Comparison Table

Spring Framework vs Spring Boot vs Spring MVC

1. Spring Framework

- **What it is:**
  A comprehensive, modular Java framework for building enterprise applications.
- **Key Features:**
  - Core concepts: **IoC (Inversion of Control)** & **DI (Dependency Injection)**
  - Supports multiple modules: AOP, JDBC, ORM, Security, Messaging, etc.
  - Not limited to web development — can be used for desktop, batch, and integration apps.
- **Use Case:**
  You want **full control** over configuration and choose modules manually.

---

2. Spring MVC

- **What it is:**
  A **web module** within the Spring Framework for building **Model-View-Controller** web applications.
- **Key Features:**
  - Handles HTTP requests & responses
  - Supports REST API creation
  - Works with JSP, Thymeleaf, FreeMarker, etc. for the View layer
- **Use Case:**
  You need a **web application** or REST API inside the Spring ecosystem, but still want to configure things manually.

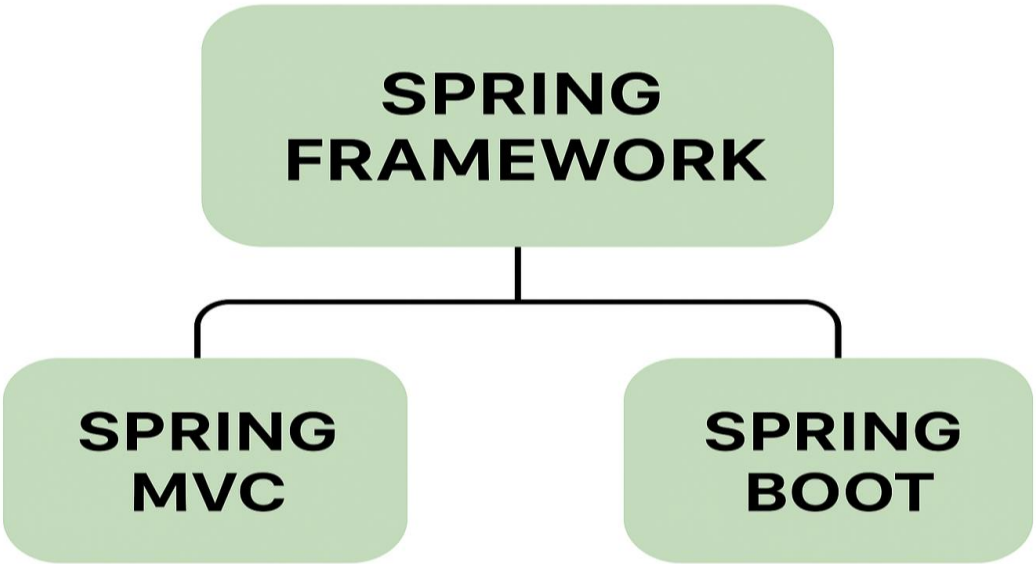
---

3. Spring Boot

- **What it is:**
  An **extension of Spring Framework** that simplifies configuration and development.
- **Key Features:**
  - **Auto-configuration** (no need for boilerplate XML)
  - **Embedded servers** (Tomcat, Jetty) — no separate deployment needed
  - Pre-built **starter dependencies** for faster setup
  - Production-ready features (Actuator, health checks)
- **Use Case:**
  You want to **quickly build** Spring applications (including MVC & REST APIs) with **minimal setup**.

Quick Analogy

- **Spring Framework** → The full kitchen with all tools and appliances.
- **Spring MVC** → A specific cooking station in that kitchen (web app cooking).
- **Spring Boot** → A smart kitchen where everything is pre-set and ready to start cooking instantly.



Aspect	Spring Framework	Spring Boot	Spring MVC
Purpose	A complete framework for building enterprise Java apps	Makes Java development and deployment faster and easier	Focuses on creating web applications using the MVC pattern
Configuration	Requires a lot of manual setup (XML or Java-based)	Minimal setup; uses smart defaults	Moderate setup using XML or annotations
Flexibility	Very flexible and customizable	Uses default settings but allows customization	Flexible, tailored for web apps
Main Use Cases	Complex, large-scale business applications	Microservices, standalone apps, quick development	Structured web applications with MVC design
Learning Curve	Steep – requires deep understanding of many components	Easy to start – beginner friendly	Moderate – easier for those familiar with web dev
Boilerplate Code	More boilerplate due to detailed setup	Less boilerplate – follows conventions	Moderate – some setup for controllers and views
Development Speed	Slower due to extensive configuration	Fast – comes with sensible defaults	Moderate – faster than Spring Core but slower than Boot