Buildserver: Buildbot und Jenkins

Seminar - Effiziente Programmierung

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Dennis Markmann
E-Mail-Adresse:	5markman@informatik.uni-hamburg.de
Matrikelnummer:	6767293
Studiengang:	Software-System-Entwicklung

Betreuer: Christian Hovy

Hamburg, den 09.04.2017

Inhaltsverzeichnis

1 Einleitung

Das Erstellen von Software-Builds ist heutzutage in der Software-Entwicklung ein unerlässlicher Prozess, um ein fertiges Anwendungsprogramm zu erzeugen. Für kleinere Projekte reicht hierfür meist aus, dass der Vorgang für Entwickler durch die IDE möglich ist. Sinnvoller ist meistens jedoch, dass diese Aufgabe an einen sogenannten Buildserver übergeben wird. Damit können Software-Builds vollautomatisch durchgeführt werden ohne, dass die Entwickler sich selbst darum kümmern müssen.

Im Kapitel 2 dieser Ausarbeitung werden knapp die Grundlagen zu Software-Builds erklärt und im Folgenden detaillierter auf die Vorstellung der Funktionen und Vorteile (Kapitel 3) von Buildservern eingegangen. Als praktische Beispiele für diese Prinzipien werden die Anwendungen Jenkins (Kapitel 5) und Buildbot (Kapitel 6) verwendet. Zwischen diesen findet anschließend ein Vergleich ihrer Vor- und Nachteile sowie der entsprechenden Zielgruppen statt (Kapitel 7).

Der Hauptteil der Erklärungen erfolgt jedoch exklusiv am Beispiel der Anwendung Jenkins, da sich viele Gemeinsamkeiten der Tools sonst wiederholen würden.

2 Grundlagen

Im Folgenden werden die Grundlagen vermittelt, die nötig sind um zu verstehen, was ein Buildserver überhaupt tut. Dafür wird zuerst erklärt, wie Software-Builds ganz allgemein ablaufen. Anschließend wird der Begriff des Buildservers selbst erläutert, sowie bereits knapp auf den Kontext selbigen in Software-Projekten eingegangen.

2.1 Software-Builds

Der Ablauf eines Software-Builds ist üblicherweise wie folgt:

Im ersten Schritt wird der vorhandene Code kompiliert und mit den benötigten Bibliotheken verknüpft. Dabei wird der Quellcode von der gewählten Programmiersprache in eine vom Computer ausführbare Form gebracht.
Außerdem wird das Ziel festgelegt, das beim Start der Anwendung ausgeführt werden soll. Wird zum Beispiel das Java Umfeld betrachtet, werden die erzeugten Dateien nun noch gepackt, um die Handhabung durch den Benutzer zu erleichtern. Zum Abschluss werden die zuvor temporär benötigten generierten Dateien nun wieder entfernt [Die].

Neben diesen Hauptfunktionalitäten gibt es allerdings noch weitere Prozesse die beim Erstellen von Software-Builds notwendig sein können. Diese ergeben sich aus den Anforderungen des jeweiligen Software-Projekts. Eine Auflistung populärer Beispiele wäre:

1. Generierung des Quellcodes (noch vor dem Kompilieren)
2. Erstellung der Dokumentation anhand von Code und Kommentaren (z.B. Javadoc)
3. statische Code-Analyse und Metriken (z.B. FindBugs, Testabdeckung)
4. Deployment der erzeugten Anwendung
5. Ausführung von Tests (z.B. JUnit)

[Mat]

2.2 Buildserver

Allgemein lässt sich sagen, dass ein Buildserver ein Werkzeug ist, das es ermöglicht diesen Prozess zum Erzeugen von Software-Builds automatisch durchzuführen. Dabei handelt es sich um Software die auf Servern installiert wird. Meist wird hierfür aus Gründen der Leistung und Ausfallsicherheit mindestens ein eigener Hardware-Server verwendet.

Buildserver sind im Folgenden jeweils als Hardware-Server-Lösung zu verstehen und teilweise auch als CI-Server bezeichnet. CI steht für Continuous Integration, näheres dazu ist im Kapitel 4 zu finden.

