Die perfekte Entwicklungsumgebung - gelebte Agilität mit Hudson, Sonar, Maven & Co.

Mario Herb
esentri software GmbH
Ettlingen

Schlüsselworte:

Agile Entwicklung, Artifactory, Continuous Integration, Deployment Pipeline, Hudson, Java, Maven, Open Source, Software Qualität, Sonar

Einleitung

Die Komplexität für Entwickler in Softwareprojekten steigt nach wie vor an. Dabei stellt die Auswahl und vor allem die Einbindung von Bibliotheken, Frameworks und Technologien in einen durchgängigen Entwicklungsprozess vor allem verteilte und agile Teams vor große Herausforderungen.
Dieser Vortrag wird anhand eines durchgängigen praktischen Beispiels zeigen, wie sich moderne Open Source-Helfer (Subversion, Maven, Hudson, Sonar, Artifactory, ...) nahtlos ineinander integrieren lassen und wie sich dadurch enorme Qualitäts- und Zeitersparniseffekte für agile Teams erzielen lassen.

Folgende Grundphilosophie dient uns dabei als Basis:

Ausgehend von verschiedenen agilen Entwicklungsmethoden hat sich Continuous Integration als fest stehender Begriff etabliert um den Anforderungen, die sich durch das agile Vorgehen ergeben, gerecht zu werden.

„Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

So lautet das erste Prinzip des agilen Manifests, welches schon 2001 von den Mitbegründern der agilen Bewegung veröffentlicht wurde. Alle agilen Vorgehensweisen zielen darauf ab, möglichst früh zu ausführbarer Software zu gelangen und dann über den regelmäßigen Austausch mit dem Kunden in kurzen Abständen die oft auch wechselnden Anforderungen an die zu erstellende Software in optimaler Weise zu erfüllen. Daraus ergeben sich einige Bedingungen, welche an den Release-Prozess gestellt werden, um dieses Vorgehen zu ermöglichen.

Neue Releases müssen

- schnell
- automatisiert
- wiederholbar
- verlässlich
- vorhersagbar
- mit messbarer Qualität

erstellt werden können. Im späteren Projektverlauf spricht man daher auch oft von der sogenannten „Deployment-Pipeline“, welche insbesondere den automatisierten Build-Prozess zusammen mit automatisierten Tests und auch (teil-) automatisierte Deployments auf verschiedene Systeme (Entwicklung, Test, Produktion) umfasst. All dies kann durch die in diesem Vortrag vorgestellten Systeme geleistet werden.

Der Kerngedanke „Continuous Integration“

Die agile Entwicklung wird in optimaler Weise unterstützt, wenn primär sichergestellt ist, dass es jederzeit möglich ist ein lauffähiges Deployment des aktuellen Entwicklungsstands zu erzeugen. Dies ist das Hauptziel der Continuous Integration (CI). Hierbei ist ein CI-Server, der nach jeder Änderung des Quellcodes die komplette Anwendung baut und testet, von zentraler Bedeutung. Dadurch wird sowohl die syntaktische und mittels automatische Unit-Tests, die funktionale Korrektheit der Anwendung geprüft. Darüber hinaus kann durch weitere automatisierte Checks (Sonar) die Qualität des erstellten Quellcodes analysiert und dokumentiert werden.
Es wird bei CI empfohlen, dass jeder Entwickler seine Änderungen so frühzeitig und so häufig wie möglich eincheckt. Man motiviert dabei vor allem große Änderungen über mehrere kleine Schritte umzusetzen und dabei gleichzeitig die Lauffähigkeit des Systems zu erhalten. Dies erfordert eine gewisse Disziplin seitens aller Entwickler, aber die Vorteile liegen auf der Hand:

- Integrationsprobleme werden frühzeitig entdeckt
- Unit-Tests entdecken Fehler durch Änderungen sehr früh
- Man erreicht eine hohe Verfügbarkeit des aktuellsten Systems, was in vielerlei Hinsicht hilfreich ist (Demonstrationen, Tests, Vertriebszwecke, ...)
- Fehler wie z.B. vergessene Dateien beim Check-In werden sofort erkannt
- Man kann jederzeit vor einem Release gut abschätzen, wie weit fortgeschritten der aktuelle Stand ist
- Probleme bei zu weit auseinanderlaufenden Entwicklungssträngen werden vermieden

