COMARCH
Technologies

Wprowadzenie do Mavena

Paweł Wójcik

2015.07.02

Agenda

COMARCH Technologies

- Wstęp
- Konfiguracja
- Cykl życia
- Zasięg (scope)
- Realizacja celów
- Tworzenie projektu
- Struktura projektu (plik pom.xml)
- Zależności
- Pluginy
- Ćwiczenia praktyczne
- Podsumowanie

- Co to jest Maven?
- W jakim celu go stosujemy?
- Jakie są z tego korzyści?

Konfiguracja

COMARCH Technologies

- https://maven.apache.org -> Download -> Maven 3.2.5 (`apache-maven-3.2.5-bin.zip`)

- `M2_HOME=C:\...\apache-maven-3.2.5`
- `M2=%M2_HOME%\bin`
- zmienna Path - dodanie na końcu `%M2%`
- `mvn -version`
- ew. `JAVA_HOME=C:\...\jdk1.8`…
- zmienna Path - dodanie `%JAVA_HOME%\bin`

Cykl życia (maven build lifecycle)

COMARCH Technologies

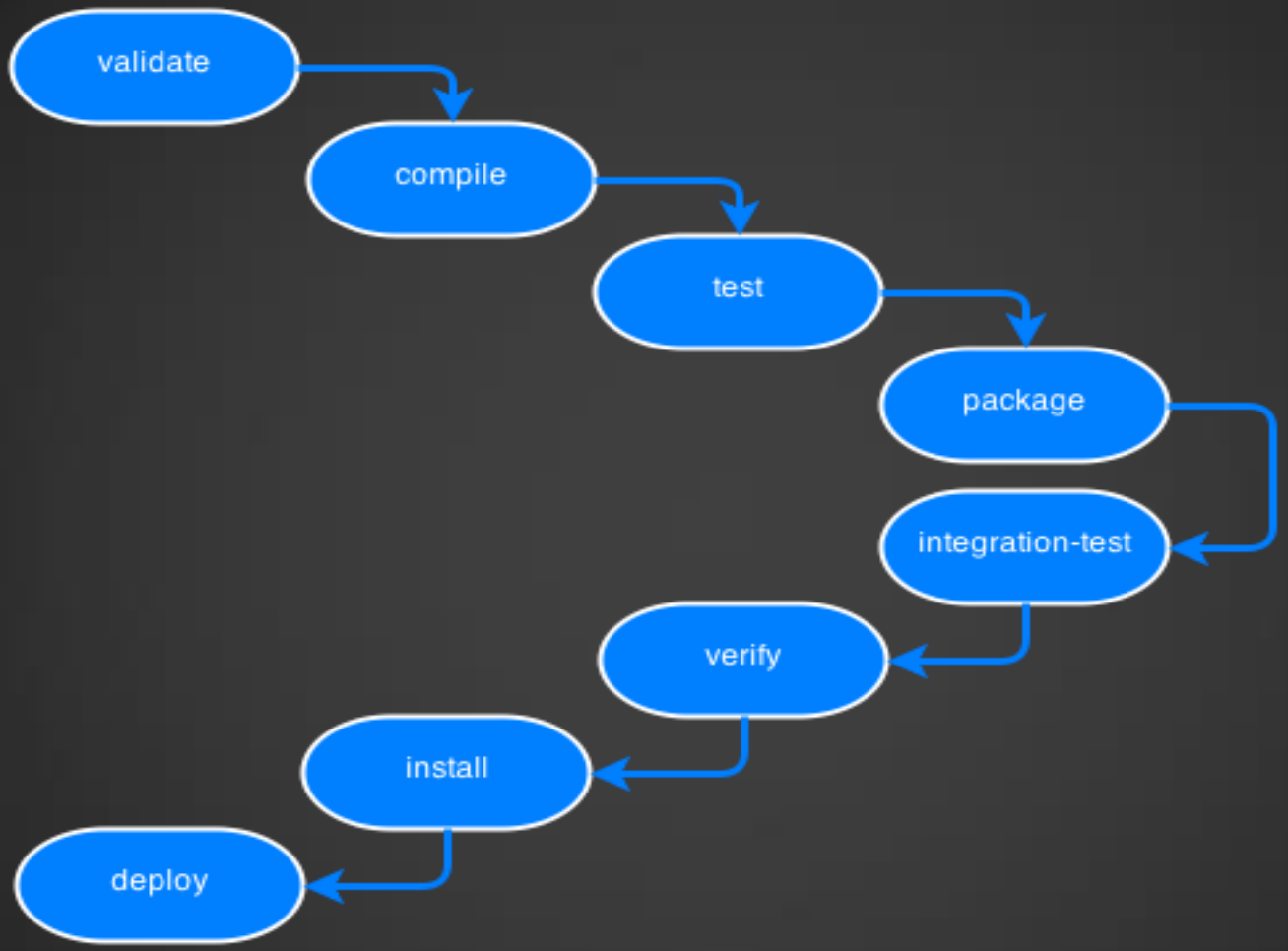
- Fazy domyślnego (**default**) cyklu życia:
  - **validate** - sprawdza poprawność projektu;
  - **compile** - kompiluje kod źródłowy;
  - **test** - wykonuje testy jednostkowe;
  - **package** - pakuje skompilowany kod w paczki dystrybucyjne (np. Jar, war);
  - **integration**-test – deployuje paczkę w środowisku testów integracyjnych;
  - **verify** - sprawdza poprawność paczki;
  - **install** - umieszcza paczkę w lokalnym repozytorium aby mogła być używana przez inne moduły;
  - **deploy** - umieszcza (publikuje) paczkę w zdalnym repozytorium.

