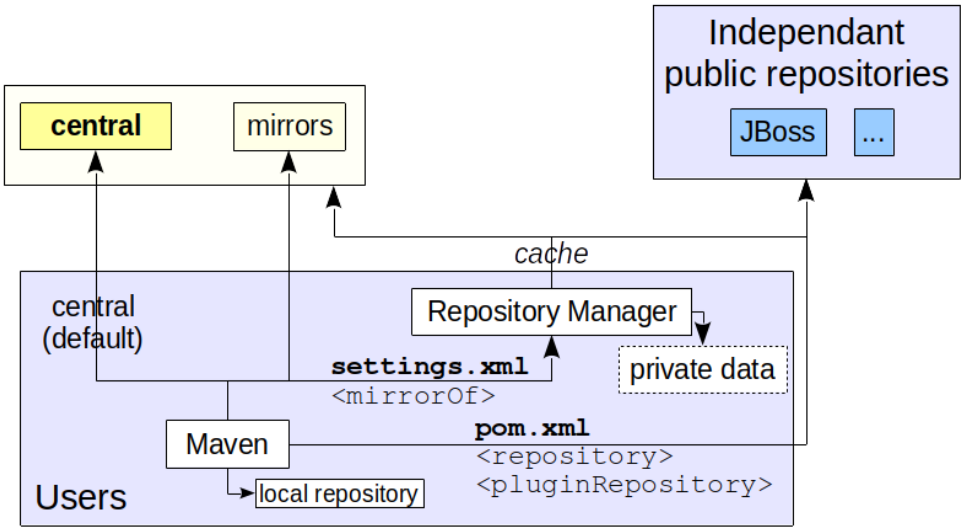
UNIT II COMPILE AND BUILD USING MAVEN & GRADLE 6

Introduction, Installation of Maven, POM files, Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles, Maven repositories(local, central, global),Maven plugins, Maven create and build Artifacts, Dependency management, Installation of Gradle, Understand build using Gradle

---

MAVEN REPOSITORIES

Apache Maven uses repositories to store artifacts. It stores all the project jars, plugins, library jars and can be accessed easily by maven. The dependencies are downloaded from repositories, and artifacts we build are stored (installed, uploaded) into repositories as well. This is one of the fundamental concepts of Maven.



A repository in Maven holds build artifacts of varying types. There are exactly two types of repositories: local and remote:

- **The local repository** is a directory on the computer where Maven runs. It caches remote downloads and contains temporary build artifacts. It is usually inside the home directory (.m2). The location of the local repository can be changed in settings.xml file using **localRepository** element.

- **Remote repositories** refer to any other type of repository, accessed by a variety of protocols such as file:// and https://. Artifacts are deployed from remote locations. It acts as the central repository. These repositories might be a truly remote repository set up by a third party to provide their artifacts for downloading (for example, repo.maven.apache.org).

Local and remote repositories are structured the same way so that scripts can run on either side, or they can be synced for offline use. The layout of the repositories is completely transparent to the Maven user.

MAVEN PLUGINS

Maven plugins are essential components in the Apache Maven build system, designed to extend its functionality. They perform tasks such as compilation, testing, packaging, deployment, and others.

Maven plugins are divided into two main types:

1. Build Plugins
2. Reporting Plugins

1. Build Plugins

These plugins are executed during the build process, and they are defined in the <build> section in the pom.xml file. For example, the Compiler Plugin, Surefire Plugin, and Assembly Plugin.

Common Build Plugins:

- **Compiler Plugin:** Compiles Java source code into bytecode. maven-clean-plugin → deletes old files
- **Surefire Plugin:** Runs unit tests using frameworks like JUnit or TestNG. maven-surefire-plugin → runs tests
- **Failsafe Plugin:** Executes integration tests, typically for verifying larger system components. maven-failsafe-plugin → runs tests
- **JAR Plugin:** Packages compiled code into a JAR (Java Archive) file. maven-jar-plugin → runs tests

- **WAR Plugin:** Creates a WAR (Web Archive) file for web applications. maven-war-plugin → runs tests
- **install plugin:** Install the build artifact into the local repository maven-install-plugin → puts .jar in local repo
- **Spring Boot Maven Plugin:** Packages Spring Boot applications into executable JARs or WARs. spring-boot-maven-plugin

2. Reporting Plugins

These plugins are used for generating reports about the project. These are defined in the <reporting> section in the pom.xml file. Examples include the Surefire Report Plugin for generating test reports and the Javadoc Plugin for generating API documentation.

Example 1: Build Plugins

Here we created a Maven project with basic dependencies. Below we provide the pom file for your reference.

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

Example 2: Reporting Plugins

And here, we use java doc reporting plugin for generating project report.

```
<reporting>
    <plugins>
      <!-- Javadoc Plugin -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>3.3.1</version>
        <configuration>
          <source>1.8</source>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
```

DEPENDENCY MANAGEMENT

In Maven, dependency management refers to how your project handles external libraries (dependencies) it needs to compile and run. Maven uses a central repository to download these libraries and places them in your local repository (.m2 folder). The project dependencies are collectively specified in pom.xml file, inside **<dependencies>** element. When we run a Maven project or execute a maven goal, these dependencies are resolved and loaded from the local repository. If these dependencies are not in the local repository, then Maven will download them from the remote repository and cache them in the local repository.

Declaring Dependencies (in pom.xml)

Add dependencies inside the <dependencies> tag:

xml

CopyEdit

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.7.3</version>
  </dependency>
</dependencies>
```

Transitive Dependencies

Transitive dependency means when library A depends upon library B and Library B depends upon Library C, then A depends upon both B and C. The command **dependency:tree** is used to view all the transitive dependencies.

Dependency Scope

Defines where the dependency is used:

Scope	Used For
compile	Default – available everywhere
provided	Provided by container (e.g., Tomcat)
runtime	Needed at runtime only
test	Needed only for unit testing
system	Similar to provided, but local
import	For importing dependencies from BOM

Example:

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

Dependency Management Section

This is used when there are multi-module projects. The dependency versions defined in the parent pom.xml, will be inherited by the child modules.

```
<dependencyManagement>
  <dependencies>
```

```
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.12.0</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

In child module:

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
  </dependency>
</dependencies>
```

Version is automatically applied from the parent.

Excluding Transitive Dependencies

There can be version mismatch issues(due to transitive dependencies) between the project artifacts and the artifacts from the platform of deployments, such as Tomcat or any other server. To avoid version conflicts, Maven provides <exclusion> element to break the transitive dependency.

```
<dependency>
  <groupId>com.example</groupId>
  <artifactId>some-lib</artifactId>
  <version>1.0</version>
  <exclusions>
    <exclusion>
      <groupId>commons-logging</groupId>
```

```
    <artifactId>commons-logging</artifactId>
  </exclusion>
 </exclusions>
</dependency>
```