Maven

Maven is a powerful *project management tool* that is based on POM (project object model).

It is used for projects build, dependency and documentation.

Understanding the problem without Maven:

There are many problems that we face during the project development. They are discussed below:

1) **Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

2) **Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

3) **Building and Deploying the project:** We must have to build and deploy the project so that it may work.

What is Build Tool

A build tool takes care of everything for building a process. It does following:

- ❑ Generates source code (if auto-generated code is used)
- ❑ Generates documentation from source code
- ❑ Compiles source code
- ❑ Packages compiled code into JAR of ZIP file
- ❑ Installs the packaged code in local repository, server repository, or central repository

Verify maven

To verify whether maven is installed or not, open the command prompt and write:

mvn −version

Convention over Configuration

Maven uses Convention over Configuration, which means developers are not required to create build process themselves.

Item	Default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

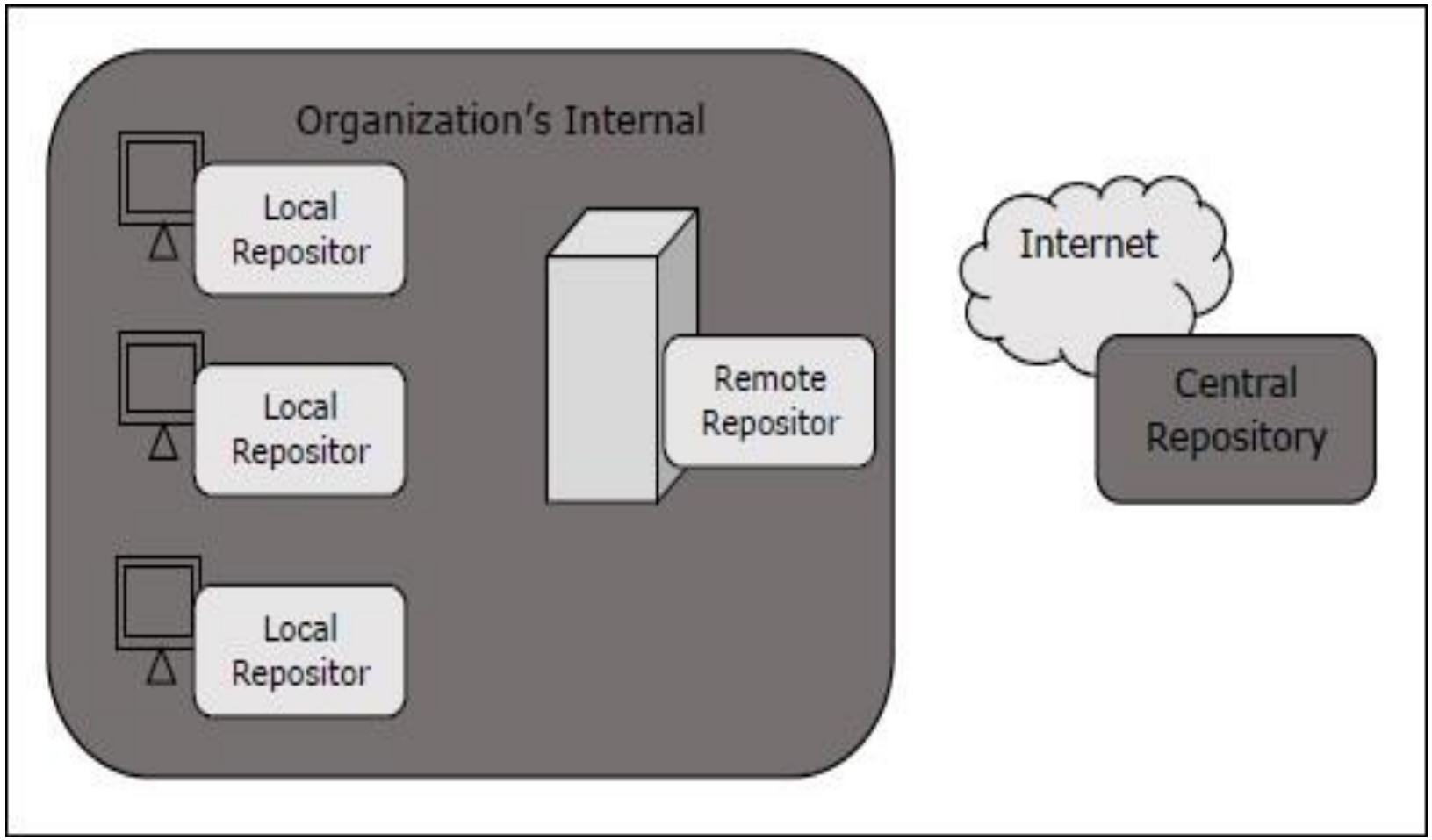
Maven Repository

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

- Local Repository
- Central Repository
- Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Central repository then Remote repository.


