Maven_learning: (Basics)

Monday, November 6, 2017

Set M2_HOME - to maven installation directory and PATH - append the bin path of maven

mvn archetype:generate - generates the project template according to the type (archetype) of project we select at the start
(Folder structure + pom.xml)
Archetype -will control project stucture
groupId - similar to package. (Group all the artifacts under this groupId)
artifactId - Like Class name - o/p of our project. (war/jar)
Version - 1.0-SNAPSHOT (SNAPSHOT - still in dev, not released)
Package - name of the folder where the project shd be stored (packaging - jar)
groupId, artifactId, version and package get into pom.xml

Pom.xml will have the above stuffs with the dependencies (groupId, artifactId, version, scope - tells maven when to use the dependency)

We can publish our repo or have the project in our local repo
Project that is present in local repo can be used by other projects by mentioning groupId, artifactId and version in the dependencies (pom.xml)

Issue these commands from the directory where pom.xml is present
mvn compile - compiles all the java files in the project and generates class files
mvn package - packages and gives jar

Maven build life cycle:
Phases - Compile, Test, Package (Configurable, else it has a default build life cycle)
Some phases:
-> **validate** phase (Checks if config is proper, code is in proper structure)
-> **compile** compiles java files and stores class files in a folder called target - (automatically triggers validate before compilation)
-> **test** - executes test cases (validate and compile are run before it automatically)
-> **package** - .jar, … output artifact places it in target directory (validate, compile and test are run automatically)
-> **install** - installs the artifact into a local maven repository (Not to the application server) usually /home/…/m2/.. Places the jar (artifacts) there
-> **deploy** - to publish artifact to remote repo so that other users can use (Not on application server)

Adding dependencies in maven:
mvn clean - removes all the classes and distributions that were available earlier (remove target folder : example)

Usual way - download the dependency jars, add it to classpath. Only after this is taken care of, we will be able to compile the class files that are dependent on it.
Maven - takes care of all that automatically if we mention the dependency
-> Mention the import in the java file
-> Go to pom.xml
-> Add a dependency tag

```
<dependency>
    <groupId></groupId>
    <artifactId></artifactId>
    <version></version>
</dependency>
```

How do I get the groupId, artifactId and version? - Search on sites like mvnrepository.com and search for the dependency. We can get groupId, artifactId and version. So, just mention it in pom.xml (In a dependency tag)
-> Compile now - mvn compile - maven goes ahead and gets dependencies (downloads jars - resources) and then compiles.
-> Default scope - compile scope <scope>compile</scope> - if we do not specify scope, it means, this dependency is available during compile time.

Setting up a web application using maven: (Example)
mvn archetype:generate
Choose the archetype - a webapp, version

Phases/Commands:
mvn archetype:generate
mvn compile
mvn test
mvn package
mvn install
mvn clean
mvn <plugin>:<what, the plugin shd do>

Mention the groupId and artifactId and other stuffs

Maven compiler - Is present by default. (It is a plugin)
If we wanna configure it in a custom way, an explicit mention about it is required in pom.xml
Additional node that takes care of plugins: build node

```
<build>
    <plugins>
        <plugin>
            <groupId></groupId>
            <artifactId></artifactId>
            <version></version>
            <configuration>
                <source></source> - jvm version (1.8, say)
                <target></target> - jvm version (1.8)
            </configuration>
        </plugin>
    </plugins>

</build>
```

mvn compile
mvn package - build the war (artifact)

Src - will have a default jsp (Hello world)
Deploy this war in tomcat and check

Scopes:
Compile - default scope - consider dependency during compile time
Test
Provided

Maven plugins - extending maven functionalities
Maven - modular architecture coz all the core components of maven are plugins

Other helpful plugins: tomcat plugins, jetty plugins

mvn <plugin>:<what the plugin shd do>
Example: mvn jetty:run

We can configure the plugin to watch for changes, automatically compile and deploy them
Example: scanIntervalSeconds configuration:

```
<configuration>
    <scanIntervalSeconds>2</scanIntervalSeconds > //watch every 2s and compile the latest code
</configuration>
```

Maven has plugin for IDEs and IDEs have plugin for maven.. We can configure 'em

*/adventure/kruthikacs/learnings