Nr.2 | April 2017 | ISSN 2366-7915 | www.doag.org

Red Stack Magazin

Aus der Praxis

DevOps für die Backend-Entwicklung

Im Interview

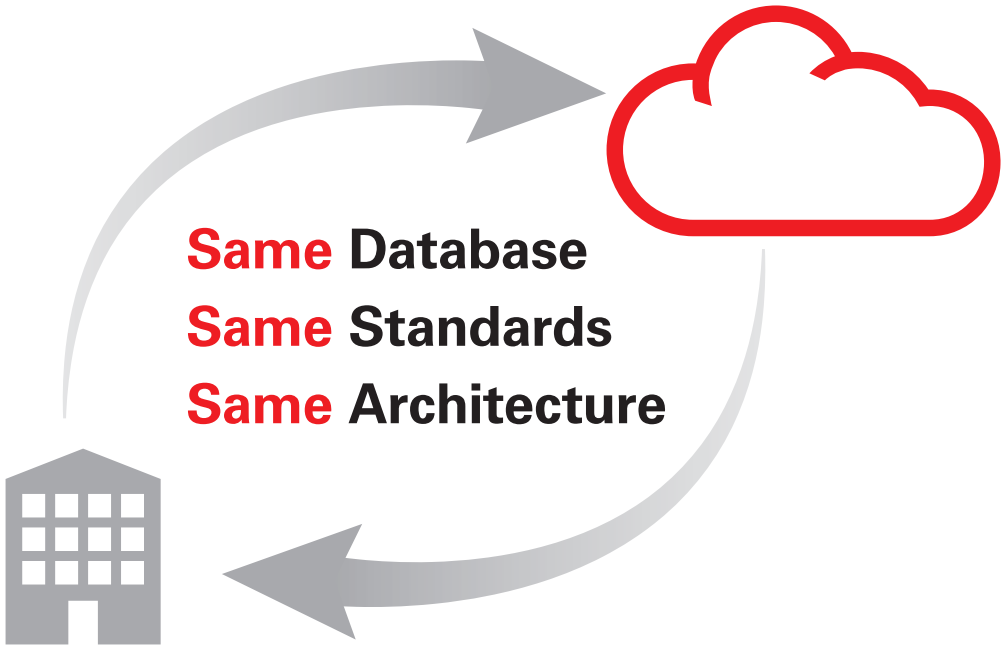
Gernot Grünsteidl, Alte Leipziger-Hallesche Konzern

Exadata

Proof of Concept mit Licht und Schatten

Push a Button

Move Your Database to the Oracle Cloud



... or Back to Your Data Center

ORACLE®

cloud.oracle.com/database
or call 1.800.ORACLE.1

Robert Szilinski
Leiter Development Community

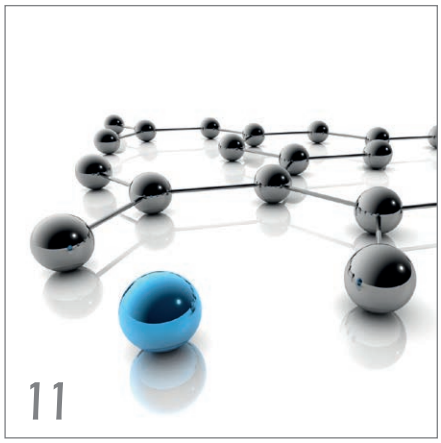
Liebe Mitglieder, liebe Leserinnen und Leser,

vor fünf Jahren war DevOps noch weitgehend unbekannt. Mittlerweile hat es sich zu einem der absoluten Top-Themen in den Unternehmen entwickelt und wenn man den Analysten glaubt, dann wird 2017 das Jahr des DevOps.

Die Effekte gehen weit über die reine Software-Entwicklung hinaus. Mit DevOps brechen Teams das Silodenken zwischen „Development" und „Operations" auf und die Zusammenarbeit wird gefördert. Auch das Business profitiert direkt von der immensen Geschwindigkeit, die den Takt der Digitalisierung in der Software-Entwicklung aufnimmt. Durch die Automatisierung von Deployments und permanente Tests für die Qualitätskontrolle ist es möglich, die Zeiten zwischen den Releases einer Software dramatisch zu reduzieren; fachliche Anforderungen oder Verbesserungen können schnellstmöglich zur Verfügung gestellt werden (Continuous Delivery). Die kleinen Änderungsschritte senken dabei auch das Gesamtrisiko für das Unternehmen.

Die Einführung von DevOps-Konzepten ist jedoch kein Selbstläufer und gerade auch in der Oracle-Welt in allen Dimensionen zu betrachten. Das betrifft nicht nur das reine Development, sondern vor allem auch Datenbank- und Infrastruktur-Themen. Technisch können für eine Einführung von DevOps die Cloud-Strategie von Oracle sowie viele frei verfügbare Tools und Technologien helfen, die sich sinnvoll kombinieren lassen. In dieser Ausgabe haben wir ein paar Beispiele zusammengestellt und wünschen in diesem Sinne viel Spaß beim Lesen und Umsetzen.

Ihr



„DevOps“ steht für die Einrichtung einer Kette von Prozessen

Container bieten Vorteile gegenüber traditionellen Infrastruktur-Plattformen

Der Oracle Developer Cloud Service im praktischen Einsatz

3 Editorial

5 Timeline

7 Aus der Ferne betrachtet: Oracle Support – Quo vadis?
Dr. Dietmar Neugebauer

8 „Wir wünschen uns einen Support auf Augenhöhe …“
Interview mit Gernot Grünsteidl

DevOps

11 DevOps für die Backend-Entwicklung
Oliver Lemm

14 WebLogic-WLST-Programmierung unter Eclipse mit dem Oracle-OEPE-Plugin und PyDev
Michael Schulze

20 Der Wunsch nach mehr Agilität: Microservices und Container
Mario Herb und Matthias Fuchs

26 Database as a Service – Fakten und erste Erfahrungen
Tobias Deml

30 Application Lifecycle Management mit dem Oracle Developer Cloud Service
Stefan Kühnlein

Datenbank

36 Adaptive Features oder wie ich lernte, ruhig zu bleiben, um die Bombe zu beherrschen
Ludovico Caldara

43 Objektorientierte Entwicklung in der Datenbank – geht denn das?
Anja Hildebrandt

Entwicklung

50 Continuous Integration in Apex-Projekten
Sven Böttcher

58 Oracle Keystores – der Schlüssel zum Glück
Andreas Chatziantoniou

69 Datenmodellierung in Cassandra und die Cassandra Query Language (CQL)
Jan Ott

Exadata

62 Konsolidierungsplattform Exadata: PoC-Erfahrungsbericht mit Licht und Schatten
Gregor Büchner und Uwe Simon

Intern

73 Termine

74 Neue Mitglieder

74 Inserenten

74 Impressum

Timeline

2. Januar 2017

Die AOUG startet das Call for Papers für die Wiener Anwenderkonferenz am 19. und 20. Juni 2017. Die Hauptveranstaltung startet in diesem Jahr früher, wird auf zwei Tage ausgedehnt und wegen der Vielzahl an Themen um weitere Breakout-Sessions und Workshops erweitert. Der erste Tag läuft unter dem Motto „AOUG Education Day" und bietet in drei parallelen Tracks Raum für Seminare und Workshops an.

13. Januar 2017

Diesmal geht es im DOAG-Webinar um die Oracle-Database-Cloud-Performance. Der unabhängige Berater Randolf Geist beantwortet wichtige Fragen, darunter „Wie steht es um die Performance in der Oracle Database Cloud?", „Wie konsistent im Sinne der Performance verhält sich eine Datenbank dort, gibt es größere Schwankungen?", „Wie sieht es mit CPU-, wie mit I/O-lastigen Benutzungsprofilen aus?" oder „Gibt es Unterschiede zwischen ASM bei RAC-Einsatz und den vorkonfigurierten Filesystemen bei Single-Instance?". Er unterstreicht seine Antworten durch entsprechende Testergebnisse. Die einstündigen, virtuellen Seminare erweisen sich als perfekte Ergänzung zu weiteren DOAG-Veranstaltungen. Sie können unter „http://www.doag.org/events/webinar/aufzeichnungen-webinare.html" auch nachträglich angesehen werden.

18. Januar 2017

Die Forms-Gemeinschaft der DOAG startet hochmotiviert ins neue Jahr. Mehr als 60 Teilnehmer kommen im Berliner Humboldt Carré zusammen, um sich über den aktuellen Stand der Oracle-Technologie auszutauschen. Ihr Fazit: Das Interesse an Oracle Forms ist nach wie vor groß. Damit das so bleibt, sind aber aktive Nachwuchsarbeit und ein besserer Wissensaustausch nötig. „Wer sind eigentlich diese erfahrenen Forms-Entwickler?", fragt Frank Hoffmann, seines Zeichens selbst Forms-Experte, gleich zu Beginn der Veranstaltung. Daraufhin zeichnet er in seinem unterhaltsamen wie appellierenden Vortrag zur Zukunft von Oracle Forms ein klares Bild des typischen Forms-Anhängers und kam zu dem Schluss: Der typische Forms-Entwickler entspricht in der Regel dem erfahrenen Zauberer Gandalf aus „Herr der Ringe" – weise zwar, aber schon im fortgeschrittenen Alter. Damit spricht Hoffmann eine der größten Herausforderungen der Forms-Anhänger an: die fehlenden jungen Gefährten. Denn wer kann in Zukunft dafür sorgen, dass Forms weiterlebt, wenn der Mehrheit der Forms-Experten in den nächsten zehn Jahren der Renteneintritt bevorsteht? Hoffmanns Appell lautete daher: „Forms geht nur weiter, wenn alle mitmachen." Um die Zeit bis zum nächsten DOAG Forms Day zu überbrücken, hat das Gefährten-Trio um Jürgen Menge, Gerd Volberg und Frank Hoffmann mehrere Plattformen ins Leben gerufen, auf denen sich alle Forms-Interessierten über Demos, Source Code und ihre Fragen rund um Forms austauschen können (siehe „https://tinyurl.com/jxvjvb8"). Die zentrale E-Mail-Adresse für Forms-Aktivitäten innerhalb der DOAG ist „forms@doag.org".

Neben der Modernisierung standen insbesondere auch die Neuerungen von Forms 12c auf dem Programm

24. Januar 2017

Die DOAG führt für die Mitarbeiterinnen und Mitarbeiter in der Geschäftsstelle einen Erste-Hilfe-Kurs durch. Ganz im Stil der DOAG ist dieser sehr praxisnah mit viel Fallbeispielen gestaltet. So ist das Team auch auf medizinische Notfälle gut vorbereitet.

25. Januar 2017

Das Organisationsteam der JavaLand 2017 trifft sich im Phantasialand in Brühl, um vor Ort die letzten Vorbereitungen zu treffen. Aufgrund der erwarteten hohen Teilnehmerzahl gilt es vor allem, die Besucherströme optimal zu lenken. So werden ein neuer Zugang vom Hotel Lingbao auf das Veranstaltungsgelände eingerichtet und die Kapazität der Bus-Shuttles verdreifacht.

Die Java-Community traf sich vom 28. bis 30. März 2017 im Phantasialand in Brühl

25. Januar 2017

Die Generalversammlung der AOUG in Wien entlastet den Vorstand für das vergangene und wählt ihn für ein neues Vereinsjahr. Zudem werden die weiteren Schritte für die kommende AOUG-Anwenderkonferenz am 19. und 20. Juni 2017 in Wien beschlossen.

30. Januar 2017

Zum Ende des Frühbucher-Rabatts haben sich rund zwanzig Prozent mehr Teilnehmer für die JavaLand 2017 angemeldet. Damit zeichnet sich ein neuer Besucherrekord ab.

8. Februar 2017

Das vierte DOAG DevCamp ist auch dieses Mal wieder ein spannender Mix aus spontanen Sessions und vielen offenen Diskussionen. In lockerer Atmosphäre diskutieren etwa 60 Teilnehmer über aktuelle Trends und Themen rund um die Software-Entwicklung. Den Anfang macht der Software-Entwickler Jens Schauder mit seiner Keynote „Clean Code". Ein Vergleich mit den hygienischen Verhältnissen im Mittelalter soll den Entwicklungsstand der Softwarebranche darstellen: „Wir leben im IT-Mittelalter", ruft der Java-Profi seinen Zuhörern zu. Schauder mahnt sein Publikum in der Keynote hinsichtlich der Ursachen, die zu unsauberem Code führen könnten. Mit viel Humor präsentiert er einige Code-Beispiele und versucht, seine Zuhörer für das Thema zu sensibilisieren. Sein Credo: Code ist von Menschen gemacht und es ist wichtig, dass er sauber ist, damit wir ihn verstehen. Nach diesem humorvollen Weckruf starten die Teilnehmer gut gelaunt in die Planung der Sessions. Jeder Teilnehmer kann sich einbringen; schnell füllt sich der Session-Plan und bildet einen interessanten Überblick zu Themen wie Java, JavaScript, Forms, PL/SQL und Code-Tests. Networking ist dank des offenen Barcamp-Formats zu jeder Zeit möglich. Offene Türen zu den Sitzungsräumen laden zum Reinschauen in die parallel ablaufenden Sessions ein. Für viele Teilnehmer ist dies das erste Barcamp und mal etwas anderes als die üblichen Veranstaltungsformate. Die Themenauswahl und vor allem die Tiefe der Sessions steigern schon die Vorfreude auf das nächste DevCamp im Jahr 2018. Am Rand der Veranstaltung reden Fried Saacke, DOAG-Vorstand und Geschäftsführer, und Robert Szilinski, Leiter der DOAG Development Community, über Wege, um die Oracle ADF Community enger mit der DOAG zusammenzubringen.

Insgesamt wird das diesjährige DevCamp überaus positiv von den Teilnehmern angenommen

14. Februar 2017

Bereits zum zehnten Mal findet der Primavera Community Day statt, in diesem Jahr in München. Das Event ist ein Gemeinschaftsprodukt der DOAG, AOUG sowie SOUG und verbindet aktuelles Know-how mit zahlreichen Networking-Möglichkeiten. Im Fokus stehen konkrete Fragen rund um die Enterprise-Software Primavera sowie allgemeine Themen wie „IoT" und „Projektmanagement". Am Vorabend findet ein Get-Together in der Münchner BMW-World statt, mit Rundgang und gemeinsamem Abendessen.

14. Februar 2017

Der Anwender-Beirat der DOAG trifft sich mit dem Vorstandsvorsitzenden Stefan Kinnen bei MyToys in Berlin. Das Gremium aus der Businesswelt berät die DOAG strategisch. Die acht Teilnehmer diskutieren vier Stunden lang über Lizenzierung, Support und Cloud-Strategie von Oracle. Die Atmosphäre ist sehr gut und konstruktiv. Das nächste Treffen ist für Mai in Düsseldorf geplant.

17. Februar 2017

Der DOAG-Vorstand beschließt im Umlaufverfahren, Martin Schmitter als regionalen Repräsentanten der Regionalgruppe NRW einzusetzen. Er löst Andreas Stephan ab, der das Amt aus beruflichen Gründen nicht mehr ausfüllen kann. Martin Schmitter bedankt sich für das in ihn gesetzte Vertrauen. Er habe großen Respekt vor der Aufgabe und freue sich auch sehr darauf.

Martin Schmitter, regionaler Repräsentant NRW

Dr. Dietmar Neugebauer
Ehemaliger DOAG-Vorstandsvorsitzender

Aus der Ferne betrachtet: Oracle Support – Quo vadis?

„Akzeptanz von My Oracle Support steigt, allgemeine Zufriedenheit sinkt", berichtet das Red Stack Magazin in seiner letzten Ausgabe über die Ergebnisse der DOAG-Support-Umfrage. Oracle hat zwar versucht, diese Ergebnisse mit Bezug auf die geringe Umfragebeteiligung zu relativieren, aber die Unzufriedenheit der Kunden ist offensichtlich. Man muss auf Oracle- oder DOAG-Veranstaltungen nur das Thema „Support" ansprechen, schon bekommt man unvermittelt ähnliche Rückmeldungen wie bei der Umfrage: schlechte Qualität bei der Bearbeitung der Service Requests, Unzufriedenheit mit den Prozessen und Abläufen sowie zu hohe Reaktionszeiten. Mit der Schließung des deutschsprachigen Supports und der Zentralisierung auf drei Supportcenter in Rumänien, Indien und den USA hat sich das Ergebnis sicherlich nicht verbessert.

Mehrmaliges Anfordern von Informationen, wodurch die Bearbeitung immer wieder hinausgezögert wird, mangelndes fachliches Wissen des Bearbeiters, keine Rückmeldung darüber, ob an dem Problem gearbeitet wird, mangelnde interne Kommunikation mit dem Produkt-Development, wenn es sich offensichtlich um einen Bug in der Software handelt, Service Requests, die sechs Monate und länger laufen, sehr schwierige sprachliche Kommunikation selbst für englische Muttersprachler – dies sind nur die häufigsten Rückmeldungen der Oracle-Kunden. Was häufig einfach fehlt, ist die Sichtweise, dass bei einem Problem zwei Seiten partnerschaftlich interagieren – die eine Seite, die das Problem hat und beschreibt, und die andere Seite, die nicht nur die Abarbeitung, sondern auch die Lösung des Problems in Abstimmung mit der Kundenanforderung zum Ziel hat.

Was macht der Kunde? Ein Teil hat resigniert und hilft sich selbst mit Workarounds oder Googeln im Internet. Ein anderer Teil kauft zusätzlich teuren Advanced Customer Support bei Oracle ein, um die Nachverfolgung und Eskalation von Service Requests von Oracle-Mitarbeitern durchführen zu lassen.

Die Unzufriedenheit mit dem Oracle-Support hat inzwischen auch das Management der Oracle-Kunden erreicht. Immerhin fallen pro Jahr Support-Kosten von mehr als zwanzig Prozent der nicht gerade niedrigen Lizenzkosten an. Da kann man dann auch erwarten, dass bei bekannten Problemen (Performance, wrong results) mit neuen Features wie „dynamic query optimizer" der Support rechtzeitig eine Vorabinformation an den Kunden schickt, bevor alle in das gleiche Problem laufen.

Seitens Oracle sind in der Stellungnahme im letzten Red Stack Magazin viele dieser bekannten Support-Probleme angesprochen worden. Es ist allerdings nicht zu erkennen, welche Maßnahmen ergriffen wurden. Hier wäre es sehr zu begrüßen, wenn Oracle dies möglichst bald offen kommuniziert.

Die Computerwoche hat am 16. Dezember 2016 in dem Artikel „Oracle – Cloud wächst, On-Premise schrumpft" darauf hingewiesen, dass mit der Abnahme von On-Premise-Lizenzverkäufen wohl auch das Wartungsgeschäft bei Oracle rückläufig wird. Weniger Support-Einnahmen gleich sinkende Qualität? Da sollte man bei Oracle zumindest im Auge behalten, dass bei der DOAG-Umfrage auch der Anteil der Kunden zugenommen hat, die sich Support durch einen Drittanbieter vorstellen könnten. Vielleicht setzt der eine oder andere Kunde beim Oracle-Support den Rotstift an und hinterfragt Kosten und Leistung – dann „Quo vadis, Oracle-Support"?

Gernot Grünsteidl (links) im Gespräch mit Dr. Dietmar Neugebauer

„Wir wünschen uns einen Support auf Augenhöhe …“

Die optimale Nutzung der Datenbanken ist die Basis für das Geschäft von Versicherungen. Wolfgang Taschner, Chefredakteur des Red Stack Magazin, und der frühere DOAG-Vorstandsvorsitzende Dr. Dietmar Neugebauer sprachen darüber mit Gernot Grünsteidl, Bereichsleiter Datenbanken/Datenkommunikation beim Alte Leipziger-Hallesche Konzern.

Der ALTE LEIPZIGER – HALLESCHE Konzern

Die ALTE LEIPZIGER Lebensversicherung a.G. und die HALLESCHE Krankenversicherung a.G. sind die Muttergesellschaften des Konzerns. Sie bilden einen Gleichordnungskonzern nach § 18 Abs. 2 AktG. Beide Gesellschaften besitzen die Rechtsform des Versicherungsvereins auf Gegenseitigkeit (VVaG). Bei dieser Rechtsform sind die Versicherungsnehmer zugleich Mitglieder und damit Träger der Gesellschaften. Der VVaG garantiert die Unabhängigkeit des Konzerns und damit die langfristige Stabilität der Geschäftspolitik.

Als langjährig erfolgreicher Finanzdienstleister bietet der Konzern seinen Kunden alle Produkte rund um die Themen Versicherungen und Finanzen an. Ein besonderer Schwerpunkt liegt auf dem Personenversicherungsgeschäft, speziell auf den Bereichen der Lebens- und Krankenversicherung. Die Produktpalette wird durch Sach- und Rechtsschutzversicherungen sowie durch lukrative Investmentfonds-, Bauspar- und Baufinanzierungsprodukte optimal ergänzt.

In welchem Geschäftsbereich ist die Alte Leipziger tätig?

Gernot Grünsteidl: Der Alte Leipziger-Hallesche Konzern gehört mit Beitragseinnahmen von etwa viereinhalb Milliarden Euro und rund dreitausend Mitarbeitern zu den führenden Erstversicherungsgruppen in Deutschland. Das Angebot umfasst private und betriebliche Altersversorgung, Absicherung des Berufsunfähigkeitsrisikos, privaten Krankenversicherungsschutz, Sachversicherungen, Bausparen und Baufinanzierung sowie Investmentfonds-Lösungen.

Was sind dabei die besonderen Herausforderungen an die IT?

Gernot Grünsteidl: Neben dem immer aktuellen Thema „Kosten" stehen momentan die Aspekte der Digitalisierung im Fokus. Aber erstens stellt sich die Frage, was man unter Digitalisierung versteht und wo der Unterschied zu dem, was wir bisher gemacht haben, ist, und zweitens spreche ich lieber von Herausforderungen aufgrund des geänderten Kundenverhaltens. Damit wird auch schneller klar, welche Anforderungen entstehen.

Was verstehen Sie unter Digitalisierung?

Gernot Grünsteidl: Mit der Strategie „VerNetzt 20.20" werden wir aktiv unsere Zukunft gestalten. Im Blickpunkt steht die noch konsequentere Ausrichtung der Produkte, Vertriebskanäle und Betriebsprozesse auf die Kunden. Wir werden in den nächsten Jahren die notwendigen Investitionen umsetzen, um die digitale Transformation, insbesondere in die kanalübergreifende Interaktion mit Kunden und Vertriebspartnern sowie in moderne Vertriebskanäle und Betriebsprozesse, zu forcieren. Der Konzern nutzt die Chancen dieses herausfordernden und dynamischen Marktumfelds aus einer Position starker Kundenorientierung, hoher Produktqualität und starker Bilanzkraft. Als Erstes denkt man hier natürlich an Kunden-Apps, aber wichtig ist meiner Meinung nach eine Omnikanal-Strategie, also wie man welchen Kunden für welches Produkt erreicht.

Wie ist Ihre IT aufgebaut?

Gernot Grünsteidl: Als ich vor mehr als fünfundzwanzig Jahren zur Alten Leipziger kam, basierte die IT komplett auf Siemens BS2000. Kurz darauf erfolgte der Wechsel zur IBM und hin zur Client-Server-Architektur. Dadurch wurden die gesamten Anwendungen neu entwickelt. Parallel dazu haben wir das Finanzwesen von SAP eingeführt. Unsere großen Bestandssysteme laufen heute noch auf DB/2 im Hintergrund, im Vordergrund sind Windows-Server und -Clients im Einsatz. Darüber hinaus setzen wir etliche Standard-Software-Produkte ein, die mitunter nur von wenigen Anwendern genutzt werden, beispielsweise im Wertpapiergeschäft. Im Zuge dessen hat auch die Oracle-Datenbank Einzug in unser Haus genommen. Später kam dann zur Konsolidierung der Datenbanken und zum Betrieb eines Data Warehouse eine Exadata zum Einsatz.

Wie ist das Verhältnis zwischen Standard-Software und Eigenentwicklung?

Gernot Grünsteidl: Unsere Bestands- und Leistungssysteme für die Versicherungssparten und das Data Warehouse sind Eigenentwicklungen. Das wird auch auf absehbare Zeit so bleiben. Die Programmierung erfolgt größtenteils in Cobol, C# und Java. Im Finanzbereich ist SAP der Standard. Hinzu kommt weitere Standard-Software, die wir in bestimmten Nischen einsetzen.

Welche Produkte von Oracle kommen bei Ihnen zum Einsatz?

Gernot Grünsteidl: Als Datenbank das Data Warehouse und für viele der unter „sonstige Standard-Software" genannten Bereiche.

Wie halten Sie Ihre Datenbanken auf dem aktuellen Stand?

Gernot Grünsteidl: Wir machen ein quartalsweises Patchen für DB/2 z/OS und halbjährlich für Oracle/Exadata.

Wie rollen Sie Patches auf die Oracle-Datenbanken aus?

Gernot Grünsteidl: Wir patchen die rund dreißig Datenbanken im Oracle-Umfeld durch Platinum-Support oder auch mal über den Oracle Advanced Customer Support.

Wie zufrieden sind Sie mit den Oracle-Datenbanken?

Gernot Grünsteidl: Wir setzen die Version 12 ein und sind mit der Leistung zufrieden. Lediglich die Sache mit den Pluggable Databases scheint mir noch nicht ganz ausgereift zu sein.

Wie gehen Sie mit den Online-Tools von Oracle-Support um?

Gernot Grünsteidl: Wir nutzen My Oracle Support, weil wir es müssen. Bei der Kommunikation vermissen wir die Möglichkeit, Text zu formatieren. Den Community-Bereich würden wir gerne öfter nutzen, wenn er übersichtlicher und besser in My Oracle Support integriert wäre. Das Platinum-Portal mit Configuration Items ist umständlich, es gibt keine Automatik und zur Pflege müssen Calls aufgemacht werden.

Wie sind generell Ihre Erfahrungen mit Oracle Support?

Gernot Grünsteidl: Seit der Verlagerung des europäischen Sup-

ports nach Rumänien haben wir eine deutliche Verschlechterung wahrgenommen. Auch die verschiedenen Ausprägungen beziehungsweise Erweiterungen wie Platinum-Support oder Advanced Customer Support machen es dem Anwender nicht immer einfacher, sein Problem an der richtigen Stelle abzusetzen. Immer wenn man denkt, es kann nicht mehr schlimmer kommen, gibt es nochmal eine Steigerung. Der Exadata-Support wiederum wird fast komplett aus Indien geliefert. Die Qualität hat auch hier in den letzten zwei Jahren nochmal gefühlt enorm nachgelassen. Die Ingenieure haben heute oft ein zu geringes Fachwissen, geben schnell an weitere Instanzen ab und sind dann nur noch Sprachrohr für technische Sachverhalte, die sie nicht verstehen. Dabei geht viel Information verloren. Auf der anderen Seite bekommt man schnelle Lösungen wie das Setzen seltener Underscore-Parameter ohne weitere Erklärungen zu Nebenwirkungen. Auch die Prozesse funktionieren nicht; normale Eskalationen bringen so gut wie nichts mehr. Selbst bei eskalierten Fällen mit Einbeziehen des deutschen Vertriebs und Managements kommt man oft nicht weiter. Die Antwortzeiten sind zum Teil inakzeptabel. Auch die Kommunikation ist ein großes Problem. Manchmal hat man das Gefühl, die Service Requests würden von Robotern mit wenigen Standardsätzen bearbeitet. Oft fehlt auch einfach nur die Transparenz, ob beziehungsweise was im Hintergrund beim Support mit dem Service Request passiert. Wir wünschen uns einen Support auf Augenhöhe, mit der Möglichkeit, komplexe Sachverhalte mit den Ingenieuren zu diskutieren.

Wie ist im Vergleich dazu der Support bei SAP?

Gernot Grünsteidl: Da wir bei SAP weitgehend nur den Standard einsetzen, gibt es hier kaum Probleme. Die Dinge, die wir dem Support melden, werden zufriedenstellend beantwortet.

Wie beurteilen Sie die Zukauf-Strategie von Oracle?

Gernot Grünsteidl: Für uns als klassischen Oracle-Datenbank-Kunden waren die meisten Zukäufe bisher irrelevant, möglicherweise ändert sich das mit den Cloud-Services.

Welchen Status haben Cloud-Anwendungen in Ihrem Unternehmen?

Gernot Grünsteidl: Es gab erste kleine Überlegungen und das Thema bleibt natürlich unter Beobachtung, bisher hat sich aber nichts Konkretes ergeben.

Wie gehen Sie an das Thema „Big Data" heran?

Gernot Grünsteidl: Hier fehlen uns noch konkrete Ansätze. Wenn man hier investiert, muss auch etwas dabei herauskommen.

Wie sind Ihre Erfahrungen mit der Exadata?

Gernot Grünsteidl: Die Exadata ist bei uns seit April 2013 produktiv im Einsatz. Die Maschine hat sich als Konsolidierungsplattform in technischer wie auch lizenztechnischer Hinsicht angeboten. Sie erfüllt hinsichtlich Performance und Hochverfügbarkeit unsere Erwartungen. Allerdings habe ich das Gefühl, dass hier einiges „mit Blech" erschlagen wird. Wir werden demnächst auf ein neueres Modell umsteigen.

In welche Richtung wird sich Ihre IT in den kommenden Jahren entwickeln?

Zur Person: Gernot Grünsteidl

Nach der Ausbildung an der Physikalisch-Technischen Lehranstalt in Wedel und dem Wehrdienst im DV-Betrieb der Marine war er ab dem Jahr 1990 zwei Jahre System-Administrator bei der NUR Touristic, dort dann für drei Jahre Datenbank-Administrator für DB2 und System-Programmierer für CICS. Im Mai 1995 begann Gernot Grünsteidl als DB2-Administrator beim Unternehmensverbund Alte Leipziger und ist dort seit Juli 1998 Bereichsleiter Datenbanken/Datenkommunikation. Im April 2016 kam durch eine Neuorganisation der IT die Verantwortung für diesen Bereich bei der Hallesche Krankenversicherung hinzu.

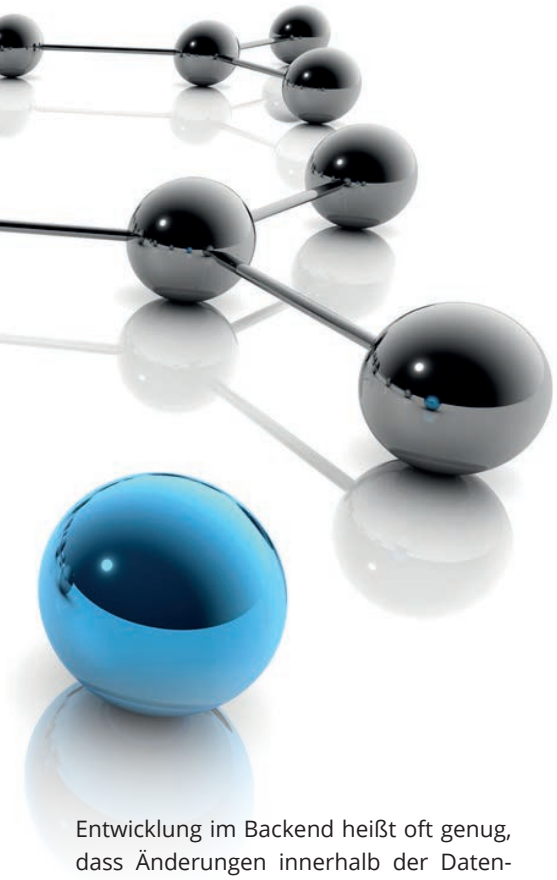
Gernot Grünsteidl: Die wichtigsten Ziele sind Digitalisierung und Kostensparen. Ein aktuelles Projekt ist die Etablierung des Inkasso/Exkasso-Moduls FS-CD für Versicherungen von SAP bei der Halleschen.

Welche Wünsche haben Sie an Oracle?

Gernot Grünsteidl: Mit dem Umfang der Produkte sind wir soweit zufrieden. Unser Wunsch ist eine deutliche Verbesserung des Supports, insbesondere für Exadata-Kunden.

Wie sehen Sie den Stellenwert einer Anwendergruppe wie der DOAG?

Gernot Grünsteidl: Die Arbeit der Anwendergruppen schätze ich sehr hoch ein, hier helfen sich Anwender untereinander und die entstandenen persönlichen Kontakte können einem sehr viel Zeit sparen. Es muss nicht jeder Einzelne jedes Problem alleine lösen, idealerweise läuft man dadurch erst gar nicht hinein. Wir sind auch in der GSE, der DSAG und im SAP-Arbeitskreis Versicherungen vertreten, außerdem gehöre ich dem neu gegründeten Anwender-Beirat der DOAG an, der den DOAG-Vorstand praxisrelevant berät.



DevOps für die Backend-Entwicklung

Oliver Lemm, MT AG

Die Software-Entwicklung muss immer schneller das Produkt an den Kunden oder auf den Markt zu bringen, denn nur produktive Software bedeutet Nutzen. Während bei der Java- oder #Net-Entwicklung Automatisierung und Continuous Delivery weit verbreitet sind, wird dies in der Backend-Entwicklung oft vernachlässigt oder als zu komplex angesehen.

Entwicklung im Backend heißt oft genug, dass Änderungen innerhalb der Datenbank durchgeführt werden, aber keinerlei Prozesse existieren und eine Reihe von händischen Arbeitsschritten erfolgen. Gerade bei diesen händischen Aktivitäten können Fehler entstehen und es geht viel Zeit verloren für Schritte, die man immer wieder gleich durchführt.

Darüber hinaus existieren individuelle Monopole, wenn nur ein einzelner DBA in der Lage ist, Änderungen auf den Datenbanken durchführen zu können. Selbst wenn solche Aufgaben auf mehrere Personen verteilt sind, müssen die Verbindungsdaten und vorhandenen Umgebungen oder sonstige Informationen ausgetauscht werden, die oft genug nicht allen Beteiligten bekannt sind.

Wäre es in solchen IT-Bereichen nicht wünschenswert, wenn Installationen über eine einfache Oberfläche anzustoßen wären und gleichzeitig für Sicherheit und Transparenz in den Prozessen gesorgt wird? Genau hier setzt der Ansatz von DevOps mittels Automatisierung an.

DevOps, CI/CD und Automatisierung

Der Ausdruck „DevOps“ steht dabei für die Einrichtung einer Kette von Prozessen, die auch als „Deployment-Pipeline“ bezeichnet wird, in der jeder Schritt von der Software-Entwicklung bis hin zur finalen Installation auf der Produktion abgebildet ist. Ein Bestandteil davon ist der Ansatz von Continuous Integration (CI) und Continuous Delivery (CD), der beschreibt, dass man durch immer wieder automatisiert durchgeführte Installations- und Auslieferungsprozesse die Qualität einer Software steigern kann. Maßgeblich dafür ist die Automatisierung, da man die immer wieder durchzuführenden Schritte bei einer Installation nur schneller erledigen kann, wenn diese automatisiert sind. Zusätzlich erhöhen öfter durchgeführte Installationen die Qualität der Software, da Fehler früher beziehungsweise öfter auffallen und sich damit die Wartezeit verringert, um einen einzelnen Sachverhalt produktiv zu nehmen.

Die Transparenz

Ein großes Problem vieler Datenbank-Anwendungen ist die fehlende Transparenz, wenn Änderungen erfolgen. Sind Entwicklungsschritte nicht automatisiert oder werden von einzelnen Personen durchgeführt, ist oft den restlichen Personen nicht klar, um welche Veränderungen es sich handelt. Tritt beim Einspielen von Datenbank-Änderungen ein Fehler auf, muss oft die ganze Datenbank zurückgespielt werden, was Zeit und damit Geld kostet. Fehlt dann noch die Person, die die Änderungen durchgeführt hat, ist es zudem schwer, den Grund des Fehlers herauszufinden.

Als Lösung dieses Problems dient der Einsatz eines Versionierungssystems wie Subversion oder Git. Wurden die Änderungen bisher direkt von einer Entwicklungs- in eine Test- oder Produktiv-Umgebung gespielt, so werden nach Einführung von CI/CD alle Änderungen auf der Entwicklungs-Umgebung durchgeführt und danach als Skripte in der Versionierung festgehalten.

Damit ist transparent, welche Person an welchem Tag an welchen Objekten welche Änderungen durchgeführt hat. Zudem besteht die Möglichkeit, Änderungen an einzelnen Objekten auch über längere Zeiträume zu verfolgen. Gerade dieser Sachverhalt hilft enorm, um bei Fehlern zu beurteilen, seit wann ein Bug im System vorhanden ist. Außerdem bietet es Entwicklern, die neu in einem Projekt sind, die Möglichkeit, die ursprünglichen Entwickler herauszufinden und dort nachzufragen, warum eine bestimmte Änderung durchgeführt wurde und welchen Zweck diese hatte.

Unterschiede zur Frontend-Entwicklung

Wer sich schon mit der Versionierung im Backend auseinandergesetzt hat, wird feststellen, dass es einige maßgebliche Unterschiede zur Frontend-Entwicklung gibt. Während im Frontend in den meisten Fällen die gesamten aktuellen Sourcecodes zum Aufbau einer Anwendungsversion benutzt werden, ist dies im Backend anders. Existiert beispielsweise eine Tabelle bereits, kann logischerweise das Skript zur Erstellung dieser Tabelle nicht

nochmals ausgeführt werden. Genauso verhält es sich mit Skripten zur Anpassung von Daten. Eine Migration von Daten kann nach einmaligem Einspielen meist kein zweites Mal laufen; dies ist maßgeblich davon abhängig, dass die Tabellen, in denen die Daten liegen, genau in der Struktur vorhanden sind, wie sie in SQL abgefragt beziehungsweise aktualisiert werden. Um diesen Problemen zu begegnen, gibt es zwei mögliche Lösungsansätze: Entweder man erstellt pro neuer Version einer Anwendung ein neues Datenbank-Schema, in dem alle Objekte neu erstellt und die vorhandenen Daten migriert werden, oder man sorgt dafür, dass nur die Skripte ausgeführt werden, die notwendig sind.

Den ersten Ansatz findet man oft bei fertigen Produktlösungen wieder, bei denen für ein Haupt-Release ein neues Schema erstellt wird und alle vorhandenen Daten nach Installation der Objekte per Migrationsskript überführt werden. Der zweite Ansatz wird in der Regel bei Projekten gewählt, die individuell entwickelt werden. Aus Sicht des Autors liegt das daran, dass, während ein fertiges Produkt oft von verschiedensten Vorversionen her zur aktuellen Version innerhalb einer Installation gebracht wird, die Individuallösungen so stark verzahnt sind, dass ein Schemawechsel kritisch und zu aufwändig wäre.

Wählt man nun den zweiten Weg, gibt es verschiedene Ansätze, dafür zu sorgen, dass nur die richtigen Skripte beziehungsweise die Skripte der passenden Version ausgeführt werden. Einerseits kann pro Skript eine Versionszuordnung erfolgen oder es werden innerhalb der Versionierung nur gezielt die Objekte für eine Version markiert, die entsprechend relevant sind.

