Titel: Continuous Integration mit Hudson
Autor: Simon Wiest
Jahr: 1. Auflage 2011
Seitenzahl: 301
Preis: 39,90 EUR
ISBN: 978-3-89864-690-1

1 Allgemeine Bemerkungen

Das Buch „Continuous Integration mit Hudson“ richtet sich in erster Linie an Entwickler im IT-Bereich, die ihre Integrationsarbeiten nicht mehr jeweils am Monatsende auf einen Schlag und mit den entsprechenden Problemen duchführen wollen, sondern sich stattdessen mit Hilfe von Hudson so organisieren wollen, dass die Integration von Quellcodeänderungen automatisiert über Nacht oder nötigenfalls auch nach jeder Quellcodeänderung abläuft.

2 Gliederung des Buches

Kapitel 1: Einleitung

Im ersten Kapitel wird der Begriff „Continuous Intergration“ eingeführt.

Der Autor schildert ein häufiges Szenario, welches sich bei solchen Softwareteams abspielt, die das Werkzeug der kontinuierlichen Intergation noch nicht einsetzen. Sie erleben, laut Autor, jedesmal kurz vor der Sowtwareauslieferung die sog. „Integrationshölle“, da sie die Integration aller Programmkomponenten fälschlicherweise erst im allerletzten Moment angehen und dadurch viele Überraschungen erleben, welche wiederum die Auslieferung der Software verzögern.

Als Lösung für die Vermeidung solcher unangenehmen Situationen sieht der Autor die Automatisierung des Intergrationsprozesses an, die nicht nur die Integration übernimmt, sondern daneben auch Tests, Dokumentation und Codeänderungen mitumfassen kann.

Unter Programmen für diese sog. Continuous Integration (CI) gilt Hudson als das meist verwendete CI-System (es wird bereits bei grossen Unternehmen wie eBay, Yahoo, Oracle, SAP und Allianz eingesetzt). Mit seiner Hilfe kann man viele Prozesse in der Softwareentwicklung automatisieren, wie z.B. eben unter anderem regelmässige Integration.

Kapitel 2: CI in 20 Minuten

Im zweiten Kapitel erläutert der Autor im Detail das Konzept der „Continuous Integration“. Wir erfahren, dass es sich hierbei um folgenden Zyklus handelt:

1. Übertragung aller durch die Entwickler überprüften Codeänderungen an ein Versionskontrollsystem (z.B. Subversion)
2. Weitermeldung an den CI-Server bzw. der CI-Server registriert den Vorgang von alleine
3. Veranlassung des Bauens eines Builds auf den Build-Servern durch den CI-Server
4. Rückübertragung der Buildergebnisse (Programme, Testergebnisse oder Dokumentationen) zur Auswertung und Archivierung von den Buildservern auf den CI-Server
5. Benachrichtigung des Entwicklers via Kommunikationskanäle (E-Mail, RSS, Issue-Tracker usw.) über den positiven bzw. negativen Ausgang des Builds. Dann beginnt der CI-Zyklus wieder von vorne, da der Entwickler je nach dessen Ausgang Codeänderung vornehmen muss und dadurch den CI-Prozess erneut in Gang setzt.

Kapitel 3: Welche Vorteile bringt CI

Im dritten Kaiptel werden Vorteile des Continuous Integration aus wirtschaftlicher Sicht aufgezeigt. Hierzu zählen:

Reduzierte Risiken Integrationen gehören zur Normalsituation und sind ein Teil des Arbeitsprozesses im CI-System. Anders ist die Situation bei Projekten, die kein CI einsetzen, die also mit ständiger Ungewissheit konfrontiert sind, was die Länge der für die Integation einzukalkulierende Zeit anbetrifft.

Verbesserte Produktqualität Durch Automatisierung und häufiges Testen werden Fehler früher gemeldet und behoben. Dadurch steigt die Qualität des Endproduktes erheblich.

Den erheblichen Vortiel im CI-System sieht der Autor darin, dass man eine sichtbare Verbesserung bereits nach der Einführung sehen kann, denn man besitzt ständige Verfügbarkeit über ein prinzipiell auslieferbares Produkt.

