Projektek felépítése

Automatizált build, függőségmenedzsment és kapcsolódó eszközök (Ant, Maven stb.)

Simon Károly
simon.karoly@codespring.ro

Ant

Ant

- (Leginkább Java) Szoftverprojektek build-elésének automatizálása.
- Another Neat Tool, Apache projekt, nyílt forráskódú.
- A (leginkább C/C++ esetében alkalmazott) Make és hasonló eszközök hátrányainak (pl. saját, speciális fájlformátum, túlérzékenység a szintaktikai helyességre stb.) kiküszöbölését célozza.
- Ant: Java-ban megírt, platformfüggetlen, könnyen kiterjeszthető (Java osztályok segítségével), XML alapú.
- build.xml
- Projects, targets, tasks, properties, dependencies, extension points, filters…
- A legtöbb IDE támogatja.
- Léteznek bizonyos limitációk → nagyobb vállalati rendszerek esetében szükséges lehet más eszközök alkalmazása (pl. Apache Maven).

Ant – HelloWorld

- ```
  md src
  ```
- ```
  package hello;
  public class HelloWorld {
        public static void main(String[] args) {
              System.out.println("Hello World");
        }
  }
  ```
- ```
  md build\classes
  javac -sourcepath src -d build\classes src\hello\HelloWorld.java
  java -cp build\classes hello.HelloWorld
  ```
- ```
  echo Main-Class: hello.HelloWorld>myManifest
  md build\jar
  jar cfm build\jar\HelloWorld.jar myManifest -C build\classes
  java -jar build\jar\HelloWorld.jar
  ```

Ant – build.xml

- ```
  <project>
        <target name="clean">
              <delete dir="build"/>
        </target>
        <target name="compile">
              <mkdir dir="build/classes"/>
              <javac srcdir="src" destdir="build/classes"/>
        </target>
        <target name="jar">
              <mkdir dir="build/jar"/>
              <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
                    <manifest>
                          <attribute name="Main-Class"
                                      value="hello.HelloWorld"/>
                    </manifest>
              </jar>
        </target>
        <target name="run">
              <java jar="build/jar/HelloWorld.jar" fork="true"/>
        </target>
  </project>
  ```
- ```
  ant compile
  ant jar
  ant run
  ```
- ```
  ant compile jar run
  ```
  ```

Ant – build.xml

- project: name, default (target), basedir
  (ha nem adjuk meg az xml-t tartalmazó könyvtár)
- target (compile, jar, run stb.): name, depends (függőségek megadásának lehetősége)
- task: `<name attribute1="value1" attribute2="value2" ... />`
- properties: property task, hivatkozás ${...} (pl. ${builddir}/classes)
- token filters: állományok másolásakor @token@ (a token a szűrőben megadott név) alakú elemek helyettesítése
- path, classpath elemek, fileset, dirset (+include, exclude), references:

```
<classpath>
        <pathelement path="${classpath}"/>
        <fileset dir="lib"> <include name="**/*.jar"/> </fileset>
        <pathelement location="classes"/>
        <dirset dir="${build.dir}">
                <include name="apps/**/classes"/>
                <exclude name="apps/**/*Test*"/>
        </dirset>
        <filelist refid="third-party_jars"/>
</classpath>
```

- Parancssor argumentumok: value, line, file, path stb.
  (pl. `<arg value="-l -a"/>`, `<arg line="-l -a"/>`)

Ant – tasks

- Archive tasks: Manifest, Jar, Unjar, Zip, Unzip, War, Unwar, Ear stb.
  - Jar attributes: destfile, basedir, includes, excludes, manifest stb.
- Compile tasks: Javac, Rmic stb.
  - Javac attributes: srcdir, destdir, classpath, includes, excludes stb., stb.
- Execution tasks: Java, Exec (system command), Ant stb.
  - Java attributes: jar/classname, classpath, fork (class execution in another VM) stb.
    + arg és jvmarg beágyazott elemek
- File tasks: Mkdir, Copy, Delete, Move, Get (from URL), Attrib, Filter stb.
- Property tasks: Property, LoadFile stb.
- Testing tasks: Junit, JunitReport
- Deployment tasks: ServerDeploy
- Documentation tasks: Javadoc
- Logging tasks: Record
- EJB tasks
- Remote tasks: FTP, Scp, Sshexec stb.
- Mail tasks: Mail
- Extensions tasks: Jarlib-available, Jarlib-resolve stb.
- Pre-process tasks: Import, Include, Javah stb.
- CVS tasks: Cvs, ClearCase stb.
- Audit tasks: Jdepend
- Other tasks: Echo, Sql, Splash, Sound stb., stb.

Properties, dependencies

- ```
  <project name="HelloWorld" basedir="." default="main">
          <property name="src.dir" value="src"/>
          <property name="build.dir" value="build"/>
          <property name="classes.dir" value="${build.dir}/classes"/>
          <property name="jar.dir" value="${build.dir}/jar"/>
          <property name="main-class" value="hello.HelloWorld"/>

