# **🌱 Spring Boot: Introduction**

**🔹 What is Spring Boot?**

**Spring Boot** is a framework built on top of the **Spring Framework** that makes it easy to create **standalone, production-grade Spring-based applications** with minimal configuration.

It provides:

* **Auto-configuration** (automatically configures Spring beans based on classpath and settings)
* **Embedded servers** (like Tomcat, Jetty, Undertow)
* **Opinionated defaults** (sensible defaults for project setup)
* **No need for XML configurations** (mostly annotation and properties-based)

**📜 Why Spring Boot Came?**

**Before Spring Boot:**

* Developers had to do **manual setup** of XML/Java configuration, server setup, dependency management, servlet configuration, etc.
* High **boilerplate code** and complex configuration.

**Spring Boot Solved This By:**

* Auto-configuring components.
* Bundling embedded servers.
* Reducing boilerplate code.
* Faster development and deployment (dev → test → prod pipeline).

**📌 When and Where to Use Spring Boot?**

| **Use Case** | **When to Use** |
| --- | --- |
| Building Microservices | Simplifies creation and deployment. |
| REST APIs | Quick setup with auto-configuration and Spring MVC. |
| Batch processing apps | Easy integration with Spring Batch. |
| Cloud-native apps | Works well with Spring Cloud, Docker, Kubernetes. |
| Rapid Prototyping | Start with minimal code and configuration. |

**✅ Rules/Features of Spring Boot**

| **Feature** | **Description** |
| --- | --- |
| @SpringBootApplication | Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. |
| Embedded Servers | No need to deploy WARs, just run .jar. |
| Starter Dependencies | Preconfigured dependencies (e.g., spring-boot-starter-web). |
| Application Properties | Use application.properties or application.yml for easy config. |
| Actuator | For monitoring and managing the application. |

**🌍 Real World Examples of Spring Boot**

**✅ Example 1: REST API for E-commerce App**

* Use Spring Boot to expose product catalog via /api/products.
* Auto-configures Tomcat, connects to MySQL, exposes REST controllers.

**✅ Example 2: Microservice for Payment Gateway**

* Independent payment service using Spring Boot.
* Registers with service registry (e.g., Eureka), communicates with Order Service.

**✅ Example 3: Real-Time Notification System**

* Spring Boot app using WebSocket for live notifications.
* Easy integration with messaging systems like Kafka or RabbitMQ.

## **🌐 Intro to Spring Boot Web (Servlet) Stack**

**🔹 What is Spring Boot Web Stack?**

It’s the **Servlet-based stack** of Spring Boot built using **Spring MVC**, used to create **web applications and RESTful APIs**.

Uses:

* Embedded **Servlet containers** (Tomcat by default).
* **DispatcherServlet** for routing HTTP requests.
* **Controllers** to handle HTTP methods (GET, POST, etc.).

**📜 Why it Came?**

**Before Spring Boot Web:**

* Required manually deploying WAR files on external servers.
* Configuration-heavy servlet and web.xml setups.
* Complicated routing and bean management.

**Spring Boot Web Stack Simplifies**:

* Running web applications with a single command.
* Auto-configuring MVC components.
* Eliminating XML-based servlet configurations.

**📌 When and Where to Use It?**

| **Use Case** | **When to Use** |
| --- | --- |
| REST APIs | Create RESTful web services. |
| Traditional MVC Web Apps | When using JSP, Thymeleaf, etc. |
| Form Handling | When interacting with forms, validations. |
| Content Serving | Serving HTML, static files (images, CSS). |
| Handling HTTP Requests | GET, POST, PUT, DELETE logic. |

**✅ Rules/Components in Spring Boot Web Stack**

| **Component** | **Description** |
| --- | --- |
| @RestController / @Controller | Handle incoming web requests. |
| @RequestMapping | Maps HTTP methods to Java methods. |
| DispatcherServlet | Central servlet that dispatches requests to controllers. |
| spring-boot-starter-web | Brings in Spring MVC, Jackson, embedded Tomcat, etc. |
| Static Resources | src/main/resources/static for HTML, CSS, JS. |
| Templates | Use Thymeleaf, FreeMarker, or JSP under src/main/resources/templates. |

**🌍 Real World Examples of Spring Boot Web Stack**

**✅ Example 1: RESTful Inventory Service**

* Exposes endpoints like GET /inventory/{id}, POST /inventory.
* Uses @RestController, @RequestMapping, and JSON conversion via Jackson.