Voraussetzungen für eine CI-Umgebung

Die Mindestvoraussetzungen für CI bildet neben dem bereits erwähnten CI-Server ein Versionskontrollsystem. Es muss grundsätzlich ein zentrales Repository vorhanden sein, in welches die Änderungen von jedem Entwickler integriert werden können. Des weiteren ist für den Betrieb einer CI-Umgebung ein mittels Skript ausführbarer Build-Mechanismus erforderlich. Diese beiden Komponenten werden in diesem Vortrag konkret durch Subversion und Apache Maven realisiert.
Allerdings besteht die Möglichkeit noch weitere Komponenten zu integrieren, um die Möglichkeiten von CI weiter zu steigern. Deshalb wird in diesem Beispiel die Umgebung durch einen zusätzlichen zentralen Maven Artifact Repositories (Artifactory) und ein zentrales System zur statischen Code-Analyse (Sonar) ergänzt. Die Artifactory verwaltet dabei alle zum Build benötigten wie auch die vom Build erzeugten Artefakte. Sonar ermöglicht durch zahlreiche Metriken und viele unterschiedliche zusätzliche Plugins die kontinuierliche zentrale Überwachung der Qualität des erzeugten Quellcodes.

Ablauf bei CI

Der Ablauf, der nach einem Commit in das zentrale Repository des Versionskontrollsystems angestoßen wird, ist auf Abbildung 1 zu sehen. Der Prozess ist vollständig automatisiert und Entwickler wie auch Projektverantwortliche können anschließend auf vielfältige Art und Weise über die Ergebnisse oder auch Fehlschläge informiert werden. Zusätzlich besteht die Möglichkeit mittels Browser oder über andere Schnittstellen auf Ergebnisse und Reports (über Builds, Tests, Code-Analyse) zuzugreifen und deren zeitliche Entwicklung zu verfolgen.

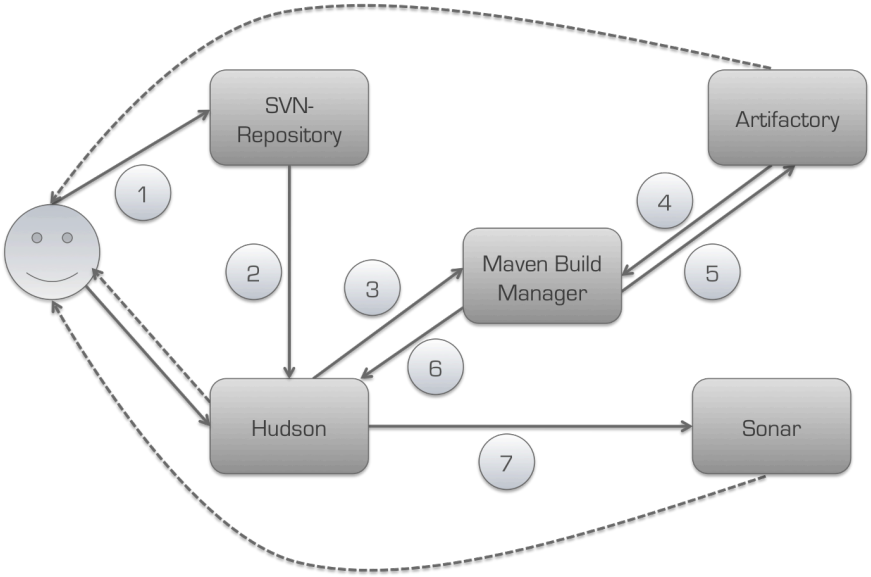

Abbildung 1: Ablauf bei CI

Komponenten einer Java-basierten Open Source CI-Umgebung

Im Folgenden werden nochmals die einzelnen Komponenten einer Java-basierten Open Source-CI Umgebung aufgeführt und insbesondere die in der Praxis bewährten Vorteile dieser Infrastruktur hervorgehoben.

Subversion dient als zentrales Quellcode- und Dokumenten-Repository und stößt initial nach erfolgtem Einchecken den Build durch den CI-Server an. Durch die Versionierung des Quellcodes ist das Löschen von Dateien unproblematisch, da alte Versionen jederzeit wiederhergestellt werden können. Syntaktische Konflikte zu erkennen ist absolute Grundvoraussetzung zur gemeinschaftlichen Arbeit von mehreren Entwicklern am gleichen Projekt. Zusätzlich ist es hiermit immer möglich Entwicklungsstände zu bestimmten Zeitpunkten zu markieren (Tagging) oder auch mehrere nebenläufige Entwicklungsstränge zu betreiben.

Artifactory ist als zentrales Artefakt-Repository für viele gleichzeitige Zugriffe ausgelegt. Es dient als Proxy für verschiedenste öffentliche Repositories und schafft durch lokales Caching eine gewisse Unabhängigkeit von den nicht notwendigerweise immer erreichbaren Repositories im Netz. Eine einfach zu bedienende Web-Oberfläche kann zur Pflege der hinterlegten Artefakte verwendet werden. Weitere Sicherheitsfunktionen wie Benutzer- und Rollenverwaltung und eine LDAP-Anbindung entsprechen gängigen Standards.

