AVTOMATIZACIJA GRADNJE APLIKACIJ

Gradnja aplikacij

- Pomemben del zvezne integracije
- Pripomore k učinkovitejši zadnji fazi integracijskega cikla
- Poskrbi za ustrezno pripravo in namestitev izdelka v testno okolje
- Glavna tri (priporočena) orodja za avtomatizacijo grajenje so Maven, Jenkins in Nexus

Orodja za avtomatizacijo buildov

- Prevajanje programske kode
- Upravljanje z odvisnostmi
- Pakiranje programske kode
- Izvajanje testov
- Namestitev paketov
- Priprava dokumentacije
- Najpogosteje uporabljeni:
  - Apache Maven
  - Gradle
  - Apache Ant
  - Make
  - Pants
  - SBT

APACHE MAVEN

Uvod v Maven

- Orodje na katerem temelji avtomatizirano grajenje (build) projektov

- Prednosti uporabe Maven
  - Izboljšana vidnost in transparentnost razvojnega procesa
  - Apliciranje splošno sprejetih dobrih praks (npr. verzioniranje)
  - Standardizacija (npr. enotna struktura projektov)
  - Izboljšano upravljanje z odvisnostmi (binarne odvisnosti)
  - Lažja izmenjava kreiranih artefaktov (JAR, WAR, EAR, RAR)
  - Ponovna uporaba
  - Samodejno generiranje spletne strani in dokumentacije
  - Zmanjšan čas vpeljave novih razvijalcev

Uvod v Maven

- Srce vsakega Maven projekta je datoteka pom.xml
- POM (Project Object Model)
- pom.xml definira:
  - Naziv projekta
  - Verzijo
  - Odvisnosti
  - Cilje (goals)
  - Vtičnike (plugins)
  - Razne metapodatke

- V pom.xml so obvezne samo 4 vrstice (groupId, artifactId, version in modelVersion)
- Pri definiranju pom datotek lahko uporabimo koncept dedovanja
- Vsak pom.xml deduje od t.i. super POM-a

Uvod v Maven

- Primer preproste pom.xml datoteke:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.demo</groupId>
      <artifactId>projektA</artifactId>
      <packaging>war</packaging>
      <version>0.0.1-SNAPSHOT</version>

