Universidad de El Salvador

Facultad Multidisciplinaria de Occidente

Departamento de Ingeniería y Arquitectura

Programación 2.5

Guía "Introducción a Maven"

Objetivo: Que el estudiante sea capaz de implementar librerías de terceros en sus proyectos para facilitarle más herramientas a la hora de programar.

Requisitos:

- Tener Instalado NetBeans.
- Acceso a internet.

Propósito de usar MAVEN:

Programar no significa inventar la rueda sino aprovechar los recursos que ya existen de la mejor forma para lograr los objetivos propuestos.

Maven se utiliza en la gestión y construcción de software. Posee la capacidad de realizar ciertas tareas claramente definidas, como **la compilación del código y su empaquetado**. Es decir, hace posible la creación de software con dependencias incluidas dentro de la estructura del **JAR**.

Maven y Artefactos:

Maven solventa este problema a través del concepto de Artefacto. Un Artefacto puede verse como una librería con esteroides (aunque agrupa más conceptos). Contiene las clases propias de la librería pero además incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias etc).

Dependencias del POM

Para definir un Artefacto necesitamos crear un fichero POM.xml (Proyect Object Model) que es el encargado de almacenar toda la información que hemos comentado anteriormente para que el proyecto se pueda compilar y empaquetar.

Un grupo es un conjunto de artefactos. Es una manera de organizarlos. Así por ejemplo todos los artefactos de Spring Framewok se encuentran en el grupo org.springframework.

Veamos un ejemplo:

```
<dependency>
  <groupid>org.springframework</groupid>
  <artifactid>spring-orm</artifactid>
  <version>3.0.5.RELEASE</version>
  <scope>runtime</scope>
</dependency>
```

Esta es la manera de declarar una dependencia de nuestro proyecto con un artefacto. Se indica el identificador de grupo, el identificador del artefacto y la versión.

Scope (alcance)

El scope sirve para indicar el alcance de nuestra dependencia y su transitividad. Hay 6 tipos:

1. Compile: es la que tenemos por defecto sino especificamos scope. Indica que la dependencia es necesaria para compilar. La dependencia además se propaga en los proyectos dependientes.
2. Provided: Es como la anterior, pero esperas que el contenedor ya tenga esa libreria. Un claro ejemplo es cuando desplegamos en un servidor de aplicaciones, que por defecto, tiene bastantes librerías que utilizaremos en el proyecto, así que no necesitamos desplegar la dependencia.
3. Runtime: La dependencia es necesaria en tiempo de ejecución pero no es necesaria para compilar.
4. Test: La dependencia es solo para testing que es una de las fases de compilación con maven. JUnit es un claro ejemplo de esto.
5. System: Es como provided pero tienes que incluir la dependencia explicitamente. Maven no buscará este artefacto en tu repositorio local. Habrá que especificar la ruta de la dependencia mediante la etiqueta <systemPath>
6. Import: este solo se usa en la sección dependencyManagement.

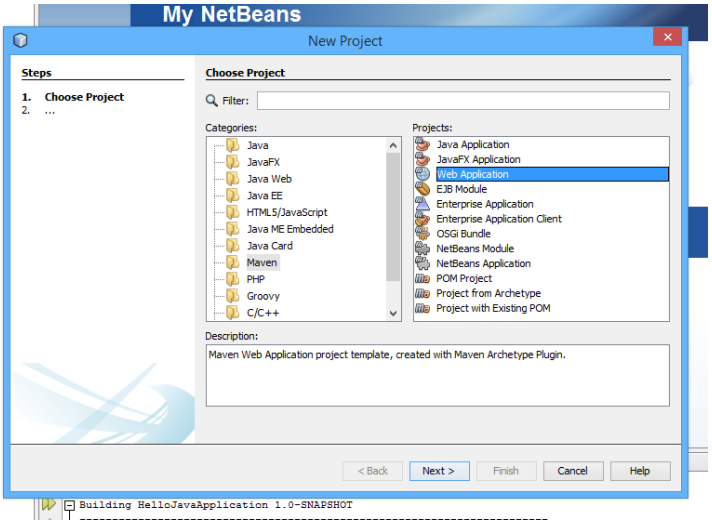
¿Qué son los Plugins de Maven?