Eine besondere Rolle beim CI-System kommt der Dokumentation als Qualitätsmerkmal zu, denn es sollte nicht nur danach gefragt werden, was getestet werden soll, sondern auch wie viele Tests durchgeführt werden und wo getestet wird.

Ausserdem benötigt man für ein gut funktionierendes CI-System immer auch einen gut organisierten Informationsfluss.

Kapitel 4: Die CI-Praktiken

Im vierten Kapitel geht der Autor ausführlicher auf die CI-Praktiken ein:

Gemeinsame Codebasis wird normarlerweise in einem Versionskontrollsystem verwaltet und stellt eine der wichtigsten CI-Praktiken dar. Der Autor versucht, die Frage danach zu beantworten, wieviel man an Daten ins Versionskontrollsystem einchecken sollte. In den Build-Skripten müssen lediglich Informationen über Betriebsystem, Compiler oder Datenbanken dokumentiert sein, die zu Beginn des Builds überprüft werden.

Automatisierter Build stellt eine weitere der zentralen Praktiken innerhalb des Continuous Integration-Systems dar. Neben der gemeinsamen Codebasis werden weitere Daten für den Automatisierten Build erforderlich.

Zu den Hauptaufgaben des automatisierten Builds gehören:

- das Kompilieren des Quellcodes
- das Paketieren zu einer Distribution
- Verteilung des Softwareprodukts
- Installation des Softwareprodukts (optional)

Automatisierte Builds lassen sich bei Bedarf sogar mehrmals am Tag durchführen.

Zu den am meisten verwendeten Build-Werkzeugen gehören:

- Shell-Skrpte
- Perl, Ruby, Python, Groovy
- Ant, Maven (beides bei Java-Anwendungen)
- SQL-Client
- eine interaktive Applikation mit grafischer Benutzeroberfläche

Der Autor empfiehlt, bei der Auswahl der Werkzeuge bzw. der Technologien darauf zu achten, dass sie möglichst auf allen benötigten Plattformen verfügbar sind. Ausserdem sollte man, laut Autor, bei der Wahl des zentralen Buildwerkzeugs auf die Möglichkeit achten, beliebige externe Kommandos einbinden zu können. Die Behandlung von externen Kommandos kann man in zwei Bearbeitungsphasen durchführen:

1. durch Aufruf bestehender Skripte werden einzelne Abschnitte des Build-Prozesses gebaut, die in unterschiedlichen Technologien realisiert wurden.
2. mithilfe von Mitteln des zentralen Build-Werkzeuges werden die externen Skripte nachgebildet und somit die Anzahl der verwendeten Technologien nach und nach reduziert

Der Autor empfiehlt, bestimmte Konfigurationen aus dem Build-Skript in Konfigurationsdateien auszulagern, z.B. Benutzerdaten wie Passwörter o.ä.

Ausserdem wird empfohlen, darauf zu achten, dass der Build-Prozess angemessene Fortschrittsmeldungen ausgibt, unter anderem folgende logische Schritte:

- Kompilieren
- Testen
- Inspizieren
- Packen
- Verteilen

Es hat sich eingebürgert, dass ein Build-Prozess mindestens alle 15-30 Sekunden ein Protokoll ausgibt, welches mit einem Zeitstempel versehen ist.

Häufige Integration bedeutet mindestens einmal am Tag folgende Arbeitsschritte eines Entwicklers:

1. Aktualisierung der lokalen Arbeitskopie auf den Stand der gemeinsamen Codebasis
2. Testen mit eigenen Änderungen
3. bei erfolgreichem Testen durch privates Build einchecken ins Versionskontrollsystem

Selbsttestender Build stellt einen vollautomatisierten Build-Prozess dar, welcher die komplette Kette der Produktherstellung bis hin zur Auslieferung des Endprodukts abbildet. Beim vollautomatisierten Build-Vorgang werden folgende Ebenen der Reihe nach geprüft bzw. Tests durchgeführt:

1. Compiler (Quelltexte werden geprüft; Fehlermeldungen und Warnungen)
2. Unit-Tests (bei Java-Projekten werden einzelne Klassen geprüft)
3. Komponententests (Prüfen zusammengehörender Codeblöcke)
4. Systemtests (Prüfen des kompletten Softwareprodukts)
5. Inspektionen (Einsetzen von statistischen Codeanalysen zur Erhebung der Codeabdeckung)

Es hat sich in vielen Teams eingebürgert, die Builds und Tests nach jeder Änderung durchzuführen, was der Autor sehr begrüsst, und hierbei folgende zwei Vorgehensweisen anzuwenden:

Idealfall: geringe Buildzeiten: unter 15 Min für Kompilation und einen schnellen Test

Alternative: Kombination aus einem schnellen Build mit geringem Testumfang und hoher Frequenz mit einem langsameren Build mit vollständigem Testumfang und dafür reduzierter Frequenz

Desweiteren werden drei weitere wichtige Ansätze vorgestellt:

Staffeln des Builds (build staging, pipelining) Bei diesem Ansatz werden Builds in mehrere Stufen wie folgt zerlegt:

- Kompilieren
- Testen
- Paketieren u.a.

Schlägt eine der Stufen fehl, bricht der Build an dieser Stelle ab und gibt den Weg für andere Builds frei.

Modularisierung des Softwareproduktes wird vorgenommen, damit Änderungen nach Bedarf nur an Teilen des Produkts und nicht komplett vorgenommen werden können.

Parallelisierung (distributed builds) Nach der Modularisierung kann die Parallelisierung vorgenommen werden. Das bedeutet, dass Module an mehreren Rechnern gleichzeitig gebaut werden können.

Im weiteren Verlauf des Kapitels geht der Autor kurz auf die Bedeutung und Problematik der Tests der Produktionsumgebung ein, denn es ist nicht immer möglich, die Software in der Umgebung des späteren Produktionsbetriebs zu testen. Aus diesem Grund empfiehlt der Autor, die Ziel-Produktionsumgebung sowohl in Bezug auf die Hardware als auch auf das Datenvolumen so detailiert wie möglich nachzubilden. Die benötigten Systeme können dabei mit Hilfe der Virtualisierung von Rechnern und bei kurzzeitiger Anmietung von Rechenzeit von Cloud-Computing-Anbietern zur Verfügung gestellt werden.

Um den CI-Prozess zu visualisieren, kann man alle Schritte des Erstellungsprozesses mit Hilfe von folgenden Berichten anzeigen lassen, die man auswerten kann:

Aktive Benachrichtigungen sind nur für Fehlermeldungen reserviert, um Struktur und Überblick über die Meldungen zu haben.

Öffentliche Statusanzeige einerseits in Form eines Informationsradiators im Flur, welches den aktuellen Zustand des CI-Systems anzeigt, oder in kleinen Arbeitsgruppen in Form einer Ampel mit drei Signalen (rot-gelb-grün)

Detaillierte Informationen auf Abruf Diese Informationen können über Webbrowser aus dem Archiv des CI-Systems jederzeit abgerufen werden, z.B. von verteilten Teams.

Schliesslich erläutert der Autor, wie eine automatisierte Ausbringung (Deployment) der Anwendung einerseits und der dazugehörigen Daten andererseits in CI-Systemen funktioniert. Die Aktualisierung einer Anwendung wird einfach durch den Austausch mit einem vollständigen, neuen Stand realisiert. Die Aktualisierung der Daten erfolgt hingegen durch Migration über Migrationsskripte, die Datenbankschemata in der Tabelle mit SQL-Anweisungen aktualisieren. Eine bessere Alternative stellt LiquiBase (http:www.liquibase.org) dar, welches Hersteller-neutral ist und mit allen Datenbanksystemen arbeiten kann.

Kapitel 5: Hudson im Überblick

Im fünften Kapitel erfahren wir die Eckpunkte der Entstehungsgeschichte von Hudson: wie Kohsuke Kawaguchi es im Jahr 2006 bei Sun Microsystems in seiner Freizeit erfunden hat und wie im April 2007 die erste Version (1.100) von Hudson veröffentlicht wurde.