      <dependencies>
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit </artifactId>
                  <version>4.8.1</version>
                  <scope>test</scope>
            </dependency>
      </dependencies>
</project>
```

Privzeta vrednost je jar, v tem primeru lahko izpustimo. Druge opcije: war, ejb, rar, ear, pom in drugi (custom).

Verzija

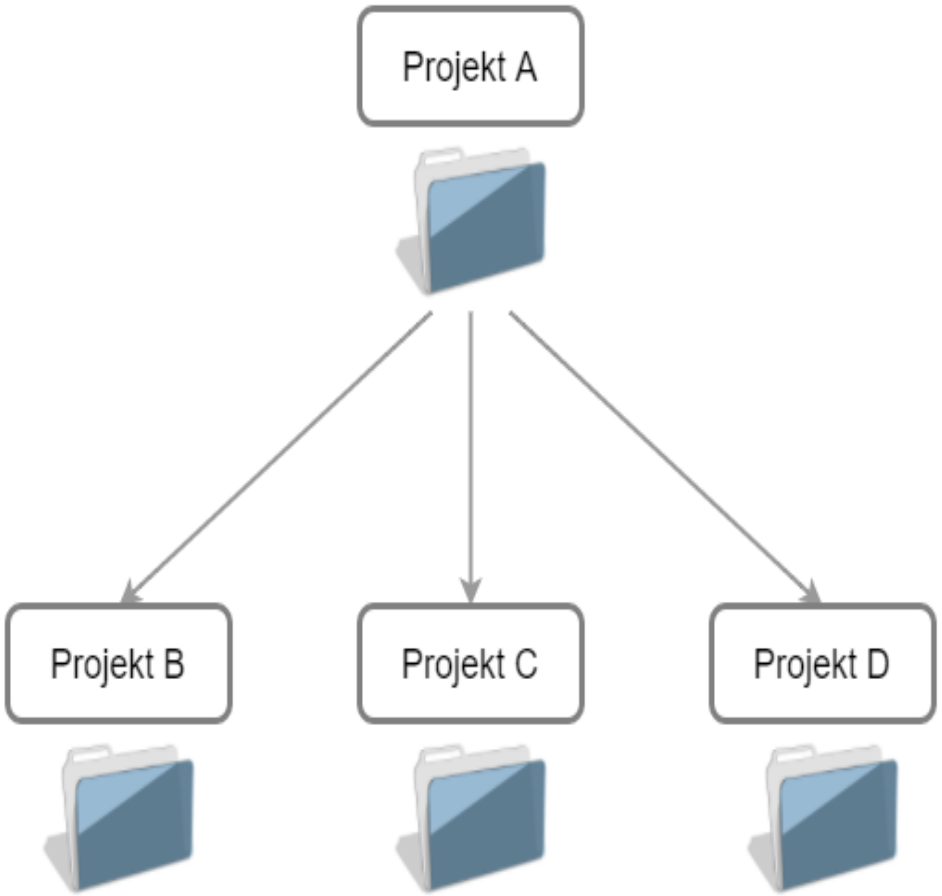
Seznam odvisnosti

Naziv artefakta, ki se kreira ob build-u:

projektA-0.0.1-SNAPSHOT.war

Uvod v Maven

- Dedovanje datotek pom.xml:



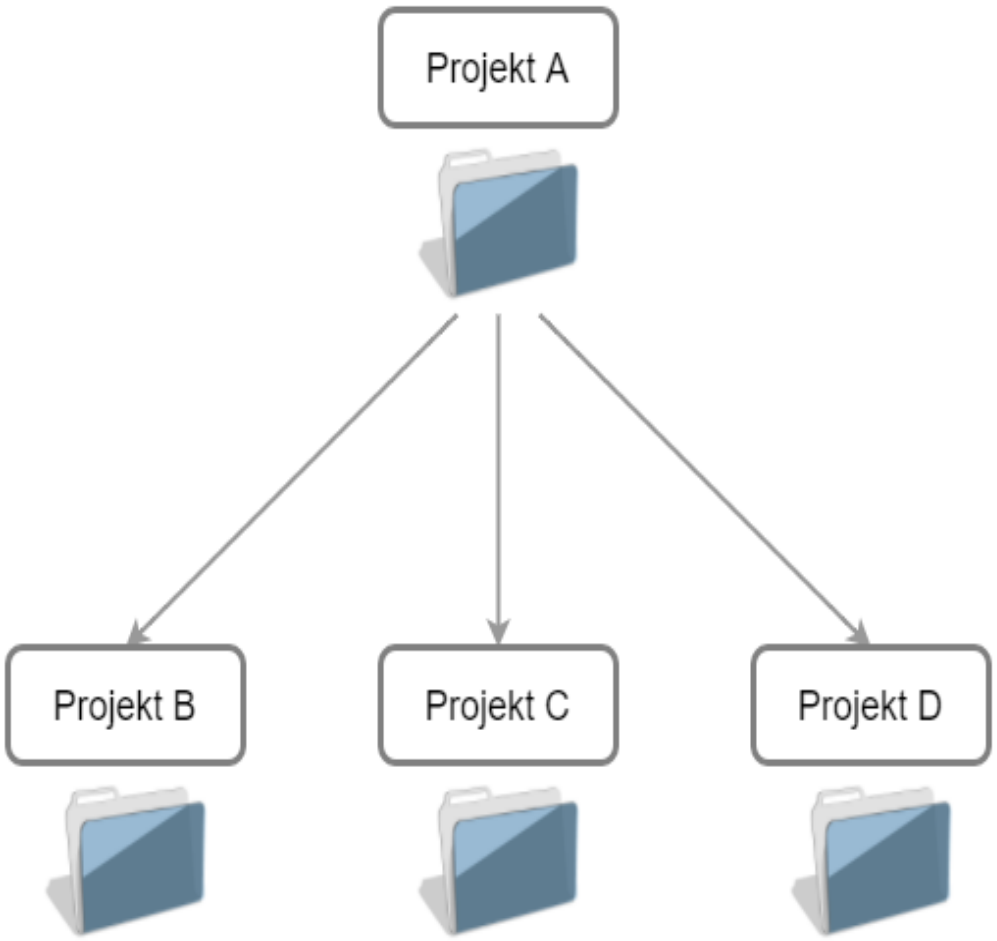

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.demo</groupId>
    <artifactId>ProjektA</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
</project>
```

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>projektA</artifactId>
        <groupId>com.demo</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>projektB</artifactId>
</project>
```

Projekti B, C in D podedujejo verzijo, groupId, način pakiranja (war) ter vse odvisnosti in konfiguracijo vtičnikov.

Uvod v Maven

- Agregacija Maven projektov (več-modulov):




```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.demo</groupId>
    <artifactId>ProjektA</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>
    <modules>
        <module>projektB</module>
        <module>projektC</module>
        <module>projektD</module>
    </modules>
</project>
```

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.demo</groupId>
    <artifactId>projektB</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</project>
```

Vsi ukazi nad A se izvedejo tudi nad B, C in D.