Ein weiteres Problem innerhalb der Backend-Entwicklung ist die Abhängigkeit zwischen Objekten. Die Skripte einer Installation können nicht in beliebiger Reihenfolge erfolgen, da eine neue Spalte an einer Tabelle nur befüllt werden kann, wenn das Skript zur Erweiterung der Tabelle vor dem DML-Skript ausgeführt wird.

Definition der Prozesse

Sind die Probleme im Skripting und mit den Abhängigkeiten gelöst, gilt es als Nächstes, alle Prozesse zu identifizieren, die bisher von Hand beziehungsweise nacheinander durchgeführt wurden. Dabei sind Schritte wie das Zusammenbauen einer Installationsdatei, die Durchführung der Installation sowie anschließende Tests als ein Prozess zu identifizieren. Neben den Prozessen zur eigentlichen Installation können im Backend-Bereich auch optimal Export- und Import-Prozesse wie Datapump oder die Erstellung einzelner Schemata bis hin zur Vervielfältigung von PDBs definiert sein.

Die Oberfläche

Nachdem alle einzelnen Prozesse definiert sind, werden diese in eine Oberfläche integriert, die es jeder Person, die berechtigt ist, ermöglicht, den jeweiligen Prozess auszuführen. Das momentan wohl verbreitetste Build-Tool an dieser Stelle ist Jenkins. Dabei handelt es sich um eine webbasierte Oberfläche, mit der man per Mausklick Prozesse starten kann. Ein Prozess (in Jenkins als „Job" bezeichnet) kann dabei beliebige Operationen ausführen, was im einfachsten Fall der Batch-Prozess ist, den der DBA vorher von Hand aufgerufen hat.

Dabei ist der Vorteil, dass in einem solchen Job eine Zielumgebung definiert ist oder vom Benutzer ausgewählt werden kann, ohne dass alle genauen Verbindungsparameter oder technische Details bekannt sein müssen. Dadurch ist es auch einem Tester, der kein Datenbank-Knowhow besitzt, möglich, eine Testumgebung mit einer bestimmten Version einzuspielen. Zusätzlich werden die Verbindungsparameter nicht an jede beliebige Person verteilt und durch die genau aufgeführten Befehle im jeweiligen Job sind die einzelnen Arbeitsschritte dokumentiert und transparenter für die Personen, die mit der Entwicklung und den Skripten zu tun haben.

Der Mehraufwand

Wer im ersten Moment vor den Änderungen zurückschreckt und einen gefühlt sehr hohen Mehraufwand scheut, dem sei gesagt, dass die Einführung von DevOps beziehungsweise CI/CD nicht immer alle bisherigen Arbeitsschritte auf einmal ablöst. Oft kann ein Einstieg darüber erfolgen, dass man mit einem Jenkins Server einen Export und Import per Web-Oberfläche oder ein, von der eigentlichen Installation getrennten Prozess, vereinfacht zur Verfügung stellt. Zusätzlich sollte man sich vor Augen führen, wie viel Aufwand es im Normalfall bedeutet, gleiche Schritte immer und immer wieder von Hand durchzuführen.

In erster Linie ist die Test-Automatisierung zu nennen, die nach jeder noch so kleinsten Änderung eine Anwendung komplett von vorne bis hinten testen kann, ohne dass durch den eigentlichen Test ein Mehraufwand entsteht. Dies liegt daran, dass sich jeder Prozess (Job) auch jede Nacht automatisiert durchgeführt lässt, ohne dass auch nur eine Person involviert ist. Schlagen solche Tests dann fehl, können beispielsweise alle Entwickler morgens eine Benachrichtigung darüber erhalten, welche Änderung Fehler hervorgerufen hat. Zudem lassen sich dadurch auch die Aufwände für die eigentlichen Tester reduzieren, da der Installationstest schon vor dem eigentlichen Test erfolgen kann, ohne dass ein Tester aktiv werden muss.

Die Praxis

Setzt man nun eine solche Automatik in der Praxis ein, kann dies zum Beispiel so aussehen: Man exportiert den aktuellen Stand der Produktion per Datapump, kopiert die erzeugten Dateien danach auf die Testumgebung, löscht dort die vorhandenen Daten und importiert danach die aktuellen Daten.

Von Hand kann eine solche Operation schnell fünfzehn Minuten betragen. Der gleiche Vorgang, über Jenkins gestartet und alle Schritte automatisiert durchgeführt, benötigt gerade einmal ein paar Sekunden. Zusätzlich kann eine solche Operation natürlich komplett automatisiert nächtlich erfolgen, sodass auf der Testumgebung in jeder Nacht der aktuelle Produktionsstand bereitgestellt wird. In einem Jahr würde man so einen Gesamtaufwand von acht Tagen einsparen. Daneben entsteht auch ein qualitativer Vorteil, da durch den täglichen Abzug der Daten die Testfälle eine sehr große Abdeckung der Produktivdaten erreichen.

Vor allem in Systemen, die über einen langen Zeitraum betrieben werden,

ist eine hohe Anzahl von Daten-Konstellationen vorhanden, die ansonsten nur selten auf der Testumgebung zur Verfügung stehen. Einerseits benötigt man keine komplexen Mechaniken, um Daten zu erzeugen, und andererseits entsprechen die Daten zu großen Teilen der Realität, was die Testergebnisse wesentlich aussagekräftiger macht.

Ein weiteres Szenario ist der Einsatz von Automatisierung beim Zusammenbauen komplexer Patches, die von mehreren Personen entwickelt wurden. Bei einem Team von zehn Entwicklern, die wöchentlich eine Auslieferung bereitstellen, könnte es ohne solche Techniken dazu kommen, das man vier Tage entwickelt und einen ganzen Tag benötigt, um einen solchen Patch zu synchronisieren.

Setzt man nun eine Automatisierung ein, die täglich auf einem Installations-Testsystem einen solchen Patch mehrmals zusammenbaut und installiert, lassen sich Fehler sukzessive beheben. Man würde jeweils sofort nach den Änderungen einer Person den jeweiligen Patch testen und könnte einen eventuellen Fehler sofort beheben. Auch Seiteneffekte würden früher zu Tage treten und der Auslieferungstag selbst würde nicht zum ewigen Martyrium werden. Durch die Automatisierung wäre jeder Entwickler etwa zwei Stunden in der Woche mit der Fehlerbehebung beschäftigt. Bei zehn Entwicklern wäre das jede Woche ein Volumen von fast sieben Tagen an Zeitersparnis.

Fazit

Auch wenn DevOps, CI/CD und Automatisierung oft mit Tools wie Docker, Vagrant, Micro Services, Chef und Infrastructur as Code in Verbindung gebracht werden, kann man die Thematik auch Stück für Stück einführen, ohne die bisher von Hand geschriebenen Skripte komplett abzulösen. Zudem ist es üblich, dass in der gleichen Firma die Backend- und Frontend-Entwicklung getrennt voneinander erfolgt. Ist dann das Frontend-Team schon mit einem Jenkins Server und Versionierung unterwegs, können sich die Backend-Entwickler an der vorhandenen Infrastruktur beteiligen und man bekommt eine Prozesskette zur Installation einer neuen Umgebung.

Dieser Artikel deckt nicht den kompletten Technologie-Stack ab, sondern hilft vielmehr, innerhalb der oft monolithischen Datenbank-Entwicklung die Geschwindigkeit zu erhöhen und ein Gefühl dafür zu schaffen, was in der Backend-Entwicklung zu optimieren ist.

Oliver Lemm
oliver.lemm@mt-ag.com

WebLogic-WLST-Programmierung unter Eclipse mit dem Oracle-OEPE-Plug-in und PyDev

Michael Schulze, OPITZ CONSULTING GmbH

Eine Eclipse-IDE unter Einsatz des Oracle-OEPE-Plug-ins lässt sich so konfigurieren, dass diese Umgebung für die Entwicklung und das Testen von WLST-Skripten im Oracle-WebLogic-Umfeld dient. Die dabei entstehende Testumgebung ist Basis für einige WLST-Beispiele.

In diesem Artikel entstehen aus Eclipse lokale WebLogic-Domains unter dem Einsatz von WLST-Offline-Skripten. Diese Domains werden in der nächsten Ausgabe dann sukzessive ergänzt. Für das weitere Verständnis zunächst ein paar Grundlagen zu den eingesetzten Tools.

Eclipse

Eclipse ist eine grafische Open-Source-Entwicklungsumgebung, die das Ziel verfolgt, ein universelles Werkzeug für die Entwicklung bereitzustellen. Sie basiert im Kern auf Java, die Entwicklungssprache ist jedoch unabhängig und lässt sich durch Plug-ins erweitern. Eclipse wurde im Jahr 2001 von IBM ins Leben gerufen und ist sehr populär *[1]*.

OEPE

Das Oracle Enterprise Pack for Eclipse (OEPE) liefert zertifizierte Tools für die bekannte Eclipse-IDE. Diese lassen sich insbesondere für die Entwicklung mit Oracle-Produkten einsetzen. OEPE ist frei erhältlich und steht im Oracle Tech Network (OTN) zum Download zur Verfügung. Enthalten sind aktuell unter anderem folgende Werkzeuge *[2]*:

- Tools für das Mobile-Application-Framework (MAF)
- Tools für das Application-Development-Framework (ADF)
- Unterstützung für den Application Ser-

ver, etwa WebLogic (auch für WLST) und GlassFish
- Tools für Cloud Computing: Oracle Cloud (DB as a Cloud), Java Cloud Service
- Tools für Datenbanken: Database Explorer, Schema Viewer, SQL-Editor

OEPE liegt aktuell in der Version 12.2.1.5 (Stand: 12/2016) vor und Oracle bietet es zum einen als komplette (vorkonfigurierte) Eclipse-Entwicklungsumgebung (Windows, Linux, MacOS) oder über ein Repository als separates Plug-in an.

Mittels Plug-in lässt sich eine bestehende Eclipse-Umgebung um die oben genannten Tools erweitern. OEPE kann unter *„http://www.oracle.com/technetwork/developer-tools/eclipse/downloads/index.html“* heruntergeladen werden. Unter *„http://download.oracle.com/otn_software/oepe/12.2.1.5/neon/repository“* steht das Repository für die nachträgliche Integration des OEPE-Plug-ins in eine bestehende Eclipse-Umgebung.

Für die Erstellung der Testumgebung sind hier insbesondere die in OEPE integrierten WebLogic Server und Scripting-Tools relevant, mit denen es möglich ist, aus Eclipse heraus mit WebLogic Servern zu kommunizieren und WLST über Jython in Verbindung mit einer integrierten Konsole zu nutzen. Außerdem lassen sich Verbindungen des WebLogic Servers definieren, über die auf die MBeans des Servers zugegriffen werden kann *[3]*.

Jython und WLST

Jython ist eine Implementierung der objektorientierten Sprache Python, die für die Ausführung auf der Java-Plattform entwickelt wurde *[4]*. In WebLogic 12.2.1 wird Jython in der Version 2.2.1 verwendet; *Listing 1* zeigt, wie man dies ermittelt.

Das WebLogic Scripting Toolkit (WLST), basierend auf Jython (unter Verwendung von spezifischen WebLogic-Bibliotheken), ist das zentrale Werkzeug für die Automatisierung per Skript im WebLogic-Umfeld (Administration, Konfiguration und Monitoring von WebLogic Servern). Es verfügt über ein Commandline Interface (CLI), das interaktives Arbeiten ermöglicht, und kann im Online- oder Offline-Modus betrieben werden. Ein Beispiel für den Offline-Modus wäre eine automatische Domain-Erstellung per Skript, bei der kein Connect zu einem WebLogic Server erforderlich ist. Im Online-Modus greift man direkt auf den WebLogic Server oder den Node-Manager zu und kann damit zum Beispiel administrative Tätigkeiten in WebLogic-Umgebungen per Skript realisieren. WLST führt dabei Python Source Code innerhalb einer JVM aus *[5]*.

```
$ java -jar $ORACLE_HOME/oracle_common/util/jython/jython.jar -version
# Output:
Jython 2.2 on java
```

Listing 1

```
unzip eclipse-inst-linux64.tar.gz
$HOME/eclipse/eclipse-install/eclipse-install
```

Listing 2

Abbildung 1: Eclipse Installation und Setup

PyDev

PyDev ist eine Erweiterung der Eclipse-Umgebung. Das Plug-in stellt eine komplette Python-IDE für die Entwicklung mit Python, Jython und IronPython bereit. Die Umgebung *[6]* bietet verschiedene Vorteile, darunter:

- Syntax Highlighting
- Code Completion
- Type Hinting
- Code Analysis
- Refactoring
- Debugging
- Interactive Console
- Code Coverage

Mit der Installation des OEPE 12.2.1.5 wird eine ältere, kompatible Version des PyDev (2.7.5) ausgeliefert. Sie dient als Grundlage für die WLST-Programmierung unter Eclipse. Die aktuelle PyDev-Version (5.5) ist inkompatibel und kann daher leider nicht verwendet werden.

Die Installation der WLST-Testumgebung

Für die Installation sind zunächst diese technischen Voraussetzungen zu erfüllen:

- Linux-Desktopsystem (OEL) , 4 GB RAM, 2 Prozessoren

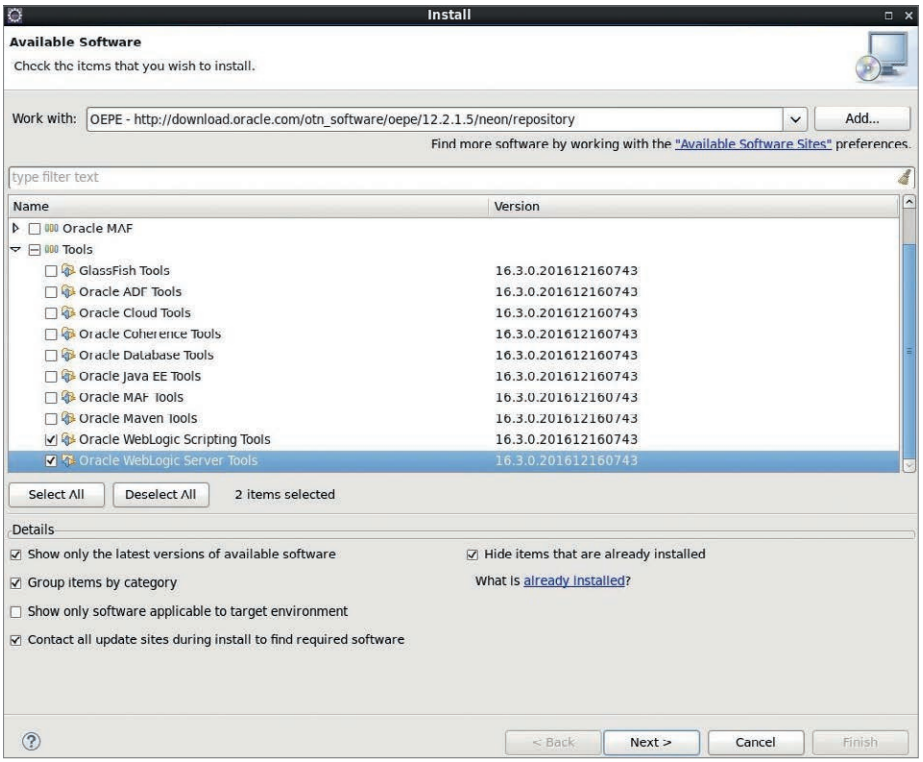


Abbildung 2: OEPE-Plug-in-Installation

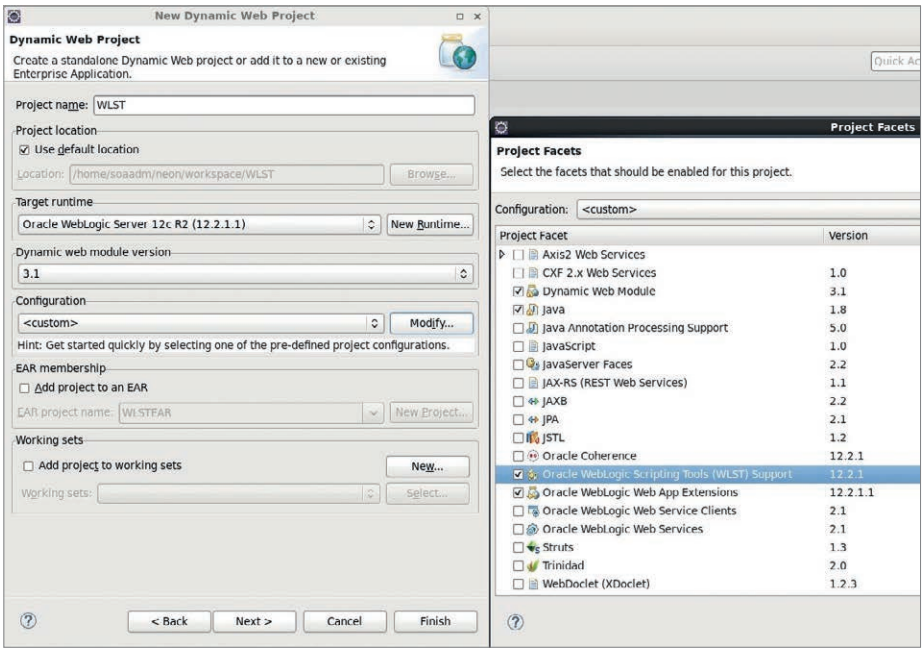


Abbildung 3: Facet-Auswahl des Oracle WebLogic WLST

```
# Zeile 56
set('ListenAddress','192.168.50.91')
# Zeile 143
writeDomain(beaHome + '/testdomain122110')
```

Listing 3

- Installation von Java 8 SDK (Linux x64)
- Installation der WebLogic-Server-Software 12.2.1.1.0 (Infrastruktur)
- Installation der WebLogic-Software 12.2.1.2.0 (Infrastruktur)
- Einbindung aller festgelegten IP-Adressen mittels IP-Bonding

Eclipse installieren

Ist das System wie beschrieben vorbereitet, folgt die Installation der Eclipse-IDE, die als Grundlage für die zu erstellende WLST-Testumgebung dient. Beim Software-Download hilft ein Installer, der in der Setup-Phase der Installation konfiguriert werden kann *[7]*. Die bezogene Zip-Datei wird beim Installationsvorgang zunächst auf das System nach „$HOME/eclipse“ transferiert und dann in zwei Schritten installiert *(siehe Listing 2)*.

In der Setup-Phase empfiehlt es sich, J2EE zu wählen, es enthält alle wesentlichen Komponenten für die J2EE-Entwicklung *(siehe Abbildung 1)*. Darüber hinaus bietet es auch eGit als Versionsverwaltungstool, das später auch noch eine Rolle spielen wird.

Das OEPE-Plug-in installieren

Für die Installation des OEPE-Plug-ins in Eclipse sind folgende Schritte auszuführen:

- Im Menü unter „Help/Install New Software“ den Button „Add“ betätigen
- Einen Namen festlegen
- Einen Repository-Eintrag definieren
- „*http://download.oracle.com/otn_software/oepe/12.2.1.5/neon/repository*“ aufrufen
- Mit „OK“ bestätigen

Es dauert einen Moment, bis die Repository-Einträge übertragen wurden. Danach sind diese Punkte zu aktivieren *(siehe Abbildung 2)*:

- Tools
- Oracle WebLogic Scripting Tools
- Oracle WebLogic Server Tools

Nach Bestätigung der Einstellungen und der License Agreements wird das OEPE-Plug-in nun mit allen Abhängigkeiten

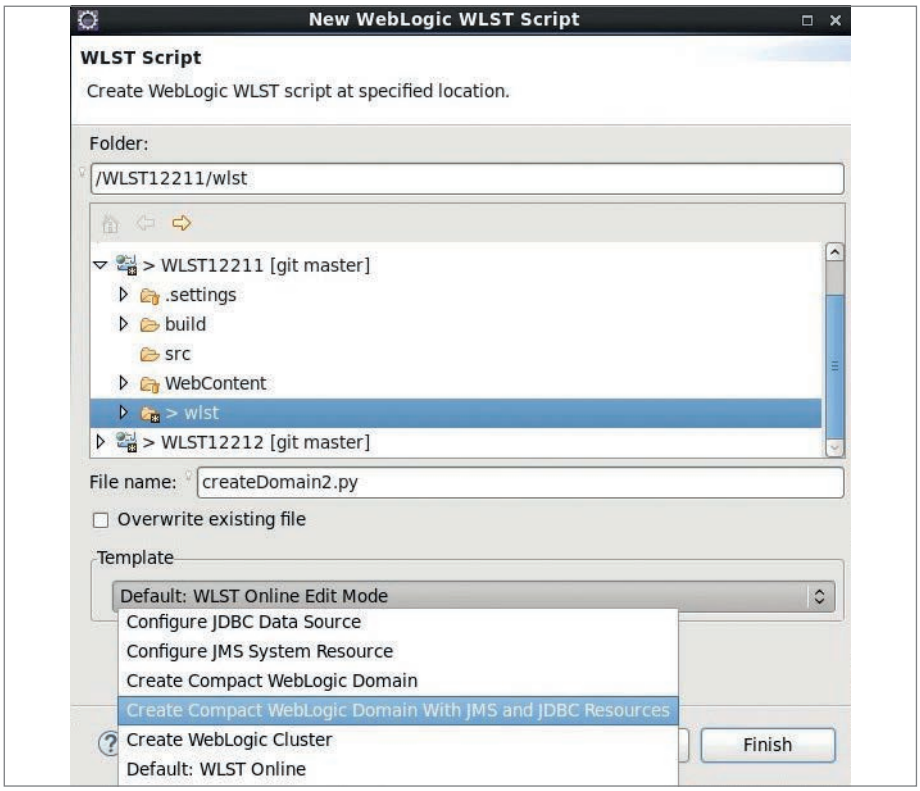


Abbildung 4: Wahl des WLST-Templates

```
# starten der Domain
$DOMAIN_HOME/bin/startWebLogic.sh &
# ermitteln ob RUNNING
grep "RUNNING" $DOMAIN_HOME/servers/AdminServer/logs/AdminServer.log
```

Listing 4

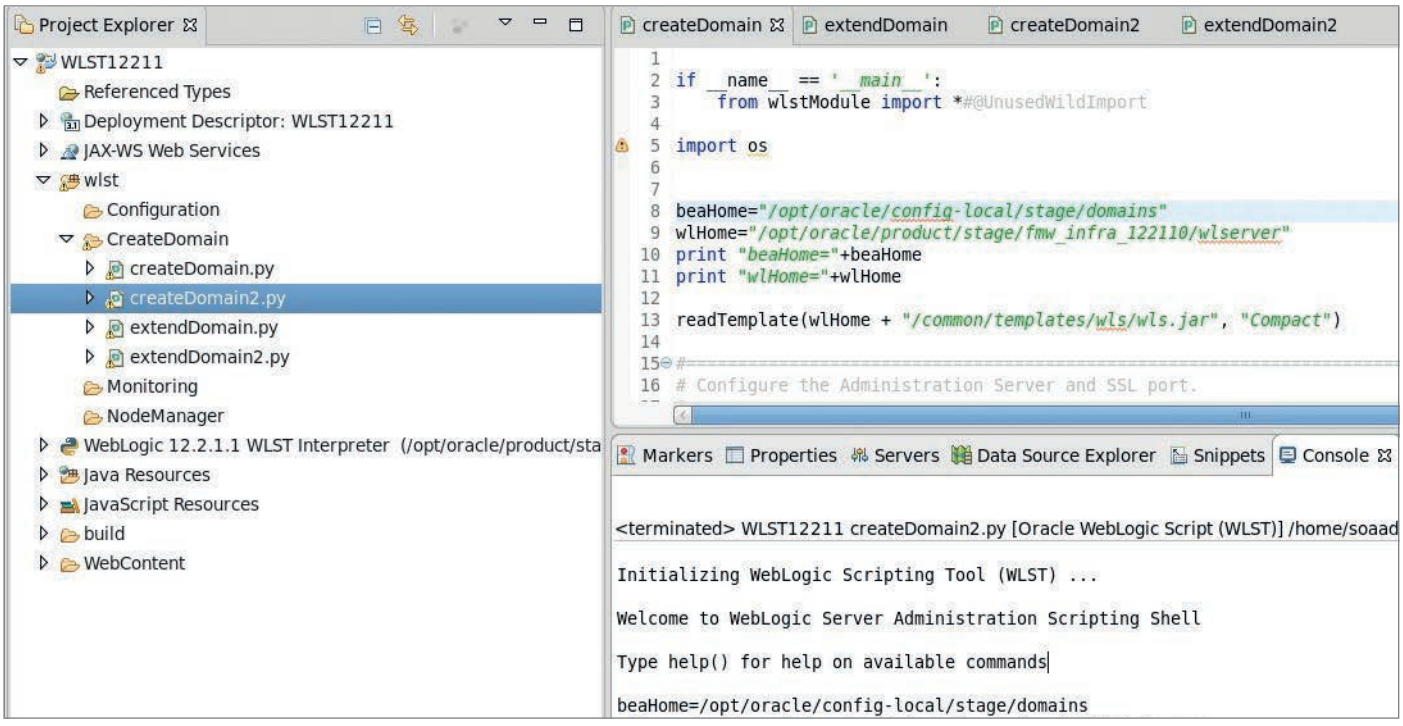


Abbildung 5: Create a domain (OFFLINE)

installiert. Das eingangs beschriebene Plug-in PyDev wird in diesem Prozess automatisch in der Version 2.7.5 mit installiert. PyDev ist die Grundlage für die Entwicklung von WLST-Modulen und hat sich als Python-Entwicklungsumgebung im Eclipse-Umfeld mit vielen Vorteilen etabliert. Nach einem Neustart von Eclipse sind nun alle nötigen Voraussetzungen installiert.

Ein erstes (noch leeres) WLST-Projekt erstellen

Der Projekt-Wizard erstellt ein neues Eclipse-Projekt. In Eclipse ist dazu im Menü unter „Datei/New/Project/Web" die Auswahl „Dynamic Web Project" erforderlich. Im nächsten Schritt ist ein Projektname zu vereinbaren und über den Button „New Runtime" die gewünschte Server Runtime auszuwählen (Oracle/„Oracle WebLogic Server").

Im weiteren Dialog werden das WebLogic-Home-Verzeichnis der vorab installierten WebLogic-Software sowie das Java-Home-Verzeichnis angegeben. Mit diesen Angaben können die richtige WebLogic-Version ermittelt und die benötigte Runtime installiert werden (in diesem Fall

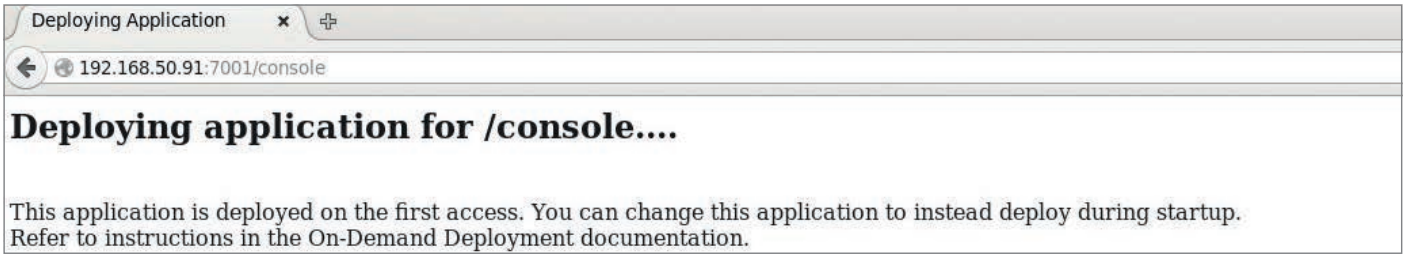


Abbildung 6: Admin-Konsole nach Startup

```
# Zeile 46
set('ListenAddress','192.168.50.100')
# Zeile 153
writeDomain(beaHome + '/testdomain122120')
```

Listing 5

```
$DOMAIN_HOME/bin/startWebLogic.sh &
# ermitteln ob RUNNING
grep "RUNNING" $DOMAIN_HOME/servers/AdminServer/logs/AdminServer.log
```

Listing 6

WebLogic 12.2.1.1). Ist dies abgeschlossen, aktiviert man noch die WLST-Funktionalität, diesmal über den Button „Modify". Hier ist der Eintrag „Oracle WebLogic Scripting Tools (WLST) Support" zu markieren. „Finish" schließt den Prozess ab. Im Projekt-Explorer erscheint jetzt das Projekt inklusive des enthaltenen WLST-Zweigs. Dort kann nun mit der Entwicklung gestartet werden *(siehe Abbildung 3)*.

WLST-Beispiel 1: Erstellen einer lokalen WebLogic-Domain im Offline-Modus

Folgende Schritte werden in diesem Beispiel ausgeführt:

- Erstellen einer WebLogic-Domain mittels WLST-Offline-Skript
- Starten der WebLogic-Domain
- Checkup über Admin-Konsole

Zunächst wird im erstellten Eclipse-Projekt unterhalb des WLST-Zweigs eine neue WLST-Datei erstellt; dazu mit der rechten Maustaste unter „New"/„Oracle WebLogic Script" auswählen. Es öffnet sich ein Dialog, in dem ein Dateiname vergeben werden kann, zudem kann man hier aus verschiedenen, mit OEPE mitgelieferten Templates wählen. Wir verwenden zur Erstellung unserer Domain das Template „Create Compact WebLogic Domain with JMS and JDBC Resources" *(siehe Abbildung 4)*. Das Python-File erscheint nun im Editor und wird noch modifiziert, in diesem Fall um eine IP-Adresse und den Domainnamen *(siehe Listing 3)*.

Im Projekt-Explorer wird die Datei nun ausgeführt (rechte Maustaste „Run As"/„WLST Run"), es öffnet sich die integrierte WLST-Konsole, in der alle Interaktionen verfolgt werden können *(siehe Abbildung 5)*. Die Domain ist nun im Filesystem entstanden und kann gestartet werden *(siehe Listing 4)*. Es folgt ein kurzer Test über

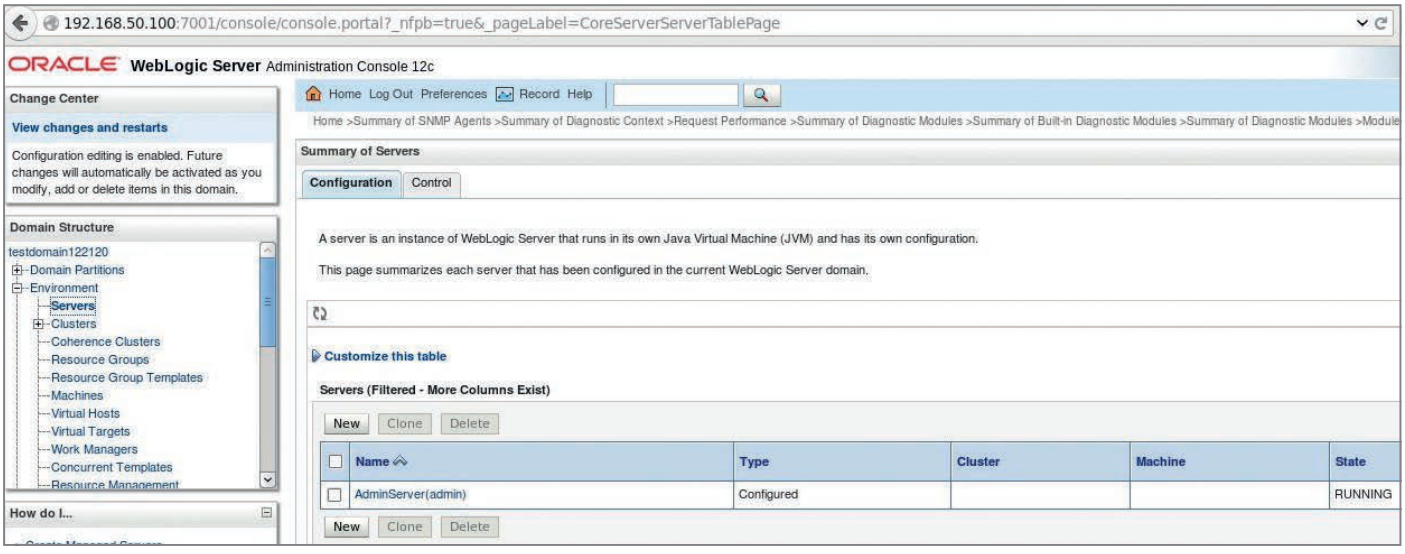


Abbildung 7: Admin-Konsole nach Domainstart (12.2.1.2)

die Admin-Konsole, diese wird beim ersten Aufruf zunächst eingerichtet *(siehe Abbildung 6)*.

Erstellen eines weiteren WLST-Projekts mit aktueller Runtime

Um zu demonstrieren, dass durchaus mit mehreren Server-Runtimes gearbeitet werden kann, wird nun (analog zum ersten Projekt) ein zweites erstellt. Dabei ist zu beachten, dass nun ein anderes WebLogic-Server-Homeverzeichnis verwendet werden muss, um mit der aktuellsten WebLogic-Version 12.2.1.2 arbeiten zu können. Hat alles funktioniert, gibt es nun ein neues Projekt, das wieder einen WLST-Zweig enthält, in dem nun analog zu Beispiel 1 eine Domain erstellt werden kann.

WLST-Beispiel 2: Erstellen einer weiteren WebLogic Domain

Nachfolgend wird das Skript aus dem zweiten Beispiel genutzt, um eine weitere Domain im Offline-Modus zu erstellen. Lediglich die Zeilen „IP-Adresse“ und „Domain-Name“ müssen hier noch angepasst werden *(siehe Listing 5)*.

Im Projekt-Explorer ist das Skript nun ausführbar (rechte Maustaste „Run As“/ „WLST Run“). Die Domain ist im Filesystem entstanden und kann gestartet werden *(siehe Listing 6)*. Es ist jetzt auch kontrollierbar, ob die Domain verfügbar ist *(siehe Abbildung 7)*.

Fazit

In diesem Artikel wurde eine WLST-Entwicklungsumgebung unter Verwendung der bekannten Eclipse-IDE mit dem Oracle OEPE-Plug-in erstellt. Diese eignet sich insbesondere für Test-Cases mit verschiedenen WebLogic-Versionen und Konfigurationen. In zwei ersten Projekten wurden WebLogic-Domains mithilfe von WLST-Offline-Skripten erzeugt. Abschließend gab es einen Überblick über die bisher erstellte Testumgebung.

In der nächsten Ausgabe werden wir in dieser Testumgebung mittels WLST-Online weitere WebLogic-Komponenten generieren. Dazu gibt es Beispiele für den praktischen Einsatz (Konfiguration, Monitoring etc.) und die Anbindung einer SOA-12c-Testdomain als Remote-Beispiel *(siehe Abbildung 8)*.

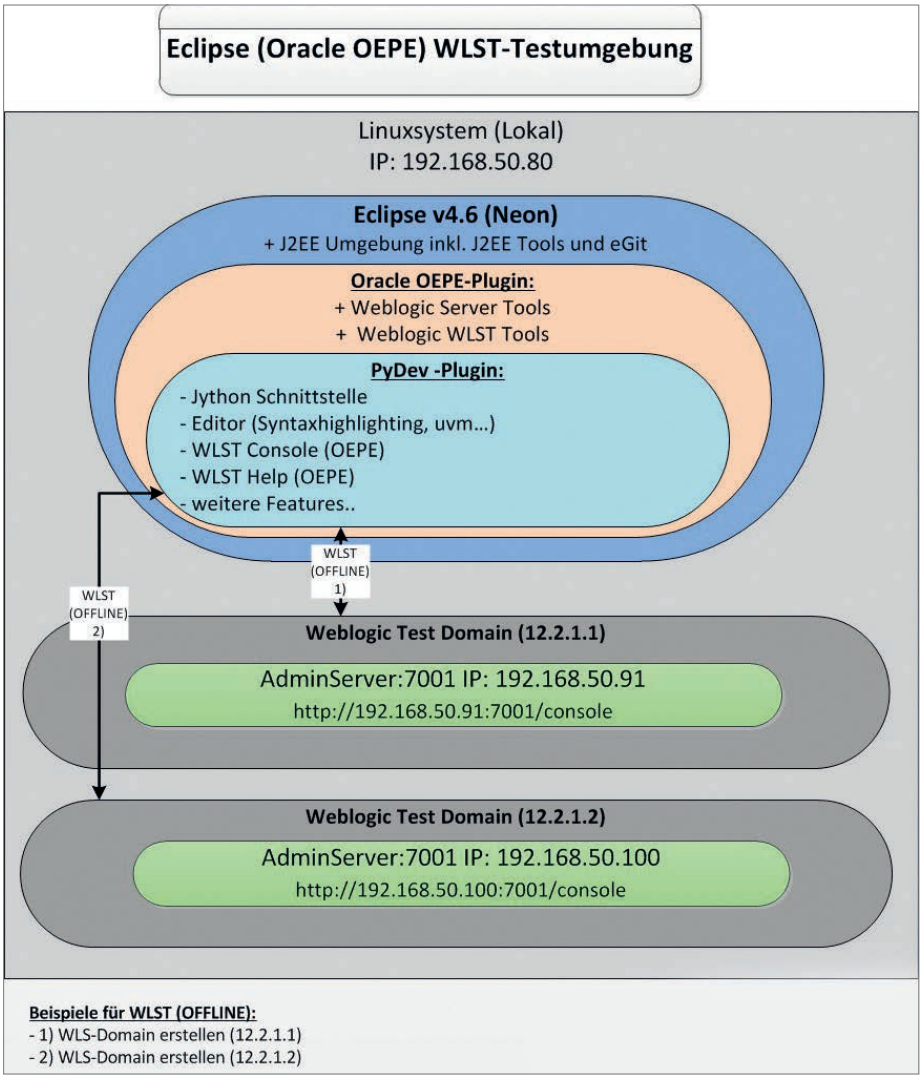


Abbildung 8: Überblick Testszenario

Quellen

[1] *http://java-seite.de/java-ide/eclipse-ide*
[2] *https://docs.oracle.com/middleware/oepe12215/oepe/develop/GUID-E2DE8F30-7657-46FC-AF95-1179EF5889D5.htm#OEPUG117*
[3] *https://docs.oracle.com/cd/E47843_06/12124/OEPUG/WebLogic.htm#A1181630*
[4] *http://www.jython.org/archive/21/docs/whatis.html*
[5] *https://docs.oracle.com/cd/E24329_01/web.1211/e24491/using_wlst.htm#WLSTG118*
[6] *http://www.pydev.org/manual_adv_features.html*
[7] *https://www.eclipse.org/downloads/download.php?file=/oomph/epp/neon/R2a/eclipse-inst-linux64.tar.gz*

Michael Schulze
michael.schulze@opitz-consulting.com

Der Wunsch nach mehr Agilität: Microservices und Container

Mario Herb und Matthias Fuchs, esentri AG

Im Rahmen der fortschreitenden Digitalisierung werden an digitale Service-Angebote und die dafür benötigten IT-Systeme und Applikationen immer höhere qualitative Anforderungen gestellt.

