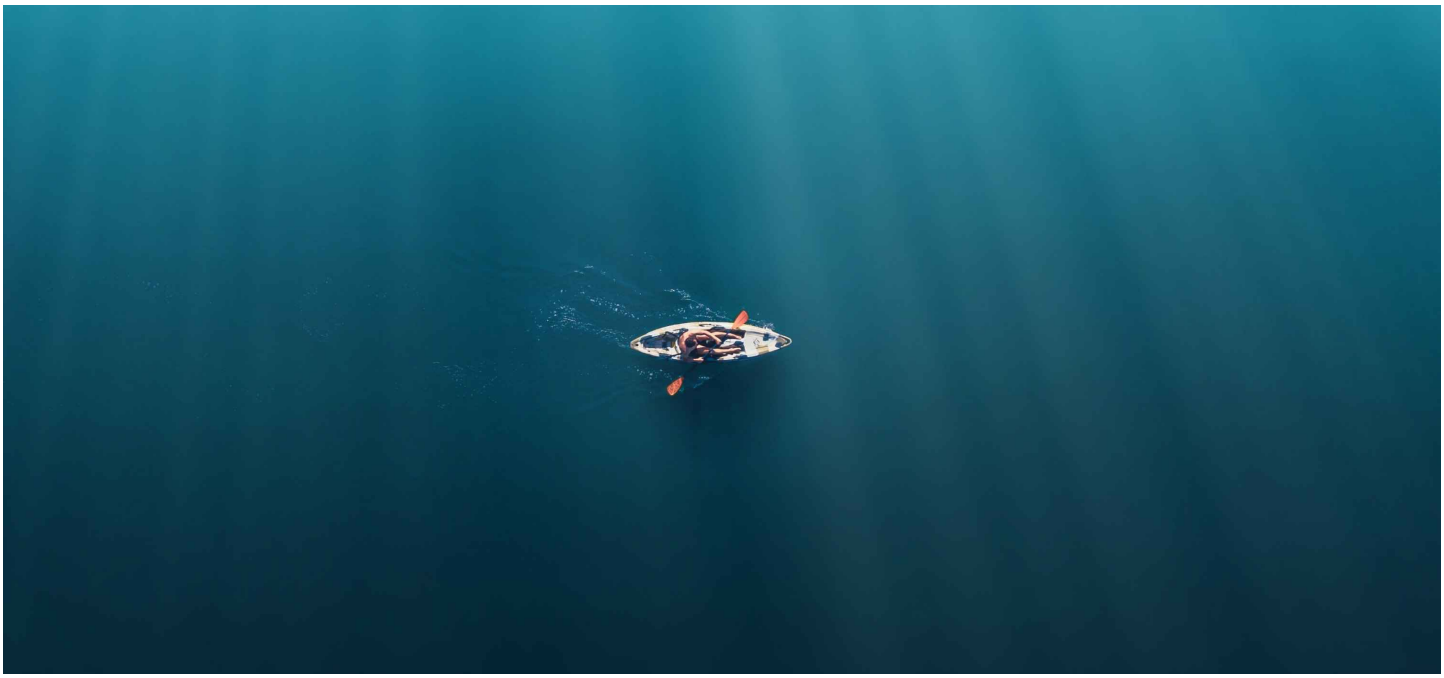


1-Spring Boot Basics

Author: Vincent Lau

Learning Objectives

> Describe what Spring Boot is and why it is used.
>
> Describe the difference between Spring Core and Spring Boot.
>
> Explain how Spring Boot bootstraps an application.
>
> Understand auto-configuration and component scanning.
>
> Describe why Spring Boot has easier packaging and deployment than Spring.

Overview

- Spring Boot is an open source Java-based framework used to create a Microservice. It is developed by Pivotal Team and is used to **build stand-alone** and **production-ready spring applications**.

