

# Spring Boot

# Introduction to Spring Boot

- **Spring Boot** is a project that is built on top of the **Spring Framework**.
  - It is a Spring module that provides the RAD (Rapid Application Development) feature
- It provides an **easier and faster** way to **set up, configure, and run** both **simple** and **web-based** applications.
- **Spring Boot** is a **microservice-based framework**
  - An open-source Java-based framework used to create standalone microservices with production-ready features.
- **Microservice** is an **architectural design** that **creates**
  - scalable, loosely coupled, and testable applications which have a single function module with well-defined interfaces.
- The **microservices** can be **owned** and **maintained** by a **small team** and is **adapted in an Agile manner**

# Difference between Spring and Spring Boot

	Spring	Spring Boot
What is it?	An <b>open-source web application framework</b> based on <b>Java</b> .	An extension or module built on the <b>Spring framework</b> .
What does it do?	Provides a flexible, completely configurable environment using tools and libraries of prebuilt code to create customized, loosely coupled web apps.	Provides the ability to create, standalone Spring applications that can just run immediately without the need for annotations, XML configuration, or writing lots of additional code.
When should I use it?	Use Spring when we want: <ul style="list-style-type: none"><li>•Flexibility</li><li>•An unopinionated approach.</li><li>•To remove dependencies from your custom code.</li><li>•To implement a very unique configuration.</li><li>•To develop enterprise applications.</li></ul>	Use Spring Boot when we want: <ul style="list-style-type: none"><li>•Ease of use</li><li>•An opinionated approach.</li><li>•To get quality apps running quickly and reduce development time.</li><li>•To avoid writing boilerplate code or configuring XML.</li><li>•To develop REST APIs.</li></ul>
What's its key feature?	Dependency injection	Autoconfiguration
Does it have embedded servers?	No. In Spring, you'll need to set up the servers explicitly.	Yes, Spring Boot comes with built-in HTTP servers like Tomcat and Jetty.
How is it configured?	The Spring framework provides flexibility, but its configuration has to be built <b>manually</b> .	Spring Boot configures Spring and other third-party frameworks automatically by the default " <b>convention over configuration</b> " principle.
Do I need to know how to work with XML?	In Spring, knowledge of <b>XML configuration</b> is <b>required</b> .	Spring Boot does <b>not require XML configuration</b> .
Are there CLI tools for dev/testing apps?	The Spring framework alone <b>doesn't provide CLI</b> tools for developing or testing apps.	As a Spring module, Spring Boot <b>has a CLI tool</b> for developing and testing Spring-based apps.
Does it work from an opinionated or unopinionated approach?	<b>Unopinionated</b>	<b>Opinionated</b>

# What is dependency injection?

- **Dependency injection** (DI) is a design technique used to achieve inversion of control (IoC).
- In object-oriented programming like Java, **objects that depend on other objects** are called **dependencies**.
- Typically, the receiving or **dependent object** is called a **client** and the **object that the client is dependent on** is called a **service**.
- So, **dependency injection** **passes** the **service** to the **client**, or "injects" the **dependency using code** called an **injector**.
- **DI eliminates** the need for the client to specify which service to use—the **injector** does that work for the client.

# Features of Spring Boot

- **Web Development**

- It is well suited Spring module for web application development. We can easily create a self-contained HTTP server using embedded Tomcat, Jetty or Undertow. We can use the spring-boot- starter-web module to start and running application quickly.

- **SpringApplication**

- It is a class which provides the convenient way to bootstrap a spring application which can be started from main method. You can call start your application just by calling a static run() method.

- **Application Events and Listeners**

- Spring Boot uses events to handle variety of tasks. It allows us to create factories file that are used to add listeners. we can refer it by using ApplicationListener key.
- Always create factories file in META-INF folder like: **META-INF/spring.factories**

- **Admin Support**

- Spring Boot provides the facility to enable admin related features for the application. It is used to access and manage application remotely. We can enable it by simply using spring.application.admin.enabled property.

- **Externalized Configuration**

- Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments. Application use YAML files to externalize configuration.