          <target name="clean">
                      <delete dir="${build.dir}"/>
          </target>
          <target name="compile">
                      <mkdir dir="${classes.dir}"/>
                      <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
              </target>
              <target name="jar" depends="compile">
                      <mkdir dir="${jar.dir}"/>
                      <jar destfile="${jar.dir}/${ant.project.name}.jar"
                                          basedir="${classes.dir}">
                                  <manifest> <attribute name="Main-Class" value="${main-class}"/>
                                  </manifest>
                      </jar>
              </target>
              <target name="run" depends="jar">
                      <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
              </target>
              <target name="clean-build" depends="clean,jar"/>
              <target name="main" depends="clean,run"/> </project>
  ```
- ant
  ```

Ant – using external libraries

- ```
  package hello;
  import org.apache.log4j.Logger;
  import org.apache.log4j.BasicConfigurator;
  public class HelloWorld {
          static Logger logger = Logger.getLogger(HelloWorld.class);
          public static void main(String[] args) {
                  BasicConfigurator.configure();
                  logger.info("Hello World");
          }
  }
  ```
- ```
  <project name="HelloWorld" basedir="." default="main">
  ...
          <property name="lib.dir" value="lib"/>
          <path id="classpath">
                  <fileset dir="${lib.dir}" includes="**/*.jar"/>
          </path>
          ...
          <target name="compile">
                  <mkdir dir="${classes.dir}"/>
                  <javac srcdir="${src.dir}" destdir="${classes.dir}"
                                                  classpathref="classpath"/>
          </target>
          <target name="run" depends="jar">
                  <java fork="true" classname="${main-class}">
                          <classpath>
                                  <path refid="classpath"/>
                                  <path location="${jar.dir}/${ant.project.name}.jar"/>
                          </classpath>
                  </java>
          </target>
  ...
  </project>
  ```

Ant – resources

- Configuration file:
- ```
  log4j.rootLogger=DEBUG, stdout
  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
  log4j.appender.stdout.layout.ConversionPattern=%m%n
  ```
- ```
  ...
  <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"
                                                classpathref="classpath"/>
        <copy todir="${classes.dir}">
                  <fileset dir="${src.dir}" excludes="**/*.java"/>
        </copy>
  </target>
  ...
  ```

Ant – testing (JUnit)

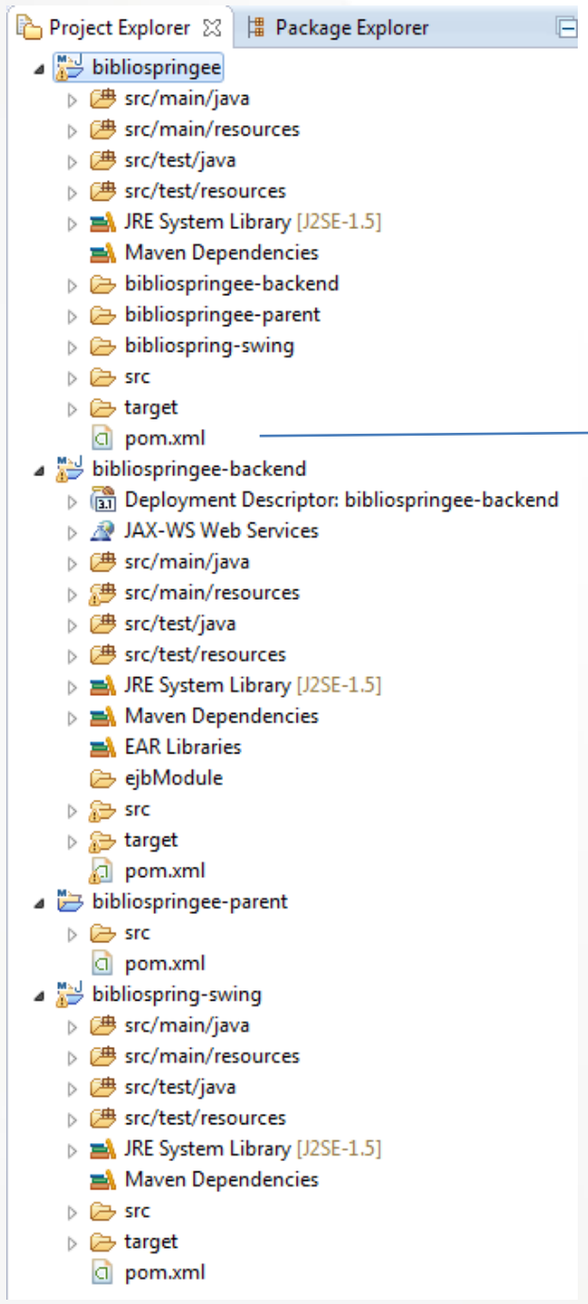
- ```
  public class HelloWorldTest extends junit.framework.TestCase {
        public void testNothing() { }
        public void testWillAlwaysFail() {
                fail("An error message");
        }
  }
  ```
- ```
  ...
  <property name="report.dir" value="${build.dir}/junitreport"/>
  ...
  <target name="junit" depends="jar">
        <mkdir dir="${report.dir}"/>
        <junit printsummary="yes">
                <classpath>
                        <path refid="classpath"/> <path refid="application"/>
                </classpath>
                <formatter type="xml"/>
                <batchtest fork="yes" todir="${report.dir}">
                        <fileset dir="${src.dir}" includes="*Test.java"/>
                </batchtest>
        </junit>
  </target>
  <target name="junitreport">
        <junitreport todir="${report.dir}">
                <fileset dir="${report.dir}" includes="TEST-*.xml"/>
                <report todir="${report.dir}"/>
        </junitreport>
  </target>
  ```

