WHAT IS DEPENDENCY ?

In Spring Boot (and Java in general), a *dependency* is an external library or module that your application needs to function properly. These dependencies are included in the project and automatically managed, making it easier to reuse code and avoid reinventing the wheel for common functionalities (e.g., JSON processing, database interaction, security, etc.).

```
+---------------------------------------------+
|               Project                 |
|             (pom.xml)                 |
+---------------------------------------------+
           |        |         |
           |        |         |
           v        v         v
+---------------+  +---------------+  +---------------+
|  Dependency  |  |  Dependency  |  |  Dependency  |
|           |  |           |  |           |
|  <groupId>   |  |  <groupId>   |  |  <groupId>   |
|  <artifactId> |  |  <artifactId> |  |  <artifactId> |
|  <version>   |  |  <version>   |  |  <version>   |
+---------------+  +---------------+  +---------------+
           |        |         |
           v        v         v
+-------------------------------------------------------+
|                Maven Repository                |
|             (Local / Central Repo)             |
+-------------------------------------------------------+
           |
           v
+-------------------------------------------------------+
|              Dependency Download                 |
|        (Added to Project Build Path)               |
+-------------------------------------------------------+
```

Using Maven

In Maven, dependencies are managed in the `pom.xml` file. Each dependency is defined with its group ID, artifact ID, and version, like this:

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

To import a dependency:

1. Open your `pom.xml` file.
2. Inside the `<dependencies>` section, add the `<dependency>` tags for any libraries you need.
3. Save the file. Maven will download the dependencies and include them in your project.

WHAT IS GROUPID & ARTIFACTID?

Group ID (`groupId`)

The `groupId` represents the *group*, *organization*, or *company* that the project belongs to. It is often structured as a reversed domain name (e.g., `com.example`, `org.springframework.boot`) to prevent naming conflicts.

For example:

- `org.springframework.boot`: Group ID for Spring Boot libraries.
- `com.fasterxml.jackson.core`: Group ID for Jackson libraries, which are commonly used for JSON processing.

2. Artifact ID (`artifactId`)

The `artifactId` is the *name of the project* or *library*. This ID should be unique within the `groupId` namespace, as it represents the specific component you're including.

For example:

- `spring-boot-starter-web`: Artifact ID for the Spring Boot web starter (used to create web applications).
- `spring-boot-starter-data-jpa`: Artifact ID for Spring Data JPA, used for working with databases in Spring.

