**Maven**

Maven is a **build automation and project management tool** primarily used for Java-based projects. It simplifies project compilation, dependency management, packaging, and deployment.

* Maven was developed by Apache company
* Maven is used for only java based projects / applications
* The tool helps get the right jar files for each project as there may be different version of separate packages.
* Easy to add new dependencies.

**Key features of Maven**

* Dependency Management- Automatically downloads required libraries from repositories
* Project structure standardization- Follow a predefined directory structure
* Build Automation- Handles compiling, testing, packaging(Jar/War/Ear), and deploying
* Plugins & Lifecycle- Uses plugins to execute tasks in predefined lifecycle phases(ex: compile , test , packages)
* POM (Project Object Model) - Configuration is done in an XML file (pom.xml), which defines dependencies, plugins, and project metadata.

**What is a Build Tool in DevOps**

A build tool in DevOps is software that automates the process of compiling source code into executable artifacts (e.g., JAR, WAR, or Docker images). It handles compilation, dependency management, packaging, testing, and deployment

**Build tools**

1. **Java –** Apache Maven , Apache ANT (older version)
2. **Python –** Pybuilder
3. **.NET –** Ms build (Microsoft build engine)
4. **Nodejs –** Guip , Grunt , Gradle , Web pack

**How Does Maven Work**

Maven is a **build automation and dependency management tool** primarily used for Java projects. It simplifies building, testing, and deploying applications by following a **standardized project structure** and using an XML-based configuration file (pom.xml).

1. POM (Project Object Model)

 The heart of a Maven project is the pom.xml file.

 It defines dependencies, plugins, and build configurations

1. Dependency Management

 Maven automatically downloads required **JAR files** (libraries) from remote repositories.

 Uses **Maven Central Repository** by default

1. Build Lifecycle

 Maven defines a sequence of **build phases** (validate → compile → test → package → install → deploy).

 Commands trigger these phases automatically

1. Plugins

 Plugins extend Maven’s functionality (e.g., compiling, testing, deploying).

 Example: maven-compiler-plugin for compilation

**Types of packages in Maven**

JAR (Java Archive)

* Default packaging type in Maven.
* Used for **standard Java applications** (Spring Boot, Microservices, Standalone apps).
* Creates a .jar file containing compiled .class files and dependencies

WAR (Web Application Archive)

* Used for **Java web applications** that run on servlet containers like **Tomcat, JBoss, or WildFly**

EAR (Enterprise Application Archive)

 Used for **Enterprise Java EE applications**.

 Combines multiple modules:

* EJB modules (.jar)
* Web applications (.war)
* Library dependencies

**Why we use pom.xml in maven**

* Maven works for pom.xml / pom.xml2 (Project object model)
* Pom.xml used to manage the java applications
* Pom.xml is also know as super pom / parent pom
* It contains all the dependency libraries
* Should be unique pom.xml
* Each project contain one pom.xml file

**What Are Plugins in Maven**

In Maven, **plugins** are components that extend the functionality of the build process. They provide specific tasks that help automate common tasks such as **compiling code**, **running tests**, **creating JAR/WAR files**, **deploying artifacts**, **generating reports**, and more.

A **plugin** consists of one or more **goals**, each of which performs a single task. These goals can be executed at different phases of the **build lifecycle**

**Two types of plugins**

1. **Build plugins**

In build plugins are used only one time

1. **Added plugins**

Added plugins are reusable

**Three types of repositories**

1. **Local Repository**

 **Location**: It is stored on your local machine, typically in the ~/.m2/repository directory.

 **Purpose**: This repository is used to store artifacts that are downloaded from remote repositories or installed manually.

 **Usage**: When you build a project, Maven checks this repository first to see if the required dependencies are already available. If not, it will fetch them from a remote repository and store them locally for future use.

1. **Central Repository**

 **Location**: This is the default repository provided by Maven (hosted by **Maven Central**).

 **Purpose**: This is the default **remote repository** from which Maven downloads dependencies if they are not available in the local repository.

 **Usage**: Maven automatically connects to this repository to fetch common libraries like JUnit, Spring, and others

1. **Remote Repository**

 **Location**: A remote repository can be hosted on an external server (e.g., a private repository for your organization, or a service like **Nexus** or **Artifactory**).

 **Purpose**: Remote repositories are used to share and distribute artifacts across different projects or teams. These repositories can host both **third-party libraries** and **internal project artifacts**.

 **Usage**: Maven will access remote repositories to search for dependencies that are not present in the local or central repositories

**Maven in devops should be utalized in 3 scenarios**

1. If the initiative has number of significant dependencies
2. If the dependencies version needs to be upgraded frequently
3. The task involves rapid documentation , compilation & building of source code as jar or zip files

**ANT:** It is the older version of maven

* Developed by apache company
* Can build any kind of projects
* It has no life cycle
* Java unit test case not there in ANT
* Script are not reusable

**Maven Lifecycle**

**1.Default:** It takes the source code from developers and perform few function

1. **validate**: Validates the project structure and checks that all necessary information is provided.
2. **compile**: Compiles the source code into .class files. ---> mvn compile
3. **test**: Runs the unit tests on the compiled code. ----> mvn test
4. **package**: Creates a distributable artifact (e.g., JAR, WAR, EAR).---> mvn package
5. **verify**: Verifies that the package is valid (e.g., checks code quality, test results).
6. **install**: Installs the artifact to the local repository (~/.m2/repository). ---> mvn install
7. **deploy**: Deploys the artifact to a remote repository (e.g., Nexus, Artifactory).

**2.Clean**

1. **pre-clean**: Executes tasks before cleaning

2. **clean**: Deletes the target/ directory to remove any previously compiled files and artifact

3. **post-clean**: Executes tasks after cleaning.

**3.** **Site**

1. **pre-site:** Executes tasks before generating the site**.**

**2. site:** Generates the project's site, including reports and documentation**.**

**3. post-site:** Executes tasks after generating the site**.**

**4. site-deploy:** Deploys the generated site to a web server or repository for access by

others