Knowledge Transfer

MAVEN

November, 2014

Sherban Popescu
Olivier Dupuy

Welcome to the MAVEN…

presentation …

Why MAVEN

- What is a Maven?
  - *A tool for building and managing Java-based projects (maven.apache.org/what-is-maven.html)*

- Clear advantages over other traditional solutions (Ant). Maven can manage:
  - Builds
  - Documentation
  - Dependencies
  - Releases
  - SCM (ex: plugin for SVN)
- *Convention* over *Configuration*
  - Maven offers powerful **plugins** for automating the building and assembling of software
  - Existing defaults cover most of the common usages (most of the defaults can be customized)
  - Common interface for building software – can be as simple as:
    - Check out project
    - Run ***mvn install***

MAVEN – Hands on

- Demo for creating a project
  - show basic and effective POM
  - show standard Maven directory layout

A Maven Project - concepts

- Each project defined in pom.xml (Maven is declarative)
  - effective POM (user defined POM plus inherited POM)
- Project lifecycle
- Project dependencies
- Set of plugin goals to be executed at defined phases in the lifecycle

  (a plugin is a collection of goals like Jar plugin, Compile plugin, Surefire plugin, etc.)

Project coordinates: unique project identifier

Name	Description
groupId	Defines a unique base name of the organization or group that created the project. Ex: com.oacis.osi (similar to a package name)
artifactId	Defines the unique name of the project. Ex: OSIPlatformRest
version	This defines the version of the project. Ex: 3.0.0

Maven standard directory layout

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/config	Configuration files
src/main/scripts	Application/Library scripts
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

Maven POM (Project Object Model)

- The POM contains all necessary information about a project, as well as information about the plugins used in the build process.
- described in detail at http://maven.apache.org/pom.html
  - **minimal POM:**

```
<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>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>

</project>
```

Maven POM (Project Object Model) – contd.

- packaging type: default is jar, but other types are:

  pom, jar, maven-plugin, ejb, war, ear

```
<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">
  ...
  <packaging>war</packaging>
  ...
</project>
```

- The types define the default list of goals that execute to each corresponding build lifecycle stage

- POMs are hierarchical: POMs that extend a parent POM inherit certain values from that parent

Maven Build Lifecycle

- is a well defined sequence of **phases** which define the order in which the goals are to be executed

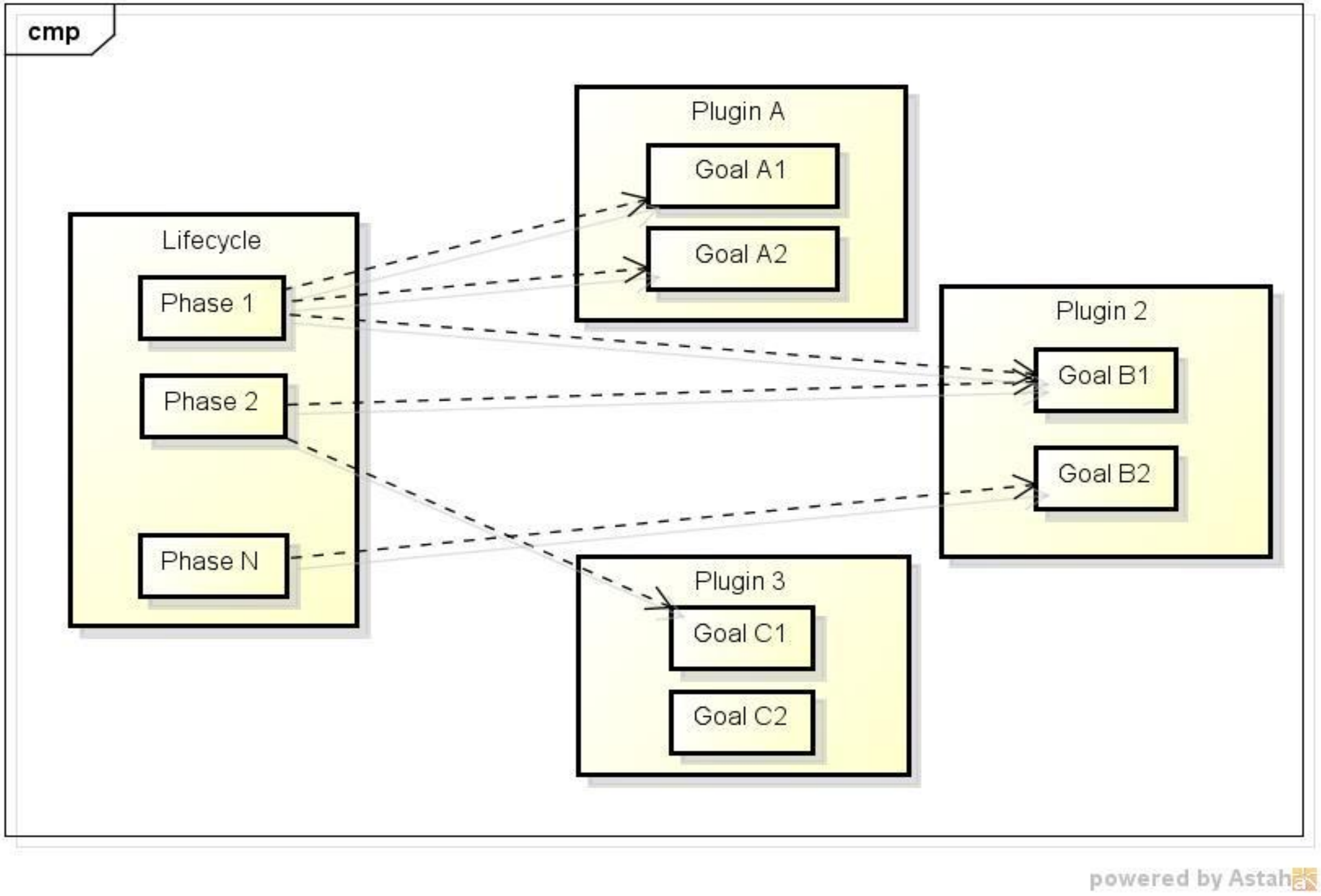
- A **phase** is a specific stage in the build process in which certain goals are executed