- This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

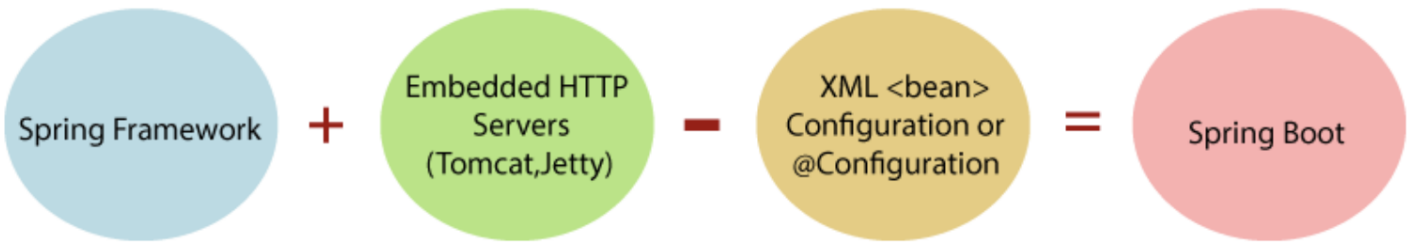
What is Spring Boot?

- `Spring Boot` is basically an **extension of the Spring framework**, which **eliminates** the **boilerplate configurations** required for setting up a Spring application.
- It takes an **opinionated view** of the Spring platform, which paves the way for a faster and more efficient development ecosystem(Opinionated software - Spring Boot is built and **designed in such a way that it makes it easy to do things in a certain way**).
- 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.

Goals of Spring Boot

- Spring Boot is designed with the following goals:
  - To minimize boilerplate code, like **Lombok/ Bean Configuration**
  - 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
- By achieving the above goals, Spring Boot **reduces development time, developer effort,** and **increases productivity.**

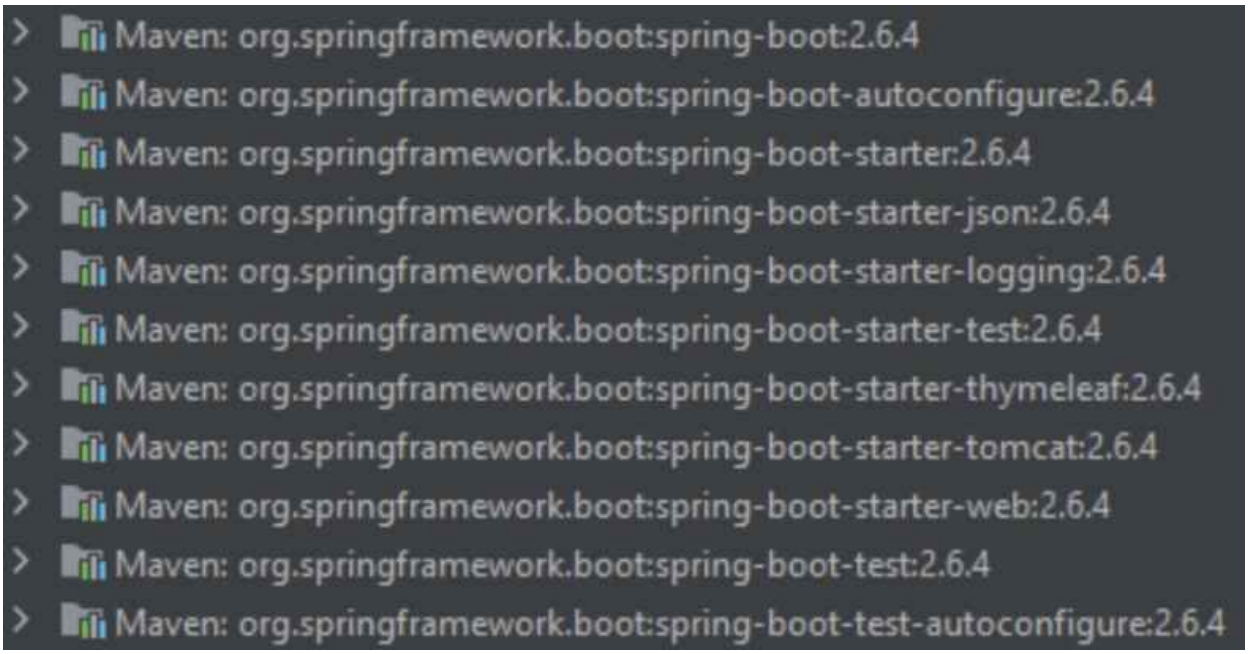
Why Spring Boot?