Uvod v Maven

- Maven definira tri življenjske cikle:
  - Default (namenjen korakom buildanja in nameščanja)
  - Clean:
    - pre-clean
    - clean
    - post-clean
  - Site (oblikovanje dokumentacije):
    - pre-site
    - site
    - post-site
    - site-deploy

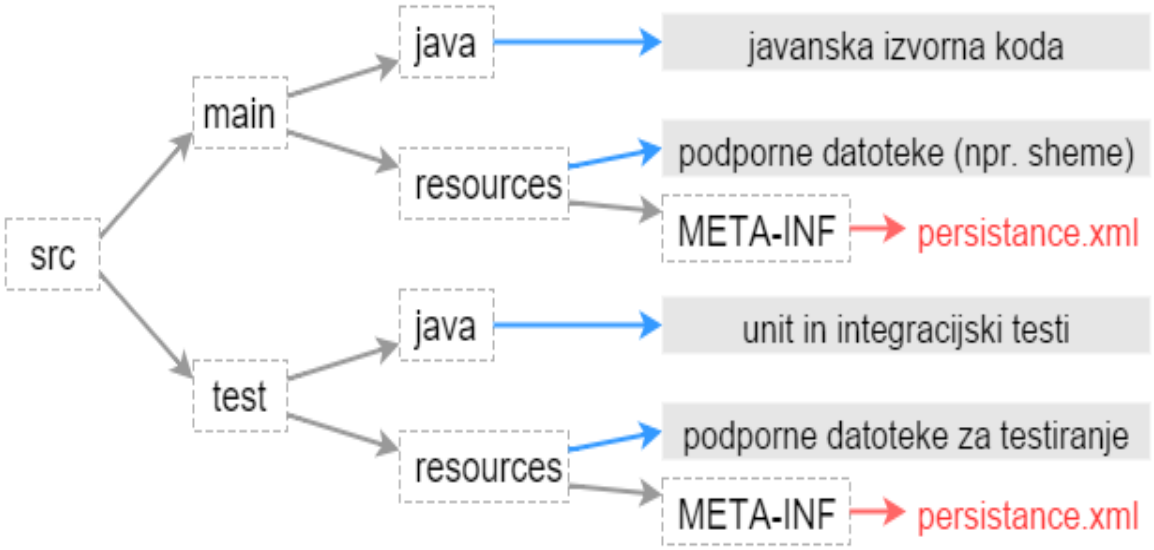
- Ob izvedbi določene faze se samodejno izvedejo tudi vse predhodne faze

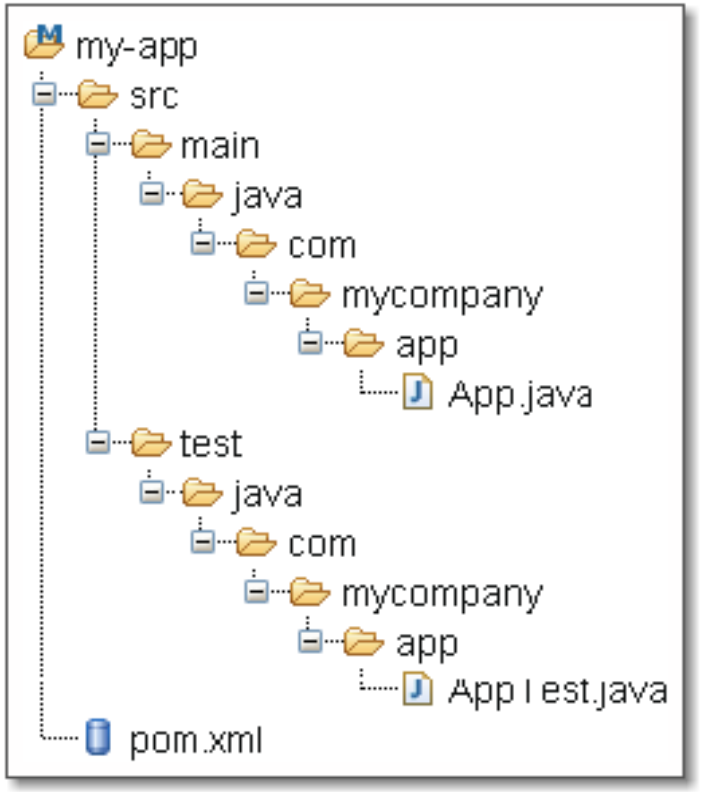
Uvod v Maven

- Faze Maven build življenjskega cikla (default):
  - **validate**
  - **initialize**
  - generate-sources
  - process-sources
  - generate-resources
  - process-resources
  - **compile**
  - process-classes
  - generate-test-sources
  - process-test-sources
  - generate-test-resources
  - process-test-resources
  - test-compile
  - process-test-classes
  - test
  - prepare-package
  - **package**
  - pre-integration-test
  - **integration-test**
  - post-integration-test
  - verify
  - **install**
  - **deploy**

