Continuous Delivery: Problémák és megoldások (http://bit.ly/1rjQOnD)

Viczián István
IP Systems Kft.
http://jtechlog.blogspot.hu

Agenda - miről lesz szó

- Continuous Delivery alapfogalmak
- Maven ellentmondások
- Megoldások

Agenda - miről nem lesz szó

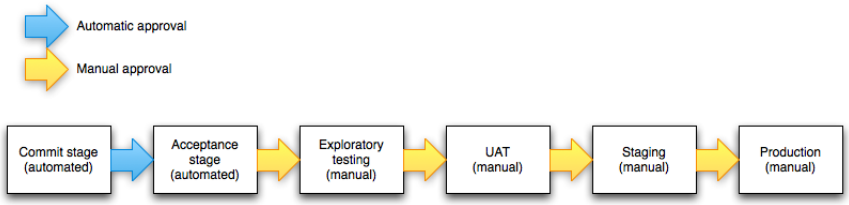
- Ant
- Gradle
- Élet a unit és integrációs teszteken kívül (pl. Kódminőség ellenőrzés, tesztelés konténeren, felületi tesztelés, deploy to container)

Continuous Delivery

- Az ügyfélnek minél előbb kijuttatni a fejlesztésünk
- Mielőbbi visszajelzés
- Szemléletmód
- Nem eszköz vagy eszközrendszer
- Folyamatos karbantartás és fejlesztés

Alapelvek

- Minden commit esetén build, mely kimenete potenciális release
- Pipeline
- A pipeline stage-ek ugyanazon binárison dolgoznak



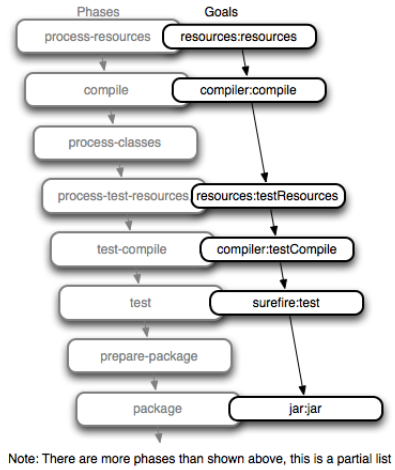
Következmények

- Nem történhet újrabuildelés
- Nem tárolhatunk környezeti információkat az artifact-ban
- Nem használhatunk bytekód instrumentációt pl. lefedettségi tesztelésnél
- Legyen gyors

Ellentmondások

- Maven lifecycle: lefutnak az előző lifecycle phase-ek
- Ősellenség: Maven release plugin
- Integrációs teszt a lifecycle része

- Integrációs tesztek lassan futnak
- Deploy a lifecycle része



release:prepare

```
check-poms: megvizsgálja, hogy az adott verzió SNAPSHOT-e. Ha nem SNAPSHOT, ak
kor már megtörtént a release, így nem futtatható mégegyszer. (Hibaüzenet: You
don't have a SNAPSHOT project in the reactor projects list.)
scm-check-modifications: megvizsgálja, hogy minden állomány be van-e commit-ol
va. Mivel a release plugin aktívan használj a verziókezelőt, ezért a POM-ban a
z scm tag-nek be kell állítva lennie, különben hibaüzenetet kapunk.
check-dependency-snapshots: megvizsgálja, hogy nincs-e SNAPSHOT függőség. Ha v
an, akkor hibaüzenetet dob. A plugin-oknál is vizsgálja, hogy van-e SNAPSHOT.
create-backup-poms: elmenti az előző POM állományt (pom.xml.releaseBackup) név
en. Ha bármi probléma van, ebből vissza lehet állni.
map-release-versions: ha a release interaktív, akkor megkérdezi a felhasználót
ól a release verzióját. Fel is ajánl egyet, az aktuális verziót SNAPSHOT nélkü
l, ha jó, elég egy Enter-t ütni.
input-variables: ha a release interaktív, akkor megkérdezi a felhasználótól a
tag nevét. Fel is ajánl egyet, az modul neve + aktuális verzió SNAPSHOT nélkül
, ha jó, elég egy Enter-t ütni (pl. fooapp-1.1).
map-development-versions: ha a release interaktív, akkor megkérdezi a felhaszn
álótól a következő verzió nevét. Fel is ajánl egyet úgy, hogy növeli a minor v
erziószámot, és hozzáteszi a SNAPSHOT-ot (pl. 1.2-SNAPSHOT). Ha megfelelő, itt
is üthetünk Enter-t.
rewrite-poms-for-release: a POM át lesz írva a release verzióra, és az scm tag
-ben is a tag url-je fog szerepelni.
generate-release-poms: a release-eléshez használt POM legenerálásra.
run-preparation-goals: az új POM-mal lebuild-eli a projektet, clean verify cél
okkal. Közben a teszt esetek is lefutnak.
scm-commit-release: commit-olja a módosított POM-ot.
scm-tag: elvégzi a tag-gelést a verziókezelőben.
rewrite-poms-for-development: a következő verzióhoz, ami már újra SNAPSHOT les
z, elkészíti a POM-ot.
remove-release-poms: eltakarítja a release-hez használt POM-ot.
scm-commit-development: commit-olja az új POM-ot a verziókezelőbe.
end-release: véglegesíti a release-t.
```