- Cykl życia **clean** czyści wynik działania wcześniejszych buildów.

- Cykl życia **site** generuje dokumentację projektu

Najczęściej używane `mvn clean install -DskipTests`

Maven build lifecycle



Rys. 1. maven build lifecycle diagram

Zasięg

COMARCH Technologies

- **compile** - domyślny zasięg – zależności o zasięgu kompilacji są dostępne we wszystkich fazach (kompilacji, uruchomienia) projektu i są propagowane do projektów zależących od mojego projektu.
- **provided** - zasięg zbliżony do compile z tą różnicą, że zakładamy dostępność tych bibliotek w runtime. Przykładem dla użycia provided może być Servlet API – potrzebujemy go do kompilacji, ale w runtime będzie udostępniony przez kontener.
- **runtime** - zasięg zakłada, że biblioteka nie jest potrzebna do kompilacji, ale tylko do uruchomienia (runtime i test).
- **test** - zasięg wskazuje, że biblioteka nie jest wymagana do normalnej pracy aplikacji i jest potrzebna jedynie w fazie testów.
- **system** - zasięg zbliżony do provided z tą różnicą, że jawnie wskazujemy jara zawierającego bibliotekę.
- **import** *(dostępny od Mavena 2.0.9)* – zasięg dostępny jedynie dla zależności typu pom – wskazuje Mavenowi, że chcemy dołożyć do naszego projektu zależności zdefiniowane w innym projekcie.

Realizacja celów

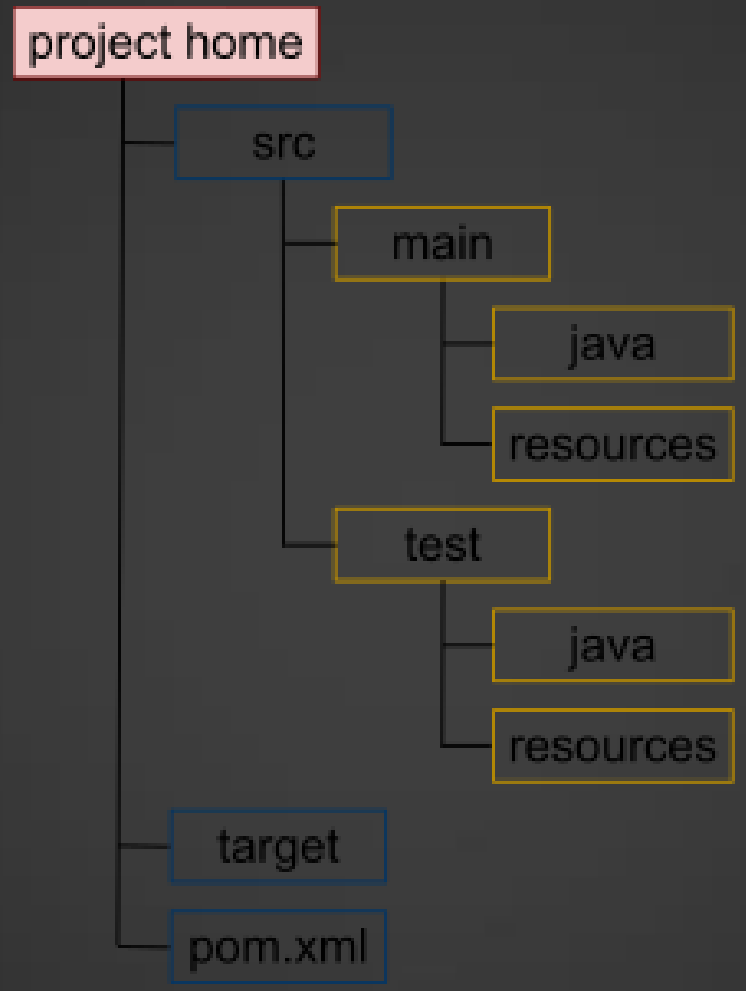
COMARCH Technologies

```
mvn [plugin]:[goal]
```

