OPITZ CONSULTING

Überraschend mehr Möglichkeiten

Orcas: Continuous Delivery für die Datenbank

Ein Framework für die kontinuierliche Auslieferung im Datenbankumfeld

Whitepaper

Orcas: Continuous Delivery für die Datenbank

Ein Framework für die kontinuierliche Auslieferung im Datenbankumfeld

Autor

Dr. Olaf Jessensky

Kontakt

Torsten Jaeschke
OPITZ CONSULTING Deutschland GmbH
Senior Consultant
torsten.jaeschke@opitz-consulting.com

Impressum

OPITZ CONSULTING Deutschland GmbH
Kirchstr. 6
51647 Gummersbach
+49 (0)2261 6001-0
info@opitz-consulting.com

Inhalt

Vorwort

Eines der Hauptanliegen in der modernen Softwareentwicklung ist es, Anwendungen möglichst problemlos zu bauen, zu testen und auf unterschiedliche Zielumgebungen auszurollen. Dazu gibt es Vorgehensweisen, die in der Fachwelt allgemein unter dem Begriff Continuous Integration (CI) [1] zusammengefasst werden.

Bei Projekten im Java/JEE-Umfeld ist das Ergebnis meist eine jar-, war- oder ear-Datei, die auf einem entsprechend konfigurierbaren Zielsystem, in der Regel auf einem JEE-Applikationsserver, installiert wird. Der Trend geht dahin, Anwendungen bereits mit einem Container auszuliefern, sodass sie ohne einen Applikationsserver lauffähig sind. In beiden Fällen wird der Build-Prozess so aufgesetzt, dass die Quelldateien der Anwendung aus einem Source Code Repository wie SVN oder Git ausgecheckt und mithilfe eines Build Tools, wie Maven, Ant oder Gradle, gebaut, getestet und gegebenenfalls auch deployt werden.

Diesen Build-Prozess steuert typischerweise ein CI-Server wie Hudson oder Jenkins. Dabei kann der CI-Server so konfiguriert werden, dass nach jedem Commit einer Änderung im Repository ein neuer Build angestoßen wird und das Ergebnis direkt danach in einer Testumgebung zur Verfügung steht.

In diesem Whitepaper stellen wir Ihnen das Framework Orcas vor, das die Auslieferung von Softwarelösungen auf Datenbankebene erheblich vereinfachen kann.

Bei der Umsetzung von Continuous Delivery in einer Build Pipeline stellen insbesondere klassische relationale Datenbanken mit festem Schema ein Problem dar. Wenn sich die Anwendung entwickelt, muss sich das Datenbankschema daran anpassen und zwar so, dass es zu der entsprechenden Version der Anwendung passt.

In der Regel müssen mit der Änderung eines Schemas auch die entsprechenden Daten migriert werden. Test- und Produktionsdaten stellen ein wichtiges Gut dar und die Integrität muss auch im Rahmen der Schema-Evolution gesichert sein. Auch muss berücksichtigt werden, dass eine Anwendung möglicherweise auf verschiedenen Umgebungen einen unterschiedlichen Stand hat. Alle bestehenden Ist-Zustände müssen also in einen neuen Soll-Zustand überführt werden.

Durch Continuous Delivery werden häufig mehr Umgebungen gepflegt als bei klassischen Vorgehen. Die Notwendigkeit der automatischen Migration steigt also. An dieser Stelle tritt unser Framework Orcas [2] auf den Plan.

1 Was leistet Orcas?

Orcas ermöglicht es, den Zustand von Tabellen in einer Oracle Datenbank mit einer SQL-ähnlichen Syntax zu beschreiben. Das Framework ermittelt das Delta eines gegebenen Zielschemas in der Datenbank zu dem Zielzustand, der in unserem Source Code definiert ist. Daraufhin überführt das Framework den Ist-Zustand in den Soll-Zustand – und sorgt dafür, dass alle Daten erhalten bleiben. Damit schließt es die Lücke, die Continuous Integration für die Datenbank bislang schwierig machte. Zudem können Entwickler mit Orcas den Source Code für die Datenbank effizient in ihrem Source Code Repository verwalten [3].

