CI gyakorlat segédlet

Kovács Gábor

2017. március 10.

1. Bevezetés

A mai gyakorlat célja egy folyamatos integrációra alkalmas környezet kliens és szerver oldala összeállításának bemutatása egy Hello, world Java alkalmazás ,,integrációján" keresztül.

A folyamatos integráció szolgáltatásai jellemzően egy távoli szerver számítógépen érhetők el, ezt illusztrálandó a távoli számítógépünk egy virtuális gép lesz. A virtuális gép operációs rendszere Ubuntu Linux.

Az alkalmazásunkat Java nyelven készítjük. A kliens oldalon egy Git kliensből és Java fejlesztői környezetből áll. A fejlesztői környezetünk szerver oldala a következő elemekből áll:

- Git szerver
- Jenkins
- Java, Ant és Maven (ezek csak együtt értelmesek, és csak Java választása esetén)
- Nexus

2. A szerver oldal előkészítése

A szerver oldalon Ubuntu csomagokból telepítjük a `git`, `ant`, `maven` alkalmazásokat, valamint a Java környezetet.

```
> apt-get install git
> apt-get install ant
> apt-get install maven
> apt-get install python-software-properties software-
   properties-common
> apt-add-repository ppa:webupd8team/java
> apt-get update
> apt-get install oracle-java8-installer
```

Az `ant` a `/usr/share/ant/` könyvárba kerül, a `maven` pedig a `/usr/share/maven/` könyvárba kerül. A `maven` által fordított állományok tárhelye a helyi fájlrendszeren a `~/.m2/repository` lesz. A Java a `/usr/lib/jvm/` alatti verziófüggő alkönyvtárba kerül.

A Jenkins és a Nexus szolgálatásokat a weboldalról letölthető Java alkalmazásként futtatjuk, azok nem igényelnek előzetes telepítést.

3. Git

A Git szerver oldali szolgáltatást jelszó vagy nyilvános kulcs alapú hitelesítéssel védjük. Ezért előkészítésként létrehozzuk a `git` felhasználót, az `adduser` során megadjuk a felhasználó jelszavát. A jelszómentes bejelentkezés lehetővé térelére már a `git` felhasználóval bejelentkezve létrehozzuk annak home könyvtárában az `.ssh` könyvtárat benne a `authorized_keys` fájlt, és beállítjuk azok hozzáférési jogosultságait.

```
> sudo bash
> adduser git
> su git
> cd
> mkdir .ssh
> chmod 700 .ssh
> touch .ssh/authorized_keys && chmod 600 .ssh/
   authorized_keys
```

Ezután bármely fejlesztő nyilvános kulcsát az `authorized_keys` fájl végére másolva a fejlesztő jelszó nélkül hozzáférhet a verziókezelő rendszerhez.

```
> cat kovacsg_rsa.pub >> authorized_keys
```

A verziókezelő szolgáltatás erőforrásait tároljuk a `/opt/git/` könyvtárban, a projektünk fájlai kerüljenek ennek a `teszt.git/` alkönyvtárába, végül állítsuk be ennek tulajdonosát és hozzáférési jogosultságait.

```
> sudo mkdir /opt/git
> sudo chown -R git:git /opt/git/
> cd /opt/git
> mkdir teszt.git
> cd teszt.git/
> git init --bare
> chown -R git:git ../teszt.git/
> chmod -R g+ws ../teszt.git/
```

4. Jenkins

A Jenkins folyamatos integrációs szolgáltatást a `http://jenkins-ci.org/` URL-ről tölthetjük le, amit a `java -jar` paranccsal önállóan is futtathatunk, vagy egy szervlet konténerbe telepíthetünk. A konfigurációs beállítások a `~/.jenkins/` könyvtárba kerülnek.

```
> java -jar jenkins.war
```