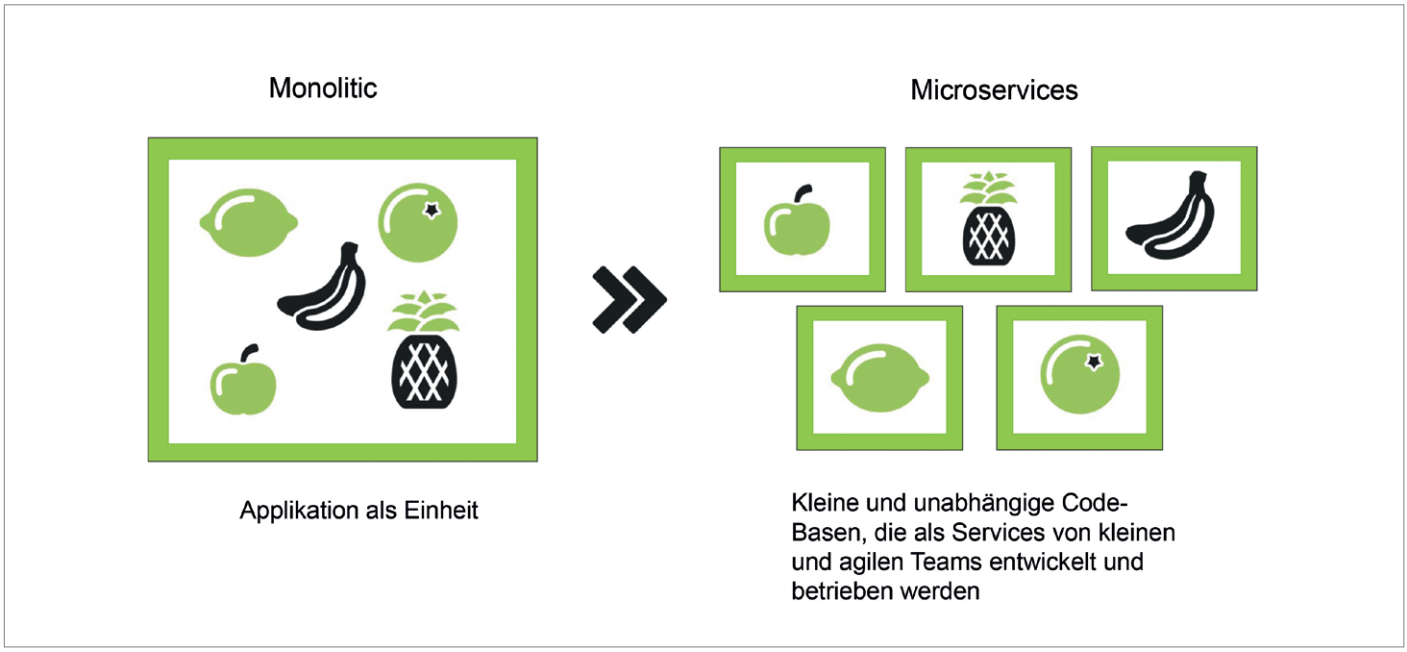


Abbildung 1: Monolith vs. Microservices

Verfügbarkeit rund um die Uhr, Zero-Downtime, dynamische Skalierbarkeit und eine hohe Robustheit (transparenter Failover, selbstheilende Systeme) sind keine utopische Vorstellung mehr, sondern prägen bereits die Erwartungshaltung der Endbenutzer. Dabei muss der Service oft in nahezu Echtzeit (oder zumindest mit direkter Rückmeldung an den Benutzer) bei jederzeit gleichbleibend hoher Qualität erfolgen. Hinzu kommt eine weitere wesentliche Herausforderung, die an die zuständigen Entwicklungsteams gestellt wird: Das Service-Angebot muss jederzeit möglichst schnell und flexibel angepasst werden können. Man spricht gemeinhin von einer möglichst hohen Agilität. Das Business wandelt sich ständig und die IT muss der Treiber und nicht der Hemmschuh des Wandels sein. Der Artikel zeigt, wie Microservices in Verbindung mit Container-Technologien eine optimale Basis bereitstellen.

Der klassische Weg besteht darin, die Applikation als eine einzige Einheit (Monolith) zu entwickeln und bereitzustellen. Je größer und komplexer Applikationen sind, desto mehr Menschen sind in die Entwicklung involviert, und je mehr Änderungen an verschiedenen Stellen der Applikation vorgenommen werden müssen, desto komplizierter wird dieser Weg. Vor allem in Bezug auf Änderungen ergeben sich somit oft langläufige Release-Zyklen. Kontinuierliche Updates erfordern hierbei, durch die sehr enge Kopplung diverser System-Komponenten innerhalb des Monolithen, einen exponentiell wachsenden Kommunikations- und Abstimmungsaufwand. Updates sind somit teuer und werden seltener durchgeführt. Dies steht ganz im Gegensatz zum „Continuous Delivery"-Paradigma. In der heutigen Zeit ändern sich Geschäftsanforderungen fortwährend. Das digitale Angebot muss ständig angepasst und erweitert werden. Lange Release-Zyklen verbunden mit Qualitätsproblemen kann man sich bei Cloud-getriebenen Digitalisierungsservices heutzutage nicht mehr leisten.

Microservices to the rescue

Große Serviceanbieter wie Amazon und Netflix haben diese Problematik früh erkannt. Sie entschieden, ihre Systemwelt in diskrete Geschäftsfunktionen zu unterteilen. Kleine und unabhängige Code-Basen werden als Services von kleinen und agilen Teams entwickelt und betrieben. Solange die Kompatibilität bezüglich der verbindenden APIs erhalten bleibt, können diese Teams ihre Services unabhängig voneinander freigeben. Man kann dies als „Divide and Conquer"-Strategie bezüglich funktionaler und personeller Aufteilung in großen IT-Projekten auffassen *(siehe Abbildung 1)*.

Es kann dabei zu Szenarien kommen, bei denen ein Service von einem neuen Feature eines anderen abhängt. Die Services sollten jedoch so geschnitten sein, dass dies selten der Fall ist. Jedes Team ist für alle Phasen des Lebenszyklus seiner Services zuständig. Die Verantwortlichkeit ist somit klar definiert, es muss kein Know-how-Transfer zwischen einer Entwicklungsmannschaft und einem Betriebsteam erfolgen.

Microservices nur in Großprojekten?

Microservices müssen nicht nur in großen Szenarien sinnvoll sein. Es gibt diverse Berichte von kleineren Teams, die den Aufwand in Bezug auf Deployment-Automatisierung und Überwachung in einem Microservice-artigen Ansatz gegangen sind und dadurch Effizienzgewinne beim Betrieb und der Weiterentwicklung ihrer Systeme erzielen *(siehe „https://segment.com/blog/why-microservices")*.

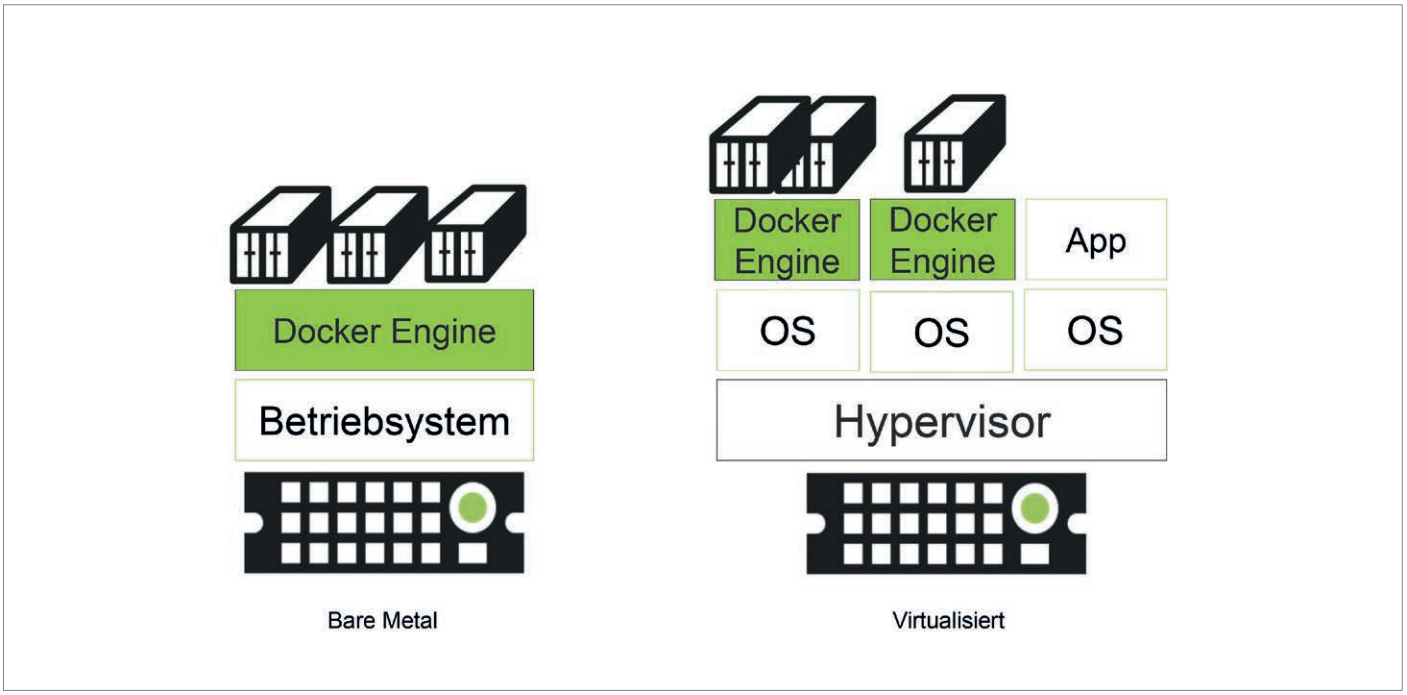


Abbildung 2: Docker und Virtualisierung

Applikationsseitige Lösungsansätze für Microservice-Querschnittsbelange

Bei der Verwaltung einer oder mehrerer Microservice-Instanzen gibt es immer wieder gleichartige Herausforderungen. Diese Querschnittsbelange lassen sich oft durch Funktionen der Container-Orchestration lösen oder müssen direkt auf Applikationsebene integriert sein. Dies ist je nach Anwendungsfall und Rahmenbedingungen zu entscheiden. In diesem Abschnitt werden kurz einige der applikationsseitigen, also nicht direkt in eine Container-Orchestration-Lösung integrierte Lösungsansätze vorgestellt. Unter anderem sind folgende Punkte zu berücksichtigen:

- Zentrale Konfiguration
- Service Discovery
- Load Balancing
- Error Management

Gerade bei einer Vielzahl von Services eignen sich Lösungen zur zentralen Verwaltung der Service-Konfigurationseinstellungen. Es erleichtert den Betrieb ungemein, die Konfiguration der Services entkoppelt von der Laufzeitumgebung zu halten und zu verwalten. Die nächste Anforderung stellt eine Möglichkeit für zentrales Service Discovery dar. Dabei wird die dynamische Adressierung von Service-Instanzen über Namen anstatt IP-Adressen ermöglicht.

Oftmals kombinieren/ergänzen Service-Discovery-Lösungen Ansätze zum Load Balancing, also um Service Calls auf mehrere infrage kommende Instanzen zu verteilen. Um die Fehleranfälligkeit eines verteilten Gesamtsystems gering zu halten, gibt es im Microservice-Universum mehrere Ideen. In verteilten Szenarien neigen Fehler dazu, kaskadierende Fehlermeldungen in der Service-Kette auszulösen. Diese sind schwerer nachzuvollziehen, wenn es an die Ursachenforschung geht. Als Verbesserung lassen sich querschnittliche Meta-Services mit Lösungen für Distributed Tracing und Distributed Logging einführen. Dabei werden Trace-Ausgaben miteinander korreliert und bei Bedarf mit entsprechenden Applikationen über die verschiedenen Service Calls hinweg analysiert.

Wenn eine der vielen Service-Instanzen sich gerade fehlerhaft verhält, sollte sie möglichst schnell aus dem Zugriff entzogen werden, sodass nur fehlerfreie Instanzen die aktuelle Last verarbeiten. Hier können diverse Implementierungen des Circuit-Breaker-Patterns (dt. „elektrische Sicherung“) helfen, die dafür sorgen, dass die zugehörigen Schnittstellen der Services überwacht und im Fehlerfall deaktiviert werden. Circuit Breaker tun dies automatisch und schalten Instanzen auch automatisch wieder zu, sobald sie sich erholt haben. Diese Ansätze überschneiden sich in einigen Bereichen mit dem Lösungsangebot von Container-Orchestrierungs-Plattformen. Eine Kombination aus beiden kann die optimale Lösung darstellen. Bekannte Ansätze im Java-/Open-Source-Umfeld sind zum Beispiel der Spring Cloud Stack oder Projekte wie Consul.io oder linkerd.

Ebenso sind kleine unabhängige Code-Basen mit klar definierten Interfaces einfacher zu testen, einzurichten und zu überwachen. Ein weiterer Vorteil besteht in der Wiederverwendbarkeit der Services, die durch diesen Ansatz erreicht wird. Außerdem sind klein geschnittene Services auf feingranularer Ebene effizienter zu skalieren, was alles zusammen zu einer höheren Kosteneffizienz führt.

Microservices haben dabei ihren Preis: Im Allgemeinen muss bei einer größeren Anzahl von einzurichtenden Services ein größerer Aufwand in die Deployment-Automatisierung gesteckt werden, um die erforderliche Agilität zu erzielen. Zudem gibt es nun mehrere Komponenten und Endpunkte, die im Betrieb des Gesamtsystems zu überwachen sind.

Ein in natürlicher Weise erzwungenes feingranulares Monitoring hat dabei aber auch wieder seine Vorteile. Üblicherweise kommen bei einem Microservice-Ansatz leichtgewichtige Protokolle und/oder Messaging-Mechanismen für die Kommunikation zwischen Service-Instanzen zum Einsatz. Dennoch hat dieser Ansatz auch Nachteile. Entfernte Aufrufe sind trotz leichtgewichtiger Protokolle deutlich teurer als Prozess-interne API-Calls. Deshalb sind entsprechende APIs oft weniger feingranular gestaltet und dadurch weniger intuitiv anzuwenden. Vor allem ist es wesentlich schwieriger, die funktionalen Verantwortlichkeiten zwischen Komponenten zu verändern oder zu verschieben, wenn dabei Prozessgrenzen überschritten werden.

Der reinen Lehre gehorchend, sollte jeder Service seine eigene Persistenzschicht besitzen. Dies ist notwendig, um Abhängigkeiten im Datenmodell entsprechend zu reduzieren und einen in sich geschlossenen Service abbilden zu können. Diese Vorgehensweise bietet weitere Vorteile in Bezug auf die Freiheit bei der Technologiewahl, was die Datenhaltung angeht. Jedoch sind damit Nachteile im Transaktionsmanagement verbunden. In aller Regel lassen sich damit keine verteilten Transaktionen realisieren, stattdessen muss programmatisch auf die Einhaltung der Gesamtkonsistenz geachtet werden. Gegebenenfalls muss beispielsweise über sogenannte „Compensation Actions“ die Gesamt-Konsistenz im Fall von Fehlern wiederhergestellt werden.

Bei einer gut geschnittenen Service-Architektur werden diese Nachteile durch die gewonnene Agilität gern in Kauf genommen. Zusammenfassend lässt sich sagen: Man investiert mit einem Microservice-Ansatz in Deployment-Automatisierung, Überwachung und die gesteigerte Komplexität eines verteilten Systems und gewinnt dafür unabhängige, Upgrade-fähige und austauschbare Komponenten sowie eine deutlich höhere Agilität und Wandlungsfähigkeit bei der Weiterentwicklung des Gesamtsystems. Die Reaktionsfähigkeit auf veränderte Business-Situationen bleibt damit durchgehend erhalten.

Die Container

Container bieten einige Vorteile gegenüber traditionellen Infrastruktur-Plattformen. Container kapseln applikationsseitig benötigte Binaries und Bibliotheken. Sie sorgen für Isolation der Prozesse und der Dateisystem-Struktur, zudem ist mit ihnen ein Schutz der verwendeten zentralen Ressourcen möglich (CPU und Memory Limits). Das klingt bisweilen nach ähnlichen Vorteilen, wie sie auch Virtualisierung bietet, dabei sind Container gemeinhin jedoch leichtgewichtiger. Container Images sind meist viel kleiner als die Images von Virtualisierungslösungen, sie booten oftmals schneller und im Allgemeinen werden durch den Einsatz von Containern deutlich weniger Ressourcen gebunden *(siehe Abbildung 2)*.

Durch diese Leichtgewichtigkeit tragen sie zu einer deutlich höheren Agilität sowohl bei der Entwicklung als auch beim Betrieb von verteilten System-Architekturen bei. Sie sind innerhalb von Sekunden einsatzfähig und können ebenso schnell auf mehrere Hosts verteilt werden. In vielen Szenarien wird jedoch aus Security-Gründen und vor allem im Sinne der Kapazitätsoptimierung Virtualisierung mit dem Einsatz von Containern kombiniert. Schließlich will jeder System-Administrator dafür sorgen, dass seine Ressourcen möglichst optimal ausgelastet sind.

Container und Microservices

Container-Technologien wie Docker ergänzen Microservices in vielen Punkten. Dennoch sei eines vorweggenommen: Es ist durchaus möglich, einen Microservice-Ansatz zu fahren, ohne Container als unterliegende Runtime-Technologie zu verwenden. Jedoch wird durch den Einsatz von Containern die gewünschte Agilität auch im Infrastruktur-Bereich gefördert. Die Unabhängigkeit der Services wird durch die zuvor beschriebene Isolation, die Container ermöglicht, auch auf Infrastruktur-Ebene gewährleistet. Das Management und Deployment einzelner Services ist durch Paketierung aller notwendigen Bibliotheken in einem Container stark vereinfacht.

Probleme aufgrund unterschiedlicher Betriebssystem- oder Bibliotheks-Versionen auf unterschiedlichen Stages (wie Entwicklung und Produktion) sollten somit der Vergangenheit angehören. Ebenso gewinnt man durch Kapselung ab Betriebssystemebene ein hohes Maß an Portierbarkeit. Services lassen sich so im Nu von A nach B umziehen.

Die Herausforderungen im Container-Umfeld

Die Herausforderungen im Einsatz von Containern, insbesondere in komplexen Szenarien mit vielen Applikationen und Containern, sind:

- Verteilung von Service-Instanzen über mehrere Hosts
- Die sogenannte „Location Transparency“, also dynamische Adressierung von Service-to-Service-Calls im Rahmen der Programmierung
- Effizientes dynamisches Load Balancing
- Effiziente Nutzung der zugrunde liegenden Rechenkapazität
- Persistenz und Datenhaltung

Die genannten Fragestellungen werden im Rahmen von Container-Orchestration-Frameworks angegangen. Eine zentrale Rolle spielen dabei der Scheduler und eine Administrationsplattform. Unter einem Scheduler versteht man ein Programm, das die Steuerung der einzelnen Prozesse übernimmt. Dieser startet und stoppt Services, überprüft die Verfügbarkeit und regelt Ressourcen in Bezug auf CPU und Memory.

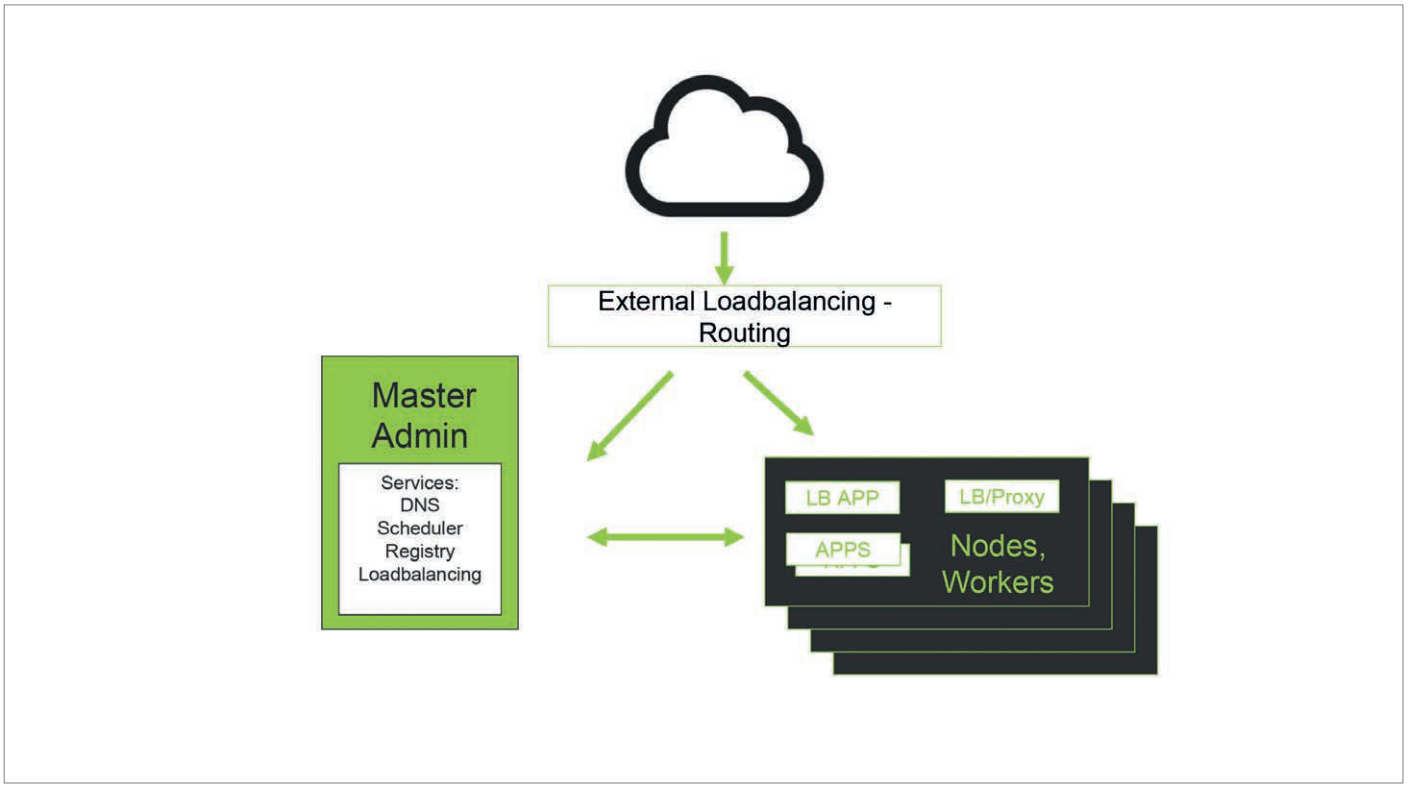


Abbildung 3: Container Stack

Container Scheduling

Die Steuerung von vielen Containern, also die Verteilung auf verschiedene Hosts und die Kontrolle der Verfügbarkeit, ist die Herausforderung bei der Verwendung von vielen Services, die auf viele Container verteilt sind. Das ist die Basis bei der Verwendung eines Container-Orchestration-Frameworks. Diese Funktion übernimmt ein Scheduler. Er ermöglicht es, die Ressourcen optimal zu nutzen und die Verfügbarkeit der Services sicherzustellen. Erste Ansätze von Schedulern gab es in Hadoop Clustern mit YARN (Yahoo), aber auch in den Anfängen von Kubernetes mit Omega (Google).

Eine optimale Verteilung der Ressourcen für alle Arten von verschiedenartigen Anwendungen ist mit einem Scheduler-Typ unmöglich. Jeder Scheduler hat somit unterschiedliche Ausprägungen. Orchestrations-Plattformen wie Kubernetes oder Docker Swarm verwenden einen eigenen Scheduler, Mesos kann unterschiedliche Scheduler wie Marathon oder Chronos benutzen *(siehe „https://medium.com/@ArmandGrillet/comparison-of-container-schedulers-c427f4f7421#.3ddqpmpr7“)*.

Zentrale Administration

Um die Verwaltungsanforderungen zu lösen, muss im Rahmen von Orchestrierungen eine zentrale Komponente eingeführt werden. Diese sind meist als „Master“ oder „Manager“ in den Frameworks bezeichnet. Somit bestehen alle Orchestrierungs-Plattformen aus zwei Teilen:

- Zentrale Master- oder Manager-Server sowie Admin-Stack
- Worker oder Agents

Der zentrale Admin-Stack verwaltet die Agents und Workers. Ein Scheduler steuert das Starten und Stoppen der Services beziehungsweise Container. Die Container werden auf den Agents oder Workers gestartet, der Admin hält alle Informationen zu Konfiguration und Systemstatus der Container. Ein zentraler Bestandteil des Admin ist somit der Scheduler; die Informationen des Schedulers sind in einer Konfigurationsdatenbank abgelegt.

Dieses Konzept wird sowohl bei Kubernetes und Mesos als auch bei Docker Swarm umgesetzt. Um die Verfügbarkeit und Entscheidungsfähigkeit der zentralen Administration bei Serverausfällen zu gewährleisten, müssen mindestens drei Server im Einsatz sein. Die Konfigurationsdaten werden auf einem Shared Storage oder einem Cluster-Configuration-System abgelegt. Als verteilte Datenbank für Konfigurationsdaten dienen meist Tools wie „etcd“ oder „zookeeper“.

Container Stack

Auf Basis der Prinzipien aus Admin und Scheduling haben sich im Rahmen der aufstrebenden Container-Lösunge noch weitere ergänzende Technologien herausgebildet, die den sogenannten „Container Stack“ bilden. Als Container-Basis-Technologie auf der Worker-Seite hat sich inzwischen Docker als De-facto-Standard etabliert. Nur Mesos kann auch native Programme ohne die Verwendung von Containern ausführen.

Ein Load Balancing ist auf Basis der Konfigurationsdatenbank leicht möglich. Entweder greift ein Service-Container direkt auf die Konfigurations-Datenbank (über „zookeeper“) zu, die Daten werden

Verwaltung von zustandsbehafteten Services über Container-Orchestration-Plattformen

Im einfachsten Fall sind die Services „self contained“ (also in sich geschlossen) und arbeiten zustandslos. Sie können relativ einfach verteilt und skaliert werden. Mittlerweile gehen die Fähigkeiten von Lösungen im Container-Orchestration-Bereich noch einen Schritt weiter. Bei zustandsbehafteten Services (wie Cassandra oder MySQL Cluster) gab es bisher folgende Herausforderung: Jeder Container muss seine Identität wahren, damit entsprechende verteilte zustandsbehaftete Systeme funktionieren können.

Die Daten eines „Shard“ (Teilmenge der Gesamtdaten, die unter Verwaltung eines bestimmten Knotens stehen) sollen beispielsweise bei Stopp- und Start-Aktionen der Container nicht verloren gehen. In einigen Container-Orchestration-Plattformen (wie Google Kubernetes, Red Hat Open Shift) ist dies inzwischen möglich. Oftmals wird hier die Analogie „pets vs. cattle“ herangezogen. Haustiere (engl. „pets“) haben in der Wahrnehmung ihrer Besitzer oft eine eigene Identität und benötigen eine spezielle Pflege, wohingegen das Vieh (engl. „cattle“) einfach in einer Herde gehalten werden kann. Dementsprechend stehen „pets“ für zustandsbehaftete Services und „cattle“ für zustandslose Service Container. Kubernetes arbeitet hier beispielsweise mit sogenannten „Pet Sets“ zur Organisation von zustandsbehafteten Containern; sie unterscheiden sich von regulären zustandslosen Containern in wesentlichen Eigenschaften:

- Sie verfügen über einen stabilen Hostnamen, der per DNS aufgelöst werden kann
- Ihnen wird eine ordinale Indexnummer zugeordnet, um die Rolle des Pet zu definieren
- Ihnen wird über Hostnamen und die ordinale Indexnummer ein stabiler (nicht-flüchtiger) Speicher zugeordnet

Features wie Zustandsüberwachung und erneutes Hochfahren des zugehörigen Containers nach einem Crash bleiben dabei erhalten, wobei der jeweilige betroffene Container über die zuvor genannten Eigenschaften seine Rolle und seine persistierte Identität im verteilten Cluster beibehält.

im Rahmen eines DNS-Servers zur Verfügung gestellt oder spezielle Load-Balancing-Container (wie HAProxy) nutzen die Konfiguration zum Verteilen der Anfragen. Hierbei können dedizierte Load-Balancer-Container auf separaten Hosts zum Einsatz kommen oder auch Load-Balancer-Instanzen (Kubernetes) auf jedem Host. Ein vorgeschalteter Load Balancer für Anfragen aus dem Internet rundet das ganze System ab *(siehe Abbildung 3)*.

Aktuelle Entwicklungen

Die momentanen Entwicklungen im Container-Stack liegen im Bereich „Persistierung und Datenhaltung.“ Sollen Daten in einer relationalen Datenbank, einer NoSQL-Datenbank oder in einem Queueing- oder Messaging-System gespeichert sein? Werden die Daten zentral für alle Applikationen vorgehalten oder gibt es getrennte Datenhaltung, so wie es die Microservice-Lehre vorsieht. Wie werden Daten zwischen den Services abgeglichen? Kann die Datenhaltung mit den Services skalieren? Wohin werden die Daten physikalisch abgelegt? Können die Container, unabhängig auf welchem Host sie laufen, immer auf die gleichen Daten zugreifen? Wie performant ist das? Wie entwickeln sich Techniken wie Flocker oder bieten Cloud-Dienstleister wie Amazon-S3-Storage die bessere Lösung für eine zentrale Datenhaltung? Die Auswahl des richtigen Persistenz-Layers inklusiv der richtigen Techniken trägt entscheidend zum Gelingen eines Container-Stack-Projekts bei.

Container Orchestration und Cloud

Die genannten Orchestrierungs-Frameworks sind bei vielen Cloud-Anbietern bereits fertig konfiguriert zu beziehen. Jeder Cloud-Provider bietet Docker an. Vorkonfigurierte Systeme für Kubernetes oder Mesos sind ebenso bei fast allen Anbietern zu bestellen. Interessant ist die Frage, ob die Container-Plattform, egal ob mit Docker Swarm, Kubernetes oder Mesos, auf virtuellen Servern oder auf dedizierter Hardware betrieben wird. Die meisten Angebote sind auf virtuellen Servern umgesetzt, auf Wunsch kann dedizierte Hardware verwendet werden. Hier geht Oracle mit dem Bare Metal Cloud Service einen großen Schritt voran.

Mario Herb
mario.herb@esentri.com

Matthias Fuchs
matthias.fuchs@esentri.com

Database as a Service – Fakten und erste Erfahrungen

Tobias Deml, Trivadis GmbH

In den letzten Monaten hat Oracle das Produkt „Database as a Service“ stark durch Veranstaltungen und Medienpräsenz beworben. Aufgrund der enormen Breite dieses Produkt-Spektrums ist es schwierig, den richtigen Service für seine Anforderungen zu finden. Der Artikel geht auf die Fakten und Variationen dieses Produkts genauer ein. Anschließend wird die Umgebung, die Oracle dem Kunden schlussendlich bereitstellt, aus diversen Blickrichtungen technisch betrachtet. Dabei kommt auch der Performance-Aspekt unter die Lupe

Wer in den letzten Jahren die Medien der IT-Welt verfolgt, bemerkt, dass Themen wie „DevOps“ häufig diskutiert und beleuchtet werden. Hinter diesem Begriff, der sich aus den Bereichen „Development“ und „IT Operations“ zusammensetzt, verbirgt sich ein Ansatz zur Verbesserung von Arbeitsprozessen. Dabei wird versucht, durch gemeinsame Anreize, Prozesse und Werkzeuge eine möglichst effektive und effiziente Zusammenarbeit der Abteilungen zu ermöglichen.

Der DevOps-Ansatz ist neben agilem Projektmanagement einer der Punkte, die unsere Infrastruktur und unser tägliches Arbeiten langfristig verändern werden. Dabei sind Aspekte wie Skalierbarkeit, Manageability und gleichzeitige Kosteneffizienz Anforderungen, die in Zukunft mehr und mehr an Bedeutung gewinnen werden.

Um all diese Anforderungen möglichst effizient erfüllen zu können, wäre ein monatliches oder gar nutzungsbasiertes Abrechnungsmodell von Vorteil. Eine potenzielle Lösung dieser Problematik ist die Verwendung von Cloud Services, um die geforderten Punkte flexibel lösen zu können.

Oracle Database Cloud Service

Oracle bietet im „Platform as a Service“-/Data-Management-Sektor eine Vielzahl verschiedener Lösungen an. Diese unterscheiden sich generell durch den Umfang der mitgelieferten Tools und beinhalteten vorgefertigte Lösungen (Backup/Patching). *Abbildung 1* zeigt eine schematische Darstellung der verschiedenen Produkte und einige zusätzliche Informationen.

Database as a Service – Virtual Image/Full Provisioning

Das Produkt „Database as a Service – Virtual Image“ ist die am wenigsten von Oracle verwaltete Lösung, aber gleichzeitig die günstigste. Hierbei handelt es sich um ein OS-Image, im aktuellen Falle Oracle Enterprise Linux 6.6, das ebenfalls die Installations-Ressourcen für eine Oracle-Datenbank beinhaltet.

Der Aufwand für die Installation und Konfiguration der Oracle-Datenbank muss selbst geleistet werden. Dies beinhaltet natürlich auch jegliche Flexibilität bei der Anpassung der Umgebung nach seinen eigenen Wünschen und Standards. Außerdem müssen Themen wie Backup und Monitoring ebenfalls selbst eingerichtet und verwaltet werden.

Diese Lösung ist hinsichtlich der Kosten am attraktivsten, man muss jedoch das Know-how für die Installation und Verwaltung der darauf befindlichen Datenbanken selbst bereitstellen.

Mit der Lösung „Database as a Service – Full Provisioning“ möchte Oracle eine andere Zielgruppe als im Vergleich zum „Virtual Image“-Produkt ansprechen. Bei der Bereitstellung dieser Services werden Tasks wie die Erstellung der Datenbank und die Einrichtung einer Backup-Lösung voll automatisch von Oracle erledigt. Außerdem sind hier sehr hilfreiche Tools, wie beispielsweise der DBaaS-Monitor zur Verwaltung und Analyse der Datenbank, mitgeliefert. Dies zielt auf Kunden ab, die zwar Datenbank-Know-how besitzen, einige administrative Tasks aber von Oracle erledigt haben möchten.

Oracle Database Schema Service

Mit dem „Oracle Database Schema Service“ möchte Oracle vorzugsweise Entwickler ansprechen. Bei dieser Lösung übernimmt Oracle jegliche Verwaltung von OS- und Datenbank-seitigen Themen. Als Kunde bekommt man Zugangsdaten zugesendet, die für die Entwicklung eigener Lösungen mit Application Express, Java-Services oder RESTful-Web-Services verwendet werden können.

Eine Sache gibt es zu beachten: Bei der Bereitstellung dieses Produkts sind keine Zugangsdaten für den Aufbau von SQL*Net enthalten; man kann somit keine Verbindung via SQL*Plus oder Ähnliches aufbauen.

Auf diese Lösung wird aufgrund des gesteckten thematischen Rahmens in diesem Artikel nicht weiter eingegangen.

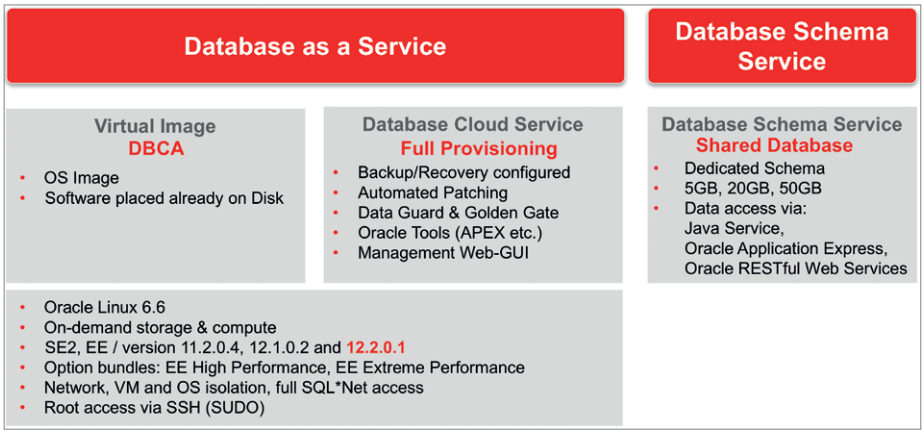


Abbildung 1: Oracle Database Cloud Service

Shapes/Versionen/Editionen

Um in der Cloud-Umgebung eine Vielzahl an Ausprägungen liefern zu können, sind die Services in diversen Leistungsklassen (Shapes), Versionen und Editionen verfügbar. Bei den Shapes wird generell folgende Unterteilung getroffen *(siehe Abbildung 2)*.

Falls aus diversen Gründen mehr Arbeitsspeicher benötigt wird, können sogenannte „High-Memory-Shapes“ gebucht werden, die die doppelte Größe an RAM je Stufe besitzen. Ein vorstellbarer Usecase wäre die Verwendung von Oracle-Database-In-Memory, bei dem generell einen größere Memory-Struktur erforderlich ist *(siehe Abbildung 3)*.

Die Verrechnung dieser Shapes geschieht über die Anzahl verwendeter OCPUs. Ein OCPU repräsentiert einen physikalischen CPU-Core in einer Compute-Cloud-Umgebung. Viele andere Cloud-Anbieter verwenden die Einheit „vCPU“, was einen CPU-Thread repräsentiert. Die Verwendung der vCPUs hat einen Nachteil: Bei Nutzung einer ungeraden Anzahl muss man davon ausgehen, dass man einen CPU-Core mit einer anderen Partei teilen muss. Außerdem ist ebenfalls nicht sichergestellt, dass bei der Nutzung von zwei vCPUs derselbe Kern verwendet wird, und somit kann sich die Beeinflussung durch Dritte noch ausweiten.

Hinsichtlich der verfügbaren Versionen gibt es Unterschiede bei den verschiedenen Services. Bei „Database as a Service – Full Provisioning“ sind die Datenbank-Versionen 11.2.0.4, 12.1.0.2 und 12.2.0.1 auswählbar. Hingegen ist beim Virtual Image die Version 12.2.0.1 leider noch nicht verfügbar. Dieser Fakt kann sich allerdings mit dem On-Premise-Release 12.2.0.1 der Oracle-Datenbank ändern.

Um die breite Variation an Editionen und Optionen einer On-Premise-Oracle-Datenbank abzubilden, wurden sogenannte „Packages“ eingeführt. Hierbei gibt es folgende Auswahlmöglichkeiten:

- *Standard Package*
  Oracle Database Standard Edition einschließlich Transparent Data Encryp-

Shapes – High Memory

Shape	Cores	Threads	Memory(GB)	Local disk
OC1M	1	2	**15**	60 GB
OC2M	2	4	**30**	60 GB
OC3M	4	8	**60**	60 GB
OC4M	8	16	**120**	60 GB
OC5M	16	32	**240**	60 GB

Abbildung 2: Leistungsklassen „General Purpose“

Shapes – General Purpose

Shape	Cores	Threads	Memory(GB)	Local disk
OC3	1	2	7.5	60 GB
OC4	2	4	15	60 GB
OC5	4	8	30	60 GB
OC6	8	16	60	60 GB
OC7	16	32	120	60 GB

Abbildung 3: Leistungsklassen „High Memory“

tion (TDE) und Hybrid Columnar Compression (HCC)
- *Enterprise PackageOracle Database* Enterprise Edition einschließlich TDE und HCC
- *High Performance Package* Oracle Database Enterprise Edition einschließlich aller Optionen außer RAC, In-Memory-Database und Active Data Guard
- *Extreme Performance Package* Oracle Database Enterprise Edition einschließlich aller Optionen

All diese Packages sind mit zwei unterschiedlichen Abrechnungsmodellen erhältlich. Einerseits die monatliche und andererseits die nutzungsbasierte Abrechnung. Der Takt für das nutzungsbasierte Modell beträgt eine Stunde.

