Spring Boot - Dependency Management

Last Updated : 29 Jul, 2025

Spring Boot framework is the most popular web development framework. No doubt, it provides an abundance of essential features and a convenient way to handle those features. At the heart of Spring Boot is the 'Dependency Management' feature.

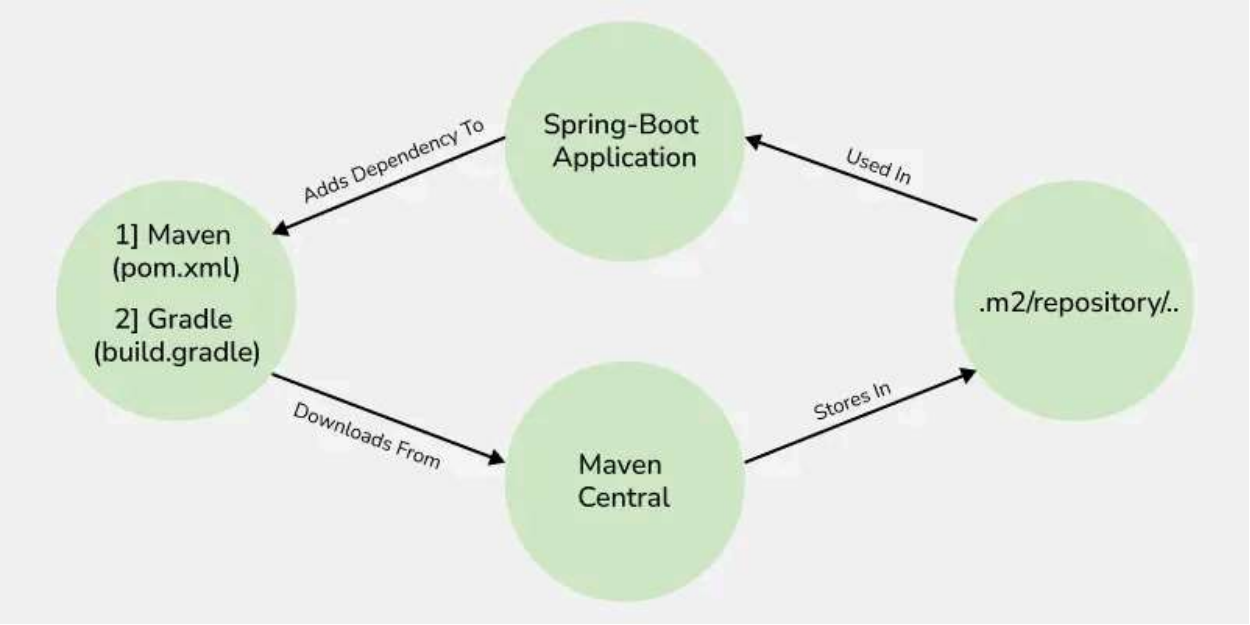
Importance of Dependency Management

- **Centralized Dependency Management:** All dependencies are managed in one place, typically in the pom.xml (for Maven) or build.gradle (for Gradle) file.
- **Automatic Version Management:** When we change the Spring Boot version, the versions of all related dependencies are automatically updated.
- **Conflict Prevention:** It helps prevent version conflicts between different libraries, especially in multi-module projects.

> ***Note:*** *Dependency Management is a way of managing all the required dependencies in one place and efficiently making use of them.*