Der Autor erläutert zunächst Hudson von der technischen Seite als einen CI-Server, der als Java-Webapplikation realisiert ist und der aus diesem Grund normalerweise innerhalb eines Webcontainers wie Jetty, Tomcat, JBoss, Websphere o.ä. betrieben wird.

Weiter im Kapitel stellt der Autor weitere Systemkomponenten vor, die in der Systemlandschaft des Hudson-Servers mitintegriert sind und mit ihm interagieren. Es folgt ein Zyklus der Arbeitskette der Systemkomponenten mit der Interaktion mit dem CI-Server Hudson:

- Entwicklerrechner
- SCM-System (System-Configuration-Manager) oder Versionskontrolle
- CI-Server - Hudson

Der Autor stellt später das Innenleben des Hudson-Datenmodells vor, welches aus folgenden Haupt-Objekten besteht:

- Hudson - im Zentrum
- Jobs (Projekte wie *Free-Style-Projekt* oder *Maven-2-Projekt* und externe Jobs)
- Builds
- Artefakte
- SCM
- Benutzer
- Plugins
- Queue
- Slaves

Als nächstes erläutert der Autor die Benutzerschnittstellen, über welche man mit Hudson auf drei Möglichkeiten zugreifen kann:

- über eine webbasierte grafische Oberfläche (GUI)
- über eine REST-ähnliche Anwendungsstelle (API) (REST: Representational State Transfer)
- über eine Kommendozeilenanwendung (CLI)

Anschließend zählt der Autor die zehn Hudson-Highlights auf, untermauert mit Argumenten, die dazu dienen sollen, die letzten Zweifel auszuräumen, und sich und andere Menschen endgültig für Hudson zu begeistern:

1. Schnelle Installation
2. Effiziente Konfiguration
3. Unterstützung zahlreicher Build-Werkzeuge
4. Anbindung von Versionsmanagementsystemen
5. Grafische Aufbereitung von Testberichten
6. Vielfalt an Benachrichtigungsarten, auch durch *eXtreme Feedback Devices (XFD*

Zum Schluß vergleicht der Autor eine Handvoll ausgewählte CI-Systeme mit Hudson und stellt deren Architektur und Merkmale vor mit deren sowohl positiven als auch negativen Seiten dar:

- Proprietäre Eigenentwicklungen
- CruiseControl
- ThoughtWorks Cruise
- Atlassian Bamboo
- JetBrains TeamCity

Kapitel 6: Installieren und Einrichten

Im sechsten Kapitel wird erläutert, wie man Hudson installiert und danach eine Systemkonfiguration vornimmt. Folgende Installationsarten werden vorgeführt:

- Schnell-Start-Installation
- Manuelle Installation
- Automatisierte Installation

Dabei erklärt der Autor, welche Systemvoraussetzungen erforderlich sind, um unter anderem Hudson als Webanwendung für die Koordination der Buildprozesse zu betreiben. Ausserdem wird erläutert wie man:

- Hudson startet und stoppt
- Backups vornimmt
- Hudson deinstalliert
- Plugins installiert

Kapitel 7: Hudson im täglichen Einsatz

Im siebten Kapitel stellt der Autor Techniken vor, mit deren Hilfe man Jobs, Testberichte, Benachrichtigungs- und Dokumentationswerkzeuge erstellt. Ausserdem werden Plugins vorgestellt, die für das Bug-Tracking von Bedeutung sind (Atlassian JIRA und Mantis).

Kapitel 8: Hudson für Fortgeschrittene

Im achten Kapitel stellt der Autor Techniken vor, mit deren Hilfe man Builds vor der Ausführung mit Parametern versehen kann (*parametrisierte Builds*), um z.B. ein bestimmtes Projekt mit einer ganz bestimmtes Subversion-Revision zu bauen u.ä. Dabei erläutert der Autor im Detail, wie man Parameter definiert.