Organization's Internal
Local Repositor
Local Repositor
Local Repositor
Remote Repositor
Internet
Central Repository

Maven Local Repository

By default, maven local repository is %USER_HOME%/.m2 directory. For example: C:\Users\Ramana\.m2.

To change the local repository:

apache-maven-3.1.1\conf\settings.xml.
<localRepository>/path/to/local/repo</localRepository>

Maven Central Repository

Maven central repository is located on the web.

It has been created by the apache maven community itself.

The path of central repository is:
https://repo1.maven.org/maven2/

The central repository contains a lot of common libraries that can be viewed by this url
https://search.maven.org/#browse.

Remote Repository

It is used to define custom repository containing required libraries or other project jars.

```
<repositories>
    <repository>
      <id>companyname.lib1</id>
      <url>http://download.companyname.org/maven2/lib1</url>
    </repository>
    <repository>
      <id>companyname.lib2</id>
      <url>http://download.companyname.org/maven2/lib2</url>
    </repository>
  </repositories>
```

Maven pom.xml file

POM is an acronym for Project Object Model.

The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

POM Example

```
<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.companyname.project-group</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
</project>
```

All POM files require the project element and three mandatory fields: groupId, artifactId, version.

Projects notation in repository is groupId:artifactId:version.

Project root

This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification.

Model version

Model version should be 4.0.0. [Latest is 5]

groupId

This is an Id of project's group.

This is generally unique amongst an organization or a project.

For example, a banking group com.company.bank has all bank related projects.

artifactId

This is an Id of the project. This is generally name of the project. For example, consumer-banking.

Along with the groupId, the artifactId defines the artifact's location within the repository.

version

This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example −

com.company.bank:consumer-banking:1.0

com.company.bank:consumer-banking:1.1.

Super POM

The Super POM is Maven’s default POM.

All POMs inherit from a parent or default.

This base POM is known as the Super POM, and contains values inherited by default.

Maven use the effective POM to execute relevant goal. It helps developers to specify minimum configuration in the project pom.xml. We can also override the configurations.

To display the super POM :

