Spring Boot is a Java framework that makes it easier to create and run Java applications. It simplifies the configuration and setup process, allowing developers to focus more on writing code for their applications.

Spring Boot, a module of the Spring framework, facilitates **Rapid Application Development**(RAD) capabilities.

**What is Spring Boot?**

**Spring Boot**is an open-source Java framework used to create a Micro Service. Spring boot is developed by Pivotal Team, and it provides a faster way to set up and an easier, configure, and run both simple and web-based applications. It is a combination of Spring Framework and Embedded Servers. The main goal of Spring Boot is to reduce development, unit test, and integration test time and in Spring Boot, there is no requirement for XML configuration.

## Spring Boot – Basics

# Introduction to Spring Boot

**Spring** is widely used for creating scalable applications. For web applications, Spring provides **Spring MVC**, a commonly used module for building robust web applications. The major drawback of traditional Spring projects is that configuration can be time-consuming and overwhelming for new developers. Making an application production-ready requires significant effort.

**The solution to this is “Spring Boot”.** Spring Boot is built on top of the [Spring Framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/) and includes all Spring features while simplifying configuration. It has become a developer favorite due to its **rapid development** capabilities, allowing developers to **focus on business logic** instead of struggling with configurations.

Spring Boot is a**microservice-based framework** that enables **quick development** of production-ready applications.

### Features of Spring Boot

Spring Boot is built on top of the conventional Spring framework, providing all the features of Spring while being significantly easier to use. Here are its key features:

**1.** **Auto-Configuration:**Spring Boot **eliminates the need for heavy XML configuration**, which is common in traditional [Spring MVC](https://www.geeksforgeeks.org/spring-mvc-framework/) projects. Instead, everything is auto-configured. Developers only need to add the appropriate configuration to utilize specific functionalities. For example, if we want to use [Hibernate (ORM)](https://www.geeksforgeeks.org/introduction-to-hibernate-framework/), we can simply add the **@Table** annotation to our model/entity class and the**@Column** annotation to map it to database tables and columns.

**2. Easy Maintenance and Creation of REST Endpoints:** Creating [REST APIs](https://www.geeksforgeeks.org/rest-api-introduction/) is incredibly easy in Spring Boot. With annotations like **@RestController** and **@RequestMapping**, developers can quickly define endpoints. By 2025, Spring Boot has introduced even more advanced annotations like **@GetMapping, @PostMapping, @PutMapping,** and **@DeleteMapping**, making REST API development more intuitive and efficient.

**For example:**

@RestController

@RequestMapping("/api")

**public** **class** **MyController** {

@GetMapping("/hello")

**public** String sayHello() {

**return** "Hello, World!";

}

}

**3. Embedded Tomcat Server:**Unlike traditional Spring MVC projects, where you need to manually install and configure a Tomcat server, Spring Boot comes with an [embedded Tomcat server](https://www.geeksforgeeks.org/embedding-tomcat-server-in-maven-project/). This allows applications to be hosted directly without additional setup. By 2025, Spring Boot also supports other embedded servers like Jetty and Undertow, giving developers more flexibility based on their application requirements.

**4. Easy Deployment:** Spring Boot simplifies deployment by allowing applications to be packaged as JAR or WAR files. These files can be directly deployed to a Tomcat server or cloud environment. By 2025, Spring Boot has enhanced its deployment capabilities with seamless integration into Kubernetes and Docker, making it easier to deploy and scale applications in cloud-native environments.

**5. Microservice-Based Architecture**: Spring Boot is designed for microservices, which are small, independent modules that focus on a single functionality. For example, in a hospital management system, you might have separate services for patient registration, billing, and database management. By 2025, Spring Boot has further optimized its microservice support with features. In a monolithic system, all features are bundled into a single codebase, making it difficult to maintain and scale. In a microservice-based system, each feature is divided into smaller, independent services. This modular approach makes the system easier to maintain, debug, and deploy. Each service can be built using different technologies suited to its specific requirements.

* **Reactive Programming:** For building non-blocking, scalable applications.
* **Distributed Tracing**: For monitoring and debugging microservices.
* **Service Mesh Integration:** For better communication between microservices.

