Build tools: Make, Ant and Maven.

Christina Zenzes, Pieter van den Hombergh

Fontys Hogeschool voor Techniek en Logistiek

February 21, 2018

What is building?

Building is the action creating a runnable application or a library from input source files.

- A typically modern application consists of multiple source files.
- For compiled languages, the source files need to be compiled to a binary format. C, C++: object files.
- For some languages, the object files must be **linked** to libraries. For C and C++: XXX.ddl, XXX.dynlib or libXX.so.
- The resulting binary is runnable.

Saving Time In Build

When creating big applications (large code base) compilation (and linking) can take quite some time.

- Therefor the we want to minimize the work done in this step: Only (re)compile those files that have changed(by editing) since the last compilation. Based on time stamps.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).

And it should do all this without someone having to press a button.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).
2. Checkout the version you want to build from the version control system

And it should do all this without someone having to press a button.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).
2. Checkout the version you want to build from the version control system
3. Build the system using the/a build script (which also comes from the repository).

And it should do all this without someone having to press a button.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).
2. Checkout the version you want to build from the version control system
3. Build the system using the/a build script (which also comes from the repository).
4. Test the built components and assemblies (unit testing, integration testing)

And it should do all this without someone having to press a button.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).
2. Checkout the version you want to build from the version control system
3. Build the system using the/a build script (which also comes from the repository).
4. Test the built components and assemblies (unit testing, integration testing)
5. If all tests pass, package (jar) the system and TAG it with an appropriate TAG.

And it should do all this without someone having to press a button.

Steps in an automatic build and test process

In a software development process where testing is considered cruicial, the following steps are typical to build and test the software:

1. Create a fresh (empty) directory. (Or delete the contents of an old one).
2. Checkout the version you want to build from the version control system
3. Build the system using the/a build script (which also comes from the repository).
4. Test the built components and assemblies (unit testing, integration testing)
5. If all tests pass, package (jar) the system and TAG it with an appropriate TAG.
6. If any tests fails, bug the developers, e.g. by mailing them

And it should do all this without someone having to press a button.

Simple build tools

Well known tools are

Make

- Make (gnu make, microsoft nmake). A `Makefile` with a peculiar but simple syntax is used.

Simple build tools

Well known tools are

Make

- Make (gnu make, microsoft nmake). A `Makefile` with a peculiar but simple syntax is used.
  - IDE either use them under the water level or is at least able to produce them.

Simple build tools

Well known tools are

Make

- Make (gnu make, microsoft nmake). A `Makefile` with a peculiar but simple syntax is used.
  - IDE either use them under the water level or is at least able to produce them.
  - Generated Makefiles are difficult to understand. The IDE's typically throw in every possible option, without really needing them all.

Simple build tools (2)

Well known tools are

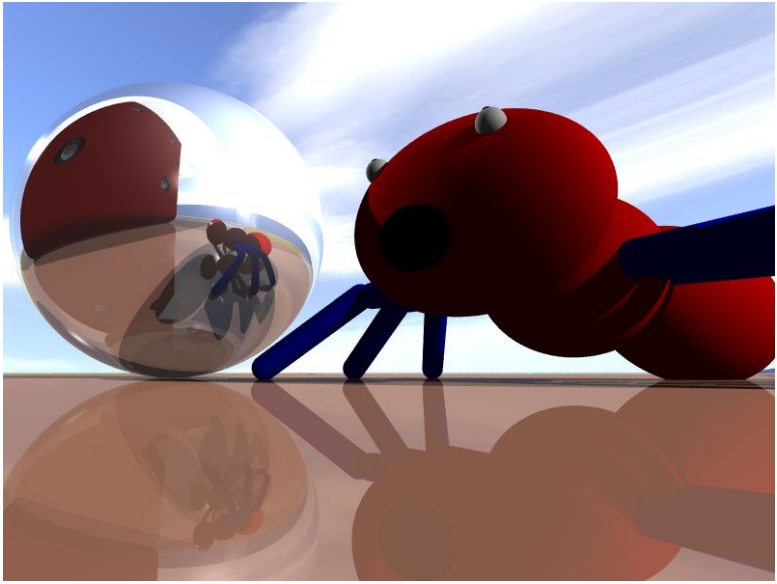
Ant

- Ant is the **make** replacement in the java world. Open (apache.org) source and available on any platform that supports java (and the java XML libs).
- ant uses a programatic style of declaring the build steps, which is a bit strange because xml is not very well suited for this.
- ant is the default build tool for netbeans projects.
- To be found at `http://ant.apache.org`

Maven

- Maven is also an XML based build tool, but does much more, in particular:
  - It can fetch dependencies from so called maven repositories, so you need not download them by hand, and make sure that each developer uses the same version.

