Automated deployment with Maven and friends
\textit{Going the whole nine yards}

John Ferguson Smart
Principle Consultant
Wakaleo Consulting

Wakaleo Consulting
Optimizing your software development process
Learn about how build automation techniques can also be used to deploy your application
Speaker’s qualifications

- John Ferguson Smart
- Consultant, Trainer, Mentor, Author,...
- Works with Enterprise Java, Web Development, and Open Source technologies
- Author of ‘Java Power Tools’ (O’Reilly)
- Writes articles for sites like JavaWorld, DevX and Java.net, and blogs on Java.net
- Frequent speaker at conferences and Java User Groups
- Likes to write about himself in the third person
Agenda

- What will we cover today
  - Build automation fundamentals
  - The Maven Release Cycle
  - The role of the Maven Enterprise Repository
  - Automating Deployment
  - Automating Web Testing
  - Automating Web Deployment
  - Deploying Database Updates
  - Scripting the Deployment Process
  - A real-world example
Build automation

Build automation - the Big Three

- Maven
- Gradle
- Gant

Build scripting

Automated code quality

- Sonar
- JUnit
- EMMA
- Pmd

Automated testing

- JUnit
- Web
- EasyB
- JBehave
- dbUnit
Build automation

➡️ Step 4 - Automate your deployment process!
Release automation - an overview

1) Maven Release Cycle

2) Automated deployments

3) Automated web tests

4) Database updates

5) Groovy build scripting

Continuous Integration

CI build server

Application server

SCM server

Enterprise Repository

Database

Build script

maven

LIQUID BASE

Groovy build scripting
Release Foundations

Building your release process on solid building blocks:

- The Maven Enterprise Repository
- The Maven Release Cycle
The Maven Enterprise Repository

- A proxy/cache for public libraries
- A repository for internal libraries
  - Snapshots
  - Releases
The Maven Enterprise Repository

- Several options available
  - Nexus
  - Artifactory
  - Archiva
  - File server...
The Maven Release Cycle

- Develop against snapshot versions
  - Each release produces a new time-stamped artifact
  - Snapshot versions are identified by the SNAPSHOT keyword

- Release stable versions
  - Stable, tested release
  - Each artifact is unique

```xml
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

```xml
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0</version>
```
Deploying a release involves many tasks

- Commit all current changes to version control
- Upgrade the version number(s) and commit the changes
- Tag the release version
- Build and deploy this version
- Update the version number and commit the changes
The Maven Release Plugin automates this process:

- Automates the release process
- Updates version numbers
- Commits changes and creates new SCM tags
- Builds, tests and deploys to the Enterprise Repository
Try this at home!

You will need:

- A valid SCM configuration
- The maven-release-plugin

```xml
<scm>
  <connection>scm:git:git://github.com/wakaleo/babble.git</connection>
  <developerConnection>scm:git:git://github.com/wakaleo/babble.git</developerConnection>
</scm>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <configuration>
    <preparationGoals>clean verify install</preparationGoals>
  </configuration>
</plugin>
```

Need this for multi-module projects
The Maven Release Cycle

Using the Maven Release Plugin

$ mvn release:prepare

Audits your code...

- Check that there is no uncommitted code
- Check that there are no snapshot dependencies
- Prompts for release tag, branch and new version numbers
The Maven Release Cycle

Using the Maven Release Plugin

$ mvn release:prepare

...then does the bookkeeping

- Updates version numbers to release versions
- Creates a new release tag in SCM
- Updates version numbers to next SNAPSHOT versions
- Commit these changes to SCM

Agile2009 - Making Agile Real
The Maven Release Plugin - interactive or automated?

```
$ mvn release:prepare
```

- By default, Maven will prompt you for:
  - Release version number
  - Release tag
  - Next snapshot version number

- This is not appropriate for automated builds
  - Use batch mode to use sensible default values

```
$ mvn release:prepare -B
```

No questions asked.
The Maven Release Cycle

- Using the Maven Release Plugin
  
  ```
  $ mvn release:perform
  ```

- Do the real work

  - Checkout a release version
  
  - Build, test and deploy to the Enterprise repository

---

SCM server

Enterprise Repository

1.0.1

1.0.1

public Babbler register