## Spring Boot Architecture

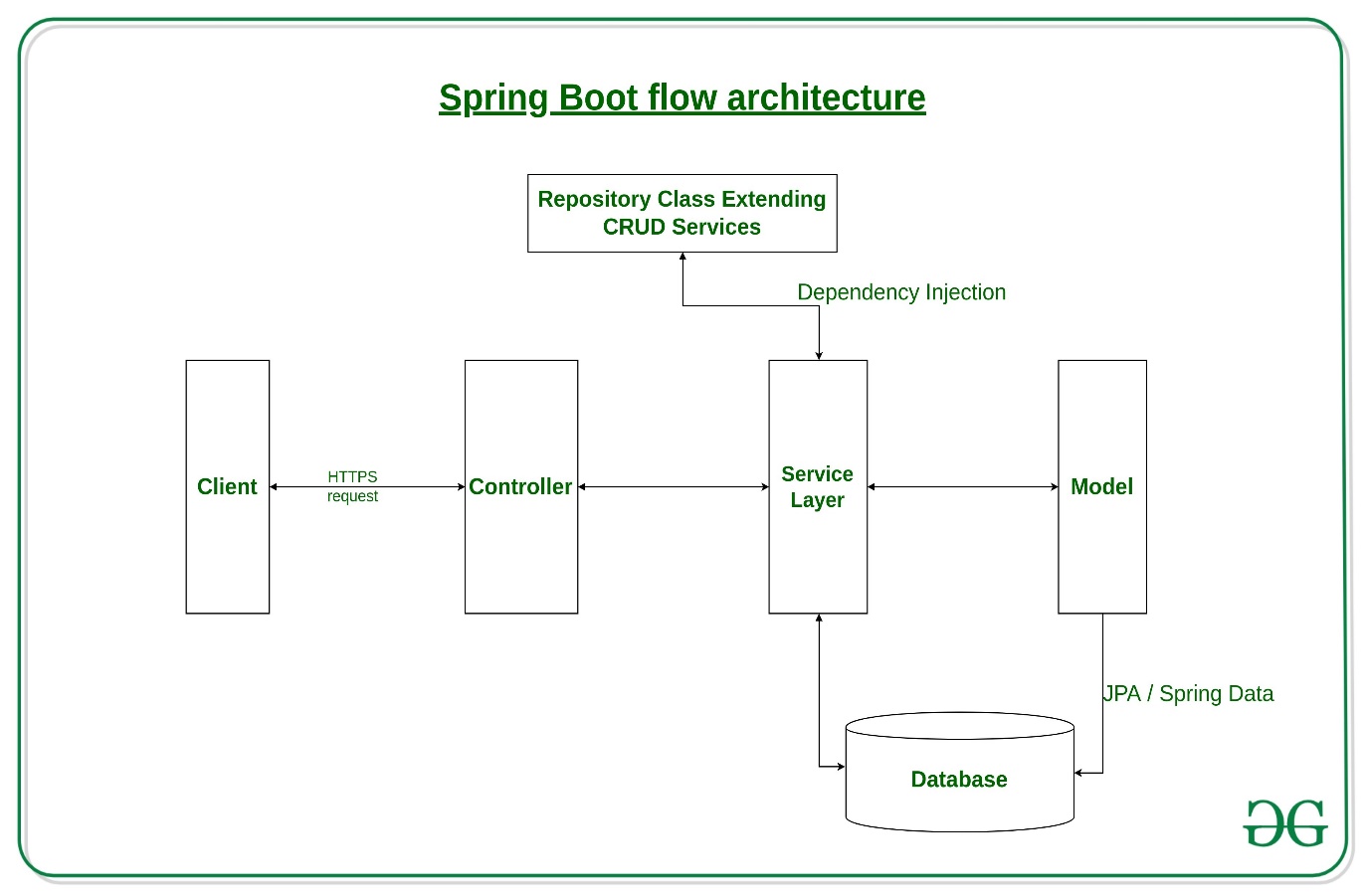
To understand the architecture of Spring Boot, let’s examine its different layers and components.

### Layers in Spring Boot

Spring Boot follows a layered architecture with the following key layers:

* **Client Layer:**
  + This represents the external system or user that interacts with the application by sending HTTPS requests.
* **Controller Layer (Presentation Layer):**
  + Handles incoming HTTP requests from the client.
  + Processes the request and sends a response.
  + Delegates business logic processing to the Service Layer.
* **Service Layer (Business Logic Layer)**:
  + Contains business logic and service classes.
  + Communicates with the Repository Layer to fetch or update data.
  + Uses Dependency Injection to get required repository services.
* **Repository Layer (Data Access Layer)**:
  + Handles CRUD (Create, Read, Update, Delete) operations on the database.
  + Extends Spring Data JPA or other persistence mechanisms.
* **Model Layer (Entity Layer):**
  + Represents database entities and domain models.
  + Maps to tables in the database using JPA/Spring Data.
* **Database Layer:**
  + The actual database that stores application data.
  + Spring Boot interacts with it through JPA/Spring Data.

