AEM Assignment - 01

**DATE : 18-03-2025**

1. **Maven Life Cycle**

Maven operates on a build lifecycle, which is a well-defined sequence of phases that dictate how a project is built. The three main lifecycles in Maven are:

Default: Handles project building and deployment. Key phases include:

* validate: Checks if the project is correct and all necessary information is available.
* compile: Compiles the source code.
* test: Runs unit tests.
* package: Packages the compiled code into a distributable format (e.g., JAR or ZIP).
* install: Installs the package into the local repository.
* deploy: Deploys the package to a remote repository.
* Clean: Removes all files generated by the previous build (e.g., clean phase).
* Site: Generates project documentation (e.g., site phase).

In AEM, the default lifecycle is commonly used to build and deploy artifacts like OSGi bundles and content packages to an AEM instance.

1. **What is pom.xml file and why we use it?**

The pom.xml (Project Object Model) file is the core configuration file in a Maven project. It defines:

* Project Metadata: Group ID, artifact ID, version, etc.
* Dependencies: Libraries or modules the project relies on.
* Build Configuration: Plugins, goals, and profiles for building the project.
* Modules: Sub-projects (in a multi-module setup).

In AEM, the pom.xml is critical because it:

* Manages dependencies like the AEM Uber JAR or third-party libraries.
* Configures plugins (e.g., filevault-package-maven-plugin) to build AEM-specific artifacts (content packages, bundles).
* Specifies how to deploy these artifacts to AEM instances (e.g., via profiles like autoInstallPackage).

1. **How dependencies work?**

Dependencies in Maven are external libraries or modules a project needs. They are declared in the <dependencies> section of the pom.xml. Here’s how they work:

* **Declaration**: Specify groupId, artifactId, and version (e.g., AEM Core Components).
* **Resolution**: Maven downloads these dependencies from repositories (e.g., local .m2 repository or remote repositories like Maven Central or Adobe’s public repository).
* **Transitive Dependencies**: Maven automatically resolves and downloads dependencies of your dependencies.
* **Scope**: Defines when a dependency is needed (e.g., compile, test, provided).

In AEM, dependencies like the AEM Uber JAR (com.adobe.aem:uber-jar) provide APIs for development, while OSGi bundles or content packages might be embedded as dependencies in the all module.

1. **Check the Maven repository.**

The Maven repository is where dependencies and artifacts are stored. There are two types:

* **Local Repository**: Located at ~/.m2/repository on your machine. It caches downloaded dependencies and locally built artifacts.
* **Remote Repository**: Online repositories like Maven Central or Adobe’s public repository (https://repo.adobe.com/nexus/content/groups/public/).

To check:

* **Local**: Browse ~/.m2/repository to see cached artifacts.
* **Remote**: Add the repository to <repositories> in pom.xml and use mvn dependency:resolve to verify availability.
* In AEM, ensure the Adobe repository is configured in pom.xml or settings.xml (e.g., <id>adobe-public</id>).

1. **How all modules build using Maven?**

In a multi-module AEM project, the parent pom.xml lists all modules under <modules> (e.g., core, ui.apps, ui.content, ui.frontend, all). When you run mvn clean install:

* Maven processes the parent POM first.
* It then builds each module in the order specified, resolving inter-module dependencies.
* For example: core compiles Java code into an OSGi bundle.
* ui.frontend builds front-end assets (e.g., CSS/JS via Webpack).
* ui.apps and ui.content create content packages.
* all aggregates everything into a single deployable package.

The all module typically embeds other modules’ artifacts, ensuring a unified deployment to AEM.

1. **Can we build a specific module?**

Yes, you can build a specific module by navigating to its directory and running a Maven command, or by using the -pl (project list) flag from the root. Examples:

* **From module directory**: cd ui.apps && mvn clean install
* **From root**: mvn clean install -pl ui.apps

You can also use profiles (e.g., -PautoInstallPackage) to deploy the specific module to AEM. This is useful for incremental builds, saving time during development.

1. **Role of ui.apps, ui.content, and ui.frontend folder?**

These are standard modules in an AEM Maven project:

* **ui.apps**:
  + Contains immutable code under /apps, such as components, templates, and client libraries (JS/CSS).
  + Built into a content package and deployed to AEM’s JCR repository.
  + Role: Houses the application logic and rendering code.
* **ui.content**:
  + Contains mutable content under paths like /content, /conf, or /etc.
  + Includes sample content, configurations, or ACLs.
  + Role: Manages content that can be modified at runtime (e.g., by authors).
* **ui.frontend**:
  + Optional module for front-end development using tools like Webpack.
  + Compiles JS, CSS, and other assets, then transforms them into AEM client libraries.
  + Role: Enables modern front-end workflows, separate from AEM’s backend.

Together, they separate concerns: ui.apps for code, ui.content for content, and ui.frontend for front-end assets.

1. **Why we are using run mode?**

Run modes in AEM allow configurations to be applied based on the environment or instance type (e.g., author, publish, dev, prod). They are used to:

* Load specific OSGi configurations (e.g., /apps/myproject/config.author for author instances).
* Customize behavior without changing code (e.g., logging levels in dev vs. prod).
* In Maven, run mode-specific configs are placed in ui.config or ui.content under paths like /apps/myproject/config.<runmode>.

This ensures flexibility and environment-specific customization.

1. **What is publish env?**

The **publish environment** in AEM is the instance where content is served to end users (public-facing). Key points:

* Runs on port 4503 by default (vs. 4502 for author).
* Optimized for performance and caching, not authoring.
* Receives content replicated from the author instance via workflows or manual replication.
* In Maven, you deploy to publish using profiles like -PautoInstallPackagePublish or by setting <publish.crx.port> in pom.xml.

1. **Why we are using dispatcher?**

The **Dispatcher** is AEM’s caching and load-balancing layer, typically running on Apache HTTP Server. It’s used to:

* **Cache Content**: Store static HTML and assets to reduce load on the publish instance.
* **Improve Performance**: Serve cached content quickly to users.
* **Security**: Filter requests and protect the publish instance.
* In Maven projects, the dispatcher module (e.g., dispatcher.ams or dispatcher.cloud) contains configuration files deployed to the Dispatcher.

It’s critical for scalability and speed in production.

1. **From where can we access the crx/de?**

CRX/DE (Content Repository Extreme Development Environment) is AEM’s JCR repository explorer. You can access it at:

* **URL**: http://localhost:4502/crx/de (for a local author instance).
* **Credentials**: Default is admin/admin (change in production).
* **Purpose**: View and edit the JCR structure (e.g., /apps, /content), debug, or manually manage nodes.

Ensure you’re logged into an AEM instance with sufficient permissions.