Continuous Integration in der Datenbankentwicklung

Dominic Weiser
ISTEC Industrielle Software-Technik GmbH
Nobelstraße 12, 76275 Ettlingen

Schlüsselworte
Continous Integration, Oracle DB, Git, Jenkins, utPlSql,

Einleitung

Continuous Integration (CI) ist im Java-Umfeld schon seit einigen Jahren ein Begriff. Dennoch konnte sich das Thema in der Datenbank-Entwicklung noch nicht wirklich etablieren. Viel zu oft zeigt sich in der Praxis, dass Entwicklerteams noch durch harte Datei-Sperren zusammen an Projekten arbeiten. Obwohl diese Arbeitsweise nicht mehr "up to date" ist, wird sie in vielen Büros immer noch praktiziert, auch wenn die Nachteile ganz klar überwiegen. So ist es beispielsweise sehr häufig der Fall, dass sich die Entwickler gegenseitig behindern. Im einfachsten Fall, weil ein Entwickler eine Sperre auf eine Datei hält, welche ein Kollege bearbeiten möchte. Im schlimmeren Fall werden Änderungen eingespielt, welche die Kompilierbarkeit der Sourcen behindern.

Continuous Integration

Das Thema Continuous Integration wurde bereits mit aufkommenden Popularität von Exteme-Programming immer bekannter und weit verbreiteter. Obwohl die meisten Entwickler mittlerweile wissen was mit CI gemeint ist, kennen viele noch nicht die genaue Definition:

“Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.” (1.4.2006 Martin Fowler [1])

Technologien

Übertragen wir dieses Manifest des CI auf die Entwicklung in Datenbanken-Segment ergeben sich daraus einige technologische Anforderungen. So ergibt sich aus „...each person integrates at least daily...“ die Anforderung an ein Source Verwaltung System. Rein theoretisch würde es reichen die Quelldateien zumindest täglich an eine zentrale Stelle zu kopieren, dennoch ermöglichen Verwaltungssystem zusätzliche Funktionalität. Im nächsten Schritt wird von „...verified by an automated build...“. Hier ist die Anforderung schon konkreter definiert. Durch automatisierte Builds müssen die Quellen überprüft werden. Somit muss ein Build System eingesetzt werden. Darüber hinaus müssen die Funktionalitäten („...including test...“) ab getestet werden.

Source Verwaltung

Im Allgemeinen dienen Source Verwaltung Systeme dazu Änderungen an Dateien zu erfassen und für den Anwender nachvollziehbar zu machen. Wenn man an dieser Stelle in den Markt hört gibt es unter Software Entwicklern nur noch eine Wahl: Git. Jeder spricht davon und jeder setzt es ein. Wirklich alle? Nein, denn es gibt noch weit verbreitete andere Source-Verwaltungssysteme wie CVS, Subversion (SVN) oder andere. Jedoch für welches System soll man sich jetzt entscheiden oder sollte das alte System vielleicht migriert werden? Um diese Frage zu beantworten muss man sich selber eine

die Frage stellen, möchte ich eine zentrale oder eine verteilte Verwaltung? Denn genau darin liegen die gravierenden Unterschiede. Bei CVS oder SVN liegen die zurückgespielten Dateien an einem Ort. Dies kann sehr hilfreich sein, wenn Entwickelter Code in genau einem Projekt oder Produkt zum Einsatz kommt. Im Gegensatz dazu gibt es zum Beispiel Git, welches verteilt verwaltet wird. So ist es mögliche einzelne Programmteile aus unterschiedlichen Repositories lokal zusammenzuführen und weiter zu entwickeln. Anschließend lassen sich die lokalen Änderungen wieder in die verteilten Repositories zurückspielen. Unterschiede werden hierbei meistens vom System selber wieder zusammengeführt.

Jenkins/Hudson

Wenn wir uns nun wieder auf das Manifest von CI berufe fehlt als nächster Schritt der Automatisierte Build „...automated built...“. Auch diese Build Systeme sind im Java Umfeld seit ca. 2008 ein fester Bestandteil der Entwicklungsumgebung. Die prominentesten Vertreter sind die Webanwendungen hier Hudson und sein Fork Jenkins. Beide Systeme dienen der automischen Überwachung der im Repository befindlichen Sourcen. So gibt es üblicherweise die Quellen durch verschiedene Trigger gesteuert zu Analysieren. Zum einen wäre dies zum Commit (Push) Zeitpunkt. Dies hier geschieht oft eine schneller Build, so dass hier keine Tests durchgeführt werden. Ein weiterer Trigger ist zeitgesteuert. So lassen sich beispielsweise nachts oder am Wochenende komplette Testpläne durchführen.

