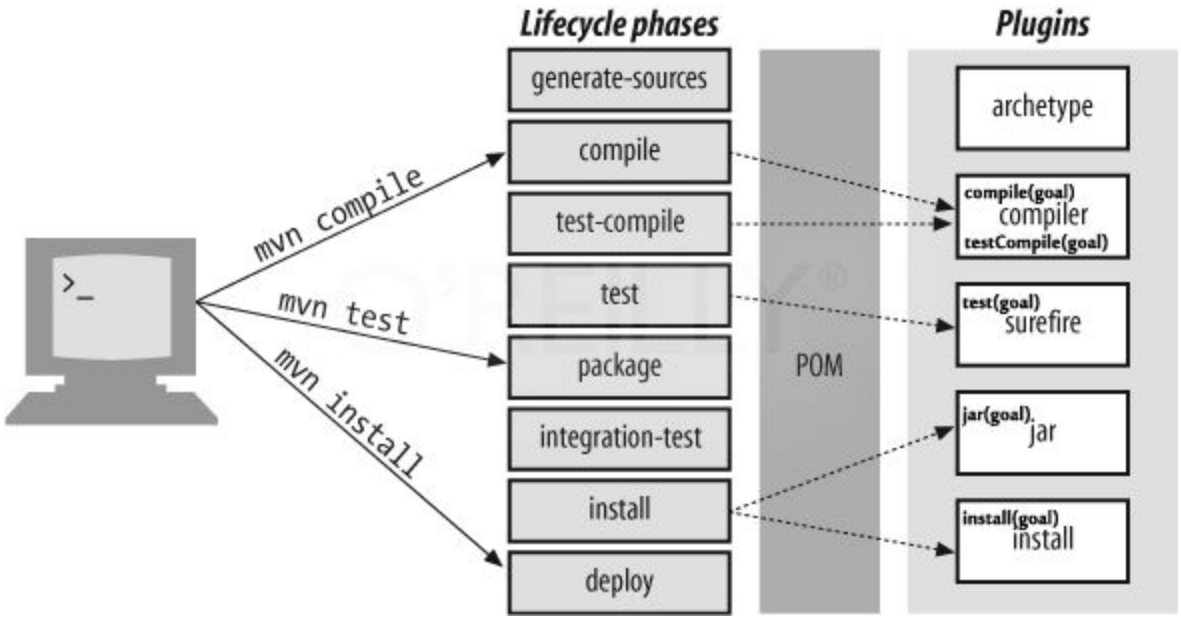


"Maven **passes** all phases (up to and including the given)" instead of *"runs"* or *"executes" (e.g. mvn compile) --*

"mvn <phase>"

Performing "mvn clean" does the following:

Calls the "clean phase" in the "clean life cycle". Since phases are executed in orde, it will call the "pre-clean phase", followed by the "clean phase"

Plugin goal's code is executed (e.g. mvn archetype:generate) --

"mvn <plugin name>:<goal-name>"

"mvn pre-clean clean:clean" is not the same as calling "mvn clean"!

Plugin is a collection of goals; Analogy : Plugin is a class and goals are methods within the class. Most of Maven's functionality is in plugins.

*Life cycle is a sequence of named **phases.***

Maven is based around the central concept of a build lifecycle; each is made up of phases. There are three built-in build lifecycles and each has phases:

1. *Default (aka build)*
    a. ***validate** - validate the project is correct and all necessary information is available.*
    b. ***Initialize** - initialize build state, e.g. set properties or create directories.*
    c. ***generate-sources** - generate any source code for inclusion in compilation.*
    d. ***process-sources** - process the source code, for example to filter any values.*
    e. ***generate-resources** - generate resources for inclusion in the package.*

f. ***process-resources*** *- copy and process the resources into the destination directory, ready for packaging. compile compile the source code of the project. process-classes post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.*
g. ***generate-test-sources*** *- generate any test source code for inclusion in compilation. process-test-sources process the test source code, for example to filter any values. generate-test-resources create resources for testing.*
h. ***process-test-resources*** *- copy and process the resources into the test destination directory.*
i. ***test-compile*** *- compile the test source code into the test destination directory*
j. ***process-test-classes*** *- post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above.*
k. ***test*** *- run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.*
l. ***prepare-package*** *- perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above)*
m. ***package*** *- take the compiled code and package it in its distributable format, such as a JAR.*
n. ***pre-integration-test*** *- perform actions required before integration tests are executed. This may involve things such as setting up the required environment.*
o. ***integration-test*** *- process and deploy the package if necessary into an environment where integration tests can be run.*
p. ***post-integration-test*** *- perform actions required after integration tests have been executed. This may including cleaning up the environment.*
q. ***verify*** *- run any checks to verify the package is valid and meets quality criteria.*
r. ***install*** *- install the package into the local repository, for use as a dependency in other projects locally.*
s. ***deploy*** *- done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.*

2. *Clean*
    a. ***pre-clean*** *- execute processes needed prior to the actual project cleaning*
    b. ***clean*** *- remove all files generated by the previous build*
    c. ***post-clean*** *- execute processes needed to finalize the project cleaning*
3. *Site*
    a. ***pre-site*** ***-*** *execute processes needed prior to the actual project site generation*
    b. ***Site -*** *generate the project's site documentation*
    c. ***post-site*** *- execute processes needed to finalize the site generation, and to prepare for site deployment*
    d. ***site-deploy*** *- deploy the generated site documentation to the specified web server*