In der Abbildung 2.1 ist ein CI-Server mitsamt seinen Schnittstellen innerhalb eines Software-Projekts beispielhaft dargestellt. Der grundsätzliche Ablauf ist hierbei wie folgt: Das Entwicklerteam macht eine Anpassung am Quellcode der Software und commited diese in das verwendete Versionskontrollsystem wie z.B. Git oder Subversion. Der CI-Server prüft stets auf Änderungen und zieht sich anschließend den aktualisierten Code vom Versionskontrollsystem. Daraufhin wird ebenfalls automatisch ein Build durchgeführt sowie die Ergebnisse z.B. auf der Webseite oder per E-Mail dargestellt. Dadurch muss sich das Entwicklerteam lediglich um die Entwicklung des Quellcodes kümmern und bekommt ohne weiteres Eingreifen Benachrichtigungen über den Stand der Builds.

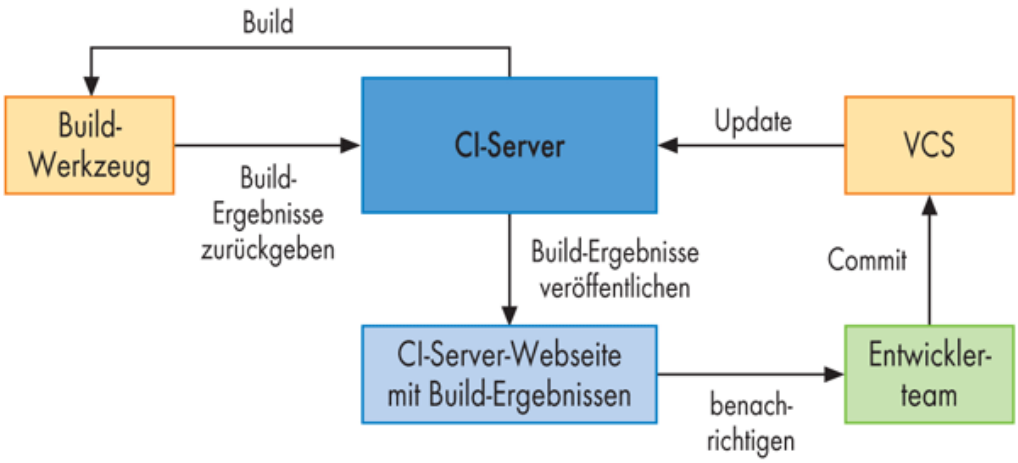

Abbildung 2.1: Buildserver Übersicht
[BF]

3 Funktionsweise

Es gibt drei wesentliche Kategorien von Vorteilen, die durch die Verwendung eines Buildservers im Vergleich zu lokalen Software-Builds entstehen: Reporting, Automatisierung und Zentralisierung. In diesem Kapitel wird die Funktionsweise von Buildservern erklärt und anhand dessen beschrieben wie es dabei zu den jeweiligen Vorteilen kommt.

3.1 Reporting

Die wohl wichtigste Funktion von CI-Servern ist das Reporting. Hierbei geht es darum, dass die Ergebnisse über den Erfolg der Builds automatisch wieder an das Entwickler Team zurückgegeben werden. Dieses Feedback kann auf unterschiedliche Art und Weise erfolgen. Im Jenkins Standard erfolgt eine Benachrichtigung per E-Mail an die Entwickler, die seit dem letzten erfolgreichen Build commited haben. Auch über die Weboberfläche ist der Status der jeweiligen Builds einsehbar. Weitere Möglichkeiten bieten verschiedene Plugins, zu denen mehr in dem Kapitel UI-Erweiterungen zu lesen ist.

Neben der reinen Benachrichtigung über den Erfolg oder Misserfolg können allerdings noch unzählige weitere Daten erhoben werden. Prominente Beispiele hierfür wären die Anzahl der Warnings, Finbugs oder die Testabdeckung. Die Trends über diese Metriken können anschließend auch direkt vom Buildserver ausgewertet und als Statistiken angezeigt werden [Con].

3.2 Automatisierung

Ein weiterer wichtiger Vorteil von Buildservern ist, dass die Builds nicht manuell ausgelöst werden müssen. Dies passiert vollautomatisch und kann auf unterschiedliche Art und Weise stattfinden. Bei der gängigsten Variante wird ein Versionskontrollsystem wie Git oder Subversion verwendet, um Code Änderungen im Entwicklerteam zu teilen. Mit diesen Systemen wiederum kann sich der Buildserver verknüpfen, um stets direkt über Änderungen zu erfahren. Sollte nun ein Commit ausgelöst werden, wird das automatisch vom Buildserver erkannt, der neue Code Stand geladen und ein Build Vorgang angestoßen.