Struktura Maven projektov

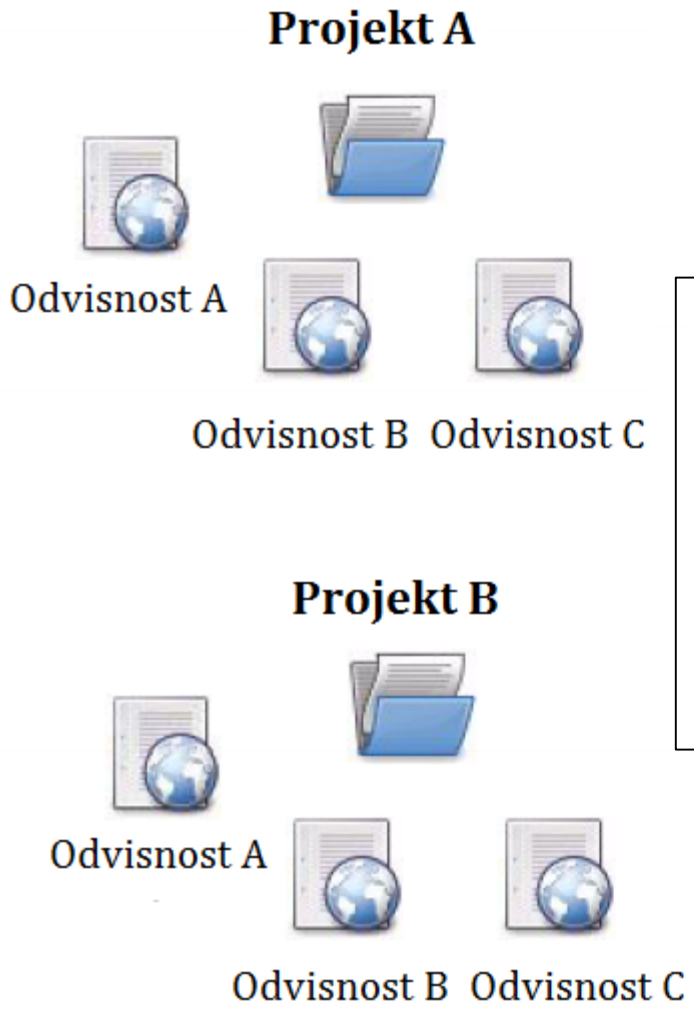
- Uporaba enotne strukture projektov omogoča razvijalcem hitrejše razumevanje vsebine projekta.





Upravljanje odvisnosti

- Upravljanje odvisnosti

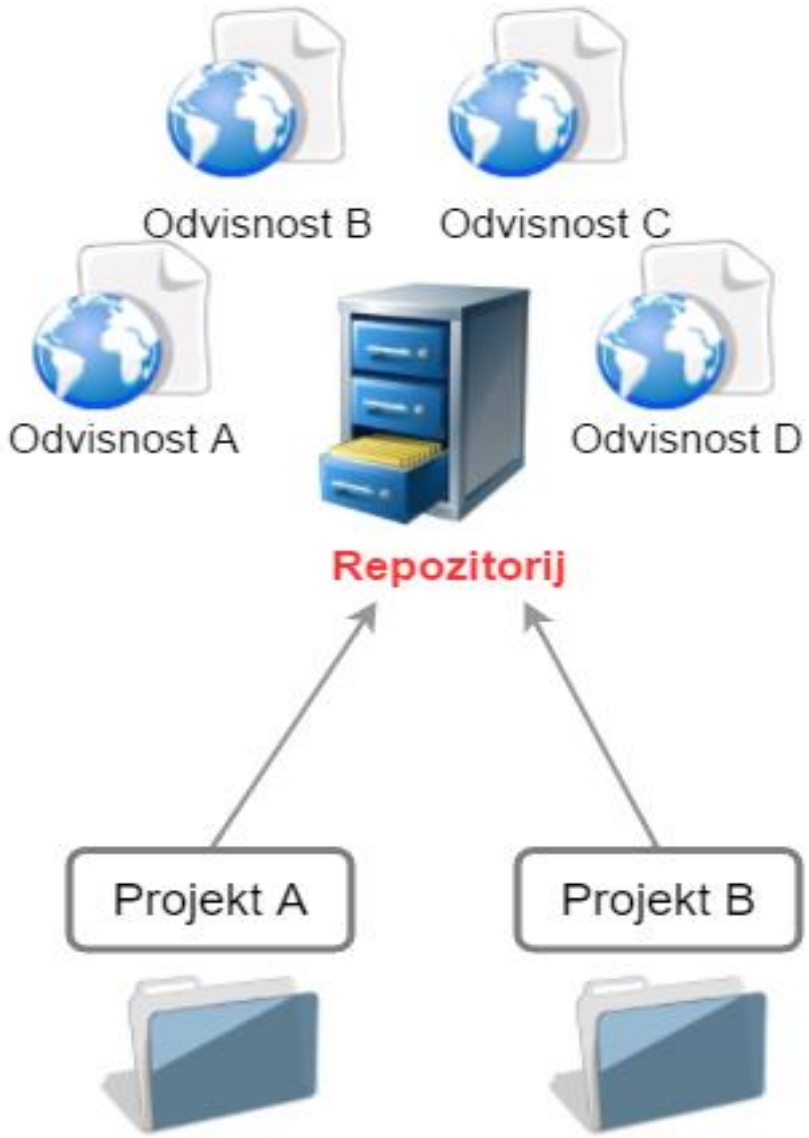


Primer slabe prakse: podvajanje odvisnosti za vsak projekt (npr. kopiranje v mapo lib):

