

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:

- Image: Amazon linux
- Type: t2.micro
- Storage: 8 gig
- 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.

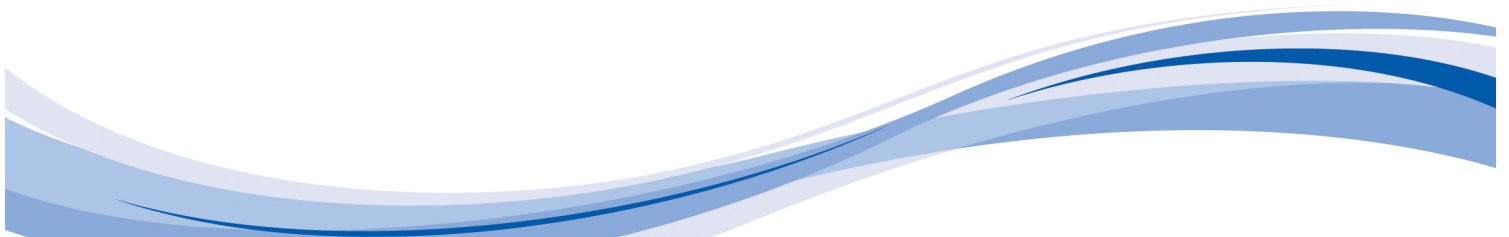
Part 1: Installing Maven

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

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

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

```
$ sudo yum install java-1.8.0-openjdk-devel.x86_64 -y
$ sudo yum remove java-1.7.0 -y
```



3. Install Maven. It 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://apache.mirrors.ionfish.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
```

Part 2: Hello Maven!

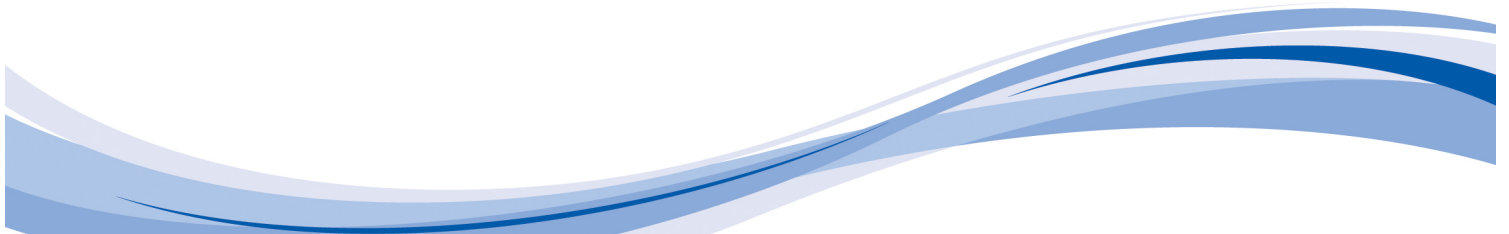
Now we want to use maven to generate an archetype project. These are stored sets of instructions that tell maven what to download and setup when creating a new project.

4. Make sure you are in your home directory then type:

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

This will generate a project using the maven-archetype-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.qa*
- b. For artifactID type: *hello-maven*
- c. For version, package and the final confirmation just press enter to accept the default.



We could have given the full information with:

```
$ mvn archetype:generate -DgroupId=com.qa -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.

5. 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

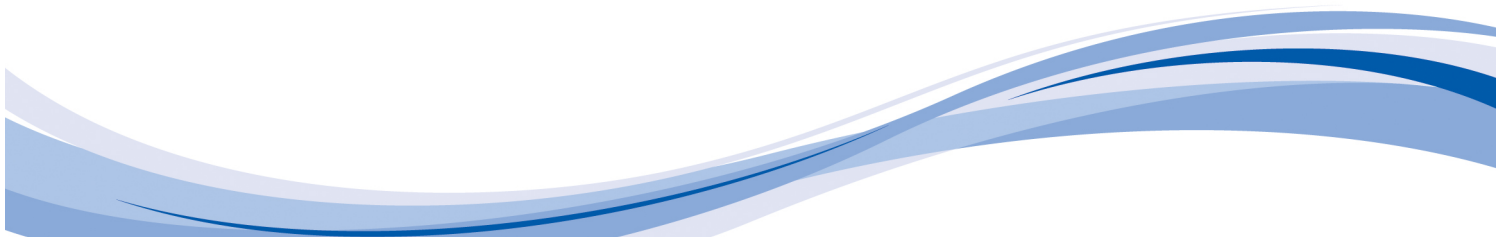
```
$ cd hello-maven
```

```
. .
├─ pom.xml
└─ src
    ├─ main
    │   └─ java
    │       └─ com
    │           └─ qa
    │               └─ App.java
    └─ test
        └─ java
            └─ com
                └─ qa
                    └─ AppTest.java

9 directories, 3 files
```

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.

(List from: <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>)

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.

