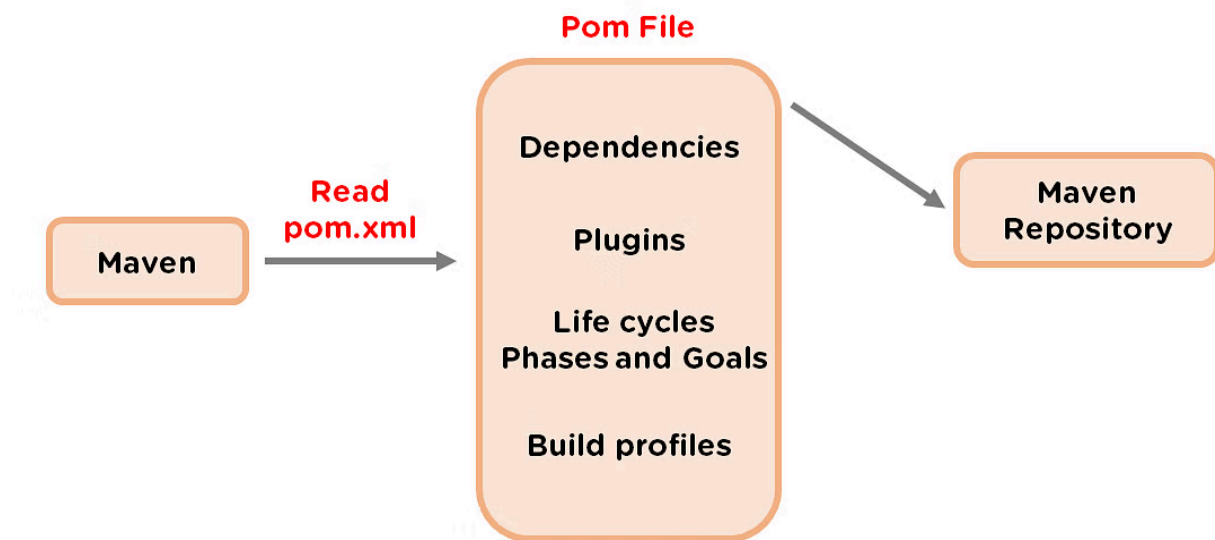


Maven™

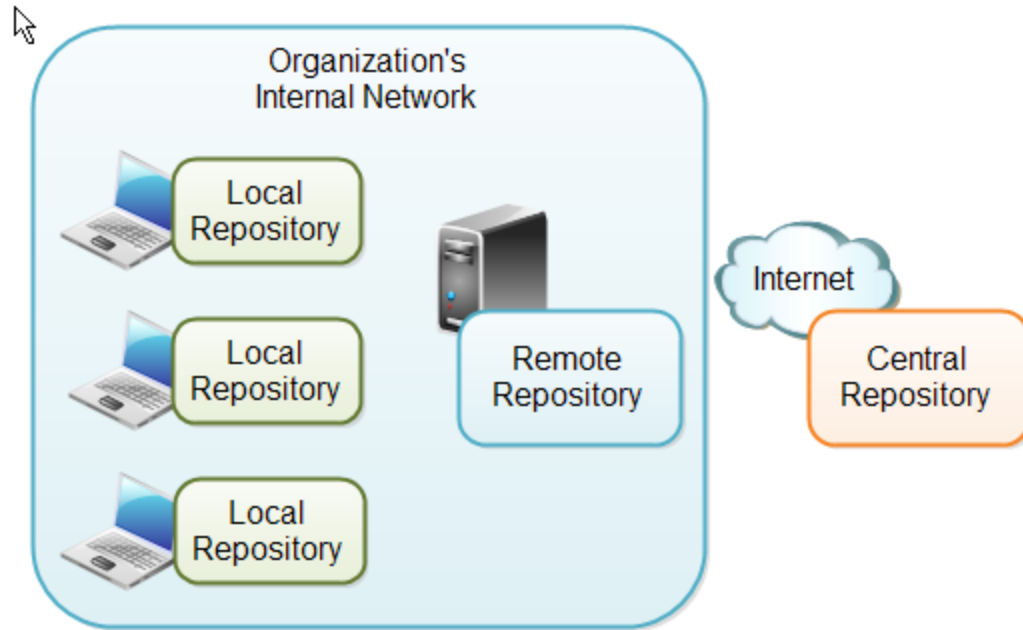
1. What is Maven?