Unit Tests

Verfolgt man den Ansatz seine Software mittels Tests abzusichern, muss sich zuerst die Entscheidung treffen welche Vorgehen dabei angewendet wird. Ein nativer Ansatz wäre, jede Methode einzeln zu testen. Dieser Versuch würde jedoch sehr schnell dazu führen, dass jede geschriebene Methode durch enorm viel mehr an Test-Methoden ergänzt werden muss. Das würde den Entwicklungsaufwand nur unnötig in die Höhe treiben. Aus diesem Grund haben sich die Funktionalen Tests (Unit für Funktionseinheit) durchgesetzt. Es müssen also nur noch die Funktionalitäten ab getestet werden. Dies kann auch dazu führen, dass ein Test sich auf mehrere Schichten (Frontend, Middleware und Backend) auswirkt.

Setzen wir nun die Scheuklappen wieder auf und betrachten nur unsere Datenbank bekommen wir durch die bis hierher definierte Infrastruktur ganz klare Anforderungen an unsere Tests. Nehmen wir zusätzlich noch an, dass wir kein Budget für Lizensierte Software haben, stoßen wir relativ schnell auf das utPLSQL Projekt. Dieses Projekt wurde ursprünglich durch Steven Feuerstein entwickelt. Zwischenzeitlich wird es seit 2017 durch ein Team auf GitHub als OpenSource Projekt weiterentwickelt.

How to

Nachdem in den vorangegangenen Abschnitten die Grundlagen einer Continuous Integration Umgebung analysiert wurden, befassen wir uns nun mit der konkreten Aufsetzung einer solchen Infrastruktur. Dazu sind die folgenden Schritte:

1. Aufsetzten der Source-Verwaltung (GIT)
    a. Ein Account bei GitHub anlegen
    b. Einloggen
    c. Neues Repository anlegen oder ein vorhandenes Branchen
2. Anbinden der Datenbank Quell Dateien an das Source-Verwaltung (SqlDeveloper)
    a. Team -> Git -> Clone...
    b. Name für den Klon vergeben, Ziel URL eintragen und die Zugangsdaten.
    c. Branch auswählen und Klonen

3. Aufsetzen des Jenkins Servers
    a. .war Datei von Jenkins Herunterladen
    b. Jenkins auf einem Server Deployen
    c. Installations-Prozess durchführen
4. Implementieren der Unit Tests
    a. Testpackage anlegen
    b. Test Funktionen hinzufügen
5. Einrichten der Jenkins Jobs.
    a. Jenkins aufrufen
    b. Element anlegen
    c. „Free Style“ Softwareproject bauen
    d. Source-Code-Management hinzufügen
    e. Build-Auslöser (zeitgesteuert) z.B. H 2 * * * für jede Nacht in der Stunde 2
    f. Build verfahren Maven Goal z.B. mvn test
    g. Post-Builds-Aktion z.B. E-Mail an alle, die den Build fehlschlagen liesen.

Continuous Delivery

Bis jetzt haben wir uns eine Funktionierende CI Umgebung geschaffen. Welche Schritte werden nun noch zur Continuous Delivery (CD) benötigt? Nicht viele, denn eigentlich muss jetzt nur noch der Server des Kunden bzw. der Produktiv-Server an unser Jenkins angebunden werden. Dazu eignet sich am besten ein extra Job. Dieser wird entweder Manuell ausgeführt, wenn alle Test-Instanzen durchlaufen wurden oder automatisch wenn alle Tests erfolgreich durchlaufen wurden. Dazu lässt sich dieser Job durch einen Post-Build Schritt triggern. Von einem direkten Deployment ohne Testabdeckung ist in jedem Fall abzuraten. Denn entgegen der weitverbreiteten Meinung, bedeutet CD nicht, dass jeder commit automatisch ausgeliefert wird. Ziel ist viele kleinere Auslieferungen zu erhalten um somit lange Inbetriebnahmen zu vermeiden.

Fazit

Continuous Integration ist bestimmt schon lange kein neues Thema mehr. Dennoch findet es in viel zu wenigen Projekten Einsatz. Das ist sehr schade, denn wer sich ein wenig mit der Thematik auseinander setzt wird sehr schnell feststellen, dass das Aufsetzen eines CI-Systems in sehr wenigen Schritten umzusetzen ist. Danach ist das System mit ein wenig Selbstdisziplin (gerade in kleinen Projekten) mit wenig Wartungsaufwand verbunden.

Quellen

[1] https://www.martinfowler.com/articles/continuousIntegration.html

Kontaktadresse:
Dominic Weiser
ISTEC Industrielle Software-Technik GmbH
Nobelstraße 12
D-76275 Ettlingen

Telefon:	+49 (0) 7243-7005 164
Fax:	+49 (0) 7243-7005 199
E-Mail	Dominic.Weiser@istec.de
Internet:	www.istec.de