The power of an ant



Simple build tools (3)

- Ant defines so called tasks. You can define your own, but there is a great ant community with a lot of predefined tasks. The only problem is finding them (try google) and getting the documentation. (Once a weak spot in open source).
- Ant build scripts are written in XML
- Ant runs on and has access to the java VM.
- Ant extensions can be defined in java.
- Typical ant tasks are build and unit testing. But much more can be done.
- There is a .NET counterpart nowadays. It is called, guess what, Nant.

Ant build script build.xml, excerpts properties

.

Extract from build.xml

```
<property name="appname"
  description="name of the application and jar file"
  value="shoppingcart"
/>
<!-- customize for the fully qualified
     Classname with THE main() method -->
<property name="appmain"
  description="class with the main method to start the application"
  value="shop.Main"
/>

<!-- Location of the system jar files.
   Set for /usr/share/java
   which is the default in
   SuSE linux variants and maybe more. -->
<property name="javalib"
  description="Location of the jar libs."
  location="/usr/share/java"
/>
```

Compile target

```
1  <target name="compile" depends="initbuild"
2    description="Compile the sources without tests ">
3    <javac srcdir="${src}" debug="false" destdir="${build}" />
4  </target>
5
6  <target name="compiletests" depends="compile,initbuild"
7    description="Compiles the sources with tests ">
8    <javac srcdir="test" debug="false" destdir="${testbuild}">
9       <classpath refid="project.classpath"/>
10   </javac>
11 </target>
```

Ant test target

```
<target name="unittest"
   depends="compiletests"
   description="Runs the tests">
  <junit haltonfailure="true">
     <classpath
        refid="project.classpath"/>
     <formatter type="brief"
         usefile="false"/>
     <batchtest>
       <fileset dir="${testbuild}"
          includes="**/*Test.class"/>
     </batchtest>
  </junit>
</target>
```

Ant install file

Yes I know it is a simple jar file. But this is java. Other people call it **xcopy** deploy and think it is fancy.

```
<target name="jar"
  depends="compile"
  description="Compile the sources and makes an executable jar ">
  <mkdir dir="${jar}" />
  <jar jarfile="${jarfile}" basedir="${build}">
     <manifest>
    <attribute name="Main-Class"
          value="${appmain}"/>
     </manifest>
  </jar>
  <echo message="appmain=${appmain}, appname=${appname}"/>
  <echo message="run application with command on next line"/>
  <echo message="java -jar ${jarfile}"/>
</target>
```

There is little ant cannot do

```
<target name="publish"
  description="Bundle the whole stuff and publish to directory"
  depends="jar,javadoc">
  <copy todir="${publish.dir}/api">
    <fileset dir="api/"/>
  </copy>
  <copy todir="${publish.dir}">
    <fileset file="jar/*.jar"/>
  </copy>
  <zip destfile="${publish.dir}/${zip.filename}">
    <fileset dir="../${appname}"
    excludes="**/.svn,${zip.excludes}"/>
  </zip>
</target>
```

Using build properties

- The build.xml file can define placeholders with default values.
- These properties can be override/redefined in a file called `build.properties`.
- This allows the use of a generic build.xml file and reuse it in various projects.
- With `ant -p` or `ant -projecthelp` you get a list of possible ant targets.

Example of build.properties

```
1 # specific for shoppingcart job
2 appmain=shop.Shop
3 appname=shop
4 publish.dir=/home/www/public_html/sen1/shop
5 zip.filename=shop.zip
6 zip.excludes=**/*.class Solution.java
```

What is Maven?

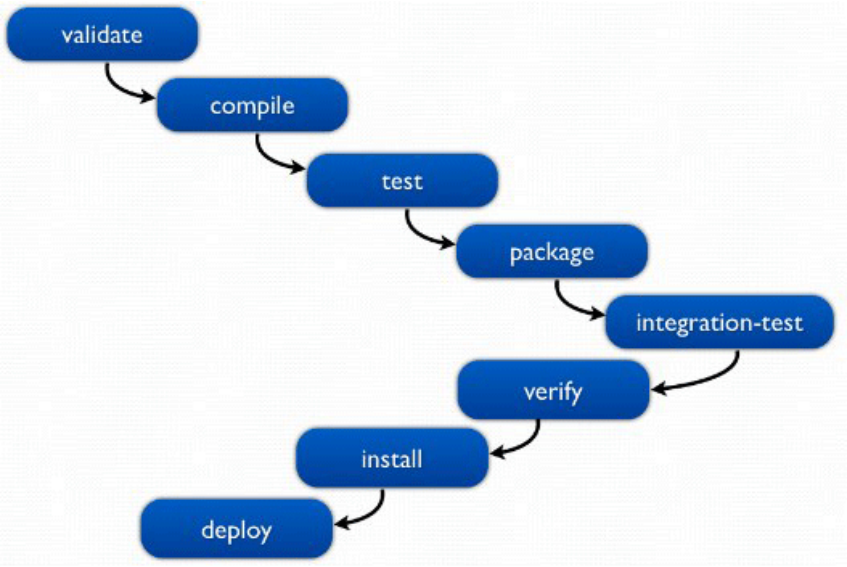
Maven is a software management tool which provides:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Release
- Distribution
- mailing list