- Zaradi podvajanja je potrebnega več prostora
- Počasna izvedba operacij “check-out”
- Težavno sledenje verzijam

Maven repozitoriji

- Maven repozitoriji



Dobra praksa: uporaba binarnega repozitorija

Repozitorij: skupna lokacija za vse odvisnosti projektov:

- Obstaja samo ena kopija
- Odvisnosti so shranjene izven projekta
- Odvisnosti so definirane v pom.xml

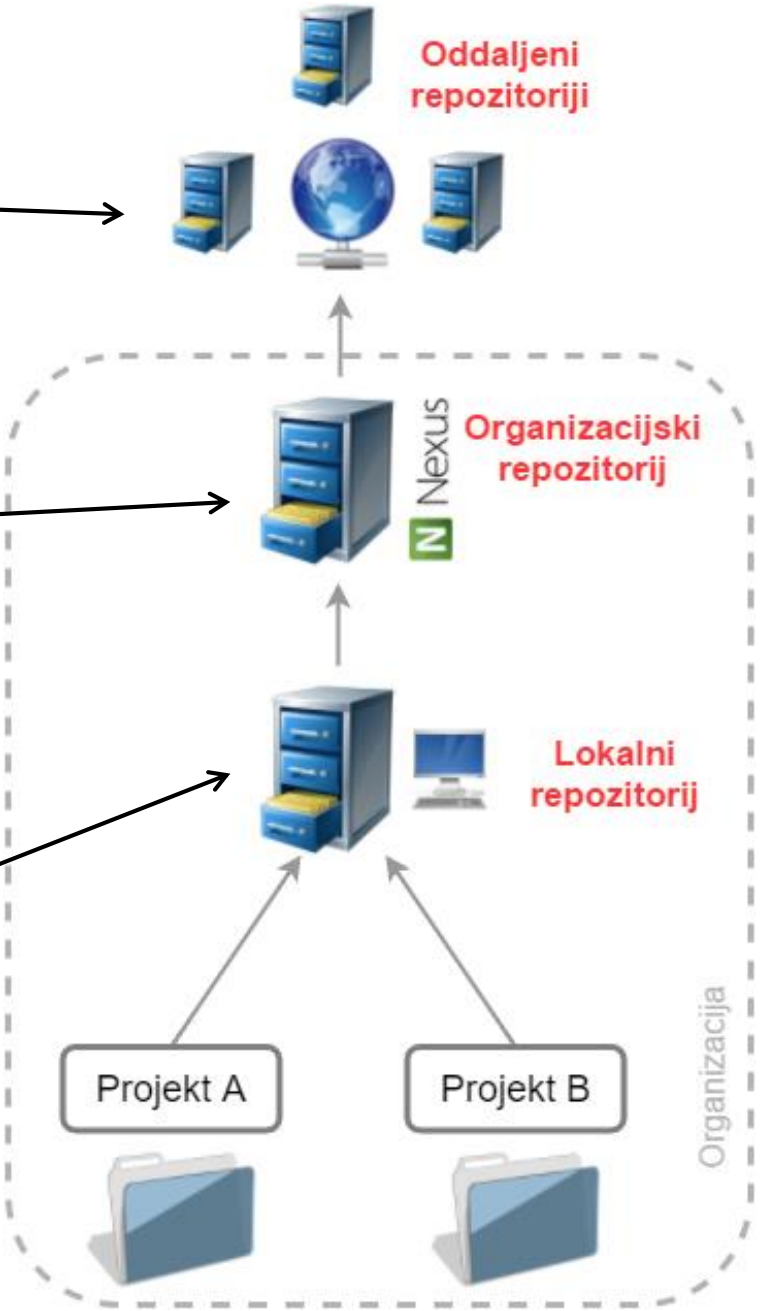
```
<dependencies>
    <dependency>
        <groupId>com.demo</groupId>
        <artifactId>odvisnostA</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>
```

Maven repozitoriji

Privzet oddaljen repozitorij je maven central (repo1.maven.org), uporabljamo pa lahko tudi druge.