**✅ Example 2: Job Application Form Portal**

* Uses @Controller and Thymeleaf to render job forms.
* Handles POST form submission and validations.

**✅ Example 3: Admin Dashboard**

* Uses MVC structure: Controllers + Views (Thymeleaf) + CSS/JS.
* Accesses and displays data from a database via Spring Data JPA.

**🧩 Summary Table**

| **Feature** | **Spring Boot** | **Spring Boot Web Stack** |
| --- | --- | --- |
| Focus | App bootstrapping & config simplification | Web development with MVC/REST |
| Key Dependency | spring-boot-starter | spring-boot-starter-web |
| Handles | Autoconfig, Dependency Mgmt | HTTP routing, Controllers, REST APIs |
| Use Case | Any Spring app (web, batch, cloud) | Web and RESTful APIs |

**Advantages of Spring Boot:**

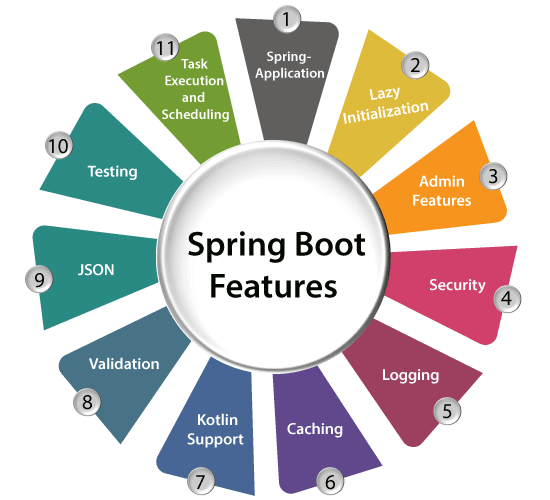
* It creates stand-alone Spring applications that can be started using Java -jar.
* It tests web applications easily with the help of different Embedded HTTP servers such as Tomcat, Jetty, etc. We don’t need to deploy WAR files.
* It provides opinionated ‘starter’ POMs to simplify our Maven configuration.
* It provides production-ready features such as metrics, health checks, and externalized configuration.
* There is no requirement for XML configuration.
* It offers a CLI tool for developing and testing the Spring Boot application.
* It offers the number of plug-ins.
* It also minimizes writing multiple boilerplate codes (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
* It increases productivity and reduces development time.

**Limitations of Spring Boot:**

Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

**Spring Boot Features:**

* Web Development
* SpringApplication
* Application events and listeners
* Admin features
* Externalized Configuration
* Properties Files
* YAML Support
* Type-safe Configuration
* Logging
* Security



## **1. Spring Stereotype Annotations**

**🔹 What & Why**

Spring stereotype annotations are meta-annotations that mark a class as a Spring-managed bean **and** convey its role in your application’s architecture. They enable component-scanning to automatically detect and register beans without XML.

| **Annotation** | **Layer/Role** | **Meta-annotation** |
| --- | --- | --- |
| @Component | Generic stereotype | @Component |
| @Service | Service/business logic | @Component |
| @Repository | Persistence/DAO layer | @Component + exception translation |
| @Controller | MVC web controller | @Component + web MVC |
| @RestController | REST endpoints | @Controller + @ResponseBody |
| @Configuration | Bean definitions/config | @Component + config |

**Origins:** Introduced to reduce XML boilerplate and let Spring auto-detect beans via classpath scanning. Stereotypes also embed semantic metadata (e.g. @Repository turns JPA exceptions into Spring’s DataAccessException).

**📌 Rules & Best Practices**

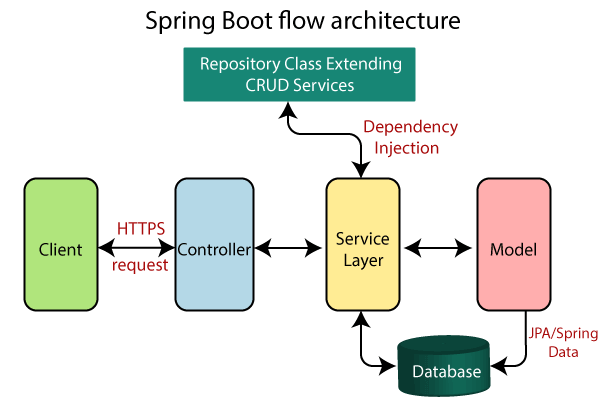
1. **Only one stereotype per class.** Don’t combine, e.g., @Service @Controller on same type.
2. **Layer alignment:**
   * @Repository on DAO classes
   * @Service on business-logic or domain services
   * @Controller/@RestController on web layer
3. **Exception translation:** @Repository enables Spring’s exception-translation AOP for persistence frameworks.
4. **Scanning config:** Component scanning picks up only stereotype-annotated classes (or those explicitly included).
5. **Scope defaults:** All stereotypes default to singleton scope (one instance per Spring context).

