1. Change name of application
2. Add plugin

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.2.0</version>

<executions>

<execution>

<goals>

<goal>test-jar</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

1. Copy json file to an external folder
2. mvn resources:copy-resources -DoutputDirectory=target/classes -Dresources.directory=json -Dresources.includes=\*\*/\* & mvn package
3. Copy jar-test created in lib folder.
4. Delete test folder
5. Install in local maven repository

mvn install:install-file -Dfile=path/to/your/jarfile.jar -DgroupId=com.example -DartifactId=example-artifact -Dversion=1.0.0 -Dpackaging=jar

1. Change project name
2. Remove plugin dependency
3. Add external jar dependency

<dependencies>

<dependency>

<groupId>com.example</groupId>

<artifactId>example-artifact</artifactId>

<version>1.0.0</version>

</dependency>

</dependencies>

Hosting on GIT

**Step 1: Prepare Your GitHub Repository**

1. **Create a New Repository**: Go to GitHub and create a new repository where you will host your JAR files. Make sure the repository is set to public if you want to share it with others.
2. **Clone the Repository Locally**: Clone the repository to your local machine. For example:

shCopy code

git clone https://github.com/yourusername/your-repository.git

**Step 2: Add Your JAR File to the Repository**

1. **Prepare the Directory Structure**: Maven repositories have a specific directory structure based on the groupId, artifactId, and version of the artifact. Create directories in your cloned repository to match this structure. For example, if your groupId is **com.example**, artifactId is **mylibrary**, and version is **1.0.0**, the structure should be:

bashCopy code

/com/example/mylibrary/1.0.0/

1. **Copy Your JAR and POM File**: Place your JAR file and a POM file describing the project in the appropriate version directory. The POM file is not strictly necessary but recommended as it can contain valuable metadata about the project.

**Step 3: Generate Maven Metadata**

Maven requires metadata files to resolve dependencies properly. You can generate these manually or use Maven commands to generate them for you. To do this manually, you will need to create a **maven-metadata.xml** file in your artifact's directory that looks something like this:

xmlCopy code

<metadata> <groupId>com.example</groupId> <artifactId>mylibrary</artifactId> <versioning> <release>1.0.0</release> <versions> <version>1.0.0</version> </versions> <lastUpdated>yyyyMMddHHmmss</lastUpdated> </versioning> </metadata>

Replace the placeholder values with your actual groupId, artifactId, and version. The **lastUpdated** field should be the current timestamp in the format **yyyyMMddHHmmss**.

**Step 4: Push to GitHub**

After adding your JAR file, POM file, and metadata, commit and push these changes to your GitHub repository:

shCopy code

git add . git commit -m "Add mylibrary JAR" git push origin main

**Step 5: Use Your GitHub Repository as a Maven Repository**

To use your GitHub-hosted Maven repository in a project, you need to add it as a repository in your **pom.xml** or **build.gradle** file:

**Maven pom.xml:**

xmlCopy code

<repositories> <repository> <id>my-maven-repo</id> <url>https://raw.githubusercontent.com/yourusername/your-repository/main/</url> </repository> </repositories>