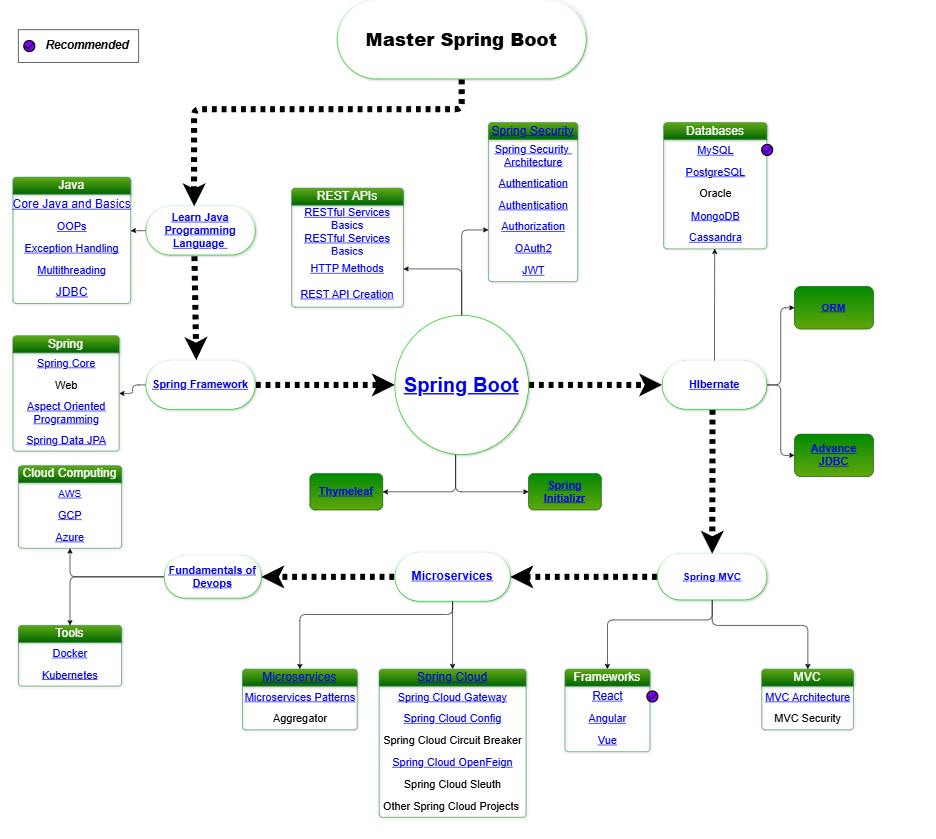
### Spring Boot Flow Architecture



### Request Flow in Spring Boot

* A Client makes an HTTPS request (GET/POST/PUT/DELETE).
* The request is handled by the Controller, which is mapped to the corresponding route.
* If business logic is required, the Controller calls the Service Layer.
* The Service Layer processes the logic and interacts with the Repository Layer to retrieve or modify data in the Database.
* The data is mapped using JPA with the corresponding Model/Entity class.
* The response is sent back to the client. If using Spring MVC with JSP, a JSP page may be returned as the response if no errors occur.

## A Roadmap to Learn



# Difference between Spring and Spring Boot

**Spring**

[**Spring**](https://www.geeksforgeeks.org/introduction-to-spring-framework/) is an open-source lightweight framework that allows Java developers to build simple, reliable, and scalable enterprise applications. This framework mainly focuses on providing various ways to help you manage your business objects. It made the development of Web applications much easier compared to classic Java frameworks and Application Programming Interfaces (APIs), such as Java database connectivity (JDBC), JavaServer Pages (JSP), and Java Servlet. This framework uses various new techniques such as Aspect-Oriented Programming (AOP), Plain Old Java Object (POJO), and dependency injection (DI), to develop enterprise applications. The **Spring framework** can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP. Spring Object-Relational Mapping (Spring ORM). Spring Web Flow, and Spring Web MVC. You can use any of these modules separately while constructing a Web application. The modules may also be grouped to provide better functionalities in a Web application.

**Spring Boot**

[**Spring Boot**](https://www.geeksforgeeks.org/introduction-to-spring-boot/) is built on top of the conventional spring framework. So, it provides all the features of spring and is easier to use than spring. Spring Boot is a microservice-based framework and makes a production-ready application in very less time. In Spring Boot everything is auto configured. We just need to use proper configuration for utilizing a particular functionality. Spring Boot is very useful if we want to develop REST API.

**Why Spring Boot over Spring?**

Let us know, if Spring was solving all the problems, then what is the need for Spring Boot at all?

