Projekte in Maven Central verfügbar machen

# Auszug in die weite Welt

Es gibt heute kaum noch ein größeres Java-Projekt, das vollkommen ohne Open-Source-Frameworks oder Libraries auskommt. So gut wie alle Java-Frameworks nutzen zur Auflösung von Abhängigkeiten Maven Central. Interessant wird es dann, wenn man das eigene Open-Source-Projekt ebenfalls über Maven Central verfügbar machen möchte. Hierfür existiert kein einfacher Uploadbutton, das heißt, wir müssen zur Veröffentlichung einen anderen Weg gehen. Dieser Artikel gibt einen Überblick über die Anforderungen, die ein Projekt erfüllen muss, und die Schritte, die notwendig sind, um ein Open-Source-Projekt vom eigenen Rechner nach Maven Central ausliefern zu können.

von Christian Robert

Kaum ein größeres Java-Projekt kommt heutzutage ohne Open-Source-Frameworks oder Libraries aus. Ob es nun große Abhängigkeiten wie das Spring Framework oder kleine, aber dennoch nützliche Hilfsmittel wie slf4j oder JUnit sind – Frameworks und Libraries bestimmen auch in Enterprise-Anwendungen die Landschaft. Spätestens seit dem Siegeszug von Apache Maven gilt auch die Herausforderung, eben jene Abhängigkeiten im eigenen Projekt (inklusive der Abhängigkeiten von Abhängigkeiten) auf dem aktuellen Stand zu halten, als gelöst.

Hinter den Kulissen verwenden hierfür eigentlich alle Java-Build-Frameworks Maven Central [1] als Quelle für die benötigten Artefakte. In der Praxis läuft dies nahezu unbemerkt und fehlerfrei.

Interessant wird es dann, wenn man das eigene Open-Source-Projekt ebenfalls über Maven Central verfügbar machen möchte. Hier gibt es keinen einfachen Uploadbutton, das heißt, wir müssen zur Veröffentlichung einen anderen Weg gehen.

Als Beispielprojekt, an dem wir die notwendigen Schritte deutlich machen wollen, dient das Projekt *devlauncher* des Autors, das auf GitHub zur Verfügung steht [2].

## Maven Central

Maven Central, die Standardanlaufstelle für alle Artefakte, die von Java-Build-Frameworks wie Apache Maven, aber auch von Apache Ivy, Gradle und anderen Produkten durchsucht wird, erlaubt kein direktes Bereitstellen von Inhalten. Stattdessen müssen neue Versionen von Artefakten zunächst in „Zuliefer-Repositories" eingestellt werden, aus denen sie anschließend nach Maven Central synchronisiert werden.

Die Maven-Central-Dokumentation [3] listet diese Repositories auf. Für unser Beispiel verwenden wir als Zuliefer-Repository, in das wir unsere Artefakte einstellen werden, das Sonatype OSS Repository [4].

## Vorbereitung der pom.xml

Alle Projekte, die über Mavel Central ausgeliefert werden, müssen spezielle Voraussetzungen erfüllen und klar definierte Metainformationen in ihrer *pom.xml*-Datei bereitstellen.

*Hinweis: Sonatype bietet ein Maven-Parent-Projekt an, das in der entsprechenden Dokumentation [5] referenziert wird. Aufgrund von Problemen beim nicht interaktiven Modus sowie der für dieses Beispiel er-*

**Listing 1: Metaangaben in pom.xml**

```
<name>
  Launcher application for Web Applications
</name>
<description>
  Launcher application for Web Applications
</description>
<url>
  https://github.com/perdian/devlauncher
</url>
```

*wünschten expliziten Darstellung dessen, was „hinter den Kulissen" abläuft, wird hier darauf verzichtet, dieses Parent-Projekt zu verwenden.*

Zunächst müssen wir sicherstellen, dass die Standard-Maven-Metainformationen vollständig vorhanden sind (Listing 1).

Wichtig für die Einordnung von Open-Source-Artefakten ist außerdem die Lizenz, unter der das Projekt zur Verfügung gestellt wird (Listing 2).

Des Weiteren wird die Angabe des Quellcodeverwaltungssystems benötigt, in dem die Sourcen der Anwendung vorhanden sind (Listing 3).

Auch der bzw. die Autor(en) muss bzw. müssen angegeben werden (Listing 4).

Nun müssen wir definieren, in welches Repository die erzeugten Artefakte während der Maven-Release-Operation übertragen werden (Listing 5).