Hinsichtlich der Auswahl der Abrechnungsart ist entscheidend, wie lange die Verwendungsdauer der Umgebung pro Monat beträgt. Als nicht genutzt gilt die Ressource lediglich, wenn die Umgebung heruntergefahren ist. Somit gibt es bei einer hohen Verwendung einen Break-Even-Point, ab dem die monatliche Abrechnung kostengünstiger ist.

Maintenance

Beim „Oracle Database Cloud Service – Full Provisioning" ist neben der vorkonfigurierten Backup-Lösung eine Vielzahl von Tools und Werkzeugen mitgeliefert. Dabei können einige wiederkehrende Tasks durch bereitgestellte Utilities als API-Aufruf oder per Knopfdruck erledigt werden. Unter diese Tasks fallen unter anderem das Bereitstellen neuer Umgebungen und das Patchen beziehungsweise Upgraden bestehender Systeme.

Um die vollen Vorteile einer Datenbank-Cloud-Lösung nutzen zu können, gibt es ebenfalls die Möglichkeit, die Leistungsklasse (Shape) eines Service zu erhöhen oder zu verringern. Dies kann via Kommandozeile oder über die Web-GUI erfolgen. Bei diesem Task lässt sich außerdem die Größe der Diskgruppen anpassen. *Abbildung 4* zeigt den entsprechenden „Scale Up/Down Service"-Dialog, in dem man die nötigen Informationen für die Skalierung der Cloud-Umgebungen eintragen kann. Nach der Bestätigung dieses Dialogs werden der Service heruntergefahren, die geforderten Anpassungen vorgenommen und die Umgebung wieder gestartet.

Um administrative Tätigkeiten wie beispielsweise Parameter-Änderungen oder Performance-Analysen möglichst einfach erledigen zu können, liefert Oracle beim „Full Provisioning"-Service eine Apex-basierte Anwendung namens „DbaaS Monitor" mit. Damit ist es möglich, diverse Tasks in der Datenbank-Umgebung über eine entsprechende Benutzeroberfläche zu erledigen. *Abbildung 5* zeigt den Home-Screen des DBaaS Monitor mit generellen Informationen über die betrachtete Datenbank.

Ähnlich wie im Oracle Enterprise Manager Database Express lassen sich hier durch die Auswahl verschiedener Reiter genauere Informationen wie beispielsweise die Auslastung des Servers oder die Belegung der Tablespaces erheben. Wenn in der Infrastruktur keine Cloud-Control-Umgebung existiert, ist der DbaaS Monitor eine gute Alternative, um kleine administrative Aufgaben zu erledigen.

Performance

Das Misstrauen hinsichtlich der Cloud-Produkte ist oftmals auch in deren Performance begründet. Ein typischer Grund dafür ist die Annahme, dass man sehr wenig Kontrolle darüber besitzt, ob eine andere Partei um dieselben Ressourcen konkurriert oder ob im schlimmsten Falle eine Überbelegung gewisser Ressourcen vorherrscht.

Bei den aktuellen Oracle-Database-Cloud-Service-Produkten kommen Intel-Prozessoren vom Typ Xeon E5-2699 v3 @ 2.30GHz zum Einsatz. Diese haben in den getätigten CPU-Tests eine respektable Leistung abgeliefert. Sie liegen im Preis-Leistungs-Verhältnis im Vergleich zu anderen Cloud-Anbietern im oberen Bereich. Die Verwendung dieser Chips ist aktuell so dokumentiert, aber falls Oracle künftig einen Wechsel auf eine neuere CPU-Generation plant, wird dies vermutlich implizit geschehen. Somit kann man davon ausgehen, dass in beispielsweise einem Jahr andere CPUs verwendet werden. Aus diesem Grund ist man bei der Leistungsfähigkeit der verwendeten OCPUs immer abhängig von der von Oracle verwendeten Hardware.

Die Leistungsfähigkeit des IO-Subsys-

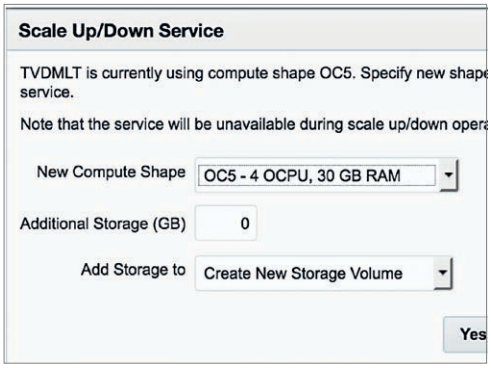


Abbildung 4: Dialog „Scale Up/Down Service"

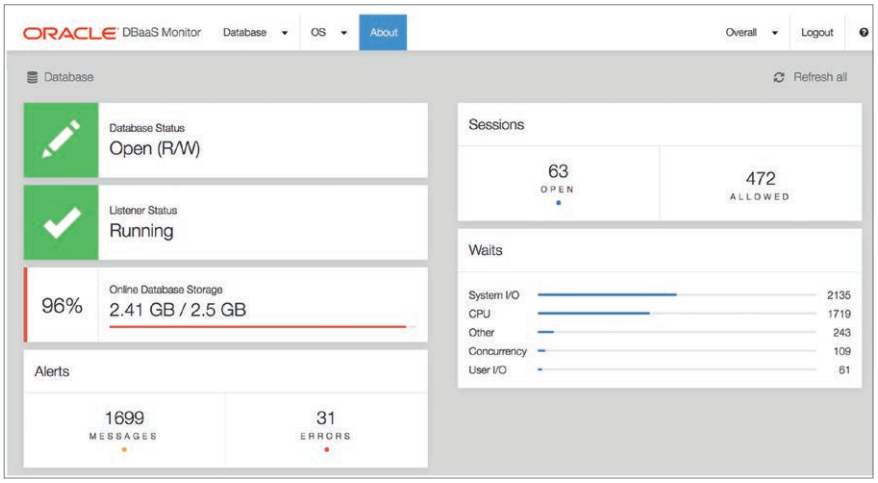


Abbildung 5: DbaaS Monitor

```
SQL*Plus: Release 12.2.0.1.0 Production on Wed Nov 9 15:00:35 2016

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Connected to:
Oracle Database 12c EE Extreme Perf Release 12.2.0.1.0 - 64bit Production

SQL>
SET SERVEROUTPUT ON
     DECLARE
      lat  INTEGER;
      iops INTEGER;
      mbps INTEGER;
     BEGIN
      DBMS_RESOURCE_MANAGER.CALIBRATE_IO (1, 10, iops, mbps, lat);
      DBMS_OUTPUT.PUT_LINE ('max_iops = ' || iops);
      DBMS_OUTPUT.PUT_LINE ('latency  = ' || lat);
      DBMS_OUTPUT.PUT_LINE ('max_mbps = ' || mbps);
      END;
      /

SQL> SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12

max_iops = 108948
latency  = 0
max_mbps = 1103
```

Abbildung 6: Ausführung Calibrate_IO

tems war sehr überraschend. Eine vorherrschende Problematik bei Cloud-Produkten ist die Latenz beim Single-Block-Read-IO. Hier erwartet man, wie bei den bestehenden On-Premise-Umgebungen, Verzögerungen von weniger als drei beziehungsweise höchstens fünf Millisekunden. Eine aktuelle Problematik bei verschiedenen Cloud-Anbietern ist, dass diese Latenz eher im Bereich von zehn Millisekunden oder mehr liegt, was bei einer transaktionalen oder gar OLTP-ähnlichen Applikationen nicht für akzeptable Antwortzeiten sorgt.

Die Messung der Leistungsfähigkeit ist mit verschiedenen Tools und Utilities durchgeführt worden. Die Latenz für einen Single-Block-Read-IO beträgt bei den Oracle Database Cloud Services (ODCS) Umgebungen weniger als eine Millisekunde. Dies wird gar bei einem mehrfach parallelen Workload wie beispielsweise mit SLOB unter 0,6 Millisekunden gehalten. Selbstverständlich ist die Storage-Infrastruktur in der Oracle-PaaS-Cloud aktuell nicht unter Voll-Last, nichtsdestotrotz sind das sehr beeindruckende Werte, die auf ein sehr gutes Bonding der Storage-Infrastruktur zu den Compute-Instanzen hinweisen. *Abbildung 6* zeigt einen Storage-Test mit Calibrate_IO, dessen Ergebnisse beachtenswert sind. In dem Output ist ablesbar, dass bei dem Testlauf folgende KPIs erreicht wurden:

- Maximale IOPS-Operationen: 108.948
- Latenz: 0 ms
- Maximaler Durchsatz: 1,1 GB/s

Dies sind wirklich sehr gute Werte für ein IO-Subsystem. Die Überlegung ist, welche Umgebung man On-Premise bereitstellen müsste und welche Kosten damit verbunden wären, um diese Leistungswerte zu bewerkstelligen.

Fazit

Die Produktpalette des Oracle Database Cloud Service bietet ein weites Spektrum an Lösungen für verschiedenste Zielgruppen. Die Herausforderung ist, die eigenen Anforderungen genau zu kennen und diese auf die verschiedenen angebotenen Lösungen adaptieren zu können.

Nach den Performance-Tests zu urteilen liefern die Cloud-Umgebungen hinsichtlich des technischen Setups ein solides Bild ab. Besonders hervorheben ist die IO-Performance, deren Entwicklung man allerdings noch beobachten muss, wenn die Cloud-Infrastruktur einer höheren Last unterliegt.

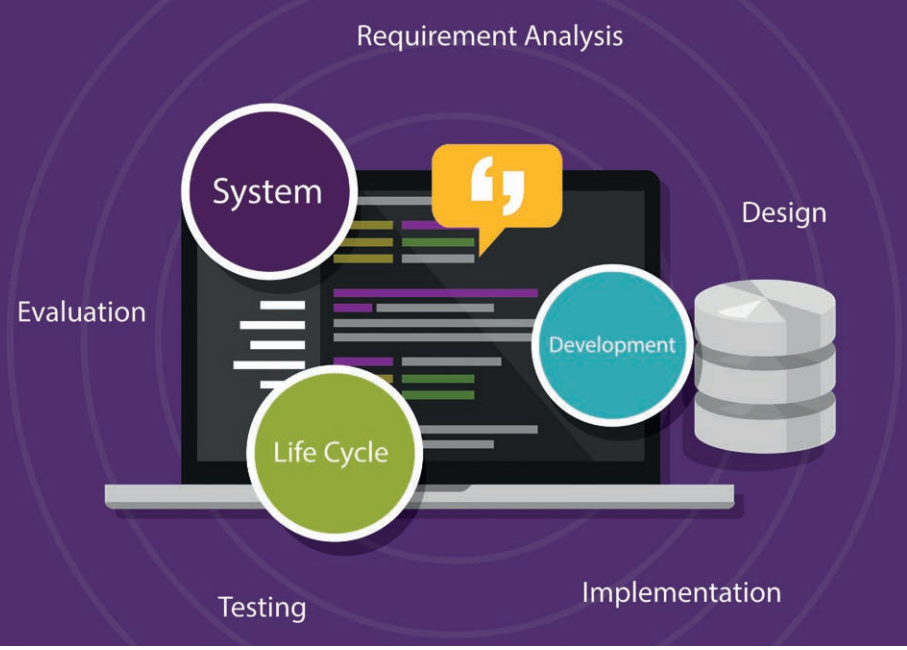
Bei den von Oracle verwalteten Lösungen wird einen breite Palette an Tools und Utilities mitgeliefert. Die Konzepte dieser Werkzeuge sind durchdacht, jedoch ist nach dem Geschmack des Autors die Variation etwas zu hoch. Er würde sich wünschen, mehrere Funktionen in einem Tool zu vereinen, um die Verwendung etwas einfacher zu gestalten.

Während der Autor sich mit dieser Thematik beschäftigt hat, ist ihm immer wieder aufgefallen, dass bei vielen Tasks immer noch tiefes Datenbank- und IT-Know-how erforderlich ist. Spätestens bei gegebenenfalls auftretenden Performance-Problemen braucht man die Kapazitäten und Möglichkeiten, damit umgehen zu können.

Bei der Etablierung von Cloud-Lösungen sind Aspekte zu betrachten, die man vielleicht aus der Vergangenheit noch nicht kennt, wie beispielsweise die Latenz der eigenen Internet-Verbindung oder das SLA des Internet-Providers. Dies sind Faktoren, die man zumindest geprüft und evaluiert haben muss, um ein Cloud-Adaptionsprojekt erfolgreich abschließen zu können.

Bei der detaillierten Betrachtung der verschiedenen Cloud-Lösungen fällt auf, wie viele Infrastruktur-Optimierungsprojekte vielleicht kostengünstiger und gar attraktiver abgeschlossen hätten werden können. Dies ist der Grund dafür, dass man bei solchen Projekten ebenfalls die Cloud-Lösungen nüchtern betrachten und evaluieren sollte. Die Oracle Database Cloud Services sind technisch sehr attraktiv und der Autor kann sich vorstellen, dass diese Lösungen mittelfristig guten Anklang finden werden.

Tobias Deml
tobias.deml@trivadis.com



Application Lifecycle Management mit dem Oracle Developer Cloud Service

Stefan Kühnlein, OPITZ CONSULTING Deutschland GmbH

In den letzten Jahren hat sich die Art und Weise, wie Anwendungen entwickelt werden, sehr stark verändert. Viele Projektteams nutzen inzwischen agile Entwicklungsmethoden, Continuous Integration und Continuous Delivery. Unterstützt wird diese Veränderung der Anwendungsentwicklung durch Software, die zum Teil auch als Open Source erhältlich ist. Damit das Team diese Software nutzen kann, muss ein Entwickler oder ein Administrator die Installation und die Wartung übernehmen. In vielen Fällen braucht die Bereitstellung der Entwicklungsumgebung deshalb einen gewissen Vorlauf. Aber auch während der Projekt-Laufzeit werden für die Administration und Wartung weitere Ressourcen gebunden. Der Oracle Developer Cloud Service (DevCS) verspricht hier Abhilfe.

Durch die Nutzung von Cloud-Technologien können Unternehmen ihre Entwicklungsproduktivität steigern und Software flexibler weiterentwickeln. Mit der schnellen Bereitstellung einer vollständigen Entwicklungs- oder Testumgebung sparen sie Zeit und Kosten – unabhängig von der Anzahl der Entwicklungsteams und der benötigten Umgebung. Hierzu stellt Oracle mit dem DevCS eine integrierte Umgebung mit allen Tools zur Verfügung, die den vollständigen Lifecycle einer Anwendung vereinfachen und unterstützen. Durch deren Verwendung können Entwicklerteams sofort mit der Entwicklung beziehungsweise der Wartung von Anwendungen starten: Die manchmal endlos erscheinenden Installationen und Konfigurationen von Tools wie Git, Jira, Hudson und zusätzlichen Tools entfallen.

Im Gegensatz zu vielen PaaS-Services von Oracle kann der DevCS nicht separat bezogen werden. Der DevCS ist ein fester Bestandteil von Java Cloud Service, Java Cloud Service SaaS Extension, Oracle Messaging Cloud Service, Oracle Mobile Cloud Service, Oracle SOA Cloud Service oder Oracle Application Container Cloud Service *(siehe Abbildung 1)*. In naher Zukunft wird er auch als Teil des Container Cloud Service verfügbar sein. Mit dem DevCS stellt Oracle eine Plattform sowohl für Collaboration als auch für Continuous Integration und Continuous Delivery bereit, die von den genannten Services genutzt werden kann.

Unabhängig davon, ob ein Team erste Schritte mit einer Trail-Version *[1]* oder mit einer Subscription eines der genannten Services beginnt, wird der DevCS mit provisioniert, sodass die Initialisierung des ersten Projekts sofort beginnen kann.

Initialisierung eines neuen Projekts

Die Initialisierung eines neuen Projekts ist im DevCS sehr einfach und erfolgt über einen entsprechenden Wizard. In drei Schritten wählt der Anwender die wichtigsten Informationen aus. Zuerst wird er nach dem Projekt, der Sichtbarkeit des Projekts sowie der Projekt-Sprache gefragt. Im zweiten Schritt wählt er für die Erstellung des Projekts ein Template aus. So kann der Anwender an dieser Stelle

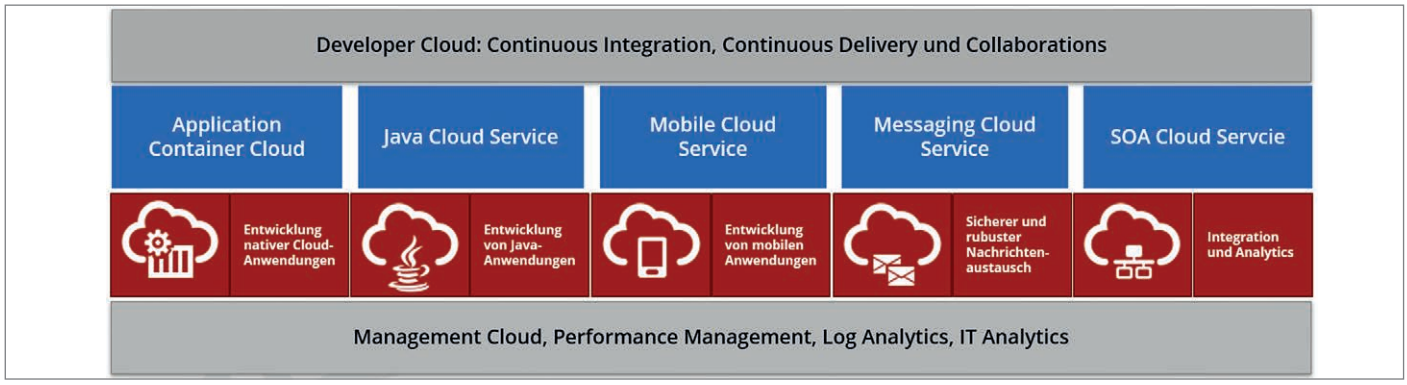


Abbildung 1: Überblick über den Developer Cloud Service

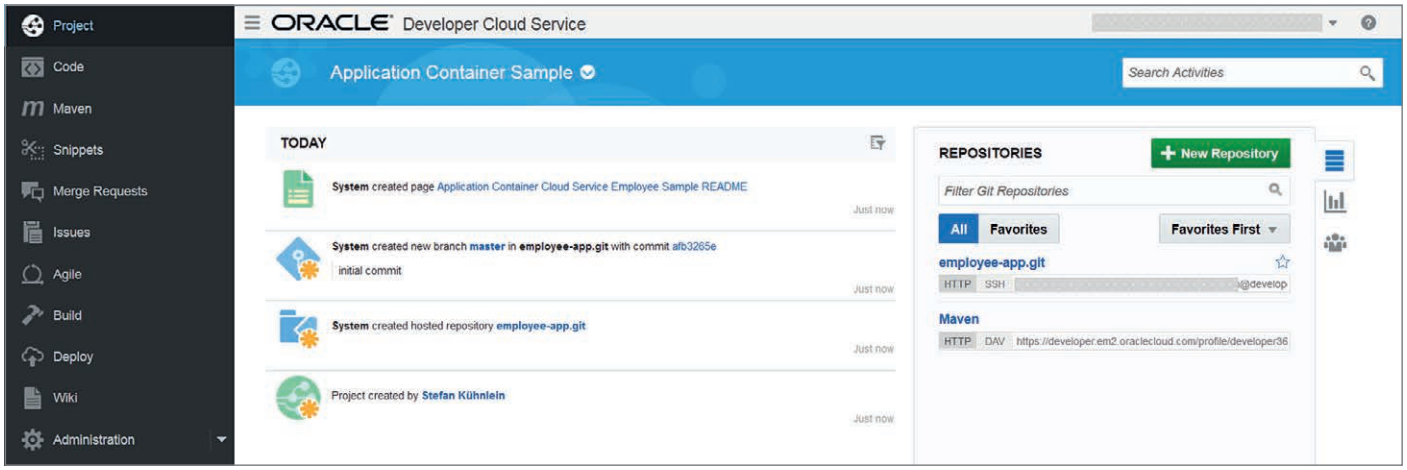


Abbildung 2: Projekt-Übersicht im DevCS

zwischen einem leeren Projekt, einer initialen Projekt-Erstellung oder einem Beispielprojekt auswählen. Zuletzt muss er noch einige Projekt-Eigenschaften für das Repository des Software Configuration Management (SCM) sowie das Wiki-Markup einstellen. Mit Abschluss des Wizard werden im DevCS das neue Projekt und entsprechende Services initialisiert. Bei der Erstellung eines neuen Projekts erhält der Anwender automatisch die Rolle des Projekt-Administrators. *Abbildung 2* zeigt den DevCS nach der Initialisierung des mitgelieferten Beispielprojekts.

Für alle genannten Services stellt der DevCS entsprechende webbasierte Benutzeroberflächen bereit. Alternativ unterstützt Oracle die Integration des DevCS für die gängigsten Entwicklungsumgebungen wie JDeveloper, NetBeans und Oracle Enterprise Pack for Eclipse (OEPE). Für die Integration der Oracle Cloud in die Entwicklungsumgebung IntelliJ stellt Viatra *[2]* ein entsprechendes Plug-in bereit. So können Entwickler zum Beispiel von der Entwicklungsumgebung direkt auf das Git-Repository oder auf ihre Aufgaben zugreifen. Ein wesentlicher Vorteil der Integration des DevCS in die jeweilige Entwicklungsumgebung ist, dass bei einem Commit von geänderten Ressourcen diese einer Aufgabe zugewiesen werden können. *Abbildung 3* zeigt die Kategorien,

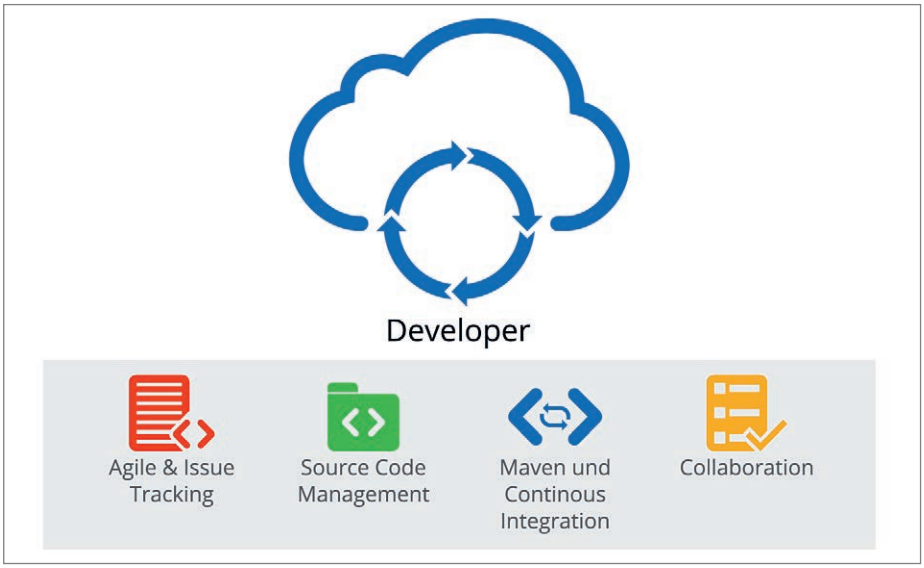


Abbildung 3: Kategorien des DevCS

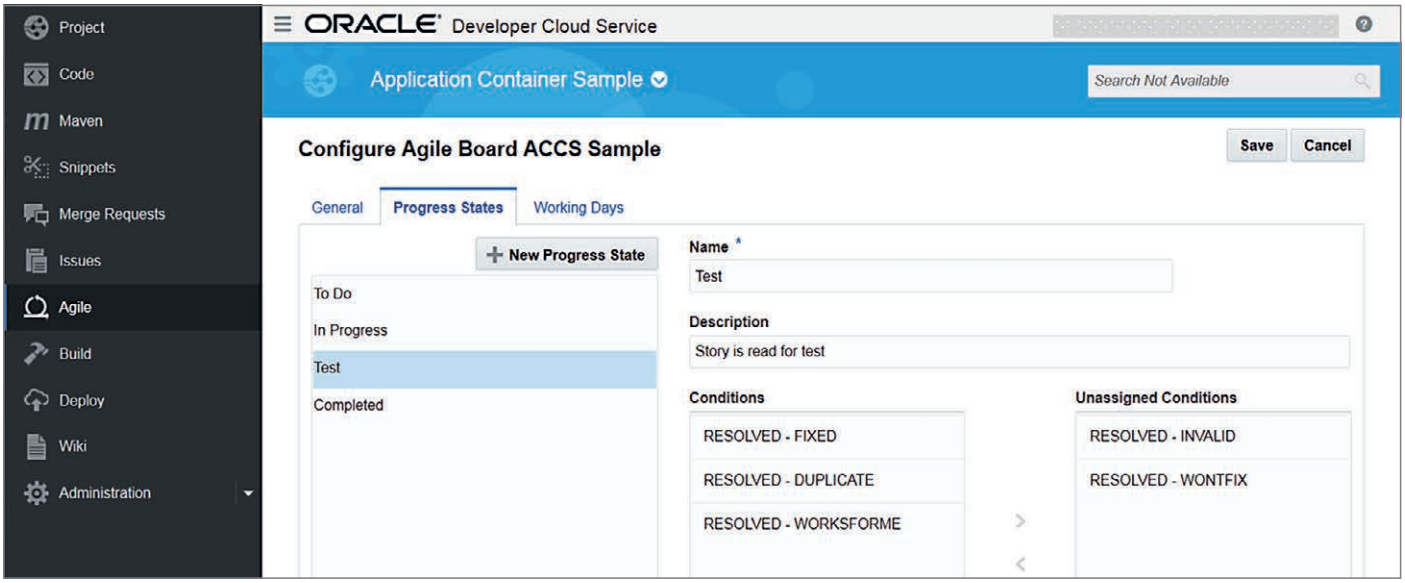


Abbildung 4 : Konfiguration des Boards

in die sich die Services des DevCS innerhalb eines Projekts einteilen lassen.

Agile und Issue Tracking

Das agile Manifest hat sich inzwischen in den meisten Unternehmen etabliert, sodass diese Vorgehensmethodik aus dem Projektalltag nicht mehr wegzudenken ist. Die zentralen Ziele und Vorteile der agilen Software-Entwicklung bestehen darin, die zu entwickelnde Software schneller und zugleich mit einer höheren Qualität auszuliefern und somit die Akzeptanz der Endanwender zu steigern. Damit dies gelingen kann, sind jedoch eine exakte Planung der Sprints sowie eine enge Verfolgung der Issues notwendig. Dies kann effektiv nur mit einer entsprechenden Tool-Unterstützung erfolgen. Zu diesem Zweck ist im DevCS sowohl ein agiles Board als auch die Verfolgung von User Stories und Issues integriert. Beide, das agile Board ebenso wie das Issue Tracking, basieren auf Jira. Wie bereits im vorherigen Abschnitt erläutert, nimmt der Anwender beim Anlegen des Projekts die notwendigen Konfigurationen für das Board und das Issue Tracking vor, sodass er die Planung und den Aufbau eines neuen Boards ohne weitere Konfiguration beginnen kann.

Um ein neues Board zu erstellen, muss der Entwickler im Dashboard lediglich den Eintrag „Agile“ und den entsprechenden Wizard mithilfe des Buttons „New Board“ auswählen. Für die Erstellung eines neuen Boards erfasst er nur den Namen und stellt die Suche sowie die Auswahl der Schätzgröße (Story Points oder Aufwand) im Vorfeld ein. In diesem neu erstellten Board kann der Entwickler nun die jeweiligen Sprints sowie deren User Stories erfassen. Nachdem alle User Stories erfasst sind, kann der entsprechend Sprint im Backlog aktiviert werden. Initial besteht der aktive Sprint aus den drei Phasen „To Do“, „In Progress“ und „Completed“. Reichen diese Phasen für die Projekt-Organisation nicht aus, kann der Anwender über die Konfiguration des Boards noch weitere Phasen hinzufügen. *Abbildung 4* zeigt die Konfiguration des Boards um die Erweiterung der Phase „Test“. Bei der Konfiguration des Boards können aktuell keine eigenen Bedingungen definiert werden.

Der Administrator kann das Issue Tracking im Bereich „Administration“ anpassen, um neue Produkte, Releases und Komponenten anzulegen. Zur Erfassung projektspezifischer Informationen in den Issues kann er zusätzliche benutzerdefinierte Felder anlegen, die in den Issues angezeigt werden.

Source Control Management

Eine weitere wichtige Säule des DevCS stellt das Source Control Management dar. Dieses basiert auf der Open-Source-Software Git. Deren Stärken liegen unter anderem in der parallelen und unabhängigen Entwicklung verschiedener Branches sowie der sogenannten „Stages Area“.

Im Rahmen der Erstellung eines neuen Projekts erstellt das Entwicklerteam ein neues Git-Repository, das für die Versionsverwaltung der Sources verwendet werden kann. Initial werden auf der Seite „Code“ Informationen zum Setup eines lokalen Repository angezeigt. Existiert bereits ein Repository, das unter die Versionsverwaltung des DevCS gestellt werden soll, kann das Team auch ein bestehendes externes Repository importieren. Sollte das bestehende Repository durch ein Passwort geschützt sein, legt man für den Import einen Benutzer mit Rechten zum Lesen des Repository an. Nach dem ersten Commit kann der Entwickler auf der Seite „Code“ die Informationen zum Erstellen eines lokalen Repository ausblenden und das Filesystem des Repository darstellen, um damit zu arbeiten. In diesem Bereich lassen sich nach einem bestimmten Source-Code suchen, die einzelnen Commits auswerten sowie Branches und Tags erstellen beziehungsweise vergleichen.

Merge Requests

Auf Basis der Branches lassen sich im DevCS Code-Reviews initialisieren und

durchführen. Diese sind ein beliebtes Hilfsmittel, um Fehler zu vermeiden und Design- oder Implementierungs-Probleme zu identifizieren, die sich zum Beispiel auf die Performance der Anwendung auswirken. Für die Durchführung von Code-Reviews muss der Entwickler im DevCS einen sogenannten „Merge Request" einrichten. Dieser kann im DevCS nur angelegt werden, wenn man für die Entwicklung verschiedene Branches verwendet. Nach der Erstellung kann er durch den Reviewer bearbeitet werden. Als Review stehen folgende Aktionen zur Verfügung *(siehe Abbildung 5)*:

- Hinzufügen von Kommentaren
- Akzeptieren oder Zurückweisen des Merge Request
- Übernehmen der Änderungen in den Ziel-Branch
- Schließen des Merge Request

Maven

Mit Maven wurde ein weiteres Open-Source-Werkzeug der Apache Software Foundation in den DevCS aufgenommen. Das Werkzeug hat in den letzten Jahren immer mehr an Beliebtheit gewonnen, da es im Vergleich zu Ant den Grundgedanken „Konvention vor Konfiguration" für den gesamten Zyklus der Software-Erstellung umsetzt. Mithilfe von Maven wird der Entwickler rundum unterstützt – von der Anlage eines Software-Projekts über das Kompilieren, Testen und Paketieren bis zur Verteilung der Software.

Alternativ zu Maven besteht noch die Möglichkeit, im DevCS mit Gradle zu arbeiten. Gradle setzt ebenfalls wie Maven auf das Programmierparadigma „Convention over Configuration" und verwendet die gleiche Verzeichnisstruktur. Dank dieser Standards müssen für die Aufgaben des Build-Managements nur sehr wenige Einstellungen vorgenommen werden.

In vielen Fällen dürfen für die Entwicklung nur speziell freigegebene oder käuflich erworbene Bibliotheken verwendet werden. Um diese Bibliotheken bereitzustellen, ist ein Repository notwendig. Der DevCS stellt für jedes Projekt ein eigenes Maven-Repository bereit. Dieses wird mit der Initialisierung des Projekts automatisch erstellt. Die Informationen für den Zugriff auf das Repository werden unter dem Menüpunkt „Maven" angezeigt *(siehe Abbildung 6)*.

Über diese Seite können Entwickler die Bibliotheken im Repository verwalten. So lassen sich neue Bibliotheken dem Repository hinzufügen oder löschen oder man kann nach bereits importierten Bibliotheken suchen. Neue Artefakte können dem Repository entweder direkt im DevCS oder über das Maven Command Line Interface hinzugefügt werden.

Continuous Integration

Für die Automatisierung des Builds ist im DevCS eine Cloud-optimierte Version von Hudson integriert. Je nachdem, welche Programmiersprache für ein Projekt gewählt wurde, stehen entsprechende Build-Tools zur Verfügung. So können für den automatischen Build der Java-basierten Anwendungen Gradle, Maven und Ant verwendet werden. Für die Entwicklung von Anwendungen mit JavaScript oder Node.JS erfolgt die Automatisierung des Builds entweder mit npm, Gulp, Grunt oder Bower. Analog zu einer On-Premise-Installation von Hudson lässt sich auch bei der Cloud-basierten Lösung das Ereignis für den Start eines Builds definieren. Der DevCS bietet die folgenden Varianten für den Start des Build-Prozesses:

- Zu einem fest definierten Zeitpunkt
- Als Ergebnis eines vorgelagerten Build-Prozesses
- Mit dem Committen von Änderungen im Git-Repository

Für die Durchführung des Builds lassen sich im DevCS mehrere Build-Schritte definieren. Hierzu können über den Button „Add Build Step" beliebig viele Schritte hinzugefügt werden *(siehe Abbildung 7)*.