# Features of Spring Boot

- **Properties Files**
  - Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: **server-port = 8082** and many others. It helps to organize application properties.
- **YAML Support**
  - It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.
- **Type-safe Configuration**
  - Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.
- **Logging**
  - Spring Boot uses Common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies, if there is no required customization is needed.
- **Security**
  - Spring Boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of Endpoints are available for develop a secure Spring Boot application.

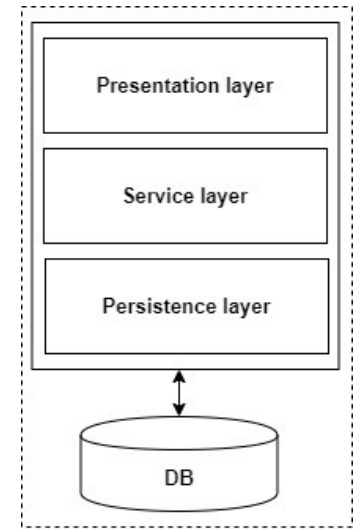
# Advantages of Spring Boot

- **Spring Boot works well with several servlet containers**
  - Spring Boot works well with some of the most popular embedded servlet containers.
  - Spring Boot uses [Tomcat](#) as its default.
- **Bootstrapping saves memory space**
  - Spring Boot uses Boot Initializer to compile the source language.
  - This bootstrapping technique makes it possible for users to save space on their devices and load applications quickly.
- **Decreased boilerplate code**
  - Spring Boot's in-memory database and embedded server (Tomcat) decrease or eliminate the boilerplate code typically needed to set up an application.
- **No XML configuration required**
- **WAR files are not required**
  - Spring Boot can rely on JAR (Java resource). It may use WAR (web application resource) files, they are not necessary
- **POM dependency management**
  - Spring Boot doesn't force to use a parent POM (project object model).
  - Adding the spring-boot-dependencies artifact will manage dependencies without relying on a parent POM or XML file..
- **A large community of helpful users**
  - Spring Boot has a large community of users full of people who enjoy sharing their insights and creations.

# Breaking the monolithic way of developing software

- **Monolithic architecture**

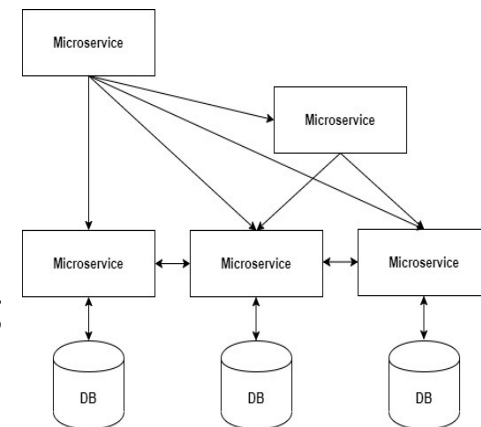
- In monolithic application, “**mono**” represents the single **codebase** containing all the required functionalities
- If all the **functionalities** of a project exist in a **single codebase**, then that application is known as a **monolithic application**.
  - As an example, if we are given a **problem statement** and were asked to **design a system** with **various functionalities**.
  - The entire application is **tested and deployed** at once



*Monolithic architecture*

- **Micro Services**

- It is an **architectural development style**
  - The application is made up of smaller services
- Each service can have a **single functionality**, single data repository.
- **Each service is independent of each other** so that the **change in one service doesn't impact the whole application**.
- Each service can **communicate** with each other using **Inter Process Communication (IPC)** calls via web services/APIs.
- **Each service can be tested independently**
- Deploying each application independently in an isolated environment which can share different external services.



*Microservices architecture*



# Micro Services contd..

- When to use micro services?
  - Migrate legacy applications to the new technology
  - Changing technology stack for one service would not affect the whole application.
  - Create high-performance and scalable services which only serves the single purpose that requires high computation/memory/resources.
  - Experiment with the Agile methodology where requirements come in periodic intervals and the same can be delivered in a short span of time unlike monolithic where all the bundles are installed at once.
- When not to use micro services?
  - Cost increased since each microservice runs in isolated virtual machines.
  - If the development team is of small size, then it must manage all independent small size services which may lead to big changes in the capacity and productivity of the team.
    - This can put a strain on all the operating units and the developers of that particular service.
  - If the requirement is that tiny, then it cannot be further broken into different service creations.
  - Some support of tools in legacy applications may not be supported in the microservice architecture because of tool limitation.