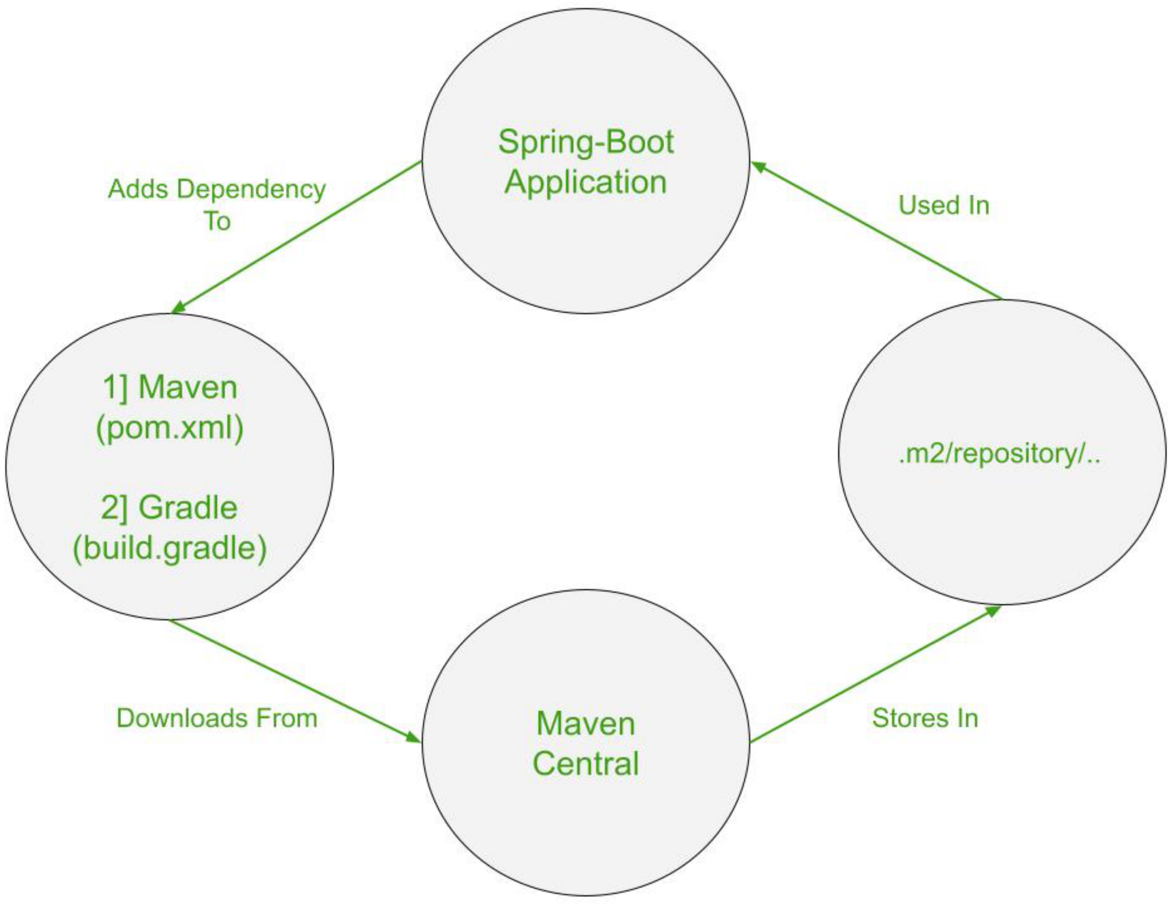
Common Dependencies in Spring Boot

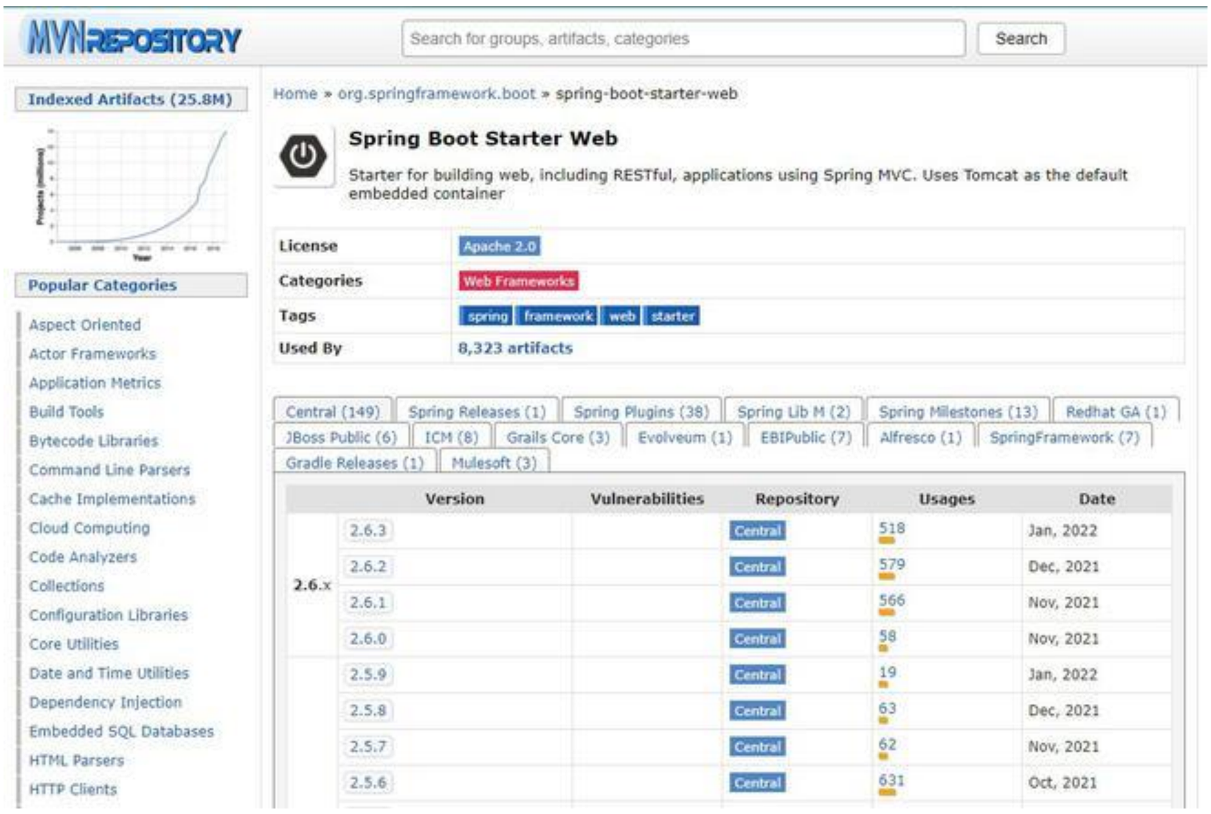
Some common dependencies for Spring Boot applications include:

- **spring-boot-starter-web**: For building web applications, including RESTful services.
- **spring-boot-starter-data-jpa**: For working with relational databases via JPA.
- **spring-boot-starter-security**: For adding authentication and authorization to your application.