Maven

Maven

- Apache Software Foundation
- Automatizált build, függőség- és projektmenedzsment eszköz + projektek felépítésére vonatkozó minták/receptek.
- Konvenciók szerepe…
- POM – Project Object Model: xml állomány, amelyben meg vannak határozva a fordítási lépések, a termék típusa, verziószám, függőségek, fordításhoz szükséges plug-in-ok, a projekt szerkezete.
- Több modulból álló projektnél modulonként egy-egy POM állomány szükséges.
- Központi POM + almodulokra specifikus tulajdonságokat leíró POM állományok.
- POM-ok közötti "öröklődési" viszony.
- Alkalmazható recept: "parent" modul, amely a közös tulajdonságokat/ függőségeket összesítő POM-ot tartalmazza.

Maven

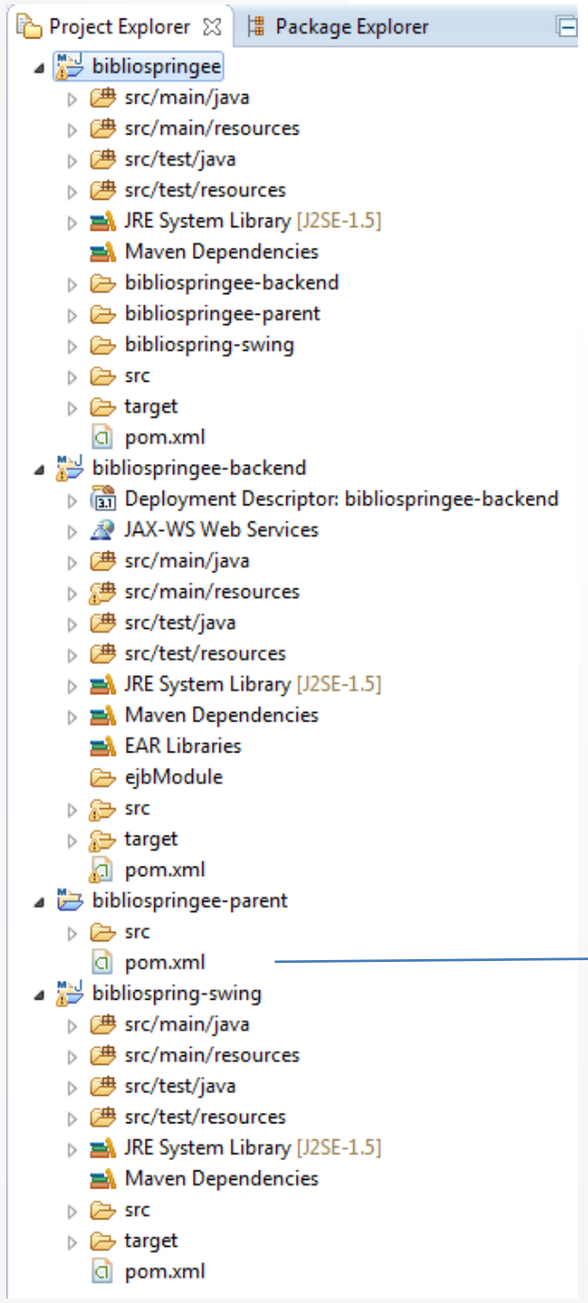



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>edu.codespring.bibliospringee</groupId>
    <artifactId>bibliospringee-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>bibliospringee-parent/</relativePath>
  </parent>