# System requirements

Tool	Version
Java	17 and higher
Apache Maven	3.3 and higher
Gradle	6.3 and higher
Spring Tool Suite	3.9.4.RELEASE and higher
Eclipse	EE and Web Developers

# The 12-factor app

- Any developer, who builds the application that runs as a service, should incorporate the 12-factors in their application

## I. Codebase

One codebase tracked in revision control, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Config

Store config in the environment

## IV. Backing services

Treat backing services as attached resources

## V. Build, release, run

Strictly separate build and run stages

## VI. Processes

Execute the app as one or more stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep development, staging, and production as similar as possible

## XI. Logs


Treat logs as event streams

## XII. Admin processes

Run admin/management tasks as one-off processes

# Spring Initializr

- One of the best ways to create a Spring Boot application is to generate the skeleton project from **Spring Initializr**
  - <https://start.spring.io/>



**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy

☒ **Maven**

**Spring Boot**

☐ 3.1.0 (SNAPSHOT) ☒ **3.1.0 (M1)** ☐ 3.0.6 (SNAPSHOT) ☐ 3.0.5

☐ 2.7.11 (SNAPSHOT) ☐ 2.7.10

**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ **Jar** ☐ War

Java ☐ 19 ☒ **17** ☐ 11 ☐ 8

**Dependencies** ADD DEPENDENCIES... CTRL + B

**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**GENERATE** CTRL + G **EXPLORE** CTRL + SPACE **SHARE...**

Activate Wi  
Go to Settings

# Build Tools- Maven

- **Need of Build Tools:**

- For **building heavy projects**, there is a need of **segregating codes into components** so that they can be **reused again within the same project or by other projects**.
- Here, **the project** is the **build module** for **an application**.
- For using these components in the project, need to **add components into our classpath** so that our project gets these components and the build is successful.
- To **add the components to classpath**, there are **build tools available** on the Internet
- The famous build tools are **Maven** and **Gradle**.

- **Maven:**

- **Maven** is the **project management build tool** developed by **Apache Org** which adds the functionalities of Java libraries through **dependencies**.
  - Maven is a **stage-driven build tool** and its **lifecycle is divided into stages** such as **validate**, **compile**, **test**, **package**, **verify**, **install**, and **deploy**.
- **Developers** can **create their own dependencies** for the purpose of modularity.
  - They can also use the dependencies stored in the repository.
- The core component of the Maven project is **pom.xml**.

# Understanding the entry point class

```
package com.author.kickstart;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
@SpringBootApplication  
public class KickstartApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(KickstartApplication.class, args);  
    }  
}
```

- The main function of the class is calling the run function of the **SpringApplication** class
  - When this is called during runtime, it loads all the **dependencies**

# SpringBootApplication annotation

- The **@SpringBootApplication** annotation describes the class where it is used in a Spring Boot application.
- This annotation is a combination of the following annotations:
  - [@SpringBootConfiguration](#) : allow to register extra beans in the context or import additional configuration classes
  - [@EnableAutoConfiguration](#): enable [Spring Boot's auto-configuration mechanism](#)
  - [@ComponentScan](#): enable @Component scan on the package where the application is located

# Program

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class LearnSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(LearnSpringBootApplication.class, args);
    }
}
```

```
public class Course {
    private long id;
    private String name;
    private String author;
    public Course(long id, String name, String author) {
        super();
        this.id = id;
        this.name = name;
        this.author = author;
    }
    public long getId() {return id;}
    public String getName() {return name;}
    public String getAuthor() {return author;}
    @Override
    public String toString() {
        return "Course [id=" + id + ", name=" + name + ", author=" + author + "];"
    }
}
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class CourseController
{
    @RequestMapping("/courses")
    public List<Course> retrieveAllCourses()
    {
        return Arrays.asList(
            new Course(1, "Learn AWS", "in28minutes"),
            new Course(2, "Learn DevOps", "in28minutes"),
            new Course(3, "Learn Azure", "in28minutes"),
            new Course(4, "Learn GCP", "in28minutes"));
    }
}
```