- Typical sequence of phases:
  - Prepare-resources (resource copying)
  - Compile (java source compilation)
  - Package (jar, war, etc.)
  - Install (copy the packages to local repository, making it available to other projects locally)

- A **goal** represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases.
  - Ex: mvn clean dependency:copy-dependencies package

  (*clean* and *package* are phases, *dependency:copy-dependencies* is a goal)

Maven Build Lifecycle



Maven Lifecycle example: Build(Default) Phase

- **validate** - checks if the project is correct and all information is available
- **compile** - compiles source code in binary artifacts
- **test** - executes the tests
- **package** - takes the compiled code and package it, for example into a JAR file.
- **integration-test** - takes the packaged result and executes additional tests, which require the packaging
- **verify** - performs checks if the package is valid
- **install** - install the result of the package phase into the local Maven repository
- **deploy** - deploys the package to a target, i.e. remote repository

Maven Lifecycle Execution

- As Maven moves through the phases in a lifecycle, it will execute the goals attached to each particular phase.
- Each phase may have zero or more goals bound to it.
  - Ex: **mvn install – MAVEN** executes all phases from the beginning and up to the <u>install</u> phase. For each phase it executes it's all goals
  - **mvn install –** is equivalent to the following command:

```
mvn resources:resources \
compiler:compile \
resources:testResources \
compiler:testCompile \
surefire:test \
jar:jar \
install:install
```

- More convenient to use the short form based on lifecycle than using the plugin goals

Maven Lifecycle Execution

- Maven has 3 built-in life cycles. These are:
  - default
  - clean
  - site

- Since the default lifecycle cannot be executed directly, a phase or goal has to be specified:
  - mvn install
  - mvn package

- There are lifecycles that can be executed directly:
  - mvn clean
  - mvn site

- To obtain help info from maven about a specific phase or goal:
  - mvn help:describe -Dcmd=compile (phase)
  - mvn help:describe -Dcmd=compiler:compile (goal)

Maven repositories

- A repository is a collection of project artifacts (libraries, plugins and metadata) stored in a directory structure that closely matches a project's coordinates

- Three types of repositories:
  - **Local** repository
  - **Central** (Maven) repository
  - **Remote** repository (user defined repositories)

Maven repositories

- Plugins and other artifacts are downloaded as needed
- Maven core is very small, no plugins until they are required
- Default remote repository location is search.maven.org
  - From here it downloads the artifacts
- From remote repositories, the artifacts are downloaded to the local repository (C:\Users\T878550\.m2\repository)
- After downloading, only the local repository will be used for that artifact
- Users can define their own repository for jar files that are not provided by the Maven Central Repository, but are needed for the project. This is specified in the pom file by the <repository> tag.