Was der Ansatz allerdings nicht abdeckt, sind Probleme bei der Migration von Daten in neue Tabellen oder Tabellenspalten. Hier bietet Orcas die Möglichkeit, SQL-Skripte als sogenannte Einmal-Skripte auszuführen: Dazu merkt sich das Programm, welche Skripte in einem bestimmten Zielschema ausgeführt wurden und führt nur neu hinzugekommene Skripte aus.

An dieser Stelle wird der Unterschied zwischen Orcas und marktüblichen Tools wie Liquibase [4] oder Flyway [5] deutlich. Während diese generell nur Einmal-Skripte für die Migration von einem bestimmten Ist-Zustand in einen bestimmten Soll-Zustand verwalten, wird in Orcas nur der Ziel-Zustand beschrieben, das Delta zum Ist-Zustand hingegen selbst ermittelt.

Die Vorteile sind

- eine größere Flexibilität durch die Möglichkeit, ein Deployment unabhängig vom Ist-Zustand des Zielsystems durchzuführen,
- die Möglichkeit, den Entwicklungsstand zu erkennen, der sich bei Liquibase und Flyway nur aus der Summe aller Migrationsskripte ableitet.

Der Preis dafür ist die Einschränkung von Orcas auf Oracle RDMS.

2 Eine Beschreibungssprache für Datenbankobjekte

Der mit Orcas verfolgte Lösungsansatz beinhaltet die Definition einer domänenspezifischen Sprache, in der die zu realisierenden Datenbankobjekte beschrieben werden. Die Syntax dieser Sprache ist, mit einigen Abweichungen und Einschränkungen bezüglich des Sprachumfangs, so nah wie möglich an SQL angelehnt.

Das folgende Beispiel zeigt ein einfaches Tabellenskript für die Tabelle „`orders`“:

```
create table orders
(
  ordr_id         number(15)                not null,
  version         number(15)   default "0"  not null,
  bpar_id         number(15)                not null,
  orderdate         date                    not null,
  tracking_number   varchar2(20)            not null,

  constraint ordr_pk primary key (ordr_id),
  constraint ordr_uc unique (tracking_number),
  constraint ordr_bpar_fk foreign key (bpar_id)
  references business_partners (bpar_id)
);
```

Alle Datenfelder und Objekte, die sich auf die Tabelle „`orders`“ beziehen, werden innerhalb des Befehls `create table` aufgeführt. Das Beispiel demonstriert die einfache SQL-ähnliche Syntax für einen Primärschlüssel, für einen Unique Constraint und für einen Fremdschlüssel.

3 Flexible Erweiterbarkeit der Syntax

Eines der herausragenden Features von Orcas ist die Möglichkeit, eigene Syntaxerweiterungen zu definieren. Auf diese Weise kann das Framework mit geringem Aufwand an die Erfordernisse des jeweiligen Projekts angepasst werden. Als einfaches Beispiel möchte ich die in Orcas enthaltene Domain-Extension vorstellen, die individuell konfiguriert werden kann und die für die einheitliche Behandlung von Tabellen in einem Projekt äußerst nützlich ist.

Eine Tabelle der Domain `id_table` soll automatisch immer eine numerische PK-Spalte mit der Namenskonvention `<table_name>_id` erhalten. Der zugehörige PK-Constraint soll den Namen `<table_name>_pk` haben. Dafür wird lediglich im Tabellenskript angegeben, dass die Tabelle zu der Table Domain `id_table` gehört.

Hier ein Beispiel:

```
create table tab_a domain id_table
(
  somevalue        varchar2(100)
);
```

Im nächsten Code-Auszug sehen wir, wie einfach diese Funktionalität implementiert werden kann.