```java
Babbler babbler
if (registered)
throw new Name
```
Automating the Maven Release Plugin

- Don’t use the maven-release-plugin on your own machine
- Run it on a build server instead
  - Accessible to all
  - Reference environment
  - No risk of local code changes
Automated deployment

- Automating the deployment process
  - Don’t rebuild, reuse!
    - Download application binaries from the Enterprise repository
    - Externalize configuration as much as possible
    - Patch if necessary with target environment configuration
    - Deploy to target environment
Automated deployment

- Deployment strategies - direct reuse
  - Reuse from the repository without modification
  - Externalized environment-specific configuration
    - JNDI
    - Local environment-specific properties files
    - ...

Diagram:
- Enterprise Repository
- Application Server
- WAR 1.0.1
Automated deployment

- Deployment strategies - patching
  - Reuse binaries from the repository
  - Modify internal configuration files for the target environment
  - Can use overlays in Maven

NOTE: Use patching only if you can’t deploy your binaries directly!
Automated deployment

Patching - Deploying a web application in 5 easy steps!

- Create a deployment project
- Most of the directories are empty
- These are the files that will patch the web application for different environments
Patching - Deploying a web application in 5 easy steps!

```xml
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>${webapp.version}</version>
  <type>war</type>
</dependency>

<properties>
  <webapp.version>${project.version}</webapp.version>
</properties>
```

- **Add a parameterized dependency to your WAR**
- **Provide a sensible default target version**
Automated deployment

Patching - Deploying a web application in 5 easy steps!

```xml
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <warName>myapp-web</warName>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
        <excludes>
          <exclude>WEB-INF/classes/config.properties</exclude>
        </excludes>
      </overlay>
      ...
  </overlays>
</configuration>
</plugin>
```

Incorporate the web application, but exclude the files to be patched.
Automated deployment

Patching - Deploying a web application in 5 easy steps!

<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    ...
    <webResources>
      <resource>
        <directory>${patch.path}</directory>
      </resource>
    </webResources>
  </configuration>
</plugin>

Include patched files in the final bundle
Automated deployment

Patching - Deploying a web application in 5 easy steps!

- The exact patch files is defined in profiles
- For staging and production, you can also override the version being deployed
- Provide the version to be deployed as a parameter
- The profile indicates the target environment

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>

<profile>
  <id>staging</id>
  <properties>
    <patch.path>src/main/patches/staging</patch.path>
    <webapp.version>${target.version}</webapp.version>
  </properties>
</profile>
```

$ mvn package -Dtarget.version=1.0.5 -Pstaging
Automated web deployment

- Automating the deployment process with Cargo
  - Start, stop and install application servers
  - Deploy to different application servers
  - Run a stand-alone instance
  - Or deploy to an existing server
Automated web deployment

- Deploying to an embedded Jetty instance
- The simplest way to use Cargo with Maven

```xml
<project>
    <build>
        ...
        <plugins>
          <plugin>
            ...
            <version>1.0</version>
          </plugin>
        </plugins>
    </build>
</project>
```

Use the Cargo Maven plugin

Start embedded Jetty instance

Jetty is now running

```bash
$ mvn cargo:start
...
[INFO] [cargo:start]
[INFO] No container defined, using a default [jetty6x, embedded] container
[INFO] [beddedLocalContainer] Jetty 6.x Embedded starting...
...
2009-07-21 16:47:36.470::INFO: Started SelectChannelConnector @ 0.0.0.0:8080
[INFO] [beddedLocalContainer] Jetty 6.x Embedded started on port [8080]
[INFO] Press Ctrl-C to stop the container...
```

Jetty is now running
Automated web deployment

- Deploying to a “real” application server
  - Deploy to a locally installed instance
  - Download and install a server as required
  - Deploy to a running server
  - Start and stop a server
  - ...

Agile2009 - Making Agile Real
Deploying to a local Tomcat server

```xml
<project>
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <version>1.0</version>
        <configuration>
          <container>
            <containerId>tomcat6x</containerId>
            <home>/usr/local/tomcat</home>
          </container>
          <configuration>
            <type>standalone</type>
            <home>target/tomcat6x</home>
          </configuration>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

- Use a locally-installed Tomcat installation
- Tomcat home directory goes here
Automated web deployment

Deploying to a local Tomcat server

```xml
<project>
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <version>1.0</version>
        <configuration>
          <container>
            <containerId>tomcat6x</containerId>
            <zipUrlInstaller>
              <url>http://www.ibiblio.org/.../tomcat-6.0.20.zip</url>
              <installDir>${user.home}/tools/tomcat</installDir>
            </zipUrlInstaller>
          </container>
          <configuration>
            <type>standalone</type>
            <home>target/tomcat6x</home>
          </configuration>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Or download a copy of Tomcat...