  <artifactId>bibliospringee</artifactId>
  <name>BiblioSpring</name>
  <packaging>pom</packaging>
  <modules>
    <module>bibliospringee-parent</module>
    <module>bibliospringee-backend</module>
    <module>bibliospring-swing</module>
  </modules>
</project>
```

Maven - POM

- **<project>** - gyökérelem, a megfelelő XML névterekkel
- **<modelVersion>** - támogatott POM verzió, jelenleg 4.0.0
- **<groupId>** - tulajdonképpen csomagnév, javasolt a Java névkonvenció alkalmazása
- **<artifactId>** - a projekt/almodul neve, javasolt a szülő nevét is belevenni (pl. bibliospringee-swing)
- **<version>** - a modul verziószáma (javasolt a x.x.x-SNAPSHOT, vagy utótag nélkül, ha végleges build/release)
- Teljes artifact név: groupId:artifactId:version
- **<packaging>** - célállomány típusa (pom, jar, maven-plugin, ejb, war, ear stb.), alapértelmezetten jar.
- **<modules>** - burkoló modulok esetében, meg lehet adni az almodulokat (<module> elemeken belül, az artifactId-k segítségével).
- További elemek: name, description (opcionálisak) stb.

Maven - függőségek



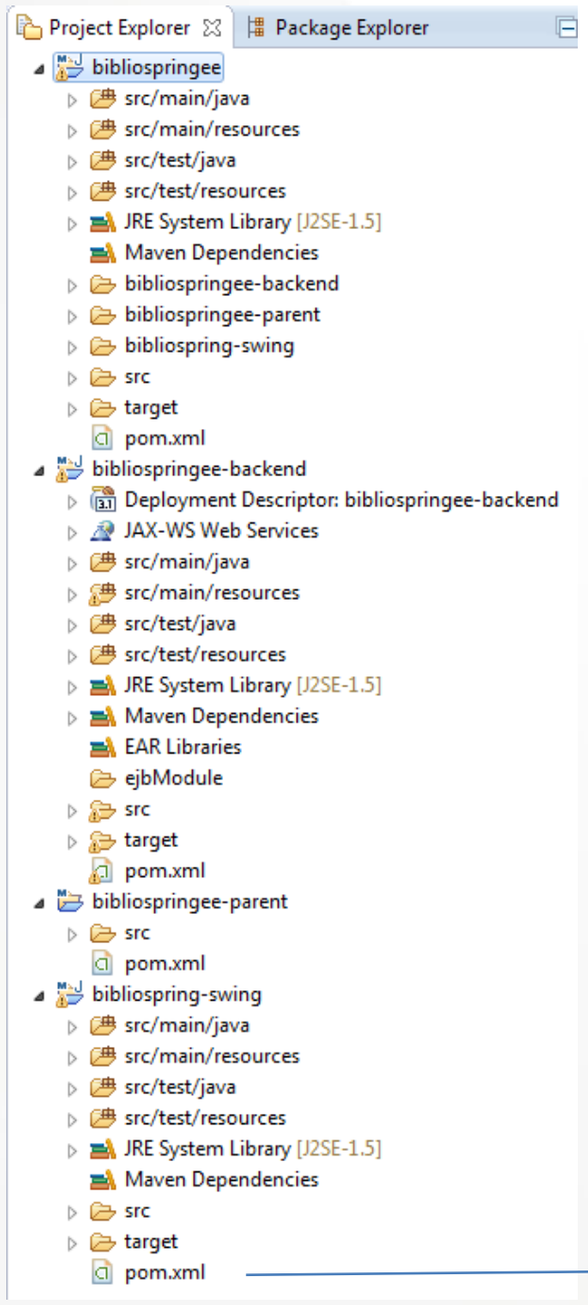

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.codespring.bibliospringee</groupId>
  <artifactId>bibliospringee-parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <dependencyManagement>
      <dependencies>
         <!-- EJB & JPA -->
         <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>6.0</version>
            <scope>provided</scope>
         </dependency>
            ...
      </dependencies>
      ...
  </dependencyManagement>

</project>
```

Maven - függőségek

- **<dependency>** elem a <dependencies> tag-en belül: groupId, artifactId, version belső elemek
- Öröklődhetnek az almodulokban – a dependencies elemet dependecyManagement elemmel kell burkolni, amennyiben jelezni szertnénk, hogy az adott POM-ban csak menedzselve vannak a függőségek, az almodulok fogják ezeket használni.
- A függőségeknek megfelelő csomagokat a Maven központi repository-ból tölti le első használatkor, majd lokális tárolóba helyezi.
- Amennyiben a csomag nem található meg a központi repository-ban, meg kell határozni a repository elérési útvonalát:

```
<repositories>
      <repository>
            <id>prime-repo</id>
            <name>PrimeFaces Maven Repository</name>
            <url>http://repository.primefaces.org</url>
      </repository>
</repositories>
```

Maven – plugin-ok használata




```
<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>

  <parent>
    <groupId>edu.codespring.bibliospringee</groupId>
    <artifactId>bibliospringee-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>../bibliospringee-parent/</relativePath>
  </parent>

  <artifactId>bibliospringee-swing</artifactId>

  <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <configuration>
                   <mainClass>edu.codespring.bibliospringee.swing.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>
 </build>

</project>
```

Maven - életciklus

Alapértelmezett életciklus:

- **validate** – leellenőrzi, hogy a projekt szerkezete helyes-e, a leíró állományok rendben vannak-e illetve, hogy megvan-e az összes szükséges függőség
- **compile** – a forráskódok fordítása
- **test** – unit tesztek futtatása a lefordított forráskódon
- **package** – a forráskódok becsomagolása (JAR, WAR, EAR)
- **integration-test** – a csomag telepítése egy környezetbe, ahol az általunk megadott integrációs tesztek lefuthatnak
- **verify** – bármilyen általunk megadott ellenőrzés futtatása a csomagon
- **install** – az elkészült csomag bemásolása a lokális repository-ba, ezáltal használható lesz függőségként egy másik projektben, persze csak lokálisan
- **deploy** – a csomag bemásolása egy távoli, általában céges repository-ba, hogy más fejlesztők is elérhessék