Eine weitere Option ist es, in bestimmten Zeitintervallen neue Versionen zu bauen, was sich besonders für sehr zeitintensive Builds eignet. Ein Verwendungsbeispiel wäre z.B. jede Nacht einen Build auszuführen, der jegliche vorhandenen Tests durchführt und mehrere Stunden den Buildserver blockiert. Somit wäre dieser nicht sinnvoll ausführbar,

während parallel Entwickler neue Versionen bereitstellen und auf ihre Builds warten müssten.

Es gilt also generell zu bedenken, dass Build Vorgänge unterschiedlich lange dauern können. Dies hängt allgemein davon ab, wie umfangreich die zu bauende Anwendung ist, aber ebenso davon, welche zusätzlichen Funktionen ausgeführt werden sollen.
Entsprechend ergibt sich oftmals, dass Builds unterschiedlicher Komplexität parallel ausgeführt werden, um möglichst zeitnah bereits über Ergebnisse zu erfahren. In der Praxis anzufinden ist häufiger folgende Unterteilung:

1. Fast
2. Middle
3. Slow
4. Nightly

Wobei Fast maximal binnen weniger Minuten ein Ergebnis liefern sollte und lediglich eine neue Version baut und entsprechend prüft, ob diese kompilierbar ist.
Middle kann außerdem grundlegende Tests, wie z.B. JUnit ausführen und somit außerdem die Entwickler über Anwendungsfehler benachrichtigen.
Slow berücksichtigt hingegen auch zeitaufwändigere Tests, die z.B. erst einmal viele neue Einträge auf einer Datenbank anlegen müssen.
Nightly läuft, wie zuvor beschrieben, nur einmal jede Nacht und kann zusätzlich Schnittstellen und Anwendungstests beinhalten, die entweder besonders zeitaufwändig sind oder aber einfach nicht besonders häufig getestet werden müssen [Fre].

3.3 Zentralisierung

Der dritte große Bereich der Vorteile von Buildservern ist die Zentralisierung. Hierbei geht es darum, dass lediglich eine einzige Instanz eines Buildservers in einem Unternehmen existieren muss. Diese ist dann im Falle von Jenkins und Buildbot über den Browser aller Mitarbeiter erreichbar. Es ist also nicht jeder Mitarbeiter dazu gezwungen eine eigene Anwendung auf ihrem Rechner zu installieren und konfigurieren, nur um Software zu bauen. Damit verbunden ist es ebenso unwichtig, dass jeder im Detail Fachwissen über die Anwendung haben muss. Es reicht vollkommen aus, dass die Entwickler ein Grundwissen über das Feedback haben, das sie dadurch bekommen. Auch das manuelle Starten von Builds durch Nicht-Entwickler, zum Beispiel für eine Version zum manuellen Testen oder Ausliefern beim Kunden, ist somit unproblematisch möglich.
Ein weiterer durch die Zentralisierung bedingter Vorteil ist, dass die Entwickler ihre Rechenleistung nicht unnötig zum Bauen verschwenden oder gar darauf warten müssen, dass die Builds abgeschlossen sind und in der Zeit nicht arbeiten können.
Ebenfalls positiv ist die Tatsache, dass auf einem Buildserver zwangsweise mit einem sauberen Stand ohne Altlasten oder sonstige lokale Abhängigkeiten gearbeitet wird. Das beugt Aussagen wie: 'Also auf meinem System funktioniert das ganze' vor [Fre].

4 Continuous Integration

Generell geht es beim Thema Continuous Integration darum, dass jegliche Code Änderungen stets automatisch getestet werden, um schnell zu genau den neuen Änderungen Feedback zu erhalten, ob es Probleme gibt oder nicht. Dadurch wird eine schnelle Behebung von Fehlern deutlich einfacher und es entfällt in vielen Fällen aufwändiges Suchen von Fehlerquellen.

Es gibt, wenn man von Continuous Integration spricht, jedoch drei abgegrenzte Begriffe, die den Umfang beschreiben in dem automatisiert wird.