Die Auslieferung eines Projekts in das Sonatype OSS Repository muss neben dem Standardartefakt *JAR* immer auch die Sourcen sowie das generierte JavaDoc enthalten. Hierzu müssen wir die entsprechenden Maven-Plug-ins anpassen (Listing 6).

Die letzte Voraussetzung ist, dass die generierten Artefakte mit dem PGP-Schlüssel des Releaseentwicklers signiert werden. Der öffentliche Schlüssel muss zur Verifikation der Signatur auf dem Keyserver *hkp://pool.sks-keyservers.net/* zur Verfügung gestellt werden.

Da das Standard-Maven-GPG-Plug-in (*maven-gpg-plugin*) eine nicht ganz so komfortable Schlüsselverwaltung (gerade in nicht interaktiven Umgebungen wie z. B.

### Listing 2: Lizenzinformationen

```
<licenses>
  <license>
    <name>
      The Apache Software License, Version 2.0
    </name>
    <url>
      http://www.apache.org/licenses/LICENSE-2.0.txt
    </url>
    <distribution>repo</distribution>
  </license>
</licenses>
```

### Listing 3: Quellcodeverwaltungssystem

```
<scm>
  <url>https://github.com/perdian/devlauncher</url>
  <connection>
    scm:git:http://github.com/perdian/devlauncher.git
  </connection>
  <developerConnection>
    scm:git:http://github.com/perdian/devlauncher.git
  </developerConnection>
</scm>
```

### Listing 4: Autoreninformationen

```
<developers>
  <developer>
    <id>perdian</id>
    <name>Christian Robert</name>
    <email>dev@perdian.de</email>
  </developer>
</developers>
```

### Listing 5: Releaseinformationen

```
<distributionManagement>
  <snapshotRepository>
    <id>sonatype-nexus-snapshots</id>
    <name>Sonatype Nexus Snapshots</name>
    <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  </snapshotRepository>
  <repository>
    <id>sonatype-nexus-staging</id>
    <name>Nexus Release Repository</name>
    <url>https://oss.sonatype.org/service/local/staging/deploy/
                                                    maven2/</url>
  </repository>
</distributionManagement>
```

### Listing 6: Konfiguration der Maven-Plug-ins

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-source-plugin</artifactId>
  <version>2.1.2</version>
  <executions>
    <execution>
      <id>attach-sources</id>
      <goals>
        <goal>jar-no-fork</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>2.7</version>
  <executions>
    <execution>
      <id>attach-javadocs</id>
      <goals>
        <goal>jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

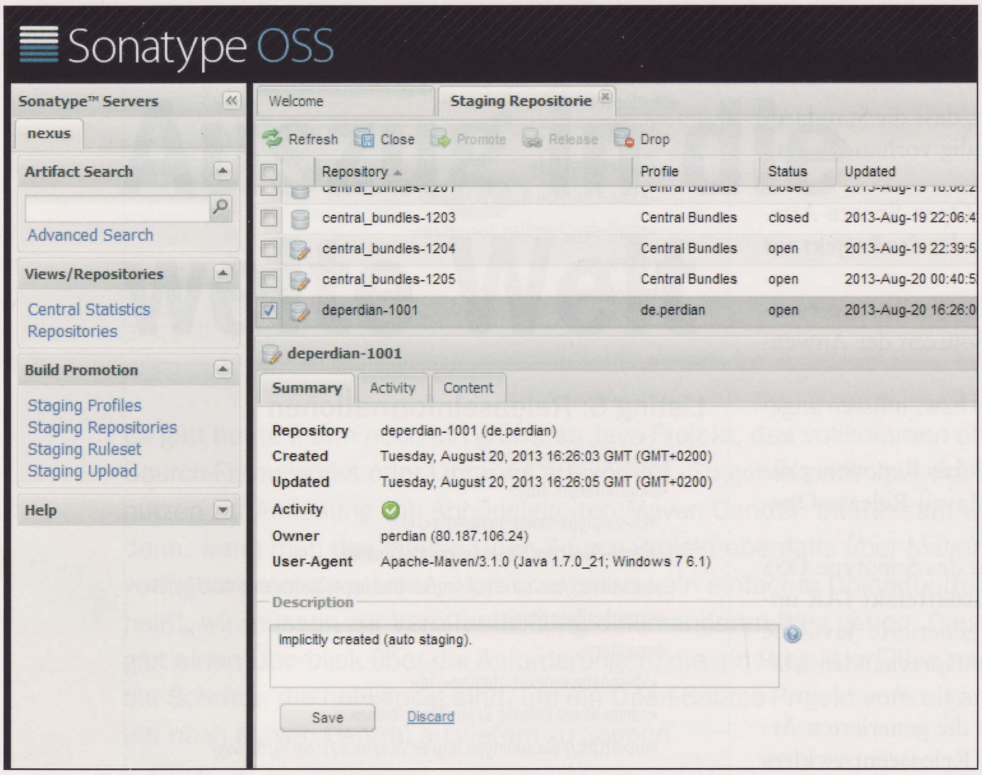


Abb. 1: Für die hochgeladenen Artefakte wurde ein Repository erstellt

einem Build im Continuous Integration Server) bietet, verwenden wir zum Verschlüsseln als Alternative das Maven-PGP-Plug-in von Kohsuke Kawaguchi, dem Hauptentwickler von Jenkins ([6], Listing 7).