Life Cycle of Dependency Management

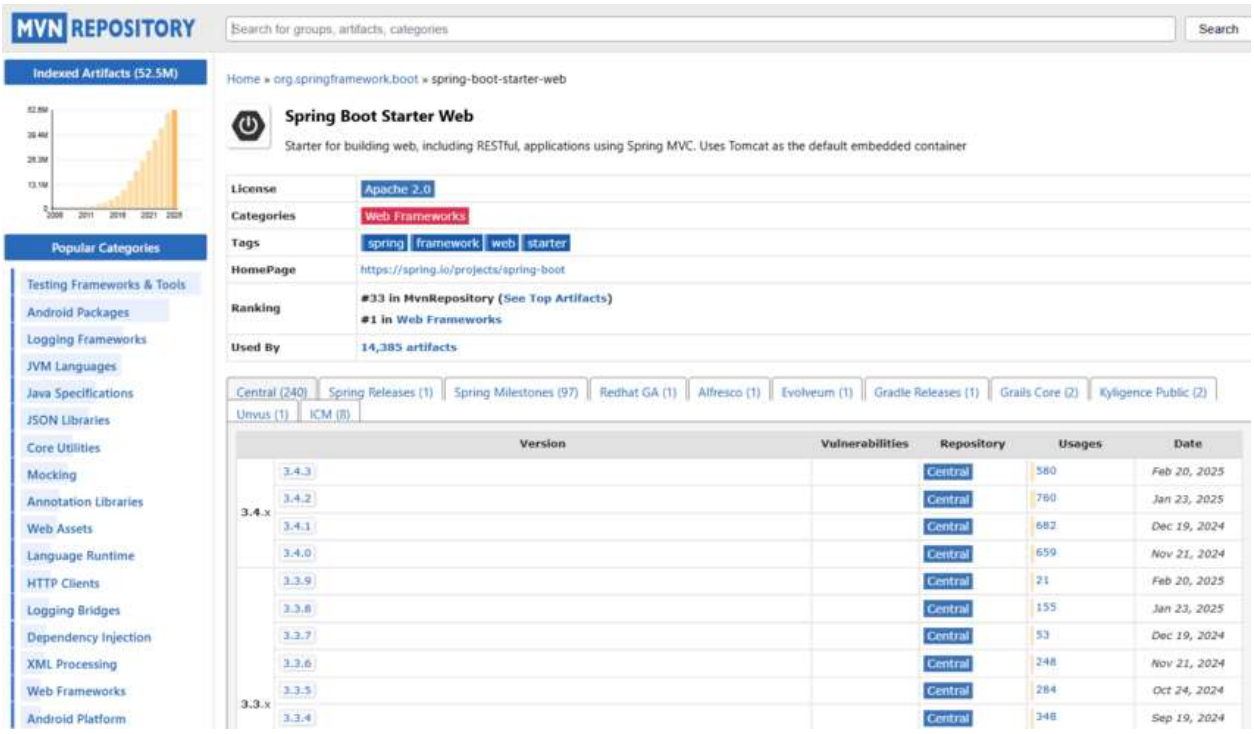
The below diagram demonstrates the type of Dependency Management



life-cycle

Working of Dependency Management in Spring-Boot

- Dependency is nothing but a 'Library' that provides specific functionality that we can use in our application.
- In Spring-Boot, Dependency Management and Auto-Configuration work simultaneously.
- It is the auto-configuration that makes managing dependencies supremely easy for us.
- We have to add the dependencies in the pom.xml/build.gradle file.
- These added dependencies will then get downloaded from Maven Central.
- The downloaded dependencies will get stored into the '.m2' folder in the local file system.
- The Spring-Boot application can access these dependencies from '.m2' and its sub-directories.
- Example -(**.m2** -> **repository** -> **org**, etc)



Project Build Systems

- Spring Boot supports two main build system Maven and Gradle.
- **Maven:** Dependencies are managed in the **pom.xml** file.
- **Gradle:** Dependencies are managed in the **build.gradle** file.
- Maven and Gradle use a different syntax for managing dependencies.
- Also, we don't need to mention the version of the dependencies, as Spring-Boot configures them automatically. Though we can mention the version or override as well.
- The curated list published contains all the Spring Modules and third-party libraries that you can use with Spring-Boot.

Key features of Maven build

- It uses the default Java compiler.
- It has UTF-8 source encoding
- A useful feature of not mentioning the version information of dependencies is inherited from POM (spring-boot-dependencies).
- Resource filtering and plugin configurations.

- Resource filtering is also for 'application.properties' and 'application.yml'.

Spring-Boot Starters

Spring Boot Starters are a set of convenient dependency descriptors provided by Spring Boot that simplify the setup of your application by grouping commonly used libraries and configurations into a single, reusable module.

> ***Example:*** *'spring-boot-starter-jdbc'*

Types of Starters

- **Application Starters:** For building specific types of applications (e.g., spring-boot-starter-web).
- **Technical Starters:** For technical features like logging or security (e.g., spring-boot-starter-security).
- **Production-ready Starters:** For monitoring and managing production applications (e.g., spring-boot-starter-actuator).

