Exercise – Installing and using Maven

Objective

The objective of this exercise is to look at installing maven and then generating a couple of projects using maven archetypes.

Setup

We will be installing and using maven on a machine in the amazon web services cloud. Create a new instance with the following options:

- AMI – Amazon Linux AMI 2018.x
- Type: t2.micro
- Storage: 8 GiB
- Tag: Name = Maven
- Security group: Open port 8080/tcp for access from anywhere

Do not use the same machine as your GitLab server, otherwise you will have issues with clashing port numbers.

Part 1: Installing Maven

1. Connect to your machine and ensure everything is up to date.

```
$ sudo yum -y update
```

2. Maven requires the Java Development Kit – install it with

```
$ sudo yum remove java -y
$ sudo yum install java-1.8.0-openjdk-devel -y
```

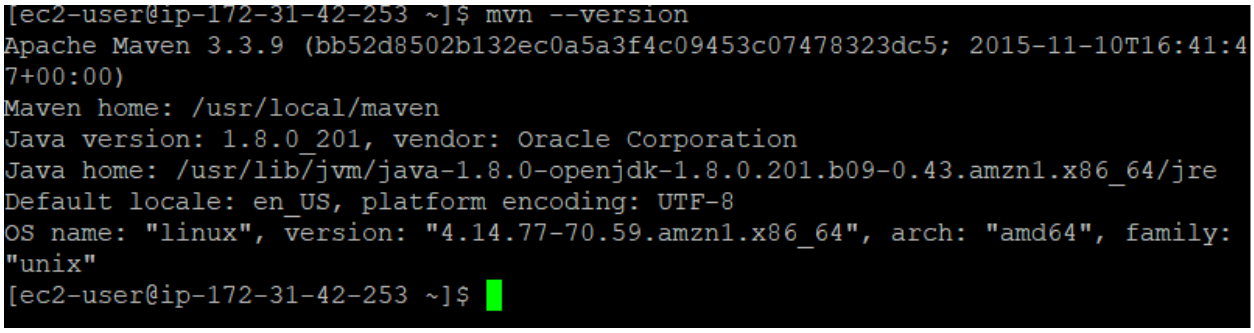
Maven is not included in the yum repositories by default so we will need to manually install it.

```
# get the zip file from the website

$ wget
http://mirror.ox.ac.uk/sites/rsync.apache.org/maven/maven-
3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz

# unzip this to /usr/local

$ sudo tar xzf apache-maven-3.3.9-bin.tar.gz -C /usr/local


# create a symbolic link between the program and an easier to
remember directory name

$ sudo ln -s /usr/local/apache-maven-3.3.9 /usr/local/maven

# Finally, link between the mvn binary and the /bin folder

$ sudo ln -s /usr/local/maven/bin/mvn /bin/mvn
```

We can test to see if maven installed correctly with:

```
$ mvn --version
```

Output:



Part 2: Hello Maven!

Now we want to use maven to generate an archetype project. These are stored sets of instructions that tell maven what to download and setup when creating a new project.

1. Make sure you are in your home directory of the EC2 machine then type:

```
$ mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
```

This will generate a project using the maven-archtype-quickstart set of instructions. The -D flag has told it which archetypeArtifactID we want to use. As we did not give any other instructions then maven will prompt for other information it needs.

a. For groupID type: **com.pmw**

b. For artifactID type**: hello-maven**

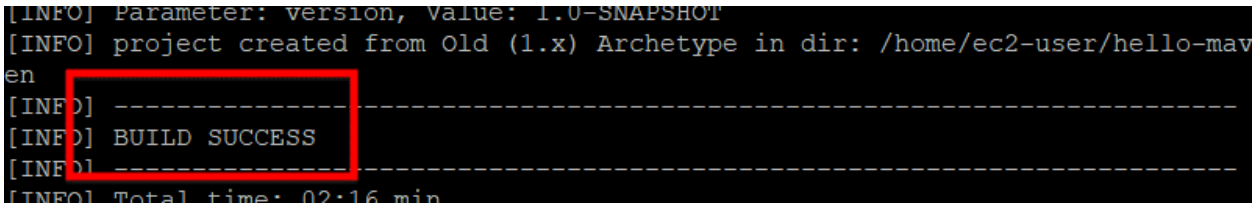
c. For version, package and the final confirmation just press **enter** 3 times to accept the default.

Alternatively, **for scripting**, we could have given the full information with:

```
mvn archetype:generate -DgroupId=com.pmw  -DartifactId=hello-maven -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

The **lack of spaces** around the equals sign is important, as is the capitalisation. Maven requires these to be correct otherwise it cannot understand what we have asked for.

Output:



2. If this has built successfully you will have a folder called hello-maven. Go into the folder and explore the structure of the project. This is the standard layout for a maven project

```
$ cd hello-maven

$ sudo yum install tree -y

$ tree
```

Output:

The Project Object Model (POM) is the control file for maven projects – pom.xml. It tells maven how to build the project, what dependencies it needs and how to run it. You can have a look at what is included by default by typing:

```
$ cat pom.xml
```

We can now build and run the project, or just run the tests. There are various goals we can give maven which will tell it what is required. The default goals are:

- **validate** - validate the project is correct and all necessary information is available
- **compile** - compile the source code of the project
- **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package** - take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test** - process and deploy the package if necessary into an environment where integration tests can be run
- **verify** - run any checks to verify the package is valid and meets quality criteria
- **install** - install the package into the local repository, for use as a dependency in other projects locally
- **deploy** - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Executing any of these goals will also execute any before it. So if we were to run **mvn test** maven would first validate and compile the project.

3. Test your hello-maven project by typing

```
$ mvn test
```

```
 T E S T S
-------------------------------------------------------
Running com.pmw.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.03 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

It should state that one test was run and passed. Maven looks for all tests in the /test directory and it knows how to run them as JUnit as it is included in the pom.xml as a dependency.

4. To compile and run the project we need to tell it to package and then to run the .jar file generated using java. Type the following to see the output of your program.

```
$ mvn package

#run the compiled java application

$ java -cp target/hello-maven-1.0-SNAPSHOT.jar com.pmw.App
```

Output:

```
[ec2-user@ip-172-31-41-191 hello-maven]$ java -cp target/hello-maven-1.0-SNAPSHOT.jar com.pmw.App
Hello World!
[ec2-user@ip-172-31-41-191 hello-maven]$
```