Some phases are not usually called from the command line (e.g. pre-, post-*, and process-*) - these phases sequence the build, producing intermediate results that are not useful outside the build.*

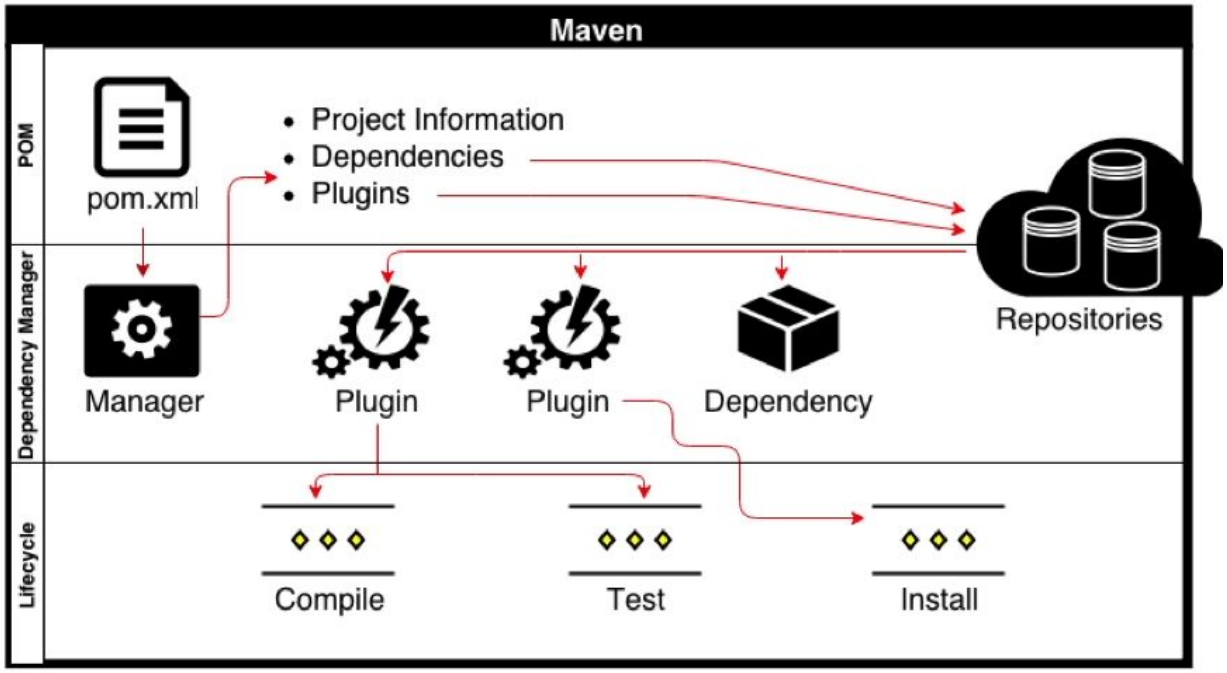
Plugins are artifacts that provide one or more goals. These goals are bound to life-cycle phases, but this is largely dependent upon the packaging type (jar, war, etc.)

A build phase is made up of Plugin Goals which are "registered with" (or "bound to") build phases. These goals can individually be executed via the command line. For example, a java project can be compiled with the compiler-plugin's compile-goal by running: mvn compiler:compile. This particular goal is associated with the "compile" phase.

A plugin goal represents a specific task that can be execute. It can be bound to zero or more build phases:

- *A goal not bound to any build phase can be executed outside of the build lifecycle by direct invocation (e.g. mvn <plugin>:<goal>).*
- *A goal bound to multiple phases will be executed as each phase is processed*

Some phases have goals bound to them by default. And for the default lifecycle, these bindings depend on the packaging value.



When configuring a plugin in a POM, goal needs to be specified.

In case a plugin definition does not have a default build phase, the phase can be specified/bound with the plugin goal.

```
<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/maven-v4_0_0.xsd"
>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.maventest</groupId>
  <artifactId>aproject</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>aproject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
       <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
       </dependency>
  </dependencies>
  <build>
       <plugins>
       <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-antrun-plugin</artifactId>
       <version>1.1</version>
       <executions>
       <execution>
              <id>id.pre-clean</id>
              <phase>pre-clean</phase>
              <goals>
              <goal>run</goal>
              </goals>
              <configuration>
              <tasks>
              <echo>in pre-clean phase</echo>
              </tasks>
              </configuration>
       </execution>
       <execution>
              <id>id.clean</id>
              <phase>clean</phase>
              <goals>
              <goal>run</goal>
              </goals>
              <configuration>
              <tasks>
```

```
            <echo>in clean phase</echo>
            </tasks>
            </configuration>
        </execution>
        <execution>
          <id>id.post-clean</id>
          <phase>post-clean</phase>
          <goals>
            <goal>run</goal>
          </goals>
          <configuration>
            <tasks>
              <echo>in post-clean phase</echo>
            </tasks>
          </configuration>
        </execution>
        </executions>
        </plugin>
        </plugins>
    </build>
</project>
```