

# Large-Scale Development (LSD)

## Assignment 5: Der Build-Server *Jenkins*

Autoren: Felix Hefner, Max Jando, Severin Kohler

Stand: 13. November 2017

### 1 Einleitung

Im Rahmen der Vorlesung LSD (Large-Scale-Development) sollten die Autoren dieses Dokuments den Opensource-Build-Server *Jenkins* aufsetzen und eine Build-Pipeline anlegen, welche den in vorherigen Assignments bereits behandelten Tomcat 6.0.5 baut. Dieses Dokument ist in folgende Teile untergliedert:

1. Installation von Jenkins auf einem Ubuntu-Server
2. Konfiguration der Build-Jobs in Jenkins
  1. Erstellung der Jenkinsfiles
  2. Weitere Konfiguration auf der Weboberfläche von Jenkins

### 2 Installation von Jenkins auf einem Ubuntu-Server

Da das Ubuntu Server-Betriebssystem mit der Paketverwaltung `apt-get` ausgeliefert wird, konnte diese auf relativ simple Weise dazu benutzt werden, das Jenkins-Paket zu installieren. Jedoch war dieses nicht in den Standard-Paketquellen zu finden, sodass eine spezielle von den Entwicklern der Software hinzugefügt werden musste. Im Großen und Ganzen wurde sich hierbei an die offizielle Anleitung<sup>1</sup> gehalten. Genauer wurde wie folgt vorgegangen:

1. Manuelles Hinzufügen des Publickeys der Jenkins.io - Server, damit diesen vertraut wird
2. Hinzufügen der Jenkins-Paketquelle durch

```
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable
binary/ > /etc/apt/sources.list.d/jenkins.list '
```

---

<sup>1</sup><https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu>

3. Aktualisieren der Paketquellen sowie Installation des Pakets durch  
`sudo apt-get update ; sudo apt-get install jenkins`
4. Da der Paketmanager nach der Installation von selbst den Befehl `service start jenkins` aufruft, läuft Jenkins ab sofort unter `http://<IP-des-Servers>:8080`.

### 3 Konfiguration der Build-Jobs in Jenkins

Die Build-Jobs, die in Jenkins zum Kompilieren des Java-Codes, zum Ausführen der Tests sowie zum Deployen des fertigen `.jar`-Archivs wurden primär über sogenannte *Pipelines* anhand von *Jenkinsfiles* erstellt. Hierbei handelt es sich um Scripte, welche in der Sprache Groovy geschrieben werden und sich in Stages unterteilen. Dies sind Abschnitte, welche später auch in der Weboberfläche von Jenkins sichtbar werden. Der Inhalt des **Jenkinsfile** wurde zunächst direkt in der Weboberfläche eingetragen, nach dem ersten Build-Durchlauf konnte jedoch diese Datei aus dem Github-Repository heruntergeladen werden und anschließend von dort verwendet werden. Insgesamt wurden mehrere Build-Jobs für folgende Zwecke erstellt, damit die Aufgaben strikt getrennt sind:

- **Jenkins\_CI\_Build:** Jenkins-Job zum Kompilieren des Sourcecodes, Testens und erstellen einer `.jar`-Datei.
- **Jenkins\_CI\_Deployment:** Deployment (kopieren des `.jar`-Files an eine definierte Stelle sowie beenden des alten Tomcat-Prozesses und starten eines neuen) der Ergebnisse von **Jenkins**.
- **Jenkins\_Prod\_Build:** Siehe **Jenkins\_CI\_Build**, lediglich ein weiterer Build, um eventuelle Auslieferung (Production) anbieten zu können, ohne die Entwicklung im CI-Build zu beeinflussen. Dieser Build wird erst gebaut, sobald der CI-Build erfolgreich war.
- **Jenkins\_Prod\_Deployment:** Analog zu **Jenkins\_CI\_Deployment**

#### 3.1 Erstellung der Jenkinsfiles

Das folgende Listing zeigt eines der erstellten Jenkinsfiles, welches für dieses Projekt benutzt wurde. Es handelt sich um die Datei **Jenkinsfile\_CI\_build**, welche für den ersten Build-Job **Jenkins\_CI\_Build** benutzt wird. Das Script für **Jenkins\_Prod\_Build** fällt identisch aus. Es wurde jedoch an einem separaten Pfad abgelegt, um es zukünftig leichter austauschen zu können.

```
5 | node {  
    //-----  
    stage name: 'clean '  
    //-----
```

```

11 checkout scm
   env.JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
   env.PATH="${env.JAVA_HOME}/bin:${env.PATH}"
   sh "cd 'tomcat' ; mvn 'clean'"

   //-----
   stage name: 'compile'
   //-----

17   sh "cd 'tomcat' ; mvn 'compile'"

   //-----
   stage name: 'test'
   //-----

23   sh "cd 'tomcat' ; mvn test"

   //-----
   stage name: 'assembly'
   //-----

29   sh "cd 'tomcat' ; mvn assembly:single"
   }