Der erste Teil des Beispiels zeigt die Definition einer Column Domain, die auf der Ebene von Tabellenspalten wirkt. Diese wird in der darunter stehenden Table Domain verwendet.

```
define column domain pk_column generate-primary-key
(constraint-name(table-name || "_pk"))
(
  number(10) not null
);
define table domain id_table
(
  add column column-name
(table-name||"_"||column-name)(id domain pk_column)
);
```

Die Liste der Anwendungen für solche Erweiterungen reicht von dem hier gezeigten PK-Beispiel über die Anlage von Monitoring-Feldern für den User, der eine Änderung vornimmt, und für den Zeitpunkt der Änderung bis hin zur Implementierung von Historisierungslösungen.

4 Abgleich mit der Datenbank

Für den Abgleich mit dem Ist-Zustand des zu bearbeitenden DB-Schemas verfügt Orcas über eine in Java implementierte Logik, die den Ist-Zustand aus dem Data Dictionary der Oracle Datenbank ermittelt und mit dem Soll-Zustand vergleicht. Dabei werden SQL-Befehle generiert, die den Soll-Zustand herstellen und am Schluss des Abgleichs auf der Datenbank ausgeführt werden.

Standardmäßig steht danach das vollautomatische Deployment der Änderungen auf der Zieldatenbank auf dem Plan. Das Vorgehen lässt sich sehr einfach in eine Continuous Delivery Pipeline integrieren.

Wahlweise kann Orcas so konfiguriert werden, dass das resultierende SQL-Skript nicht direkt ausgeführt, sondern im Dateisystem abgelegt wird. In dem Fall kann der Anwender das Skript vor der Ausführung überprüfen. Häufig wird diese Vorgehensweise gewählt, wenn Änderungen an einem Produktivsystem nur speziell berechtigten Personen erlaubt sind und diese genau dokumentiert werden müssen.

Das hieraus resultierende Skript beschreibt den Weg von einem bestimmten Ist-Zustand in den Soll-Zustand und entspricht damit im Ergebnis dem, was man auch mit Liquibase oder Flyway erhält. Mit dem Unterschied, dass der Entwickler bei Orcas die Datenbankobjekte wie normalen Source Code weiterpflegt und das Delta-Skript für ein bestimmtes Deployment automatisch erzeugen kann.

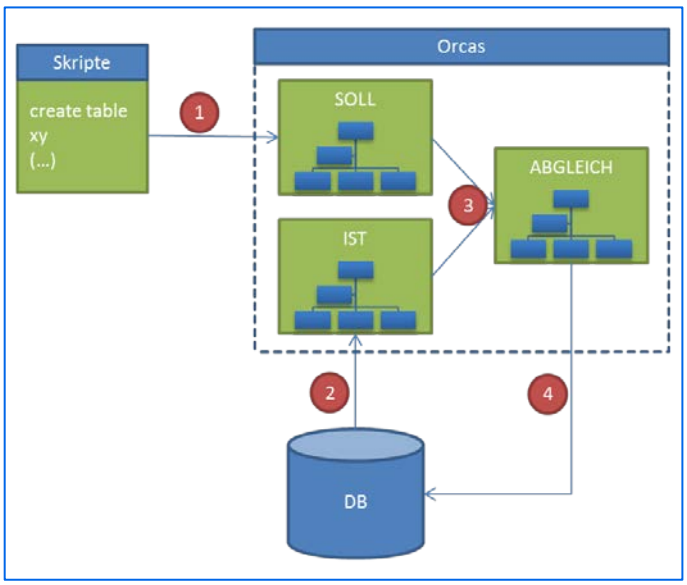

Abbildung 1: Soll/Ist-Abgleich für Datenbankobjekte

5 Integration in den Build-Prozess

Die Einbindung von Orcas in einen kontinuierlichen Build-Prozess geschieht mittels eines der unterstützten Buildwerkzeuge. Momentan sind dies Maven, Gradle und Ant. Damit kann Orcas in einen bestehenden CI-Build, der zum Beispiel in Maven realisiert ist, eingebunden werden. Sinnvollerweise baut der User den Build-Prozess so auf, dass zuerst Datenbanktabellen ausgeführt werden und danach andere Datenbankobjekte wie Views und Packages, die sich auf diese Tabelle beziehen. Die meisten dieser Objekte erhalten keinen Anwendungsstatus wie die Tabellen, sondern können wie normaler Source Code neu in der Datenbank kompiliert werden. Diese Kategorie von Objekten wird von Orcas durch die Ausführung der hinterlegten SQL-Skripte deployt.