### Testlauf

Nachdem nun alle notwendigen Einstellungen in der *pom.xml*-Datei getätigt wurden, können wir einen Testlauf des Builds unseres Projekts starten, um zu verifizieren, dass alle Plug-ins korrekt eingebunden wurden.

Da – wie beschrieben – die Artefakte mit einem PGP-Schlüssel signiert werden müssen, wir diesen inkl. Passphrase jedoch nicht innerhalb der *pom.xml*-Datei angeben wollen, muss dieser beim Build über die Kommandozeile übergeben werden:

**Listing 7: Konfiguration von pgp-maven-plugin**

```
<plugin>
  <groupId>org.kohsuke</groupId>
  <artifactId>pgp-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <goals>
        <goal>sign</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```
$ mvn clean install
  -Dpgp.secretkey=keyfile:/home/perdian/
                                key-secret.asc
  -pgp.passphrase=literal:xxxxxxxxxxx
```

Als erstellte Dateien innerhalb des *target*-Ordners finden wir nun die in Listing 8 gezeigten.

Es wurden also alle benötigten Artefakte (JAR, POM, JavaDoc sowie Sources) sowie die dazugehörigen PGP-Signaturen korrekt erstellt. Damit sind alle Vorbereitungen abgeschlossen, und wir können die Artefakte in das Sonatype OSS Repository hochladen.

### Sonatype-OSS-Repository-Account

Zunächst jedoch müssen wir einen Account im Sonatype OSS Repository einrichten, über den die Artefakte in das Repository eingestellt werden können [7].

Damit wir die notwendigen Berechtigungen erhalten, um Artefakte für die *groupId* unseres Projekts in das Repository einzustellen, muss zunächst ein Ticket unter https://issues.sonatype.org/browse/OSSRH erstellt werden. Als entsprechender Issue Type muss *New Projekt* verwendet werden.

Neben der obligatorischen Beschreibung des Projekts muss die *groupId* definiert werden, für die später Artefakte in das Repository eingestellt werden sollen. In unserem Beispiel also *de.perdian.apps.devlauncher* bzw. allgemeiner *de.perdian*, das damit auch alle Unterpackages mit einschließt.

Nachdem dieses Ticket erstellt worden ist, müssen wir nun warten, bis es von einem Mitarbeiter von Sonatype bearbeitet und als „resolved" markiert worden ist. Erst dann können wir mit der eigentlichen Auslieferung unserer Artefakte beginnen. In der Regel dauert dies nicht länger als ein bis zwei Tage.

### Artefakte ausliefern

Nun haben wir sowohl unser Projekt vorbereitet als auch unseren Account im Sonatype OSS Repository erfolgreich angelegt. Nach der Freischaltung können wir also endlich damit beginnen, unsere Artefakte tatsächlich zur Auslieferung zu bringen.

Ähnlich wie bei der Angabe des PGP-Schlüssels sowie der Passphrase müssen wir bei der Auslieferung über die Maven-Deploy-Funktionalität allerdings noch die Zugangsdaten für das Sonatype OSS Repository definieren. Diese Zugangsdaten sind identisch mit der Anmeldung auf issues.sonatype.org. Dies geschieht, wie bei Maven üblich, über die Datei *settings.xml* unterhalb des *.m2*-Ordners im Benutzerverzeichnis (Listing 9).

Ein kompletter Maven-Build-Zyklus inklusive Deployment sieht nun wie folgt aus:

```
$ mvn clean deploy
  -Dpgp.secretkey=keyfile:/home/perdian/key-secret.asc
  -pgp.passphrase=literal:xxxxxxxxxxx
```

Geschafft! Sofern alle Zugangsdaten richtig angegeben wurden, sind die Artefakte an das Repository übertragen worden. Dort sind sie jedoch bis jetzt nur in einem Staging Repository zu finden. Um die finale Freigabe zu erteilen, müssen wir uns am Repository anmelden und die entsprechende Aktion durchführen. Das Webinterface hierzu (ein Nexus Frontend) erreichen wir unter https://oss.sonatype.org/index.html.