The reason why we need Spring Boot is we are changing or shifting towards applications like microservices and with microservices, one of the most important thing aim is we would want to be able to develop applications very quickly. So instead of building one large application, we would like to build ten small microservices, which have their own scope and their own capabilities. Spring-based applications have lots of configurations. It can be of **XML configuration**, Java configuration or annotations, etc. For example, If we want to use Spring MVC, we need to use **@ComponentScan** annotation, **Dispatcher servlet**, **view resolver**, **web jars,** etc. This kind of configuration makes it slow to develop an application. So, in this place, **Spring Boot Autoconfiguration** comes in. It looks at what types of frameworks are available at the classpath and it looks at what configurations are provided by the programmers or what configurations are provided already for the application. It will look at both of them. Data is not configured but there is hibernation on the classpath, so it will configure the data source automatically. It will configure the **in-memory**database, it will configure the dispatcher servlet automatically. This is called autoconfiguration. Spring Boot creates a starter project by which all the XML configurations and dependencies get by default.

**Difference between Spring and Spring Boot**

| **Spring** | **Spring Boot** |
| --- | --- |
| Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| The most important feature of the Spring Framework is dependency injection. | The most important feature of the Spring Boot is Autoconfiguration. |
| It helps to create a loosely coupled application. | It helps to create a stand-alone application. |
| To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc. |
| To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |
| Developers need to write boilerplate code for smaller tasks. | In Spring Boot, there is reduction in boilerplate code. |
| Developers have to define dependencies manually in the pom.xml file. | pom.xml file internally handles the required dependencies. |

**Conclusion**

Basically, there is no such comparison between Spring and Spring Boot, they solve their own problems. The comparison does not really matter because Spring solves it’s own problems of dependency injection and Spring Boot solves the problem of getting applications like Rapid development. At last, they have their own problem spaces.

# Difference between Spring MVC and Spring Boot

**1.**[**Spring MVC**](https://www.geeksforgeeks.org/spring-mvc-using-java-based-configuration/)**:** Spring is widely used for creating scalable applications. For web applications Spring provides Spring MVC framework which is a widely used module of spring which is used to create scalable web applications. Spring MVC framework enables the separation of modules namely Model View, Controller, and seamlessly handles the application integration. This enables the developer to create complex applications also using plain java classes. The model object can be passed between view and controller using maps.

* **Model –** A model can be an object or collection of objects which basically contains the data of the application.
* **View –** A view is used for displaying the information to the user in a specific format. Spring supports various technologies like **freemarker**, **velocity**, and **thymeleaf**.
* **Controller –** It contains the logical part of the application. *@Controller* annotation is used to mark that class as a controller.
* **Front Controller –** It remains responsible for managing the flow of the web application. Dispatcher Servlet acts as a front controller in Spring MVC.

**2.**[**Spring Boot**](https://www.geeksforgeeks.org/introduction-to-spring-boot/)**:** Spring Boot is built on top of the conventional spring framework. So, it provides all the features of spring and is yet easier to use than spring. Spring Boot is a microservice-based framework and making a production-ready application in very less time. In Spring Boot everything is auto-configured. We just need to use proper configuration for utilizing a particular functionality. Spring Boot is very useful if we want to develop REST API. Spring Boot provides the facility to convert our project into war or jar files. Also, the instance of Tomcat can be run on the cloud as well. There are four main layers in Spring Boot :

* **Presentation Layer –** As the name suggests, it consists of views (i.e. frontend part).
* **Data Access Layer –** CRUD (create, retrieve, update, delete) operations on the database comes under this category.
* **Service Layer –** This consists of service classes and uses services provided by data access layers.
* **Integration Layer –** It consists of web different web services(any service available over the internet and uses [XML](https://www.geeksforgeeks.org/html-and-xml-gq/) messaging system).

**Difference between Spring MVC and Spring Boot :**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **SPRING MVC** | **SPRING BOOT** |
| **1.** | **Spring MVC is a Model View, and Controller based web framework widely used to develop web applications.** | **Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.** |
| **2.** | **If we are using Spring MVC, we need to build the configuration manually.** | **If we are using Spring Boot, there is no need to build the configuration manually.** |
| **3.** | **In the Spring MVC, a deployment descriptor is required.** | **In the Spring Boot, there is no need for a deployment descriptor.** |
| **4.** | **Spring MVC specifies each dependency separately.** | **It wraps the dependencies together in a single unit.** |
| **5.** | **Spring MVC framework consists of four components : Model, View, Controller, and Front Controller.** | **There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer.** |
| **6.** | **It takes more time in development.** | **It reduces development time and increases productivity.** |
| **7.** | **Spring MVC do not provide powerful batch processing.** | **Powerful batch processing is provided by Spring Boot.** |
| **8.** | **Ready to use feature are provided by it for building web applications.** | **Default configurations are provided by it for building a Spring powered framework.** |

## Prerequisite (Spring Core Concepts)