Ein wesentlicher Bestandteil eines automatisierten Build-Prozesses ist die Qualitätssicherung der bereitzustellenden An-

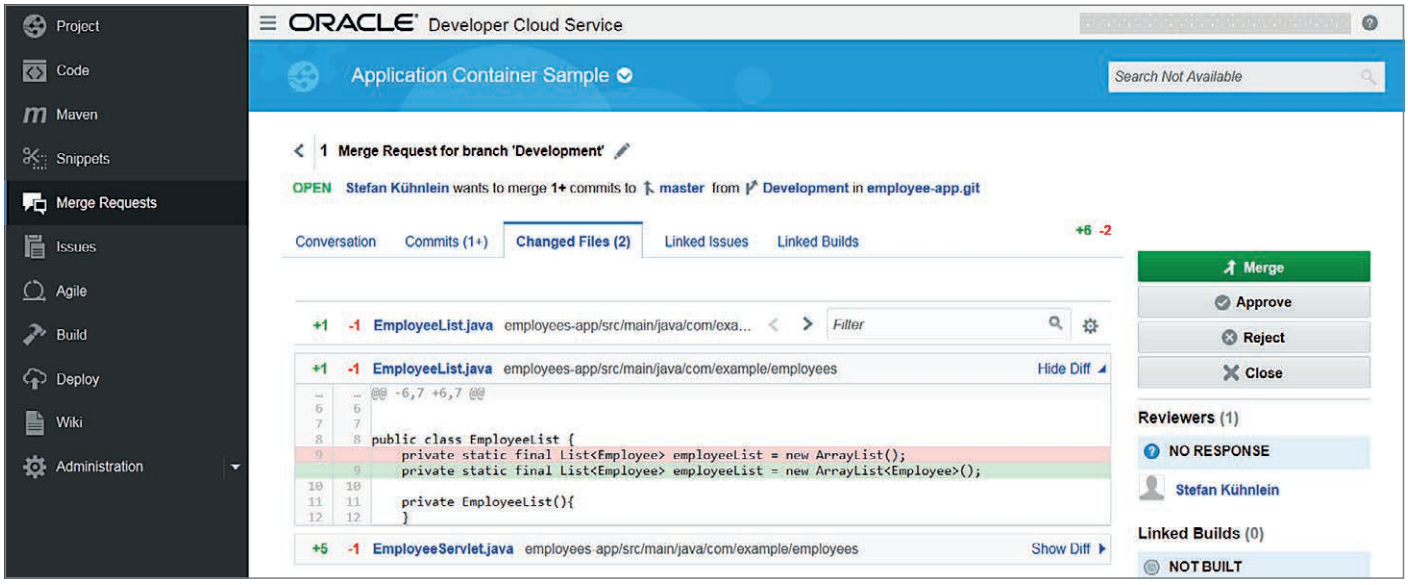


Abbildung 5: Merge Request im DevCS

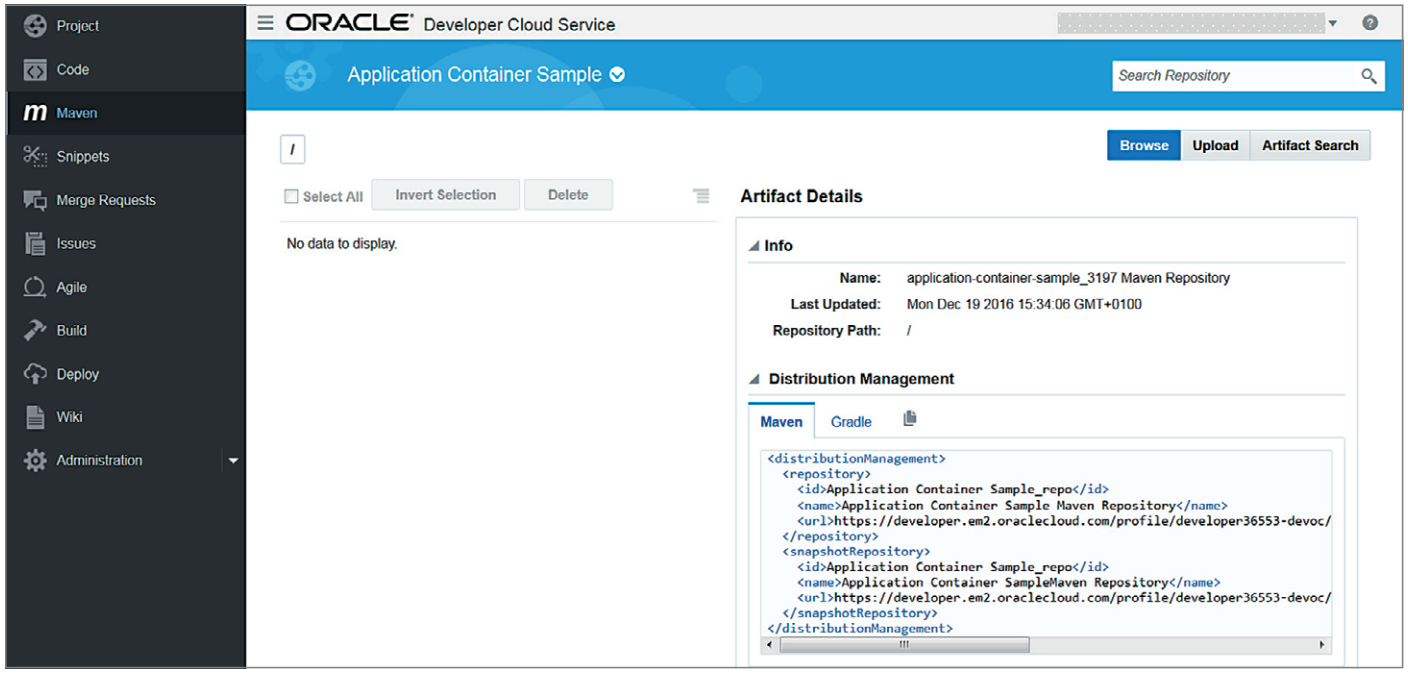


Abbildung 6: Verwaltung der Artefakte im Maven-Repository

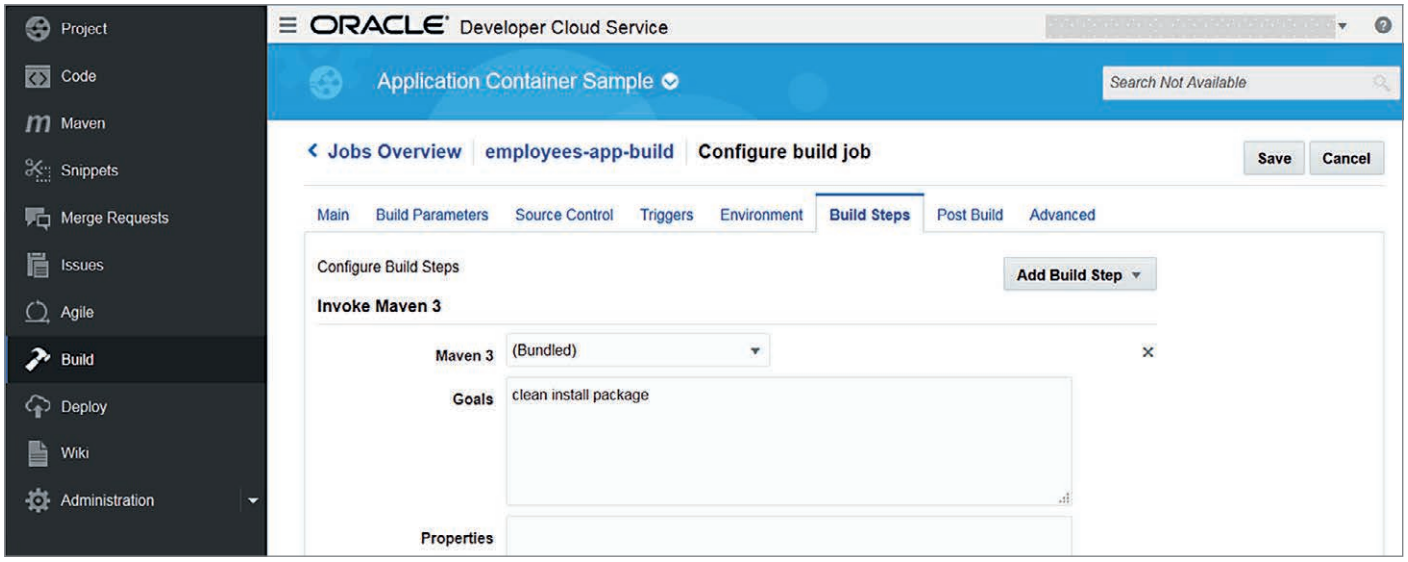


Abbildung 7: Konfiguration eines Build-Schritts

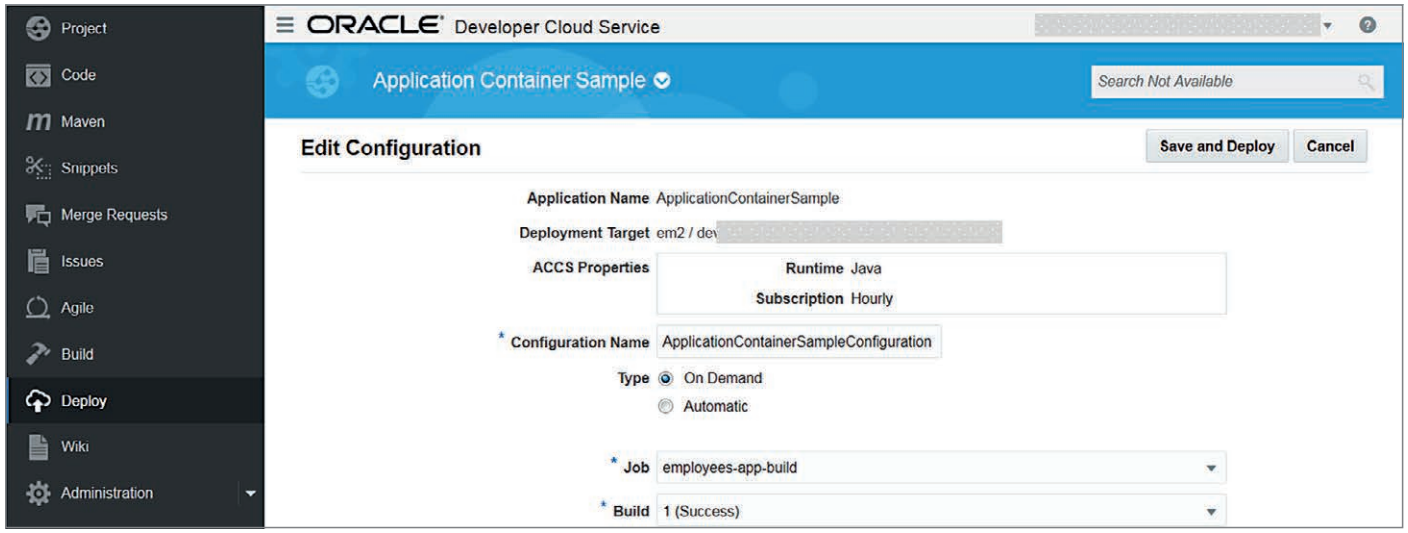


Abbildung 8: Konfiguration des Deployments im ACCS

wendung. Zu diesem Zweck wurden in den DevCS einige beliebte Frameworks integriert. Hierzu zählen für die Ausführung von automatischen Tests im Backend JUnit und im Frontend Selenium. Für die statische Überprüfung des Quellcodes ist FindBugs integriert. Aktuell verfügt der DevCS noch über keine Docker Build Tools. Jedoch arbeitet Oracle aktuell schon daran, eine Integration zwischen DevCS und dem Container Cloud Service zu ermöglichen.

Deployment

Der letzte Schritt in einer Continuous Delivery Pipeline stellt das Deployment der gebauten Anwendung dar. Auch dieser Schritt unterstützt der DevCS, und zwar auf den folgenden Ziel-Plattformen:

- Oracle Java Cloud Service – SaaS Extension in einer eigenen Identity Domain
- Oracle Java Cloud Service – SaaS Extension in ein anderes Data Center oder eine andere Identity Domain
- Public Oracle Java Cloud Service Instance
- Oracle Application Container Cloud Service Instance

Je nachdem, auf welcher Ziel-Plattform die Anwendung bereitgestellt werden soll, unterscheiden sich die Wizards zum Anlegen einer Deployment-Konfiguration. Die Deployment-Konfigurationen verwaltet der Anwender auf der Seite „Deploy". Für das Deployment im Oracle Java Cloud Service – SaaS Extension beziehungsweise im Oracle Application Container Cloud Service gibt er lediglich das Data-Center, die Identity Domain sowie User und Passwort der Zielumgebung an. *Abbildung 8* zeigt die vollständige Konfiguration für das Deployment der Beispiel-Anwendung im Application Container Cloud Service.

Das Deployment in einen Oracle Java Cloud Service erfolgt entweder über einen SSH-Tunnel oder über das RESTful-Management-Interface. Für alle Deployment-Konfigurationen müssen weitere Informationen wie Job, Build und das zu bereitstellende Artefakt angegeben werden. Das Deployment einer Anwendung kann automatisch nach jedem Build erfolgen, über die Auswahl des Typs „Automatic". Bei der Auswahl von „On Demand" wird das Deployment nur beim Speichern beziehungsweise über das Menü „Redeploy" ausgeführt.

Collaboration

Alle restlichen Services (Wiki und Snippets) des DevCS lassen sich der Kategorie „Collaboration" zuordnen. Zum einen bietet der DevCS ein Wiki für die Zusammenarbeit im Team und für die Erstellung von Projektdokumentationen. Für die Erstellung von Wiki-Seiten stehen drei verschiedene Markups zur Verfügung:

- Markdown
- Confluence
- Textile

Die gewünschten Markups müssen bereits bei der Erstellung des Projekts angegeben sein: Man sollte sie zu einem späteren Zeitpunkt nicht mehr verändern. Leider offenbart sich im Wiki-Service des DevCS noch eine kleine Schwäche: Für die Erstellung der Texte müssen die Autoren die Syntax des entsprechenden Markups kennen, da der Wiki-Bereich nicht über einen WYSIWYG-Editor verfügt. Auch können die Inhalte des Wikis leider nicht exportiert werden.

Im Bereich „Collaboration" bietet der DevCS auch einen Service zur Verwaltung von Snippets an. Snippets sind kleine Schnipsel von nützlichen und wiederverwendbaren Fragmenten wie Code oder Befehle, die man mit anderen Projekt-Mitgliedern teilen kann. Diese werden im DevCS wie Code behandelt und im Git-Repository gespeichert.

Die Speicherung der Snippets erfolgt in einem separaten Repository und somit getrennt vom eigentlichen Code der zu entwickelnden Anwendung. Somit kann der Entwickler diese mithilfe von Git-Befehlen anzeigen lassen, bearbeiten und ihren Verlauf verfolgen. In der Regel enthält ein Snippet entweder ein Skript oder eine Sequenz von Befehlen. Jedoch kann ein Snippet auch für private oder öffentliche Anmerkungen verwendet werden, die nicht im Wiki des Projekts erscheinen.

Administration

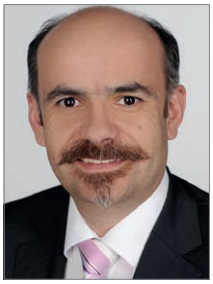
Über den Menüpunkt „Administration" kann der Administrator die Projekte administrieren. In diesem Bereich verändert er die Eigenschaften des Projekts, erfasst für das Issue Tracking weitere Komponenten wie Produkt, Release, Tags und Sprints, verwaltet interne wie externe Git-Repositories oder wertet die Metriken aus. Darüber hinaus kann er in der Administrationssicht „Webhooks" anlegen. Das sind Benachrichtigungen, die vom DevCS versendet werden, etwa bei der Aktualisierung eines Issue, dem Push in das Git-Repository, der Aktualisierung eines Merge Request oder wenn ein Build erstellt werden soll.

Fazit

Der DevCS stellt eine Plattform mit einer Vielzahl von Werkzeugen zur Verfügung, die bei der Anwendungsentwicklung unterstützen. Mit seiner Verwendung können Entwicklungsteams die Zeiten für Setup und Wartung erheblich minimieren. Insbesondere durch die Integration der vorgestellten Kategorien sind im DevCS die notwendigsten Tools integriert, sodass für viele Projekte keine weiteren zusätzlichen Tools beziehungsweise Plattformen benötigt werden und die Zusammenarbeit der Teams wesentlich vereinfacht werden kann.

Weiterführende Links

[1] *https://cloud.oracle.com/en_US/tryit*
[2] *https://plugins.jetbrains.com/plugin/7370?pr=idea und http://docs.oracle.com/en/cloud/paas/developer-cloud/csdcs*



Stefan Kühnlein
stefan.kuehnlein@opitz-consulting.com

Adaptive Features oder wie ich lernte, ruhig zu bleiben, um die Bombe zu beherrschen

Ludovico Caldara, Trivadis AG

Die Oracle-Datenbank 12c wurde im Jahr 2013 veröffentlicht. Es dauerte jedoch einige Zeit, bis Oracle-Anwender mit dem Upgrade ihrer Datenbanken auf die neue Version begonnen haben. Der Premier Support für die Version 11g endete bereits im Januar 2015, und doch haben im Jahr 2016 viele Anwender immer noch keine Migration nach 12c gestartet.

Der Autor hatte die Gelegenheit, die letzten beiden Jahre einen Kunden zu betreuen, der beschlossen hat, das neue Release einzusetzen und schnellstmöglich nach 12c zu migrieren, um die (zahlreichen) neuen Funktionalitäten, die diese Version bietet, optimal zu nutzen. Bei der Migration der allerersten Produktions-Datenbanken nach 12c erkannten die Anwender schon bald, dass einige Abfragen zeitintensiver waren als zuvor. Nach kurzen Nachforschungen fand der Autor heraus, dass diese Verzögerung bei den meisten Abfragen auf die neuen „adaptiven Features“ zurückzuführen ist, die in 12c aus genau dem entgegengesetzten Grund eingeführt wurden: zur Leistungssteigerung. Zunächst einmal eine kurze Zusammenfassung der neuen Features: Adaptive Features (Adaptive Query Optimization) gliedern sich in zwei Bereiche – adaptive Pläne und adaptive Statistiken *(siehe Abbildung 1)*.

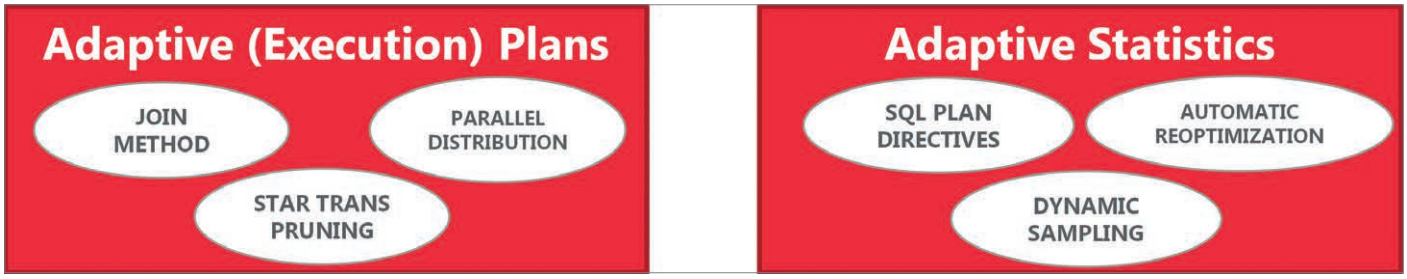


Abbildung 1: Adaptive Pläne und adaptive Statistiken

Adaptive Pläne

Wie der Name schon andeutet, sind adaptive Pläne Ausführungspläne, die sich während der Ausführung anpassen und mehr oder weniger dynamisch ändern können. Beim Parsen eines neuen Cursors ist sich der Optimizer möglicherweise nicht sicher, welche Operation die richtige ist. In diesem Fall erstellt er einen adaptiven Plan: Er definiert neben einem „Standard"-Ausführungsplan alternative Teil-Ausführungspläne, die im ersten Teil der Ausführung deaktiviert bleiben.

Über „STATISTICS COLLECTOR" wird entschieden, welche der Methoden tatsächlich ausgeführt werden soll (genau: ein Collector pro Entscheidungszweig im adaptiven Plan). Die Ergebnisse werden zwischengepuffert und es wird überprüft, ob der Schwellenwert erreicht ist. Gegebenenfalls deaktiviert der „STATISTICS COLLECTOR" den aktuellen Plan und aktiviert den alternativen Plan. Sollte es sich um die letzte Ausführungsoperation handeln, wird der „STATISTICS COLLECTOR" deaktiviert und der tatsächlich verwendete Plan als „endgültig" markiert. *Listing 1* zeigt zwei Joins, die sich bei der Ausführung des Statements ändern können. Um den vollständigen Ausführungsplan mitsamt inaktiven Operationen anzuzeigen, muss der Parameter „Format" mit dem Wert „+adaptive" angegeben sein.

Es gibt drei grundlegende Typen von dynamischen Plänen: „Parallel Distribution Method", „Join Method" und „Star Transformation Bitmap Pruning". Der Optimizer berechnet beim Parsing jeweils einen Schwellenwert: die Anzahl der Eingabe-Datensätze (Eingabe entweder für die parallele Verteilung aus der inneren Tabelle eines Joins oder nach der Dimension gefiltert in einem „BITMAP MERGE" bei einer Star Query Transformation), nach der eine Methode einer anderen vorzuziehen ist.

Mit der „Parallel Distribution Method" wird eine neue Verteilungsmethode namens „Hybrid Hash"eingeführt. Diese kann sich je nach Schwellenwert, wie hier im vorliegenden Falle dem zweimaligen Parallelitätsgrad, ändern. Sind die Zeilen größer oder gleich dem Schwellenwert, wird Hash/Hash verwendet, andernfalls Broadcast/Round-Robin. Dieses Verhalten entspricht der Datenbank-Version

```
SQL> select * from table(dbms_xplan.display(format=>'+adaptive'));
-------------------------------------------------------------------------------------------------------------------
|   Id  | Operation                                | Name            | Rows  | Bytes | Cost (%CPU)| Time
|
-------------------------------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT                         |                 |   665 | 35910 |     9   (0)| 00:00:01
|  *  1 |  HASH JOIN                               |                 |   665 | 35910 |     9   (0)| 00:00:01
|-    2 |   NESTED LOOPS                          |                 |   665 | 35910 |     9   (0)| 00:00:01
|-    3 |    STATISTICS COLLECTOR                  |                 |       |       |            |
|  *  4 |     HASH JOIN                            |                 |   105 |  4830 |     7   (0)| 00:00:01
|-    5 |      NESTED LOOPS                        |                 |   105 |  4830 |     7   (0)| 00:00:01
|-    6 |       STATISTICS COLLECTOR               |                 |       |       |            |
|     7 |        TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS          |   105 |   840 |     2   (0)| 00:00:01
|  *  8 |         INDEX RANGE SCAN                 | ORD_CUSTOMER_IX |   105 |       |     1   (0)| 00:00:01
|-    9 |       TABLE ACCESS BY INDEX ROWID        | CUSTOMERS       |     1 |    38 |     5   (0)| 00:00:01
|- * 10 |        INDEX UNIQUE SCAN                 | CUSTOMERS_PK    |       |       |            |
|    11 |      TABLE ACCESS FULL                   | CUSTOMERS       |   319 | 12122 |     5   (0)| 00:00:01
|- * 12 |    INDEX RANGE SCAN                      | ORDER_ITEMS_UK  |     6 |    48 |     2   (0)| 00:00:01
|    13 |   INDEX FAST FULL SCAN                   | ORDER_ITEMS_UK  |   665 |  5320 |     2   (0)| 00:00:01
-------------------------------------------------------------------------------------------------------------------
Note
-----
   - this is an adaptive plan (rows marked '-' are inactive)
```

Listing 1

```
SQL> select sql_id, child_number, IS_RESOLVED_ADAPTIVE_PLAN
2 from v$sql where sql_text like 'SELECT C.CUST_EMAIL%';

SQL_ID        CHILD_NUMBER I
------------- ------------ -
8d1g1zszjk0ky            0 Y
```

Listing 2

```
-- snippet of trace 100053
...
AP: Costing Nested Loops Join for inflection point at card 26.25
AP: Costing Hash Join for inflection point at card 26.25
AP: Searching for inflection point at value 26.25
...
```

Listing 3

12.1, in den nachfolgenden Versionen wird sich dies leicht verändern.

Mit der „Join Method" berechnet der Optimizer einen Schwellenwert. Wird dieser bei der Ausführung der Abfrage erreicht, kann der Wechsel zwischen „Nested Loops" und „Hash Join" stattfinden (oder umgekehrt, je nachdem wie viele Zeilen beim Parsen bezüglich der inneren Tabelle geschätzt wurden).

Ein beträchtlicher Unterschied zwischen „Join Method" und „Parallel Distribution" besteht darin, dass bei Letzterem bei jeder Cursor-Ausführung der Ausführungsplan wechseln kann, während dies bei der „Join Method" nur bei der allerersten Ausführung möglich ist. Die Spalte „IS_RESOLVED_ADAPTIVE_PLAN" in „V$SQL" gibt an, ob der endgültige Plan gewählt wurde *(siehe Listing 2)*.

Was bedeutet dies aus Performance-Sicht? Da nur die erste Ausführung den endgültigen Plan beeinflussen kann, ändert sich im Vergleich zur den herkömmlichen Ausführungsplänen eigentlich nichts. Ist der endgültige Plan wirksam, so wendet der Cursor immer denselben Plan an, es sei denn, „Adaptive Cursor Sharing" greift ein und erzeugt einen Child-Cursor, der vermutlich einen anderen adaptiven Plan generieren wird.

Wie wird der Schwellenwert für die Join-Methode berechnet? Aus der Optimizer-Trace-Datei einer Abfrage, die einen adaptiven Plan erzeugt, wird ersichtlich, dass der Optimizer versucht, die beiden Operationen (NL und HJ) für eine unterschiedliche Zeilenzahl zu kalkulieren, die möglicherweise von der inneren Tabelle ausgegeben werden *(siehe Listing 3)*.

Was bedeutet dies aus Performance-Sicht? Da nur die erste Ausführung den endgültigen Plan beeinflussen kann, ändert sich im Vergleich zur den herkömmlichen Ausführungsplänen eigentlich nichts. Ist der endgültige Plan wirksam, so wendet der Cursor immer denselben Plan an, es sei denn, „Adaptive Cursor Sharing" greift ein und erzeugt einen Child-Cursor, der vermutlich einen anderen adaptiven Plan generieren wird.

Wie wird der Schwellenwert für die Join-Methode berechnet? Aus der Optimizer-Trace-Datei einer Abfrage, die einen adaptiven Plan erzeugt, wird ersichtlich, dass der Optimizer versucht, die beiden Operationen (NL und HJ) für eine unterschiedliche Zeilenzahl zu kalkulieren, die möglicherweise von der inneren Tabelle ausgegeben werden *(siehe Listing 3)*.

Ohne Trace-Datei ist es nicht möglich, den Schwellenwert für ein Statement zu ermitteln. Es lässt sich zudem nur schwer herausfinden, ob der Plan sich geändert hat: Wenn ein Plan auf die Join-Methode umschaltet, ändert sich der Wert „PLAN_HASH_VALUE" in den dynamischen Views, doch dem DBA stehen keinerlei sonstige Indizien für die Änderung zur Verfügung.

Was die Fehlerbehebung betrifft, fehlen in diesem Release nach Meinung des Autors einige Informationen. Erfahrungsgemäß stellen adaptive Ausführungspläne an sich jedoch nur sehr selten eine Fehlerquelle dar. Schlechte Pläne werden eher durch andere Faktoren bewirkt wie zum Beispiel schlechte Statistiken im Data-Dictionary oder aus den Adaptive Statistics Features.

Der letzte Typ heißt „Star Transformation Bitmap Pruning". Im Star-Schema wird die Faktentabelle mit verschiedenen Dimensionen-Tabellen verbunden. Für jede Dimension mit Filterkriterien findet eine Art Join zwischen Bitmap-Index auf der Fakten- und der Dimensions-Tabelle statt. Ergibt die Join-Abfrage viele Zeilen, kann die Operation sich negativ auf die Performance auswirken. Wenn der „STATISTICS COLLECTOR" mit der „Star Transformation Bitmap Pruning"-Methode erkennt, dass die Dimension nicht genug filtert, kann für den Zugriff zur entsprechenden Dimension Pruning genutzt werden: Der Plan überspringt die Dimension einfach und verbindet sie später.

Adaptive Statistiken

Adaptive Statistiken sind eine Reihe von Funktionalitäten, zu denen „Automatic Reoptimization", „SQL Plan Directives" und „Dynamic Statistics" gehören. Einfach ausgedrückt, besteht Automatic Reoptimization aus zwei Features: „Statistics Feedback", vormals als Cardinality Feedback bereits in 11.2 vorhanden, sowie „Performance Feedback".

In diesem Artikel wird nicht näher auf „Performance Feedback" eingegangen, da dieses Feature (zumindest nach Erfahrung des Autors) kaum negative Auswirkungen auf die Performance der Datenbanken hat. Erwähnenswert ist jedoch, dass abhängig vom Parameter „parallel_degree_policy" Performance-Statistiken von parallelen Ausführungen gesammelt werden können. Wenn der DOP nicht optimal ist, wird eine Reoptimierung bei der nächsten Ausführung des Statements ausgelöst.

Auch „Statistics Feedback" hat sich in 12c geändert. Eingeführt wurde dieses Feature in Release 11.2 unter dem Namen „Cardinality Feedback". Einfach ausgedrückt hat es die Aufgabe, bei der Ausführung herauszufinden, wie stark die tatsächlichen Kardinalitäten von denen abweichen, die der Optimizer geschätzt hat. Bei signifikanter Abweichung wird eine Korrektur in der SQL-Area gespei-

chert, um für nachfolgende Ausführungen derselben Anweisung einen besseren Plan zu finden. Gegebenenfalls wird die sofortige Reoptimization des Statements ausgelöst: Es wird bei der nächsten Ausführung erneut geparst und ein neuer Child-Cursor generiert. In 12c ist diese Funktionalität um Informationen zu Join-Kardinalitäten erweitert. Ob eine Reoptimierung erfolgt, kann über die Spalte „V$SQL.IS_REOPTIMIZABLE“ abgefragt werden *(siehe Listing 4)*.

Bei den Hinweisen der zweiten Ausführung werden die beiden anderen Adaptive-Statistics-Features erwähnt: „Dynamic Statistics“ und „SQL Plan Directives“. „Dynamic Statistics“ ist die neue Bezeichnung für Dynamic Sampling in 12c. Mit der neuen Version wird eine Verbesserung des bisherigen „Dynamic Sampling“, namens „Adaptive Dynamic Sampling“ (ADS), eingeführt. Der „Dynamic Sampling“-Level (Parameter „optimizer_dynamic_sampling“) gibt an, in welchem Umfang „Dynamic Sampling“ zum Einsatz kommt und wie viele Datenblöcke in diesem Fall gelesen werden.

Ist in 12c der Parameter auf den Level „11“ gesetzt, greift Adaptive Dynamic Sampling ein, die Anzahl der gelesenen Datenblöcke ist dabei nicht festgelegt. Deren tatsächliche Anzahl kann eine große Variation aufweisen und die Anzahl der ausgeführten Dynamic-Sampling-Abfragen beim Parsing jeder Abfrage kann leicht mehrere Dutzende erreichen. Der verwendete Level des „Dynamic Sampling“ und Texte sind in SQL-Trace-Dateien und in der SQL-Area hinterlegt, sie beginnen jeweils mit „SELECT /* DS_SVC */“ *(siehe Listing 5)*.

Aus diesem Grund kann die Parse-Zeit für das Statement bis zu mehreren Sekunden (beziehungsweise Minuten) dauern, sogar bei Abfragen, deren Ausführung nur den Bruchteil einer Sekunde in Anspruch nehmen würde. Aus Performance-Sicht wird deutlich, dass Anwendungen mit viel Parsing keinen großen Nutzen aus „Adaptive Dynamic Sampling“ ziehen. Von dieser Funktionalität profitieren hauptsächlich OLAP-Umgebungen.

Trotz der Tatsache, dass der Default-Wert des Parameters „optimizer_dynamic_sampling“ gleich „2“ ist, kommt „Adaptive Dynamic Sampling“ zum Einsatz, wenn die parallele Ausführung in Erwägung gezogen wird, oder bei Anwendung von „SQL Plan Directives“ beim Parsing.

Statistics Feedback speichert die Korrekturdaten der falsch eingeschätzten Kardinalitäten in der SQL-Area für einen bestimmten Cursor. Diese Daten gehen verloren, wenn der Cursor veraltet ist oder aus dem Shared Pool entfernt wird. In 12c persistiert Statistics Feedback die Informationen zu involvierten Objekten/Spalten in sogenannte „SQL Plan Directives“. Dort werden keine Kardinalitätsdaten gespeichert. Dahinter steckt der Gedanke, dass eine Fehleinschätzung bezüglich einer Operation mit einer Tabel-

```
SQL> select count(*) from t where val=1 and id>0;

  COUNT(*)
----------
     50000

SQL> select sql_id, child_number, executions, is_reoptimizable from
v$sql where sql_id='04drgw5qk4m3u';

SQL_ID        CHILD_NUMBER EXECUTIONS I
------------- ------------ ---------- -
04drgw5qk4m3u            0          1 Y

SQL>  select count(*) from t where val=1 and id>0;

  COUNT(*)
----------
     50000

SQL>  select * from table(dbms_xplan.display_cursor(format=>'allstats
last'));
[...]
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - statistics feedback used for this statement
   - 1 Sql Plan Directive used for this statement
```

Listing 4

```
SELECT /* DS_SVC */ /*+ dynamic_sampling(0) no_sql_tune no_moni-
toring optimizer_features_enable(default) no_parallel result_
cache(snapshot=3600) */ SUM(C1) FROM (SELECT /*+ qb_name("innerQuery")
NO_INDEX_FFS( "LI")  */ 1 AS C1 FROM ADAPTIVE."PRODUCT" SAMPLE
BLOCK(1.85843, 8) SEED(1)  "LI" WHERE ("LI"."ENT_ID"=194924)) inner-
Query
```

Listing 5

```
SQL> select count(*) from t where val=1 and id>0;
```

Listing 6

le/Spalte (Joins, Gleichheitsbedingungen etc.) sich gegebenenfalls auch bei einem anderen Statement mit ähnlichen Bedingungen wiederholen kann. Daher können Abfragen nicht nur von „SQL Plan Directives" profitieren, wenn sie diese selbst erstellt haben, sondern diese können von allen Statements, die einen ähnlichen Zugriff auf das Objekt beinhalten, verwendet werden. *Listing 6* zeigt die Erstellung einer „SQL Plan Directive" für die Abfrage.

In 11.2 würde eine leicht geänderte Abfrage mit derselben Bedingung zu einem „Hard Parse" führen und denselben Index aufgrund falscher Statistiken wiederverwenden. Da die Spalten in der „Where"-Klausel jedoch dieselben sind, gibt die „SQL Plan Directive" dem Optimizer den Befehl, stattdessen „Adaptive Dynamic Sampling" anzuwenden. Der Plan verwendet ab der ersten Ausführung die „TABLE ACCESS FULL"-Operation *(siehe Listing 7)*. Dies beweist, dass „SQL Plan Directives" vorteilhaft sind und die Performance zahlreicher Statements verbessern können.

Die „SQL Plan Directives" sind in der View „DBA_SQL_PLAN_DIRECTIVES" aufgelistet, die betreffenden Objekte/Spalten können aus „DBA_SQL_PLAN_DIR_OBJECTS" selektiert werden. Es werden drei Arten von Fehleinschätzungen berücksichtigt (Spalte „DBA_SQL_PLAN_DIRECTIVES.REASON"):

- JOIN CARDINALITY MISESTIMATE
- SINGLE TABLE CARDINALITY MISESTIMATE
- GROUP BY CARDINALITY MISESTIMATE

In allen Fällen weisen „SQL Plan Directives" den Optimizer an, beim Statement-Parsing „Dynamic Sampling" durchzuführen. Bei der Fehleinschätzung von Kardinalitäten einzelner Tabellen, die mehrere Spalten betreffen, können „SQL Plan Directives" auch zur Erstellung von „Extended Statistics" (Column Groups) führen. In diesem Fall wird „Dynamic Sampling" noch so lange durchgeführt, bis die Statistiken der neu erstellten Spaltengruppe erfasst sind. Danach ist die entsprechende „SQL Plan Directive" veraltet und wird vom Optimizer nicht mehr angewendet.

Die Bombe

Wie kommt es, dass die Anwender sich bei all den cleveren, neuen und adaptiven Funktionalitäten über eine schwache Performance in der Version 12c beschweren? Wie alle neuen Funktionalitäten funktionieren die neuen „Adaptive Query Optimization"-Features in den meisten Situationen bestens. In bestimmten komplexeren Fällen können sie jedoch katastrophale Ergebnisse herbeiführen. Je nach den spezifischen Gegebenheiten stoßen Anwender auf einen sehr hohen Verbrauch von CPU-Ressourcen, übermäßige „Library Cache"-Konflikte, die vermehrte Verwendung des „Result Cache", was dazu führt, dass dessen Größe nicht ausreicht, und suboptimale Ausführungspläne.

Die Formel ist stets gleich: „Statistics Feedback" generiert „SQL Plan Directives" auf Basis von Fehleinschätzungen. „SQL Plan Directives" erzwingen „Adaptive Dynamic Sampling". Dieses verbraucht viele Ressourcen, generiert Konflikte und versorgt den Optimizer manchmal mit, im Vergleich zu den gespeicherten Statistiken, suboptimalen dynamischen Statistiken. In 12c ist „Statistics Feedback" besonders offensiv und kann aus verschiedenen Gründen „SQL Plan Directives" erstellen:

- Fehlende Spaltengruppen
- Histogramme der falschen Art/Größe
- Fehleinschätzung von Join-Kardinalitäten: Dies geschieht besonders häufig durch massenhafte Erstellung von „SQL Plan Directives“ auf hochnormalisierten Schemata und für fast jede Abfrage, die mehr als ein paar Joins enthält

Auch die meisten Abfragen der von den DBAs genutzten „Dictionary Views“ wenden „Adaptive Dynamic Sampling“ aufgrund der „SQL Plan Directives“ an *(siehe Listing 8)*.

Wie dem Beispiel zu entnehmen ist, können die von einer Anweisung genutzten „SQL Plan Directives“ ermittelt werden, indem ein „Explain Plan“ für das Statement ausgeführt wird und der Plan im „+metrics“-Format angezeigt wird. Eine andere praktische Möglichkeit, die verwendeten „SQL Plan Directives“ anzuzeigen, ist die Erstellung eines Optimizer-Trace (Event „10053“).

Hinweis: Im vorhergehenden Beispiel wird im „Explain Plan“ das „Dynamic Sampling“-Level „2“ angezeigt – ein Fehler in 12.1.0.2. Das tatsächlich verwendete Level ist Level „11“. Wenn „Dynamic Sampling“ aufgrund der „SQL Plan Directives“ zum Einsatz kommt, wird in 12.1.0.2 stets Level „11“ (ADS) verwendet.

Manchmal werden „SQL Plan Directives“ generiert, obwohl der Optimizer einen guten Plan ermittelt hat: Enthält eine große Tabelle Millionen von verschiedenen Werten und liegt eine starke Ungleichverteilung der Daten vor, kann mit den Histogrammen möglicherweise keine perfekte Schätzung erreicht werden und „Statistics Feedback“ greift unter Umständen trotz eines guten Ausführungsplans ein. In diesem Fall kann das „Adaptive Dynamic Sampling“ anschließend zu einer noch schlechteren Schätzung führen, wodurch der Ausführungsplan des Optimizers schlecht wird. Es gibt viele Lösungen für dieses Problem:

- Manche Anwender beschließen, alle adaptiven Features zu deaktivieren, indem sie „optimizer_adaptive_features“ auf „false“ setzen. Dadurch ist sowohl die Erstellung von „SQL Plan Directives“ als auch die Verwendung von adaptiven Plänen abgeschaltet. „Adaptive Dynamic Sampling“ ist jedoch immer noch verfügbar und wird für parallele Ausführungen verwendet oder wenn der „optimizer_dynamic_sampling“ auf „11“ gesetzt ist.
- Andere Anwender setzen „optimizer_features_enabled“ auf die Version 11.2.0.4 (oder älter) zurück. Dies ist in der Regel keine gute Idee, da dadurch viele Verbesserungen, die die neue Version mitbringt, verloren gehen.

Es gibt selbstverständlich andere, weniger drastische Lösungen:

- Durch Nullsetzen des verborgenen Parameters „_optimizer_dsdir_usage_control“ wird das Auslösen von „Adaptive Dynamic Sampling“ durch „SQL Plan Directives“ deaktiviert, jedoch nicht die Erstellung der „SQL Plan Directives“.
- Wird der verborgene Parameter „_optimizer_enable_extended_stats“ auf „false“ gesetzt, deaktiviert dies die Anwendung von „Extended Statistics“ (jedoch nicht deren Erstellung).
- Wird der verborgene Parameter „_optimizer_adaptive_plans“ auf „false“ gesetzt, deaktiviert dies die Erstellung von adaptiven Plänen (im Allgemeinen generieren adaptive Pläne selten Fehler – es kann jedoch zutreffen).
- Durch Nullsetzen des verborgenen Parameters „_sql_plan_directive_mgmt_control“ wird die Erstellung von „SQL Plan Directives“ deaktiviert. Dies ist wohl der praktischste und nützlichste Weg, um Performance-/Stabilitätspro-

```
SQL> select count(*)+1 from t where val=1 and id>0;

COUNT(*)+1
----------
     50001

SQL> select * from table(dbms_xplan.display_cursor(format=>'allstats
last'));
...
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 2 Sql Plan Directives used for this statement
```

Listing 7

```
SQL> explain plan for select count(*) from dba_tables;

Explained.

SQL> select * from table(dbms_xplan.display(format=>'metrics'));
...
Sql Plan Directive information:
------------------------------

  Valid directive ids:
    250882735634668447
    9542766902214822429
    2083381208315424718
    6551113436581718606
    16273932978853091121

  Used directive ids:
    11259849835960924452
    3270421438167445494
    2733281086066737731
```

Listing 8

```
SQL> explain plan for select count(*)+1 from t where val=1 and id>0;

Explained.

SQL> select * from table(dbms_xplan.display(format=>metrics'));
...
Sql Plan Directive information:
------------------------------

  Valid directive ids:
    17154883816957472097
    18093103661535947380

SQL>  BEGIN
  2    FOR rec IN
  3    (SELECT d.directive_id AS did
  4    FROM dba_sql_plan_directives d
  5    JOIN dba_sql_plan_dir_objects o
  6    ON (d.directive_id  =o.directive_id)
  7    WHERE o.owner       ='OE'
  8    AND d.directive_id IN (17154883816957472097,18093103661535947380)
  9    )
 10    LOOP
 11      DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE ( rec.did, 'ENABLED','NO');
 12      DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE ( rec.did, 'AUTO_DROP','NO');
 13    END LOOP;
 14 END;
 15 /

PL/SQL procedure successfully completed.
```

Listing 9

bleme bei frisch upgegradeten Datenbanken zu vermeiden.
- Durch Nullsetzen des Parameters „optimizer_adaptive_sampling“ wird die Anwendung von „Dynamic Sampling“ global deaktiviert, auch bei vorhandenen „SQL Plan Directives“.
- Die selektive Deaktivierung von „SQL Plan Directives“ ist auch eine Lösung (die sanfteste); dieser Vorgang muss auf „Per Directive“-Basis erfolgen. Bitte beachten: Die Deaktivierung einer Directive mit dem internen Status „MISSING_STATS“ deaktiviert die Erstellung von „Extended Statistics“ nicht. *Listing 9* zeigt ein Beispiel zur Deaktivierung von „SQL Plan Directives“, die von der Abfrage verwendet werden.

