Spring Boot Maven Plugin

Para usar el `spring-boot-maven-plugin` en un proyecto de Spring Boot basado en Maven, sigue estos pasos para configurar y utilizar el plugin para construir, ejecutar y gestionar tu aplicación. A continuación se presenta una guía completa:

---

1. Asegúrate de que tu proyecto sea un proyecto de Spring Boot

Antes de usar el `spring-boot-maven-plugin`, confirma que tu proyecto está configurado como un proyecto de Spring Boot. Esto generalmente implica:

- **Extender de `spring-boot-starter-parent` (Recomendado)**:
  - En tu `pom.xml`, establece `spring-boot-starter-parent` como el padre para gestionar las dependencias y versiones del plugin de Spring Boot.
  - Ejemplo:

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.4</version> <!-- Reemplaza con tu versión de Spring Boot -->
    <relativePath/> <!-- Buscar el padre en el repositorio -->
</parent>
```

- **Alternativamente, usando `spring-boot-dependencies` BOM (Bill of Materials)**:
  - Si no puedes usar `spring-boot-starter-parent`, importa el `spring-boot-dependencies` BOM en la sección `dependencyManagement`.
  - Ejemplo:

```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>2.5.4</version> <!-- Reemplaza con tu versión de Spring Boot -->
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Usar `spring-boot-starter-parent` es recomendado por su simplicidad, ya que gestiona automáticamente las versiones de los plugins.

---

2. Añade el `spring-boot-maven-plugin` a tu `pom.xml`

Para usar el plugin, necesitas declararlo en la sección `<build><plugins>` de tu `pom.xml`.

- **Si usas `spring-boot-starter-parent`:**
  - Añade el plugin sin especificar la versión, ya que es gestionada por el padre.
  - Ejemplo:

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

- **Si no usas `spring-boot-starter-parent`:**
  - Especifica la versión explícitamente, coincidiendo con la versión de Spring Boot en uso.
  - Ejemplo:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>2.5.4</version> <!-- Reemplaza con tu versión de Spring Boot -->
        </plugin>
    </plugins>
</build>
```

---

3. Utiliza los objetivos del plugin

El `spring-boot-maven-plugin` proporciona varios objetivos para ayudar a construir, ejecutar y gestionar tu aplicación Spring Boot. A continuación se presentan los objetivos más comúnmente utilizados:

- `spring-boot:run`
  - Ejecuta tu aplicación Spring Boot directamente desde Maven usando un servidor web incrustado (por ejemplo, Tomcat).
  - Útil para desarrollo y pruebas.
  - Comando:

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

- `spring-boot:repackage`
  - Reempaqueta el archivo JAR o WAR generado por `mvn package` en un "fat JAR"o WAR ejecutable que incluye todas las dependencias.
  - Este objetivo se ejecuta automáticamente durante la fase `package` si el plugin está configurado.
  - Comando:

    ```
    mvn package
    ```

  - Después de ejecutar, puedes iniciar la aplicación con:

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

- `spring-boot:start` **y** `spring-boot:stop`
  - Usados para pruebas de integración para iniciar y detener la aplicación durante las fases `pre-integration-test` y `post-integration-test`, respectivamente.
  - Ejemplo:

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

- `spring-boot:build-info`
  - Genera un archivo `build-info.properties` que contiene información de construcción (por ejemplo, hora de construcción, versión).
  - Esta información puede ser accedida en tu aplicación usando el bean `BuildProperties` de Spring Boot o las anotaciones `@Value`.
  - Comando:

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

---

4. Personaliza la configuración del plugin (Opcional)

Puedes personalizar el comportamiento del `spring-boot-maven-plugin` añadiendo opciones de configuración en el `pom.xml`. A continuación se presentan algunas personalizaciones comunes:

- **Especificar la clase principal**:
  - Si el plugin no puede detectar automáticamente la clase principal, especifícala manualmente.
  - Ejemplo:

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

- **Excluir dependencias del fat JAR**:
  - Excluye dependencias que son proporcionadas por el entorno de ejecución (por ejemplo, un contenedor servlet externo).
  - Ejemplo:

```
<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>
```

- **Establecer argumentos de la aplicación**:
  - Configura argumentos para pasarlos a la aplicación cuando se ejecuta con `spring-boot:run`.
  - Ejemplo en `pom.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>
```

  - Alternativamente, pasa argumentos vía la línea de comandos:

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

- **Construir archivos WAR**:
  - Si construyes un archivo WAR para su despliegue en un contenedor servlet externo, asegúrate de que el empaquetado de tu proyecto esté configurado como `war` en `pom.xml`:

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

  - También podrías necesitar excluir contenedores servlet incrustados (por ejemplo, Tomcat) si son proporcionados por el entorno. Añade la siguiente dependencia como `provided`:

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

---

5. Notas clave

- **Comportamiento predeterminado**:

  - Cuando ejecutas `mvn package`, el plugin reempaqueta automáticamente el JAR/WAR durante la fase `package` para hacerlo ejecutable.
  - El artefacto reempaquetado es el artefacto principal, y el JAR/WAR original se adjunta con un clasificador (por ejemplo, `original`).

- **Flujo de trabajo de desarrollo**:
  - Usa `mvn spring-boot:run` para desarrollo y pruebas rápidas.
  - Usa `mvn package` para construir un JAR o WAR ejecutable desplegable.

- **Pruebas de integración**:
  - Usa `spring-boot:start` y `spring-boot:stop` para gestionar el ciclo de vida de la aplicación durante las pruebas de integración.

Resumen

Para usar el `spring-boot-maven-plugin`: 1. Asegúrate de que tu proyecto sea un proyecto de Spring Boot, generalmente extendiendo `spring-boot-starter-parent` o usando `spring-boot-dependencies` BOM. 2. Añade el plugin a la sección `<build><plugins>` de tu `pom.xml`. 3. Usa comandos Maven como: - `mvn spring-boot:run` para ejecutar la aplicación. - `mvn package` para construir un JAR/WAR ejecutable. - Otros objetivos como `spring-boot:start`, `spring-boot:stop` y `spring-boot:build-info` según sea necesario. 4. Opcionalmente, personaliza la configuración del plugin en `pom.xml` para casos de uso avanzados.

Esta configuración te permite construir, ejecutar y gestionar aplicaciones Spring Boot de manera eficiente usando Maven.