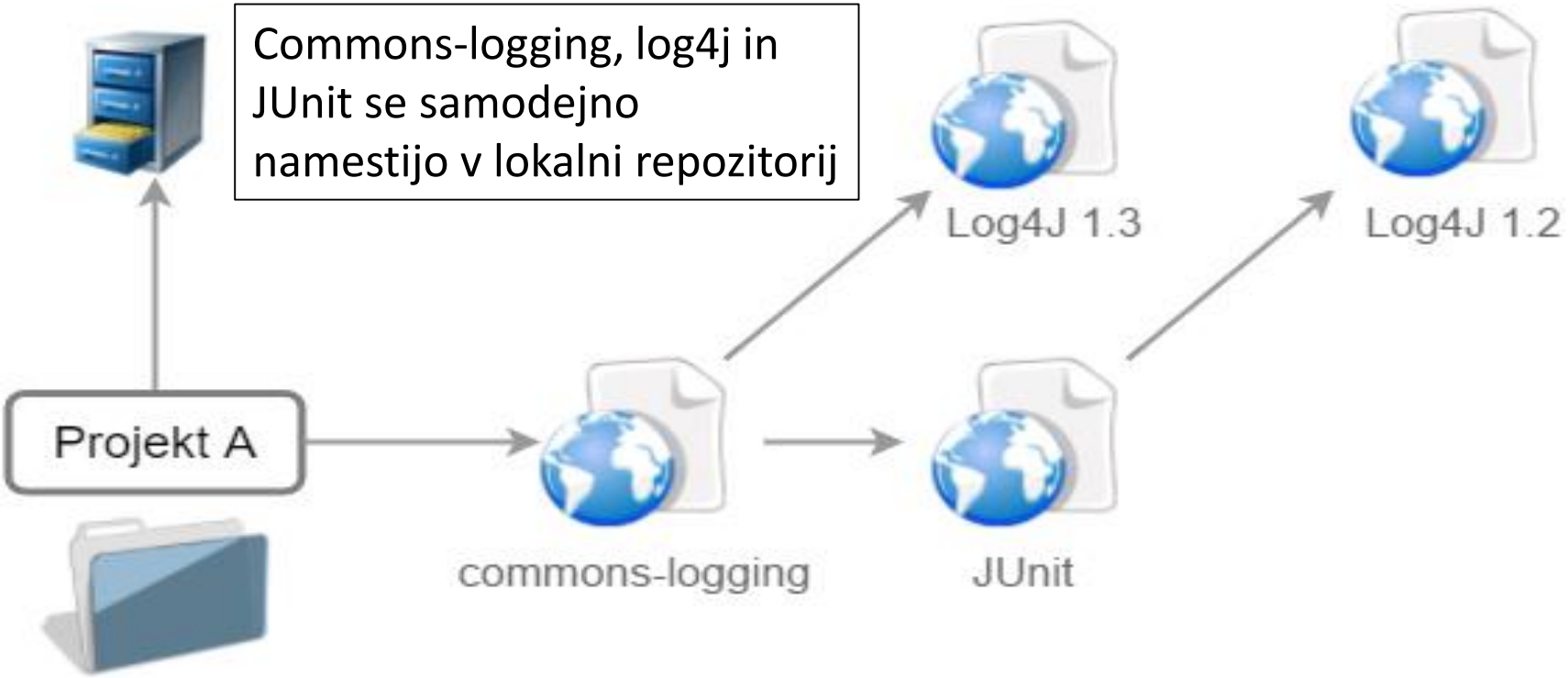
Hrani vse artefakte (tako privatne kot tudi zunanje). Na ta način izboljšamo varnost in hitrost. Uporabimo lahko npr. Nexus, Artifactory, ipd.

Predstavlja predpomnilnik (cache) za artefakte iz oddaljenih repozitorijev. Nahaja se v datotečnem sistemu: USER_HOME/.m2/repository



Upravljanje odvisnosti

- Tranzitivne odvisnosti: Maven prebere pom.xml datoteko uporabljene odvisnosti in samodejno vključi potrebne knjižnice
- Ni omejitve glede števila nivojev



Upravljanje odvisnosti

- Ob definiciji odvisnosti ji je potrebno določiti doseg (scope).
- Maven podpira naslednje tipe dosegov:
  - **compile**: Odvisnost se nahaja v vseh classpath-ih projekta. Te odvisnosti se propagirajo tudi na odvisne projekte. Privzet doseg.
  - **provided**: Pričakujemo, da bo odvisnost v času izvajanja zagotovil JDK ali vsebnik. Odvisnost je na voljo le v času prevajanja in testiranja in ni tranzitivna.
  - **runtime**: Odvisnost ni potrebna za prevajanje, ampak le za izvajanje.
  - **test**: Odvisnost ni potrebna za samo izvajanje aplikacije, ampak le za izvedbo testov.
  - **system**: Odvisnost se ne nahaja v repozitoriju, pot do nje podamo eksplicitno.
  - **import**: Ta scope lahko uporabimo le nad odvisnostmi tipa `pom` v sekciji `<dependencyManagement>`, uporabna pa je za nadziranje verzij uporabljanih odvisnostih, predvsem pri dedovanju.