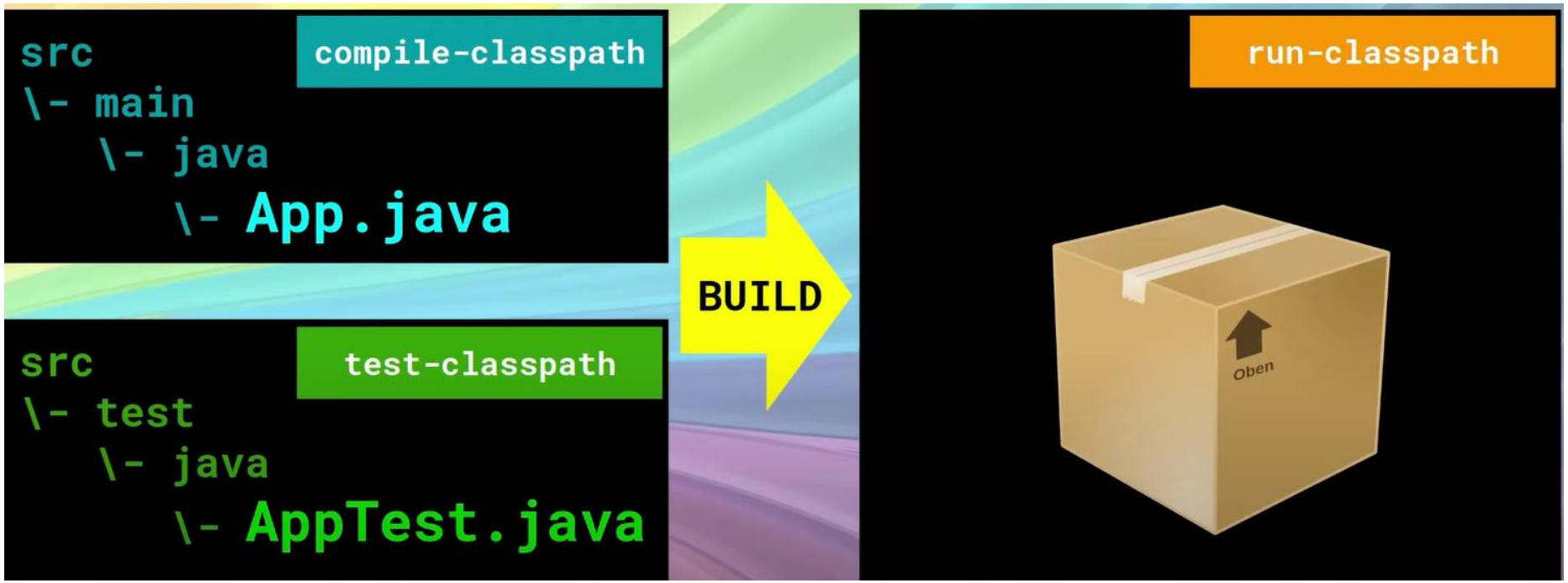
mvn help:effective-pom

Maven Class paths

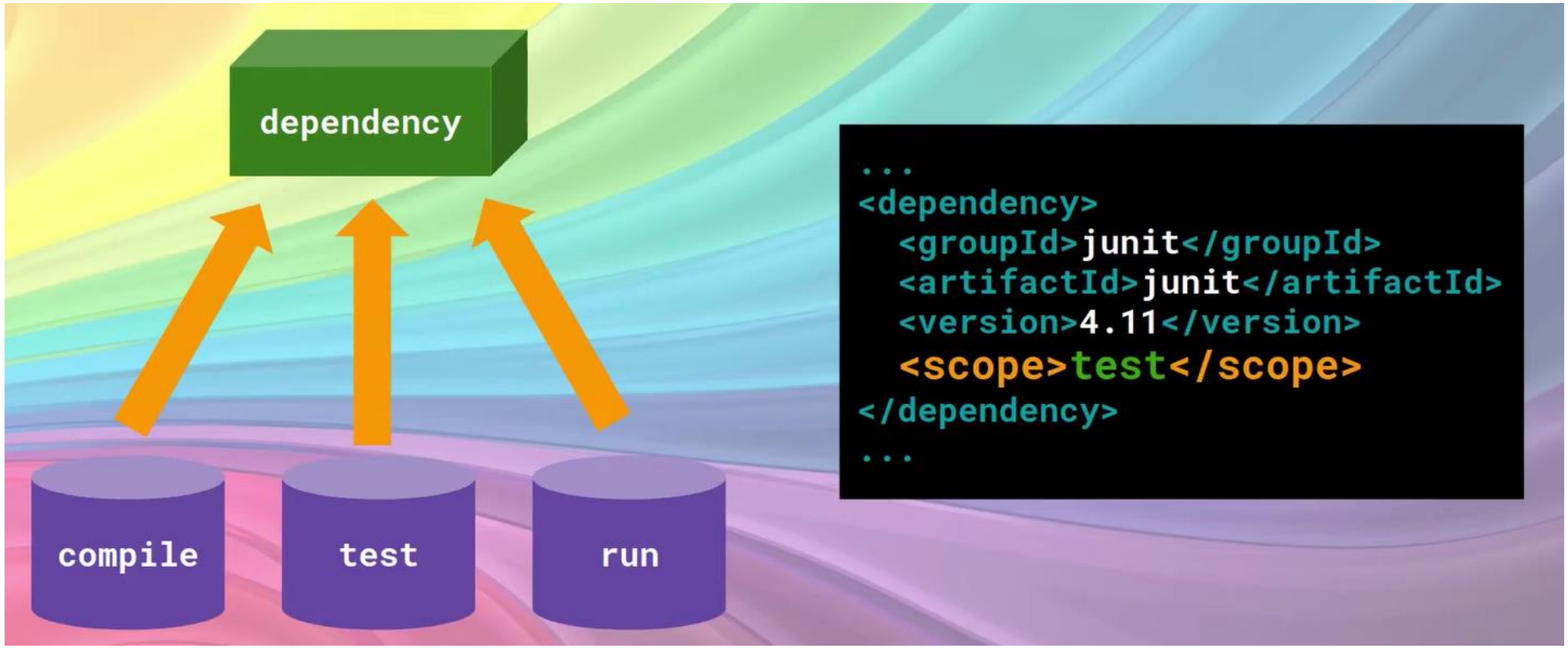
compile-classpath : Application code is being compiled

test-classpath: Application code is being tested

run-classpath: Application code is running



Dependency Scope



Dependency Scope

Dependency scope is used to limit the transitivity of a dependency and to determine when a dependency is included in a classpath.

There are 6 scopes:

Compile (default scope)
Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects. Available for compile-classpath, test-classpath and runtime-classpath

provided
This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes. A dependency with this scope is added to the classpath used for compilation and test, but not the runtime classpath.

