Build System

Build System

Build Installer

Voraussetzungen

- Linux oder Mac OsX (Windows Nutzer: vielleicht funktioniert es mit cygwin, habe ich aber nicht probiert)
- typische Commandozeilen Tools, u.a. `bash`, `zip`, `unzip`, `xargs`, `grep`, `find`, `tree` im PATH
- aktuelles maven (z.Z. 3.2.5) im PATH (Bitte KEIN maven aus der Paketverwaltung, dass ist immer völlig veraltet!)
- aktuelles ant (z.Z. 1.9.4) im PATH (Bitte KEIN ant aus der Paketverwaltung, dass ist immer völlig veraltet!)
- Java7 SDK im PATH (bei Java6 handelt man zusätzlich Probleme mit dem signieren von JARs ein), d.h. `java`, `javac`, `pack200`, `unpack200`, `jar`, `jarsigner`, `keytool`, usw. sind im PATH
- Launch4j in Version 3.0.2

Einmalig für die ausgecheckten Quellen

- `cd <nuclos>; cp build.properties_template build.properties`
- `build.properties` editieren und an die lokalen Verhältnisse anpassen. `3rdparty.dir` ist ein (zunächst) leeres Verzeichnis, in denen die Tomcats und PostgreSQL Installationspakete hinein geladen werden.

Eigentlicher Build des Installers

- `cd nuclos-installer`
- `ant main`

Lokaler Maven Build

- Voraussetzungen: Siehe 'Installer Build'
- `cd <nuclos>`
- `mvn -Pquick clean install`

Bekannte Probleme

- Es ist möglich, aber nicht einfach, den Installer Build unter Mac OsX zu benutzen.
- Dagegen gibt es (ungelöste) Schwierigkeiten, den Installer Build unter vagrant zu benutzen.
- Die Signiertools (jarsigner, keytool) von Java 6 *funktionieren nicht zuverlässig* und führen regelmäßig zu Schwierigkeiten beim Build (s.u.)
- Um Java 6 und Java 7 Build zu erzeugen, müssen einige Schwierigkeiten von Maven umgangen werden (s.u.)
- Die unterschriebenen JARs (bzw. deren `pack200` Pakete) werden vom Build lokal im Verzeichnis `~/jar.cache` abgelegt, um den Build zu beschleunigen. Dies ist notwendig, da der Jenkins zum Erzeugen aller signierten JARs für einen Nuclos 4.4 Build *etwa 6 Stunden* benötigt.
- Mittelfristig wäre zu prüfen, ob nicht ein Umstieg von Maven (und ant und Bash Skripte) auf Gradle den Build vereinfachen könnte.

Java 6 jarsigner

Der Java 6 `jarsigner` hat regelmäßig Schwierigkeiten, JARs richtig zu unterschreiben und merkt es dann noch nicht einmal. Die Probleme erkennt man daher erst beim Client Start über Web Start. In der Vergangenheit waren u.A. folgende JARs betroffen:

- aspectjweaver-1.8.1
- xmlbean-2.5.0

Das ist auch der Grund, warum es unter Java 6 nicht möglich ist, ein all-in-one JAR zu erzeugen.

Java 6 *und* Java 7 Builds mit maven

Maven betreibt einen lokalen JAR Cache (normalerweise unter `~/.m2/repository`), der für selbst gebaute JARs und heruntergeladene Abhängigkeiten verwendet wird. Wir bauen jedoch Java 6 und Java 7 JARs. Damit in einem Java 6 Build nicht auf einmal Java 7 JARs auftauchen (die dann nicht funktionieren), ist es notwendig, für die beiden Builds *unterschiedliche* JAR Caches zu verwenden.

Der JAR Cache kann in der `settings.xml` (normalerweise `~/.m2/settings.xml`) konfiguriert werden. Maven lässt sich jedoch davon überzeugen, eine andere `settings.xml` Datei zu verwenden. Auf dem Jenkins wird daher die Standard settings.xml Datei (die als JAR Cache `~/.m2/repository` verwendet) für die Java 6 Builds benutzt, während für Java 7 Builds eine `~/.m2/settings-j7.xml` Datei verwendet wird (die als JAR Cache `~/.m2/repository-j7` benutzt). Entsprechende setting Dateien sind (Nuclos intern) in Slack zu finden.

JAR Signatur und *Timestamp*

Oracle ermutigt sehr stark, *signierten* JARs einen *Timestamp* zu verpassen (d.h. signierte JARs *ohne* Timestamp führen zu einer *Warnung* bei der Überprüfung der Signatur). Daher bekommen die von uns signierten JARs seit einiger Zeit einen (cryptographischen) Timestamp. In den den `modjar*.sh` Scripten ist dafür die Zeile

```
SIGN_OPTIONS="-tsa http://tsa.starfieldtech.com/ -tsacert $ALIAS"
```

verantwortlich. Ein großer Nachteil dieser Timestamps ist jedoch, dass dadurch das Signieren der JARs deutlich länger braucht. Insbesondere ist die Antwortzeit des Dienstes stark erhöht, wenn man bereits einige JARs signiert hat. Auch deswegen ist der `~/jar.cache` zwingend erforderlich.

Hier wird ein kostenloser (nicht-dokumentierter) Timestamp Service (tsa.starfieldtech.com) aus dem Internet verwendet. Die Zeit, um signierte JAR mit Timestamp zu erzeugen, könnte sich durch die Verwendung eines *kommerziellen* Timestamp Service wahrscheinlich deutlich reduzieren.

Link Sammlung

Maven

Maven Tipps und Tricks

- Maven Properties

Ant

Ant Installer Build

- Problem mit dem Maven 2.x Ant Tasks

jarsigner / JARs signieren

- Timestamp Support in Java7
- keystore-explorer
- Export private keys from jks keystore
- jksexportkey
- Java keystore types
- portecle (GUI)