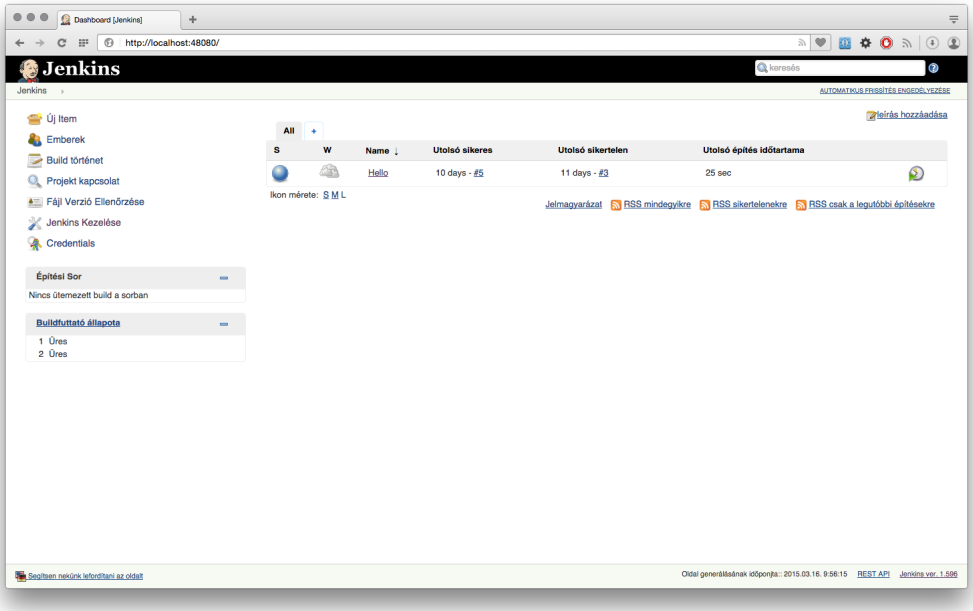
A Jenkins konfigurálását egy webfelületen keresztül végezhetjük le, ami alapértelmezés szerint a helyi gép 8080-as portján érhető el. Mivel esetünkben a távoli gép egy virtuális szerver, NAT beállításokkal a 8080-as portot átirányítjuk a helyi gép valamely szabad portjára.

A Jenkins konfigurációját a kiegészítők kezelésével kezdjük. A Jenkins alapértelmezés szerint támogatja az Ant és Maven szkripteket, viszont a verziókezelők közül hiányzik a Git. Ezért a *Jenkins kezelése* menü *Kiegészítők Kezelése* almenüjében az elérhető pluginek közül kiválasztjuk a *GIT plugin*t, majd telepítjük azt.

A *Rendszerbeállítások* menüben konfiguráljuk a Java alapú fejlesztői környezetünket a fenti Git, Ant, Maven, Java telepítések beállításai alapján. A konfigurációk maradjan a `~/.jenkins` könyvtárban.

Home könyvtár	`~/.jenkins`

A JDK szekcióban adjuk meg a Java telepítés adatait, a JAVA_HOME környezeti változó értékét.



1. ábra. Jenkins menedzsment felülete

JDK installations	
Name	Oracle Java 1.8
JAVA_HOME	`/usr/lib/jvm/java-8-oracle`
Install automatically	false

A Git szekcióban megadjuk a `git` bináris állomány elérési útját.

Git	
Path to Git executable	git
Install automatically	false

Az Ant szekcióban adjuk meg a Ant telepítés adatait, a ANT_HOME környezeti változó értékét.

Ant installations	
Name	Apache Ant
ANT_HOME	`/usr/share/ant`
Install automatically	false

A Maven szekcióban adjuk meg a Maven telepítés adatait, a MAVEN_HOME környezeti változó értékét, és a helyi maven gyűjtemény elérési útját.

Maven installations	
Name	Apache Maven
MAVEN_HOME	`/usr/share/maven`
Maven Project Configuration	
Local Maven Repository	Default (/.m2/repository)

A Jenkins további beállításaiban konfiguráljuk a webes elérés URI-ját, az adminisztrátor email címét, a shellt, és az SMTP szervert.

Jenkins Location	
Jenkins URL	`http://127.0.0.1:8080/`
System Admin e-mail address	kovacsg@tmit.bme.hu
Shell	
Shell executable	/bin/bash
E-mail Notification	
SMTP server	10.211.55.4

Új projektként egy Maven projektet hozunk létre *Hello* néven. (Ha a komplex Hello szolgáltatást C nyelven készítenénk, akkor freestyle projektet hoznánk létre.) A projekt konfigurációs oldalát a 4 ábra mutatja.