Nach Auswahl des Menüpunkts View/Repositories | Repositories erhalten wir eine Liste aller aktuell vorhandenen Staging Repositories. Auch für die von uns gerade hochgeladenen Artefakte ist ein solches Repository erstellt worden (**Abb. 1**).

Um die im Staging Repository vorhandenen Artefakte nun freizugeben und in das Release-Repository zu übertragen, müssen wir das Repository zunächst über die Operation *Close* schließen. Hierbei wird vom Repository eine Reihe an Überprüfungen durchgeführt, die sicherstellen, dass auch tatsächlich alle benötigten Informationen und Inhalte vorhanden und valide sind.

Sind alle diese Überprüfungen erfolgreich durchgelaufen, so geht das Staging Repository in den *Closed*-Zustand über.

Erst jetzt ist die Operation *Release* verfügbar, die endgültig die Artefakte aus dem Staging Repository in das Release-Repository überführt, das anschließend nach Maven Central synchronisiert wird und damit unser finales Projekt zur Verfügung stellt.

*Hinweis: Beim allerersten Release muss, nachdem die Freigabe für die groupId erstellt wurde, ein entsprechender Kommentar zum JIRA-Ticket hinzugefügt werden, damit dieses allererste Release noch einmal per Hand von einem Sonatype-Mitarbeiter überprüft wird. Alle weiteren Releases erfordern diese manuelle Überprüfung dann jedoch nicht mehr.*

**Listing 8: Erstellte Dateien**

```
de.perdian.apps.devlauncher-devlauncher-javadoc.jar.asc
de.perdian.apps.devlauncher-devlauncher-sources.jar.asc
de.perdian.apps.devlauncher-devlauncher.jar.asc
de.perdian.apps.devlauncher-devlauncher.pom.asc
devlauncher-1.0.0-SNAPSHOT.jar
devlauncher-1.0.0-SNAPSHOT.pom
devlauncher-1.0.0-SNAPSHOT-javadoc.jar
devlauncher-1.0.0-SNAPSHOT-sources.jar
```

**Listing 9: settings.xml**

```
<settings>
  ...
  <servers>
    <server>
      <id>sonatype-nexus-snapshots</id>
      <username>JIRA-ID</username>
      <password>JIRA-Passwort</password>
    </server>
    <server>
      <id>sonatype-nexus-staging</id>
      <username>JIRA-ID</username>
      <password>JIRA-Passwort</password>
    </server>
  </servers>
  ...
</settings>
```

## Zusammenfassung und Ausblick

Auf den ersten Blick erfordert das Einstellen eines eigenen Projekts in Maven Central einen erheblichen administrativen Aufwand – sowohl bei der Anpassung des eigenen Projekts als auch bei der Integration in das Sonatype OSS Repository. All dies dient jedoch dazu, den Prozess der Abhängigkeitsauflösung über Maven Central weiterhin so qualitativ hochwertig und problemlos zu halten, wie er aktuell ist.

Eine einmal an Maven Central ausgelieferte Version ist für den Autor weder änderbar, noch lässt sie sich wieder zurückrufen. Es ist daher dringend zu empfehlen, zur Releaseverwaltung auf das Maven-Release-Plug-in [8] zurückzugreifen, um auch bei zukünftigen Releases eine einfache und konsistente Vergabe von Versionsnummern sowie die korrekte Einbindung in den Build-Prozess sicherzustellen.

**Christian Robert** ist Senior Developer für Mobile-Lösungen bei SapientNitro in Köln. Seit über zehn Jahren beschäftigt er sich mit der Konzeption und Entwicklung von Individualsoftware im Java-Umfeld. Seine aktuellen Schwerpunkte liegen in der Entwicklung von pragmatischen und dennoch (oder gerade deswegen) effizienten Softwarelösungen im mobilen Umfeld. Außerdem interessiert er sich intensiv für die Ideen der Software-Craftsmanship-Bewegung.

## Links & Literatur

[1] http://search.maven.org/

[2] https://github.com/perdian/devlauncher

[3] http://maven.apache.org/guides/mini/guide-central-repository-upload.html

[4] http://nexus.sonatype.org/oss-repository-hosting.html

[5] https://docs.sonatype.org/display/Repository/Sonatype+OSS+Maven+Repository+Usage+Guide

[6] http://kohsuke.org/pgp-maven-plugin

[7] https://issues.sonatype.org/

[8] https://maven.apache.org/maven-release/maven-release-plugin/