Sehr ausführlich geht der Autor auf die Vorstellung von Darstellungsarten und Bearbeitungsmöglichkeiten der benutzerdefinierten Ansichten und des Dashboards ein.

Im weiteren Verlauf des Kapitels erfahren wir über die Möglichkeiten, Multikonfigurationsprojekte und verteilte Builds zu realisieren.

Zum Schluss gibt der Autor einige wertvollte Tipps bzgl. der Absicherung von Hudson. Ausserdem erwähnt er noch weitere nützliche Plugins:

- Claim
- Configuration Slicing
- Build-Promotion
- Description Setter
- Green Balls
- Locale
- Contionuous Integration Game

Kapitel 9: Hudson erweitern

Im neunten Kapitel stellt der Autor an einem praktischen Beispiel die Techniken vor, mit deren Hilfe das Hudson-Plugin-Framework über Extensions erweitert werden kann. Wir erleben hier anhand der Entwicklung eines Beispielplugins hautnah den Entstehungsprozess eines Plugins und lernen dabei die Handhabung von Hudson-Extensions.

Kapitel 10: Aufwand einer CI-Einführung

Im zehnten Kapitel geht der Autor explizit auf Schwierigkeiten technikbezogene Schwierigkeiten bei der Einführung eines CI-Systems ein, nämlich, dass der Erstaufwand relativ gross ist, wenn man vollautomatisierte Builds einsetzen möchte. Der Autor empfiehlt hierfür, Schulungen zu organisieren und zwar für unterschiedliche Zielgruppen:

- Systemadministratoren (technische Schulung: Installation, Konfiguration u.ä.)
- Hauptbenutzer (fachliche Schulung: Anlegen, Konfigurieren, Verwalten)
- Benutzer (für tägliche Benutzung: Abrufen von Berichten, Analysen - nur Entwickler)

Kapitel 11: Tipps zur Einführung von CI

Im elften Kapitel gibt der Autor einige Tipps zur leichteren Einführung von CI. Er schlägt unter anderem vor, mit einem Pilotprojekt zu starten, welches eventuell nur einen Rechner beinhaltet. Dabei sollte man stets folgendes beachten:

- Build-Zeiten müssen kurz gehalten werden
- Codeanalyse muss konstant gesteigert werden
- Messen - nur bei Bedarf und möglichst nur Fehler und Warnungen
- Verfolgen von neuen Plugins
- den Spass nicht vergessen: „Chuck Norris"- Plugin für Zitatensammlung oder Leuchtbären-Plugin als eXtreme Feedback Device (XFD)

Kapitel 12: Fazit und Ausblick

Im zwölften Kapitel fragt sich der Autor, wohin die Reise mit CI in der Zukunft geht. Nach seiner Meinung „verschiebt sich der Fokus der Build-Werkzeuge von der einfachen Ablaufsteuerung immer mehr in Richtung Lebenszyklusmanagement einer Anwendung (*application life-cycle management, ALM)*" (S. 293)

3 Kritik

Als einziger Kritikpunkt gilt dem Fehlen eines Abkürzungsverzeichnisses, denn es ist lästig, jedes Mal im Text die Stellen nachzuschlagen, in denen ein Begriff zum ersten Mal aufgetaucht ist. Insbesondere zu erklären wären folgende Begriffe:

- REST
- POM (die einzelnen Bestandteilen dieser Abkürzung wurden gar nicht erläutert)
- SCV
- PMD
- SCM
- LDAP u.a.

3.1 Tippfehler

Seite 15 (Ablauf der CI) Es fehlt im nachfolgenden Text die schriftliche Beschreibung des Pfeils Nr. 5 von Abb. 2-1 - muss nachgepflegt werden

Seite 23 zweiter Absatz von unten, dritter Aufführungspunkt *...die meisten Projekte den Zeitplan bereits überzogen haben und die Neven bereits blank liegen.*
muss geändert werden in:
Nerven

Seite 92 „Eingebetteter Container (Winstone)“ zweiter Aufführungspunkt *... selbst bei Hunderten von Projekten und mehrend tausend Builds*
muss geändert werden in:
mehreren

