Continuous Integration für die Oracle-Datenbank und Apex

Peter Busch und Dominic Ketteltasche, MT AG

Immer kürzer werdende Intervalle von Releases und Hotfix-Bearbeitung machen eine Automatisierung für das Deployment von Datenbankobjekten und Anwendungen erforderlich. Gesicherte und geprüfte Abläufe sind dafür unerlässlich.

Um die entsprechenden Auslieferungen und Sicherheitsmechanismen effektiv nutzen zu können, werden unterschiedliche Werkzeuge benutzt. Continuous Integration (CI) ist der Oberbegriff, unter dem der gesamte Ablauf wiederkehrend abgehandelt wird. Eine Automatisierung der Prozesse führt zu einer Verkürzung der Zeitspanne zwischen den einzelnen Prozess-Schritten, aber auch zu Verbesserungen bei der Laufzeit der aufgerufenen Skripte. Außerdem liefert die Automatisierung der Prozesse eine einheitliche, projektweite Verfahrensweise für die unterschiedlichen Umgebungen.

Dieser Artikel stellt das bei der MT AG zum Einsatz kommende Continuous-Integration-Verfahren vor (siehe *Abbildung 1* in wiederkehrender Ausführung) – am Beispiel eines Projekts, in dem schon in der Entwicklungsphase fertige Teile an den Kunden geliefert wurden. Das Einspielen der Datenbankobjekte sowie der Applikation musste so einfach wie möglich erfolgen. Die Lösung dafür war CI. Anhand des