Upravljanje odvisnosti

- V starševskem (parent) pom-u lahko z uporabo sekcije `<dependencyManagement>` nadziramo verzije uporabljenih odvisnostih v podedovanih projektih.

Starševska datoteka pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    <dependencies>
<dependencyManagement>
```

pom.xml

```
…
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
</dependency>

…
```

Verzije ni potrebno ročno nastavljati, ampak se samodejno nastavi na 4.12. Podobno velja tudi za doseg, ki se deduje iz starševske datoteke pom.xml.

Upravljanje odvisnosti

- Znotraj `<dependencyManagement>` lahko uporabimo odvisnost tipa *bill of materials (BOM)*, ki združuje več odvisnosti, npr. odvisnosti nekega ogrodja.

Starševska datoteka pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.kumuluz.ee</groupId>
            <artifactId>kumuluzee-bom</artifactId>
            <version>${kumuluzee.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
<dependencyManagement>
```

pom.xml

```
<dependencies>
    <dependency>
        <groupId>com.kumuluz.ee</groupId>
        <artifactId>kumuluzee-core</artifactId>
    </dependency>
</dependencies>
```

Vtičniki

- Maven vtičniki (plugins) omogočajo spreminjanje in konfiguracijo Maven build procesa
- Vtičnike vključimo v pom.xml podobno kot odvisnosti
- Vtičnike tipično zvežemo na želeno build fazo in opravimo konfiguracijo
- Tudi vtičniki se nahajajo v Maven repozitorijih
- Primer: konfiguracija vtičnika za namestitev rešitev na IBM WebSphere Application Server:

```
<plugin>
    <groupId>com.orctom.mojo</groupId>
    <artifactId>was-maven-plugin</artifactId>
    <executions>
        <execution>
            <phase>install</phase>
            <goals>
                <goal>deploy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Vtičniki

Standardni:

- maven-compiler-plugin
- maven-surefire_plugin
- maven-failsafe-plugin
- maven-ear-plugin
- maven-ejb-plugin
- maven-jar-plugin
- maven-rar-plugin
- maven-ant-plugin
- maven-release-plugin
- maven-scm-plugin
- maven-eclipse-plugin
- …

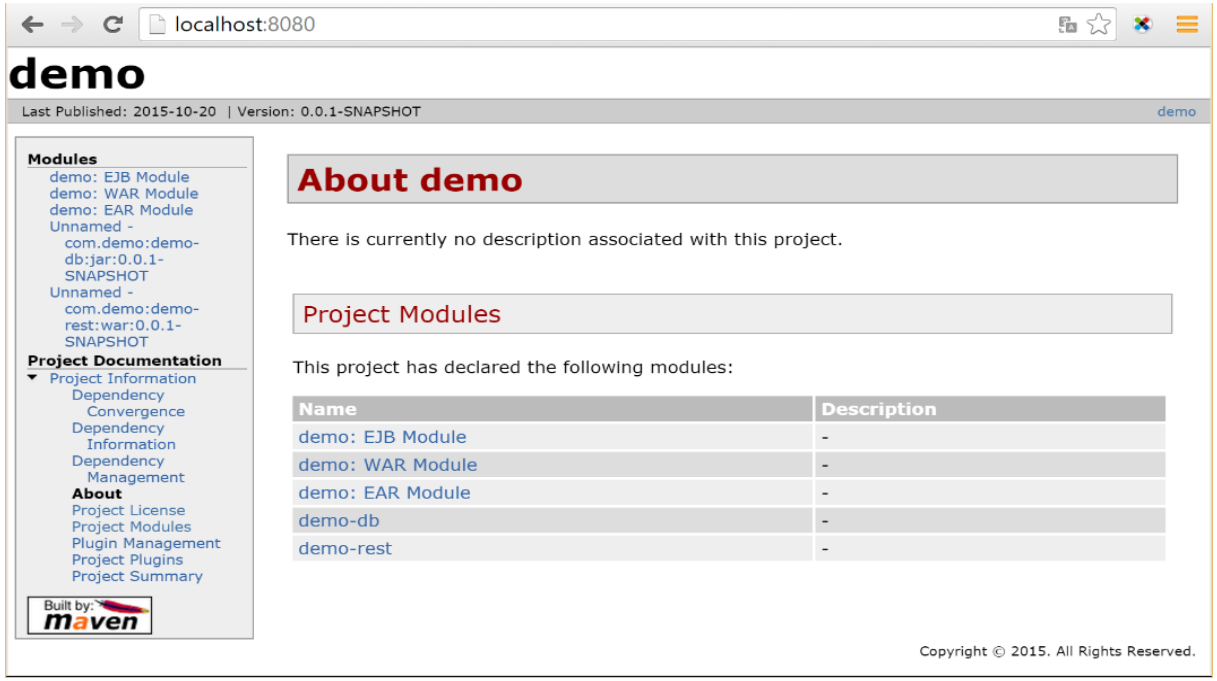
Ostali:

- maven-soapui-plugin
- wildfly-maven-pluginVtičniki Vtičniki
- maven-emma-plugin
- maven-cargo-plugin
- was-maven-plugin
- cxf-java2wsdl-plugin
- dbunit-maven-plugin
- maven-jetty-plugin
- jaxb2-maven-plugin
- cobertura-maven-plugin
- cactus-maven
- …

Izdelava spletne strani z dokumentacijo

- **mvn site**: izdelava HTML strani glede na podatke v pom.xml
- Generirane HTML strani se nahajajo v mapi `target/site`
- **mvn site-deploy**: namestitev strani