release:perform

```
verify-completed-prepare-phase: megvizsgálja, hogy a prepare cél le lett-e fut
tatva.
checkout-project-from-scm: a target könyvtárba a teljes projektet checkout-olj
a, méghozzá az előbb tag-geltet.
run-perform-goals: fork-ol egy új Maven példányt, és elindítja a checkout-olt
projekten a deploy célt.
```

Megoldás - első lépés

- Build nem SNAPSHOT verzióval dolgozik
- Maven versions plugin: verzióemelés pl. revision, gyakorlatilag release
- A fejlesztők SNAPSHOT verziókat használnak

```
mvn versions:set -DnewVersion=1.1

mvn versions:revert
```

Megoldás - második lépés

- Külön integrációs teszt projekt
- Csak integrációs teszt futtatása (profile)

- Bináris újrafelhasználása a local repository-ból
- Nem fordít integrációs teszteket a commit stage
- Nem fut unit teszt az integration test stage-ben

Commit profile esetén csak a base modul, it profile esetén csak az integration-tests modul aktív.

```
<profiles>
     <profile>
       <id>commit</id>
          <modules>
            <module>base</module>
          </modules>
      </profile>
     <profile>
       <id>it</id>
          <modules>
            <module>integration-tests</module>
          </modules>
      </profile>
</profiles>
```

Csak secondary artifact deploy.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.2</version>
            <executions>
                <execution>
                    <id>default-jar</id>
                    <phase>never</phase>
                    <configuration>
                        <finalName>unwanted</finalName>
                        <classifier>unwanted</classifier>
                    </configuration>
                </execution>
                <execution>
                    <goals>
                        <goal>test-jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Megoldás - második (és fél) lépés

- Integrációs tesztek embedded adatbázison (platformfüggetlenség!)
- Flyway séma migrációra
- Spring context cache, fork elkerülése
- Külön szálon integrációs tesztek production adatbázison
- Optimalizált teszt futtatás külön szálon
- Minimálisan tartott alapadatkör
- Párhuzamosítás?

> Flyway: The agile database migration framework for Java,
> http://flywaydb.org/
> Spring: @DirtiesContext

Megoldás - harmadik lépés

- Teszt lefedettség mérés: Jacoco
- Dynamic instrumentation: Java agent - futásidőben

Jacoco report futtatáshoz kell a forrás, secondary artifactként deployolva.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>2.2.1</version>
            <executions>
                <execution>
                    <id>attach-sources</id>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
```

```
        </plugin>
    </plugins>
</build>
```

Függőség a primary és secondary artifact-ra.

```
<dependency>
    <groupId>${project.groupId}</groupId>
 <artifactId>base</artifactId>
 <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>${project.groupId}</groupId>
 <artifactId>base</artifactId>
 <version>${project.version}</version>
    <classifier>sources</classifier>
</dependency>
```

Jacoco futtatása, kell neki a forrás és a class fájlok is:

- Maven dependency plugin kicsomagolja a forrás és class fájlokat
- Jacoco Maven plugin végzi az instrumentációt
- Mivel a Jacoco Maven pluginnek nem lehet megmondani a forrás és class fájlok helyét, Ant plugin-nel riport futtatás