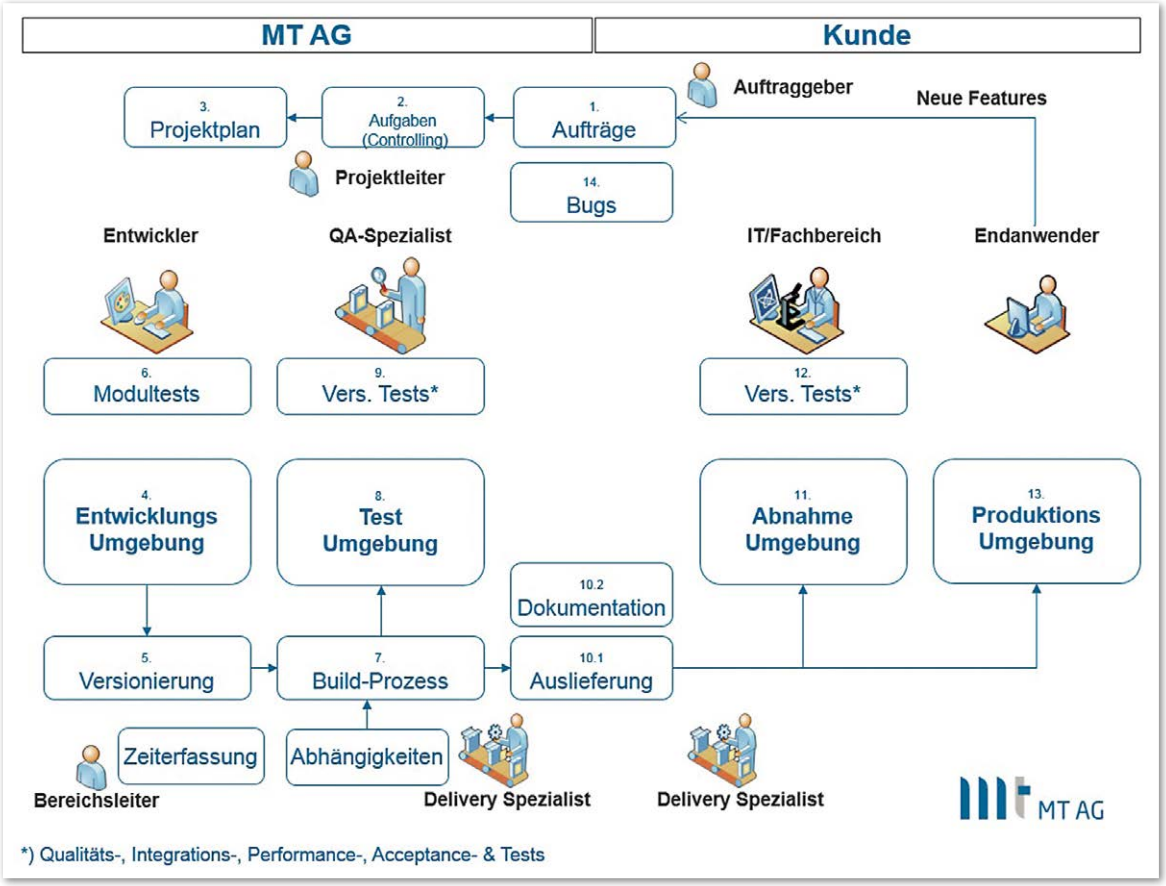

Abbildung 1: Referenz-Architektur für die Software-Entwicklung bei der MT AG

Projekts sind nachfolgend die verschiedenen Elemente aufgezeigt.

Continuous Integration in der Praxis

Als Tools sind TortoiseSVN für die Versionierung und Bug Tracker von Jira im Einsatz. Das Projekt basiert auf Windows-Server und Oracle-Datenbanken. Die Skriptverarbeitung erfolgt durch Batch-Prozesse, die das webbasierte System Hudson im Hause startet und überwacht. Das aktuelle Release wird dem Kunden jeweils als ZIP-Datei zur Verfügung gestellt. Darin sind alle Skripte enthalten, die für die Auslieferung notwendig sind.

In dieser kurzen Beschreibung stellt sich direkt eine Besonderheit und auch Herausforderung dieses Projekts dar. Die Entwicklung erfolgt in einer völlig anderen Umgebung (inhouse MT AG) als der Deployment-Umgebung (Kunde).

Die Architektur stellt sich wie folgt dar: Die Test- und Entwicklungsumgebung für die Release-Arbeiten und die Hotfix-Bearbeitung sind auf den hauseigenen Servern der MT AG installiert. Die Abnahme- und Produktionsumgebung ist beim Kunden vor Ort. Eine Umgebung, die der Produktion entspricht, ist bei der MT AG eingerichtet. Über diese kann dem Kunden eine aktuelle Version des Entwicklungsstands zur Verfügung gestellt werden.

Anforderungen des Kunden werden in dem Tracking-Tool Jira aufgenommen und an die entsprechenden Entwickler weitergereicht. Es wird der Stand der Arbeiten eingetragen und bei Bedarf der Status entsprechend gewechselt. Dort können auch die verbliebenen und verbrauchten Zeiten angezeigt werden.

Für jedes Datenbankobjekt wird ein Skript erzeugt. DML-Skripte und Strukturänderungen (DDL-Skripte) können fachlich und objektabhängig zusammengefasst sein. Die Skripte sind in eine feste Verzeichnisstruktur unterteilt. Alles, was erzeugt und ausgeliefert wird, unterliegt einer Versionierung. Die geänderten Skripte werden in ein Release-Verzeichnis eingebunden. Zusätzlich erfolgt noch die Aufnahme der Skriptnamen mit dem entsprechenden Release in eine Deployment-Anwendung. Über diese sind fachliche Abhängigkeiten eingebbar. Weitere Abhängigkeiten entwickeln sich durch die vorgegebene Verzeichnisstruktur *(siehe Abbildung 2)*.

Aufgrund dieser sich daraus ergebenden Reihenfolge sind die Skripte in sogenannten „Ausführungsskripten“ zusammengefasst. Im Installationsablauf wird bei einem Wechsel der Ausführungsskripte ein Re-Compile aller invaliden Objekte in den betroffenen Schemata ausgeführt.