1. Continuous Build
2. Continuous Integration
3. Continuous Delivery

Continuous Build beschreibt hierbei lediglich, dass der Server in der Lage ist, Code aus Repositories zu ziehen und den Code zu kompilieren.
Sobald der Funktionsumfang allerdings darüber hinausgeht und auch Tests durchgeführt werden und entsprechendes Feedback an die Entwickler gegeben werden kann, spricht man bereits von Continuous Integration. Da heutzutage quasi jede Buildserver Anwendung diese Kriterien erfüllt, spricht man hier allgemein auch einfach von CI-Servern.
Continuous Delivery wiederum beschreibt, dass auch die Auslieferung, z.B. bei Kunden, automatisch stattfindet und nicht nur das Bauen der Versionen. Diese Kriterien werden in der Grundversion meist noch nicht von vielen Anwendungen unterstützt, durch die Verwendung von Plugins ist beispielsweise Jenkins aber als Continuous Delivery Server verwendbar [Mil].

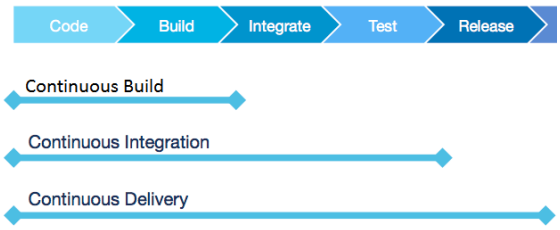

Abbildung 4.1: Vergleich der CI Stufen (angepasst)
[Mil]

5 Jenkins

Jenkins ist die heutzutage wohl populärste Buildserver Anwendung überhaupt. Diese entstand im Jahre 2011 als Fork aus dem Vorgänger Hudson. Geschrieben ist das Tool in Java und entsprechend auch betriebssystemunabhängig ausführbar. Der Hauptanwendungsbereich ist somit auch im Feld der Java-Entwicklung zu sehen, jedoch werden alle gängigen Sprachen theoretisch unterstützt. Besondere Merkmale von Jenkins sind die riesige Plugin Auswahl, sowie die Funktionsfähigkeit ohne großen Konfigurationsaufwand [Kaw].

5.1 Arbeitsweise

Wichtige Eigenschaft zur Arbeitsweise sowohl von Jenkins als auch Buildbot ist, dass der Build-Server jeweils in Master und Slaves unterteilt ist.
Der Master dient der Kommunikation und Steuerung. Er bekommt mit, wenn Änderungen am Code stattfinden, stößt den Versionsbau durch die Slaves an, stellt die fertige Anwendung bereit und kommuniziert die Ergebnisse zurück an die Entwickler.
Die Slaves hingegen sind lediglich dafür da, auf Befehl des Masters die Version zu bauen und zurück zu geben.
Grund für diese Aufteilung sind verschiedene Vorteile. So ist es möglich und üblich, diese Instanzen hardwareseitig auf unterschiedliche Geräte aufzuteilen. Das dient zum einen der Ausfallsicherheit, denn so wird bei einem Build, der ewig dauert oder gar hängt, nicht gleich der ganze Server blockiert. Außerdem wäre es bei entsprechend großen Projekten von der Performance her sehr unrealistisch nur auf ein Gerät zu setzen. Hier wird teilweise enorme Performance benötigt, die nur von vielen Maschinen zusammen aufgebracht werden kann. Ebenso wird durch solch eine Aufteilung die Verwendung von Backupservern deutlich einfacher umsetzbar [BB].

Um einmal ein Realbeispiel aufzuführen:

Yahoo verwendete im Jahre 2013 einen Master, 3 Backup Master und 50 Slaves, um ihre Software zu bauen. Die Hardware des Masters beinhaltete 96GB RAM, 2 x Xeon E5645 2.40GHz, 4.80GT QPI (12 Kerne, 24 Threads). Es wurde damit ein Durchschnitt von 8000 Builds pro Tag erreicht [Pat16].