In 12.2 ändert sich einiges

Hinweis: Zum Zeitpunkt dieses Artikels ist Oracle 12.2 nur in der Oracle Cloud verfügbar. Die aufgezeigten Informationen gelten für das Oracle-Cloud-Release und sollten sich mit der GA-Version nicht ändern, was jedoch nicht garantiert werden kann:

- Im nächsten Release lösen „SQL Plan Directives“ ohne expliziten Befehl kein „Adaptive Dynamic Sampling“ mehr aus. Die Erstellung von „SQL Plan Directives“ ist nach wie vor standardmäßig aktiviert.
- „Statistics Feedback“ wird bei der Erstellung neuer „SQL Plan Directives“ weniger offensiv vorgehen. Es wird insbesondere bei Join-Fehleinschätzungen nicht eingesetzt.
- Der Parameter „optimizer_adaptive_features“ wird in zwei verschiedene Parameter „optimizer_adaptive_plans“ (standardmäßig „true“) und „optimizer_adaptive_statistics“ (standardmäßig „false“) aufgespaltet.
- Durch diese beiden neuen Parameter kann nur ein Teil der Features aktiviert werden, was den DBAs eine bessere Kontrolle ermöglicht.
- Es wird eine neue „DBMS_STATS (AUTO_STATS_EXTENSION)“-Eigenschaft eingeführt, die eine Deaktivierung der automatischen Erstellung von „Extended Statistics“ ermöglicht (diese neue Eigenschaft ist derzeit auch in 12.1.0.2 nach Installation des Patch „21171382“ verfügbar). Die Erstellung von „Extended Statistics“ ist standardmäßig abgeschaltet.

Die neuen Features in 12.2 lassen vermuten, dass dies in 12.1.0.2 über Patch „21171382“ (keine automatische Erzeugung von „Extended Statistics“) und Nullsetzen von „_optimizer_dsdir_usage_control“ ähnlich gehandhabt wird. Genaueres hierzu werden wir erfahren, sobald 12.2 allgemein verfügbar ist.

Hinweis: Die kompletten Listings zu diesem Thema stehen unter *„http://www.ludovicocaldara.net/dba/adaptive-features-code-listing/“*.

Links und Quellen

- *https://blogs.oracle.com/optimizer/entry/optimizer_adaptive_features_in_the*
- *https://antognini.ch/2016/10/adaptive-query-optimization-configuration-parameters-preferences-and-fix-controls/*
- *http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-12c-1963236.pdf*
- *http://www.slideshare.net/pachot/soug-2014-sqlplandirectives*
- *http://www.itoug.it/wp-content/uploads/2016/05/AdaptiveDynamicSampling_OTNMi_2016.pdf*
- *http://blog.dbi-services.com/matching-sql-plan-directives-and-queries-using-it/*
- *https://community.oracle.com/docs/DOC-918264*
- *https://blogs.oracle.com/optimizer/entry/dynamic_sampling_and_its_impact_on_the_optimizer*
- *http://www.toadworld.com/platforms/oracle/w/wiki/11453.spd-sql-plan-directives-in-12c-part-i*

Ludovico Caldara
ludovico.caldara@trivadis.com

Objektorientierte Entwicklung in der Datenbank – geht denn das?

Anja Hildebrandt, buw Management Holding GmbH & Co. KG

Objektorientierte Entwicklung bildet seit vielen Jahren die Grundlage für viele Software-Entwicklungsprojekte, da sie sich besonders gut für die Implementierung wiederverwendbarer Komponenten und komplexer Anwendungen eignen. Dass aber auch die Oracle-Datenbank Möglichkeiten zur objektorientierten Programmierung bietet, wissen die wenigsten. Dieser Artikel zeigt die Grundlagen der objektorientierten Entwicklung mittels Objekt-Typen.

In PL/SQL basiert die objektorientierte Programmierung auf Objekt-Typen, die bereits seit Oracle 8*i* zur Verfügung stehen, aber immer noch ein Schattendasein fristen.

Objekt-Typen sind durch den Entwickler definierte Datentypen, die es ihm erlauben, Objekte der realen Welt (wie Bücher, Konten, Kunden) in der Datenbank abzubilden.

```
CREATE OR REPLACE TYPE t_customer AS OBJECT
( customer_id          NUMBER,
  cust_street_address  VARCHAR2(60),
  cust_city            VARCHAR2(30),
  cust_postal_code     VARCHAR2(10),
  cust_email           VARCHAR2(30),
  phone_number         VARCHAR2(25)
);
```

Listing 1: Einfache Deklaration eines Objekt-Typs

```
DECLARE
  customer t_customer;
BEGIN
  customer := NEW t_customer(123, 'Rheiner Landstraße
195','Osnabrück','49078','anja.hildebrandt@buw.de','0123/1234567');

  dbms_output.put_line('Id: ' || customer.customer_id);
  dbms_output.put_line('Address: ' || customer.cust_street_address ||
', ' || customer.cust_postal_code || ' ' || customer.cust_city);
END;

--

Id: 123
Address: Rheiner Landstraße 195, 49078 Osnabrück
```

Listing 2: Instanziierung eines Objekt-Typs

```
CREATE OR REPLACE TYPE t_customer AS OBJECT
( customer_id          NUMBER,
  cust_street_address  VARCHAR2(60),
  cust_city            VARCHAR2(30),
  cust_postal_code     VARCHAR2(10),
  cust_email           VARCHAR2(30),
  phone_number         VARCHAR2(25),

  MAP MEMBER FUNCTION getSalesVolume RETURN NUMBER,
  NOT FINAL MEMBER PROCEDURE setDefaults,
  NOT FINAL NOT INSTANTIABLE MEMBER PROCEDURE persist
) NOT FINAL
  NOT INSTANTIABLE;
```

Listing 3: Angepasste Spezifikation für „t_customer“

```
CREATE OR REPLACE TYPE t_customer_b2c UNDER t_customer
( cust_first_name      VARCHAR2(50),
  cust_last_name       VARCHAR2(50),
  credit_limit         NUMBER(9,2),

  CONSTRUCTOR FUNCTION t_customer_b2c RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION t_customer_b2c(p_customer_id IN NUMBER) RETURN
SELF AS RESULT,

  OVERRIDING MEMBER PROCEDURE setDefaults,
  MEMBER PROCEDURE persist,
  OVERRIDING STATIC FUNCTION getCustomerType RETURN VARCHAR2
) FINAL;
\

CREATE OR REPLACE TYPE t_customer_b2b UNDER t_customer
( company_name      VARCHAR2(50),
  sales_discount    NUMBER(2),

  CONSTRUCTOR FUNCTION t_customer_b2b RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION t_customer_b2b(p_customer_id IN NUMBER) RETURN
SELF AS RESULT,

  OVERRIDING MEMBER PROCEDURE setDefaults,
  MEMBER PROCEDURE persist,
  OVERRIDING STATIC FUNCTION getCustomerType RETURN VARCHAR2
) FINAL;
\
```

Listing 4: Spezifikation für untergeordnete Typen

In der Objektorientierung würde man sie als „Klassen“ bezeichnen. Da ein Objekt-Typ eine Art Daten-Typ ist, ist er auf dieselbe Weise wie herkömmliche Daten-Typen wie „NUMBER“ oder „VARCHAR2“ nutzbar.

Deklaration und Instanziierung

Listing 1 zeigt die Deklaration eines einfachen Objekt-Typs. Im Beispiel werden keine Methoden deklariert, jedoch besitzt jeder Objekt-Typ automatisch einen standardisierten Konstruktor. *Listing 2* zeigt die Initialisierung einer Instanz des zuvor deklarierten Objekt-Typs. Wie zu erkennen ist, erfordert der Standard-Konstruktor, dass für jedes Attribut ein Wert angegeben wird. Es ist natürlich möglich, eigene Konstruktoren zu definieren. Dazu später mehr.

Das Keyword „NEW“ ist optional. Es kann auch weggelassen werden, erhöht aber bei Verwendung die Lesbarkeit des Quellcodes.

Vererbung

Die Vererbung von Attributen und Methoden ist ein Kern-Feature der objektorientierten Entwicklung und natürlich auch im Bereich der Objekt-Typen verfügbar. Um die Verwendung zu demonstrieren, passen wir zunächst unsere Deklaration für den Typ „t_customer“ etwas an *(siehe Listing 3)*. Der Typ erhält am Ende der Spezifikation die Stichwörter „NOT FINAL“ und „NOT INSTANTIABLE“. „NOT FINAL“ bedeutet hier, dass von diesem Typ abgeleitet werden kann.

Die Verwendung von „NOT INSTANTIABLE“ verhindert die Erstellung einer Instanz des Typs selbst. Die Instanziierung in *Listing 2* würde nun auf einen Fehler laufen. Auf diese Art ist es möglich, in PL/SQL einen abstrakten Objekt-Typ beziehungsweise eine abstrakte Klasse zu erstellen.

Da sich die Stichwörter sowohl auf den gesamten Typ als auch auf einzelne Methoden anwenden lassen, ist auch die Definition von abstrakten Methoden innerhalb eines abstrakten Typs möglich (Methode „persist“ in *Listing 3*).

Erstellen wir nun zwei vom Typ „t_customer“ abgeleitete Objekt-Typen „t_customer_b2b“ und „t_customer_b2c“ *(siehe Listing 4)*. Hier wird das Stichwort „UNDER“ genutzt, um die Vererbung anzuzeigen. Alle Attribute und Methoden, die im übergeordneten Typ deklariert wurden, stehen nun auch den neuen, untergeordneten Objekt-Typen zur Verfügung und werden um weitere Attribute und Methoden erweitert.

Die Methode „setDefaults“ wurde bereits im Typ „t_customer“ deklariert, jedoch mit dem Stichwort „NOT FINAL“ versehen, was es den abgeleiteten Typen ermöglicht, diese Methode zu überschreiben. Dies erfolgt durch das Keyword „OVERRIDING“ vor der jeweiligen Methoden-Deklaration.

Methoden

Nachdem nun eine kleine Objekt- beziehungsweise Klassen-Hierarchie etabliert ist, wird es Zeit, sich die verschiedenen verfügbaren Methodenarten näher anzusehen. Sobald Methoden in einer Typ-Spezifikation deklariert sind, müssen sie auch im Type-Body implementiert werden. Ausgenommen hiervon sind abstrakte Methoden.

Sind alle Methoden eines abstrakten Typs auch abstrakte Methoden, so kommt dieser Typ sogar gänzlich ohne Body aus. In allen anderen Fällen ist die Erstellung eines Type-Body nötig. *Listing 5* zeigt die „CREATE TYPE BODY“-Statements für unsere Beispiel-Typen.

Betrachten wir zunächst den abstrakten Typ „t_customer“. Wie bereits erwähnt, benötigt die abstrakte Methode „persist“ an dieser Stelle keine Implementierung, muss aber von abgeleiteten Typen implementiert sein.

Die Methode „setDefaults“ ist eine einfache „MEMBER“-Methode und dient hier der Initialisierung einiger Attribute mit Default-Werten. Sie kann zum Beispiel in einem Konstruktor aufgerufen werden, wenn eine neue Instanz eines Typs erstellt und mit Standardwerten vorausgefüllt werden soll. Eine „MEMBER“-Methode kann sowohl eine Funktion als auch eine Prozedur sein. Da sie der Manipulation von Attributen einer Instanz dient, kann sie nur auf einer bestehenden Instanz eines Objekt-Typs aufgerufen wer-

```
CREATE OR REPLACE TYPE BODY t_customer IS

  MAP MEMBER FUNCTION getSalesVolume RETURN NUMBER IS
    res NUMBER;
  BEGIN
    SELECT SUM(ORDER_TOTAL)
      INTO res
      FROM demo_orders
     WHERE customer_id=self.customer_id;
    RETURN res;

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 0;
  END;

  NOT FINAL MEMBER PROCEDURE setDefaults IS
  BEGIN
    self.customer_id:=0;
    RETURN;
  END;
END;
\

CREATE OR REPLACE TYPE BODY t_customer_b2c IS

  CONSTRUCTOR FUNCTION t_customer_b2c RETURN SELF AS RESULT IS
  BEGIN
    self.setDefaults();
    RETURN;
  END;

  CONSTRUCTOR FUNCTION t_customer_b2c(p_customer_id IN NUMBER) RETURN
SELF AS RESULT IS
  BEGIN
    SELECT customer_id, cust_first_name, ...
      INTO self.customer_id, self.cust_first_name, ...
      FROM demo_customers
     WHERE customer_id = p_customer_id;
    RETURN;
  END;

  OVERRIDING MEMBER PROCEDURE setDefaults IS
  BEGIN
    (SELF as t_customer).setDefaults();
    self.credit_limit := 1000;
    return;
  END;

  MEMBER PROCEDURE persist IS
  BEGIN
    IF self.customer_id=0 then
      SELECT demo_cust_seq.nextval
        INTO self.customer_id
        FROM dual;
    END IF;

    MERGE INTO demo_customers ...;
    RETURN;
  END;

  STATIC FUNCTION getCustomerType RETURN VARCHAR2 IS
  BEGIN
    RETURN 'B2C';
  END;
END;
\
```

Listing 5: Implementierung der spezifizierten Methoden

```
DECLARE
  cusB2B t_customer := NEW t_customer_b2b(p_customer_id => 1);
  cusB2C t_customer := NEW t_customer_b2c(p_customer_id => 2);
BEGIN
  IF cusB2B > cusB2C THEN
    dbms_output.put_line('Kunde 1 > Kunde 2');
  END IF;
END;
```

Listing 6: Vergleich zweier Instanzen über die „MAP"-Methode

```
ORDER MEMBER FUNCTION compare(obj in t_customer_b2c) RETURN NUMBER IS
  obj_salesVolume NUMBER;
  self_salesVolume NUMBER;
BEGIN
  SELECT nvl(SUM(ORDER_TOTAL)  ,0)
    INTO obj_salesVolume
    FROM demo_orders
   WHERE customer_id=obj.customer_id;

  SELECT nvl(SUM(ORDER_TOTAL)  ,0)
    INTO self_salesVolume
    FROM demo_orders
   WHERE customer_id=self.customer_id;

  IF self_salesVolume>obj_salesVolume then
    RETURN 1;
  ELSIF self_salesVolume<obj_salesVolume then
    RETURN -1;
  ELSE
    RETURN 0;
  END IF;
END;
```

Listing 7: Alternative Implementierung einer Vergleichsmethode („ORDER")

den. Jede „MEMBER"-Methode kann auf die Attribute der aktuellen Instanz über das Stichwort „SELF" zugreifen. Es wird die Punktnotation verwendet.

Unser Typ „t_customer" enthält außerdem eine „MAP MEMBER"-Methode. „MAP" und „ORDER"-Methoden dienen dem Vergleich zweier Objekt-Instanzen. Was bedeutet das in unserem Beispiel?

Skalare Datentypen wie „NUMBER" oder „DATE" haben vordefinierte Order-Methoden. Wir wissen, dass 7 > 5 und 24.12.2016 > 01.06.1980 gilt. Aber wie können wir Kunden vergleichen? Der Einfachheit halber definieren wir, dass die Sortier-Reihenfolge der Kunden auf der Summe ihrer bisherigen Bestellungen basiert.

Um nun die Möglichkeit zu schaffen, dass auch Kunden-Objekte wie skalare Daten-Typen verglichen werden können, gibt es zwei Möglichkeiten. Zum einen die „MAP"-Methode, die die aktuelle Instanz eines Objekt-Typs auf einen skalaren Wert abbildet (Methode „getSalesVolume" in *Listing 5*). Diese skalaren Werte lassen sich dann vergleichen *(siehe Listing 6)*.

Die zweite Möglichkeit wäre eine „ORDER"-Methode *(siehe Listing 7)*. Diese Variante vergleicht zwei Objekte desselben Typs direkt und eignet sich daher gut, wenn der Vergleich sehr komplexe Betrachtungen erfordert und kein einfaches Mapping auf einen skalaren Wert möglich ist. Zu bemerken ist, dass innerhalb einer Hierarchie nur eine der beiden Methoden genutzt werden kann und dass diese auf der obersten Ebene der Hierarchie implementiert sein muss.

Methoden in abgeleiteten Typen

Betrachten wir nun die Implementierung eines abgeleiteten Objekt-Typs am Beispiel „t_customer_b2c" in *Listing 5*. Hier finden wir unter anderem ein einfaches Beispiel für eine statische Methode. Statische Methoden werden mit dem Stichwort „STATIC" gekennzeichnet; sie sind Methoden des Typs und nicht der Instanz eines Typs. Das bedeutet, dass keine Instanz eines Objekt-Typs notwendig ist, um diese Methode auszuführen; es genügt, den Namen des Objekt-Typs voranzustellen *(siehe Listing 8)*.

Ein weiterer wichtiger Methoden-Typ, der bereits erwähnt wurde, sind die Konstruktoren. Eine Konstruktor-Funktion ist durch das Stichwort „CONSTRUCTOR" gekennzeichnet und hat als Rückgabewert immer „SELF AS RESULT".

Das Beispiel in *Listing 5* hat zwei Konstruktoren. Die erste Konstruktor-Methode erstellt eine leere Instanz des Typs und initialisiert sie mit einigen Default-Werten. Hierzu wird die vom Super-Typ geerbte Methode „setDefaults" überschrieben. Interessant ist hierbei, dass durch die Verwendung von „(SELF AS t_customer)" das Aufrufen der Methode im Super-Typ möglich ist und der dort implementierte Code wiederverwendet werden kann.

Die zweite Konstruktor-Methode erhält die „customer_id" als Parameter und initialisiert die neue Instanz mit den zur ID gehörenden Kundendaten. *Listing 9* zeigt die Verwendung der Konstruktor-Methoden.

Up- und Down-Casting

Oracle erlaubt ein implizites Up-Casting eines abgeleiteten Typs und bietet den „TREAT"-Operator zum expliziten Down-

```
BEGIN
  dbms_output.put_line(t_customer_b2c.getCustomerType);
END;
```

Listing 8: Aufruf einer statischen Methode

```
DECLARE
  cusB2B t_customer_b2b := NEW t_customer_b2b();
  cusB2C t_customer_b2c := NEW t_customer_b2c();
BEGIN
  dbms_output.put_line('B2B Customer Id: ' || cusB2B.customer_id);
  dbms_output.put_line('B2B Sales Discount: ' || cusB2B.sales_dis-
count);

  dbms_output.put_line('--------------------');

  dbms_output.put_line('B2C Customer Id: ' || cusB2C.customer_id);
  dbms_output.put_line('B2C Credit Limit: ' || cusB2C.credit_limit);
END;

--

B2B Customer Id: 0
B2B Sales Discount: 5
--------------------
B2C Customer Id: 0
B2C Credit Limit: 1000
```

Listing 9: Instanziierung über Konstruktoren

```
DECLARE
  cus t_customer;
  cusB2B t_customer_b2b := NEW t_customer_b2b();
  anotherCusB2B t_customer_b2b;
BEGIN
  /* impliziter Upcast */
  cus := cusB2B;

  /* expliziter Downcast */
  anotherCusB2B := TREAT(cus AS t_customer_b2b);
END;
```

Listing 10: Up- und Down-Casting

```
CREATE TABLE CUSTOMERS OF t_customer;
```

Listing 11: Anlegen einer Objekt-Tabelle

```
MEMBER PROCEDURE persist IS
BEGIN
  IF self.customer_id=0 THEN
    SELECT demo_cust_seq.nextval
      INTO self.customer_id
      FROM dual;
    INSERT INTO customers VALUES (self);
    COMMIT;
  ELSE
    UPDATE customers c SET VALUE(c)=self WHERE VALUE(c).customer_
id=self.customer_id;
    COMMIT;
  END IF;

  RETURN;
END;
```

Listing 12: Methode zum Speichern einer Instanz in der Objekt-Tabelle

Casting eines Super-Typs. „TREAT“ ist zusätzlich dazu in der Lage, auch ein explizites Up-Casting eines Sub-Typs vorzunehmen. *Listing 10* zeigt beide Wege anhand eines Beispiels.

Objekt-Tabellen

Bislang haben wir Objekt-Typen als losgelöste Daten-Typen betrachtet oder sie als Zugriffsschicht für normale relationale Tabellen genutzt. Es besteht jedoch auch die Möglichkeit, Objekt-Tabellen anzulegen. Das heißt, jede Zeile der Tabelle besteht genau aus einer Instanz eines Objekt-Typs. Dabei kann es sich durchaus um verschiedene Sub-Typen einer Hierarchie handeln. Interessant ist dieser Ansatz beispielsweise für Log-Mechanismen. Auf diese Art lassen sich Fehler, Warnungen oder Debug-Meldungen genau mit den Informationen speichern, die wirklich erforderlich ist.

Listing 11 zeigt das „CREATE“-Statement zur Erstellung einer Tabelle für die Speicherung von Kunden. Als Objekt-Typ ist hier der abstrakte Typ „t_customer“ definiert. Das führt dazu, dass wir sowohl Instanzen vom Typ „t_customer_b2b“ als auch vom Typ „t_customer_b2c“ speichern können.

Listing 12 zeigt beispielhaft eine Methode zum Speichern der Instanz in der Objekt-Tabelle, die in *Listing 13* entsprechend angewendet wird, um unsere gerade erzeugte Tabelle mit Datensätzen zu füllen.

Wenn wir uns nun ansehen, wie die Daten in der Tabelle aussehen, stellen wir fest, dass wir nicht sofort alle Informationen zum Kunden sehen können *(siehe Abbildung 1)*. Das liegt daran, dass die Tabelle als Liste von „t_customer“ definiert ist. Auch hier hilft uns wieder der „TREAT“-Operator weiter.

Abbildung 2 zeigt eine erweiterte Abfrage, die ein explizites Down-Casting des Typs durchführt. Außerdem zeigt diese Beispiel-Abfrage, wie ein gezieltes Filtern auf bestimmte Objekt-Typen möglich ist. Hierzu wird das Stichwort „IS OF ()“ genutzt. In Klammern können ein oder mehrere Namen von Objekt-Typen angegeben sein.

Zu erwähnen sei hier auch noch, dass das Down-Casting mit „TREAT“ als Ergebnis „NULL“ liefert, sofern der Objekt-Typ

nicht übereinstimmt. Zu erkennen ist dies gut an der Tatsache, dass die ersten Spalten des zweiten Datensatzes leer sind. Natürlich lassen sich auch bei der Verwendung von „TREAT“ einzelne Attribute auswählen. Hierzu wird wieder die Punkt-Notation angewendet *(siehe Abbildung 3)*.

Type-Evolution

Nehmen wir einmal an, unsere Objekt-Hierarchie ist im Einsatz und wir haben bereits diverse Kunden gespeichert, stellen nun aber fest, dass wir weitere Attribute oder auch zusätzliche Methoden einfü-

```
DECLARE
  b2b t_customer_b2b := NEW t_customer_b2b();
  b2c t_customer_b2c := NEW t_customer_b2c();
BEGIN
  b2b.company_name          := 'buw Management Holding';
  b2b.cust_street_address   := 'Rheiner Landstraße 125';
  b2b.cust_city             := 'Osnabrück';
  b2b.cust_postal_code      := '49078';
  b2b.persist();

  b2c.cust_first_name       := 'Anja';
  b2c.cust_last_name        := 'Hildebrandt';
  b2c.cust_street_address   := 'Musterstraße 1a';
  b2c.cust_city             := 'Musterstadt';
  b2c.cust_postal_code      := '12345';
  b2c.persist();
END;
```

Listing 13: Anlegen von Kunden in der Objekt-Tabelle

SQL | Output | Statistics

```
select * from customers;
```

	CUSTOMER_ID	CUST_STREET_ADDRESS	CUST_CITY	CUST_POSTAL_CODE	CUST_EMAIL	PHONE_NUMBER
1	108	Rheiner Landstraße 125	Osnabrück	49078		

Abbildung 1: Selektion von Kunden aus der Objekt-Tabelle

SQL | Output | Statistics

```
select TREAT(VALUE(C) AS T_CUSTOMER_B2B) as b2b
     , TREAT(VALUE(C) AS T_CUSTOMER_B2C) as b2c
 from customers c
where VALUE(C) IS OF (T_CUSTOMER_B2B, T_CUSTOMER_B2C);
```

	B2B.CUSTOMER_ID	B2B.CUST_STREET_ADDRESS	B2B.CUST_CITY	B2B.CUST_POSTAL_CODE	B2B.CUST_EMAIL	B2B.PHONE_NUMBER
1	108	Rheiner Landstraße 125	Osnabrück	49078		

Abbildung 2: Selektion von Kunden aus der Objekt-Tabelle mit „TREAT“

SQL | Output | Statistics

```
select c.*
     , TREAT(VALUE(C) AS T_CUSTOMER_B2B).COMPANY_NAME as COMPANY_NAME
     , TREAT(VALUE(C) AS T_CUSTOMER_B2B).SALES_DISCOUNT as SALES_DISCOUNT
     , TREAT(VALUE(C) AS T_CUSTOMER_B2C).CUST_FIRST_NAME as CUST_FIRST_NAME
     , TREAT(VALUE(C) AS T_CUSTOMER_B2C).CUST_LAST_NAME as CUST_LAST_NAME
     , TREAT(VALUE(C) AS T_CUSTOMER_B2C).CREDIT_LIMIT as CREDIT_LIMIT
  from customers c
 where VALUE(C) IS OF (T_CUSTOMER_B2B,T_CUSTOMER_B2C);
```

	CUSTOMER_ID	CUST_STREET_ADDRESS	CUST_CITY	CUST_POSTAL_CODE	CUST_EMAIL	PHONE_NUMBER	COMPANY_NAME	SALES_DISCOUNT
1	108	Rheiner Landstraße 125	Osnabrück	49078			buw Management Holding	5
2	109	Musterstraße 1a	Musterstadt	12345				

Abbildung 3: Selektion einzelner Attribute aus der Objekt-Tabelle

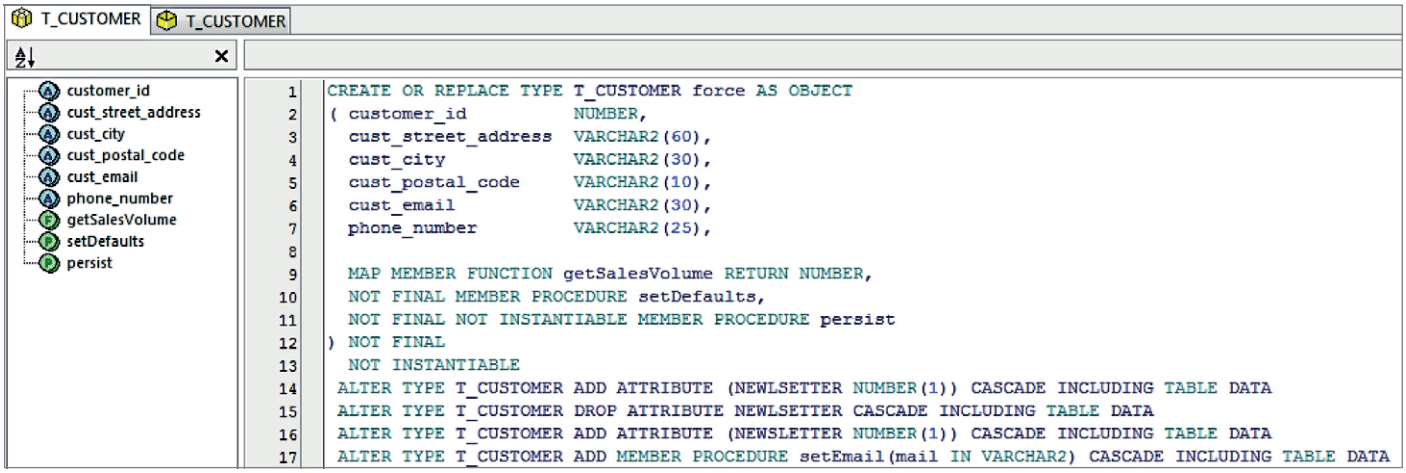


Abbildung 4: Durch „ALTER" geänderte Type-Spezifikation

```
ALTER TYPE t_customer ADD ATTRIBUTE (NEWLSETTER NUMBER(1)) CASCADE INCLUDING TABLE DATA;
ALTER TYPE t_customer DROP ATTRIBUTE NEWLSETTER CASCADE INCLUDING TABLE DATA;
ALTER TYPE t_customer ADD ATTRIBUTE (NEWSLETTER NUMBER(1)) CASCADE INCLUDING TABLE DATA;
ALTER TYPE t_customer ADD MEMBER PROCEDURE setEmail(mail IN VARCHAR2) CASCADE INCLUDING TABLE DATA;
```

Listing 14: Änderungen an Objekt-Typen

gen müssen. Beim Umgang mit Packages sind wir es gewöhnt, unsere Änderungen durchzuführen und dann einfach zu kompilieren. Sofern wir keinen Fehler eingebaut haben, sollte das Package immer noch funktionieren.

Obwohl Packages und Objekt-Typen im Aufbau sehr ähnlich sind, wird uns die Anpassung von Objekt-Typen jedoch nicht ganz so leicht gemacht. Änderungen an Typen müssen über das „ALTER TYPE"-Kommando erfolgen. Der Body kann jedoch jederzeit einzeln kompiliert werden, der „ALTER"-Befehl ist also nur für die Änderung der Spezifikation nötig.

Listing 14 zeigt die „ALTER TYPE"-Statements für verschiedene Änderungen an unserem Beispiel-Typ „t_customer". Hier ist simuliert, dass ein neues Attribut „NEWSLETTER" hinzugefügt wird, das allerdings aufgrund eines Schreibfehlers noch einmal gelöscht und mit korrektem Namen neu angelegt wird. Außerdem wird die Spezifikation für eine neue Methode „setEmail" hinzugefügt.

Das Stichwort „CASCADE" propagiert die durchgeführten Änderungen an alle von diesem Typ abhängigen sonstigen Typen und Tabellen. Im Beispiel wurde die Option „INCLUDING TABLE DATA" benutzt, um die in der Tabelle gespeicherten Kunden direkt in das neue Format umzuwandeln.

Ein Blick in die Spezifikation zeigt, dass die „ALTER"-Befehle einfach an die Definition des Objekt-Typs angehängt werden *(siehe Abbildung 4)*. Das kann schnell unübersichtlich werden; man sollte sich daher gut überlegen, ob die Verwendung von Objekt-Typen in Projekten mit häufigen Änderungen eine gute Idee ist.

Fazit

Leider fehlen den Objekt-Typen einige Key-Features der objektorientierten Entwicklung. So ist es zum Beispiel nicht möglich, Attribute als privat zu deklarieren, sodass sie nur über entsprechende „get"- beziehungsweise „set"-Methoden manipuliert werden können. Auch die Anpassung der Spezifikation durch „ALTER"-Befehle ist gewöhnungsbedürftig. Die Autorin sieht jedoch gerade im Zusammenhang mit Wiederverwendbarkeit und Lesbarkeit des Quellcodes große Vorteile.

Im Rahmen dieses Artikels wurde nur auf einen kleinen Teil der Möglichkeiten eingegangen, die Objekt-Typen in Oracle bieten. Komplexere Themen wie die Nutzung von Collections wurden bewusst ausgelassen. Es lohnt sich also in jedem Fall, die Dokumentation zu diesem Thema einmal genauer zu lesen (siehe „Database Object-Relational Developer's Guide").

Anja Hildebrandt
anja.hildebrandt@buw.de

Continuous Integration in Apex-Projekten

Sven Böttcher, Apps Associates GmbH

Dieser Artikel zeigt, wie sich Continuous Integration in PL/SQL- und Apex-Projekten anwenden lässt. Dabei wird auf den Einsatz von Jenkins als Continuous-Integration-Server, Maven als Build-Tool für PL/SQL und Apex sowie auf das Testen mit RSpec und dem Selenium WebDriver eingegangen.

Continuous Integration (CI) ist ein etablierter Prozess, der in der Software-Entwicklung zur Verbesserung der Software-Qualität eingesetzt wird. Der Prozess besteht aus den grundlegenden und inhärenten Teilschritten der Versionierung des Quellcodes in einem Versions-Kontrollsystem („commit"), dem ständigen Zusammenfügen der einzelnen Software-Komponenten („build") sowie dem Ausführen verschiedenartiger Software-Tests („test"). Dieser Prozess lässt sich dabei durch einen CI-Server wie zum Beispiel Jenkins weitestgehend automatisieren. Zudem gibt der CI-Server den Projektverantwortlichen und Entwicklern Rückmeldung über den aktuellen Zustand des Software-Projekts, sodass auf nicht erfolgreiche Builds oder Tests schnellstmöglich reagiert werden kann.

Während Continuous Integration beispielsweise in der Java-Entwicklung schon seit Langem angewendet wird, ist dieser Begriff in PL/SQL- beziehungsweise Apex-Projekten relativ unbekannt und findet entsprechend kaum Anwendung. Dies liegt unter anderem daran, dass sich die Apex-Entwicklung grundlegend von der Programmierung in einer konventionellen Programmiersprache unterscheidet und nicht direkt ersichtlich ist, wie sich der Continuous-Integration-Prozess in Apex-Projekten einsetzen lässt, und dass der Continuous-Integration-Prozess als sehr aufwändig gilt.

Klassische Continuous-Integration-Architektur

Abbildung 1 zeigt die klassischen Continuous-Integration-Komponenten, wie sie in vielen Software-Projekten eingesetzt sind. Sie bestehen aus:

- Der Entwicklungsumgebung mit einem Versionierungstool
- Einem Versions-Kontrollsystem
- Einem CI-Server mit diversen Plug-ins, etwa zum automatisierten Bauen und Testen der Software

Der CI-Prozess ist in *Abbildung 2* dargestellt. Alle am Projekt beteiligten Entwickler aktualisieren in definierten und nicht zu langen Intervallen den von ihnen entwickelten Quellcode im Versions-Kontrollsystem. Der CI-Server prüft dieses Versions-Kontrollsystem in ebenso definierten (aber gegebenenfalls anderen) Intervallen auf Änderungen („polling") und übernimmt diese in ein lokales Repository.

Nach jeder Änderung in der Codebasis oder in Abhängigkeit von Zeitplänen wird das Software-Projekt kompiliert beziehungsweise gebaut. Nachdem dieser Schritt (erfolgreich) durchgeführt wurde, können automatisierte Testläufe durch den CI-Server angestoßen werden.

Um den Projektleitern und Entwicklern möglichst schnell ein Feedback über den aktuellen Stand des Projekts hinsichtlich von Programmierfehlern zu geben, verfügen CI-Server normalerweise über einen Feedback-Mechanismus in Form von HTML-Seiten, auf denen aktuelle und historische Fehler aufgelistet sind.

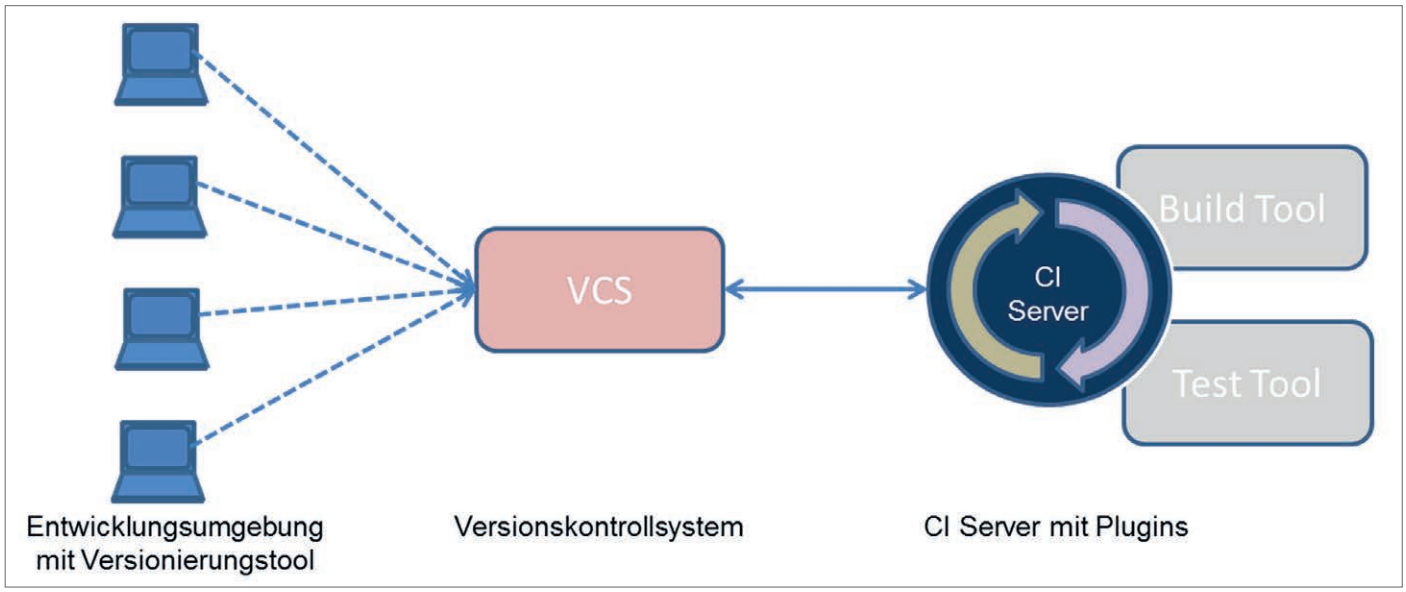


Abbildung 1: Die Continuous-Integration-Komponenten

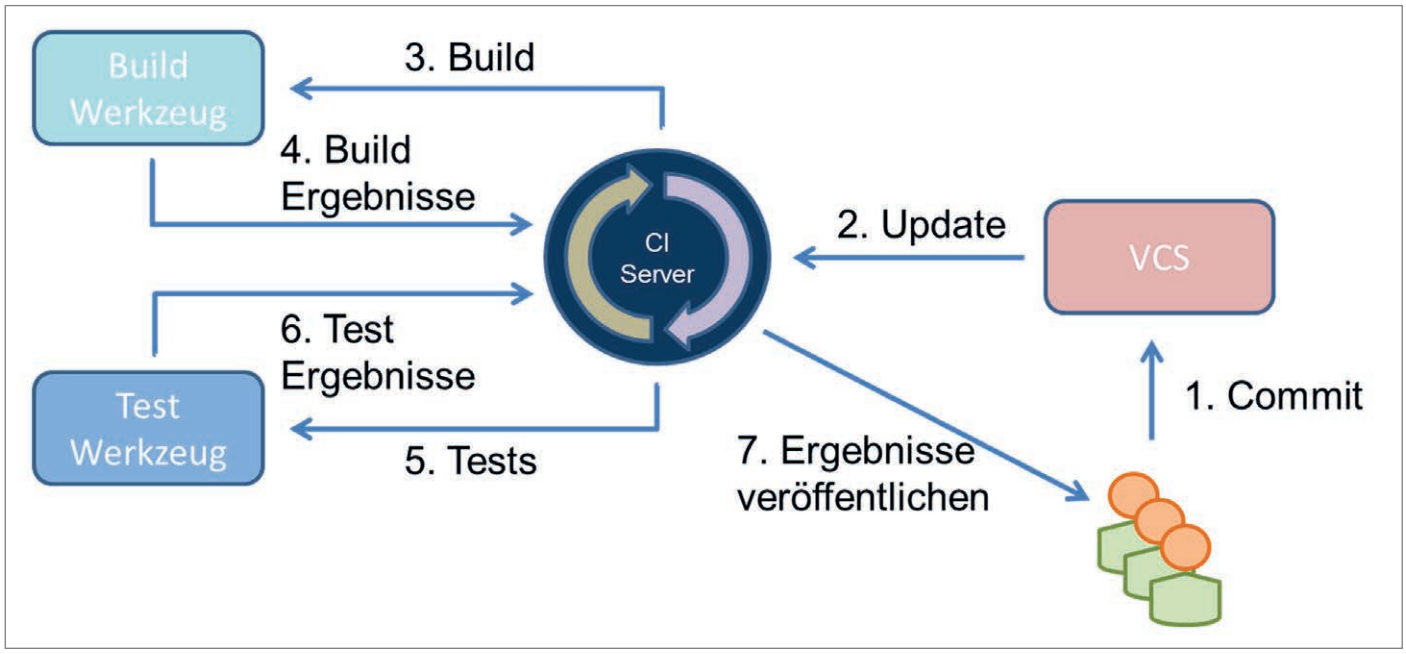


Abbildung 2: Der Continuous-Integration-Prozess

Durch diesen Prozess entsteht der Vorteil, dass die gesamte Code-Basis von allen beteiligten Entwicklern ständig integriert wird. So lassen sich insbesondere Software-Tests durchführen, die das Zusammenspiel von voneinander abhängigen Software-Modulen testen, wodurch entsprechende Probleme bereits sehr früh erkannt und beseitigt werden können. Treten solche Probleme erst in einem späten Entwicklungsstadium auf, kann eine Beseitigung sehr zeitaufwändig beziehungsweise kostenintensiv sein oder sogar zum Scheitern des Projekts führen.

