Continous Integration mit LaTeX und Jenkins

Martin Kraetke, Uwe Ziegenhagen

Angeregt durch einen englischen Blog-Artikel[1] möchten wir den Begriff »Continous Integration« vorstellen und zeigen, wie man dieses Konzept auf LaTeX-Workflows anwenden kann.

Einführung

Da nicht vorauszusetzen ist, dass jeder Leser mit dem Stichwort aus dem Titel etwas anfangen kann, soll an dieser Stelle eine kurze Einführung gegeben werden. Die Wikipedia[2] beschreibt »Contious Integration« – im folgenden durch »CI« abgekürzt – wie folgt:

> Kontinuierliche Integration (auch fortlaufende oder permanente Integration; englisch *continuous integration*) ist ein Begriff aus der Software-Entwicklung, der den Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung beschreibt. Das Ziel der kontinuierlichen Integration ist die Steigerung der Softwarequalität. Typische Aktionen sind das Übersetzen und Linken der Anwendungsteile, prinzipiell können aber auch beliebige andere Operationen zur Erzeugung abgeleiteter Informationen durchgeführt werden. Üblicherweise wird dafür nicht nur das Gesamtsystem neu gebaut, sondern es werden auch automatisierte Tests durchgeführt und Softwaremetriken zur Messung der Softwarequalität erstellt. Der gesamte Vorgang wird automatisch ausgelöst durch Einchecken in die Versionsverwaltung.
>
> Eine vereinfachte Variante der kontinuierlichen Integration – und häufig ihre Vorstufe – ist der Nightly Build (nächtlicher Erstellungsprozess). In der Praxis trifft man auch auf kontinuierliche Integration, gepaart mit einem Nightly Build, um die Vorteile beider Systeme zu kombinieren.

Zusammenfassend können wir sagen, dass CI einen Prozess beschreibt, der kontinuierlich prüft, ob Bestandteile einer Software die erforderliche Qualität besitzen. Was hat das nun mit LaTeX zu tun?

Größere LaTeX-Projekte, insbesondere dann wenn mehrere Autoren daran arbeiten, haben ähnliche Probleme wie Softwareentwickler: nicht sofort ist klar, was eine bestimmte Änderung im Code (das Laden eines neuen Paketes) bewirkt und ob das

[1] https://www.pentapie.com/latex-meets-cicd/

[2] `https://de.wikipedia.org/wiki/Kontinuierliche_Integration`

Gesamtprojekt noch erfolgreich in ein PDF kompiliert werden kann. Hier kann CI beispielsweise dafür sorgen, dass in der finalen Versionskontrolle stets eine geprüfte und damit fehlerfreie Version des Projekts abgelegt wird.

Jenkins bei le-tex

Hello World

Jenkins – Installation

In diesem Abschnitt möchte ich kurz zeigen, wie man Jenkins installieren und konfigurieren muss, um eigene LaTeX-Dokumente damit zu übersetzen.

Jenkins bildet die Softwarebasis, die wir zur Steuerung des Build-Prozesses nutzen werden und wurde ursprünglich unter dem Namen »Hudson« bei Sun Microsystems entwickelt. Jenkins benötigt Java – es basiert auf den sogenannten »Enterprise Java Beans« und wird über den Webbrowser bedient.

Im folgenden beschreibe ich das Setup unter Ubuntu, es sind aber auch Installationspakete für Windows, Mac OS X und diverse Linux-Varianten verfügbar.

Die eigentliche Installation verläuft unspektakulär, siehe das folgende Listing. Bei den meisten Debian-basierten Linuxen dürfte ein `apt-get update` mit anschließendem `apt-get install jenkins` ausreichen, für meine Linux-Version Xubuntu »Yakkety Yak« gab es jedoch noch kein Paket, sodass ich das Debian Repository einbinden musste, siehe `https://pkg.jenkins.io/debian/`.