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

runtime
This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.

test
This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive. Typically this scope is used for test libraries such as JUnit and Mockito. It is also used for non-test libraries such as Apache Commons IO if those libraries are used in unit tests (src/test/java) but not in the model code (src/main/java).

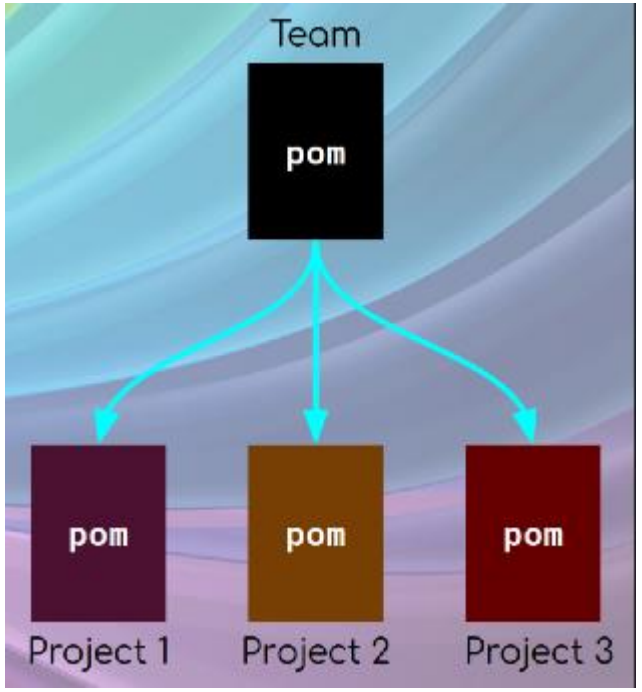
system
This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
It is available for compile-classpath and test-classpath.

```
<dependency>
    <groupId>com.baeldung</groupId>
    <artifactId>custom-dependency</artifactId>
    <version>1.3.2</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/libs/custom-dependency-
1.3.2.jar</systemPath>
</dependency>
```

Import

This scope is only supported on a dependency of type pom in the <dependencyManagement> section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

```
<dependencyManagement>
 <dependencies>
  <dependency>
    <groupId>com.company</groupId>
    <artifactId>project-deps</artifactId>
    <version>1.0</version>
    <scope>import</scope>
    <type>pom</type>
  </dependency>
 </dependencies>
</dependencyManagement>
```



Dependency Management

The dependency management section is a mechanism for centralizing dependency information.

When we have a set of projects that inherit from a common parent, it's possible to put all information about the dependency in the common POM and have simpler references to the artifacts in the child POMs.

Plugin Management:

Plugin Management contains plugin elements in much the same way, except that rather than configuring plugin information for this particular project build, it is intended to configure project builds that inherit from this one.

However, this only configures plugins that are actually referenced within the plugins element in the children or in the current POM.

Maven Plugins?

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to −

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax −

mvn [plugin-name]:[goal-name]

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.

mvn compiler:compile

Plugin Types

Maven provided the following two types of Plugins −

Sr.No.	Type & Description
1	**Build plugins** They execute during the build process and should be configured in the <build/> element of pom.xml.
2	**Reporting plugins** They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.

