Spring Boot Maven Plugin

To use the `spring-boot-maven-plugin` in a Maven-based Spring Boot project, follow these steps to configure and utilize the plugin for building, running, and managing your application. Below is a comprehensive guide:

---

1. Ensure Your Project is a Spring Boot Project

Before using the `spring-boot-maven-plugin`, confirm that your project is set up as a Spring Boot project. This typically involves:

- **Extending from `spring-boot-starter-parent` (Recommended)**:
  - In your `pom.xml`, set `spring-boot-starter-parent` as the parent to manage Spring Boot dependencies and plugin versions.
  - Example:

    ```xml
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.4</version> <!-- Replace with your Spring Boot version -->
        <relativePath/> <!-- Lookup parent from repository -->
    </parent>
    ```

- **Alternatively, Using `spring-boot-dependencies` BOM (Bill of Materials)**:
  - If you cannot use `spring-boot-starter-parent`, import the `spring-boot-dependencies` BOM in the `dependencyManagement` section.
  - Example:

    ```xml
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>2.5.4</version> <!-- Replace with your Spring Boot version -->
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    ```

Using `spring-boot-starter-parent` is recommended for simplicity, as it automatically manages plugin versions.

---

2. Add the `spring-boot-maven-plugin` to Your `pom.xml`

To use the plugin, you need to declare it in the `<build><plugins>` section of your `pom.xml`.

- **If Using `spring-boot-starter-parent`:**
  - Add the plugin without specifying the version, as it is managed by the parent.
  - Example:

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

- **If Not Using `spring-boot-starter-parent`:**
  - Specify the version explicitly, matching the Spring Boot version in use.
  - Example:

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>2.5.4</version> <!-- Replace with your Spring Boot version -->
        </plugin>
    </plugins>
</build>
```

---

3. Utilize Plugin Goals

The `spring-boot-maven-plugin` provides several goals to help build, run, and manage your Spring Boot application. Below are the most commonly used goals:

- **`spring-boot:run`**
  - Runs your Spring Boot application directly from Maven using an embedded web server (e.g., Tomcat).
  - Useful for development and testing.
  - Command:

    ```
    mvn spring-boot:run
    ```

- **`spring-boot:repackage`**
  - Repackages the JAR or WAR file generated by `mvn package` into an executable "fat JAR"or WAR that includes all dependencies.
  - This goal is automatically executed during the `package` phase if the plugin is configured.
  - Command:

    ```
    mvn package
    ```

  - After running, you can start the application with:

    ```
    java -jar target/myapp.jar
    ```

- **`spring-boot:start` and `spring-boot:stop`**
  - Used for integration tests to start and stop the application during the `pre-integration-test` and `post-integration-test` phases, respectively.
  - Example:

    ```
    mvn spring-boot:start
    mvn spring-boot:stop
    ```

- **`spring-boot:build-info`**
  - Generates a `build-info.properties` file containing build information (e.g., build time, version).
  - This information can be accessed in your application using Spring Boot's `BuildProperties` bean or `@Value` annotations.
  - Command:

    ```
    mvn spring-boot:build-info
    ```

---

4. Customize Plugin Configuration (Optional)

You can customize the behavior of the `spring-boot-maven-plugin` by adding configuration options in the `pom.xml`. Below are some common customizations:

- **Specify the Main Class**:
  - If the plugin cannot automatically detect the main class, specify it manually.
  - Example:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <mainClass>com.example.MyApplication</mainClass>
            </configuration>
        </plugin>
    </plugins>
</build>
```

- **Exclude Dependencies from the Fat JAR**:
  - Exclude dependencies that are provided by the runtime environment (e.g., an external servlet container).
  - Example:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>com.example</groupId>
                        <artifactId>some-dependency</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
```

- **Set Application Arguments**:
  - Configure arguments to pass to the application when running with `spring-boot:run`.
  - Example in `pom.xml`:

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <arguments>
                    <argument>--server.port=8081</argument>
                </arguments>
            </configuration>
        </plugin>
    </plugins>
</build>
```

  - Alternatively, pass arguments via the command line:

```
mvn spring-boot:run -Dspring-boot.run.arguments=--server.port=8081
```

- **Building WAR Files**:
  - If building a WAR file for deployment to an external servlet container, ensure your project packaging is set to `war` in `pom.xml`:

```xml
<packaging>war</packaging>
```

  - You may also need to exclude embedded servlet containers (e.g., Tomcat) if provided by the environment. Add the following dependency as `provided`:

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

---

5. Key Notes

- **Default Behavior**:
  - When you run `mvn package`, the plugin automatically repackages the JAR/WAR during the `package` phase to make it executable.
  - The repackaged artifact is the main artifact, and the original JAR/WAR is attached with a classifier (e.g., `original`).

- **Development Workflow**:
  - Use `mvn spring-boot:run` for quick development and testing.
  - Use `mvn package` to build a deployable executable JAR or WAR.
- **Integration Tests**:
  - Use `spring-boot:start` and `spring-boot:stop` to manage the application lifecycle during integration testing.

Summary

To use the `spring-boot-maven-plugin`: 1. Ensure your project is a Spring Boot project, typically by extending `spring-boot-starter-parent` or using `spring-boot-dependencies` BOM. 2. Add the plugin to the `<build><plugins>` section of your `pom.xml`. 3. Use Maven commands like: - `mvn spring-boot:run` to run the application. - `mvn package` to build an executable JAR/WAR. - Other goals like `spring-boot:start`, `spring-boot:stop`, and `spring-boot:build-info` as needed. 4. Optionally, customize the plugin's configuration in `pom.xml` for advanced use cases.

This setup enables you to efficiently build, run, and manage Spring Boot applications using Maven.