- Spring Boot is highly popular because of the features and benefits it offers:
  - It creates **standalone** Spring applications that can be started using `java -jar` .
  - It includes **embedded HTTP Servers such as Tomcat and Jetty** to avoid complexity in application deployment.

- It provides opinionated **starter packages** to simplify build configuration and ease dependency management.
- It provides **production-ready** features such as **metrics, health checks, and externalized configuration**.
- It enables **auto-configuration**; manual configurations are kept to a minimum.
- It offers **annotation-based** Spring application, avoiding the need for XML configuration.

Limitations of Spring Boot



- Spring Boot's starter packages can **include dependencies that are not used in the application**. These dependencies increase the size of the application. We can **exclude** these unused dependencies in maven if necessary.

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, different to Springboot, `Spring` still has the minimum dependencies required to create a web application:

```
<!-- Spring  -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>5.3.16</version>
```

```
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.16</version>
</dependency>
```

- Unlike Spring, **`Spring Boot`** requires only one dependency to get a web application up and running:

```
<!-- springboot -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.6.4</version>
</dependency>
```

- Another good example is testing libraries. In Springboot, we usually use Spring Test, **JUnit**, Hamcrest, and **Mockito** libraries. In a Spring project, we need to add all of these libraries as dependencies.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <version>2.6.4</version>
    <scope>test</scope>
</dependency>
```

- Spring Boot provides a number of **starter dependencies** for different Spring modules. Some of the most commonly used ones are:
  - *spring-boot-starter-data-jpa*
  - *spring-boot-starter-security*
  - *spring-boot-starter-test*
  - *spring-boot-starter-web*
  - *spring-boot-starter-thymeleaf*
- For the full list of starters, also check out the Spring documentation.

Bootstrapping an Application

- The entry point of a Spring Boot application is the class which is annotated with `@SpringBootApplication` :

