# Spring vs Spring Boot – What's the Real Difference?

If you're diving into Java backend development, you've definitely come across **Spring** and **Spring Boot**. While they sound similar, their roles in development are quite different — and understanding those differences is key to writing better applications.

# What is Spring Framework?

Spring is a **comprehensive framework for building enterprise-level Java applications**. It was introduced in **2003** as a solution to the heavy and complex Java EE (J2EE) programming model.

## Core Concepts of Spring:

- Dependency Injection (DI)
   Spring encourages loose coupling by injecting required dependencies instead of creating them directly in the code.

   This helps in writing cleaner, testable, and modular code.
- Transaction Management
   Managing transactions in Spring is declarative. Just add
   @Transactional, and Spring ensures your method executes in a transaction-safe way.

• Spring MVC (Model-View-Controller)

A part of the framework that helps build clean web apps using controllers, models, and views. You write controllers to handle web requests and return data.

# Problem?

Spring required a **lot of setup**:

- Extensive **XML** configuration
- Manual bean definitions
- Explicit server configuration

It was flexible but not beginner-friendly.

# What is Spring Boot?

Spring Boot was released in **2014** to solve the pain points of the Spring Framework. It's a **toolset built on top of Spring** that aims to simplify the entire development process.

### Why was Spring Boot Introduced?

Imagine building a REST API in Spring:

- You'd need to configure the server.
- Add Jackson for JSON.
- Set up controller beans manually.
- Manage dependencies by hand.

Spring Boot solved all of this with convention over configuration.

# **Key Features of Spring Boot:**

## Auto-Configuration

Spring Boot automatically configures your application based on the dependencies you include in your project. For example, if you include Spring Web, it sets up a web server.

## • Starter Dependencies

Want to build a REST API? Just add spring-boot-starter-web.

Need JPA? Add spring-boot-starter-data-jpa.

These bundles include all related libraries so you don't have to manage them individually.

#### • Embedded Servers

No need to deploy your app to an external Tomcat server — it comes embedded . Just run java -jar yourapp.jar and it's live!

#### • Minimal Boilerplate

No XML configs, fewer annotations, faster development.

#### Actuator & Metrics

Get detailed application health, metrics, and endpoints with spring-boot-starter-actuator.

# Spring vs Spring Boot:

## Spring Framework:

- Requires **manual configuration** using XML or Java-based annotations.
- You need to manage all dependencies individually.

- Needs an external web server like Tomcat to deploy and run your app.
- More **boilerplate code** to get even a simple app running.
- Testing setups can be tedious and repetitive.
- Ideal for large, highly customized enterprise projects.
- More **flexibility and control**, but with a steeper learning curve.

## Spring Boot:

- Comes with **auto-configuration** detects what you need and configures it.
- Uses **starter dependencies** to group and manage libraries for specific use cases.
- Ships with **embedded servers** like Tomcat or Jetty no external deployment needed.
- Remove boilerplate start coding business logic right away.
- Offers **Spring Boot Starter Test** and simplified test setup.
- Optimized for building **REST APIs, microservices**, and cloud-native apps.
- Follows "convention over configuration", making it more beginner-friendly.
- Perfect for rapid development, MVPs, and startups.

## **When to Use Which?**

## Use Spring Framework if:

- You need **complete control** over every part of the system.
- You're working on a large, highly customized enterprise project.

## **Use Spring Boot** if:

- You want to get started quickly.
- You're building **microservices**, REST APIs, or cloud-native applications.
- You want fewer moving parts and rapid development.

# Analogy That Clicks:

- Spring = Building a car from scratch.
- Spring Boot = Getting a Tesla pre-built, elegant, and ready to drive.

# Problems Spring Boot Solved :

- Reduced boilerplate code
- Eliminated the need for manual XML configuration
- No need to set up and deploy on external servers

- Easier dependency management
- Production-ready features out of the box (via Actuator)
- Faster development cycle with **Spring Initializr**
- Unified approach with starter templates
- Great for microservice architectures

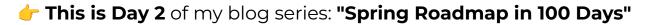
# Summary

Spring is powerful.

Spring Boot is powerful and simplified.

Spring Boot didn't replace Spring.

It made Spring accessible, productive, and ready for the modern world of microservices and cloud-native apps.



Missed Day 1? I covered why Spring Boot came into existence, go check it out.

🚨 Up Next — **Day 3**:

What are Spring Boot Starters and How Do They Make Life Easy?

Share this with a Java dev who's still stuck writing XML configs Save this if you want to build Spring Boot apps the smart way.

📌 Follow me to get all 100 Days of Spring posts! 🌼 🌷 🌸 🌺 🌼