Apache Maven is a **build** automation and **project management tool** used primarily for Java projects. It uses a POM (Project Object Model) file to define project structure, dependencies, build settings, and plugins.



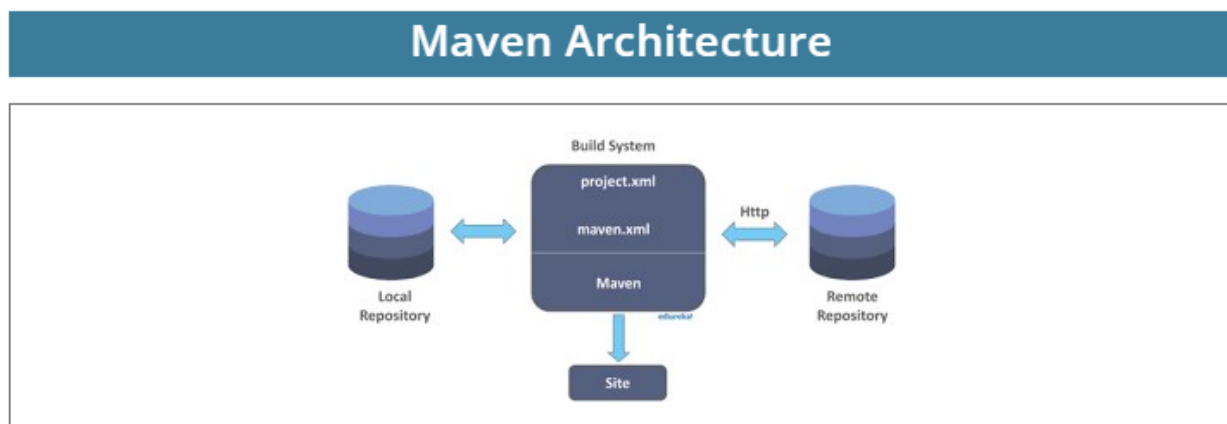
Core Concepts:

- **POM (Project Object Model):** The POM.xml file is the blueprint of your project. It contains data about your project and configuration details that Maven uses to build your project.
- **Dependencies:** Maven manages libraries that your project requires in its build path. It **automatically downloads** these libraries from the **Maven central repository** or other specified repositories.
- **Plugins:** These are extensions used to perform specific tasks within the **build process, such as compiling code or creating documentation.**
- **Lifecycles and Phases:** Maven's build process is divided into lifecycles (**default, clean, and site**), each consisting of different phases (compile, test, package, install, deploy, etc.).



Central Repository:

<https://mvnrepository.com/artifact/com.mpatric/mp3agic/0.9.0>



2. Maven Setup (Maven Step-Step)

Step 1: Prerequisites

Install Java JDK (8+)

```
java -version  
javac -version
```

Step 2: Download Maven

- Official Maven Download

Extract and set MAVEN_HOME:

```
export MAVEN_HOME=/opt/apache-maven-3.x.x  
export PATH=$PATH:$MAVEN_HOME/bin
```

Step 3: Verify Installation

```
mvn -version
```

3. Maven Project Structure

```
my-app/  
|  
├─ pom.xml  
├─ src/  
|   ├─ main/  
|   |   └─ java/  
|   |       └─ com/example/App.java  
|   └─ test/  
|       └─ java/  
|           └─ com/example/AppTest.java
```

4. POM.xml – Heart of Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

5. Packaging and Build

The Packaging Process

- 1. Define Packaging Type:** In your project's POM file, you specify the packaging type with the `<packaging>` tag. Common types include `jar`, `war`, and `ear`, each serving different project needs.
- 2. Execution of Lifecycles:** When you initiate a Maven build (for example, using the command `mvn package`), Maven starts

executing the default lifecycle up to the package phase. This includes compiling source code, executing tests, and finally, packaging the compiled code along with resources into the specified format.