**🌍 When & Where to Use**

| **Layer** | **Annotation** | **When to Use** |
| --- | --- | --- |
| Persistence | @Repository | DAO/Repository classes that interact with a database |
| Service | @Service | Classes implementing business or domain logic |
| Web MVC | @Controller | Spring MVC controllers that return views |
| REST API | @RestController | Controllers producing JSON/XML responses |
| Config | @Configuration | Classes defining @Bean methods and config props |

**✅ Real-World Examples**

1. **E-Commerce Order Processing**
   * @Repository class OrderRepository uses Spring Data JPA
   * @Service class OrderService implements order validation, payment orchestration
   * @RestController class OrderController exposes /api/orders endpoints
2. **Banking Transaction System**
   * @Repository class AccountDao for JDBC calls
   * @Service class TransactionService to apply business rules (fraud checks, balance updates)
   * @Controller class TransactionWebController for web-based admin UI
3. **IoT Sensor Data Pipeline**
   * @Repository class SensorDataStore writes to Cassandra/MongoDB
   * @Service class AggregationService computes rolling averages, alerts
   * @RestController class SensorApiController for device registration and health checks



**SPRING-BOOT-MICROSERVICES Annotations:**

**No beans.xml**

* **@Required:** It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception BeanInitilizationException.
* **@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.
* **@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.
* **@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.
* **@Bean:** It is a method-level annotation. It is an alternative of XML tag. It tells the method to produce a bean to be managed by Spring Container.
* **@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.
* **@Controller:** The @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. By default, it returns a string that indicates which route to redirect. It is mostly used with @RequestMapping annotation.
* **@Service:** It is also used at class level. It tells the Spring that class contains the business logic.
* **@Repository:** It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.
* **@SpringBootApplication:** It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

**Spring MVC and REST Annotations:**

* **@RequestMapping:** It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.
* **@GetMapping:** It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)
* **@PostMapping:** It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)
* **@PutMapping**: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)
* **@DeleteMapping:** It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)
* **@PatchMapping:** It maps the HTTP PATCH requests on the specific handler method. It is used instead of using: @RequestMapping(method = RequestMethod.PATCH)
* **@RequestBody:** It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.
* **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
* **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
* **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
* **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
* **@RestController:** It can be considered as a combination of @Controller and @ResponseBody annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
* **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

## **2. Spring Boot First Application Using Spring Initializr**

**🔹 What & Why**

**Spring Initializr** is an online (or IDE-integrated) project generator that scaffolds a Spring Boot application with your chosen metadata (group, artifact, dependencies). It exists to eliminate manual setup of build files, directory structure, and basic configuration—letting you **start coding immediately**.

**📌 Rules & Steps**

1. **Access Initializr**:
   * Web: <https://start.spring.io>
   * IDE: built-in support (IntelliJ, VS Code, Eclipse)
2. **Choose Metadata**:
   * Project (Maven/Gradle), language, Spring Boot version
   * Group / Artifact, packaging (Jar/War), Java version
3. **Select Dependencies**: Starter POMs (e.g. *Spring Web*, *Spring Data JPA*, *Spring Security*)
4. **Generate & Unpack**: Download .zip → extract into your workspace
5. **Examine Main Class**:

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

1. **Run** with mvn spring-boot:run or your IDE’s run configuration.

**🌍 When & Where to Use**

* **Any new Spring Boot project**: microservice, REST API, batch job, CLI tool.
* **Prototyping**: Spin up proof-of-concept quickly.
* **Workshops & Tutorials**: Ensures everyone starts from a common, correct baseline.