Das fertige Release

Um die Vollständigkeit der Skripte mit größtmöglicher Sicherheit zu gewährleisten, erfolgt eine Vollständigkeitsprüfung über den Vergleich der eingetragenen Skripte der Deployment-Anwendung und der getaggten Skripte im Versionierungs-Tool. Um die umgesetzten Anforderungen vor einer endgültigen Auslieferung zu prüfen, werden die Skripte in ein Test-System eingespielt. Hier erfolgt ein zweiter Test durch QS-Spezialisten (Vier-Augen-Prinzip). Sollten wider Erwarten noch Skripte für die Auslieferung fehlen, würde es an dieser Stelle auffallen. Um sicherzustellen, dass eine Auslieferung fehlerfrei erfolgt, kann das Zielsystem in der Testumgebung auf eine erforderliche Vorversion geladen werden.

Sofern ein Release ansteht, werden zwei Phasen ausgeführt. Phase 1: Die Entwicklung eines Release ist abgeschlossen, alle Skripte sind erstellt und mit dem Versions-Label getaggt. Außerdem sind die Skripte in die Deployment-Anwendung eingetragen. Der angestoßene Job von Phase1 durchläuft folgende Prüfungen:

- Stimmen die Anzahl und das Format der übergebenen Parameter?
- Sind mit den Zugangsdaten alle erforderlichen Systeme erreichbar?
- Sind die vorgegebenen Verzeichnisse und die Batch-Skripte für das Deployment vorhanden?

Für den Fall, dass die Prüfungen erfolgreich sind, wird ein Ausführungsskript mit den Skriptnamen der eingetragenen Skripte der Deployment-Anwendung erstellt. Die aufgeführten Skripte müssen, wie bereits beschrieben, im Release-Verzeichnis enthalten sein und umgekehrt. Die Abhängigkeit der Skripte wird durch die Ordnerstruktur festgelegt – Datenbank-Schema, DDL, DML, PL/SQL, Apex-Seiten-Objekte (Views, Packages etc.). Innerhalb dieser Verzeichnisstruktur kann unabhängig von der Ordnerstruktur eine zusätzliche Abhängigkeit über die Eingabe innerhalb der Deployment-Umgebung gebildet werden.

Bevor das Release an den Kunden ausgeliefert wird, müssen die Änderungen im Testsystem noch Prüfungen durchlaufen. Das Einspielen in das Testsystem dient dabei ebenfalls als Testlauf für das Deployment beim Kunden. Anschließend wird der Job von Phase 2 angestoßen. Nach dem Aufruf werden analog zu Phase 1 die dort aufgelisteten Prüfungen durchlaufen. Ab hier wird der eigentliche Prozess ausgeführt.

- Der Start des Release wird in eine LOG-Tabelle eingetragen.
- Der derzeitige Release-Stand der Datenbank muss der Vorversion entsprechen.
- Die Skripte des Release werden aufgerufen.
- Die einzelnen Skripte werden in der LOG-Tabelle protokolliert.
- Es werden drei Re-Compiles durchgeführt, um fehlerhafte Datenbank-Objekte zu entfernen. Drei reichen der Erfahrung nach aus, um durch Abhängigkeiten entstandene invalide Objekte valide zu bekommen.

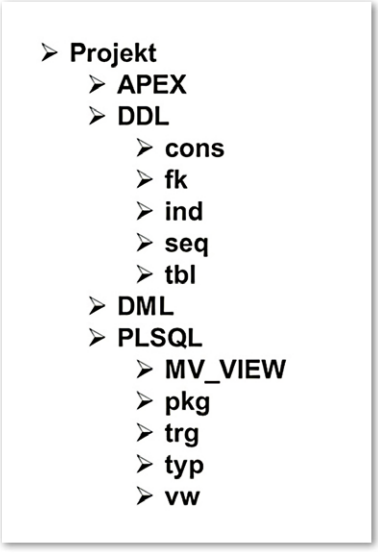

Abbildung 2: Verzeichnisstruktur in Subversion

- Es findet eine Abschlussprüfung statt.
- Der neue Release-Stand wird in die Steuerungstabelle eingetragen.
- Die LOG-Dateien werden in ein zentrales Verzeichnis abgestellt.