Maven es en realidad un marco de ejecución de complementos donde cada tarea se realiza mediante plugins. Los plugins Maven generalmente se usan para:

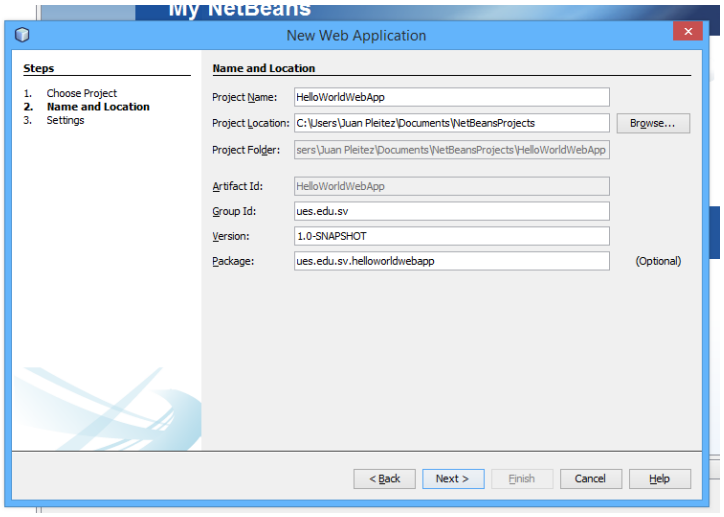
- Crear archivo jar/
- Compilar archivos de código
- Prueba unitaria del código
- Crear documentación del proyecto
- Crear informes de proyectos

Ejemplo de copiar el empaquetado a otra ruta:

1- Creamos un proyecto Maven Web Application.

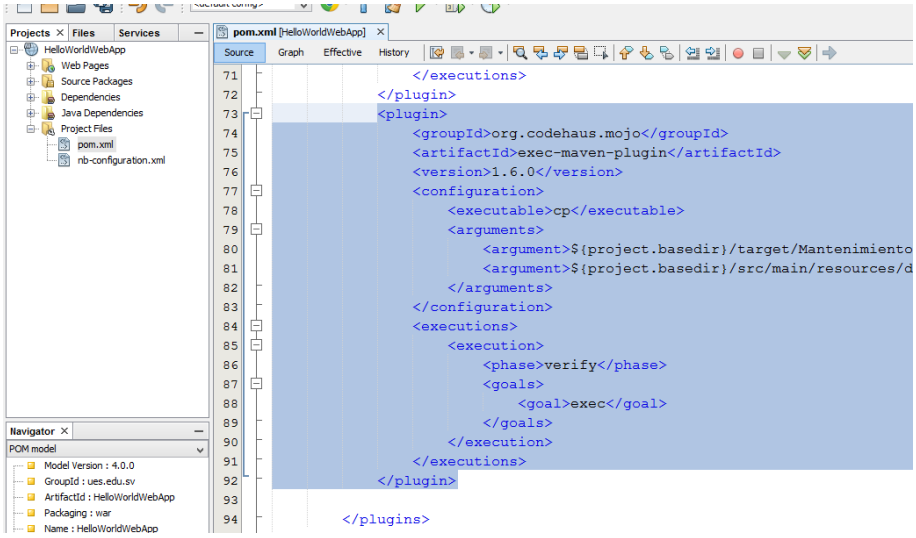


2- Lo nombramos “HelloWorldWebApp” y GroupId “ues.edu.sv” y Selecionamos GlashFis Server.



