# Advanced Build

Ádám Révész, Attila Ulbert, Norbert Pataki, Zoltán Gera

Eötvös Loránd University,
Budapest, Hungary

Introduction to
Cloud Technologies

## Contents

## Features

- Platform-independency
- XML-based syntax or DSL usage
- Abstraction: dependency
- Paradigm: not necessarily imperative
- Artifact repositories
- Extensions/plugins
- Examples:
  - Ant (`http://ant.apache.org`)
  - Maven (`http://maven.apache.org`)
  - Gradle (`http://gradle.org`)

## Differences in Operating Systems

- `cp` **vs.** `copy`
- `rm` **vs.** `del`
- ...

## Ant's Features

- Imperative
- Typically: Java
- `build.xml`

## build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="projname"
         default="deftarget"
         basedir=".">

  ...

</project>
```

## target

```
<project ...>

  <property name="jarname" value="filename.jar" />


  <target name="compile" depends="prepare">

    ...
  </target>

</project>
```

- Command line:
  `ant compile`

## target

```
<project ...>

  <target name="prepare">
    <mkdir dir="classes"/>
  </target>

</project>
```

## target

```
<project ...>

  <target name="compile" depends="prepare">
    <javac srcdir="src"
           destdir="classes"/>
  </target>

</project>
```

## delete task

```
<project ...>

  <target name="clean">
    <delete>
      <fileset dir="classes" includes="*"/>
    </delete>
    <delete dir="classes"/>
  </target>

</project>
```

## delete task

```
<project ...>

  <target name="clean" failonerror=false>
    <delete dir="classes"/>
  </target>

</project>
```

## target

```
<project ...>

  <target name="jar" depends="compile">
    <jar destfile="${jarname}">
      <fileset dir="classes">
        <include name="*.class"/>
      </fileset>
      <manifest>
        <attribute name="Main-Class" value="Main"/>
      </manifest>
    </jar>
  </target>

</project>
```

## target

```
<target name="compile" depends="compwsdl,init">
  <javac destdir="build/classes" debug="on">
    <src path="src/java"/>
    <src path="build/java"/>
    <include name="**/*.java"/>
    <exclude name="com/comp/xyz/applet/*.java"/>
    <classpath>
      <fileset dir="lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </javac>
</target>
```

## Copying files

```
<mkdir dir="build/war/WEB-INF/lib"/>
<copy todir="build/war/WEB-INF/lib">
  <fileset dir="lib">
     <include name="*.jar"/>
     <exclude name="servlet-api.jar"/>
     <exclude name="catalina-ant.jar"/>
     <exclude name="el-api.jar"/>
  </fileset>
</copy>
```

## JVM launch

```
<target name="test">
    <java classname="com.comp.foo.TestClient"
          jvmargs="-Xdebug server=y,suspend=n">
        <classpath>
            <fileset dir="lib">
                <include name="*.jar"/>
            </fileset>
        </classpath>
    </java>
</target>
```

## Generating API documentation

```
<target name="doc">
   <tstamp>
     <format property="timestamp"
             pattern="d.M.yyyy"
             locale="en"/>
   </tstamp>
   <mkdir dir="doc"/>
  <javadoc sourcepath="src"
           windowtitle="Project documentation"
           destdir="doc">
    <header><b>Very Important Project</b></header>
    <footer>Javadocs compiled ${timestamp}></footer>
    <fileset dir="src/" includes="**/*.java" />
  </javadoc>
</target>
```

## Invocation of a target

- Automatic invocation of dependencies with manual invocation of an other target (e.g. with different build settings):

```
<antcall target="targetname"/>
```

## Maven's Features

- Software project management tool
- Building project, running tests, managing dependencies, documentation
- Packages: automatic download of dependencies
- Declarative specification of the build process
- Fix, predefined directory structure, conventions
- Typically Java, but it can handle other language via plugins
- `pom.xml`
- `http://maven.apache.org/index.html`

## Project

- Project Object Model
- Project uniquely identified by project's group, artifact Id, version (GAV)
- Project can divided to multiple modules that can be handled independently

## pom.xml

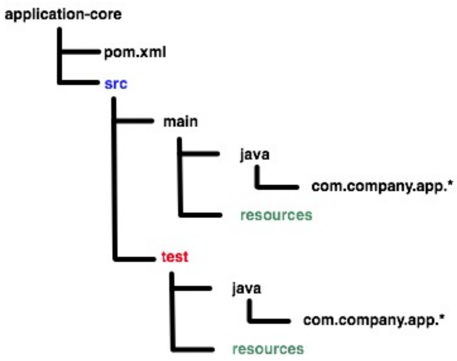
Specification of the project's important information:

- groupId, artifactId, version
- how it is build
- what is the result of the build
- testcases
- dependencies

## pom.xml