- **Przykład realizacji** (tworzenie nowego projektu)

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

Struktura projektu



Rys. 2. Struktura generowania plików

Struktura projektu

COMARCH Technologies

```
<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/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.comarch</groupId>
<artifactId>-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>Maven Quick Start Archetype</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>
</project>
```

Dokładny opis atrybutów - http://www.javaexpress.pl/article/show/Maven_2__jak_ulatwic_sobie_prace_cz_I

Zależności

COMARCH Technologies

C:\katalog_domowy\.m2\repository

Przykład:

```
<dependency>
	<groupId>javax.persistence</groupId>
	<artifactId>persistence-api</artifactId>
	<version>1.0.2</version>
</dependency>
```

Dziedziczenie oraz agregacja

COMARCH Technologies

Dziedziczenie obejmuje:

- zależności,
- deweloperów i kontrybutorów,
- listę wtyczek,
- listę raportów,
- wykonania i identyfikatory wtyczek,
- konfigurację wtyczek.

```
<parent>
   <groupId>com.comarch.fsm</groupId>
   <artifactId>master-config</artifactId>
   <version>1.13</version>
</parent>
```

Agregacja pozwala tworzyć projekty wielomodułowe (agregacyjne). Moduły stanowią projekty wymienione w POMie, które wykonywane są jako grupa. Moduły (pod-projekty) są względnymi katalogami w strukturze projektu nadrzędnego:

```
<modules>
   <module>backend</module>
   <module>frontend</module>
</modules>
```

Przykład:

```
<plugin>
	<groupId>org.apache.maven.plugins</groupId>
	<artifactId>maven-compiler-plugin</artifactId>
	<version>2.3.2</version>
	<configuration>
		<source>1.8</source>
		<target>1.8</target>
	</configuration>
</plugin>
```

Przykłady

COMARCH Technologies

Przykład:

```
<groupId>com.comarch</groupId>
<artifactId>parent</artifactId>
<version>1.0.0</version>
<packaging>pom</packaging>

<properties>
          <jdk.version>1.7</jdk.version>
          <spring.version>4.1.1.RELEASE</spring.version>
          <jstl.version>1.2</jstl.version>
          <junit.version>4.11</junit.version>
          <logback.version>1.0.13</logback.version>
</properties>

<dependency>
          <groupId>${project.groupId}</groupId>
          <artifactId>${project.artifactId}</artifactId>
           <version>${project.version}</version>
</dependency>
```

Przykład2:

```
<plugin>
   <groupId>org.liquibase</groupId>
   <artifactId>liquibase-maven-plugin</artifactId>
   <version>3.1.1</version>
   <executions>
      <execution>
         <id>changelog-update</id>
         <phase>process-resources</phase>
         <configuration>
            <skip>false</skip>
            <promptOnNonLocalDatabase>false</promptOnNonLocalDatabase>
            <propertyFileWillOverride>true</propertyFileWillOverride>
            <propertyFile>liquibase-changelog-
update.properties</propertyFile>
         </configuration>
         <goals>
            <goal>update</goal>
         </goals>
      </execution>
      <execution>
         <id>main-update</id>
         <phase>process-resources</phase>
         <configuration>
            <skip>false</skip>
            <promptOnNonLocalDatabase>false</promptOnNonLocalDatabase>
            <propertyFileWillOverride>true</propertyFileWillOverride>
            <propertyFile>liquibase.properties</propertyFile>
         </configuration>
         <goals>
            <goal>update</goal>
         </goals>
      </execution>
   </executions>
</plugin>
```

Ćwiczenia

COMARCH Technologies

1. **Tworzenie nowego projektu z poziomu konsoli**

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2. Dodanie dowolnej zależności oraz pluginu do pliku pom.xml

3. Wykonać na projekcie:
   - Kompilację projektu (`mvn compile`)
   - Wygenerowanie artefakty bez pakietowania (`mvn jar:jar`)
   - Wyczyścić projekt (`mvn clean`)
   - Wykonać pakietowanie (`mvn package`)
   - Zainstalować artefakty w lokalnym repo (`mvn install`)

4. Zapoznać się z projektem `maven-multi.zip` (wielomodułowość) oraz wygenerować i zainstalować artefakty projektu głównego.

5. Ćwiczenia w Eclipse (STS)
   - Tworzenie projektu..

...

Dziękuję za uwagę!