Continuous-Integration-Architektur in Apex-Projekten

In konventionellen Java-Entwicklungsprojekten verfügt in der Regel jeder Entwickler über eine lokale Arbeitskopie des Quellcodes und die Entwicklung erfolgt entsprechend lokal. Ist zum Beispiel Subversion (SVN) als Versions-Kontrollsystem eingesetzt, werden alle Änderungen in ein zentrales Repository übertragen. Erfahrungsgemäß entwickeln in PL/SQL- beziehungsweise Apex-Projekten zumeist alle Entwickler in einer zentralen Datenbank oder in einer zentralen Apex-Instanz *(siehe Abbildung 3)*.

Dadurch ergibt sich insbesondere das Problem, dass es bei nicht ausreichender Abstimmung zum Überschreiben von Quellcode oder Apex-Entwicklungen

kommen kann. Dies ist insbesondere dann problematisch, wenn keine Versionierung eingesetzt ist oder Änderungen vor dem Übertragen in das Versions-Kontrollsystem überschrieben werden. *Abbildung 4* zeigt einen Ansatz, dieses Problem zu umgehen und einen Continuous-Integration-Prozess, ähnlich wie in Java-Projekten, zu ermöglichen.

In diesem Szenario verfügt jeder Entwickler über eine lokale Datenbank und eine lokale Apex-Instanz. Die benötigte Infrastruktur kann beispielsweise einfach in einer virtuellen Box installiert und an alle Entwickler verteilt sein. Die eigentliche Entwicklung erfolgt nun lokal und nicht mehr in einer zentralen Datenbank. Da Versions-Kontrollsysteme im Allgemeinen dateibasiert sind, müssen der PL/SQL-Code sowie alle DDL/DCL-Befehle vor dem Übertragen in das Versions-Kontrollsystem in einer Datei gespeichert sein. Der CI-Server prüft in definierten Intervallen das Versions-Kontrollsystem und aktualisiert seine eigene Codebasis. Bei Änderungen in der SQL-, PL/SQL- oder Apex-Codebasis werden diese in der CI-Datenbank nachgezogen und automatisierte Tests durchgeführt.

Als CI-Server ist Jenkins *(siehe „https://jenkins.io“)* in Apex-Projekten sehr gut geeignet. Da Jenkins unter der MIT-Lizenz steht, fallen für die Verwendung keine Lizenzkosten an. Zudem lässt sich Jenkins durch zahlreiche Plug-ins sehr gut an die projektspezifischen Anforderungen anpassen. Für Apex-Projekte bedeutet dies, dass ein Build-Werkzeug zum Erzeugen der Datenbank-Objekte und der Apex-Anwendung sowie ein Test-Werkzeug zum Testen des PL/SQL-Codes und der Apex-Anwendung erforderlich sind.

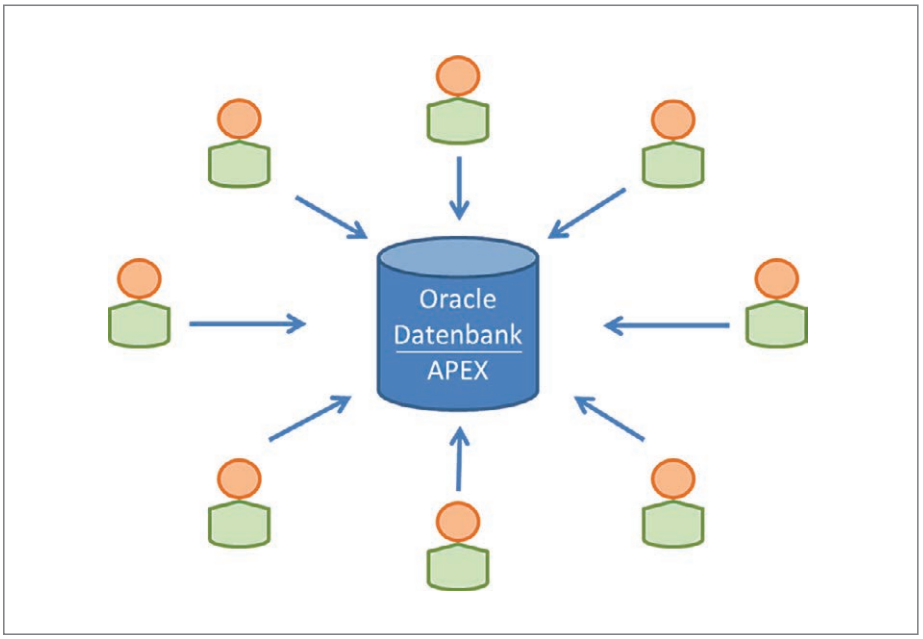


Abbildung 3: Typische Apex-Entwicklung im Team

Maven als Build-Werkzeug

Maven *(siehe „https://maven.apache.org“)* lässt sich sehr einfach in Jenkins integrieren. Dafür muss lediglich über die Jenkins-Plug-in-Verwaltung das Maven-Integration-Plug-in installiert werden. Anschließend lässt sich in Jenkins leicht ein neues Maven-Projekt anlegen *(siehe Abbildung 5)*.

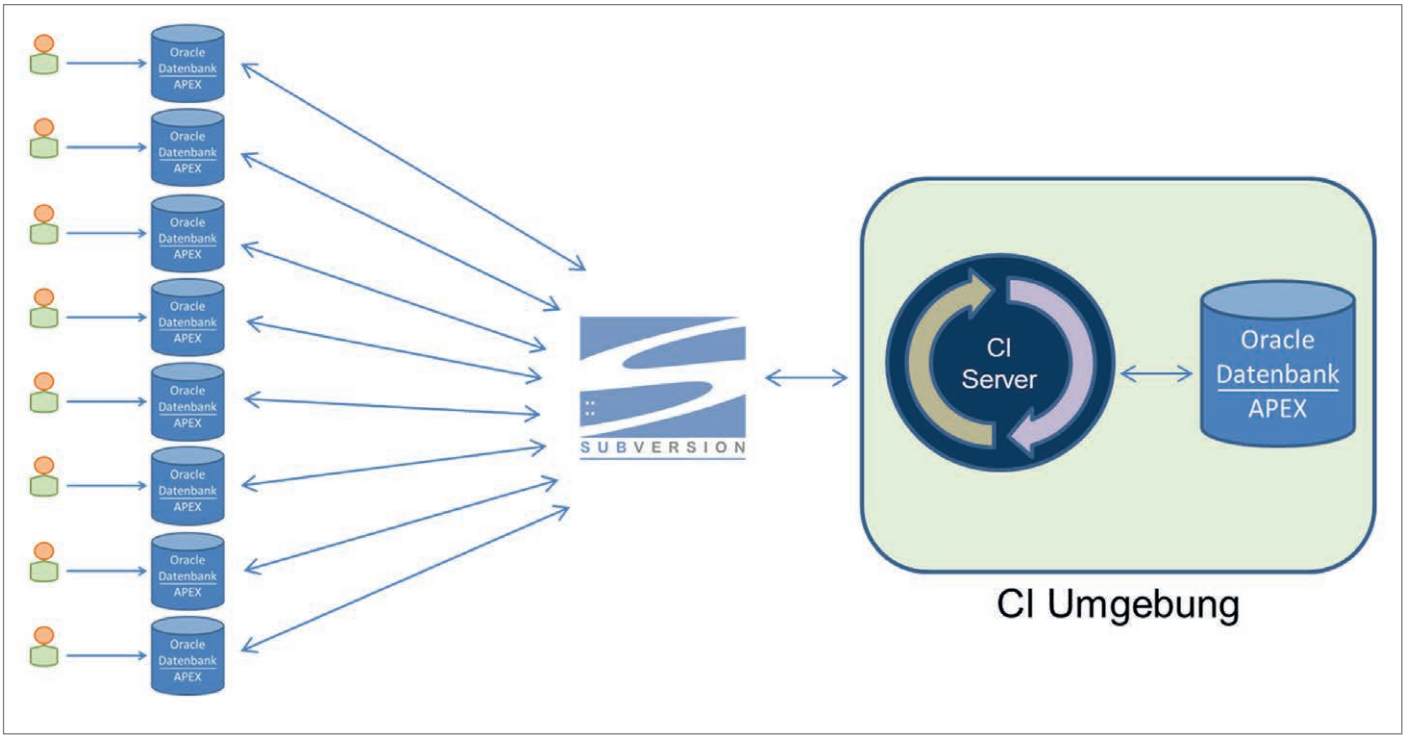


Abbildung 4: Continuous Integration in Apex-Projekten

Der eigentliche Maven-Build-Prozess wird typischerweise in einer „pom.xml“-Konfigurationsdatei beschrieben. Für den Build-Prozess von SQL- oder PL/SQL-Code bietet sich das SQL-Maven-Plug-in an. Damit lassen sich sehr einfach in Quelldateien gespeicherte SQL- beziehungsweise PL/SQL-Befehle ausführen. *Listing 1* zeigt eine beispielhafte Konfiguration dieses Plug-ins.

Da die Verbindung zur Datenbank über eine „jdbc“-Verbindung erfolgt, muss ein entsprechender Treiber vorhanden sein. Die Verbindungs-Informationen sind über einen einfachen Verbindungsstring im XML-Element „URL“ angegeben. Der Konfigurationsteil „configuration“ im „executions“-Bereich listet die Quelldateien auf, die innerhalb des Build- Prozesses ausgeführt werden sollen. Ist der Build-Prozess erfolgreich, ist dies innerhalb des Jenkins-Build-Verlaufs durch einen blauen Punkt ersichtlich. Ist der Build-Prozess fehlerhaft, wird dies durch einen roten Punkt kenntlich gemacht. Zudem gibt Jenkins eine detaillierte Auskunft über den aufgetretenen Fehler. Diese Integration wird durch das zuvor erwähnte SQL-Maven-Plug-in ermöglicht. Auch wenn diese Art der Code-Ausführung sehr komfortabel ist, ergeben sich die folgenden Probleme:

- Wenn existierende Datenbank-Objekte wie Tabellen bei jedem Build erneut erzeugt werden, kommt es zu einem Fehler, wenn diese Objekte bereits bestehen.
- Fehlerhafte Prozeduren, Funktionen oder Packages werden von der Datenbank trotz Fehler kompiliert (auch wenn sie fehlerhaft sind) und Jenkins erkennt den Build aufgrund dessen als erfolgreich. Abbildung 6 zeigt ein eigentlich fehlerhaftes Package („benutzerverwaltung“), das von Jenkins aber als fehlerfrei dargestellt wird.

Das erste Problem würde sich relativ einfach lösen lassen, wenn vor jedem Build alle Datenbank-Objekte gelöscht werden. Auch wenn dies in CI-Umgebungen vertretbar ist, könnten die gleichen Build-Skripte nicht auf anderen Instanzen (Test, QA etc.) verwendet werden. Daher sollten alle DML-Operationen durch ein „EXECUTE IMMEDIATE“, wie in *Listing 2* dargestellt, erstellt werden.

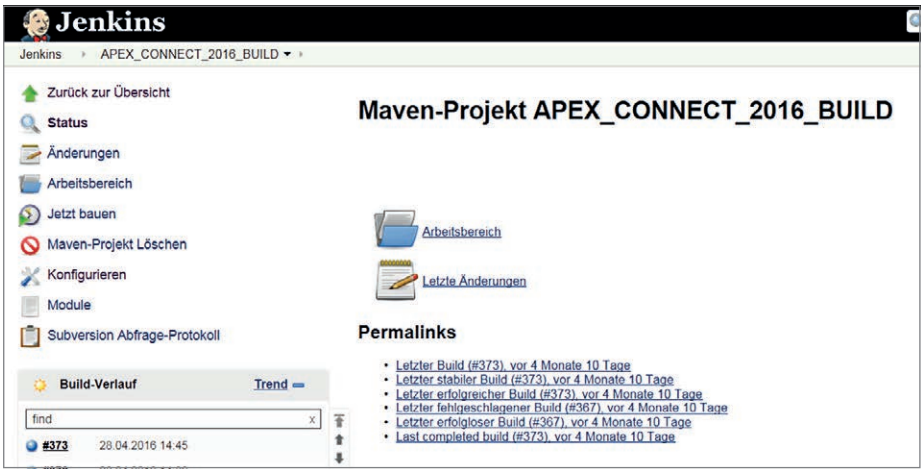


Abbildung 5: Maven-Projekt in Jenkins

```
<plugin>
 <groupId>org.codehaus.mojo</groupId>
 <artifactId>sql-maven-plugin</artifactId>
 <version>1.4</version>

 <dependencies>
  <dependency>
   <groupId>com.oracle.jdbc</groupId>
   <artifactId>ojdbc7</artifactId>
   <version>12.1.0.1</version>
  </dependency>
 </dependencies>

 <configuration>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
  <url>jdbc:oracle:thin:@192.168.56.102:1521:workshop</url>
  <username>ci_test</username>
  <password>ci_test</password>
  <skip>false</skip>
 </configuration>

 <executions>
  <execution>
   <id>test</id>
   <phase>process-resources</phase>
   <goals>
    <goal>execute</goal>
   </goals>
   <configuration>
    <autocommit>true</autocommit>
    <delimiter>/</delimiter>
    <delimiterType>row</delimiterType>
    <onError>abort</onError>
    <srcFiles>
     <srcFile>ci_test/trunk/DDL/t_benutzer.sql</srcFile>
     <srcFile>ci_test/trunk/DDL/t_benutzer_pk.sql</srcFile>
     <srcFile>ci_test/trunk/DDL/t_benutzer_uk1.sql</srcFile>
     <srcFile>ci_test/trunk/DDL/seq_t_benutzer.sql</srcFile>
     <srcFile>ci_test/trunk/PLSQL/benutzerverwaltung.pks.sql</srcFile>
     <srcFile>ci_test/trunk/PLSQL/benutzerverwaltung.pkb.sql</srcFile>
     </srcFiles>
    </configuration>
   </execution>
  </executions>
</plugin>
```

Listing 1

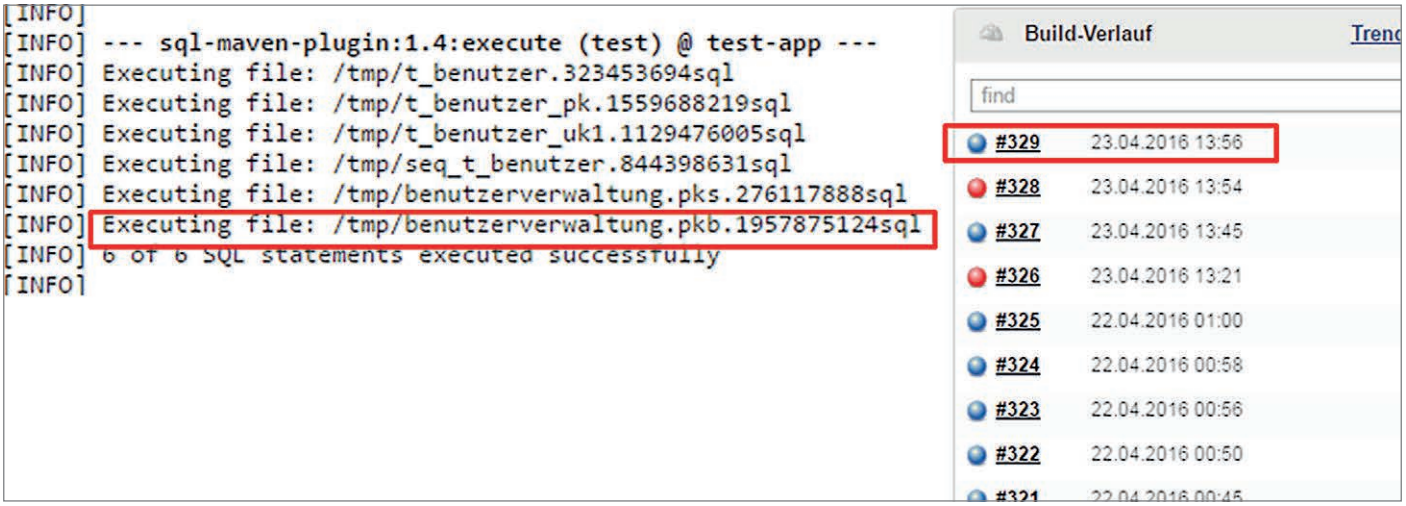


Abbildung 6: Fehlerhafte Packages werden durch den Build mit dem SQL-Maven-Plug-in in Jenkins als korrekt erkannt

Durch dieses Vorgehen fängt der Exception-Block den Fehler ab, der geworfen wird, falls ein Objekt bereits existiert („ORA-00955"). Das zweite Problem lässt sich mit dem SQL-Maven-Plug-in nicht lösen. Um auch fehlerhafte Prozeduren, Funktionen, und Packages in Jenkins als fehlerhaft darzustellen, kann das Exec-Maven-Plug-in in Kombination mit SQL*PLUS verwendet werden. Damit lassen sich Programme auf Betriebssystem-Ebene aufrufen. *Listing 3* zeigt eine beispielhafte Maven-Konfiguration. Das in *Listing 4* dargestellte SQL-Skript „install_database_objects.sql" dient dabei als aufrufendes Skript für alle weiteren SQL-Skripte und wird mit SQL*PLUS ausgeführt.

In diesem Skript wird durch die dargestellte anonyme Funktion im Fehlerfall ein Anwendungsfehler („raise_application_error") geworfen. Der eigentliche Trick besteht aber in dem SQL*PLUS-Kommando „WHENEVER SQLERROR EXIT SQL.SQLCODE". Damit endet SQL*PLUS mit einem Fehler, was von Jenkins auch als Fehler erkannt wird. *Abbildung 7* zeigt das nun als fehlerhaft erkannte Paket „benutzerverwaltung".

Neben den Datenbank-Objekten lassen sich auch ganze Apex-Anwendungen, einzelne Apex-Seiten etc. als SQL-Datei exportieren und in das Versions-Kontrollsystem übertragen. Die Installation der Apex-Objekte kann, wie für Datenbank-

```
BEGIN

 EXECUTE IMMEDIATE
  'CREATE TABLE "T_BENUTZER"
   ("BENUTZER_ID" NUMBER,
    "VORNAME" VARCHAR2(200 BYTE),
    "NACHNAME" VARCHAR2(600 BYTE)
   )';

EXCEPTION
 WHEN OTHERS THEN
  IF SQLCODE = -955 THEN
   NULL;
  ELSE
   RAISE;
  END IF;
END;
```

Listing 2

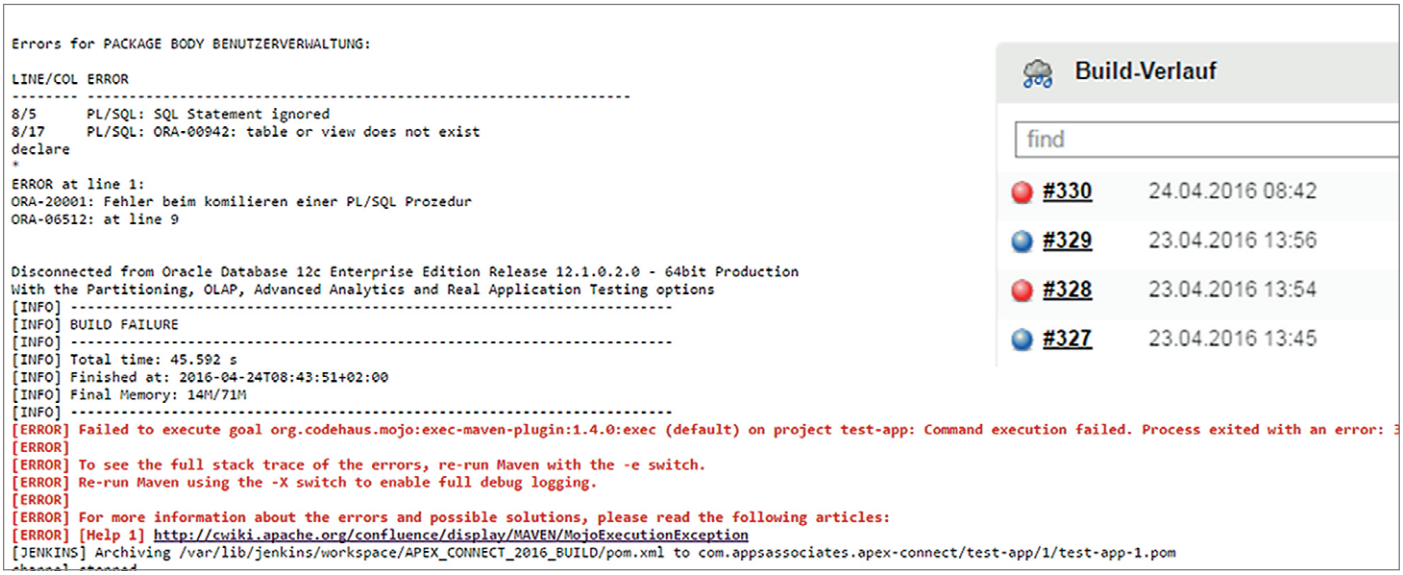


*Abbildung 7: Jenkins erkennt durch den SQL*PLUS-basierten Build fehlerhafte Packages auch als fehlerhaft*

Objekte, durch SQL*PLUS in der CI-Umgebung erfolgen. *Listing 5* zeigt ein Beispiel dafür.

In der anonymen Funktion werden zunächst die Workspace-ID, die Applikations-ID und das Schema der zu installierenden Anwendung durch das Package „apex_application_install" gesetzt. Anschließend wird mit „generate_offset" ein Offset für die internen IDs aller Apex-Objekte erzeugt. Anschließend lassen sich die einzelnen Apex-Objekte durch einen einfachen Aufruf der entsprechenden SQL-Skripte einfach installieren.

RSpec als Testwerkzeug

Als Testwerkzeug für Apex-Projekte eignet sich RSpec *(siehe „http://rspec.info")*, ein Behavior-Driven-Development-Framework (BDD) für Ruby *(siehe „https://www.ruby-lang.org/de")*. Damit lassen sich erwartete Verhaltensweisen der zu testenden Software auf Klassen-, Methoden- oder Modul-Ebene in „describe"-Blöcken beschreiben. Diese können weiter in „context"-Blöcke strukturiert werden, die, wie der Name schon sagt, einen bestimmten Kontext, in dem der Test ausgeführt wird, beschreiben. So kann es beispielsweise einen „describe"-Block für eine Methode geben. In diesem Block lassen sich nun durch „context"-Blöcke, etwa durch Variablen-Belegungen, verschiedene Ausführungsbedingungen erzeugen. Die eigentlichen Tests werden in „it"-Blöcken beschrieben. Jeder „it"-Block stellt damit einen konkreten Testfall dar.

Mit dem Ruby-PL/SQL-Plug-in steht zudem ein API innerhalb von Ruby für SQL oder PL/SQL zur Verfügung. Damit lassen sich sehr einfach aus Ruby heraus SQL-beziehungsweise PL/SQL-Befehle wie Funktions-Aufrufe ausführen. Insbesondere ist Ruby-PL/SQL innerhalb von RSpec einsetzbar. Die Installation kann einfach mit RubyGems, dem Ruby-Paketsystem, erfolgen. Beides lässt sich mit dem Paket „ruby-plsql-spec" installieren. Eine Ruby-Installation wird an dieser Stelle vorausgesetzt. *Listing 6* zeigt ein Beispiel für den Test einer PL/SQL-Funktion.

In dem Beispiel wird eine Funktion zum Einfügen eines Benutzers getestet. Die Funktion liefert eine „1", falls der Benutzer erfolgreich in der Datenbank gespeichert wurde, und eine „0" im Fehler-

```
<plugin>
 <groupId>org.codehaus.mojo</groupId>
 <artifactId>exec-maven-plugin</artifactId>
 <version>1.4.0</version>
 <executions>
  <execution>
   <phase>process-resources</phase>
   <goals>
    <goal>exec</goal>
   </goals>
  </execution>
 </executions>
 <configuration>
  <executable>sqlplus64</executable>
  <environmentVariables>
   <LD_LIBRARY_PATH>/usr/lib/oracle/12.1/client64/lib/</LD_LIBRARY_
PATH>
  </environmentVariables>
  <arguments>
   <argument>ci_test/ci_test@192.168.56.102:1521/workshop</argument>
  <argument>@install_database_objects/install_database_objects.sql</
argument>
  </arguments>
 </configuration>
</plugin>
```

Listing 3

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
set define off
/
@install_database_objects/install_packages.sql
@install_database_objects/install_apex.sql
show errors
declare
 l_count_errors number;
begin
 select count(*)
  into  l_count_errors
  from  all_errors;
 if l_count_errors > 0 then
  raise_application_error(-20001,'Fehler beim komilieren einer PL/SQL
Prozedur');
 end if;
end;
/
exit 0;
```

Listing 4

```
begin
 apex_application_install.set_workspace_id(5512166499766120);
 apex_application_install.set_application_id(102);
 apex_application_install.set_schema('CI_TEST');
 apex_application_install.generate_offset;
end;
/
@ci_test/trunk/Apex/CI_TEST.sql
@ci_test/trunk/Apex/PAGE_1.sql
@ci_test/trunk/Apex/PAGE_2.sql
@ci_test/trunk/Apex/PAGE_3.sql
@ci_test/trunk/Apex/PAGE_101.sql
```

Listing 5

```
require_relative "../spec_helper.rb"

describe "erstelle_benutzer" do

 it "soll Benutzer einfuegen" do
  expect(plsql.benutzerverwaltung.erstelle_benutzer("Sven","Boettcher")).to equal(1)
  plsql.t_benutzer.delete(:vorname => 'Sven' , :nachname => 'Boettcher')
  plsql.commit
 end

 it "soll Benutzer nicht einfuegen" do
  benutzer = {:benutzer_id => plsql.seq_t_benutzer.nextval, :vorname => 'Sven', :nachname => 'Boettcher'}
  plsql.t_benutzer.insert benutzer
  expect(plsql.benutzerverwaltung.erstelle_benutzer("Sven","Boettcher")).to equal(0)
  plsql.t_benutzer.delete(:vorname => 'Sven' , :nachname => 'Boettcher')
  plsql.commit
 end
end
```

Listing 6

fall zurück. Ein Benutzer soll eindeutig durch seinen Vor- und Nachnamen beschrieben sein. Das erfolgreiche Einfügen eines Benutzers wird in dem „it"-Block „soll Benutzer einfuegen" getestet. Das erwartete Ergebnis ist durch den „expect"-Aufruf beschrieben. Dieser nimmt als Parameter die Rückgabe der über „ruby-plsql" aufgerufenen PL/SQL-Funktion „benutzerverwaltung.erstelle_benutzer" entgegen.

Nur wenn der von der Funktion zurückgegebene Wert gleich „1" ist, ist der Test erfolgreich. Nach dem Test wird der angelegte Benutzer aus der Datenbank entfernt. Der Test „soll Benutzer nicht einfuegen" testet den Fehlerfall. Dafür wird zunächst ein Benutzer in der Datenbank angelegt und anschließend innerhalb des „expect"-Aufrufs versucht, diesen erneut anzulegen. In diesem Fall ist der Test nur dann erfolgreich, wenn die aufgerufene PL/SQL-Funktion eine „0" zurückliefert (der Fehler wurde abgefangen) und damit die Rückgabe dem erwarteten Ergebnis entspricht.

```
#!/bin/bash
source $HOME/.rvm/scripts/rvm
ruby --version
cd ci_test/tests/unit_tests
export PLSQL_COVERAGE=coverage
rspec -f RspecJunitFormatter -o test.xml --failure-exit-code 0
```

Listing 7

Die Einbindung von „ruby-plsql-spec" in Jenkins erfolgt über den Bau eines „Free-Style"-Softwareprojekts mit dem Build-Verfahren „Shell ausführen". In diesem Build-Verfahren lassen sich beliebige Shell-Programme, wie in diesem Beispiel RSpec, ausführen. *Listing 7* zeigt, wie sich der Aufruf von RSpec realisieren lässt.

Der Aufruf von „rspec" führt alle Testfälle im Ordner „ci_test/tests/unit_tests/spec" aus. Die Option „-f RspecJunitFormatter" formatiert die Ausgabe beziehungsweise die Test-Ergebnisse, die in der Datei „test.xml" gespeichert werden, im JUnit-Format. Aufgrund dieser Formatierungsmöglichkeit lassen sich die Test-Ergebnisse von RSpec einfach in Jenkins einbinden. *Abbildung 8* zeigt ein Beispiel, bei dem die beiden oben beschriebenen Tests erfolgreich waren.

Für den Test der Apex-Anwendung lässt sich Selenium einsetzen, ein Tool für die Browser-Automatisierung, das zum Testen von Web-Anwendungen verwendet werden kann. Während Selenium eigentlich ein Firefox-Plug-in ist, steht mit

Abbildung 8: Ergebnisse der RSpec-basierten Funktions-Tests in Jenkins

dem Selenium-WebDriver ein API zur Verfügung, für das es das Ruby-Plug-in „selenium-webdriver“ gibt. Dieses lässt sich einfach mit RubyGems installieren und ermöglicht die Browser-Automatisierung via Ruby und RSpec. Das bedeutet, dass Apex-Anwendungen ebenfalls mit RSpec getestet werden können.

Durch das Ruby-PL/SQL-Plug-in lassen sich zum einen automatisiert bestimmte Kontexte in der Datenbank erzeugen, zum anderen kann man automatisch testen, ob in Apex-Masken eingegebene Daten erwartungsgemäß in der Datenbank gespeichert werden.

Die Einbindung der RSpec-Testskripte in Jenkins erfolgt analog zu dem beschriebenen Test einer PL/SQL-Funktion. *Listing 8* zeigt ein Beispiel für einen Test einer einfachen Apex-Maske zum Speichern eines Benutzers in die Datenbank.

Der Test entspricht im Wesentlichen dem beschriebenen Test zum Einfügen eines Benutzers in die Datenbank durch eine PL/SQL-Funktion, nur dass die Funktion nicht direkt, sondern über eine Apex-Maske aufgerufen wird. Zunächst wird ein Treiber für Firefox erzeugt, der Test erfolgt also mittels Firefox. Daneben existieren auch Treiber für Chrome und den Internet Explorer. Anschließend wird zur Apex-Anwendung navigiert und danach erfolgt die Anmeldung. Dann beginnt der eigentliche Test. Nachdem auf die Seite 2 der Apex-Anwendung navigiert wurde, werden die Apex Items „P2_VORNAME“ und „P2_NACHNAME“ mit Werten belegt und dann auf den „Speichern“-Button geklickt. Jetzt erfolgt der eigentliche Test. Hier wird mit „ruby-plsql“ zunächst die Anzahl der Vorkommen des zuvor eingefügten Benutzers ermittelt. Ein Vergleich des Ergebnisses mit dem erwarteten Wert erfolgt, wie bereits oben beschrieben, durch „expect“. Das Ergebnis des Tests lässt sich entsprechend dem Funktionstest in Jenkins veröffentlichen.

```
require "selenium-webdriver"
require_relative "spec_helper.rb"

describe "Benutzerverwaltung" do

 before(:all) do
  @driver = Selenium::WebDriver.for(:firefox)
 end

 before(:each) do
  @url = "http://192.168.56.102:8080/ords/f?p=102"
  @driver.navigate.to(@url)
  @driver.find_element(:id, "P101_USERNAME").clear
  @driver.find_element(:id, "P101_USERNAME").send_keys("admin")
  @driver.find_element(:id, "P101_PASSWORD").send_keys("Oracle1234!")
  sleep 2
  @driver.find_element(:xpath,"//button[@type='button']").click
  @url = @driver.current_url
 end

