

# Maven plugins

# Maven Compiler Plugin

- Jak vynutit kompilaci kódu pomocí určité verze Javy? Pokud chcete, aby byl Váš kód zkompilován pod Java 7, pak přidejte do pom.xml do tagu <plugins> tento plugin:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <configuration>
    <source>1.7</source>
    <target>1.7</target>
  </configuration>
</plugin>
```

# Maven War plugin

- Od Servlet 3.0 specifikace je u webové aplikace web.xml soubor nepovinný.
- Pokud při buildu Maven vyhodí chybu Error assembling WAR: webxml attribute is required, pak přidejte do pom.xml tento plugin:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </configuration>
</plugin>
```

# Maven Jetty Plugin

- Pro tvorbu web. aplikací je nejlepší použít Servlet kontejner (Jetty nebo Apache Tomcat).
- Abyste Servlet kontejner nemuseli stahovat a integrovat ho s vývojovým prostředím, je možné použít jeho embedded verzi:

```
<plugin>
  <groupId>org.eclipse.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId>
  <version>9.1.0.RC0</version>
  <configuration>
    <scanIntervalSeconds>1</scanIntervalSeconds>
  </configuration>
</plugin>
```

Spuštění: `mvn jetty:run`

Po spuštění bude běžet na portu 8080

# Maven Tomcat Plugin

- Můžete také použít embedded Tomcat:

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.0</version>
  <configuration>
    <contextFile>src/main/webapp/WEB-INF/context.xml</contextFile>
  </configuration>
</plugin>
```

- Pro automatický redeploy při změně nějakého souboru vytvořte soubor `src/main/webapp/WEB-INF/context.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context reloadable="true" backgroundProcessorDelay="1">
</Context>
```

Spuštění: `mvn tomcat7:run`

# Maven Javadoc Plugin

- K vygenerování Javadoc dokumentace se používá Maven Javadoc plugin.
- Po pom.xml do tagu <plugins> přidejte tento plugin:

```
<plugin>  
  <groupId>org.apache.maven.plugins</groupId>  
  <artifactId>maven-javadoc-plugin</artifactId>  
  <version>2.9.1</version>  
</plugin>
```

- Pro vygenerování Javadoc dokumentace zavolejte:

```
mvn javadoc:javadoc
```

- Javadoc dokumentace bude vygenerována do adresáře target/site

# Maven Enforcer Plugin

- Maven Enforcer Plugin slouží k vynucení parametrů buildovacího prostředí (jako je verze Mavenu a Javy, typ operačního systému a další). Standardně se spouští v build fázi Mavenu validate. Abyste například vynutili, že je nutné aplikaci buildovat pouze na Windows, přidejte do pom.xml:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-enforcer-plugin</artifactId>
  <version>1.3.1</version>
  <executions>
    <execution>
      <id>enforce-versions</id>
      <goals> <goal>enforce</goal> </goals>
      <configuration>
        <rules> <requireOS> <family>windows</family> </requireOS> </rules>
      </configuration>
    </execution>
  </executions>
</plugin>
```

# Maven Assembly Plugin I.

- Pokud chcete vytvořit Java SE aplikaci a zabalit všechny závislosti do jednoho JAR souboru, pak přidejte do pom.xml do tagu <plugins> tento plugin:

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <archive>
      <manifest> <mainClass>cz.java.skoleni.helloworld.Main</mainClass> </manifest>
    </archive>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
</plugin>
```

- Vytvoření jednoho JAR souboru se závislostmi provedete pomocí:  
`mvn compile assembly:single`
- Výsledný JAR soubor bude v adresáři target.



# Maven Assembly Plugin II.

- Pokud potřebujete vytvořit z celého projektu binární distribuci (například ZIP soubor), pak také použijete Maven Assembly Plugin spolu se souborem Assembly Descriptor, ve kterém definujete co vše má být ve výsledném souboru – většinou toho tam chcete víc než jenom JAR / WAR soubor ;-)
  - <http://maven.apache.org/plugins/maven-assembly-plugin/>
  - <http://maven.apache.org/plugins/maven-assembly-plugin/descriptor-refs.html>
- Například když máte v rootu projektu soubory LICENSE.txt a README.md a chcete je zkopírovat do výsledného ZIP souboru spolu s Vaším WAR souborem.
- Na dalších stranách je konfigurace pluginu a Assembly Descriptor souboru.