Maven - életciklus

- Az életciklus fázisai szekvenciálisan követik egymást, az utolsót kell megadnunk. Pl. mvn deploy esetén minden előző fázis végrehajtásának sikeresnek kell lennie.
- Persze, kimaradhatnak fázisok – pl. ha nincsenek tesztek vagy nincsenek beállítások az integration-test és verify fázisokhoz stb.
- Továbbá: clean fázis – törli az előző build folyamat alatt generált állományokat (pl. mvn clean install)
- Egy-egy fázis goal-okból (task) áll össze (0 vagy több), ezek különbözőek (pl. package – ejb:ejb v. ejb3:ejb3 v. jar:jar v. war:war stb.), "packaging–függően", különállóan is futtathatóak (pl. mvn clean dependency:copy-dependencies package – a clean fázis után, a package és előző fázisok előtt végrehajtja a dependency:copy-dependecies goal-t)
- További fázisok: pre-clean, post-clean (clean), initialize, generate-sources, process-sources, generate-resources, process-resources (default – compile előtt), process-classes, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-test-classes (default – test előtt), prepare-package (default – package előtt), pre-integration-test, post-integration-test (default – integration-test előtt/után), pre-site, site, post-site, site-deploy (site).

Maven - settings

- settings.xml: Maven konfiguráció
  - Szerverek, felhasználónevek, jelszavak, egyéb beállítások → nem képezi részét a projektnek, nem nyilvános, a felhasználók információk, nincs hozzáférés a végfelhasználók számára.
- Globális/felhasználó-specifikus tulajdonságok:
  - $M2_HOME/conf/settings.xml
  - ${user.home}/.m2/settings.xml
  - Ha mindkettő jelen van, egybe lesznek olvasztva, a felhasználó-specifikus beállítások fognak dominálni.
- <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
        http://maven.apache.org/xsd/settings-1.0.0.xsd">

        <localRepository/>
        <interactiveMode/>
        <usePluginRegistry/>
        <offline/>
        <pluginGroups/>
        <servers/>
        <mirrors/>
        <proxies/>
        <profiles/>
        <activeProfiles/>
</settings>
```

Maven - settings

- Local repository: default ${user.home}/.m2/repository
- interactiveMode: felhasználói input bekérésének lehetősége (default: true)
- offline: ha offline módban kell működnie (default: false)
- pluginGroups: plug-in csoportok (pluginGroup elemek) listája, amelyben keresni fog, ha a használt plug-in groupId nincs megadva a parancssorból. Alapból tartalmazza az org.apache.maven.plugins és org.codehaus.mojo csopotokat.
- servers: a repositories és distributionManagement elemek a POM-on belül meghatározzák a tárolókat, de bizonyos beállításokat a settings-ben kell megadni (felhasználónevek, jelszavak stb.), hogy ne képezzék részét a POM-nak.
- ...

```
<servers>
        <server>
                <id>server001</id>
                <username>my_login</username>
                <password>my_password</password>
                <privateKey>${user.home}/.ssh/id_dsa</privateKey>
                <passphrase>some_passphrase</passphrase>
                <filePermissions>664</filePermissions>
                <directoryPermissions>775</directoryPermissions>
                <configuration></configuration>
        </server>
</servers>
...
```

Maven - settings

- Mirrors: mirror elemeken belül meghatározható, ha valamelyik repository-hoz mirror-t szeretnénk rendelni (pl. gyorsabb hozzáférés földrajzilag közelebb található szerverekhez, saját repository management rendszer használata).
  - A tükrözött tároló azonosítóját egy mirrorOf elem segítségével adhatjuk meg (a központi azonosítója central).
  - Egy tárolóra csak egy mirror-t állíthatunk be (az első beállítás lesz érvényes), ha több tárolót egységesen akarunk elérni repository management rendszer használata szükséges.
  - Beállítások a mirrorOf elemen belül: * - minden; external:* - minden, ami nincs meg lokálisan; repo1, repo2 – egyik vagy másik; *, !repo1 – minden a repo1-en kívül
  - ...

