# Advanced build

Ádám Révész, Attila Ulbert, Norbert Pataki, Zoltán Gera


Eötvös Loránd University,
Budapest, Hungary

Introduction to
Cloud Technologies

# What do we need?

Business:

- Fast feedback about correctness, i.e. test results should be available in minutes (e.g. less than 10 minutes), rather than hours or days.
- Fast releasability → Create a consistent, potentially releasable delivery package.

Technical:

- Consistency.
- Compile sources.
- Manage dependencies.
- Run tests.
  - Use tests as quality gateway.
  - Handling unstable tests.
- Create software package (incl. binaries, reports, scripts, docs, etc.)
- Publish.
- Extendibility.
- ...

# Basic concepts

Control Flow of the Build = Executional units + Dependencies

Executional unit:

- Ant: Target
- Maven: Lifecycle & Goal
- Gradle: Task

Dependencies:

- Ant: depends
- Maven: Lifecycle & Goal dependencies
- Gradle: dependsOn

**Imperative** build: the control flow must be defined explicitly.

```
...
<target name="compile" depends="init" description="compile the source">
   <javac srcdir="${src}" destdir="${build}"/>
</target>

 <target name="dist" depends="compile" description="generate the distribution">
   <mkdir dir="${dist}/lib"/>
   <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
 </target>
...
```

**Declarative** build: the control flow is derived, executional units are added automatically.

```
apply plugin: 'java'
```

Build-by-convention: certain conventions must be followed (e.g. where to put the sources, tests, resources).

# Apache Ant

Based on procedural programming.

Easy to learn.

Flexible.

Plugins.

Major drawbacks:

- Programming in XML.
- Can get almost unmanageably big.

```
<project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
      description="compile the source">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
      description="generate the distribution">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
      description="clean up">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

# Apache Maven

XML-based, has declarative elements.

Don’t have to code everything, relies on conventions / targets.

Major improvement: downloads dependencies.

Issues:

- Conflicts of dependencies.
- Customize targets.
- XML can get bloated in bigger projects.

```
<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.app</groupId>
 <artifactId>my-app</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>Maven Quick Start Archetype</name>
 <url>http://maven.apache.org</url>

 <dependencies>
   <dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.8.2</version>
     <scope>test</scope>
   </dependency>
 </dependencies>
</project>
```

# Gradle

Has a Domain Specific Language (DSL), based on Groovy →

- Flexible.
- Dependency management.
- Smaller build code.

Executional unit: task

Plugin: add tasks. Example (Java): `compileJava, processResources, jar, compileTestJava, …`

Packaging: Distribution plugin

```
apply plugin: 'java'
apply plugin: 'eclipse'
sourceCompatibility = 1.7
version = '1.0'
jar {
    manifest {
        attributes 'Implementation-Title': 'Gradle Quickstart',
                   'Implementation-Version': version
    }
}
repositories { mavenCentral() }
dependencies {
    compile group: 'commons-collections', name: 'commons-collections', version:
'3.2.2'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}
test { systemProperties 'property': 'value' }
uploadArchives {
    repositories {
       flatDir {
           dirs 'repos'
       }
    }
}
```

# Advanced issues

- Build to different platforms, free, paid, customized, etc. variants of the product.
  - Ant: many tasks
  - Maven: subproject
  - Gradle: DSL & plugin
- Supporting different types of tests (unit, integration, security, …).
  - Maven: failsafe plugin; subproject
  - Gradle: define new project-level declarative element
- Mandatory (quality gateway) and optional (unstable, long-running, ...) tests.
  - Gradle: define new project-level declarative elements (stable/unstable)
- Handing legacy builds and code.