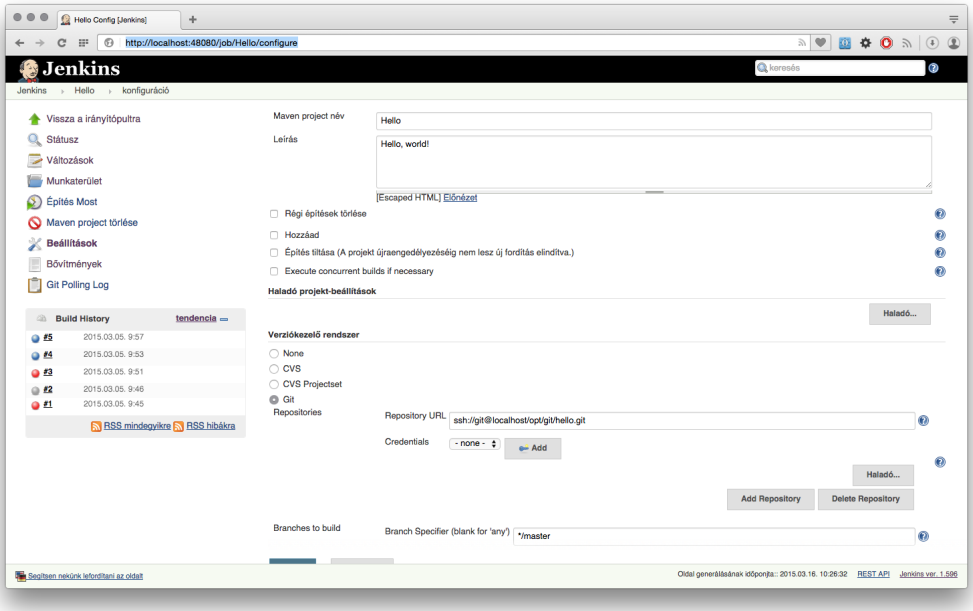
Project név	Hello
Leírás	Hello, world!

A verziókezelő szolgáltatás a projekthez, vagyis a Jenkins feladathoz tartozik. Verziókezelőként válasszuk ki a Gitet, és adjuk meg annak elérési útvonalát. A Jenkinst jelenleg nem `git` felhasználóként futtatjuk, ezért az aktuális felhasználónak jelszómentes bejelentkezést kell engedélyeznünk a Git szerverre. Ezért a Jenkins felhasználójaként létrehozunk egy nyilvános kulcsot, és azt hozzáadjuk a git felhasználó hitelesített kulcsaihoz.

```
> ssh-keygen -t dsa -b 1024
> ssh-copy-id git@localhost
```

Verziókezelő rendszer	Git
Repositories	
Repository URL	`ssh://git@localhost/opt/git/hello.git`

A Hello projekt építését periodikusan végezzük el, beállításunk alapján minden nap 3:15-kor le fog futni az automatikus építés.



2. ábra. A hello projekt konfigurációja

Build Triggers	
Build periodically	kiválasztva
Schedule	15 3 * * *

Maga az építés Maven POM konfiguráció és Maven szabályok alapján fog működni. A Maven projektünk leírója a projekt gyökérkönyvtárában lévő `pom.xml` lesz, és a takarítást végző `clean`, valamint a függőségek feloldását, a fordítást, és a Maven gyűjtőhelyre másolást végző `install` célokat tekintjük építésnek. Ennek hatására a Java projektünkből készített jar állomány bekerül a `~/.m2/repository` könyvtárba.

Build	
Root POM	`pom.xml`
Goals and options	`clean install`

5. A Hello world projekt

A fejlesztői gépen először elérhetővé tesszük a jelszómentes bejelentkezést a Git szerverre a nyilvános kulcs megfelelő helyre másolásával, amit a Git szer-

ver hozzáad a hitelesített kulcsok fájl végére. A hozzáférés SSH-n keresztül valósul meg, az SSH portot a Jenkins portjához hasonlóan konfigurálnunk kell a gazda gép és a virtuális gép közötti NAT-on.

```
> scp -p3022 ~/.ssh/id_rsa.pub git@localhost:.ssh/
    kovacsg_rsa.pub
```

Ezután létrehozunk egy üres Maven projektet. Ezt tetszőleges korszerű Java fejlesztői környezetben megtehetjük, a gyakorlaton NetBeanst használtunk. Alternatívaként konzolon az alábbi paranccsal is létrehozhatjuk ugyanezt:

```
>mvn --version mvn archetype:generate -DgroupId=hu.bme.
    tmit.agile -DartifactId=hello -DarchetypeArtifactId=
    maven-archetype-quickstart -DinteractiveMode=false
```