```
1 @SpringBootApplication
2 public class Application {
3     public static void main(String[] args) {
4         SpringApplication.run(Application.class, args);
5     }
6 }
```

- By default, Spring Boot uses an **embedded container** to run the application.
- In this case, Spring Boot uses the `public static void main` entry point to launch an embedded web server.

@Configuration

- The `@Configuration` annotation sets a class as a Java configuration class that will replace XML configuration file.

@ComponentScan

- You need to add the **`@ComponentScan`** annotation to your class file to scan the components added to your project.
- Spring Boot application scans all the beans and package declarations when the application initializes.
- Snnotations like @Component, @Configuration, @Service, @Repository are specified on classes to mark them as Spring beans.
- The @ComponentScan annotation basically tells Spring Boot to scan the current package and its sub-packages in order to identify annotated classes and configure them as Spring beans.

@EnableAutoConfiguration

- Spring Boot auto-configuration runs at application startup. This process checks what is in the classpath and configure accordingly.

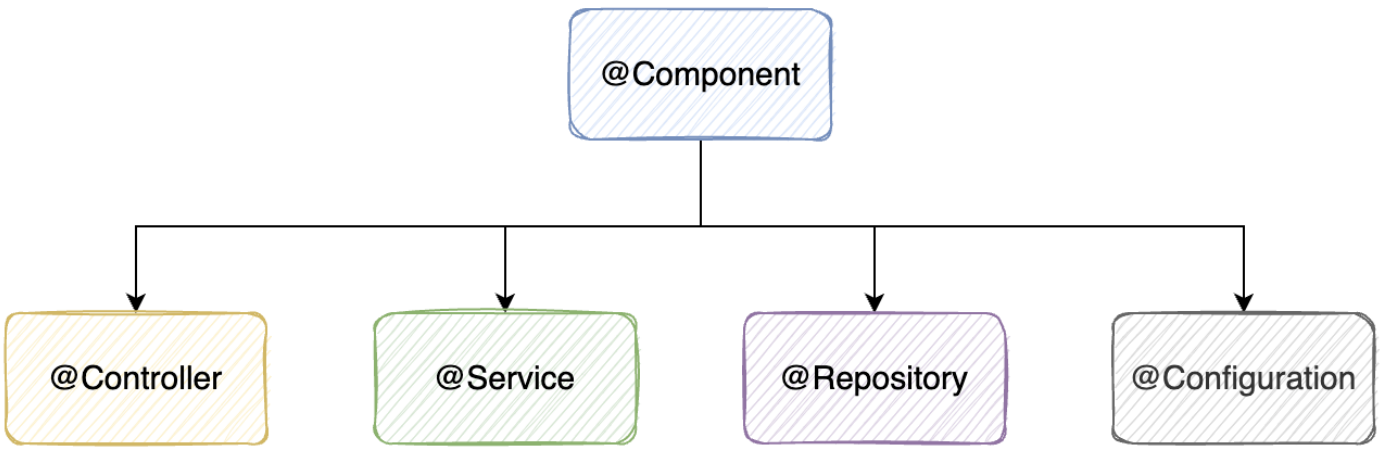
- Every time an application starts up, Spring Boot **checks the classpath covering areas** such as Web Functionality, Security and Persistence, etc.
- The main goal is to keep manual configuration to a minimum.
- For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.
- For this purpose, you need to add `@EnableAutoConfiguration` annotation to your main class file. Then, your Spring Boot application will be automatically configured.

@SpringBootApplication

- `@SpringBootApplication` encapsulates `@Configuration` , `@EnableAutoConfiguration` , **and** `@ComponentScan` annotations with their default attributes.
- If you add `@SpringBootApplication` annotation to the main class, you do not need to add the other three annotations.

For more information, read the Spring official documentation.

Core Spring Boot Annotations



@Component

- The `@Component` annotation is a class-level annotation.
- It **marks a Java class as a bean** during the process of component scanning enabled by `@ComponentScan` .
- Spring Boot **picks up the annotated class and configures it as a bean** in the application context during startup.

```java
@Component
public class EmailAddressValidator {
                ...
}

@Component
public class SomeBeanConfig {
                ...
}
```

@Controller

- `@Controller` is a **class-level** annotation. It is a specialization of `@Component` .
- It marks a class as a **web request handler**. It is often used to serve web pages, microservice architecture.
- It is mostly used with `@RequestMapping` annotation.

