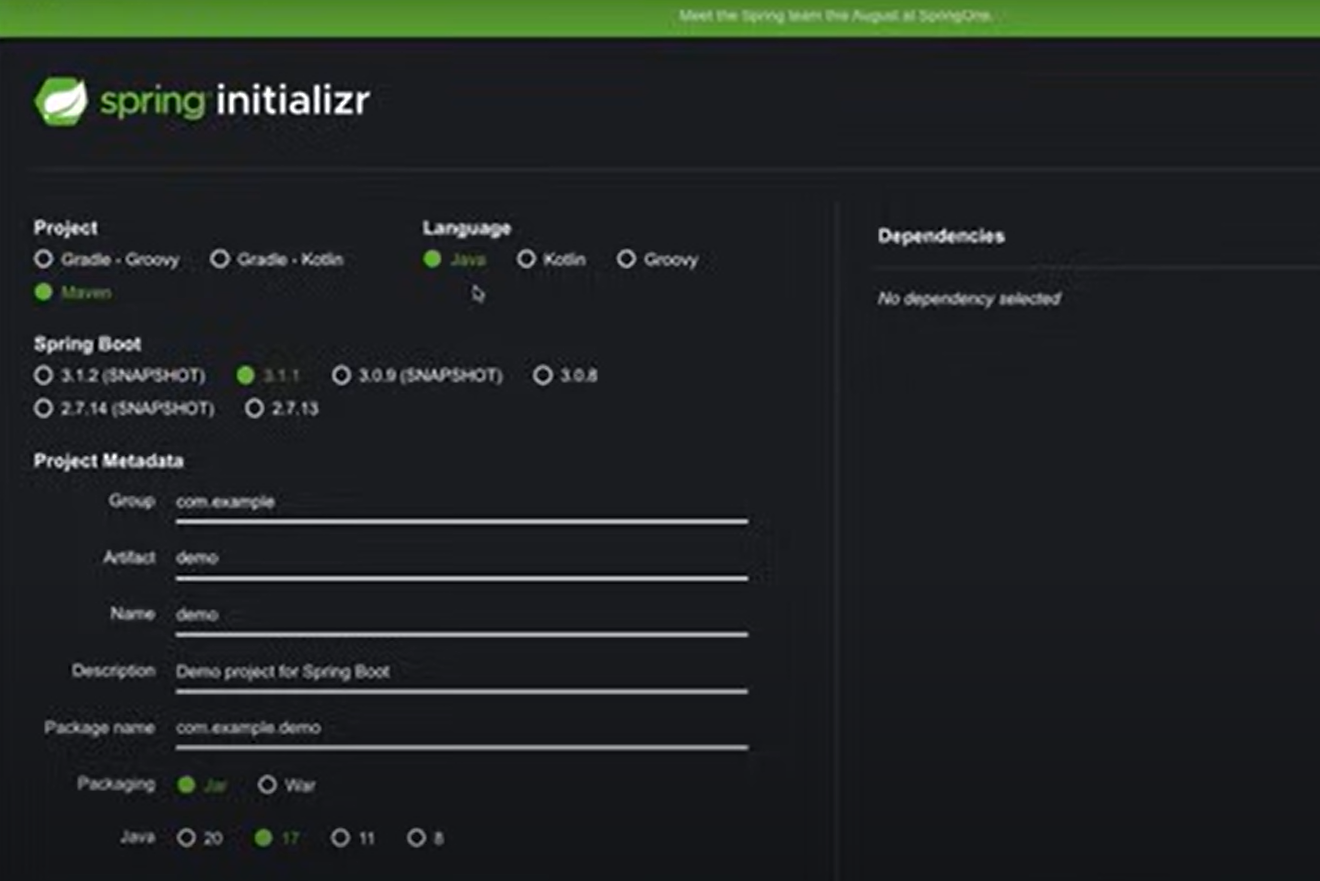
The ULTIMATE Guide to Spring Boot: Spring Boot for Beginners

# Initiating the SpringBoot

We can visit start.spring.io to get skeleton of the Spring boot.



We are faced with options like **gradle-Groovy, Gradle-Kotlin, and Maven.**

Gradle is built and dependency management tool. **Groovy** and **Kotlin** are different programming languages.

**Maven** is a build tool that compiles Java code, runs tests, and packages applications (e.g., .jar or .war files).

## Maven Role:

Dependency Management: Maven uses a configuration file (pom.xml) to manage dependencies. You declare the libraries or frameworks your project needs, and Maven automatically downloads them and their transitive dependencies from repositories like Maven Central.

Project Structure and Convention: Maven enforces a standard project structure, which is compatible with Spring Boot. For example: src/main/java -> Your application source code, src/main/resources -> Configuration files (like application.properties)

Build and Packaging: Maven automates tasks like compiling code, running tests, and packaging the application into a .jar or .war file. In Spring Boot, Maven can package the application as a standalone executable .jar file that includes an embedded web server like Tomcat.