```
<mirrors>
    <mirror>
      <id>internal-repository</id>
      <name>Maven Repository Manager running on repo.mycompany.com</name>
      <url>http://repo.mycompany.com/proxy</url>
      <mirrorOf>external:*,!foo</mirrorOf>
    </mirror>
...
</mirrors>
...
```

- Proxy-k beállítása: proxies elemen belüli proxy elemekkel (id, protocol, host, port, username, password megadása + active tulajdonság, ha pl. többet akarunk megadni, de csak egy aktív + nonProxyHosts lista)

Maven - settings

- Profiles: profile elem – a pom.xml profile elemének részleges változata, belső elemei: activation, repositories, pluginRepositories, properties.
  - Ha egy profile aktív a settings.xml-en belül, felülírja a POM vagy profile.xml-ben megadottakat.
  - Activation: a profile bizonyos körülmények között változtat bizonyos értékeken, az activation ezeket a körülményeket határozza meg. Belső elemei: jdk, os, property, file.
  - A settings.xml állomány activeProfile elemével is aktiválható.
  - …

```
<profiles>
    <profile>
        <id>test</id>
        <activation>
            <activeByDefault>false</activeByDefault>
            <jdk>1.5</jdk>
            <os>
                <name>Windows XP</name>
                <family>Windows</family>
                <arch>x86</arch>
                <version>5.1.2600</version>
            </os>
            <property>
                <name>mavenVersion</name>
                <value>2.0.3</value>
            </property>
            <file>
                <exists>${basedir}/file2.properties</exists>
                <missing>${basedir}/file1.properties</missing>
            </file>
        </activation>
        ...
    </profile>
</profiles>
...
```

Maven - properties

- Properties: értékek elérése, ${prop} jelöléssel, bármelyik pom-on belül.
  - env.x – a rendszerváltozók elérése (pl. ${env.PATH})
  - project.x – projekt tulajdonságainak elérése (POM elemeinek értéke), pl. ${project.version}
  - settings.x – a settings.xml elemeinek értéke (pl. ${settings.offline})
  - java.x – Java rendszertulajdonságok (pl. ${java.home})
  - x - <properties> elemen vagy külső állományon belül megadott elem értéke (pl. ${someVar}
  - ...

```
<profiles>
   <profile>
                ...
                <properties>
                        <user.install>${user.home}/our-project</user.install>
                </properties>
                ...
   </profile>
</profiles>
...
```

- A ${user.install} tulajdonság elérhető a POM-on belül, ha a profile aktív

Maven - repositories

- Távoli projekt tárolók, a lokális repository ezekből lesz feltöltve, így lesznek kielégítve a függőségek és biztosítva a plugin-ok. Az aktív profile-nak megfelelően ezekben lesznek keresve a release/snapshot artifact-ek.
-

```
<repositories>
        <repository>
                <id>codehausSnapshots</id>
                <name>Codehaus Snapshots</name>
                <releases>
                        <enabled>false</enabled>
                        <updatePolicy>always</updatePolicy>
                        <checksumPolicy>warn</checksumPolicy>
                </releases>
                <snapshots>
                        <enabled>true</enabled>
                        <updatePolicy>never</updatePolicy>
                        <checksumPolicy>fail</checksumPolicy>
                </snapshots>
                <url>http://snapshots.maven.codehaus.org/maven2</url>
                <layout>default</layout>
        </repository>
</repositories>
<pluginRepositories>
        ...
</pluginRepositories>
...
```

Maven - exclusions

- Opcionális függőségek megadásának lehetősége: ha A és B között opcionálisnak deklaráljuk a függőséget, egy A-tól függő C esetében B nem lesz hozzákapcsolva C-hez (csak ha ezt a C POM-ban explicit módon kérjük).
- Exclusions: pl. az A projekt függ B-től, amely függ C-től. Alapértelmezetten ebben az esetben C is bekerül A classpath-jébe, még akkor is, ha A-nak nincs szüksége erre (az A nem függ C-től). Ilyen esetekben használhatjuk az exclusion elemet.
  - Pl. használunk egy keretrendszert, amely log4j-t használ naplózásra, de mi egy más naplózási keretrendszert használunk, így nincs szükségünk a log4j-re.
  - …

