**Maven**

🡪Maven is a build automation tool.

🡪It is open source and free of cost.

🡪It is used for project management.

🡪It is build automation tool for java projects.

🡪Maven was developed by the Apache Company.

🡪Test scripts can be run in the maven.

🡪Test scripts is used for Automation testing.

🡪It manages project builds, dependencies and documentation using a **Project Object Model** file (pom.xml).

**POM (Project Object Model):**

🡪pom.xml file is developed by developers.

🡪Maven works for POM.xml/POM.xml2

🡪pom.xml used to manage the java applications.

🡪POM .xml is also known as Super POM/ Parent POM.

🡪It contains all dependency libraries.

🡪POM.xml file should be unique.

🡪Each project contains only one pom.xml file.

🡪pom.xml file describes the project’s configuration and dependency management.

**Features of Maven:**

🡪It provides a lot of features to make the development easy.

* **Dependency Management:** It simplifies the process of including third party libraries in our project. So, it can be downloaded easily.
* **Standardized Project Structure:** It provides standard project folder structure like the source code in one package, testing code in one package.
* **Build Life Cycle:** Maven has different build phases to build the project
* **Central Repository:** It uses central repository for downloading the dependencies.
* **Build Profiles:** The Maven build cycle handles different profiles like development, QA, Production etc.
* **Reporting and Documentation:** Maven can generate various types of project reports and documentation using plugins.
* **IDE Integration:** It supports different IDEs like STS, Eclipse, Intellij IDEA and NetBeans.

**What is Build tool in devops?**

🡪The build tool automates the process of compiling, testing, packaging and deploying the code in a consistent way.

Steps:

* Compile source code
* Manage dependencies
* Run unit tests
* Package output (JAR/WAR/EAR)
* Deploy artifacts

🡪An **Artifact** is a file or package generated during build process.

**Ex:** JAR, WAR, EAR, ZIP ..,etc

**Build Tools:**

* **Java –** Maven, ANT, Gradle.
* **Python –** pybuilder
* **.NET –** Msbuild
* **node.js –** npm, web pack

**How does Maven work?**

🡪Reads the pom.xml file – Loads project metadata and dependencies.

🡪Downloads Dependencies – Resolves and download from repositories.

🡪Executes build life cycle – Runs lifecycle phases.

🡪Uses Plugins – Executes specific tasks like compiling, testing and packaging.

🡪Generates output – Create JAR/WAR files and deploy artifacts.

🡪Copy the output into the TOMCAT web server.

**Workflow of Maven:**

1. **Developers writes code:**

🡪Developer creates or modifies the code in the local repository.

1. **Push code to Github:**

🡪Code is committed and pushed to the Github repository using Git.

1. **Build with Maven:**

🡪Maven downloads dependencies and compiles the code.

🡪Maven generates the artifacts(jar/war/ear) files.

1. **Package the build:**

🡪Maven packages the compiled code into the deployable format:

🡪jar – java application

🡪war – web applications

🡪ear – enterprise application

1. **Copy package to Tomcat server:**

🡪The packages artifact is copied to the Tomcat web server.

1. **End user access:**

🡪Here, the end user can access the application using browser.

**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.

🡪In maven plugins are divided into 2 types:

1. Build Plugins
2. Reporting Plugins

**Maven Build Plugins:**

🡪These are executed during the build process and these are defined in build section in the pom.xml file.

**Maven Reporting Plugins:**

🡪These plugins are used for generating the reports about the project.

🡪These are defined in the reporting section in the pom.xml file.

**Maven Repositories:**

🡪Maven uses repositories to store and manage dependencies and artifacts.

**Types of repositories:**

1. **Local Repository:**

🡪A local repository is a directory on the developer’s machine where maven stores all the artifacts resolved from the remote repositories or created by the developer.

🡪This process is done in your own laptops or systems.

🡪When a dependency is needed, Maven first check in the local repository.

🡪If the dependency is found, then it uses it. Otherwise, it fetches from the remote repository and catches it locally.

1. **Central Repository:**

🡪The central is a repository provided by maven community and maintained by the Apache Software Foundation.

