

# 1. Maven Lifecycle

Maven follows a well-defined build lifecycle, which consists of several phases that help in compiling, testing, packaging, and deploying Java-based projects. The primary build lifecycles in Maven are:

## a. Default Lifecycle (Build Process)

This is the most commonly used lifecycle, containing phases like:

- **validate** – Validates project and necessary information.
- **compile** – Compiles the source code of the project.
- **test** – Runs unit tests.
- **package** – Packages the compiled code into a distributable format (JAR/WAR).
- **verify** – Verifies the integration tests.
- **install** – Installs the package into the local repository.
- **deploy** – Deploys the package to a remote repository.

## b. Clean Lifecycle

- **pre-clean** – Performs pre-cleaning actions.
- **clean** – Deletes the target directory.
- **post-clean** – Executes any necessary cleanup tasks.

## c. Site Lifecycle

- **site** – Generates project documentation.
- **site-deploy** – Deploys the generated documentation.

# 2. What is pom.xml File and Why We Use It?

## a. Definition

The `pom.xml` (Project Object Model) file is the core configuration file of a Maven project. It contains information about the project, dependencies, plugins, and build configurations.

## b. Why We Use It?

- Defines dependencies for automatic downloading.
- Configures plugins and build execution.
- Specifies project structure and properties.
- Manages different environments using profiles.

## c. Sample pom.xml Structure:

```
<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-project</artifactId>
<version>1.0.0</version>
<packaging>jar</packaging>

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

### 3. How Dependencies Work?

- Dependencies in Maven are external libraries required by the project.
- Declaring dependencies in `pom.xml` allows Maven to fetch them automatically from repositories.
- Maven follows a dependency resolution mechanism where it checks for dependencies in:
  1. Local Repository (`.m2/repository` folder)
  2. Central Repository (<https://repo.maven.apache.org/maven2>)
  3. Remote Repositories (custom repositories defined in `pom.xml`)

### 4. Checking the Maven Repository

The **Maven Central Repository** can be accessed at: <https://mvnrepository.com/>

To check dependencies:

```
mvn dependency:tree
```

### 5. How All Modules Build Using Maven?

Maven follows a **multi-module build** approach, where multiple modules (sub-projects) are managed within a single parent project.

- A **parent project** contains multiple **child modules**.

- The `pom.xml` in the parent directory defines configurations for all modules.
- Each module has its own `pom.xml` with specific configurations.

To build all modules:

```
mvn clean install
```

## 6. Can We Build a Specific Module?

Yes, a specific module can be built using:

```
mvn clean install -pl module-name -am
```

Where:

- `-pl module-name`: Specifies the module to build.
- `-am`: Builds required dependencies for the module.

## 7. Role of `ui.apps`, `ui.content`, and `ui.frontend` Folder in AEM

- **ui.apps**: Contains code related to AEM components, templates, and client libraries.
- **ui.content**: Stores content package (pages, configurations) that need to be deployed.
- **ui.frontend**: Contains frontend-related assets (JavaScript, CSS, and React code) for the AEM application.

## 8. Why Are We Using Run Mode?

Run modes in AEM define different configurations based on the environment (e.g., development, staging, production).

Configurations can be defined under:

```
/apps/my-project/config.author
/apps/my-project/config.publish
```

- 
- AEM selects configurations based on the environment's run mode.

To check the current run mode:

```
http://localhost:4502/system/console/status-slingsettings
```

## 9. What is the Publish Environment?

- The **publish environment** in AEM is responsible for serving content to end users.
- It contains **approved content** and does not allow direct modifications.
- Content is pushed from the **author instance** to the **publish instance** using **replication**.

## 10. Why Are We Using Dispatcher?

The **AEM Dispatcher** is used for caching and security purposes.

### Roles of Dispatcher:

1. **Caching:** Stores static content to reduce load on AEM.
2. **Load Balancing:** Distributes traffic across multiple instances.
3. **Security:** Blocks unauthorized requests before they reach AEM.

Dispatcher is configured using **dispatcher.any** and **rewrite rules**.

## 11. From Where Can We Access CRX/DE?

CRX/DE is the AEM content repository where developers can manage nodes and content.

### To access CRX/DE, use:

- **Author Instance:** <http://localhost:4502/crx/de/>
- **Publish Instance:** <http://localhost:4503/crx/de/>

Here, developers can browse, edit, and create JCR nodes.