```
<plugin>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <id>prepare-covered-sources</id>
            <goals>
                <goal>unpack-dependencies</goal>
            </goals>
            <phase>generate-resources</phase>
            <configuration>
                <includeClassifiers>sources</includeClassifiers>
                <outputDirectory>${project.build.directory}/covered-sources</o
utputDirectory>
   </configuration>
  </execution>
  <execution>
   <id>prepare-covered-classes</id>
   <goals>
    <goal>unpack-dependencies</goal>
   </goals>
   <phase>generate-resources</phase>
   <configuration>
    <excludeClassifiers>sources</excludeClassifiers>
    <outputDirectory>${project.build.directory}/covered-classes</outputDirecto
ry>
            </configuration>
        </execution>
    </executions>
    <configuration>
        <includeGroupIds>${project.groupId}</includeGroupIds>
    </configuration>
</plugin>

<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${jacoco.version}</version>
    <executions>
        <execution>
            <id>jacoco-initialize</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
    </executions>
</plugin>

<plugin>
    <artifactId>maven-antrun-plugin</artifactId>
    <executions>
        <execution>
            <id>default-cli</id>
            <phase>post-integration-test</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                    <taskdef name="report" classname="org.jacoco.ant.ReportTas
k" classpathref="maven.plugin.classpath" />
                    <report>
                        <executiondata>
                            <file file="${project.build.directory}/jacoco.exec
">
      </file></executiondata>
     <structure name="Integration test coverage">
      <classfiles>
       <fileset dir="${project.build.directory}/covered-classes">
                            </fileset></classfiles>
                        <sourcefiles encoding="UTF-8">
                            <fileset dir="${project.build.directory}/cover
ed-sources">
       </fileset></sourcefiles>
     </structure>
     <html destdir="${project.build.directory}/jacoco-report"/>
```

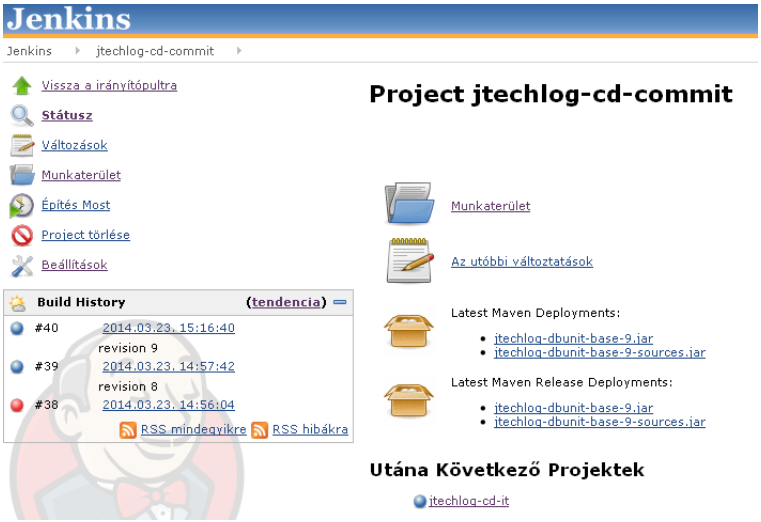
```
                    </report>