```
<dependency>
    <groupId>sample.ProjectA</groupId>
    <artifactId>Project-A</artifactId>
    <version>1.0</version>
    <scope>compile</scope>
    <exclusions>
                    <exclusion>
                                    <groupId>sample.ProjectB</groupId>
                                    <artifactId>Project-B</artifactId>
                    </exclusion>
    </exclusions>
</dependency>
```

Maven – help, release

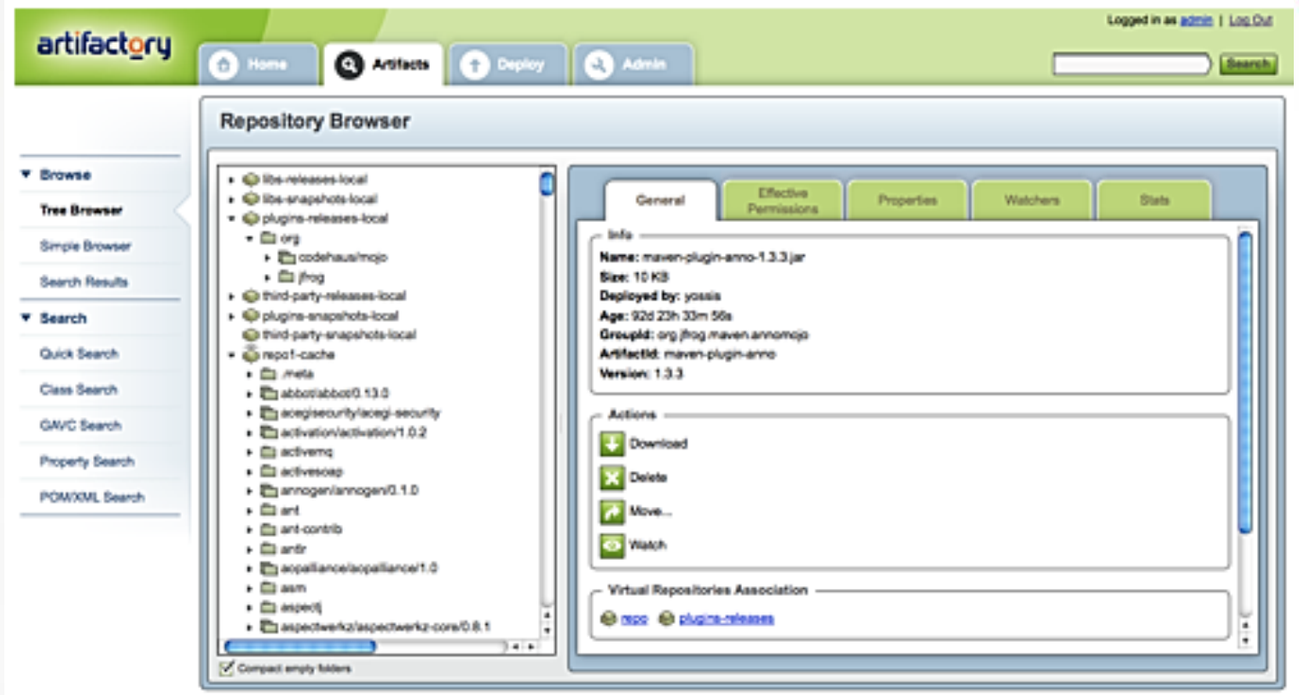
- Help plugin:
  - help:active-profiles
  - help:all-profiles
  - help:describe –DgroupId=… -DartifactId= …
  - **help:effective-pom**
  - help:effective-settings
  - help:system
  - help:expressions
  - help:evaluate
- Release plugin:
  - clean, prepare, prepare-with-pom, rollback, perform, stage, branch, update-versions
  - Pl. prepare: ellenőrizni, hogy vannak-e nem commit-olt változtatások, SNAPSHOT függőségek, SNAPSHOT verziószámok átírása, tesztek lefuttatása a módosítások után, módosított POM-ok felküldése, verziókövetőn belüli tag megadása.
    - Ha újraindítás szükséges, az előző ponttól folytatja (egyébként resume kapcsolót átállítani, vagy clean-t adni).

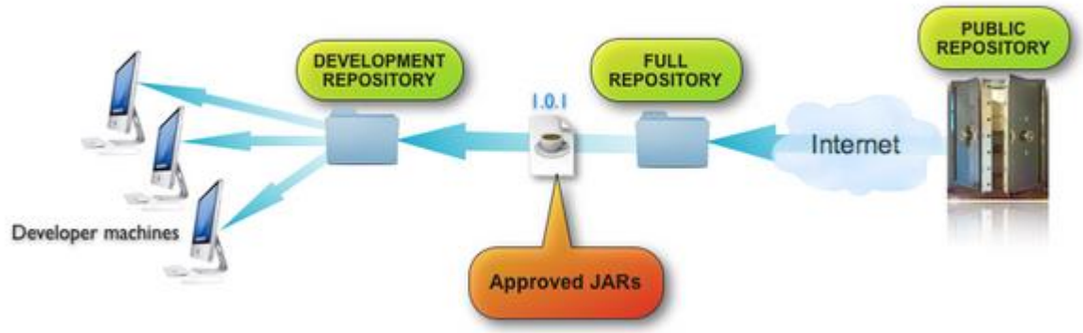
Repository Management Tools

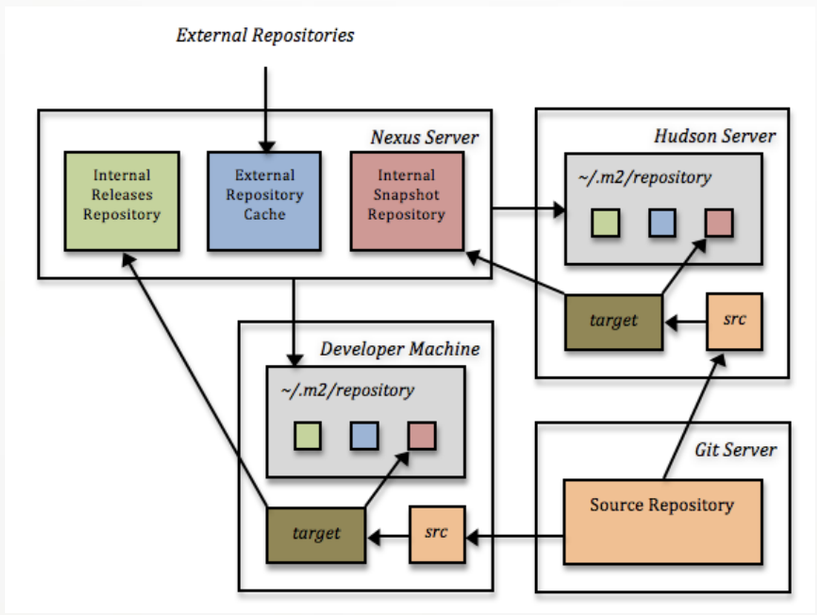
Repository Management

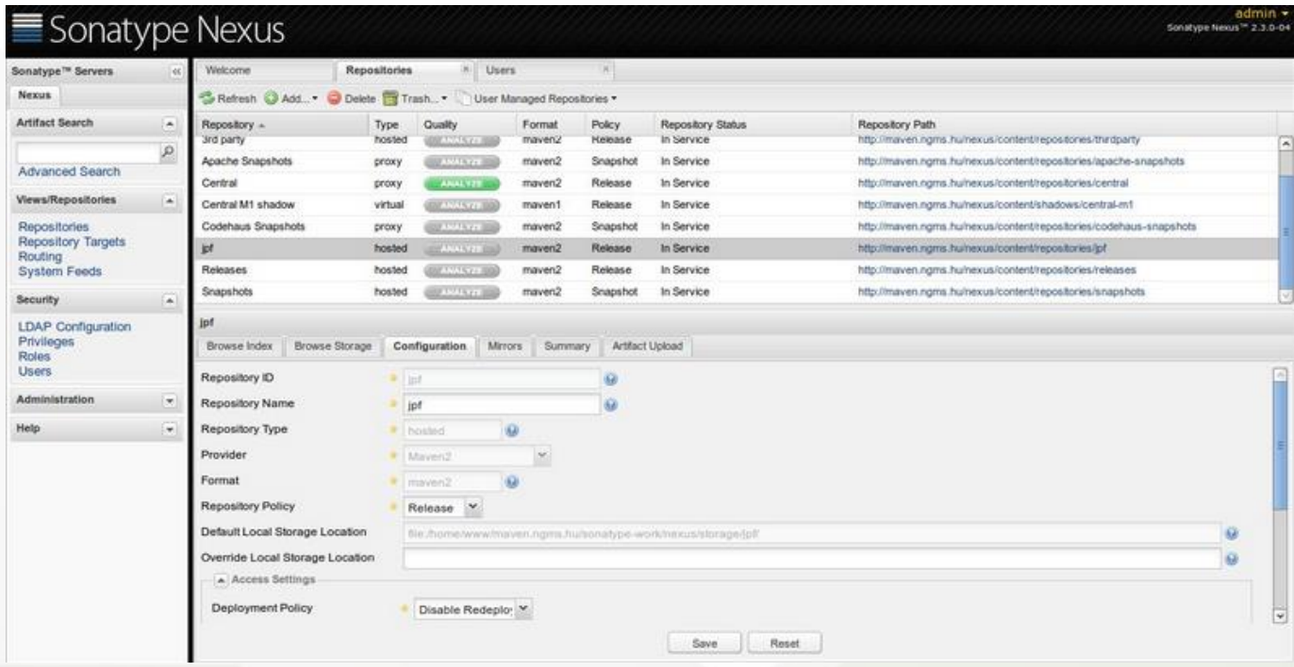
- Távoli repository-k elérésének "kiváltása", helyi proxy-k használata.