...and install it here
Automated web deployment

Deploying to a remote server

```xml
<project>
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <version>1.0</version>
        <configuration>
          <container>
            <containerId>tomcat6x</containerId>
            <type>remote</type>
          </container>
          <configuration>
            <type>runtime</type>
            <properties>
              <cargo.remote.username>admin</cargo.remote.username>
              <cargo.remote.password>
            </properties>
          </configuration>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Deploy to a remote server

Use these properties to connect to the server

Agile2009 - Making Agile Real
Deploying a separate WAR file

```xml
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.0</version>
  <configuration>
    <container>...</container>
    <configuration>...</configuration>
    <deployer>
      <deployables>
        <deployable>
          <artifactId>ebank-web</artifactId>
          <groupId>org.ebank</groupId>
          <type>war</type>
        </deployable>
      </deployables>
    </deployer>
    </configuration>
  </plugin>
```

Deploy a WAR file defined in the dependency list

```xml
<dependencies>
  <dependency>
    <groupId>org.ebank</groupId>
    <artifactId>ebank-web</artifactId>
    <type>war</type>
    <version>${target.version}</version>
  </dependency>
</dependencies>

<properties>
  <target.version>${project.version}</target.version>
</properties>
```

The exact version of the WAR is defined here
Automated web deployment

Deploying a separate WAR file
Automated web testing

- Automated functional/web tests
  - Many tools available:
    - Selenium
    - HTMLUnit
    - JWebUnit
    - Canoo Webtest
    - ...
  - And many strategies:
    - Run tests before deployment
    - Run tests against the deployed application
Automated web testing

- Basic principle - don’t rebuild, deploy!
- Create a special Maven project to run your web tests

```
myapp
myapp-core
myapp-service
myapp-web
```

```
 mvn deploy
```

```
<dependency>
```

```
jetty://
```

```
Now run tests as usual
```

Agile2009 - Making Agile Real
Automated web testing

- Basic principle - don’t rebuild, deploy!
- Create a special Maven project to run your web tests

```mvn deploy```

Or run tests against a test server

Test server
Automated web testing

- Basic principle - don’t rebuild, deploy!
- Create a special Maven project to run your web tests

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>myapp</artifactId>
  <version>1.0.1</version>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>com.example</groupId>
      <artifactId>myapp-core</artifactId>
      <version>1.0.1</version>
    </dependency>
    <dependency>
      <groupId>com.example</groupId>
      <artifactId>myapp-service</artifactId>
      <version>1.0.1</version>
    </dependency>
    <dependency>
      <groupId>com.example</groupId>
      <artifactId>myapp-webtest</artifactId>
      <version>1.0.1</version>
    </dependency>
  </dependencies>
</project>

mvn deploy

Or deploy to and test a production server

Production server
Automated web testing

- Testing Snapshots or Release Candidates - an approach
  - Use a dedicated release candidate branch
    - Changes are merged into the release candidate branches when ready
    - Automated tests are (also) run against the release candidate branch
    - Releases are run against the release candidate branch
    - Updated version numbers need to be merged back to the trunk

[mvn release]
[development trunk]
[release candidate branch]
Automated web testing

Testing Snapshots or Release Candidates - an approach

Use multiple repositories on your Enterprise server

- MyApp
- MyApp-Core
- MyApp-Service
- MyApp-Web

- MyApp-Webtest

Deployment to Release Candidate repository

1.0.1-SNAPSHOT

Now run tests as usual
Deploying Database Updates

- But what about database updates?
  - Schema updates
  - Data updates
  - Data migration
  - Database backups and updates
  - Rollbacks
Several approaches and tools exist, e.g.

- Manual SQL scripts
- Hibernate
- Liquibase
- ...
Deploying Database Updates

- Liquibase - smart database updates
  - Open source tool for managing database changes
  - Database changes are stored in XML form in the SCM
  - Database neutral
  - Many powerful features
    - Roll back changes
    - Merge changes from several developers
    - Generate a SQL update script that a DBA can apply
    - ...

Agile2009 - Making Agile Real
**Deploying Database Updates**

- **Liquibase change sets**
  - Database changes recorded manually in XML
  - More work, but allows more intelligent rollbacks

```xml
<databaseChangeLog
   xmlns="http://www.liquibase.org/xml/ns/dbchangelog/1.6"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog/1.6
   http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-1.6.xsd"
>
   <changeSet id="1" author="john">
     <createTable tableName="department">
       <column name="id" type="int">
         <constraints primaryKey="true" nullable="false"/>
       </column>
       <column name="name" type="varchar(50)">
         <constraints nullable="false"/>
       </column>
       <column name="active" type="boolean" defaultValue="1"/>
     </createTable>
   </changeSet>
</databaseChangeLog>
```

- Each changeset is unique
- Create a new table
Deploying Database Updates

➤ Liquibase change sets

➤ Lots of types of changes are supported
  ➤ DDL updates (add, modify or drop tables, columns or indexes...)

```xml
<databaseChangeLog>
  <changeSet id="2" author="simon">
    <addColumn tableName="department">
      <column name="description" type="varchar(50)"/>
    </addColumn>
    <dropColumn tableName="department" columnName="telex" />
  </changeSet>
</databaseChangeLog>
```