Ennek hatására létrejön egy `hello` könyvtár benne a projektleíró `pom.xml` fájllal és egy `src/mail/java/` alkönyvtárral, ahova a forráskódok kerülnek. A `pom.xml` egy `hu.bme.tmit.agile` csoportazonosítóval, `hello` erőforrásnévvel és `1.0-SNAPSHOT` verzióval rendelkező projektet ír le. A projekt függőségeit egy `dependencies` címkén belül `dependency` elemekben megadott csoport-azonosító, erőforrásnév, verzió hármasok felsorolásával adhatjuk meg, amelyek határására a Maven letölti számára elérhetővé tett központi tárhelyről az azonosított objektumokat, tipikusan jar fájlokat. A `build` elem beállítása futtathatóvá teszi a jar fájlt annak leírójában a `Main-Class` attribútum beállításával.

```
<?xml version="1.0" encoding="UTF-8"?>
<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 h
ttp://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>hu.bme.tmit.agile</groupId>
  <artifactId>hello</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.
        sourceEncoding>
```

```
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
          <configuration>
            <archive>
              <manifest>
                <mainClass>hu.bme.tmit.agile.hello.
                    Hello</mainClass>
              </manifest>
            </archive>
          </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Írjuk meg a Hello, world alkalmazásunkat:

```
package hu.bme.tmit.agile.hello;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

A projektünk gyökérkönyvtárában hozzunk létre egy lokális Git repositoryt, és nézzük meg a létrejövő állományokat, majd kérdezzük le a repository státuszát.

```
> git init .
Reinitialized existing Git repository in
/Users/kovacsg/NetBeansProjects/hello/.git/
> cd .git/
> ls
```

```
HEAD        config      hooks       objects
branches    description info        refs
> git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be
     committed)

        pom.xml
        src/

nothing added to commit but untracked files present (
   use "git add" to
track)
```

A könyvtárunk két, a Git számára ismeretlen objektumot tartalmaz, a `pom.xml`-t és a `src/` könyvtárat. Tegyük ezeket verziókezeltté!

```
> git add pom.xml
> git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   pom.xml

Untracked files:
  (use "git add <file>..." to include in what will be
     committed)

        src/
```

```
> git add src/main/java/hu/bme/tmit/agile/hello/Hello.
   java
> git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   pom.xml
        new file:   src/main/java/hu/bme/tmit/agile/
           hello/Hello.java
> git commit -m 'initial'
```

A helyi repositoryba a `git commit` után bekerültek a változtatásaink, a következő feladatunk ezek megosztása a többi fejlesztővel. Először is meg kell adnunk, hogy a helyi repository melyik távoli repositoryt tükrözi. Utána létre kell hoznunk az `origin` távoli repositoryban a `master` ágat.

```
> git remote add origin ssh://git@localhost:3022/opt/
   git/hello.git
> git push -u origin master
Counting objects: 12, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (12/12), 1.16 KiB | 0 bytes/s,
   done.
Total 12 (delta 0), reused 0 (delta 0)
To ssh://git@localhost:3022/opt/git/hello.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from
    origin.
```

A fájljaink feltöltődtek az integrációs szerverre, így ott elindíthatunk egy építést a 4 ábrán a menüben látható *Építés Most* linkre kattintással. Sikeres lefutás után a *Build History*ban megjelenik az építés eredménye – kék színnel, ha sikeres volt. A Hello world alkalmazásunk a `~/.m2/repository/hu/bme/tmit/agile/hello/1.0-SNAPSHOT` könyvtárba kerül, amit a `java -jar` pa-

ranccsal futtathatunk. A `hu/bme/tmit/agile` tag a csoportazonosító, a `hello` az objektumazonosító, a `1.0-SNAPSHOT` pedig a verzió.

```
> java -jar hello-1.0-SNAPSHOT.jar
Hello, world!
```

6. Az eredmények megosztása

Ahhoz, hogy a jar fájlunk ne csak a szerver egy hálózaton keresztül hozzá nem férhető könyvárában legyen, egy repository manager szoftver segítségével megosztjuk. A repository maganeger egy URL-t rendel egy helyi fájlrendszeren elérhető erőforráshoz, és képes tükrözni távoli repository managerek erőforrásait.