Managing Dependencies

To manage dependencies in your Spring Boot application, you can either apply the io.spring.dependency-management plugin or use Maven's pom support.





Indexed Artifacts (25.8M)

Popular Categories

Aspect Oriented
Actor Frameworks
Application Metrics
Build Tools
Bytecode Libraries
Command Line Parsers
Cache Implementations
Cloud Computing
Code Analyzers
Collections
Configuration Libraries
Core Utilities
Date and Time Utilities
Dependency Injection
Embedded SQL Databases
HTML Parsers
HTTP Clients

Home » org.springframework.boot » spring-boot-starter-web

Spring Boot Starter Web

Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container

License	Apache 2.0
Categories	Web Frameworks
Tags	spring framework web starter
Used By	8,323 artifacts

Central (149) | Spring Releases (1) | Spring Plugins (38) | Spring Lib M (2) | Spring Milestones (13) | Redhat GA (1) | JBoss Public (6) | ICM (8) | Grails Core (3) | Evolveum (1) | EBIPublic (7) | Alfresco (1) | SpringFramework (7) | Gradle Releases (1) | Mulesoft (3)

	Version	Vulnerabilities	Repository	Usages	Date
2.6.x	2.6.3		Central	518	Jan, 2022
	2.6.2		Central	579	Dec, 2021
	2.6.1		Central	566	Nov, 2021
	2.6.0		Central	58	Nov, 2021
	2.5.9		Central	19	Jan, 2022
	2.5.8		Central	63	Dec, 2021
	2.5.7		Central	62	Nov, 2021
	2.5.6		Central	631	Oct, 2021

_By Scope

Dependency scope defines when and how a dependency is available in your project. In Maven, there are several scopes:

- **`compile`** (default):
  - The dependency is available at compile time and runtime, and it is included in the final packaged application.
  - Used for most application dependencies.
  - Example: `spring-boot-starter-web`.
- **`provided`**:
  - The dependency is required for compilation but is not included in the final package (JAR/WAR), as it is expected to be provided by the runtime environment.
  - Commonly used for Java EE or Servlet API libraries, where the server provides these dependencies.
  - Example: `javax.servlet:javax.servlet-api`.
- **`runtime`**:
  - The dependency is only needed at runtime, not at compile time. It's included in the final package.
  - Useful for libraries like JDBC drivers, where the API might be defined elsewhere.
  - Example: database drivers like `mysql:mysql-connector-java`.
- **`test`**:
  - Available only in the test phase, and not included in the final package.
  - Used for testing frameworks like JUnit and Mockito.
  - Example: `org.junit.jupiter:junit-jupiter`.
- **`system`**:

    - Similar to `provided`, but the JAR must be explicitly declared with an absolute path on the system (rarely used and not recommended).
- **`import`** (used in Dependency Management):
    - Only used with `<dependencyManagement>` to import dependencies from another BOM (Bill of Materials).
    - Typically used in Spring Boot projects for managing versions of Spring dependencies.

2. By Functionality in Spring Boot

In Spring Boot, dependencies are often grouped by their functionality:

- **Starter Dependencies**:
    - Spring Boot offers several "starter" dependencies that bundle related libraries for specific functionalities, making it easy to add multiple dependencies at once.
    - Examples:
        - **`spring-boot-starter-web`**: For building web applications with Spring MVC, Tomcat, Jackson, etc.
        - **`spring-boot-starter-data-jpa`**: For JPA-based data access with Hibernate.
        - **`spring-boot-starter-security`**: For implementing security features.
- **Core Dependencies**:
    - Fundamental Spring dependencies, such as `spring-core`, `spring-context`, and `spring-beans`, which are essential for the Spring Framework itself.
- **Third-Party Dependencies**:
    - External libraries that you may need, such as JWT libraries for token management or Apache Commons for utility functions.
    - Example: `io.jsonwebtoken:jjwt`.

3. By Dependency Relationship

Dependencies can also be classified based on their relationship to other dependencies in the project:

- **Direct Dependencies**:
    - Dependencies explicitly declared in `pom.xml`.
    - Example: If you add `spring-boot-starter-web` in `pom.xml`, it is a direct dependency.
- **Transitive Dependencies**:
    - Dependencies of a dependency. For example, `spring-boot-starter-web` brings in multiple transitive dependencies like `spring-web`, `spring-webmvc`, `tomcat`, etc.
    - Maven automatically includes transitive dependencies unless explicitly excluded.