3- Nos vamos a el POM del proyecto y agregamos el plugin que nos sirve para copiar el empaquetado WAR a esta ruta “/src/main/resources/docker/” si no la tiene, créela.

```
<!--plugin para copiar el war-->
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>1.6.0</version>
<configuration>
<executable>cp</executable>
<arguments>
<argument>${project.basedir}/target/HelloWorldWebApp-1.0-
SNAPSHOT.war</argument>
<argument>${project.basedir}/src/main/resources/docker</argument>
</arguments>
</configuration>
<executions>
<execution>
<phase>verify</phase>
<goals>
<goal>exec</goal>
</goals>
</execution>
</executions>
</plugin>
```



Ejemplo para levantar contenedores de Docker:

Requisito: "Docker instalado".

1- Agregamos la dependencia:

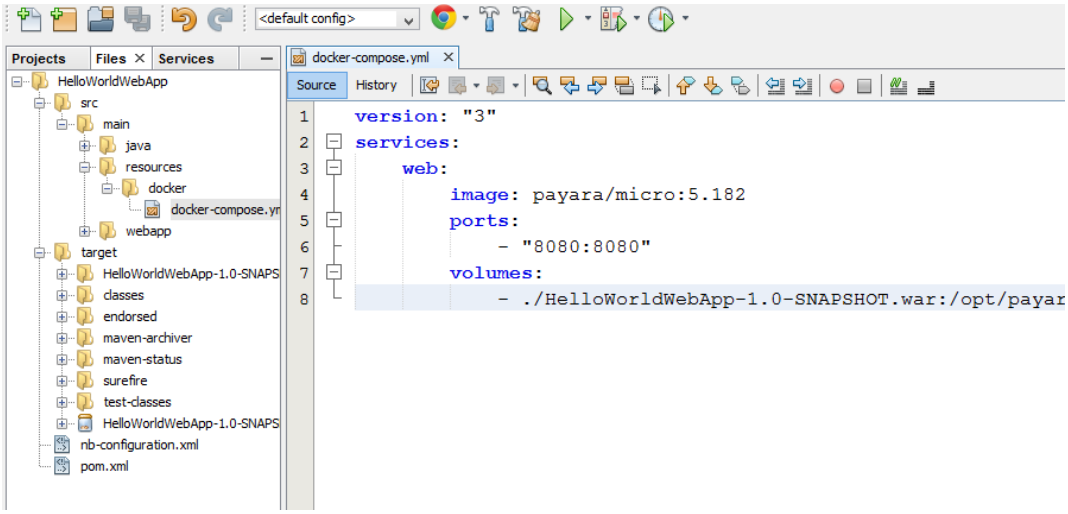
```
<dependency>
<groupId>com.dkanejs.maven.plugins</groupId>
<artifactId>docker-compose-maven-plugin</artifactId>
<version>1.0.3</version>
</dependency>
```

2- Agregamos el Plugin:

```
<!--Plugin que levanta los contenedores-->
<plugin>
<groupId>com.dkanejs.maven.plugins</groupId>
<artifactId>docker-compose-maven-plugin</artifactId>
<version>1.0.3</version>
<executions>
<execution>
<id>down</id>
<phase>verify</phase>
<goals>
<goal>down</goal>
</goals>
<configuration>
<composeFile>${project.basedir}/src/main/resources/docker/docker-
compose.yml</composeFile>
<removeVolumes>false</removeVolumes>
</configuration>
</execution>
<execution>
<id>up</id>
<phase>verify</phase>
<goals>
<goal>up</goal>
</goals>
<configuration>
<composeFile>${project.basedir}/src/main/resources/docker/docker-
compose.yml</composeFile>
<detachedMode>true</detachedMode>
</configuration>
</execution>
</executions>
</plugin>
```

3- Creamos nuestro archivo docker-composer.yml en el directorio “src/main/resources/docker” y agregamos lo siguiente:

```
version: "3"
services:
  web:
    image: payara/micro:5.182
    ports:
      - "8080:8080"
    volumes:
      - ./HelloWorldWebApp-1.0-
SNAPSHOT.war:/opt/payara/deployments/HelloWorldWebApp-1.0-SNAPSHOT.war
```



4- Compilamos y nos aseguramos de que este corriendo de fondo en nuestros contenedores.