Im Anschluss lassen sich bei Bedarf Einmal-Skripte für Datenmigrationen oder Ähnliches ausführen. Für Projekte, die bisher Liquibase benutzt haben, wurde in Orcas die Möglichkeit geschaffen, Liquibase im Rahmen eines Orcas Builds aufzurufen. Damit steht dem Entwickler weiterhin Liquibase für die Verwaltung von Einmal-Skripten zur Verfügung. Gleichzeitig können aber auch die Fähigkeiten von Orcas für den Abgleich des Schemazustands genutzt werden.

6 Einstieg in die Arbeit mit Orcas

Für seine Funktionsfähigkeit benötigt Orcas Java (ab Version 8), mindestens einen zu der Datenbankumgebung passenden JDBC Treiber und je nach Wahl des Buildwerkzeuges, Maven (ab Version 3), Gradle (mindestens Version 3.3) oder die Buildwerkzeuge Ant und Gradle (in der jeweils aktuellen Version). Für Maven und Gradle steht Orcas auch als Maven Artefakt zum Einbinden im Maven Central Repository bereit [6]. Die vorgenannten Voraussetzungen müssen auf jedem Entwicklerarbeitsplatz sowie auf jedem Buildserver, auf dem Orcas eingesetzt werden soll, gegeben sein. Für Testzwecke liefern wir eine Vagrant Konfiguration [7] mit, mit der eine VM konfiguriert und und gestartet werden kann. Diese enthält eine Datenbankinstallation, in der Orcas direkt lauffähig ist.

Wenn man ein Projekt auf der „grünen Wiese" beginnt, startet man normalerweise nach der Festlegung und der Implementierung oder der Konfiguration der benötigten Extensions mit der Erstellung von Orcas Skripten. Häufiger befinden sich Entwickler jedoch in der Situation, auf ein bereits bestehendes DB-Schema aufbauen zu müssen. Da die Syntax von Orcas leicht von SQL abweicht, benötigt man dazu ein Reverse Engineering. Zu diesem Zweck gibt es in Orcas für jedes Buildwerkzeug einen Task, der den Ist-Zustand eines DB-Schemas in Orcas Skripte übersetzt. Auf der Basis dieser Skripte kann man die Weiterentwicklung des Schemas mit Orcas fortsetzen.

7 Fazit

Mit Orcas können Entwickler die Definition von Oracle Datenbanken im Entwicklungsprozess wie gewöhnlichen Source Code verwalten und weiterentwickeln. Dadurch bietet es die Möglichkeit, die Konzepte von Continuous Integration und Continuous Delivery auch auf den Bereich der Datenbank anzuwenden. Gleichzeitig stellt es mit seinen Syntax-Extensions ein mächtiges Werkzeug bereit, das die Verwaltung von Datenmodellen erleichtert. Dabei geht das Framework weit über die Möglichkeiten von Flyway und Liquibase hinaus.

Orcas wurde unter Github [8] als Open Source veröffentlicht. Jedermann ist eingeladen, es zu verwenden und sich an seiner Weiterentwicklung zu beteiligen.