Add a new column
Drop a column
Deploying Database Updates

- Liquibase change sets
  - Lots of types of changes are supported
    - DDL updates (add, modify or drop tables, columns or indexes...)
    - Insert data...

```xml
<databaseChangeLog...>
  <changeSet id="4" author="john">
    <insert tableName="country">
      <column name="id" valueNumeric="1"/>
      <column name="code" value="AL"/>
      <column name="name" value="Albania"/>
    </addColumn>
  </changeSet>
</databaseChangeLog>
```

```xml
<databaseChangeLog...>
  <changeSet id="3" author="simon">
    <loadData tableName="countries" file="countries.csv">
      <column name="id" type="NUMERIC"/>
      <column name="code" type="STRING"/>
      <column name="name" type="STRING"/>
    </loadData>
  </changeSet>
</databaseChangeLog>
```
Deploying Database Updates

Integration with Maven

```xml
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>org.liquibase</groupId>
        <artifactId>liquibase-plugin</artifactId>
        <version>1.9.3.0</version>
        <configuration>
          <propertyFileWillOverride>true</propertyFileWillOverride>
          <propertyFile>src/main/resources/liquibase.properties</propertyFile>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

Use the Maven Liquibase plugin

And a configuration file

```
changeLogFile = changelog.xml
driver = com.mysql.jdbc.Driver
url = jdbc:mysql://localhost/ebank
username = scott
password = tiger
verbose = true
dropFirst = false
```

liquibase.properties
Deploying Database Updates

Integration with Maven

- Commands can be run directly or as part of the lifecycle, e.g.

  $ mvn liquibase:update

  $ mvn liquibase:rollback -Dliquibase.rollbackCount=1

  mvn liquibase:updateSQL

  Update any unimplemented changes

  Rollback latest changes

  Generate SQL update script
Real-world deployments are complex

- Deployments to application servers
- Database backups and updates
- Web service deployments
- ...

Some tasks need to be manual...

...but many don’t
Scripting the deployment process

- Deployment scripting - Groovy is your friend!
  - Close to Java
  - Concise and readable
  - Great for file manipulation

- How can you use Groovy?
  - Use a dedicated Maven project to fetch and/or prepare the artifact
  - Call the Groovy script from your Maven deployment project
  - Or wrap the Maven script in Groovy and add the more specific tasks
Scripting the deployment process

- Using Groovy Scripting in Maven
  - Just use the GMaven plugin
  - Embed Groovy script in your `pom.xml` file
  - Run an external Groovy script
  - Use elements of your `pom.xml` or `settings.xml` in your script
Scripting the deployment process

➤ Using Groovy Scripting in Maven
➤ Just use the GMaven plugin

```xml
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.groovy.maven</groupId>
      <artifactId>gmaven-plugin</artifactId>
      <version>1.0-rc-5</version>
      <configuration>
        <source>
          println "Hi there, I’m \${user.name}. My project is \${project.name}"
        </source>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Run some Groovy code
You can use objects from your pom.xml file
Scripting the deployment process

Using Groovy Scripting in Maven

Another example - invoking another Groovy script

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.groovy.maven</groupId>
      <artifactId>gmaven-plugin</artifactId>
      <version>1.0-rc-5</version>
      <configuration>
        <source>
          def server = settings.servers.find{ it.id.equals('dbserver') }
          """groovy update-scripts.groovy -Ddb.username=${server.username} -Ddb.password=${server.password}""".execute()
        </source>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Fetch username and password details from the settings.xml file

Invoke an external Groovy script and provide the username and password as parameters
An example: JIRA and Bamboo

Automating the release process with JIRA and Bamboo

Issues, releases, and version numbers

Automated builds and deployments

List of available versions

App Server

JIRA

Bamboo

maven

WAR

1.0.0

1.0.1

1.0.2

nx

WAR

1.0.2

App Server
Example: JIRA and Bamboo

Automating the release process with JIRA and Bamboo
Example: JIRA and Bamboo

Automating the release process with JIRA and Bamboo

Substituting variable: ${bamboo.custom.jiraversion.name} with 2.0.92
And in Hudson?

- Use the “parameterized build” plugin

---

Add parameters to your build job
And in Hudson?

Use the “parameterized build” plugin

Hudson prompts you for the parameter when you run the build
In conclusion

What have we covered?
- Using the Maven Release Cycle
- Nexus as a cornerstone of the deployment process
- Cargo to automate your application deployment
- Liquibase to manage your database updates
- Using Groovy in your Maven builds
Thanks for your attention