Repository managernek a Sonatype Nexus szoftverét használjuk. A zip formátumban elérhető alkalmazást kicsomagoljuk, a beágyazott webszerver portát módosítjuk, majd elindítjuk:

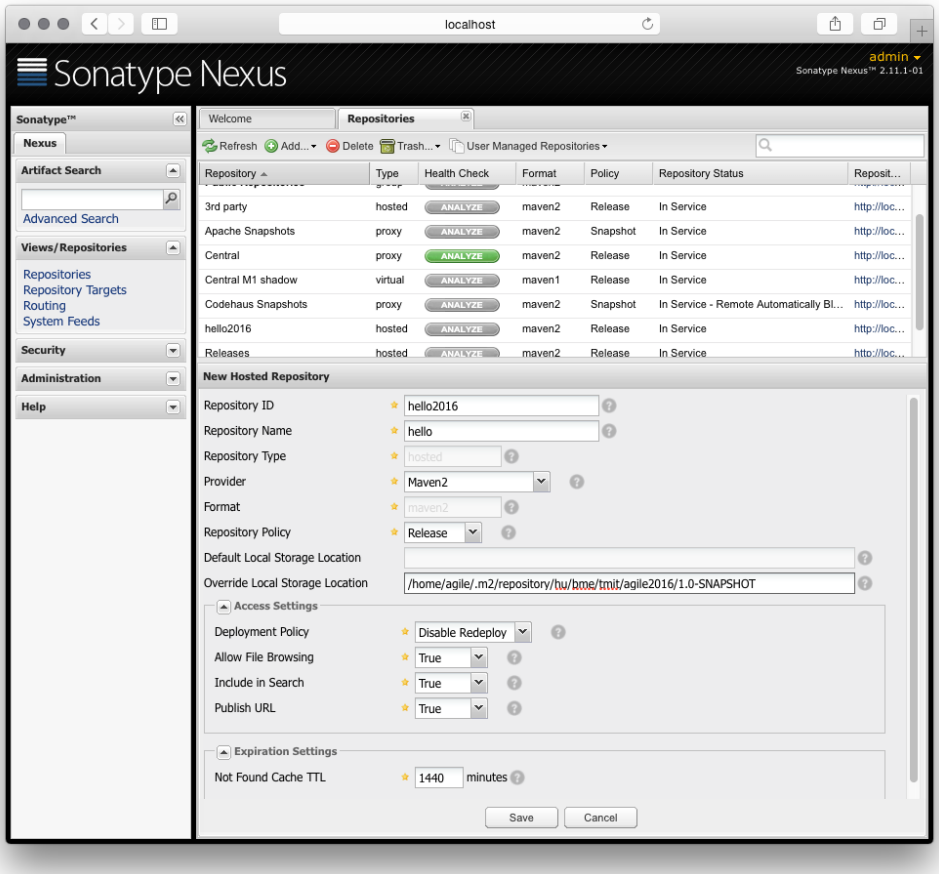
```
> cd nexus
> bin/nexus console
```

Böngészőben megnyitjuk a weboldalt (`http://localhost:48081/nexus`), és bejelentkezünk adminisztrátorként – az alapértelmezett jelszóval. Megnyitjuk a *Repositories* menüt, és létrehozunk egy hosztolt, vagyis helyi fájlrendszeren elérhető, repository-t. Az azonosító meg fog jelenni az URL-ben, a név teszőleges. Helyi tárhelyként állítsuk be a fájlrendszer kiajánlani kívánt könyvtárát.

A repository létrejött, a Maven projekt erőforrásai elérhetők másik felhasználó számára a `http://localhost:48081/nexus/content/repositories/agile2016/` URL-en. A hozzáférést nyilván korlátozhatjuk a projekt tagjaira.

Csatoljuk vissza a projekt tagjai által generált jar fájlokat az alkalmazásunkba. A feladatunk, hogy a Maven alkalmazás tudtára hozzuk, hogy nem csak a központi repositoryból dolgozhat, hanem a projekt céljaira dedikáltból is. A projektleírónkban ezért hozzáadjuk ezt az URL-t mint új repository:

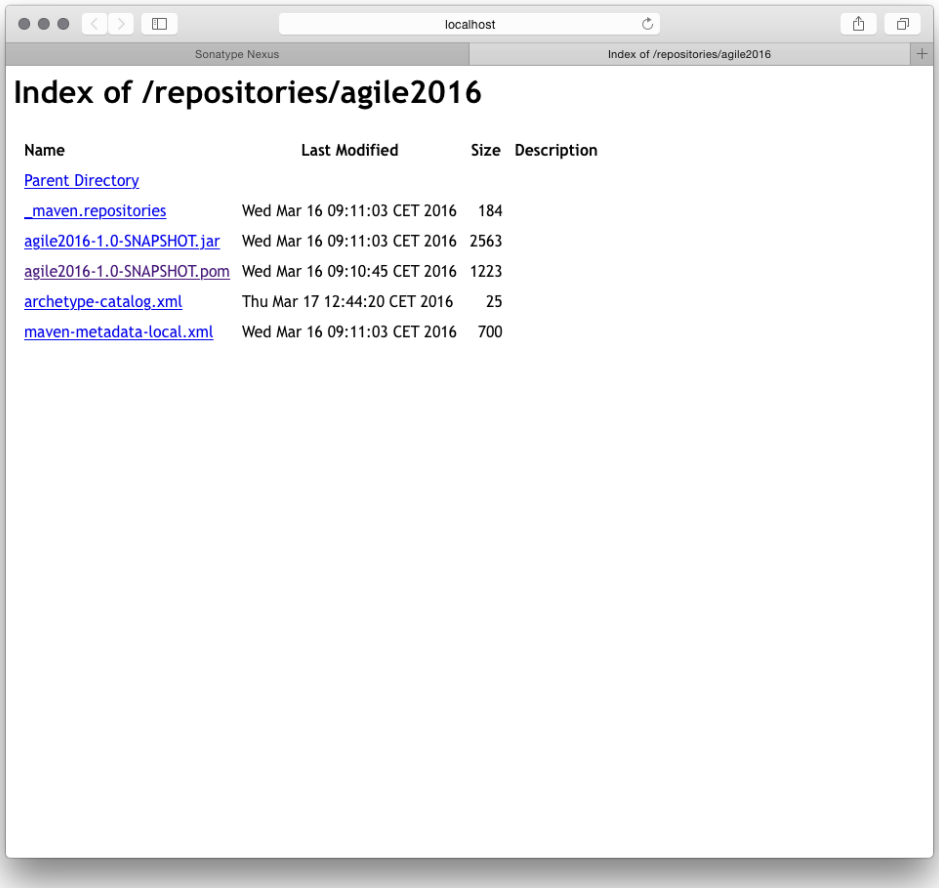
```
<project>
    ...
    <repositories>
        <repository>
            <id>agile</id>
```



3. ábra. A Nexus konfigurációja

```
            <url>http://localhost:48081/nexus/content/
                repositories/agile2016/</url>
        </repository>
    </repositories>
    ...
</project>
```

Ebben a repositoryban található erőforrásokat a csapat fejlesztői függőségként felhasználhatják a saját moduljuk készítése során, a függőségek dependency tagként jelennek meg a projektleíróban. Így megvalósítottuk a munkacso-



4. ábra. A projektünk repository-ja

portok közötti integrációt. A felhasználók git szerverre pusholt forrásai naponta, ütemezetten fordulnak, és bekerülnek a Jenkinst futtató szerver egy könyvtárába. A Nexus elérhetővé teszi azt weben keresztül, így az új verziót a kliens fejlesztői környezete automatikusan letölti, és a kör bezárult.

7. Nem Java projektek

Nem Java, hanem például C-ben írj projekt esetén a Jenkinsben a *Freestyle project* beállításaival hozunk létre új projektet. A Java projekthez ha-

sonlóan itt is meg kell adnunk a verziókezelő rendszerhez való hozzáférést. A különbség a fordításban, és telepítésben jelentkezik. Míg Maven Java projekt esetén a `pom.xml` alapján a Maven elvégezte a fordítást, jar készítést, tesztelést, telepítést stb., addig most ilyen eszközünk nem áll rendelkezésre, saját szkriptet kell írnunk, ami elvégzi ezeket a feladatokat. A szkriptet az *Execute shell* szövegdobozban kell elhelyeznünk. A szkriptet a Jenkins – akárcsak Java projektek esetén – ütemezetten, a verziókezelő eseményeitől függően vagy egyedi kérésre futtatja le. Arról, hogy a kimenet a fájlrendszeren, esetleg egy távoli szerveren a megfelelő helyre kerüljön, szintén ugyanebben a szkriptben kell gondoskodnunk. Shell szkriptről lévén szó, az képes a fájlrendszeren a Jenkins könyvárán kívül lévő más szkripteket, *Makefile*-okat meghívni, így elkerülhetjük, hogy a jelszavainkat webfelületen szöveges formában meg kelljen adnunk.