# Adobe Experience Manager

1. Maven Lifecycle

What is the Maven Lifecycle?

Maven follows a build lifecycle to standardize the process of building, testing, and deploying projects. The lifecycle consists of several phases:

Key Phases in the Maven Lifecycle

- Validate: Validates the project structure and ensures all necessary information is available.
- Compile: Compiles the source code of the project.
- Test: Runs unit tests using a suitable testing framework (e.g., JUnit).
- Package: Packages the compiled code into a distributable format (e.g., JAR, WAR, AEM package).
- Verify: Runs integration tests to ensure the package is valid.
- Install: Installs the package into the local Maven repository for use in other projects.
- Deploy: Copies the package to a remote repository for sharing with other developers or deployment.

AEM-Specific Lifecycle

In AEM, Maven is used to:

Build AEM content packages (e.g., .zip files).

Deploy packages to AEM instances (author or publish).

2. What is pom.xml and Why We Use It

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

Key Elements in pom.xml

- Project Metadata: Group ID, artifact ID, version, and packaging type.
- Dependencies: Lists all libraries and modules required for the project.
- Build Configuration: Specifies plugins, resources, and build settings.
- Parent POM: Inherits configuration from a parent POM (used in multi-module projects).
- Properties: Defines reusable variables.

Why We Use pom.xml in AEM

- It standardizes the build process across different environments.

- It manages dependencies and ensures consistency.
- It simplifies project configuration and sharing.

3. How Dependencies Work

Dependency Management

Maven downloads dependencies from a central or remote repository. Dependencies are declared in the <dependencies> section of the pom.xml file.

Dependency Scope

- Compile: Default scope; available in all classpaths.
- Test: Only available for test compilation and execution.
- Provided: Provided by the runtime environment (e.g., servlet API).
- Runtime: Required for execution but not for compilation.
- System: Similar to provided but requires an explicit path.
- Transitive Dependencies
- Maven automatically resolves and downloads transitive dependencies (dependencies of dependencies).

4. Maven Repository

Types of Repositories

- Local Repository: Located on the developer's machine (~/.m2/repository).
- Central Repository: Maven's default repository for public libraries.
- Remote Repository: Custom repositories hosted by organizations (e.g., Adobe AEM repository).
- Accessing Repositories

Maven searches for dependencies in the following order:

- Local repository
- Central repository
- Remote repositories (if configured in pom.xml or settings.xml).

5. Building All Modules Using Maven

Multi-Module Projects

AEM projects are typically multi-module, with the following structure:

- ui.apps: Contains components, templates, and client libraries.

- ui.content: Contains content structures (e.g., pages, policies).
- ui.frontend: Contains front-end code (e.g., JavaScript, CSS).

Building All Modules

Run the following command from the root directory:

- bash
- Copy
- mvn clean install

6. Building a Specific Module

Building a Single Module

Navigate to the module's directory and run:

- bash
- Copy
- mvn clean install

Alternatively, use the -pl (project list) flag:

- bash
- Copy
- mvn clean install -pl ui.apps

7. Role of ui.apps, ui.content, and ui.frontend Folders

ui.apps

- Contains AEM components, templates, and client libraries.
- Stores Java code for OSGi services and servlets.
- Example: /apps/myproject/components.

ui.content

- Contains content structures like pages, policies, and templates.
- Example: /conf/myproject/settings.

ui.frontend

- Contains front-end code (e.g., JavaScript, CSS, and assets).
- Often integrated with tools like Webpack or Gulp for front-end builds.

8. Why We Use Run Modes

Run Modes in AEM

- Run modes allow AEM instances to behave differently based on the environment (e.g., author, publish, development).
- Configurations specific to a run mode are applied dynamically.

Common Run Modes

- Author: Used for content creation and management.
- Publish: Used for serving content to end-users.
- Development: Used for local development and testing.

9. What is the Publish Environment?

Publish Environment

- The publish environment is where the final content is served to end-users.
- It is optimized for performance and security.
- Content is replicated from the author environment to the publish environment.

10. Why We Use Dispatcher

Dispatcher

- Dispatcher is a caching and load-balancing tool for AEM. It sits between the end-user and the publish instance.

Key Functions

- Caching: Improves performance by caching static content.
- Security: Acts as a reverse proxy to protect the publish instance.
- Load Balancing: Distributes traffic across multiple publish instances.

11. Accessing CRX/DE

- CRX/DE (Content Repository Extreme/Development Environment)
- CRX/DE is a web-based interface for managing the AEM content repository.

Accessing CRX/DE

- Navigate to http://<host>:<port>/crx/de (e.g., http://localhost:4502/crx/de).
- Use admin credentials to log in.

Use Cases

- View and edit content nodes.

- Manage OSGi configurations.
- Debug and troubleshoot issues.