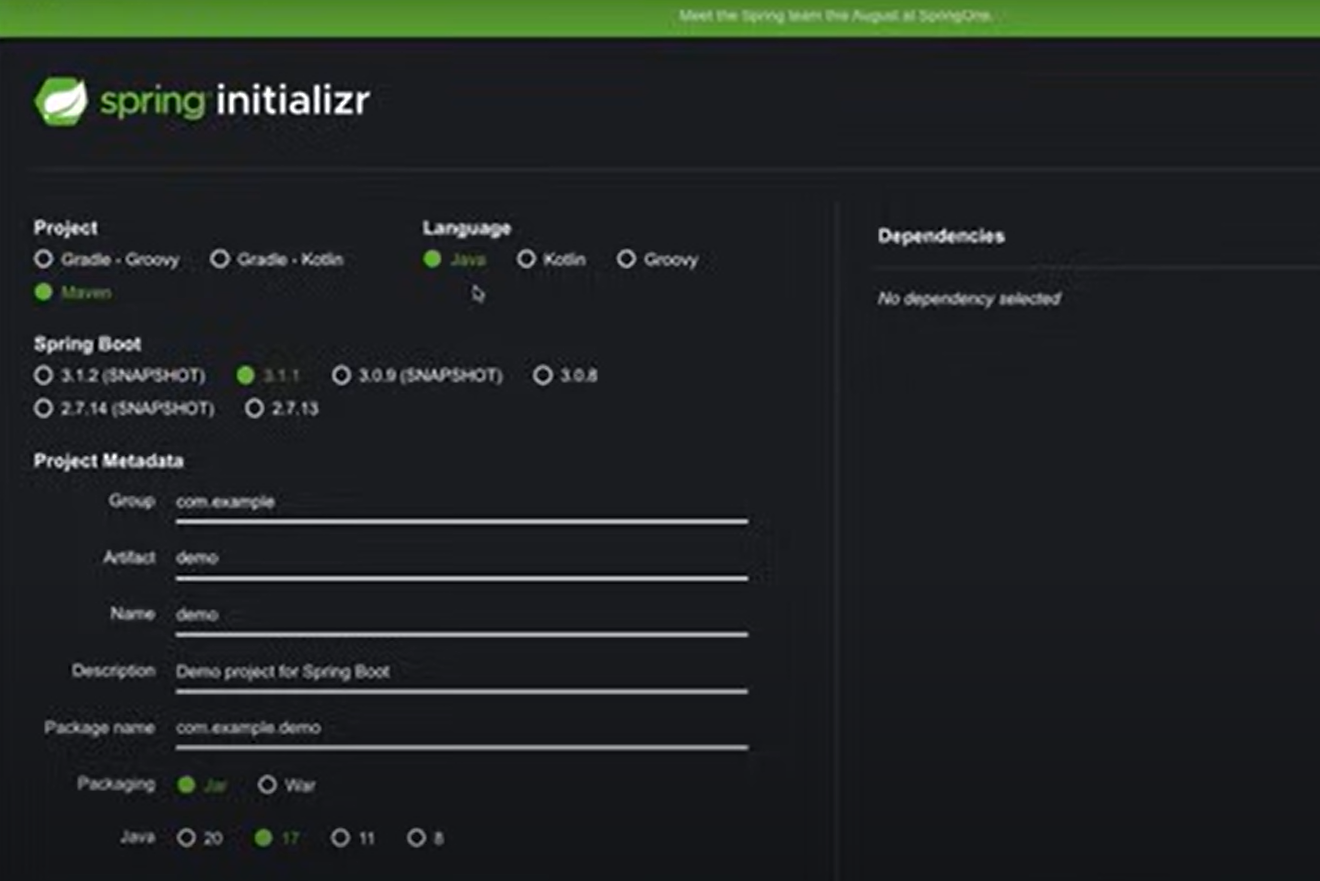
Plugins: Maven provides plugins to extend its functionality. Spring Boot projects often use the Spring Boot Maven Plugin, which simplifies running and deploying the application.

Profiles and Configuration: Maven supports profiles for environment-specific configurations, which can be useful for Spring Boot applications needing different settings for development, testing, and production environments.

Integration with IDEs: Maven makes it easy to import and configure Spring Boot projects in IDEs like IntelliJ IDEA or Eclipse, thanks to its standard conventions and dependency management.

## Spring Boot Version

Snapshot versions are the latest versions but they are unstable. For example in the following picture 3.1.1 is the general release version and it is stable and hence we should choose it.



## Project Metadata

### Group

The Group is a unique identifier for your project, often following a reverse domain name pattern (e.g., com.example).

It is conceptually similar to a Java package name and is used to avoid naming conflicts between projects. Common Practice is to use your organization's domain name in reverse (e.g., com.mycompany, org.myorg) and for personal projects, you might use something like com.yourname.

### Artifact

The Artifact is the name of your project or application (e.g., my-app). It represents the output artifact, such as a .jar or .war file. In Maven/Gradle, the artifactId is the name of the packaged build output, usually combined with the groupId to create a unique identifier for your project.

Common practice is to use lowercase with hyphens (-) as separators, e.g., my-app. Keep it short and descriptive of the application's purpose.

### Jar vs War

In Spring Initializr, when choosing between JAR and WAR for the project packaging, you're deciding how your application will be packaged and deployed.

#### JAR (Java Archive):

A JAR file is a standard packaging format for Java applications. It bundles the application code, resources, and dependencies into a single executable file.

Used for standalone applications with an embedded server (like Tomcat, Jetty, or Undertow) in Spring Boot. Suitable for microservices and modern cloud-native applications where you run the application as a self-contained process.

When you choose JAR in Spring Initializr, Spring Boot includes an embedded server (e.g., Tomcat) in the application. You can run JAR directly using: java -jar my-app.jar

#### WAR (Web Application Archive):

A WAR file is a packaging format for Java web applications designed to be deployed on an external application server (like Apache Tomcat, JBoss, or WildFly).

Used for traditional server-side web applications where an external servlet container is already set up. Suitable for legacy environments or scenarios where you must deploy to a shared server.

When you choose WAR in Spring Initializr, Spring Boot doesn't include an embedded server by default. Instead, the application is designed to be deployed to an external server that manages the application lifecycle.

## Dependencies

For the time being, we will just choose **Spring Web**.

# What is Apache Maven

11:17