Maven Principle

- Convention over configuration
  - standard naming convention
  - standard directory layout
- Declarative execution
  - Maven POM
  - Build Lifecycle
- Reuse of build logic
  - maven splits up the build logic into plugins
- Coherent organization of dependencies

Build Lifecycle



plugin

- A plugin is a piece of software which contains build logic and is bind to one or more build phase
- A plugin contains goals which define a specific task in the build.
- for example Maven provides plugins for:
  - compiling source code
  - a plugin for running tests
  - a plugin for creating JARs
  - a plugin for creating Javadocs

Super POM

- POM = Project Object Model
- contains all configuration for a Maven project
- is required for Maven project
- is a XML file

Example of super POM:
`https://maven.apache.org/guides/introduction/introduction-to-the-pom.html`

POM

- inherits default configuration from the super POM
- minimal configuration:

```
1 <project>
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>com.companyname.project-group</groupId>
4    <artifactId>project</artifactId>
5    <version>1.0</version>
6 </project>
```

Basic Configuration

- `<groupId>`: generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
- `<artifactId>`: name of the project. e.g, consumer-banking
- `<version>`: version of project

Project Inheritance

- All poms inherits standard configuration of the super POM
  - dependencies
  - developers and contributors
  - plugin lists
  - reports lists
  - plugin executions with matching ids
  - plugin configuration
- To inherit from another POM the `parent` tag must be added to the POM

```
<parent>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-parent</artifactId>
    <version>2.0</version>
</parent>
```

Project Aggregation

- A aggregation project lists other project as module
- Maven group the modules and execute them as group

```
1 <modules>
2     <module>my-project</module>
3     <module>another-project</module>
4 </modules>
```

Plugin

- To extend a build phase a plugin can be added in the build tag of the POM

```
<build>
    <plugins>
        <plugin>
            <groupId>...</groupId>
            <artifactId>...</artifactId>
            <version>...</version>
        </plugin>
    </plugins>
</build>
```

Plugin

- When the default configuration is not enough, extra goals and configuration can be added.

```
<configuration>
    .....
</configuration>
```

```
<executions>
    <execution>
        <goals>
            <goal>my-goal</goal>
        </goals>
    </execution>
</executions>
```

Plugin Managment

- Does not configure plugins information for build of the particular project
- Configure plugin information for projects which inherits the POM from the particular project
- Plugin configuration in same way as in plugins section
- Inherit project can use configuration or override them

```
1 <build>
2   <pluginManagement>
3     <plugins>
4         <plugin>
5             ...
6         </plugin>
7     ....
```

Dependencies

Dependencies are libraries or frameworks which are used in the software

How does a Maven depenencies looks

```
<dependency>
    <groupId>group-c</groupId>
    <artifactId>artifact-b</artifactId>
    <version>1.0</version>
</dependency>
```

Repositories

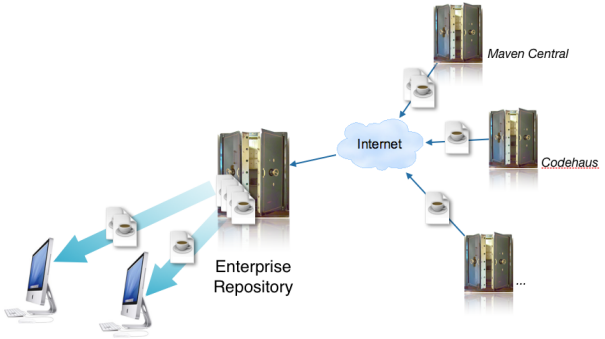


Figure: Maven Repositories example

source: http://download.java.net/general/jn_images/009/nexus-architecture.png

Note that a maven repository is not a subversion repository.

Dependencie Management

- Exactly the same as Plugin Management only for dependencies

```
<dependencyManagement>
    <dependencies>
        <dependency>
        </dependency>
      ....
```

Maven

- Build lifeycle
- POM
- Plugin
- Dependencies

Any questions??