```
<project>
    …
    <distributionManagement>
        <site>
            <id>website</id>
            <url>scp://www.demo.si/www/docs/project/</url>
        </site>
    </distributionManagement>
    …
</project>
```



Maven ukazi

- **mvn + faza življenjskega cikla + dodatni parametri**

- Primeri Maven ukazov:

$ mvn package	Prevajanje in pakiranje (JAR, WAR…)
$ mvn install	Namestitev v lokalni repozitorij
$ mvn clean	Čiščenje rezultatov prejšnjih build-ov
$ mvn test	Izvedba testov enote
$ mvn eclipse:eclipse	Generiranje Eclipse projektnih datotek
$ mvn site	Generiranje spletne strani
$ mvn deploy	Namestitev v skupni Maven repozitorij
$ mvn package –X	Pakiranje, vklopljen debug način
$ mvn clean deploy –X –P test	Čiščenje, namestitev, debug, aktivacija profila test

Maven Archetype

- Izdelavo Maven projektov si lahko poenostavimo z uporabo projektnih predlog (Maven Archetypes).
- Z uporabo predlog lahko poenostavimo kreiranje projektov ter povečamo stopno konsistentnosti in uporabo dobrih praks.
- Uporabimo lahko številne pred-pripravljene predloge, ki se nahajajo v javnih repozitorijih, kreiramo pa lahko tudi svoje.
- Archetype je Maven projekt, ki ima poleg `pom.xml` datoteke tudi deskriptor `archetype.xml`.
- Primer kreiranja Maven projekta na podlagi predloge:

mvn archetype:generate

-DgroupId = com.mycompany.app

-DartifactId = my-app

-DarchetypeArtifactId = maven-archetype-quickstart

-DinteractiveMode = false

Datoteka settings.xml

- Datoteka `settings.xml` se uporablja za vse globalne nastavitve, ki niso specifične za posamezen projekt:
  - Definicija oddaljenih repozitorijev
  - Definicija skupnih lastnosti
  - Definicija in aktivacija posameznih profilov
  - Definicija podatkov, ki so specifični za posamezno okolje (npr. podatki za povezavo na strežnik na razvojnem/testnem okolju, ipd.)

- Datoteka se lahko nahaja na dveh mestih:
  - V Maven namestitveni mapi: `$M2_HOME/conf/settings.xml`
  - V uporabniški mapi: `${user.home}/.m2/settings.xml`

- Če obstajata obe mapi, se njuna vsebina združi.

Integracija z IDE

- Maven zagotavlja napredno integracijo z IDE okolji (IntelliJ IDEA, Eclipse, JDeveloper, NetBeans, itd.)

- IntelliJ IDEA integracija omogoča:
  - Delo z Maven projektom, kot da je navaden projekt
  - Samodejno posodabljanje build path-a glede na odvisnosti v pom.xml
  - Napreden urejevalnik pom.xml datotek
  - itd.

Dobre prakse

- Dobre prakse pri uporabi Maven:
  - Uporaba organizacijskega Maven repozitorija (Nexus, Artifactory, ipd.)
  - Uporaba lastnih parent `pom.xml` datotek za posamezne vrste projektov, tako da so vse skupne odvisnosti in vtičniki definirani samo na enem mestu
  - Uporaba unit in integracijskih testov
  - Kreiranje in uporaba lastnih projektnih predlog (archetypes)
  - Pravilen pristop k verzioniranju artefaktov (releases, snapshots)
  - IDE-specifičnih datotek (`.project`, `.classpath`, `.settings`) ter mape `target` ne objavimo v SCM
  - Pred izdajo nove verzije (release) odstranimo vse SNAPSHOT odvisnosti.
  - Uporaba sekcije `<dependencyManagement>`
  - Generiranje spletne strani projekta