

<http://localhost:8080/actuator/health>

include below property to expose endpoints

management.endpoints.web.exposure.include=\*

<http://localhost:8080/actuator/beans>

<http://localhost:8080/actuator/configprops>

<http://localhost:8080/actuator/env>

<http://localhost:8080/actuator/metrics>

<http://localhost:8080/actuator/metrics/http.server.requests>

Instead of enabling all endpoints, we can selectively enable only for health and metrics like below

management.endpoints.web.exposure.include=health,metrics

**Web application in Spring Boot**

Building web application is complex and need to consider below

1. Front end tech like JSP, JSTL, Bootstrap, HTML, CSS etc.
2. Framework to use like Spring MVC
3. Security like Spring security
4. Backend integration like Database using JPA, Kafka etc.

Steps to create sample app

1. Create a sample maven web application. Archetype can be “maven-archetype-quickstart”.
2. Below directories will get created.

A screenshot of a computer

Description automatically generated

1. Add below entries in pom.xml and reload the project

A screenshot of a computer program

Description automatically generated

Spring-boot-starter-web internally has below dependencies

A screenshot of a computer program

Description automatically generated

**spring-boot-starter:** This will add support for spring-core, auto-configuration, logging and YAML

**spring-boot-starter-tomcat:** Starter for using Tomcat as the embedded servlet container. Default servlet container starter used by spring-boot-starter-web

**spring-webmvc:** Adds support for spring context, beans, aop, expression, web.

**Spring-boot-starter-json:** Adds support for JSON reading, writing and conversion.

4) Add below annotations to the main class

A screenshot of a computer program

Description automatically generated

SpringBootApplication annotation bootstraps spring boot application.

A computer screen shot of a code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Source code of SpringBootApplication class has annotations @ComponentScan that will scan all the classes within the package where the class is defined and @EnableAutoConfiguration will automatically configuration application based on dependencies added.