# Maven Assembly Plugin III.

- Do pom.xml přidejte tento plugin:

```
<!-- assembly -->
```

```
<plugin>
```


```
  <artifactId>maven-assembly-plugin</artifactId>
```

```
  <version>2.4</version>
```

```
  <configuration>
```

```
    <appendAssemblyId>>false</appendAssemblyId>
```

Název výsledného  
souboru bez přípony



```
    <finalName>${project.artifactId}-bin-${project.version}</finalName>
```

```
    <descriptors>
```

```
      <descriptor>src/main/resources/assembly/bin.xml</descriptor>
```

```
    </descriptors>
```

```
  </configuration>
```

```
</plugin>
```

Kde je Assembly Descriptor soubor



# Maven Assembly Plugin IV.

- Vytvořte soubor `src/main/resources/assembly/bin.xml`:

```
<assembly
  xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.2
    http://maven.apache.org/xsd/assembly-1.1.2.xsd">
  <id>bin</id>
  <formats>
    <format>zip</format>
  </formats>
  <fileSets>
```

**pokračování**



# Maven Assembly Plugin V.

```
<fileSet>  
  <directory>${project.basedir}</directory>  
  <outputDirectory>/</outputDirectory>  
  <includes>  
    <include>LICENSE*</include>  
    <include>README*</include>  
  </includes>  
</fileSet>
```

Kde jsou soubory

Kam se do ZIP souboru uloží

Jaké soubory / adresáře se budou kopírovat.  
Používá se Ant syntaxe:

\* = část názvu souboru  
\*\* = podadresář

```
<fileSet>  
  <directory>${project.build.directory}</directory>  
  <outputDirectory>/</outputDirectory>  
  <includes>  
    <include>*.war</include>  
  </includes>  
</fileSet>
```

```
</fileSets>
```

```
</assembly>
```

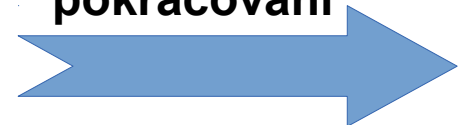
Pro spuštění se použije:  
mvn clean package assembly:single

# Maven Resources Plugin I.

- Co když potřebujete do výsledného WAR / JAR souboru v závislosti na tom, jestli aplikaci vyvíjíte nebo běží na produkci vložit příslušný konfigurační soubor?
- Vložte do pom.xml do tagu <project>:

```
<profiles>
  <profile>
    <id>prod</id>
    <build>
      <resources>
        <resource>
          <directory>src/main/resources-prod</directory>
        </resource>
      </resources>
    </build>
  </profile>
```

**pokračování**



# Maven Resources Plugin II.

```
<profile>

  <id>dev</id>

  <build>

    <resources>

      <resource>

        <directory>src/main/resources-dev</directory>

      </resource>

    </resources>

  </build>

</profile>

</profiles>
```

Spuštění dev profilu: `mvn clean package -P dev`  
Spuštění prod profilu: `mvn clean package -P prod`

# Maven Antrun Plugin

- Jestli potřebujete pomocí Mavenu spustit externí aplikaci nebo provést něco pomocí ANTu, pak k tomu použijete maven-antrun-plugin.
- Pro spuštění externí aplikace (v mém případě start.bat), která je v domovském adresáři projektu je nutné do pom.xml do tagu <plugins> přidat tento plugin:

```
<plugin>
```

```
  <groupId>org.apache.maven.plugins</groupId>
```

```
  <artifactId>maven-antrun-plugin</artifactId>
```

```
  <version>1.7</version>
```

```
  <executions>
```

```
    <execution>
```

```
      <id>default-cli</id>
```

```
      <goals> <goal>run</goal> </goals>
```

```
      <configuration>
```

```
        <target>
```

```
          <exec executable="cmd" dir="${basedir}" spawn="true"> <arg value="/c" /> <arg value="start.bat" /> </exec>
```

```
        </target>
```


```
      </configuration>
```

```
    </execution>
```

```
  </executions>
```

```
</plugin>
```

Pokud uvedete v tagu <exec> atribut spawn="true", pak Maven nečeká na její dokončení (vhodné pro spuštění nějakého podpůrného serveru jako je například testovací databáze).



Pro spuštění zavolejte: mvn antrun:run

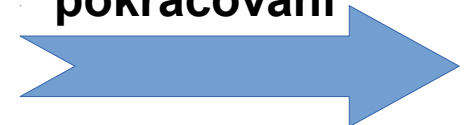
# Maven Exec plugin I.