Maven repositories – Internal Repositories

- Another *remote* repository (from Maven’ view point)
- When connecting to Internet is not desirable for the organization

```
<repositories>
        <repository>
                <id>my-local-repo</id>
                <url>file://${project.basedir}/../repo</url>
        </repository>
</repositories>
```

Maven dependency management

- Maven starts looking for dependencies in the following sequence:
  - Search in local repository
  - Search in the central (Maven) repository
  - Search in remote repository (user defined repositories)

- We only need to define direct dependency in each project pom. Maven handles the rest of dependencies (transitive) automatically.

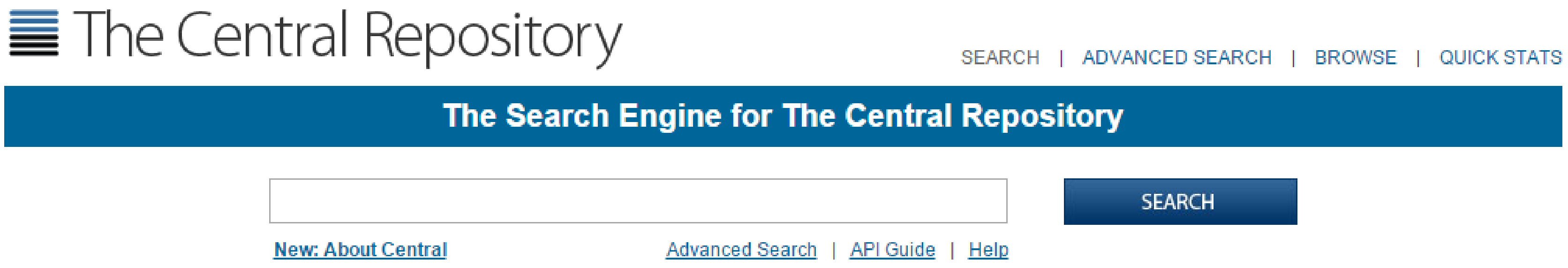
- Common dependencies can be placed at a single place using concept of parent pom (ex: third party libraries)

- Dependency <u>scope</u>: some jar files can be made available only for certain phases:
  - Ex**: <u>test</u>** scope: indicates that the dependency is only available for the test phase, so the respective jar will **not** be packaged for distribution

