# MAVEN - DEPLOYMENT AUTOMATION

 

In project development, normally a deployment process consists of following steps

- Check-in the code from all project in progress into the SVN or source code repository and tag it.
- Download the complete source code from SVN.
- Build the application.
- Store the build output either WAR or EAR file to a common network location.
- Get the file from network and deploy the file to the production site.
- Updated the documentation with date and updated version number of the application.

## Problem Statement

There are normally multiple people involved in above mentioned deployment process. One team may handles check-in of code, other may handle build and so on. It is very likely that any step may get missed out due to manual efforts involved and owing to multi-team environment. For example, older build may not be replaced on network machine and deployment team deployed the older build again.

## Solution

Automate the deployment process by combining

- Maven, to build and release projects,
- SubVersion, source code repository, to manage source code,
- and Remote Repository Manager *Jfrog/Nexus* to manage project binaries.

## Update Project POM.xml

We'll be using Maven Release plug-in to create an automated release process.

For Example: bus-core-api project POM.xml

```
<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>bus-core-api</groupId>
   <artifactId>bus-core-api</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <scm>
      <url>http://www.svn.com</url>
      <connection>scm:svn:http://localhost:8080/svn/jrepo/trunk/
      Framework</connection>
      <developerConnection>scm:svn:${username}/${password}@localhost:8080:
      common_core_api:1101:code</developerConnection>
   </scm>
   <distributionManagement>
      <repository>
         <id>Core-API-Java-Release</id>
         <name>Release repository</name>
         <url>http://localhost:8081/nexus/content/repositories/
         Core-Api-Release</url>
      </repository>
```

```
   </distributionManagement>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-release-plugin</artifactId>
            <version>2.0-beta-9</version>
            <configuration>
               <useReleaseProfile>false</useReleaseProfile>
               <goals>deploy</goals>
               <scmCommentPrefix>[bus-core-api-release-checkin]-<
               /scmCommentPrefix>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

In Pom.xml, following are the important elements we've used

| Element | Description |
|---|---|
| SCM | Configures the SVN location from where Maven will check out the source code. |
| Repositories | Location where built WAR/EAR/JAR or any other artifact will be stored after code build is successful. |
| Plugin | maven-release-plugin is configured to automate the deployment process. |

## Maven Release Plug-in

The Maven does following useful tasks using *maven-release-plugin*.

```
mvn release:clean
```

It cleans the workspace in case the last release process was not successful.

```
mvn release:rollback
```

Rollback the changes done to workspace code and configuration in case the last release process was not successful.

```
mvn release:prepare
```

Performs multiple number of operations

- Checks whether there are any uncommitted local changes or not
- Ensures that there are no SNAPSHOT dependencies
- Changes the version of the application and removes SNAPSHOT from the version to make release
- Update pom files to SVN.
- Run test cases
- Commit the modified POM files
- Tag the code in subversion
- Increment the version number and append SNAPSHOT for future release

- Commit the modified POM files to SVN.

```
mvn release:perform
```

Checks out the code using the previously defined tag and run the Maven deploy goal to deploy the war or built artifact to repository.

Let's open command console, go the **C:\ > MVN >bus-core-api** directory and execute the following **mvn** command.

```
C:\MVN\bus-core-api>mvn release:prepare
```

Maven will start building the project. Once build is successful run the following **mvn** command.

```
C:\MVN\bus-core-api>mvn release:perform
```

Once build is successful you can verify the uploaded JAR file in your repository.