- Maven exec plugin slouží ke spuštění externí aplikace:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.2.1</version>
  <executions>
    <execution>
      <phase>install</phase>
      <goals>
        <goal>exec</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Fáze exec Maven Exec pluginu se vykoná při zavolání install fáze (`mvn install`). Také by bylo možné místo toho tuto fázi vykonat zavoláním `mvn exec:exec`

**pokračování**





# Maven Exec plugin II.

```
<configuration>
  <executable>heroku</executable>
  <arguments>
    <argument>deploy:war</argument>
    <argument>--war</argument>
    <argument>target/sitemonitoring.war</argument>
    <argument>--app</argument>
    <argument>sitemonitoring</argument>
  </arguments>
</configuration>
</execution>
</executions>
</plugin>
```



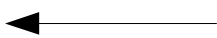
Tento plugin udělá to samé,  
jako kdybych z příkazové řádky zavolal:  
heroku deploy:war -war target/sitemonitoring.war -app sitemonitoring

# Maven Release Plugin

- Maven Release Plugin zjednodušuje verzování projektu při tvorbě nové verze projektu.
- Na začátku posloupnosti musí název verze končit textem SNAPSHOT. Pak se zvýšení verze projektu skládá z posloupnosti těchto kroků:

```
mvn clean
```

```
git commit ...
```



```
mvn release:prepare
```

```
mvn release:perform
```

Můžete používat i něco jiného než GIT:

<https://maven.apache.org/scm/scms-overview.html>

# Maven Release Plugin: SCM

- Do pom.xml je nutné přidat připojení do GITu (nebo jiného verzovacího nástroje).

- Příklad na Github:

<scm>

Přístup, který se použije pro čtení,  
vyžaduje min. read access

<connection>scm:git:git@github.com:JMENO/PROJEKT.git</connection>

<developerConnection>scm:git:git@github.com:JMENO/PROJEKT.git</developerConnection>

<url>git@github.com:JMENO/PROJEKT.git</url>

<tag>HEAD</tag>

</scm>

Přístup, který se použije pro zápis,  
vyžaduje write access

Veřejný přístup pro procházení repozitáře.  
Zatím jsem nezjistil praktické použití kromě  
dokumentace pro jiné vývojáře.

# SCM SVN I.

- Pro Subversion vypadá tag <scm> následovně:

```
<scm>
```

```
  <connection>scm:svn:svn://SERVER/URL_PROJEKTU</connection>
```

```
  <developerConnection>scm:svn:svn://SERVER/URL_PROJEKTU</developerConnection>
```

```
  <url>svn://SERVER/URL_PROJEKTU</url>
```

```
  <tag>HEAD</tag>
```

```
</scm>
```

# SCM SVN II.

- Uživatelské jméno a heslo můžete nastavit tímto způsobem:

```
<plugin>
```

```
  <groupId>org.apache.maven.plugins</groupId>
```

```
  <artifactId>maven-release-plugin</artifactId>
```

```
  <version>2.5.2</version>
```

```
  <configuration>
```

```
    <username>TODO</username>
```

```
    <password>TODO</password>
```

```
  </configuration>
```

```
</plugin>
```


# Maven release plugin: Prepare

- Příklad začíná verzí 1.0-SNAPSHOT
- `mvn release:prepare` provede následující posloupnost kroků:
  - Zjistí, jestli je vše commitované. Pokud ne, pak se vyhodí chyba.
  - Zjistí, jestli v `pom.xml` nejsou používané SNAPSHOT dependency. Pokud ano, pak se zeptá, čím se mají nahradit. Není možné pokračovat bez odstranění SNAPSHOT dependency.
  - Zeptá se na názvy. Standardně:
    - Změní verzi z 1.0-SNAPSHOT na 1.0.
    - Nový název verze bude 1.1-SNAPSHOT.
    - V repozitáři vytvoří tag `NÁZEV_PROJEKTU-1.0`
  - Provede přidání tagu v repozitáři, commit a push.
  - Postupně v jednotlivých krocích vytváří soubor `release.properties`.

# Maven release plugin: konfigurace I.

- V konfiguraci Maven release pluginu je možné definovat řadu konfiguračních parametrů:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.4.2</version>
  <configuration>
    <autoVersionSubmodules>true</autoVersionSubmodules>
  </configuration>
</plugin>
```



Pokud máte multi-module projekt,  
pak se budou moduly verzovat  
stejným číslem jako parent.

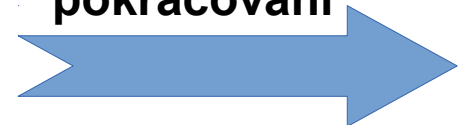
# Maven release plugin: konfigurace II.

- Nastavte ve svém settings.xml souboru:

```
<servers>
  <server>
    <id>deployment</id>
    <username>admin</username>
    <password>admin123</password>
  </server>
</servers>
```

↑  
Výchozí uživ.  
jméno a heslo  
v Sonatype  
Nexus

**pokračování**





# mvn deploy II.

- V příslušném projektu musí být nastavena adresa kam se provede deploy SNAPSHOT verzí (název verze končí na SNAPSHOT) a ostatních verzí:

```
<distributionManagement>
  <snapshotRepository>
    <id>deployment</id>
    <name>internal snapshots</name>
    <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
  </snapshotRepository>
  <repository>
    <id>deployment</id>
    <name>internal releases</name>
    <url>http://localhost:8081/nexus/content/repositories/releases/</url>
  </repository>
</distributionManagement>
```

# Maven release plugin: workflow

- Pokud se `mvn release:prepare` nezdaří (BUILD FAILURE), pak je možné:
  - Znovu zavolat `mvn release:prepare`, bude se pokračovat od bodu, ve kterém se skončilo.
  - Zavolat `mvn release:clean`. Toto smaže `release.properties` a dostanete projekt do stavu před prvním zavoláním `mvn release:prepare`.
- Pokud `mvn release:prepare` úspěšně doběhne do konce (BUILD SUCCESS), pak je možné:
  - Zavolat `mvn release:rollback`. Tímto se dostanete do stavu před zavoláním `mvn release:prepare`.
  - Zavolat `mvn release:clean`. To uděláte, pokud nechcete následně provést `mvn release:perform`.
  - Zavolat `mvn release:perform`

# Maven release plugin: rollback

- `mvn release:rollback` často volat nebudete. Kdybyste tuto operaci ale potřebovali zavolat, pak je důležité mít na paměti, že se změní verze v `pom.xml` souboru, ale neprovede se žádná změna v SCM!
- Pokud používáte Git, pak je nutné manuálně smazat tag nové verze, který se vytvoří:

```
git tag -d 12345
```

Název tagu

```
git push origin :refs/tags/12345
```

# Maven release plugin: perform

- Pokud chcete novou verzi Vaší aplikace nahrát do vzdáleného Maven repozitáře (Nexus / Artifactory), poté je nutné:
  - V `pom.xml` v tagu `distributionManagement` nadefinované repozitáře v tagu `repository` a `snapshotRepository`.
  - V `settings.xml` v tagu `servers` mít uživatelské jméno a heslo ke vzdálenému repozitáři.
  - Zavolat `mvn release:perform`

# Maven release plugin: Batch & Eclipse

- Je možné provádět operace Maven release pluginu v batch módu, což je užitečné, když je chcete vyvolávat na serveru jako je Jenkins.
  - <http://maven.apache.org/maven-release/maven-release-plugin/examples/non-interactive-release.html>
- V Eclipse je možné operace Maven release pluginu volat po integraci externí Maven instalace (Window → Preferences → Maven → Installations ...). Aktuálně to s embedded Maven serverem nefunguje.

# Tvorba vlastních pluginů I.

- Vytvořte projekt podle tohoto archetypu:
  - <http://javalibs.com/archetype/org.apache.maven.archetypes/maven-archetype-plugin>
    - groupId: com.test
    - artifactId: hello-plugin
    - version: 0.0.1-SNAPSHOT

# Tvorba vlastních pluginů II.

- Změňte třídu MyMojo:

```
import org.apache.maven.plugin.*;

import org.apache.maven.plugins.annotations.*;

import org.apache.maven.plugins.annotations.Mojo;

@Mojo(name = "touch", defaultPhase = LifecyclePhase.PROCESS_SOURCES)

public class MyMojo extends AbstractMojo {

    @Parameter(property = "echoMessage")

    private String echoMessage;

    public void execute() throws MojoExecutionException {

        getLog().info("ECHO: " + echoMessage);

    }

}
```

# Tvorba vlastních pluginů III.

- Použití 1:

1. Zavolejte v domovském adresáři pluginu: `mvn install`

2. Přidejte do projektu:

```
<plugin>

  <groupId>com.test</groupId>

  <artifactId>hello-plugin</artifactId>

  <version>1.0-SNAPSHOT</version>

  <configuration>
    <echoMessage>Hello World!</echoMessage>
  </configuration>
</plugin>
```

3. Zavolejte v domovském adresáři projektu:

`hello-plugin:touch`



# Tvorba vlastních pluginů IV.

- Nebo přidejte do projektu:

```
<plugin>
  <groupId>com.test</groupId>
  <artifactId>hello-plugin</artifactId>
  <version>1.0-SNAPSHOT</version>
  <configuration>
    <echoMessage>Hello World!</echoMessage>
  </configuration>
  <executions>
    <execution>
      <phase>compile</phase>
      <goals>
        <goal>touch</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- A zavolejte v domovském adresáři projektu: `mvn compile`

# Tvorba vlastních pluginů V.

- Jak vyřešit chybu „Plugin execution not covered by lifecycle configuration“ v Eclipse:
  - <https://www.eclipse.org/m2e/documentation/m2e-execution-not-covered.html>