Jedes Batch-Skript der Phase 2 erzeugt eine LOG-Datei. Diese Dateien werden am Ende des Laufs auf Fehler durchsucht und die Fehlertexte sowie die Logfile-Namen gesammelt ausgegeben. Gegebenenfalls erfolgt auch ein Abbruch der Phase 2.

Automatisierung

Phase 1 und Phase 2 finden ohne Mitwirken des Kunden statt. Um den Deployment-Prozess gleichzuhalten, werden über Hudson immer nur Batch-Prozesse gestartet, so wie es der Kunde auch durchführen muss. Es ist allerdings möglich, über Parameter zum Beispiel die Release-Version oder die Zielumgebung vorzugeben.

Da die Versionierung unabhängig von der des Kunden ist, besteht die Möglichkeit, notwendige SVN-Arbeiten zu automatisieren. So werden die Tag-Verzeichnisse für die einzelnen Releases per Skript angelegt, vor dem Start der Phase 1 das Verzeichnis aktualisiert und die in Phase 2 erzeugten LOG-Dateien zentral ins SVN eingecheckt.

Aktuell gibt es noch Teile des Deployments, die bislang nicht in das CI mit aufgenommen wurden. In erster Linie sind es JavaScript-Dateien, die noch manuell auf die einzelnen Server zu übertragen sind. Im Prinzip betrifft es alle Objekte, die keine DB-Objekte sind oder nicht zur Apex-Anwendung gehören. Es gehört zu den zukünftigen Zielen, diese ebenfalls mit in den automatisierten Prozess aufzunehmen.

Nach dem Abschluss der Entwicklungsarbeiten wird das Release dem Kunden ausgeliefert. Hierfür fasst ein Batchprozess alle erforderlichen Skripte in einem ZIP-File zusammen, das dem Kunden zur Verfügung gestellt wird. Der Kunde entpackt das ZIP-File und führt den Prozess der Phase 2 aus. In dem ZIP-File sind alle notwendigen Account-Dateien für die Umgebungen des Kunden enthalten. Es ist somit nur noch die Phase 2 mit der entsprechenden Account-Datei zu starten. Aufgrund der Tatsache, dass nur Batch-Skripte verwendet werden, findet beim Kunden der gleiche Prozess wie bei uns im Hause statt. Potenziell entstehende Fehler beim Release können somit nicht auf den übergebenen Skripten beruhen.

Apex-Applikation

Nach Fertigstellung der Apex-Applikation auf der Entwicklungsumgebung erfolgt der Export der Applikation mithilfe des Java-API ApexExport, das in der Apex-Installation enthalten ist. Es stellt die Applikation als SQL-Skript in das File-System.

Dieser Prozess wird über ein Batchskript ausgeführt und wie bei den Datenbank-Objekten über Hudson angestoßen.

Für den Import der Applikation in das Testsystem werden die Parameter „WORKSPACE_ID“, „APPLICATION_ID“, „SCHEMA“, „APPLICATION_ALIAS“ und der „OFFSET“ entsprechend gesetzt. Über SQL*Plus erfolgt dann die Ausführung des SQL-Files der Applikation. Im Nachgang erfolgen das „UNAVAILABLE“-Setzen der veralteten Applikationen und das Setzen der aktuellen Version auf „AVAILABLE“. Somit ist ein Start der Applikation mit dem gleichen Link immer dann möglich, wenn die „APPLICATION_ID“ oder der „APPLICATION_ALIAS“ beibehalten wird.

Da das Export-Skript der Anwendung nicht verändert wird, ist ein Einspielen mit der Import-Funktion des Application Builder in andere Umgebungen möglich. Sobald die Applikation bereit zur Auslieferung ist, wird das SQL-File ins SVN eingecheckt, entsprechend der Version getaggt und in das ZIP-File der Datenbank-Skripte mit aufgenommen.