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
# deb https://pkg.jenkins.io/debian binary/ in
# die Datei /etc/apt/sources.list mit aufnehmen
sudo apt-get update
sudo apt-get install jenkins
```

Nach der Installation – ein Neustart ist nicht notwendig – kann dann über Port 8080 des Installationsrechners auf die jenkins-Oberfläche zugegriffen werden. Im ersten Schritt müssen wir das Initial-Passwort aus einer Datei der jenkins-Installation kopieren, siehe Abbildung 1. Ein Hinweis noch dazu: Wer keine explizite Security benötigt, kann `<useSecurity>true</useSecurity>` in der `/var/lib/jenkins/config.xml` auf `false` setzen.

Im nächsten Schritt (siehe Abbildung 2) werden die Plugins ausgewählt, die wir benutzen möchten. Hier belasse ich es bei den Standard-Plugins, weitere lassen sich später nachinstallieren.

Nach der Erstellung eines Administratorkontos, siehe Abbildung 4, können wir auf die Oberfläche zugreifen und neue Aufträge erstellen (siehe Abbildung 5).

Jenkins in der praktischen Anwendung

Angeregt durch eine Diskussion auf der internen Dante-Mailingliste hat sich in den letzten Monaten eine Gruppe Enthusiasten gefunden, die an einer Einführung zum Thema »LaTeX für Geisteswissenschaftler« arbeitet. Aktuell sieben Mitstreiter haben sich zusammengetan, die verschiedene Themen bearbeiten. Die Zusammenarbeit