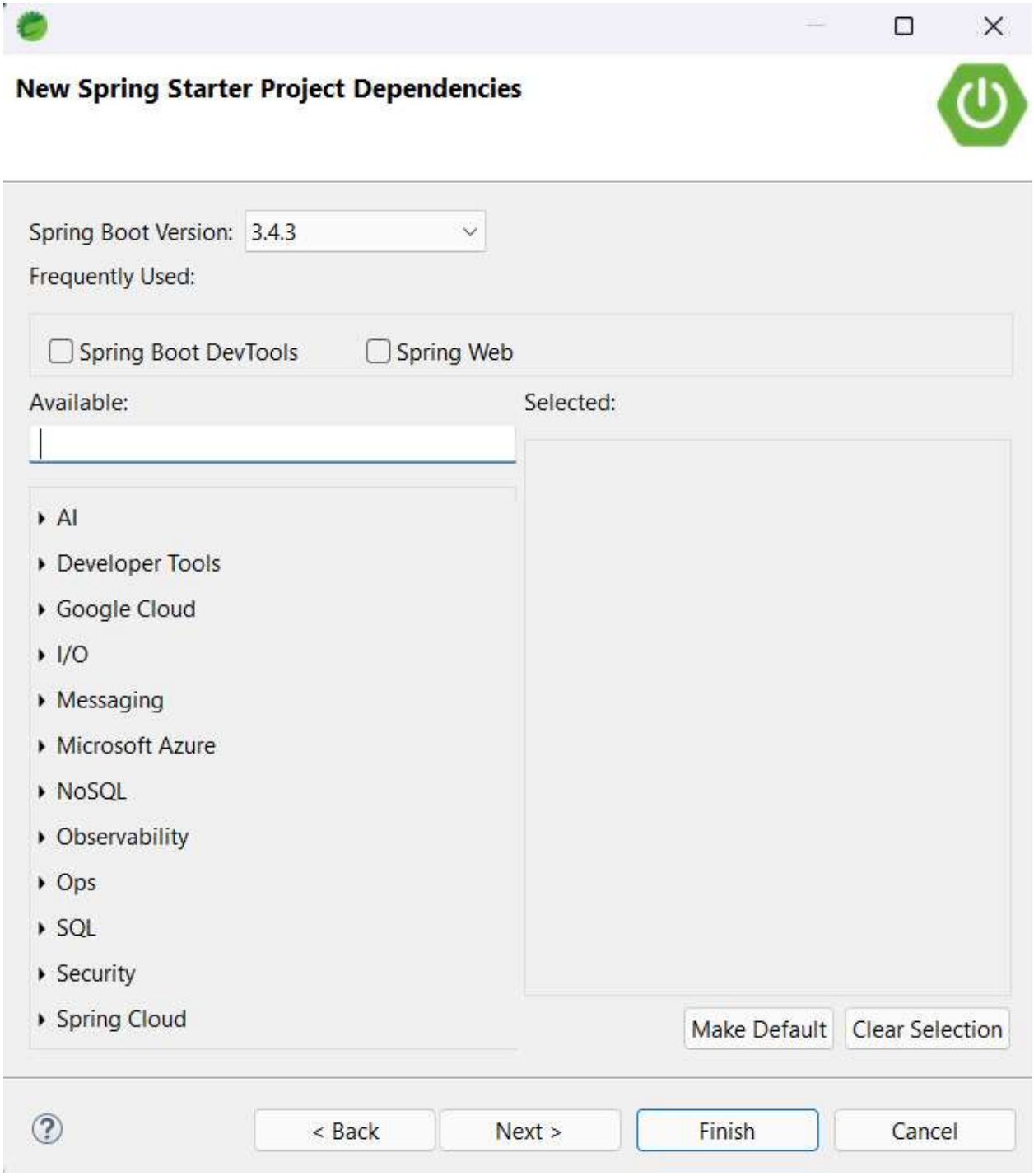
All the required dependencies of Spring-Boot are embedded in the 'dependencies' tag/block.

```
Maven -> pom.xml
<dependencies>
     <dependency>
          <groupId> ... </groupId>
          <artifactId> ... </artifactId>
          <version> ... </version>
     </dependency>
</dependencies>
```

Adding Dependencies in Spring Boot

1. Using Maven (pom.xml)

In STS, create a Spring Boot project by selecting **File > New > Spring Starter Project, adding required dependencies and clicking Finish**