```
<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.mycompany.software</groupId>
  <artifactId>app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

    ...

</project>
```

## Directory structure



## Project’s build phases

- 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

## Project's build phases

- 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.
- Further details: `http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`
- Command line:

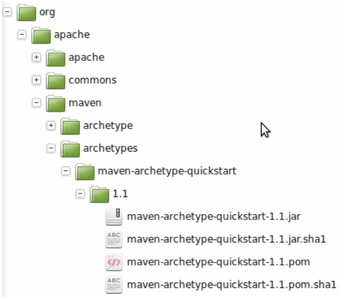
```
mvn clean install
```

## Goals

- Compilation phases consist of goals
- The goal is a task that is related to the project's compilation or management
- Multiple goals can be passed to the Maven in command
- The order of these goals depends on the phase's binding
- Goals can be defined (in the `pom.xml`)

## Repositories

- Central: `http://repo.maven.apache.org`
- Artifact repositories
- Local: own repository on the developer's computer:
  - `~/.m2/repository`



```
org
  apache
    apache
    commons
    maven
      archetype
      archetypes
        maven-archetype-quickstart
          1.1
            maven-archetype-quickstart-1.1.jar
            maven-archetype-quickstart-1.1.jar.sha1
            maven-archetype-quickstart-1.1.pom
            maven-archetype-quickstart-1.1.pom.sha1
```

## Repository

- Maven downloads the dependent artifacts and plugins at the first build, stored in the local repository
- Network traffic and build time are significantly increased
- Incremental changes
- Maven can be configured to use the specified repo or mirror: `~/.m2/settings.xml`

## Results of a build process

- `target` directory is created during compilation which stores the new files that generated at compilation time
- output, e.g. `my-app-1.0-SNAPSHOT.jar`
- `classes` directory – class files that created during compilation but not test classes
- `test-classes` – classes created from test sources
- `maven-archiver` – `pom.properties` file that defines project's GAV
- `surefire-reports` – reports of the tests

## Project hierarchies

```
<project ...>

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>parent-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <!-- subprojects -->
  <modules>
    <module>first-child-app</module>
    <module>second-child-app</module>
  </modules>
</project>
```

## Project hierarchies

```
<project ...>

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>parent-app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <groupId>com.mycompany.app</groupId>
  <artifactId>first-child-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
    ...
</project>
```

## Plugins

- Maven's functionality limited to the basic
- Many different plugins are available – e.g. C++, LaTeX, ant build, javadoc.
- One can write own plugin

## Example – javadoc plugin

```
<project ...>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.8.1</version>
        <configuration>
          ...
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

## Dependencies

```
<project ...>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- GAV uniquely specifies the required artifact
- scope defines how we use the the dependency

## Dependencies' scope

- The most important scopes:
  - `compile` – This is the default if unspecified. Dependencies that required by the compilation
  - `runtime` – Dependency required at runtime, but not required at compilation time.
  - `test` – Dependency is not required in production but it is required for the compilation and execution of testcases.

## Gradle

- Language-independent, Groovy-based build system
- projects, build tasks, task relations, plugins and artifact dependencies written in Groovy
- dependency resolution (even in maven-compatible way)
- build logic in plugins (core-set + third-party plugins)
- `gradlew`
- Configuration, build graph

## Gradle's Advantages

- Incremental build
- Parallel build
- Available Java's functionality
- More permissive than Maven

## Gradle’s Disadvantages

- DSL syntax: can be incomprehensible, not straigthforward, alternative syntax elements
- Error diagnostics

## Conclusion

- Modern Build tools
- Modern syntax
- Dependencies
- Different approaches

## Thank You

Thank you for your attention!