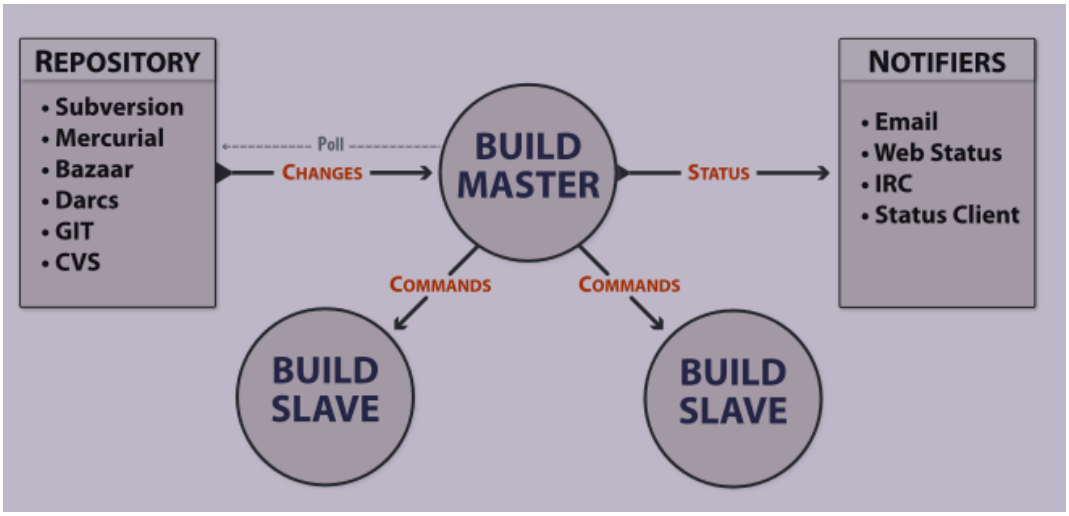

Abbildung 5.1: Buildserver Arbeitsweise
[BB]

5.2 UI-Erweiterungen

Allgemein ist es möglich, den Status der Builds in Jenkins über die bereitgestellte Weboberfläche zu sehen, auch die Benachrichtigung per E-Mail ist ein wichtiges Feature. Doch gibt es durch die Vielzahl an Erweiterungsmöglichkeiten in Jenkins auch weitere sehr interessante Möglichkeiten, auf dem aktuellen Stand zu bleiben.

5.2.1 Build-Ampel

Eine Build-Ampel kann zum Beispiel die Farbe ändern, je nachdem ob der oder die zu prüfenden Builds zuletzt erfolgreich umgesetzt werden konnten oder nicht.

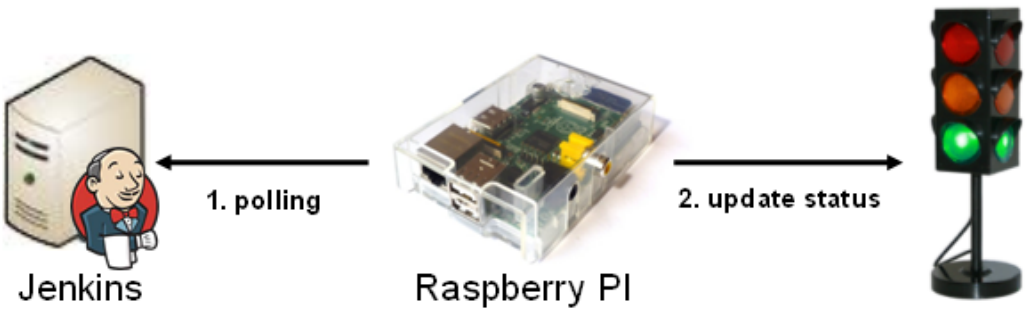

Abbildung 5.2: Build-Ampel
[Bir]

5.2.2 Android / IPhone App

Natürlich ist es in der heutigen Zeit auch möglich, sich per App direkt auf dem Smartphone benachrichtigen zu lassen, sobald ein Build fehlschlägt und mehr.

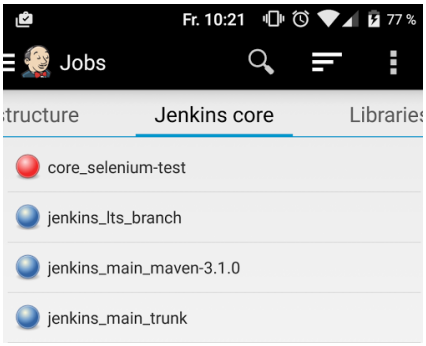

Abbildung 5.3: Android / IPhone App
[Je]

5.2.3 Tech-Dashboard

Ist hingegen die Übersicht über viele Builds auf einmal nötig erweist, sich oft die Verwendung eines großen Tech-Dashboards als Mittel der Wahl. Dieses wird meist zentral im Büro der Entwickler platziert, damit jeder auf einen Blick sieht, wie der Status ist, ohne auf ihrem eigenen Rechner die Anwendung zu wechseln.