Die Vorteile

Durch die Anforderung des Kunden, während der laufenden Release-Arbeiten dringende Korrekturen (Hotfix) an die Produktion auszuliefern, sind im Laufe des Projekts zwei weitere Umgebungen bei der MT AG notwendig geworden. Es war ebenfalls erforderlich, die Datenbank-Schemata per Data Pump und die Skripte der Releases in die neuen Umgebungen einzuspielen. An dieser Stelle ist die Steuerung über Hudson sehr hilfreich gewesen. Über Variablen und Auswahlboxen können die einzelnen Umgebungen bedient werden. Ein zentrales Verzeichnis, das von allen Servern erreicht werden kann, dient als Drehscheibe für alle Daten, bei denen keine Versionierung notwendig ist.

Mittlerweile werden fünf Umgebungen für das Projekt vorgehalten, bei denen es mittels Hudson möglich ist, die Applikationen von jeder Umgebung aus zu exportieren und in jede zu importieren. Daraus ergeben sich die folgenden Vorteile. Innerhalb der eigenen Umgebungen sind die Autoren variabler und können schneller auf Kundenwünsche reagieren. Wenn zum Beispiel ein Hotfix neben den normalen Releases notwendig ist, kann ohne großen Aufwand eine aktuelle Umgebung aufgebaut werden, die auch dem Produktionsstand entspricht. Die Laufzeiten der Skripte haben sich durch den Aufruf über Hudson und durch die Optimierungen innerhalb des CI deutlich reduziert.

Da der Kunde für seine Installation die gleichen Skripte verwendet wie wir, profitiert er in Bezug auf Sicherheit von den laufenden Änderungen. So kommen ihm beispielsweise Fehlerroutinen, die im Hause der MT AG entwickelt werden, direkt zugute.

Im Bug Tracker Jira werden alle gemeldeten Änderungen eingetragen und mit der Release-Nummer gekennzeichnet. Über seine Auswertungsmöglichkeiten ist der Stand der Release-Arbeiten schnell abzulesen. Die eingetragenen Skripte in der Deployment-Anwendung sind mit den entsprechenden Jira-Nummern versehen. So ist zu erkennen, welche Skripte für einen Jira-Eintrag erstellt wurden und wer für diese Skripte verantwortlich ist.

Im Laufe des Projekts musste die Vollauslieferung auf ein inkrementelles Verfahren umgestellt werden. Dieses Verfahren stellte eine besondere Herausforderung dar, da die erzeugten Skripte von da ab Restart-fähig sein mussten. So war seitdem bei DDL-Skripten zu überprüfen, ob im Repository die Änderungen bereits enthalten sind. Auf diese Weise ließen sich ORA-Fehler vermeiden, die ein fehlerhaftes Ende der Auslieferung bewirken würden. Bei DML-Skripten kam die Prüfung hinzu, ob die Änderungen im Datenbankschema schon vorgenommen wurden. Für Sonderfälle musste ein Exception-Handling mit dem Abfangen des Fehlers eingebaut werden.

Fazit und Ausblick

Es ist festzuhalten, dass bereits viele Optimierungen geschehen sind, sich CI aber trotzdem immer noch im Wandel befindet, sei es durch eigene Ideen oder durch Anforderungen von Kunden. Für die Zukunft sind somit noch einige Erweiterungen denkbar beziehungsweise wünschenswert:

- Der Einsatz von Java-basierten Programmen wie beispielsweise ANT oder Maven, um Plattform-unabhängig zu sein
- Automatisches Einstellen der JavaScript-Dateien auf den Webserver
- Die Installation von Dumps beziehungsweise der Rücksprung auf eine beliebige Ausgangsversion generell oder speziell vielleicht sogar über Hudson
- die inkrementelle Auslieferung der Apex-Anwendung
- Allgemein ein höherer Grad der Automatisierung
- Umstieg auf Jenkins als CI-Server, da dort die Weiterentwicklung stärker vorangetrieben wird

Peter Busch
peter.busch@mt.ag.com

Dominic Ketteltasche
dominic.ketteltasche@mt-ag.com