8 Quellen

[1] http://www.thoughtworks.com/continuous-integration
[2] http://opitzconsulting.github.io/orcas/docs/about/
[3] http://databaserefactoring.com/
[4] http://www.liquibase.org/
[5] http://flywaydb.org/
[6] http://www.mvnrepository.com/artifact/com.opitzconsulting.orcas
[7] https://www.vagrantup.com/
[8] https://github.com/opitzconsulting/orcas

9 Continuous Delivery by OPITZ CONSULTING

Vom Commit einer Source-Code-Änderung bis zum Release eines Softwaresystems in Produktion ist es meist ein weiter Weg: Änderungen müssen getestet, Konfigurationen geändert, Patches eingespielt und Datenbankschemata aktualisiert werden. Das Ergebnis sind lange Releasezyklen und "Bauchschmerzen" vorm nächsten Release. Gleichzeitig steigen die Erwartungen Ihrer Fachanwender.

Kurze Änderungszyklen

Damit die IT als Partner und Innovationstreiber im Unternehmen gesehen wird, müssen sich Softwaresysteme so schnell verändern, wie das Business selbst. Continuous Delivery ermöglicht kurze agile Iterationen und verkürzt die Zeitspanne von der Anforderung zur produktiven Unterstützung durch Ihre Software von Wochen und Monaten auf Tage und Stunden.

Automatisierung mittels Build Pipeline

Dank moderner Entwicklungs- und Testkonzepte sowie eines hohen Automatisierungsgrads bekommen Sie mehr Sicherheit und Qualität in die Lieferkette. Eine Build Pipeline sorgt für den automatisierten Ablauf, indem sie wie das Fließband in einer Fabrik die Pfade visualisiert, die jede Änderung nehmen muss. Die Build Pipeline schafft eine Übersicht über laufende und vergangene Code-Änderungen und ihre Produktionsreife. Wurden alle Stationen in der Build Pipeline erfolgreich absolviert, kann die Änderung auch mit einem Mausklick in die Produktion eingespielt werden.

Was bringt Ihnen Continuous Delivery?

- Höhere Wertschöpfung der IT durch schnelle und risikoarme Lieferung von Änderungen
- Schnellere Time-to-Market durch durchgängige Prozesse ohne aufwändige Übergaben von der Entwicklung bis zur Produktivsetzung in den Kundenprojekten.
- Bessere Release-Qualität durch automatischen, reproduzierbaren Vorgang und weniger Fehlerquellen

Auf Basis Ihrer individuellen Roadmap unterstützen wir Sie bei der Bedarfsanalyse, der Einführung von Werkzeugen, Anpassung von Build-Prozessen und Vorgehensmodellen (Scrum, Kanban) bis hin zur Unterstützung beim Aufbau einer integrierten Continuous-Delivery-Plattform:

Impulsvortrag/Bedarfsanalyse

Erfahren Sie, was Continuous Delivery für Sie bedeutet und definieren Sie mit unseren Experten ihre Umsetzungsstrategie. Lernen Sie aus Architektensicht Werkzeuge wie Vagrant, Docker und Ansible kennen und erfahren sie, wie diese zu wesentlichen Bestandteilen Ihrer Build Pipeline werden.

Roadmap

Wir helfen Ihnen zu verstehen, wo sich die Engstellen in Ihrem Entwicklungs- und Bereitstellungsprozess befinden. Aufbauend auf dieser Value-Stream-Analyse schlagen wir Ihnen Praktiken und Werkzeugen vor, mit denen Sie diese Prozesse verändern können. Wir überlegen gemeinsam mit Ihnen, welche Schritte Sie aus eigener Kraft machen und für welche Schritte Sie unsere Unterstützung in Anspruch nehmen möchten.

Umsetzung

- Aufbau einer Build Pipeline
- Integration von Log Management und Application Performance Management in den Build-Prozess
- Bereitstellung von Umgebungen auf Basis von Applikationscontainern (Docker) und virtuellen Maschinen
- Management von Datenbanken im Continuous Delivery

Weiterführende Infos und Kontakt

Viele interessante Infos rund um das Thema DevOps und Continuous Delivery haben wir hier für Sie zusammengestellt: www.opitz-consulting.com/devops

Informieren Sie sich auf unserer Homepage www.opitz-consulting.com auch zu weiteren Angeboten und Leistungsfeldern unseres Beratungshauses.