```java
@Controller
@RequestMapping("books")
public class BookController {

    @GetMapping(value = "/{bookId})
    //@RequestMapping(value = "/{bookId}", method = RequestMethod.GET)
    public Book getBookById(...) {
                        ...
    }
}
```

@Service

- It is also used at class level, and is a specialization of `@Component` .
- It tells Spring Boot that the annotated class contains **business logic**.

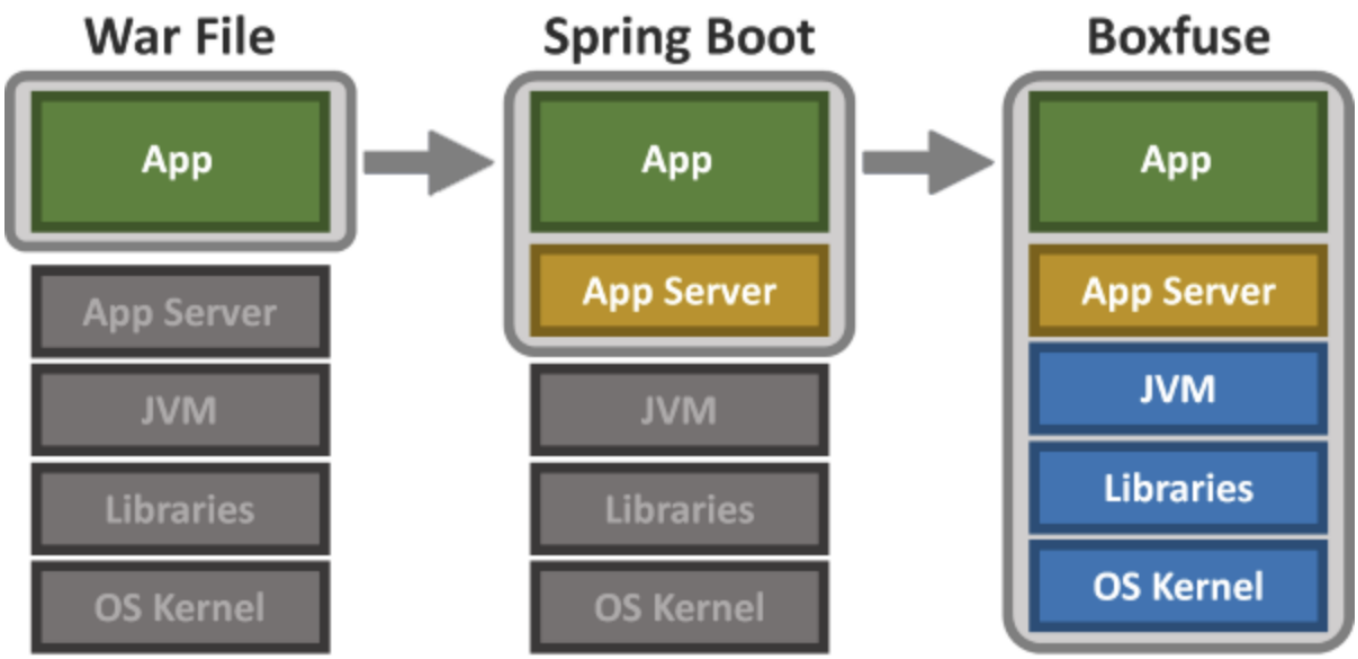
```java
@Service
public class EmailService {
    public void send(Email email) {
        ...
    }
}
```

@Repository (< Spring Version 3.0)

- A class-level annotation, as well as a specialization of `@Component` .
- A repository is a **Data Access Object (DAO)** that accesses the database directly. The repository does all the operations related to the database.

```
@Repository
public interface BookRepository extends JpaRepository<Book, Long> {
                    ...
}
```

Packaging and Deployment



- The Spring Boot Maven Plugin provides Spring Boot support in **`Maven`** , which allows packaging executable JAR (and WAR also, but forget it for spring boot first) and running an application "in-place".

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>getting-started</artifactId>
    <!-- ... -->
    <build>
        <plugins>
            <plugin>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

- Some of the advantages of Spring Boot over Spring in the context of deployment include:
  - Provides embedded container support
  - Provision to run the jars independently using the command `java -jar`
  - Option to exclude dependencies to avoid potential JAR conflicts when deploying in an external container
  - Option to specify active profiles when deploying
  - Random port generation for integration tests

Questions

- What is Spring Boot? Why Spring Boot?
- What are the key features of Spring Boot?
- How do the starter dependencies in Spring Boot ease dependency management?
- How does Spring Boot bootstrap an application?
- List the common Spring Boot annotations.
- How does Spring Boot make packaging and deployment easier?
- What are the differences between Spring and Spring Boot?