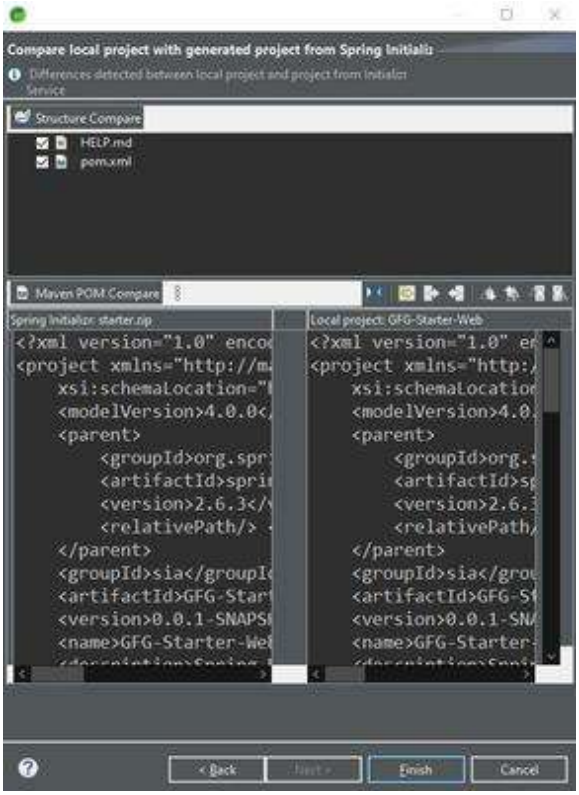


New Spring Starter Project Dependencies

To add the dependency for the current working project:

1. Right-click on project
2. Select Spring -> Add Starters
3. Search for the required dependencies and add them
4. Next

5. Select pom.xml/HELP.md or both
6. Finish



If we know the dependency, we can directly place them in the pom.xml file. For example to add the Thymeleaf template engine in your project build, one can add the following dependency in the '**<dependencies> </dependencies>**' tag.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.4.3</version>
        <relativePath/> <!-- lookup parent from repository -->
```

```
    </parent>

    <groupId>sia</groupId>
    <artifactId>GFG</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>GFG</name>
    <description>GFG Application</description>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter Web for building web applications -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot Starter Thymeleaf for rendering views -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <!-- Lombok for reducing boilerplate code -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- Spring Boot Starter Test for unit testing -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- Spring Boot DevTools for enhanced development experience -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Spring Boot Maven Plugin for packaging the application -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
```

```
                        <!-- Exclude Lombok from the packaged application -->
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Understanding/Configuring Dependencies

Starter Parent: To take advantage of auto-configured 'sensible' defaults, we should add Starter Parent in the project.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>___</version>
</parent>
```

With default configuration like above, we can override respective dependencies by overriding a 'property'.

```
<properties>
    <slf4j.version>___</slf4j.version>
</properties>
This will make sure that the mentioned version of a SLF4j library will be
used.
```

We can also manage auto-configured 'Starter Parent' and create a custom POM without the need to specify the first one with the help of artifact 'scope=import' of 'spring-boot-dependencies.

```
 <dependencyManagement>
     <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>___</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

After this, we can normally add the dependencies like the one mentioned above. But, to override the individual dependency, we need to add a respective entry before the 'spring-boot-dependencies' entry.

```
<dependencyManagement>
    <dependencies>

        <!-- Override SLF4J provided by Spring Boot -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>___</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>___</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>

    </dependencies>
</dependencyManagement>
```

But, we have to manually configure the plugin management by adding 'spring-boot-maven-plugin' explicitly. Managing the Maven plug-in is very essential as it packs the Spring-Boot application into an executable jar.

```
Maven -> pom.xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Java Version: We can also change the java version in the following:

```
<properties>
    <java.version>___</java.version>
</properties>
```

Developer Tools: A set of specific tools to make the application development process much easier. It is in the 'spring-boot-devtools'

module.

```
Maven -> pom.xml
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
     <optional>true</optional>
</dependency>
```

2. Using Gradle

In the case of a 'Starter Parent' like in Maven, here there is no 'Super Parent' to take advantage of some auto configurations. To add dependencies in Gradle, add them in the 'dependencies{ }' section. For providing executable jar, we can add the following in the dependencies section

'spring-boot-gradle-plugin'

build.gradle:

```
buildscript {
    repositories {
        mavenCentral() // Use Maven Central instead of jcenter (jcenter is
deprecated)
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:3.4.4")
    }
}
apply plugin: 'java'
apply plugin: 'org.springframework.boot'
repositories {
    mavenCentral() // Use Maven Central for dependencies
}
dependencies {
    implementation("org.springframework.boot:spring-boot-starter-web")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
}
```

For adding the '**Developer tools**', add the following in the 'dependencies' block

Gradle -> build.gradle

```
developmentOnly("org.springframework.boot:spring-boot-devtools")
```

***Note**: Each release of Spring Boot is associated with a base version of the Spring Framework, so it is highly recommended to not specify its version on your own.*