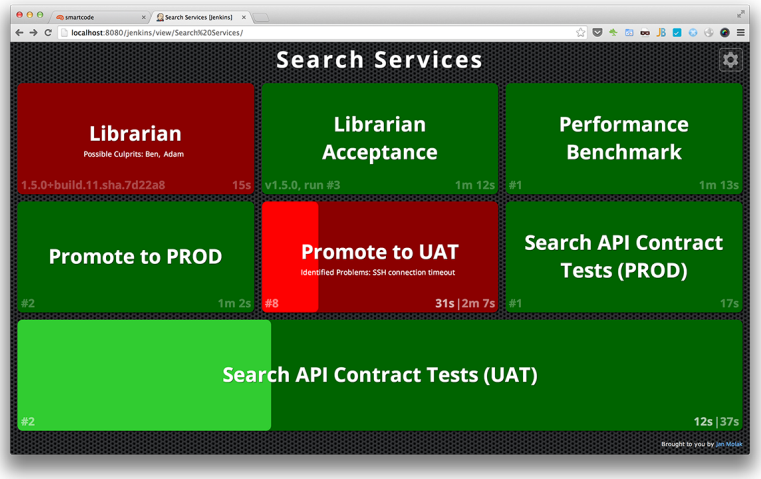

Abbildung 5.4: Tech-Dashboard
[Mol]

6 Buildbot

Buildbot ist ein weiterer wichtiger Vertreter der Build-Server und mit 14 Jahren deutlich länger auf dem Markt, als das noch relativ neue Jenkins. Als Programmiersprache wird hier auf Python gesetzt, wo auch der Schwerpunkt der Verwendung liegt. Prinzipiell ist aber auch hier die Verwendung für viele weitere Sprachen denkbar [War].

6.1 Arbeitsweise

Wichtig ist bei der Verwendung von Buildbot zu bedenken, dass dieser nicht out-of-the-box direkt lauffähig ist. Buildbot benötigt einen Container wie Docker und einiges an manueller Konfiguration, um verwendet werden zu können.
Auch wenn diese Tatsache viele erst einmal abschreckt, hat das natürlich seine Gründe: Buildbot setzt im höchsten Maße darauf, dem Entwickler selbst die Kontrolle zu lassen. Entsprechend sind die Skripte zur Ausführung der Builds selbst zu schreiben, aus dem Java-Bereich beliebte Lösungen wie Ant und Maven werden nicht unterstützt. Dieser Aufwand bietet hingegen die Möglichkeit, die volle Macht der Programmiersprache Python zu nutzen und somit deutlich speziellere Implementationen durchzuführen [Bu].

7 Vergleich

Im Folgenden die meiner Meinung nach wichtigsten Unterschiede der beiden Anwendungen:
Jenkins ist allgemein ein sehr einfach und intuitiv bedienbares Tool, wohingegen Buildbot vorerst etwas Einarbeitung und je nach Projekt einiges an Konfigurationsaufwand mit sich bringt.
Auch in Punkte Übersichtlichkeit kann Jenkins mit seinen modernen Oberflächen punkten, wohingegen Buildbot ein wenig überladen wirkt.
Von der Ausrichtung her ist Jenkins vor allem in der Java-Welt angesiedelt, wohingegen Buildbot für viele Python-Projekte eingesetzt wird. Daraus folgt auch, dass Jenkins von Haus aus Plugins mitbringt, die Ant und Maven Support anbieten. Dieses Feature ist bei Buildbot nicht vorhanden, weshalb Java-Projekte hier eher nicht sinnvoll zu verwenden wären.
Des Weiteren besticht Jenkins mit einer Vielzahl an angebotenen Plugins, wo Buildbot nicht seinen Schwerpunkt legt. Dieses ermöglicht dafür selbst sehr mächtige Skripts zu schreiben und den Funktionsumfang selbst zu bestimmen. Die Python-Unterstützung ist hier sehr von Vorteil, so etwas bietet Jenkins leider nicht an.
Relativ neu und sehr interessant ist auch die Continuous-Delivery-Fähigkeit von Jenkins durch die Verwendung von entsprechenden Plugins.