Spring Boot Maven Plugin

The Spring Boot Maven Plugin provides Spring Boot support in Apache Maven. It allows you to package executable jar or war archives, run Spring Boot applications, generate build information and start your Spring Boot application prior to running integration tests.

```
<build>
  <plugins>
    <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

If you use a milestone or snapshot release, you also need to add the appropriate pluginRepository elements, as shown in the following listing:

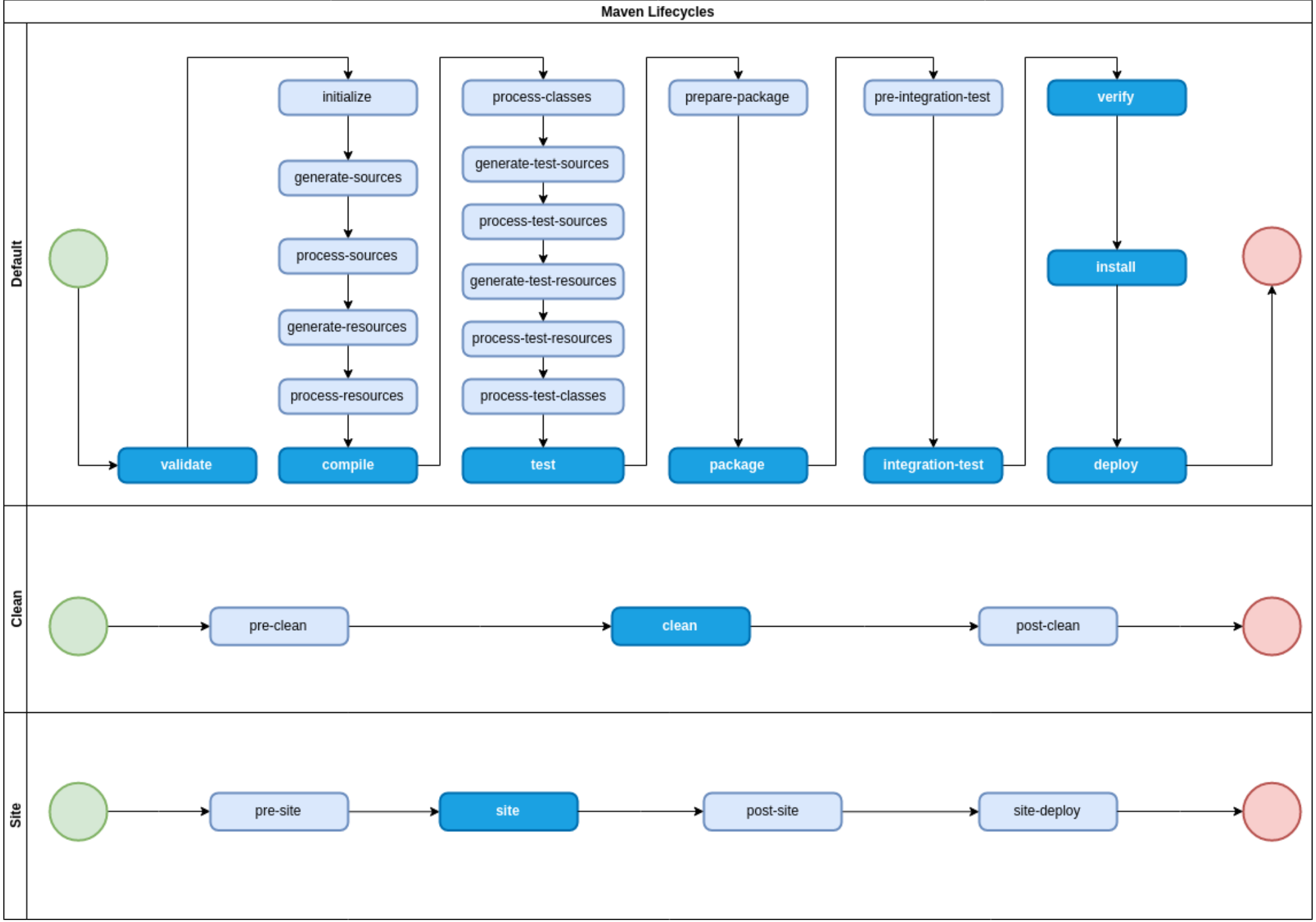
```
<pluginRepositories>
        <pluginRepository>
                <id>spring-snapshots</id>
                <url>https://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
                <id>spring-milestones</id>

        <url>https://repo.spring.io/milestone</url>
        </pluginRepository>
</pluginRepositories>
```

What is Build Lifecycle?

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.


Maven Lifecycles
Default
validate
initialize
generate-sources
process-sources
generate-resources
process-resources
compile
process-classes
generate-test-sources
process-test-sources
generate-test-resources
process-test-resources
process-test-classes
test
prepare-package
package
pre-integration-test
integration-test
verify
install
deploy
Clean
pre-clean
clean
post-clean
Site
pre-site
site
post-site
site-deploy

Exclude embedded tomcat in spring boot application:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
             </exclusion>
          </exclusions>
</dependency>
```

Use jetty server in spring boot application

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
   <exclusions>
      <exclusion>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-tomcat</artifactId>
      </exclusion>
   </exclusions>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

Using Spring Boot without the parent POM

```
<dependencyManagement>
    <dependencies>
      <dependency>
         <!-- Import dependency management from Spring Boot -->
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-dependencies</artifactId>
         <version>1.3.8.RELEASE</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

That setup does not allow you to override individual dependencies,we need to add an entry in the dependencyManagement of your project before the spring-boot-dependencies entry.

For instance, to upgrade to another Spring Data release train you'd add the following to your pom.xml.

```
<dependencyManagement>
   <dependencies>
      <!-- Override Spring Data release train provided by Spring Boot -->
      <dependency>
         <groupId>org.springframework.data</groupId>
         <artifactId>spring-data-releasetrain</artifactId>
         <version>Fowler-SR2</version>
         <scope>import</scope>
         <type>pom</type>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-dependencies</artifactId>
         <version>1.3.8.RELEASE</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```