**✅ Real-World Examples**

1. **Customer Service Microservice**
   * Dependencies: *Spring Web*, *Spring Data MongoDB*
   * Scaffolded endpoints: GET /customers, POST /customers
   * Runs on embedded Tomcat with minimal code
2. **Invoice Generation CLI App**
   * Dependencies: *Spring Shell*, *Spring Boot Actuator*
   * Provides interactive console commands to generate invoices
3. **Data Ingestion Pipeline**
   * Dependencies: *Spring Batch*, *Spring Boot DevTools*
   * Reads CSV files, processes records, and writes to a database

**🧩 Summary Comparison**

| **Feature** | **Spring Stereotypes** | **Spring Initializr Application** |
| --- | --- | --- |
| **Purpose** | Class-level bean registration & semantic role | Project-level scaffolding & bootstrapping |
| **Key Annotations/Artifact** | @Service, @Repository, @Controller, etc. | @SpringBootApplication, starter POMs |
| **Configuration Style** | Annotation-driven, component scanning | Auto-generated build + main class |
| **Typical Usage** | Coding layers (DAO/Service/Web) | Start of any Spring Boot project |

## **Simple REST Service Using Spring Web**

Building RESTful web services is one of the most common tasks in modern backend development. Spring Boot’s **spring-boot-starter-web** (a.k.a. “Spring Web”) makes it trivial to expose HTTP endpoints with minimal boilerplate. Below is a deep dive into its purpose, rules, typical usage scenarios, and three real-world examples.

**🔹 Why It Came**

1. **Reduce Boilerplate**  
   Before Spring Boot Web, you had to manually configure a servlet container (Tomcat, Jetty, etc.), register the DispatcherServlet, set up view resolvers or message converters, and wire controllers via XML or verbose Java config.
2. **Convention over Configuration**  
   Spring MVC already provided a powerful web framework; Spring Boot Web simply auto-configures it with sensible defaults (embedded server, Jackson for JSON) so you can start writing controllers immediately.
3. **Rapid API Development**  
   With embedded servers and starters, you package your service as a runnable JAR, deploy anywhere, and iterate faster—ideal for microservices, prototypes, and cloud-native apps.

**📌 Core Rules & Annotations**

| **Annotation** | **What It Does** |
| --- | --- |
| **@SpringBootApplication** | Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan to bootstrap apps. |
| **@RestController** | Marks a class as a REST controller; all handler methods return bodies (no view resolution). |
| **@RequestMapping**, **@GetMapping**, **@PostMapping** | Map incoming HTTP methods and paths to specific handler methods. |
| **@PathVariable** | Binds a URI template variable (e.g., /users/{id}) to a method parameter. |
| **@RequestBody** | Binds the HTTP request body (JSON/XML) to a Java object. |
| **@ResponseStatus** | Overrides the default HTTP status code returned by a handler. |
| **@ControllerAdvice** | Global error-handling across controllers (with @ExceptionHandler). |

**Configuration Properties (common):**

* server.port – change the embedded server port.
* spring.mvc.pathmatch.matching-strategy – control URL matching.
* spring.jackson.\* – customize JSON serialization (date formats, property naming).

**📍 When & Where to Use**

* **Microservices & REST APIs**  
  Exposing domain operations (CRUD) over HTTP for distributed systems.
* **Backend for SPAs & Mobile Apps**  
  Serving JSON payloads to JavaScript frameworks (React, Angular) or native mobile clients.
* **Internal Service Integrations**  
  Light-weight HTTP endpoints for inter-service communication or webhooks.
* **Rapid Prototyping**  
  Stand up an API quickly without external application servers.

**✅ Real-World Examples**

1. **User Management Service**
   * **Endpoints:**
     + GET /users – list users
     + GET /users/{id} – retrieve one
     + POST /users – create
     + PUT /users/{id} – update
     + DELETE /users/{id} – delete
   * **Use Case:** Central auth/user-profile microservice in a larger platform.
2. **Product Catalog API (E-commerce)**
   * **Endpoints:**
     + GET /products?category=electronics – filter
     + GET /products/{sku} – details
     + POST /products – add new SKU
   * **Use Case:** Feeds product data to front-end UI and downstream pricing/analytics.
3. **Order Processing Service**
   * **Endpoints:**
     + POST /orders – place an order (accepts JSON)
     + GET /orders/{orderId} – track status
     + PATCH /orders/{orderId}/status – update status (e.g., “shipped”)
   * **Use Case:** Part of a CQRS/event-driven architecture, integrates via Kafka or message queues for fulfillment and notifications.