Seite 100 „Sicherungen erstellen“ dritter Absatz, erster Punkt *1. Öffnen Sie ein Wartungsfensters mit der Funktion ...*
muss geändert werden in:
Wartungsfenster

Seite 101 „Einzelnes Projekt (EP), zweiter Satz *Diese Sicherung verwenden Sie typischerweise, um ein abgeschlossenes Projekt archivieren*
muss geändert werden in:
zu archivieren

Seite 112 erster Absatz, erster Satz *Wenn der Build-Prozess ohne Probleme abgeschlossen worden konnte, ...*
muss geändert werden in:
abgeschlossen werden konnte

Seite 127 “Zeitgesteuert„ zweiter Absatz, letzter Satz *...erreichen Sie in der Online-Hilfe über das blaue Fragenzeichen-Symbol rechts ...*
muss geändert werden in:
Fragezeichen-Symbol

Seite 130 zweiter Absatz, zweiter Satz *... bei welchem Build-Ergebnis nachfolgende Projekte auslöst werden sollen, ...*
muss geändert werden in:
ausgelöst

Seite 144 erster Absatz, dritter Satz *Dies muss kein Nachteil denn.*
muss geändert werden in:
Dies muss kein Nachteil sein.

Seite 150 “Checkstyle, PMD, FindBugs„ erster Absatz, zweiter Satz *Die drei Werkzeuge decken jeweils einen unterschiedlichen Themenbereiche aber ...*

muss geändert werden in:
decken ... ab

Seite 178 letzter Absatz, zweiter Satz *... Berichte aus Codeinspektionen, die letzten fehlgeschlagener Projekte usw.*
muss geändert werden in:
fehlgeschlagenen

Seite 192 "Verteilte Matrix-Builds„ zweiter Absatz, erster Satz *... mit der Matrix-Builds verteilt auf mehreren Rechnern ausführt werden*
muss geändert werden in:
ausgeführt

Seite 193 zweiter Absatz, letzter Satz *Abschnitt 8-21 zeigt die Umsetzung*
muss geändert werden in:
Abbildung 8-21

Seite 199 zweiter Absatz, erster Satz *(Abschnitt 8.5.6)*
muss geändert werden in:
Abbildung 8-23

Seite 203 letzter Absatz, vorletzter Satz *Die Beispiele in Abbildung 8-1 sollen ...*
muss geändert werden in:
Listing 8-1

Seite 205 erster Absatz, erster Satz *Betrachten wir die einzelnen Schritte in Abbildung 8-2 ...*
muss geändert werden in:
Listing 8-2

Seite 227 "Continuous Integration Game„ erster Absatz, erster Satz *... Für jeden Build werden Punkte an alle Benutzer verteilt, die zu diesem Bild mit Änderungen ...*
muss geändert werden in:
Build

Seite 264 letzter Absatz, erster Satz *Am wichtigsten ist die innere Klasse Artifact-SizeColumnDesriptor: ...*
muss geändert werden in:
ArtifactSizeColumnDescriptor

Seite 278 "Texte der Online-Hilfe„ zweiter Absatz, erster Satz *... sonst werden Sonderzeichen und Umlaute nicht korrekt in Hudson darstellt*
muss geändert werden in:
dargestellt

4 Fazit:

Das Buch “Continuous Integration mit Hudson„ von Simon Wiest bietet einen leichtverständlichen Einstieg in die Welt von Hudson und damit in die professionalisierten Abläufe der Softwareentwicklung. Auch der vom Autor propagierte kontinuierliche Umstieg dürfte einen gestandenen Softwareentwickler nicht davor abschrecken, sich das Hudson-System näher anzuschauen, und dessen Vorzüge bereits in der Erprobungsphase kennenzulernen. Die erstaunliche Flexibilität, die Hudson bereits bei den Konfigurationsmöglichkeiten bietet, etwa um eigene Benutzerschnittstellen zu bestimmen (Kommandozeile, grafische Oberfläche oder über sog. REST), ist ebenfalls ziemlich beeindruckend. An CI führt offenbar heutzutage kein Weg mehr vorbei.