                </tasks>
            </configuration>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>org.jacoco</groupId>
            <artifactId>org.jacoco.ant</artifactId>
            <version>${jacoco.version}</version>
        </dependency>
    </dependencies>
</plugin>
```

Jacoco konfigurálása: http://bo2framework.blogspot.gr/2013/10/on-continous-integration-jenkins-maven.html

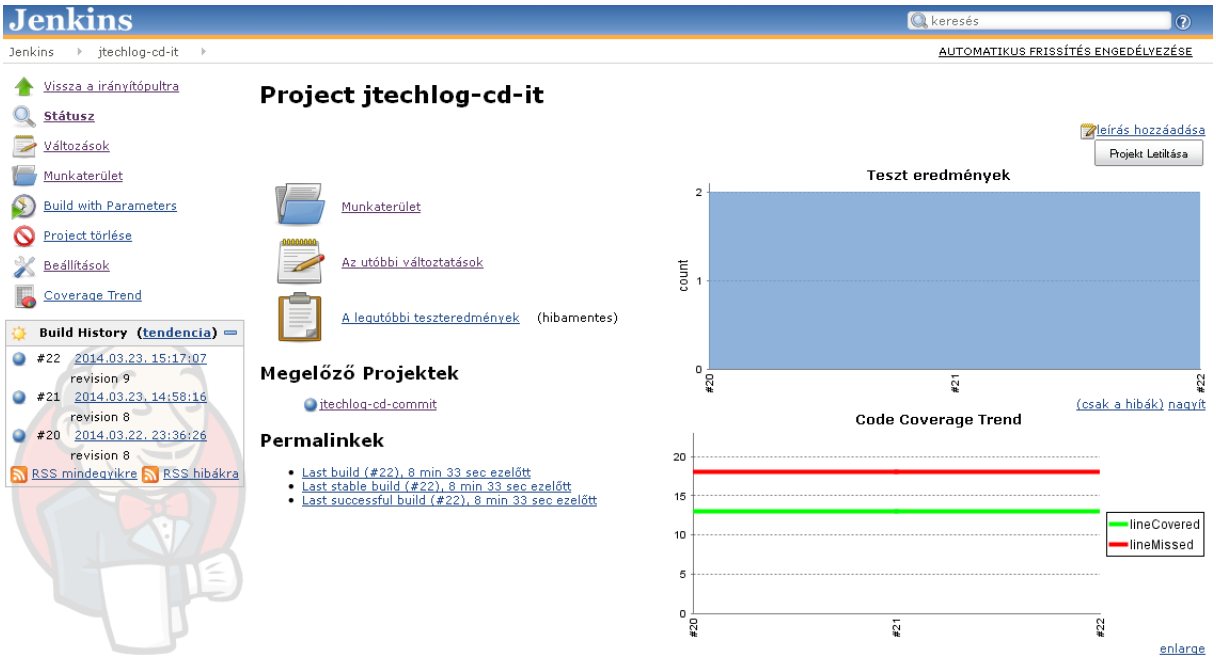
Megoldás - negyedik lépés

- Jenkins commit stage
- Block build when downstream project is building
- `versions:set`
- `deploy -Pcommit`
- `versions:revert`
- Jenkins Maven Deployment Linker plugin: Nexusba deployolt artifact linkelésére
- Jenkins description setter plugin: build leírásában legyen benne a revision szám
- Jenkins Parameterized Trigger plugin: SVN_REVISION továbbadása



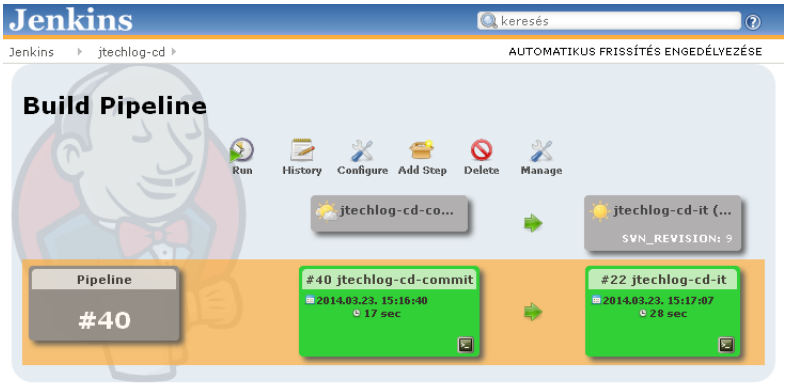
Megoldás - ötödik lépés

- Parameterized build: SVN_REVISION
- Jenkins Subversion plugin trükk, adott revision checkout: file:///D:/houg/repo/jtechlog-dbunit/trunk@${SVN_REVISION}
- `versions:set`
- `clean verify -Pit`
- `versions:revert`
- Jenkins JUnit plugin
- Jenkins JaCoCo plugin



Megoldás - hatodik lépés

- Jenkins Build Pipeline Plugin



További tippek

- Ha csak deploy kell: Maven deploy plugin deploy goal futtatása
- Ne legyen a pipeline része dokumentáció generálás
- Kódminőség ellenőrző eszköz törje meg a buildet, különben figyelmen kívül hagyjuk
- Attól, hogy magas a teszt lefedettségünk, még nem biztos, hogy megbízunk a tesztjeinkben
- Ha alacsony a teszt lefedettség, az óvatosságra int
- Vas kell, akár grid

Problémás esetek

- Több modulból álló alkalmazás esetén a modulok külön buildelése (akár fejlesztői gépen, akár build szerveren)
- Branch
- Workspace másolás
- Archiválás