**🛠 Best Practices**

1. **Stateless Controllers**  
   Don’t hold mutable state in controllers. Delegate business logic to @Service beans.
2. **Use DTOs**  
   Avoid exposing JPA entities directly; use separate Data Transfer Objects for requests/responses.
3. **Validation**  
   Leverage @Valid and JSR-380 annotations (@NotNull, @Size, etc.) on DTOs, with a @ControllerAdvice to produce consistent error responses.
4. **Versioning**  
   Plan for API evolution (e.g., URL versioning /v1/users or header-based).
5. **Documentation**  
   Integrate Swagger/OpenAPI (Springdoc or Springfox) to auto-generate interactive API docs at /swagger-ui.html.

**Spring Boot & Core Spring Concepts – MCQ Quiz**

**1. What is the main purpose of Spring Boot?**

a) Replace Spring Framework entirely  
b) Simplify Spring application setup with minimal configuration  
c) Create only web-based applications  
d) Replace Hibernate ORM  
  
**Answer:** b

**2. Which feature of Spring Boot eliminates the need for a lot of boilerplate configuration?**

a) Auto-Configuration  
b) Manual XML Configuration  
c) Hardcoded Bean Creation  
d) JDBC Templates only  
  
**Answer:** a

**3. Which file is used by default for Spring Boot configuration?**

a) spring.xml  
b) application.properties  
c) boot-config.yml  
d) context.properties  
  
**Answer:** b

**4. The annotation @SpringBootApplication is a combination of:**

a) @Configuration, @EnableAutoConfiguration, @ComponentScan  
b) @Bean, @Configuration, @Service  
c) @Controller, @Repository, @Component  
d) @EnableWebMvc, @Configuration, @Bean  
  
**Answer:** a

**5. Which of the following is NOT a Spring stereotype annotation?**

a) @Controller  
b) @Service  
c) @Entity  
d) @Repository  
  
**Answer:** c

**6. The @Service annotation is used to:**

a) Define a data access object  
b) Mark a business logic class  
c) Enable scheduling in Spring Boot  
d) Register a REST endpoint  
  
**Answer:** b

**7. Which Spring stereotype annotation is typically used in persistence layer classes?**

a) @Controller  
b) @Repository  
c) @Service  
d) @Bean  
  
**Answer:** b

**8. The default Spring bean scope is:**

a) prototype  
b) singleton  
c) request  
d) session  
  
**Answer:** b

**9. Which Spring bean scope creates a new bean instance every time it’s requested from the container?**

a) singleton  
b) prototype  
c) request  
d) session  
  
**Answer:** b

**10. Which scope is only applicable in a web-aware Spring context?**

a) singleton  
b) prototype  
c) request and session  
d) all scopes  
  
**Answer:** c

**11. In Spring Boot, which annotation is used for dependency injection by type?**

a) @Autowired  
b) @Qualifier  
c) @Inject  
d) @Primary  
  
**Answer:** a

**12. Which annotation would you use to specify exactly which bean to inject when multiple beans of the same type exist?**

a) @Autowired  
b) @Qualifier  
c) @Scope  
d) @Bean  
  
**Answer:** b

**13. What will happen if @Autowired is used and Spring cannot find the required bean?**

a) It will silently skip injection  
b) It will throw NoSuchBeanDefinitionException  
c) It will create a default bean  
d) It will set the field to null without error  
  
**Answer:** b

**14. Which annotation is used to define a bean in a @Configuration class?**

a) @Service  
b) @Bean  
c) @Component  
d) @Repository  
  
**Answer:** b

**15. Which Spring annotation enables scanning of packages for components?**

a) @ComponentScan  
b) @BeanScan  
c) @EnableComponent  
d) @AutoScan  
  
**Answer:** a

**16. In Spring Boot, which annotation is used to mark a class as a REST controller?**

a) @RestController  
b) @Controller  
c) @Service  
d) @Repository  
  
**Answer:** a

**17. The @Primary annotation in Spring is used to:**

a) Mark a bean for prototype scope  
b) Mark a bean as the default candidate for autowiring  
c) Mark a bean for lazy initialization  
d) Mark a bean for request scope  
  
**Answer:** b

**18. Which annotation in Spring Boot can be used to read values from application.properties into a Java class?**

a) @Autowired  
b) @Value  
c) @Inject  
d) @ConfigProperty  
  
**Answer:** b

**19. Which Spring bean scope creates a new bean per HTTP session?**

a) singleton  
b) prototype  
c) session  
d) request  
  
**Answer:** c

**20. Which annotation in Spring Boot can be used to enable scheduling?**

a) @EnableScheduling  
b) @EnableTasks  
c) @EnableJobs  
d) @Schedule  
  
**Answer:** a