3. **Dependency Management:** Maven automatically includes all necessary dependencies into the final package. It resolves dependencies specified in the POM file, ensuring that your project has all it needs to run successfully.
4. **Plugin Execution:** Various plugins are executed during the packaging process. For instance, the `maven-jar-plugin` is used for jar packaging, while the `maven-war-plugin` is utilized for WAR packaging. These plugins are configurable in the POM file, allowing customization of the packaging process.
5. **Result:** The output is a packaged file, located in the `target` directory of your project, ready for distribution or deployment.

Advantages of Maven Packaging

- **Standardization:** Maven promotes a standard directory structure and a uniform build process, making it easier for teams to understand and collaborate on projects.
- **Automation:** It automates tedious tasks like dependency management and package creation, saving time and reducing the potential for errors.
- **Flexibility:** Maven's extensive plugin ecosystem allows for customization of the build process to suit various project needs.

Sample pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>my-app</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
```

```

        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Example dependency -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Maven Compiler Plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>

```

Common Maven Commands:

Task	Command
Create project	mvn archetype:generate
Compile source	mvn compile
Run tests	mvn test
Package JAR/WAR	mvn package
Install to local repo	mvn install
Clean build dir	mvn clean

Run the app (plugin)	mvn exec:java
----------------------	---------------

Sample Jar Packaging

```
mvn clean package
```

Output:

target/my-app-1.0.0.jar

6. Maven Plugins You Should Know

Plugin	Use
maven-compiler-plugin	Compiles Java source
maven-surefire-plugin	Runs unit tests
maven-jar-plugin	Creates JAR files
maven-war-plugin	Creates WAR files
maven-dependency-plugin	Manages transitive dependencies
maven-site-plugin	Generates project documentation
maven-checkstyle-plugin	Checks code style compliance

7. Unit Testing with Maven

```
<dependency>
```

```
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.13.2</version>
<scope>test</scope>
</dependency>
```

Command:

```
mvn test
```

Activity	Tool / Tip
Dependency Management	Use dependency scopes and versions
Avoid Snapshots	Use stable versions for release
Build Profiles	Use <profiles> in pom.xml
Clean Workspace	Use mvn clean
Integration CI/CD	Jenkins, GitHub Actions, GitLab CI
Artifact Storage	Nexus, JFrog Artifactory

8. Maintenance Tips

9. Advanced Features

Multi-Module Projects

Parent POM controls submodules:

```
<modules>
  <module>core</module>
  <module>web</module>
</modules>
```

Maven Profiles

Customize build for different environments:

```
<profiles>
  <profile>
    <id>dev</id>
    <properties>
      <env>development</env>
    </properties>
  </profile>
</profiles>
```

Run with:

```
mvn clean install -P dev
```

10. Tools and Ecosystem

Tool	Purpose
IntelliJ IDEA / Eclipse	IDEs with Maven support
Jenkins	CI/CD pipeline
SonarQube	Code quality analysis
Nexus / Artifactory	Artifact repository manager
Docker	Containerize Maven builds
GitHub Actions	Automated Maven workflows

Use the DEV or DigitalOcean cheat-sheet for reference during the session DEV CommunityDigitalOcean.

Project setup with archetype → mvn archetype:generate

POM essentials → groupId, artifactId, version, dependencies, plugins

Directory structure conventions and why they matter
WikipediaWikipedia

Lifecycle demo: clean → compile → test → package → install

Dependency tree & scope exploration


Profile usage / Settings customization (briefly mention)

Multi-module POM/project setup overview

Plugin example — generating sources with plugin execution in generate-sources phase maven.apache.org

Quiz

Q1: What file does Maven use to define dependencies?

- a) settings.xml
- b) pom.xml 
- c) build.gradle
- d) makefile

Q2: Which goal creates a .jar file?

- a) mvn install
- b) mvn package ☒
- c) mvn compile
- d) mvn generate-sources

Q3: What is the Maven local repository path by default?

- a) /usr/lib/maven
 - b) ~/.maven
 - c) ~/.m2/repository ☒
 - d) ~/maven-repo
-