Maven ist der State-of-the-art Build Manager für Java und vereinfacht das Dependency Management. Die mögliche Integration vieler Plug-Ins erlaubt zusätzliche Möglichkeiten wie zum Beispiel die. Unterstützung weiterer Sprachen. Nach dem Prinzip „Convention-over-Configuration“ können Maven Skripte, solange der Build sich im Standard befindet, recht einfach und klein gestaltet werden. Die Modularisierung in weitere Sub-Projekte sorgt für Übersichtlichkeit. Über Archetypes lassen sich Maven-Konfigurationen in einer Art Template-Mechanismus standardisieren.

Hudson führt als CI-Server per Maven-Skript, je nach Konfiguration, regelmäßig oder nach dem Einchecken von Änderungen einen Build der entsprechend konfigurierten Projekte durch. Es besteht die Möglichkeit weitere Prozesse nach erfolgreichem Build anzustoßen, wie etwa das Deployment auf

einen Test-Server mit anschließendem automatisierten Anschalt-Test, was sich in der Praxis sehr bewährt hat. Entwickler und Projektverantwortliche können über die Ergebnisse der Hudsonaktivitäten über verschiedene Kommunikationskanäle informiert werden (Jabber, Email, automatische Einträge in Bug-Tracker etc.). Sämtliche Test-Ergebnisse werden historisiert abgelegt und können zur Analyse von Fehlern jederzeit abgerufen werden. Zur Steigerung der Performance lassen sich Distributed Builds mit mehreren Servern konfigurieren. Durch Automatic Build Chaining werden automatisch abhängige übergeordnete Projekte zum Build angestoßen sobald ein Unterprojekt gebaut wurde. File Fingerprints erlauben es leichter herauszufinden, welche Versionen von Jar-Files miteinander funktionieren und welche gegebenenfalls nicht. Neben der Web-Oberfläche gibt es eine Reihe von Plug-Ins und Tools sowie eine REST-Schnittstelle um Hudson auch von extern zu verwalten.

Sonar ermöglicht über verschiedene Plugins (Checkstyle, Findbugs, PMD, ...) per Analyse des Codes eine Möglichkeit die Einhaltung von Coding Rules zu prüfen und deckt mögliche Fehler und Schwachstellen auf. Durch weitere Metriken lassen sich beispielsweise die Testabdeckung, die Kommentierung oder auch die zyklische Komplexität des Quellcodes messen. Sämtliche Ergebnisse werden auf hoher Ebene zusammengefasst und man kann anschließend über mehrere Stufen bis auf Quellcode-Ebene in die fehlerhaften bzw. beanstandeten Code-Teile hinabsteigen.

Fazit

Der Einsatz einer CI-Umgebung in der beschriebenen Form bietet viele Vorteile. Grundsätzlich werden, durch die durch CI motivierte Vorgehensweise der häufigen und frühzeitigen Code-Integration, viele Fehler sehr frühzeitig entdeckt. Mögliche Probleme werden bereits in einem sehr frühen Stadium erkannt oder gar vermieden. Treten bei der Integration von neuen oder geänderten Code-Teilen trotzdem Probleme auf, so führt dieser Ansatz zu Diskussion und Austausch zwischen den Entwicklern, was in einem frühen Stadium im Allgemeinen zu einer qualitativ hochwertigeren und auch kostengünstigeren Lösung führt. Durch die Sicherstellung der Lauffähigkeit des Codes wird die Produktivität gesteigert und man hat durch die permanente Demofähigkeit jederzeit die Möglichkeit zur Abstimmung mit dem Kunden am aktuellen System, wie es durch die agile Vorgehensweisen gefordert wird. Weitere Möglichkeiten zur Produktivitätssteigerung ergeben sich durch die geschickte Auswahl von ersten (Smoke-)Tests, die bei jedem Build durchgeführt werden sollten. Sie überprüfen die grundsätzliche Lauffähigkeit des Systems und geben schnelle Rückmeldung. Mit Hudson ist es möglich, lang andauernden Funktions- oder Integrationstests auszulagern, damit die Entwickler nicht zu lange durch Tests blockiert werden. Zusätzlich hat es sich bewährt durch den Einsatz von weiteren Werkzeugen (Sonar) die Qualität der erzeugten Codes hoch zu halten, indem unter anderem mögliche Fehlerquellen durch die Analyse des Quellcodes direkt erkannt werden, die Quellcodeabdeckung der Tests überprüft wird und die Lesbarkeit wie auch Kommentierung des Codes sichergestellt werden kann.

Kontaktadresse:

Dipl. Inf. Mario Herb
esentri software GmbH
Pforzheimer Straße 132
D-76275 Ettlingen

Telefon: +49 (0) 7243-354 90 0
Fax: +49 (0) 7243-354 90 99
E-Mail mario.herb@esentri.com
Internet: www.esentri.com