```

Listing 1: Jenkinsfile zum Bauen, Testen und Archivieren von Tomcat mit Maven

Dieses Script ist in die vier Stages *clean*, *compile*, *test* und *assembly* aufgeteilt. In *clean* wird das in Jenkins angegebene SCM<sup>2</sup> ausgecheckt, die Umgebungsvariablen `JAVA_HOME` und `JAVA_PATH` gesetzt sowie `mvn clean` ausgeführt, was die zuvor Kompilierten Dateien löscht. In *compile* wird das gleichnamige Maven-Target aufgerufen, welches den Java-Sourcecode kompiliert. In *test* werden analog dazu mit Maven die Tests ausgeführt. Abschließend wird in *assembly* der Befehl `mvn assembly:single` auf die Shell gegeben, welcher eine ausführbare `.jar`-Datei erstellt.

Die Jenkinsfiles für das Deployment fallen deutlich kürzer aus:

```

1 node {
   //-----
   stage name: 'killing_tomcat_process'
   //-----

7   sh '/var/lib/jenkins/kill_tomcat.sh CI'

```

<sup>2</sup>Source Code Management

```

13 //-----
    stage name: 'copying_files '
    //-----

    sh 'cp "/var/lib/jenkins/workspace/Tomcat/tomcat/ \
target/tomcat-6.0.5-jar-with-dependencies.jar" \
"/var/lib/jenkins/tomcat-6.0.5-CI.jar"'

19 //-----
    stage name: 'starting_tomcat '
    //-----
    sh 'cd "/var/lib/jenkins" ; \
nohup java -jar tomcat-6.0.5-CI.jar &'
}

```

Listing 2: Jenkinsfile zum Verbreiten von Tomcat mit Maven

Zunächst wird in der Stage *kill\_tomcat\_process* mit einem kleinen selbstgeschriebenen Script<sup>3</sup> bei Bedarf der aktuell laufende Tomcat-Prozess beendet. Danach wird das zuvor erzeugte Java-Archive an die vorgesehene Stelle kopiert. Zuletzt wird dieses mithilfe des Befehls `nohup`, welcher die Ausgabe eines Befehls in eine Log-Datei umleitet, im Hintergrund (durch Verwendung von `&` am Ende des Befehls) gestartet. Hierbei ist noch zu erwähnen, dass die `.jar` und somit der Tomcat nicht startet, insofern im selben Verzeichnis nicht die Unterverzeichnisse `conf` und `webapps` liegen und mit entsprechendem Inhalt gefüllt sind. Ersteres Verzeichnis enthält ein paar Konfiguration zu Tomcat im `.xml`-Format<sup>4</sup>, letzteres Benutzerinhalt wie Servlets und JSPs.

### 3.2 Weitere Konfiguration auf der Weboberfläche von Jenkins

Im folgenden werden Schritt für Schritt sämtliche Einstellungen, die wir für die von uns angelegten Jenkins-Pipelines getätigt haben, anhand von Screenshots gezeigt und anschließend genauer erläutert.

Die nachstehende Abbildung zeigt sämtliche Jobs die wir in Jenkins angelegt haben. Die Ampellichter geben an, ob der Build erfolgreich war<sup>5</sup>. Das Wetter gibt den durchschnittliche erfolgsrate der letzten 5 builds an je schlechter das Wetter desto mehr Fehlschläge gab es<sup>6</sup>.

<sup>3</sup>Dieses kann hier bei Github gefunden werden: Script `kill_tomcat.sh` auf Github

<sup>4</sup>Hier musste zudem die Datei `server.xml` bearbeitet werden, um den Standardport `8080` von Tomcat auf `8081` zu ändern, da hier ja bereits der Jenkins läuft.