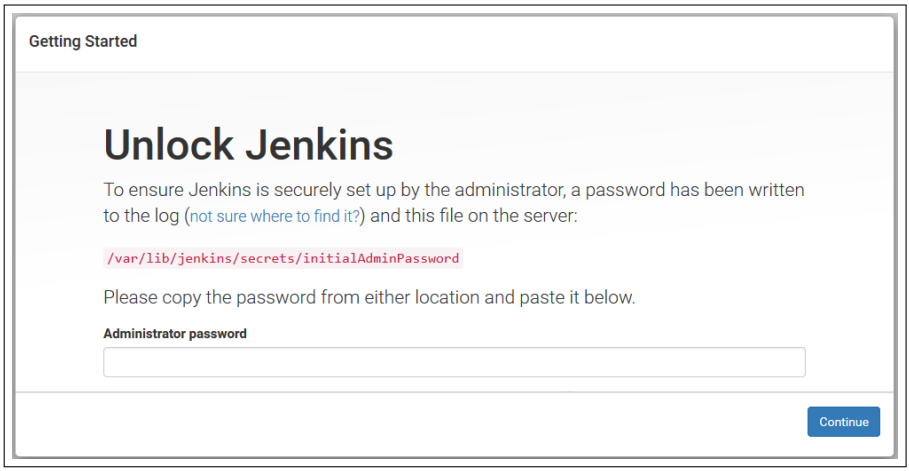

Abb. 1: Festlegung des Initial-Passworts

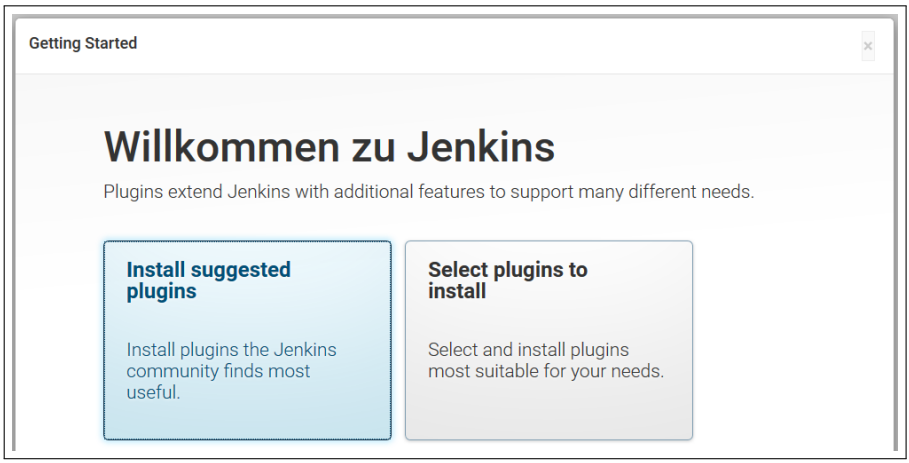

Abb. 2: Auswahl der Plugins

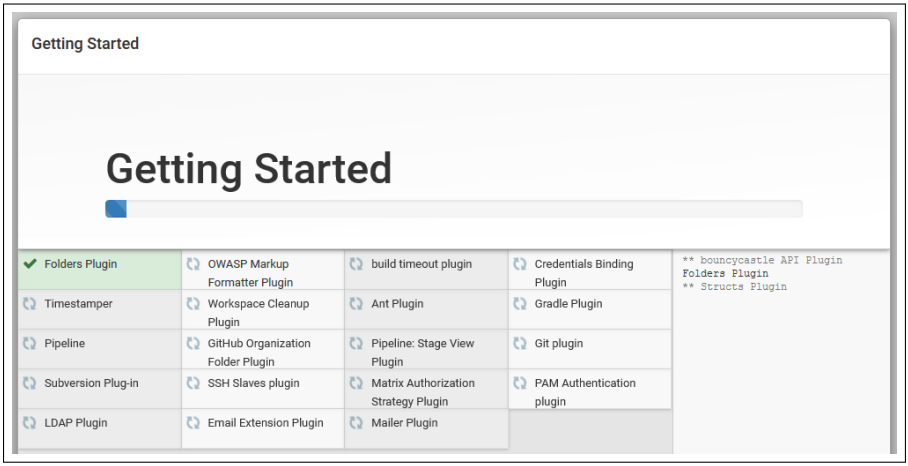

Abb. 3: Auswahl der Plugins

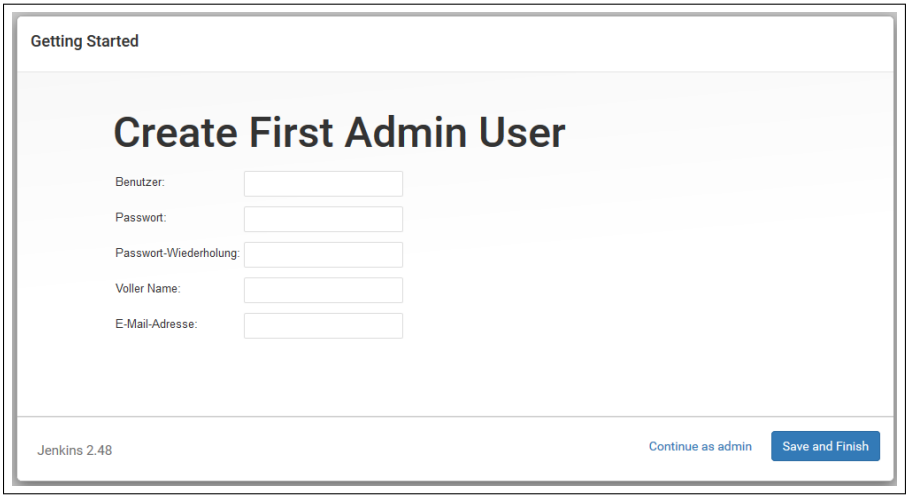

Abb. 4: Erstellung des Administrator-Kontos

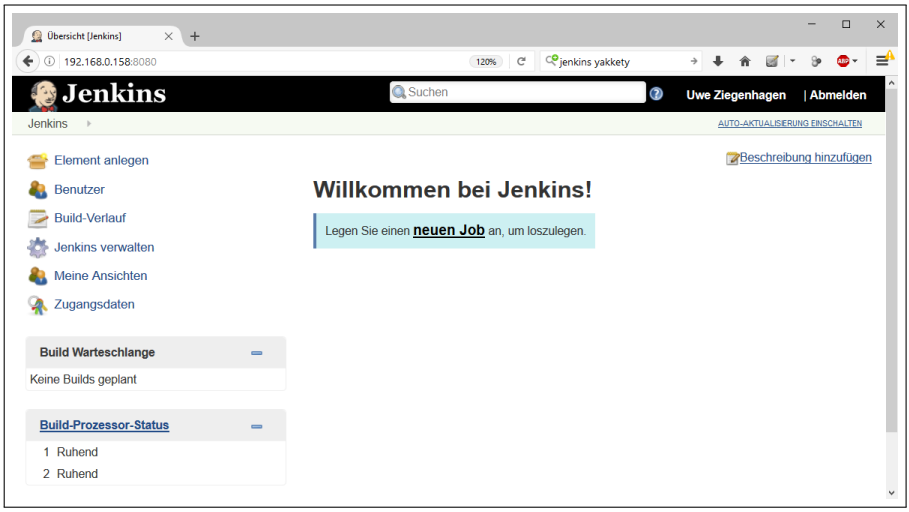

Abb. 5: Die jenkins-Oberfläche

wird über GitHub organisiert, auch die Projektdateien werden auf dieser Plattform (erreichbar unter `https://github.com/thomas-hilarius-meyer/LaTeX-fuer-Geisteswissenschaftler`) gespeichert.

Bei der Zusammenarbeit hat sich gezeigt, dass Tippfehler bei der Eingabe oder unterschiedliche Versionsstände der TeX-Installation dazu führten, dass das Gesamtprojekt nicht mehr kompilierbar war.

Hier bietet sich Jenkins an, um die Kompilierung automatisch durchführen zu lassen. Als Hardware dient ein Intel NUC mit relativ schwachem Celeron N2820, der auf acht Gigabyte RAM zugreifen kann. Für den aktuellen Zweck reicht dies aus, bei komplexeren Projekten dürfte mehr Rechenleistung zu empfehlen sein.

Fazit