Maven dependency management

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
            <version>1.3.2</version>
            <scope>test</scope>
        </dependency>
        ...
    </dependencies>
</project>
```

Maven dependency management – Central Repo.


The Central Repository
SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS
The Search Engine for The Central Repository
SEARCH
New: About Central
Advanced Search | API Guide | Help

MAVEN – Hands on

- compile the project
- run unit tests
- add/modify dependencies
- add resources
- package
- show the JAR files pushed to the repository (local)
- show transitive dependencies in POM files
- show Eclipse support for Maven

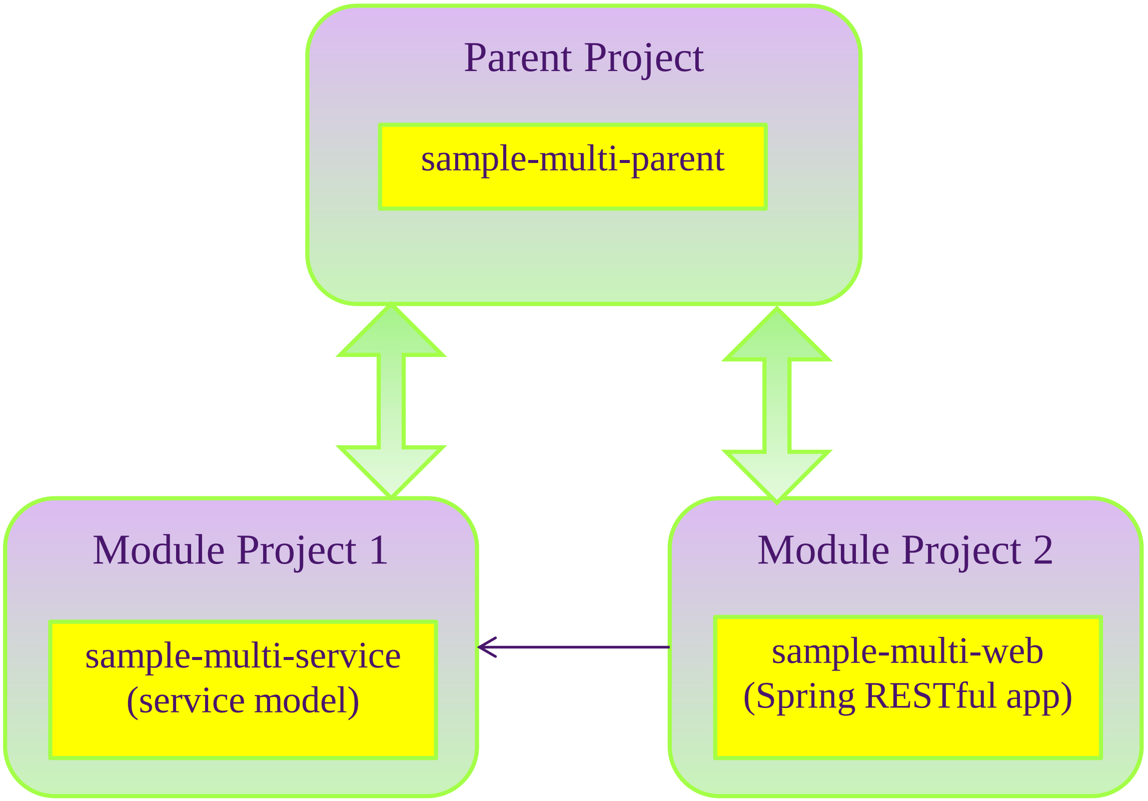
Maven support for multi module projects

sample-multi-parent
sample-multi-service
sample-multi-web

sample-multi-parent/:
pom.xml

sample-multi-service/:
pom.xml

sample-multi-web/:
pom.xml



Maven support for multi module projects (example)

```
<groupId>com.kt.demo</groupId>
<artifactId>sample-multi-parent</artifactId>
<version>1.0-SNAPSHOT</version>

<packaging>pom</packaging>
<name>sample-multi-parent</name>

<properties>
    <junit.version>4.11</junit.version>
    <spring.version>4.0.0.RELEASE</spring.version>
    <jackson.version>2.3.2</jackson.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<modules>
    <module>sample-multi-service</module>
    <module>sample-multi-web</module>
</modules>
```

Maven support for multi module projects (example)

(Child) POM.xml

```
<artifactId>sample-multi-service</artifactId>
<packaging>jar</packaging>

<parent>
    <groupId>com.kt.demo</groupId>
    <artifactId>sample-multi-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
    <relativePath>../sample-multi-parent</relativePath>
</parent>

<name>sample-multi-service</name>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
</dependencies>
```

Maven support for multi module projects (example)

(Child) POM.xml

```
    <artifactId>sample-multi-web</artifactId>
    <packaging>war</packaging>

    <name>sample-multi-web</name>

    <parent>
        <groupId>com.kt.demo</groupId>
        <artifactId>sample-multi-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../sample-multi-
parent</relativePath>
    </parent>

    <dependencies>
        <dependency>
            <groupId>com.kt.demo</groupId>
            <artifactId>sample-multi-service</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
    </dependencies>

    <build>
        <finalName>sample-multi-webapp</finalName>
    </build>
```

MAVEN – Hands on

- create a parent project of the existing project (JAR)
- make the JAR project as module
- create a new WAR project as module for the same parent
- make the WAR project dependent on the JAR project
- create a Bill of Materials (BOM)
- manage dependencies (good and bad ones)
- show provided dependencies
- show Maven site lifecycle

Wrapping up …the good and the bad

- Complete project management solution, not merely a build script (ant)
- Great with managing libraries/dependencies
- Standardized project layout
- Existing plugins for most of developer needs
- Can easily run a complex lifecycle(e.g. compile, test, deploy, integration tests)
- Most suitable for building projects that all depend on each other
- Steep learning curve, POM is XML based and can become complex
- Documentation can be cryptic sometimes
- Can become inflexible for doing something that runs contrary to Maven’s conventions (you may end up writing a plugin or calling ANT)

MAVEN – Questions?

?

MAVEN

Thank You!