Maven Best Practices and Bad Practices

Best Practices

1. Use a Standard Directory Structure

Follow Maven's standard project structure (src/main/java, src/test/java, etc.)

Ensures compatibility with other tools and frameworks.

2. Use Dependency Management

Declare dependencies with explicit versions.

Use `<dependencyManagement>` to control dependency versions across modules in multi-module projects.

Prefer well-maintained, stable dependencies.

3. Keep the `pom.xml` Clean

Avoid unnecessary dependencies and configurations.

Use parent POMs to centralize configurations where applicable.

Use meaningful artifact names and versions.

4. Use a Repository Manager

Configure a repository manager (e.g., Nexus, Artifactory) to cache dependencies locally.

Reduces build time and network dependencies.

5. Follow Versioning Best Practices

Use semantic versioning (MAJOR.MINOR.PATCH) for artifacts.

Avoid using SNAPSHOT versions in production builds.

6. Optimize Build Performance

Use the `mvn clean install -T 1C` command to enable parallel builds.

Exclude unnecessary plugins and dependencies.

Use incremental builds (`mvn compile` instead of `mvn clean install` when possible).

7. Maintain Proper Testing Strategy

Use proper test phases (`unit` tests in `test` phase, `integration` tests in `verify` phase).

Ensure tests are not skipped in CI/CD pipelines.

Use `maven-surefire-plugin` and `maven-failsafe-plugin` properly.

8. Use Profiles for Environment-Specific Configurations

Define profiles for `dev`, `staging`, and `prod` environments.

Avoid hardcoding environment-specific settings in `pom.xml`.

9. Secure Sensitive Information

Use `.mvn/settings.xml` or environment variables for credentials instead of storing them in `pom.xml`.

Use Maven encryption features for securing passwords.

10. Automate Builds and Deployments

Use CI/CD tools (Jenkins, GitHub Actions, GitLab CI) to automate Maven builds.

Use `maven-release-plugin` to manage releases properly.

Bad Practices

1. Overloading `pom.xml`

Avoid excessive plugins, dependencies, and configurations.

Don’t duplicate dependency versions; use `<dependencyManagement>`.

2. Using SNAPSHOT Versions in Production

SNAPSHOT dependencies are unstable and can cause unexpected issues.

Always release stable versions for production.

3. Committing Target Directory or Dependencies

`target/` directory and `*.jar` dependencies should not be committed to version control.

Use `.gitignore` to exclude them.

4. Not Using a Repository Manager

Relying solely on Maven Central can slow down builds and introduce availability risks.

5. Running `mvn clean` Unnecessarily

Avoid running `mvn clean` unless needed; it slows down incremental builds.

6. Hardcoding Configurations

Avoid hardcoding environment-specific configurations in `pom.xml`.

Use placeholders and profiles instead.

7. Ignoring Dependency Conflicts

Use `mvn dependency:tree` to check for conflicts.

Explicitly define versions to avoid transitive dependency issues.

8. Not Keeping Maven Up-to-Date

Running outdated Maven versions may lead to security vulnerabilities and compatibility issues.

9. Skipping Tests in Regular Builds

Avoid using -DskipTests or -Dmaven.test.skip=true frequently.

Ensure tests are properly executed before deployment.

10. Poor Logging and Debugging

Avoid suppressing warnings and logs (`-q` or `-X` without need).

Use `mvn help:effective-pom` and `mvn debug` for troubleshooting.