<sup>5</sup>Grün=Erfolg, Rot=Fehlschlag, Grau=Geplant, aber noch nicht gestartet

<sup>6</sup><https://wiki.jenkins.io/display/JENKINS/Dashboard+View>

		<a href="#">Tomcat</a>	17 Stunden - <a href="#">#61</a>	21 Stunden - <a href="#">#47</a>	11 Minuten
		<a href="#">Tomcat_CI_Deployment</a>	17 Stunden - <a href="#">#21</a>	20 Stunden - <a href="#">#7</a>	5 Minuten 17 Sekunden
		<a href="#">Tomcat_Prod</a>	17 Stunden - <a href="#">#59</a>	21 Stunden - <a href="#">#38</a>	10 Minuten
		<a href="#">Tomcat_Prod_Deployment</a>	17 Stunden - <a href="#">#20</a>	17 Stunden - <a href="#">#14</a>	2,2 Sekunden

Abbildung 1: Überblick über die Jenkins-Jobs

In der folgenden Übersicht, welche in Jenkins *Stage View* genannt wird, sind die einzelnen Build Prozesse (stages) des **Tomcat**-Jobs aufgeführt, sowie ob diese Erfolgreich waren und wie lange sie gebraucht haben. Außerdem wird die durchschnittliche Zeit der Stages angezeigt.

### Stage View

			clean	compile	test	assembly
Average stage times: (Average full run time: ~7min 1s)			7s	34s	3min 31s	41s
#59	Nov 12 18:01	No Changes	7s	37s	3min 32s	28s
#58	Nov 12 18:00	No Changes	10s	40s	3min 36s	28s
#57	Nov 12 17:56	No Changes	9s	41s	3min 29s	1min 1s
#56	Nov 12 17:55	No Changes	10s	48s	3min 34s	44s
#55	Nov 12 17:47	1 commits	7s	22s	3min 26s	48s

Abbildung 2: Überblick der Laufzeiten des Jenkins-Jobs **Tomcat**

General Build Triggers Advanced Project Options Pipeline

Pipeline name: Tomcat

Beschreibung:

[Nur Text] [Vorschau](#)

☐ Alte Builds verwerfen

☐ Dieser Build ist parametrisiert.

☐ Do not allow concurrent builds

☒ GitHub-Projekt

Project url: <https://github.com/rsd-lecture/repo-01/>

Erweitert...

Abbildung 3: Generelle Einstellungen

Hier wird der Name der Pipeline (**Tomcat**) angegeben und welche Art von Projekt (Github-Project) dies ist. Im Zusammenhang dazu muss die URL des Repositories angegeben werden von dem dann gebaut wird.

Build Triggers

☐ Builds zeitgesteuert starten

☐ GitHub hook trigger for GITScm polling

☒ Source Code Management System abfragen

Zeitplan: \*\*\*

⚠ Meinen sie mit „\*\*\*“ wirklich „jede Minute“? Vielleicht meinen Sie eher „H\*\*\*“

Letzter Lauf am Sonntag, 12. November 2017 16:02 Uhr MEZ; Nächster Lauf am Sonntag, 12. November 2017 16:02 Uhr MEZ

☐ Post-Commit-Benachrichtigungen ignorieren

☐ Starte Build, nachdem andere Projekte gebaut wurden

☐ Projekt deaktivieren

☐ Ruheperiode

☐ Builds von außerhalb starten (z.B. skriptgesteuert)

Abbildung 4: Auslöser für den Build

Mit diesem Parameter wird angegeben in welchen Intervallen überprüft werden soll ob Änderungen im Git-repository stattgefunden haben. In unserem Fall wird dies jede Minute überprüft. Wie Cron-Zeitangaben angegeben werden ist hier <sup>7</sup> erläutert. Liegen Änderungen vor werden diese gepollt und gebuildet.

<sup>7</sup><https://help.ubuntu.com/community/CronHowto>

Abbildung 5: Pipeline Einstellungen

Hier werden die Credentials<sup>8</sup> des Git-Projektes ausgefüllt um sich bei Github zu Authentifizieren. Des Weiteren muss die Git-Branch spezifiziert werden. Als letztes haben wir den Pfad zum Pipeline-Skript angegeben (Jenkinsfile\_CI\_build)

Abbildung 6: Abhängigkeiten zwischen Jenkins-Jobs

Manche unserer Jenkins-Jobs, nämlich die beiden für das Deployment, sind abhängig von den Build-Jobs, da ja nur etwas verbreitet werden kann, wenn es erfolgreich gebaut wurde. Deshalb haben wir beispielsweise bestimmt, dass der **Tomcat\_Deployment**-Job erst dann ausgeführt wird, wenn der **Tomcat**-Job erfolgreich abgeschlossen wurde.

<sup>8</sup>Username und der bei Github hinterlegte Public-Key