  - Gyors belső hálózat használatának előnyei.
  - Konfigurálhatóság (pl. cégen belül használt technológiák standardizálása, licenszekkel/szerzői jogokkal kapcsolatos megkötések bevezetése stb.).
  - Modulok közzététele cégen belül, belső (pl. projektek közti) használatra, SNAPSHOT verziók megosztása.
- Közismertebb rendszerek:
  - Apache Archiva
  - **Artifactory**
  - Sonatype Nexus
    - A Cservenák Tamás által kifejlesztett Proximity utóda

Repository Management









Continuous Integration Tools

Continuous Integration

- Az Extreme Programming és TDD stratégiákon belül bevezetett, jelenleg széles körben használt módszer (egyik fő támogató Martin Fowler).
- Több fejlesztő, több változtatás → az integráció nehézzé válhat ("integration hell").
- Megoldás: gyakori commit-ok, projekt automatizált build-elése és tesztek automatizált futtatása minden commit után/adott időközönként, minden fejlesztő számára látható report-ok generálása.
- Receptek:
  - Atomi commit műveleteket támogató verziókövető használata.
  - Build automatizálása, build szerver használata.
  - Gyakori commit-ok (mindenki, min. naponta).
  - Minden commit után build (sok fejlesztő esetén időzítve, rövid időközönként + pl. éjszakai build-ek).
  - Automatizált QA műveletek (nem feltétlenül csak unit test), mindenki számára nyilvános report-ok.
  - Automatizált deployment (teszt szerverekre).
  - Stb.

Eszközök

- **Jenkins**/Hudson stb.