Alles in allem gibt es also auch heute noch Gründe für Kenner, außerhalb von Java-Projekten, Buildbot zu verwenden. Der Trend für die meisten Fälle geht allerdings schon stark Richtung des moderneren Jenkins [JB].

8 Zusammenfassung

Zusammenfassend lässt sich sagen, dass Buildserver heutzutage ein wichtiger Bestandteil der professionellen Software-Entwicklung sind. Diese ermöglichen durch kontinuierliches Feedback den Code fehlerfreier zu halten und Risiken zu minimieren. Auch die Analyse der Metriken kann dabei helfen, sich im Entwicklerteam stets zu verbessern oder aber z.B. bei der Geschäftsführung als Argument dienen, mehr Ressourcen in Code Optimierung zu stecken.

Jenkins ist der zur Zeit prominenteste Vertreter unter den Buildservern und erfreut sich weiterhin wachsender Beliebtheit. Das ist vor allem seiner Einfachheit und starken Erweiterbarkeit zu verdanken, die durch Community Support stetig weiter anwächst.
Buildbot ist hingegen eher eine Speziallösung die für ausgefallenere Projekte mit speziellen Anforderungen Sinn macht. Das Tool kann enorme Personalisierbarkeit aufweisen, was jedoch mit entsprechendem Aufwand verbunden ist.

Allgemein bleibt zu sagen, dass Software-Entwicklung ohne regelmäßige Testausführung und Kontrollen nicht mehr zeitgemäß ist und einen enormen Risikofaktor darstellt. Aus diesem Grund ist jedem Entwickler, der nicht gerade nur für sich selbst entwickelt, dazu geraten, einen Buildserver zu verwenden, egal auf welches Tool am Ende die Entscheidung fällt.

Literaturverzeichnis

[BB] *Buildbot Basics*. https://buildbot.net/, Abruf: 2017-04-04

[BF] Björn Feustel, Steffen S.: *Continuous Integration in Zeiten agiler Programmierung*. https://www.heise.de/developer/artikel/Continuous-Integration-in-Zeiten-agiler-Programmierung-1427092.html, Abruf: 2017-04-04

[Bir] Birkner, Marcel: *Using a Raspberry PI to control an extreme feedback device*. https://www.raspberrypi.org/forums/viewtopic.php?f=37&t=50857, Abruf: 2017-04-04

[Bu] *Buildbot*. https://buildbot.net/, Abruf: 2017-04-04

[Con] Connolly, Stephen: *Metrics Plugin*. https://wiki.jenkins-ci.org/display/JENKINS/Metrics+Plugin, Abruf: 2017-04-04

[Die] Dietrich, Sebastian: *Erstellungsprozess*. https://de.wikipedia.org/wiki/Erstellungsprozess, Abruf: 2017-04-04

[Fre] French, Kelly S.: *What is the point of a "Build Server"?* https://stackoverflow.com/questions/1099133/what-is-the-point-of-a-build-server, Abruf: 2017-04-04

[JB] *Jenkins vs. Buildbot*. https://stackshare.io/stackups/buildbot-vs-jenkins, Abruf: 2017-04-04

[Je] *Jenkins*. https://play.google.com/store/apps/details?id=com.mobilabsolutions.jenkins.app&hl=de, Abruf: 2017-04-04

[Kaw] Kawaguchi, Kohsuke: *Jenkins*. https://de.wikipedia.org/wiki/Jenkins_(Software), Abruf: 2017-04-04

[Mat] Mattes, Greg: *Programming Definitions: What exactly is 'Building'*. https://stackoverflow.com/questions/1622506/programming-definitions-what-exactly-is-building, Abruf: 2017-04-04

[Mil] Miller, Tim: *Continuous Integration and Delivery in the Cloud: How RightScale Does It*. http://www.rightscale.com/blog/cloud-management-best-practices/continuous-integration-and-delivery-cloud-how-rightscale-does-it, Abruf: 2017-04-04

[Mol] Molak, Jan: *Build Monitor Plugin*. https://wiki.jenkins-ci.org/display/JENKINS/Build+Monitor+Plugin, Abruf: 2017-04-04

[Pat16] Pathania, Nikhil: Learning Continuous Integration with Jenkins. (2016), 05, S. 87

[War] Warner, Brian: *Buildbot*. https://en.wikipedia.org/wiki/Buildbot, Abruf: 2017-04-04

Abbildungsverzeichnis