🡪it contains a large number of commonly used libraries and is the default repository used by Maven, when it needs to resolve the dependencies.

🡪When a repository is not found in the local repository, Maven queries the central repository to download it.

1. **Remote Repository:**

🡪The Remote repositories are repositories other than the central repository that can be set up by third party or any organizations to host their own artifact.

🡪They are specified in the project’s pom.xml file in the Maven settings.

🡪If an artifact is not available in the local repository, then maven will check these remote repositories for the required dependencies.

**Advantages of Maven Repositories:**

* Centralized Dependency Management
* Version Control
* Consistency and Re producibility
* Local Caching
* Performance Improvement
* Custom repositories
* Integration with build tools
* Plugin Support
* Seamless Integration
* Community and support

**Maven Lifecycle:**

🡪Maven defines 3 default lifecycles with specific phases:

1. **Clean Lifecycle:**

🡪There are 3 steps:

* **pre-clean** – Setup before cleaning. Checks for jar/war/ear files.
* **clean** – It will delete the older jar/war/ear files.
* **post-clean** – The new generated jar/war/ear files will be saved.

1. **Default Lifecycle:**

🡪It takes code from the developers and perform few functions.

* **compile** – compiles entire source code
* **validate** – validates the compiled code
* **test** – tests the source code
* **package** – generates the package for code
* **install** – install all the packages generated by the package
* **verify** – It will verify the generated package
* **deploy** – Deploy artifact to remote repository

1. **Site Lifecycle:**

🡪It is just like a folder where we will deploy our application.

* **pre-site** – It receives the post-clean files.
* **site** – it receives the pre-site files.
* **post-site** – It receives the site files.
* **site-deploy** – Here, we need to publish the documentation and reports.

**Advantages of Maven:**

🡪The following are the advantages of the Maven:

* **Dependency Management:** It can handle required dependencies in the software application.
* **Consistent Project Structure:** It provides a standard project structure to understand easily by developers.
* **Build Lifecycle:** The Maven tool provides lot of build phases like verify, test and other build phases.
* **Re usability:** Maven promotes the reuse of project components through the use of dependencies and plugins.
* **Integration and IDEs:** It supports IDEs like STS, Eclipse, IntelliJ IDEA and NetBeans.
* **Automated testing:** The Maven provide testing commends for testing the source code.

**ANT (Another Neat Tool):**

🡪Apache ant is a java-based build tool.

🡪Ant is a predecessor to Maven.

🡪It is also build automation tool.

🡪It was developed by Apache company.

🡪It has no lifecycle

🡪It has build.xml file, developed by developers.

🡪junit test cases are not there in the Ant.

🡪Test scripts are not reusable

🡪It requires manual configuration.

🡪Does not manage dependencies automatically.

🡪Supports plugins and custom tasks.

**Advantages of ANT:**

🡪Simple and easy to use

🡪Highly customizable.

🡪Works well with any version of java

🡪Supports platform independent builds.

**Limitations of ANT:**

🡪No built-in dependency management

🡪Requires writing extensive xml

🡪No default lifecycle (like maven).

**Difference between ANT and MAVEN:**

|  |  |
| --- | --- |
| **ANT** | **MAVEN** |
| 🡪Uses XML-based build.xml file | 🡪Uses POM(Project Object Model) file pom.xml |
| 🡪No built-in dependency management. | 🡪Built-in dependency management through Maven central |
| 🡪No default lifecycle | 🡪Predefined build lifecycle (clean, compile, test, package, install, deploy) |
| 🡪No project structure, developer defines structure | 🡪Follows a standard directory structure |
| 🡪High flexibility, tasks defined manually | 🡪Less flexible, follows maven conventions |
| 🡪Supports custom tasks using plugins | 🡪Supports wide range of built-in and third-party plugins |
| 🡪Custom tasks written in java | 🡪Custom plugins can be created |
| 🡪Must define output manually | 🡪Outputs follow Maven conventions |
| 🡪Supported in most IDEs | 🡪Better integration with modern IDEs |
| 🡪No built-in reporting | 🡪Built-in reporting |
| 🡪Older and less popular now | 🡪More widely used for java projects. |