6. Test your hello-maven project by typing

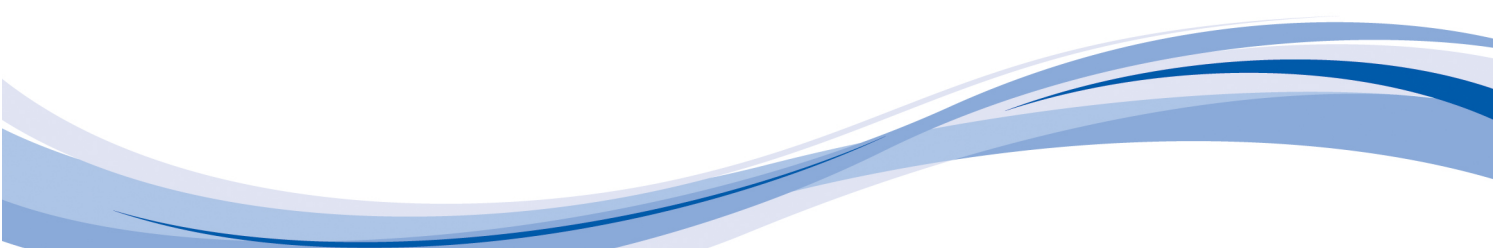
```
$ mvn test
```

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 is included in the `pom.xml` as a dependency.

7. 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
```

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



Part 3: Installing a new maven archetype

There are thousands of maven archetypes available from the main maven repository:

- <https://maven-repository.com/archetypes>

But sometimes we want to be able to create our own from scratch. For example, if we wanted a scalatra project that uses a jetty server we could write our own, or use someone else's archetype.

Scalatra is a scala based framework which allows you to build REST type webservice very quickly. Jetty is a built in webserver that will let us run the project without needing to install a webserver of our own. Maven will deal with all the dependencies for us and allow us to run this without even needing to install a scala compiler to deal with it!

We have a fork of a simple scalatra archetype available on git hub. We will need to clone this repository, install it (using maven) and then we can use this to generate projects using the simple-scalatra-archetype

8. First we need to install git to get hold of the repository

```
$ sudo yum install git -y
```

9. Clone the repository

```
$ cd ~  
$ git clone https://gitlab.com/qatraining/qadevopprac-simple-\scalatra-archetype.git simple-scalatra-archetype/
```

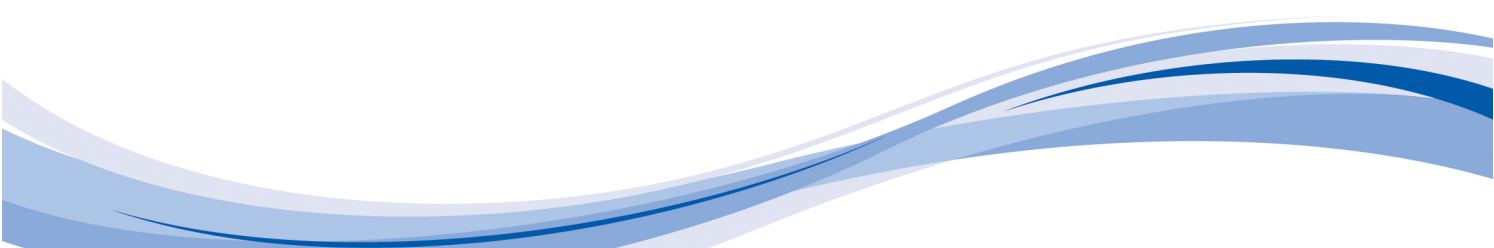
10. Move into the directory and tell maven to install this archetype and update the catalog

```
$ cd simple-scalatra-archetype  
$ mvn clean install archetype:update-local-catalog  
$ mvn archetype:crawl
```

Notice how it goes through each of the default build steps before installing it.

11. Now we can create our own simple scalatra project:

```
$ cd ~  
$ mvn archetype:generate
```



The new archetype should be the last one in the list of presented options:

```
xxxx: local -> org.scalatra:simple-scalatra-archetype (simple-scalatra-archetype)
```

type the number for the new local archetype (should be the last one)

For your GroupID use: *com.qa*

For the artefactID: *hello-scalatra*

Accept the defaults for the version and confirmation (press enter when prompted to both times).

Have a look at the project you have created. Again, don't worry if you don't understand the code. We don't need to. The developers will worry about that side.

```
$ cd hello-scalatra
```

12. To run this project we need a new command. As the Jetty plugin has been included in the *pom.xml* we have a new goal we can specify: `jetty:run` this will tell maven to start up the webserver.

```
$ mvn clean package jetty:run
```

13. When it has finished compiling and started the server point a browser at:

```
[yourip]:8080
```

Note: To say hello to your new project! To stop the webserver running press Ctrl+C

14. Finally, we want to create a git repository for our hello-scalatra project. Create a repository in gitlab, bitbucket or github and push your code to it. Use the hosted options as your own gitlab server may go wrong or be deleted and we will use this hello-scalatra repository throughout the course.
15. If you have time, create another repository on your own gitlab server and push a copy of the code to this.