 after(:all) do
  @driver.quit
 end

 it "soll Benutzer einfuegen" do
  @url = @url.gsub("f?p=102:1","f?p=102:2")
  @driver.navigate.to(@url)
  @driver.find_element(:id,"P2_VORNAME").send_keys("Sven")
  @driver.find_element(:id,"P2_NACHNAME").send_keys("Boettcher")
  sleep 2
  @driver.find_element(:id,"speichern").click
  expect(plsql.t_benutzer.count("WHERE vorname = :vorname and nachname
= :nachname", "Sven",
   "Boettcher")).to eq(1)
  plsql.t_benutzer.delete(:vorname => 'Sven' , :nachname => 'Boettch-
er')
  plsql.commit
 end
end
```

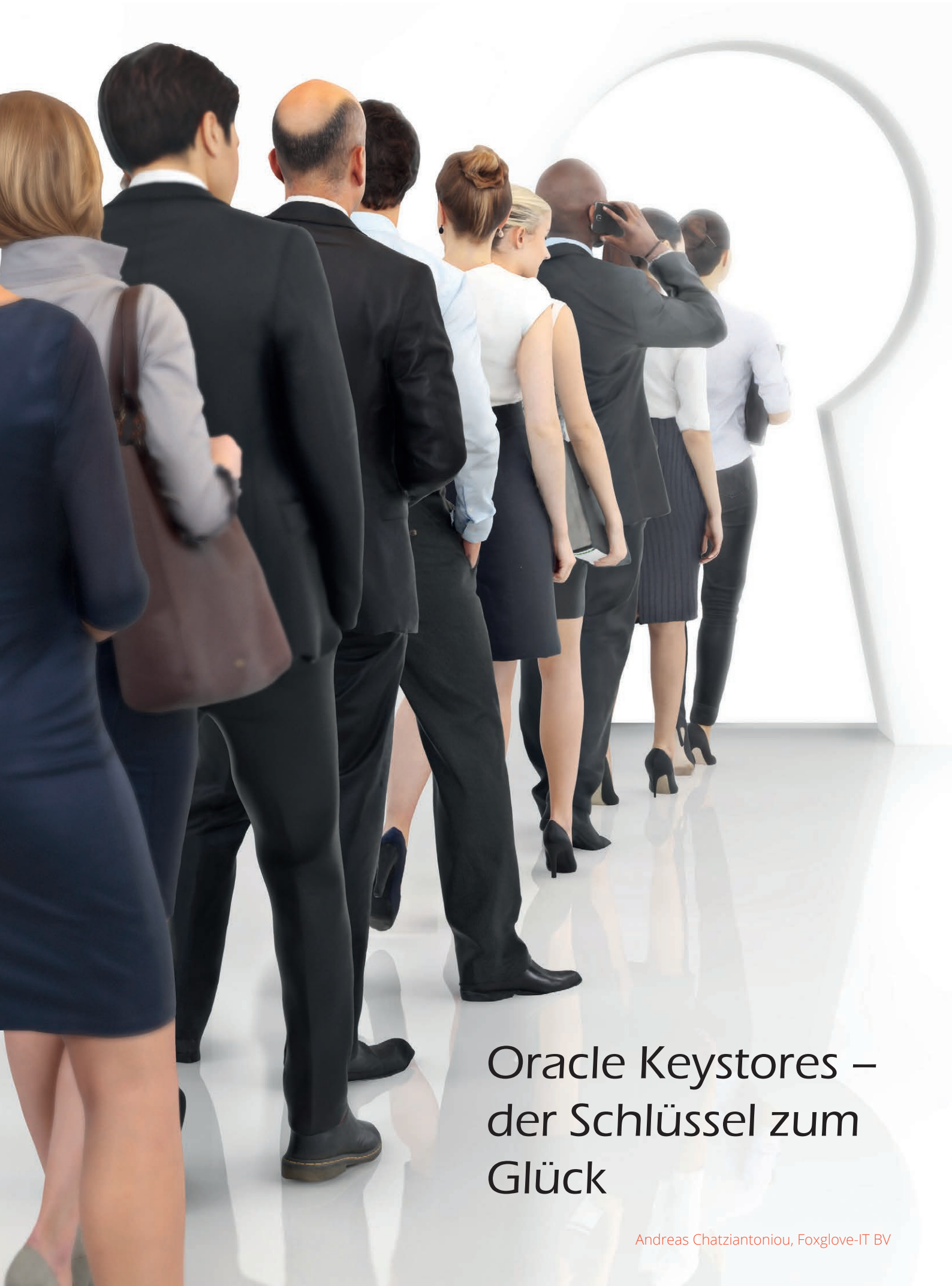
Listing 8

Fazit

Dieser Artikel zeigt, dass Continuous Integration in PL/SQL- und Apex-Projekten möglich ist. Allerdings muss für die Anwendung die Art und Weise der Entwicklung angepasst werden. Hier erzeugt insbesondere die dateibasierte Versionierung des Quellcodes und der Apex-Objekte einen nicht zu unterschätzenden Mehraufwand. Dieser Mehraufwand wird durch das Schreiben der benötigten Testskripte noch erhöht. Demgegenüber steht allerdings, dass die Qualität der zu entwickelnden Software hinsichtlich der Korrektheit und Fehlerfreiheit durch Continuous Integration erhöht werden kann.

Durch den CI-Server erhalten Entwickler und Projektleiter stets einen Überblick über den aktuellen Zustand des Projekts, sodass eine schnelle Reaktion auf auftretende Probleme möglich ist. Auch wenn der Aufwand in kleineren Projekten sehr hoch erscheint, kann dieses Vorgehen insbesondere dann einen Vorteil bringen, wenn in größeren Projekten modulare und voneinander abhängige Apex-Anwendungen parallel entwickelt werden.

Sven Böttcher
sven.boettcher@appsassociates.com



Oracle Keystores – der Schlüssel zum Glück

Andreas Chatziantoniou, Foxglove-IT BV

Seit WebLogic 12 gibt es im Rahmen der Oracle Platform Security Services (OPSS) den Keystore Service (KSS). Dieser Artikel zeigt anhand von Beispielen und einer End-to-End-Einbettung, wie die verschiedenen Fusion-Middleware-Komponenten mit OPSS und dem KSS arbeiten können. Außerdem werden die Unterschiede zum bisherigen Java-Keystore-Format (JKS) aufgezeigt.

„Security" ist seit Beginn der IT ein Thema. Betrachtet man die Komponenten eines IT-Systems, ist der Zugang zu den Daten wahrscheinlich der sensibelste Teil, auf dem Sicherheitsmechanismen greifen müssen. Denn wer die Daten kontrolliert, braucht sich nicht mehr um die verschiedenen Lagen der Zugangskontrolle in den darüberliegenden Schichten zu kümmern. Normalerweise bietet die Oracle-Datenbank genügend Möglichkeiten, um die Daten zu schützen – ob das nun die normalen Rollen und Rechte auf Tabellen sind, Virtual Private Database, Label Security, Transparent Data Encryption etc. Man kann davon ausgehen, dass die Daten nur denjenigen zugänglich sind, die diesen Zugang auch haben.

Eine Ebene höher – auf dem Niveau des Application-Servers – muss ein vergleichbares Konstrukt vorhanden sein, um die Zugangskontrolle zu implementieren. Die Oracle Keystores sind Teil der Oracle Platform Security Services. *Abbildung 1* zeigt deren Position innerhalb des OPSS.

Alle Keys des Oracle Keystore sind im zentralen OPSS Security Store abgelegt. Dieses kann entweder Datei-, LDAP- oder Datenbank-basiert sein – das wird beim Anlegen des OPSS Security Store festgelegt. Das Anlegen kann mithilfe des Repository Creation Utility (RCU), des Enterprise Manager oder per Skript erfolgen.

Das Konzept des KSS ist darauf ausgelegt, in einfachen Umgebungen zu funktionieren, bietet jedoch auch die Unterstützung für komplexe Set-ups. Die Datei-basierte Konfiguration ist sicherlich für Stand-alone-Umgebungen geeignet, läuft aber gegen die Grenze, wenn eine Hochverfügbarkeit erforderlich ist. Dann sollte der KSS in einem LDAP-Server beziehungsweise in einer Datenbank abgelegt sein. Diese müssen dann durch geeignete Mittel (LDAP-Replikation, Datenbank RAC, eventuell erweitert mit Data Guard) so eingerichtet sein, dass Multi-Datacenter-Set-ups unterstützt sind.

Abbildung 2 zeigt, wie eine Anwendung über das Credential Store Framework (CSF) den Credential Store (KSS) benutzt. Dabei holt das CSF die Credentials (Benutzername, Password, Keys etc.) aus dem Credential Store ab. Wichtig für den sicheren Betrieb der Anwendung ist die gleichzeitige Einbindung des Auditing, um die korrekte Funktionsweise der Anwendung zu überwachen.

Der Oracle Keystore Service enthält verschiedene Möglichkeiten zum Lifecycle Management. Diese sind unter anderem:

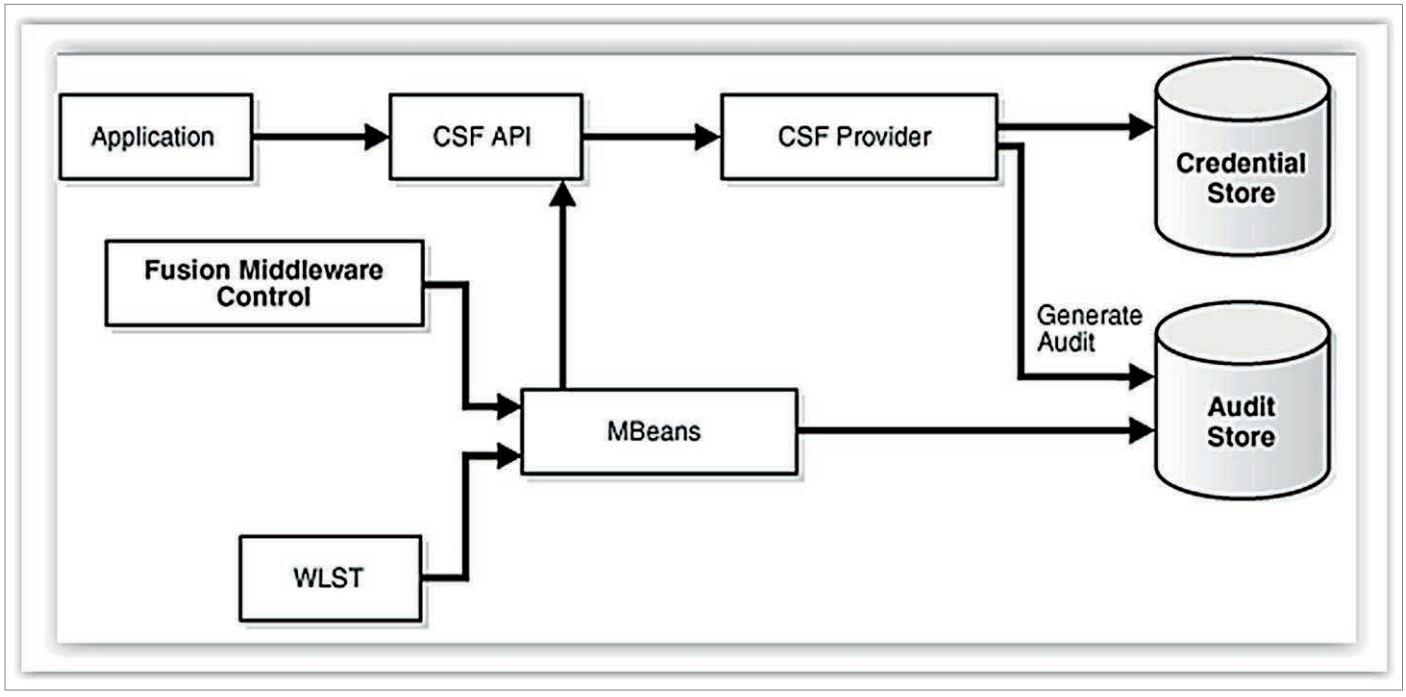


Abbildung 1: Die Position der Oracle Keystore innerhalb des OPSS

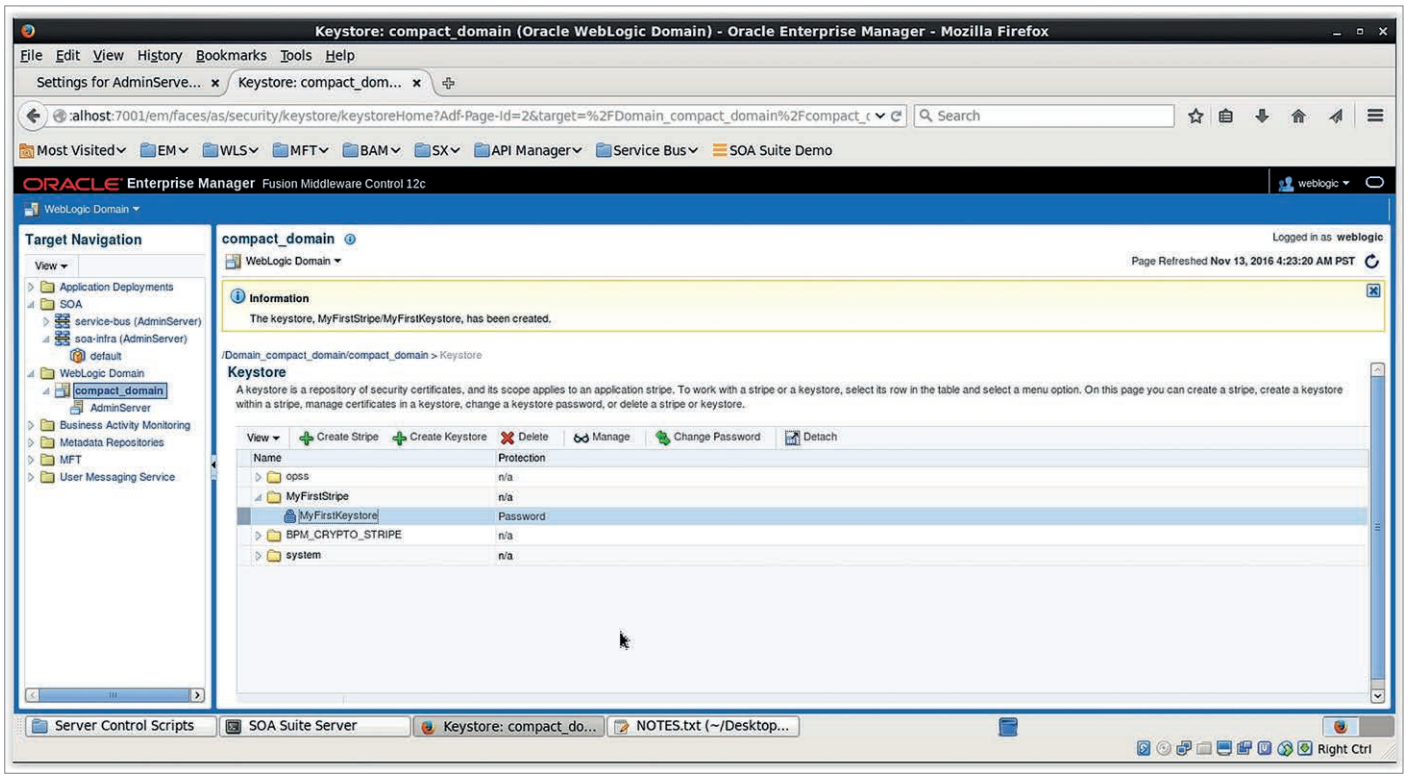


Abbildung 2: OPSS Security Services sind in den Management-Tools der Oracle Fusion Middleware integriert. Dadurch können die OPSS Security Policies und Konfigurationen mit den bekannten Werkzeugen (wie Fusion Middleware Console, WebLogic Scripting Tool, JMX MBeans) ausgeführt werden.

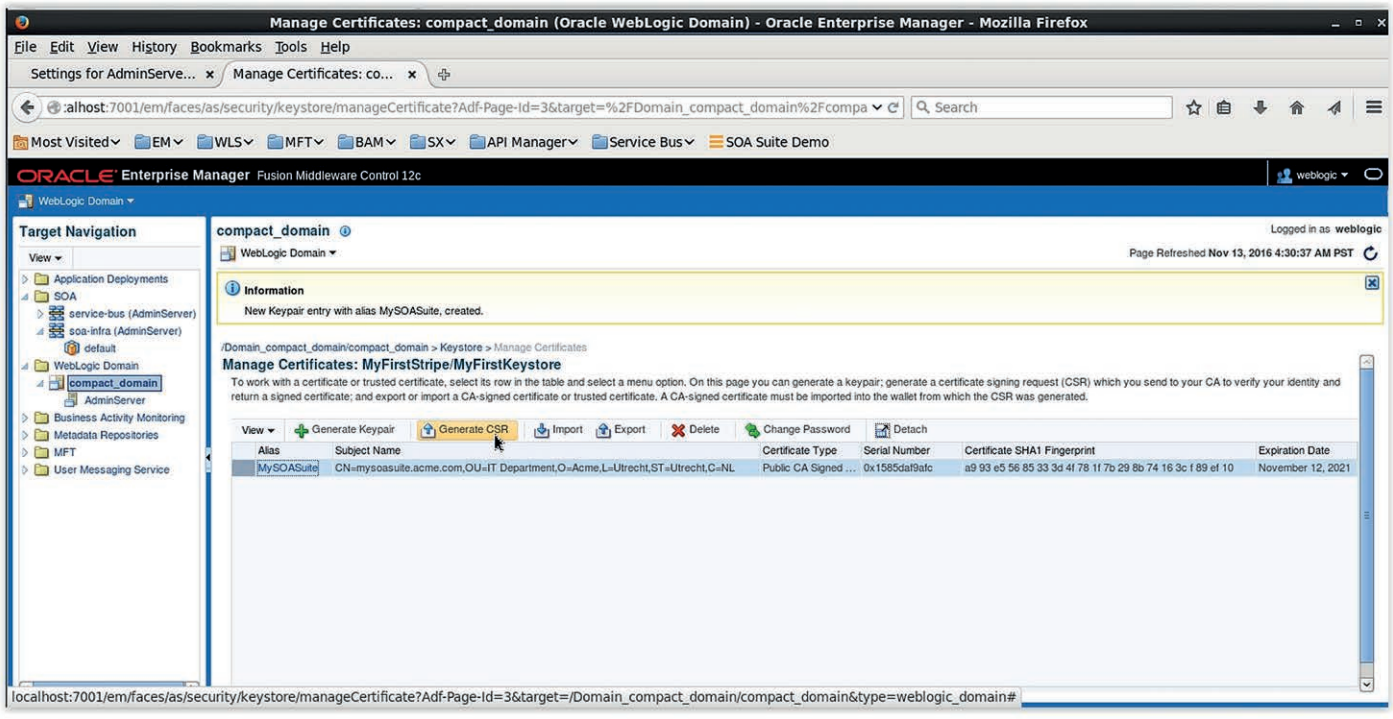


Abbildung 3: Ein Keystore in der FMW-Console

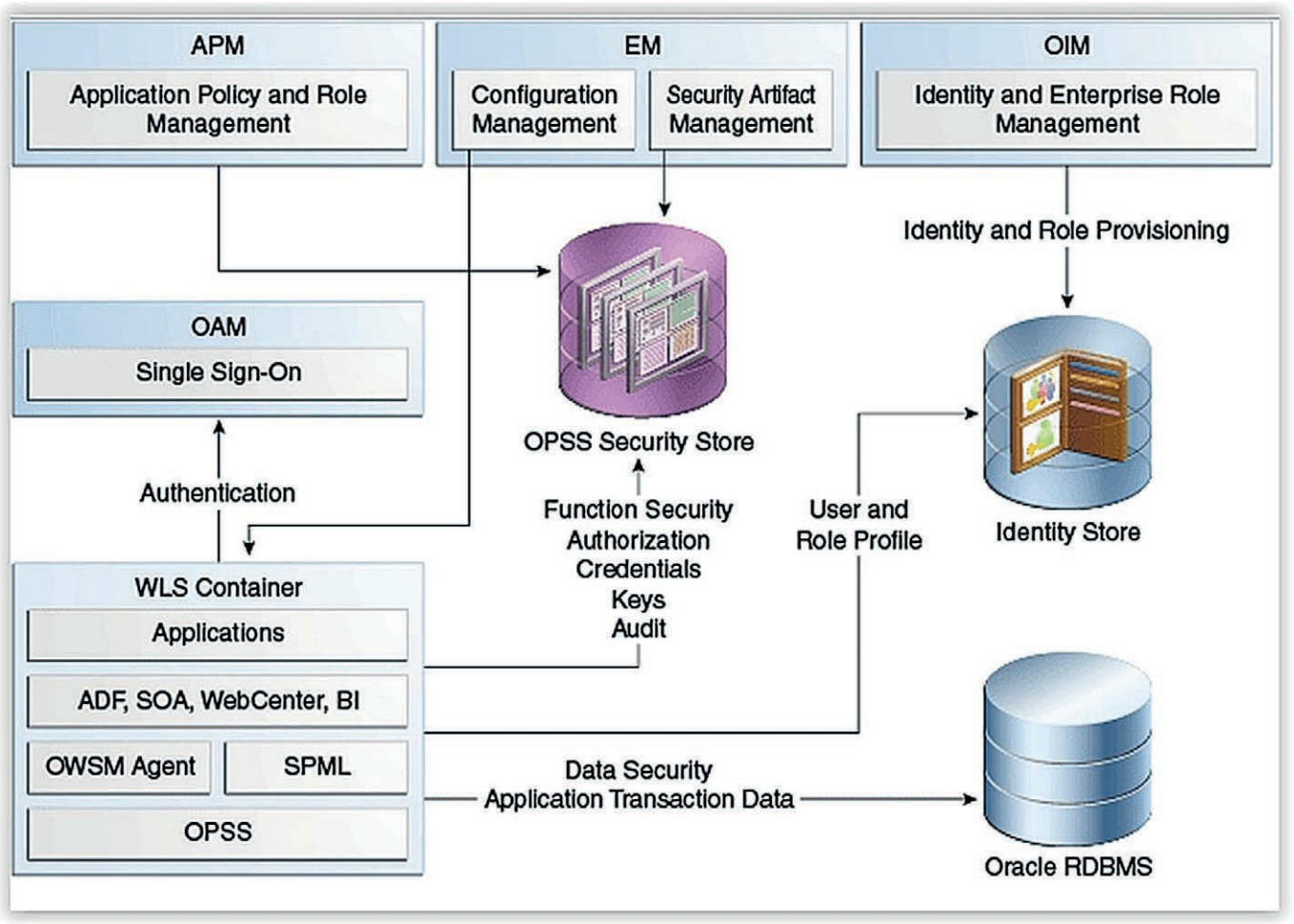


Abbildung 4: Ein Beispiel mit der FMW-Console

- Anlegen des KSS
- Update des KSS
- Löschen des KSS
- Import und Export
- Password-Änderung des KSS

Die Lifecycle-Management-Optionen sind im WebLogic Scripting Tool (WLST) und in der Fusion Middleware Console vorhanden. *Abbildung 3* zeigt einen Keystore in der Fusion Middleware Console.

Da die Sicherheit in IT-Systemen üblicherweise von Certificates abhängt, bietet der Oracle Keystore Service die Unterstützung des Certificate Management. Zu den Funktionen gehören:

- Anlegen eines Keypair
- Anlegen eines Certificate Signing Request
- Import und Export von Certificates
- Löschen eines Certificate

Auch hier sind alle Funktionen von der Fusion Middleware Console oder dem WLST ausführbar *(siehe Abbildung 4)*. Zudem können auch bestehende Certificates aus einem Java Keystore oder einer Oracle Wallet importiert werden.

Wie gezeigt, können Anwendungen die Daten des KSS benutzen, indem das CSF genutzt wird. Das CSF bietet ein API, mit dem die wichtigen Aktionen rund um den Credential Store ausgeführt werden können, wie Certificates aus einem Store holen, den Store und den Inhalt verifizieren oder die Zugangsdaten zum Store verändern. Dies erlaubt einem Entwickler dann, das Certificate aus dem Store zu extrahieren und dieses Certificate zum Verschlüsseln einzusetzen (beispielsweise mit der Java Cryptography Architecture).

Andreas Chatziantoniou
andreas@foxglove-it.nl

Konsolidierungsplattform Exadata: PoC-Erfahrungsbericht mit Licht und Schatten

Gregor Büchner und Uwe Simon, Deutsche Telekom IT GmbH, Joachim Dietsch, ORACLE Deutschland B.V. & Co. KG

In einem Proof of Concept soll die Exadata ihr Potenzial für die Konsolidierung von Oracle-Datenbanken der Telekom-IT unter Beweis stellen. In dem Artikel geht es von der Idee über die Planung bis zum PoC-Ergebnis und den Schritten danach bis zum Start eines Produktionsbetriebs. Es gibt Positives wie Negatives zu berichten.

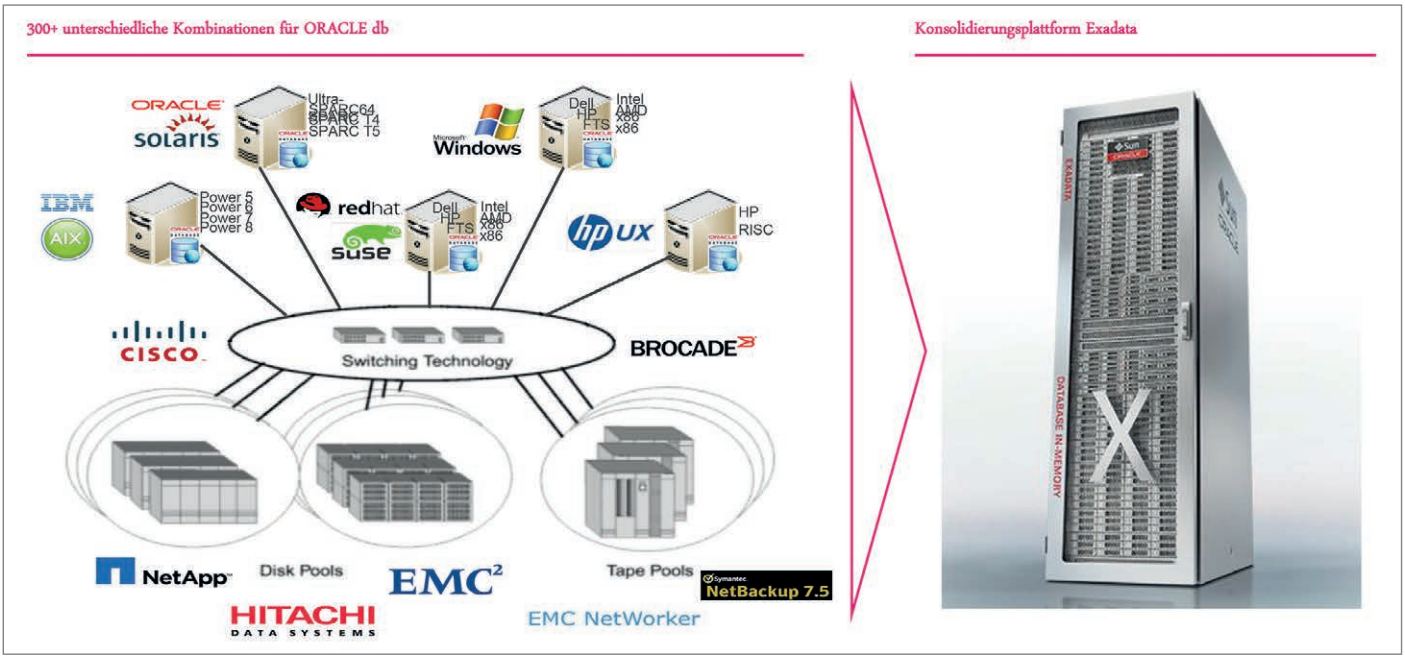


Abbildung 1: Konsolidierung und Standardisierung mit Oracle Exadata

Um einen positiven TCO zu generieren, mussten einige Annahmen erfüllt sein. So wird eine signifikante Einsparung von einzusetzenden Cores im Datenbank-Server (SMART-Scan) erwartet. Zudem gilt es, die physikalische Storage-Nutzung zu senken (HCC). Dabei darf natürlich die Gesamt-Performance nicht leiden. Einige Erwartungen wurden übererfüllt, andere weniger gut.

Das Unternehmen

Die Telekom-IT ist der interne Dienstleister der Deutschen Telekom. Sie bündelt seit dem Jahr 2012 alle IT-Einheiten des Konzerns und ist verantwortlich für das komplette IT-Service-Portfolio. Die Telekom-IT nutzt die Infrastruktur- und Cloud-Services der T-Systems Market Unit (im Wesentlichen das externe Geschäft mit Großkunden).

Die Telekom-IT betreibt mehr als tausend Oracle-Datenbanken mit einem Volumen von mehreren Peta-Byte, verteilt über mehrere Rechenzentren auf unterschiedlicher Hardware (Power/Sparc/x86) mit verschiedenen Betriebssystemen (AIX, Solaris, Linux). Für eine anstehende Datacenter-Konsolidierung ist eine kostengünstige, leistungsfähige Private-Cloud-Plattform für Oracle-Datenbanken unter Berücksichtigung unserer Security-Anforderungen gesucht. Nach einer Oracle-Insight-Studie hat sich die Telekom IT für ein PoC mit einer Exadata X5-2 entschieden. Oracle Virtual Machine (OVM) wurde eingeplant, um die Datenbanken zu isolieren und um dem Core-basierten Lizenzmodell gerecht zu werden. IO-Ressourcen-Management schützt im Fall von IO-Engpässen produktive von non-produktiven Datenbanken. Die Elastic-Configuration ermöglicht ein Demand-basiertes Wachstum der Exadata.

Die Idee

Wie beschrieben, hat die Telekom-IT eine Datacenter-Konsolidierung beschlossen. Am neuen Standort wurden dafür neue Plattformen etabliert – mit dem Ziel, für Applikationen der Telekom-IT eine standardisierte, Security-konforme Umgebung anbieten zu können.

In den heutigen Datacentern wurde ein Großteil der Server noch individuell konfiguriert und betrieben. Server von unterschiedlichen Herstellern kamen zum Einsatz. Die Installationen waren entsprechend pro Plattform standardisiert, aber nicht übergreifend. Das erschwerte Automatisierung und verlangsamte Bereitstellungszeiten entsprechend.

So entstand der Gedanke, eine optimierte Plattform für Oracle-Datenbanken zu entwickeln, die den Ansprüchen an Konsolidierung, Security und an ein standardisiertes, vereinfachtes Betriebsmodell genügen. Unterm Strich sollte die neue Plattform dann natürlich Kosten sparen und die Bereitstellungszeiten deutlich verkürzen *(siehe Abbildung 1)*.

Der Business Case

Mit einer sich ständig ändernden IT gewinnt die Business-Case-Betrachtung an Bedeutung, insbesondere in der frühen Projekt-Phase. Das Oracle-Insight-Programm unterstützt das, weil es die Business-Anforderungen analysiert und daraus eine optimale Architektur entwickelt.

Fokus einer Insight-Studie ist primär die Analyse des „Why", also die Business-Begründung. Ein weiterer Bereich, das „What" – also die Future-State-Architektur –, wurde durch ein erstes High-Level-Sizing ermittelt. Auch der dritte Bereich, das „How" – also die Roadmap –, entstand als Skizze anhand von Best-Practice-Migrations-Szenarien *(siehe Abbildung 2)*.

Das Ziel des Projekts war von Anfang an klar definiert: höhere Agilität, mindestens gleichbleibende Performance und gleichzeitig geringere Kosten. Der Current Mode analysierte zu Beginn eine potenzielle installierte Basis von mehr als 2.100 CPU-Cores, die sich auf rund 200 physikalische Server mit folgenden Plattformen verteilten (Listung nach Häufigkeit):

- AIX / Power
- Solaris / SPARC
- LINUX / x86
- HP-UX / Itanium

Zur Aufschlüsselung der Baseline im Current Mode wurden die wichtigsten Kostenfaktoren sowohl in absoluten Werten als auch in prozentualer Verteilung ermittelt *(siehe Abbildung 3)*:

- Managed Storage Costs
- Managed DB-Server Costs
- Infrastructure Costs
- DB SW-Maintenance Costs

Diese Kostenstruktur spiegelt die Ausgangslage und gleichzeitig die „do nothing"-Strategie (ebenfalls eine Option) wider. Zum Vergleich wurden die genannten Kostenfaktoren den bereits vorhandenen IaaS-Plattformen mit ihren unterschiedlichen Architekturen zugeordnet:

- IT-Serverfarm AIX
- IT-Serverfarm Solaris
- IT-Serverfarm x86

Bei einer geplanten Projekt-Laufzeit von vier Jahren konnte die Exadata-Architektur in allen vier Kostenbereichen deutliche, zusätzliche Einspar-Potenziale aufzeigen:

- 65 Prozent Gesamtersparnis gegenüber dem Current State
- Nahezu 30 Prozent gegenüber der günstigsten Implementierung auf Basis einer IT-Serverfarm

Da diese DBaaS-Architektur eine Universal-Plattform für alle anfallenden Workloads darstellen sollte (OLTP, DWH), wurde für das Sizing der Exadata eine generische Formel angewendet. „SQL Offloading" ermöglicht bei den Storage-Servern eine Auslagerung einiger Funktionen auf die CPUs der Storage Nodes, was insbesondere bei DWHs zu Beschleunigung beziehungsweise Entlastung führt. Das Sizing hat dies berücksichtigt und bei den DB-Servern die benötigten CPUs beziehungsweise Cores um 30 Prozent reduziert. Abgerundet durch eine Ramp-Up-Kurve auf Basis einer Exadata-Architektur ist ein ROI in bereits neun Monaten darstellbar *(siehe Abbildung 4)*.

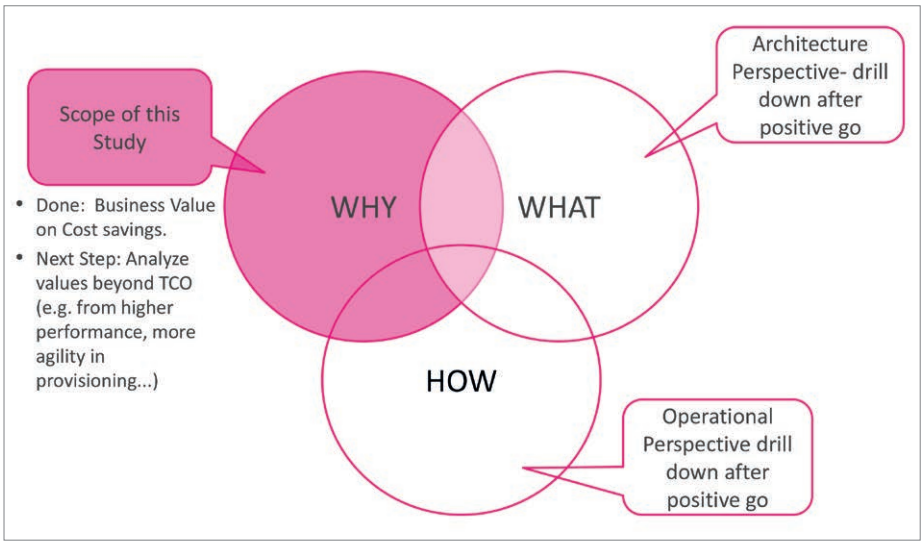


Abbildung 2: Fokus der Insight-StudieAbbildung 2: Fokus der Insight-Studie

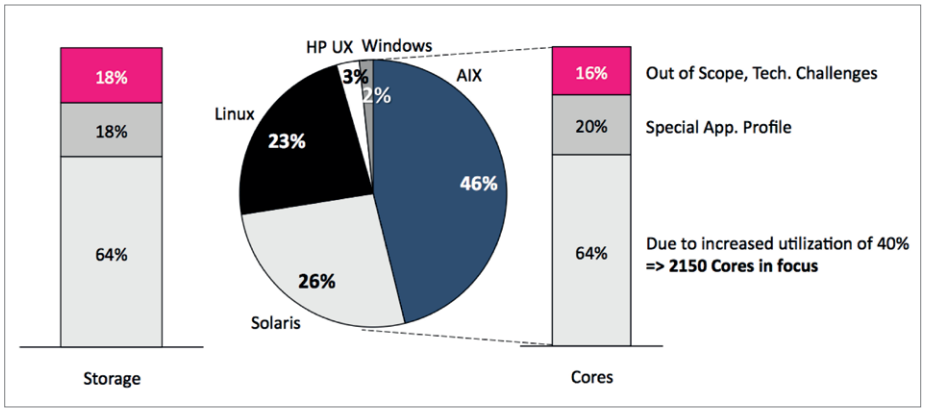


Abbildung 3: Kostenverteilung Current State (Baseline) vs. mögliche Varianten für Future State

Die Planung

Nachdem die Anforderungen an die neue Oracle-Datenbank-Plattform feststanden, wurde eine Architektur entworfen, die folgende Kernpunkte bei Bedarf erfüllen kann *(siehe Abbildung 5)*:

- Hochverfügbarkeit
- Desaster-Recovery-Fähigkeit
- Abbildung verschiedener Security-Zonen
- Isolation von Produktions- und Nicht-Produktions-Datenbanken

Nach dem Grobkonzept wurden PoC-Kandidaten ausgewählt, die eine gewisse Herausforderung im Datenbank-Umfeld darstellen: mit Blick sowohl auf die Komplexität beim Nutzen von Features der Oracle-DB als auch auf das anspruchsvolle IO-Verhalten. Das sollte sicherstellen, dass die Plattform zum einen aus technischer Sicht den allermeisten Ansprüchen gerecht wird. Zudem wollte man mit diesen in der Telekom-IT bekannten Kandidaten auch das Eis brechen für zukünftige Migrationen, getreu dem Motto: „Wenn bei denen der PoC erfolgreich war, dann haben wir erst recht keine Probleme".

Die PoC-Phase

Genehmigungen und/oder Bestellungen neuer Systeme dauern in oder zwischen Großkonzernen einige Zeit – zumal wenn die Kosten höher als üblich sind. Die Beschaffung der Exadata lief auf „Buy to try"-Basis. Dafür beschrieb der erste Schritt – abgestimmt mit Oracle – den

Proof of Concept (PoC) mit all seinen Erfüllungskriterien. Da zu den ausgewählten Systemen schon viele Performance-Kennzahlen vorliegen, war dieser Schritt einfach. Der PoC galt als bestanden, wenn die Performance der ausgewählten Tests mindestens genauso gut ist wie auf der bestehenden Performance-Testplattform und dafür weniger CPUs benötigt. Ferner gab es noch festgelegte Kriterien für die Einsparung beim Storage und für die betrieblichen Tests.

Da die technischen Voraussetzungen im Rechenzentrum vorhanden waren (Strom, Netzwerk, IP-Adressen etc.), war der physische Aufbau des PoC-Exadata-Systems in zwei Tagen erledigt. Ferner wurde noch ein Management-Server für den Oracle Enterprice Manager (OEM) bereitgestellt. Ebenso ging später der ZFS-Filer für die Backup-Anforderungen schnell in Betrieb *(siehe Abbildung 6)*.

Die Installation der Umgebung für den PoC benötigte im ersten Schritt vier Oracle Virtual Machines (OVM), die mit dem Oracle-Exadata-Deployment-Assistenten (OEDA) konfiguriert wurden. In diesen vier OVMs wollte man dann die

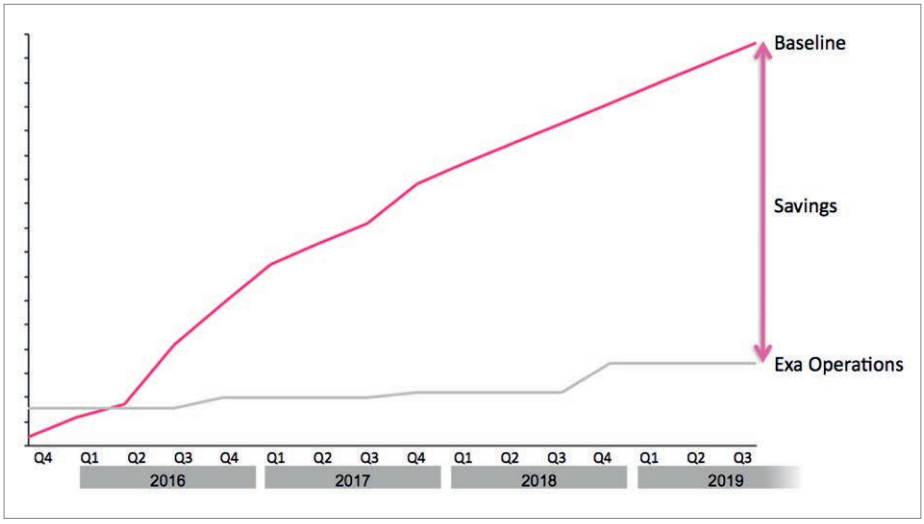


Abbildung 4: Business Case Einspar-Potenzial

sechzehn Datenbanken des PoC-Environment installieren. Als Oracle-RDBMS-Release kam Anfang 2016 noch 11.2.0.4 zum Einsatz, da die Quellsysteme zu dem Zeitpunkt noch unter 11g liefen. Mit der OEDA-Version von Anfang 2016 ließ sich die OVM noch nicht komplett in der GUI konfigurieren, da VLANs noch nicht unterstützt wurden. Wie eigentlich bei allen Projekten, bei denen sich die Architektur großer Datenbank-Systeme ändert, verursacht die Daten-Migration den meisten Aufwand; besonders, wenn man später in Produktion mit möglichst wenigen, kurzen Auszeiten auskommen möchte.

Solange man auf der gleichen CPU-

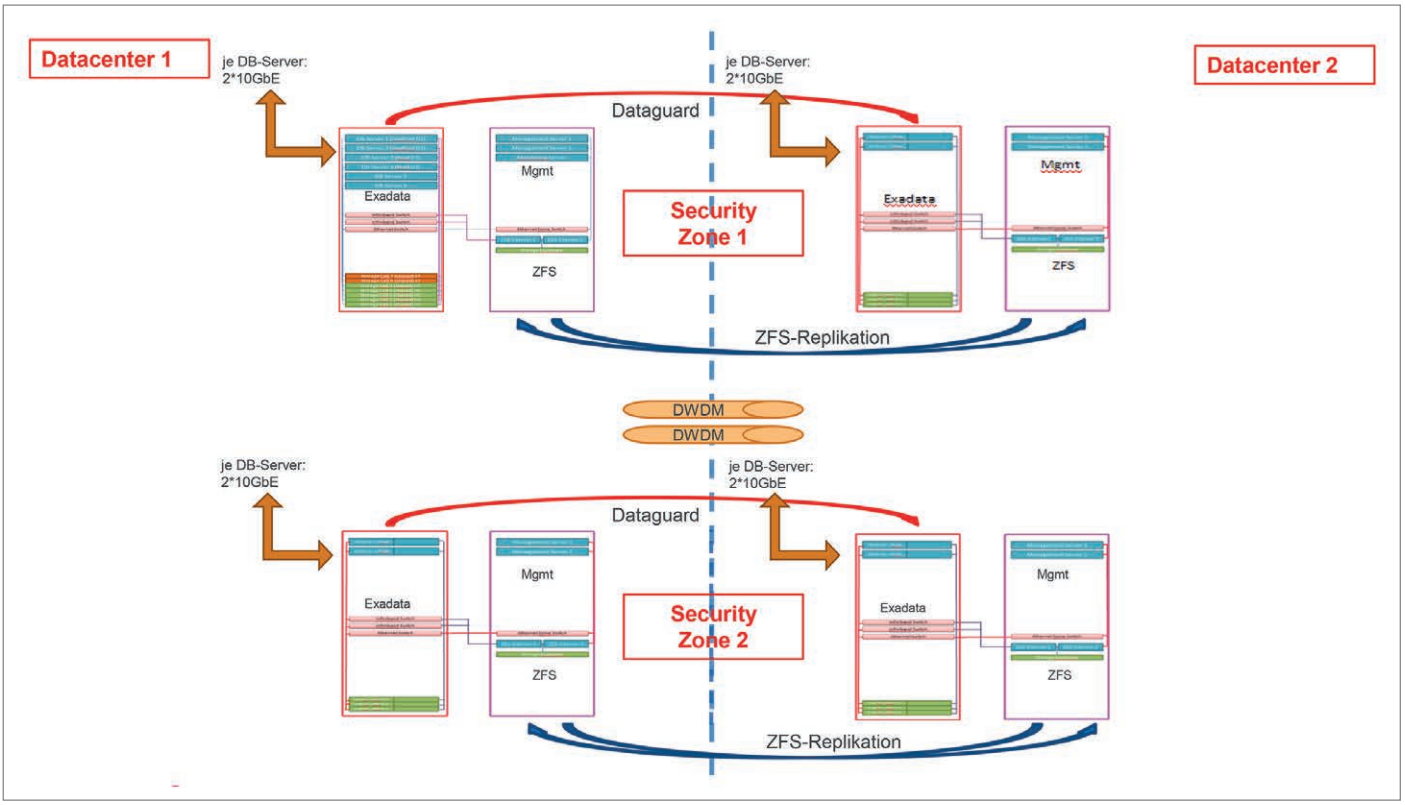


Abbildung 5: Grob-Übersicht Architektur der Konsolidierungsplattform Oracle Exadata

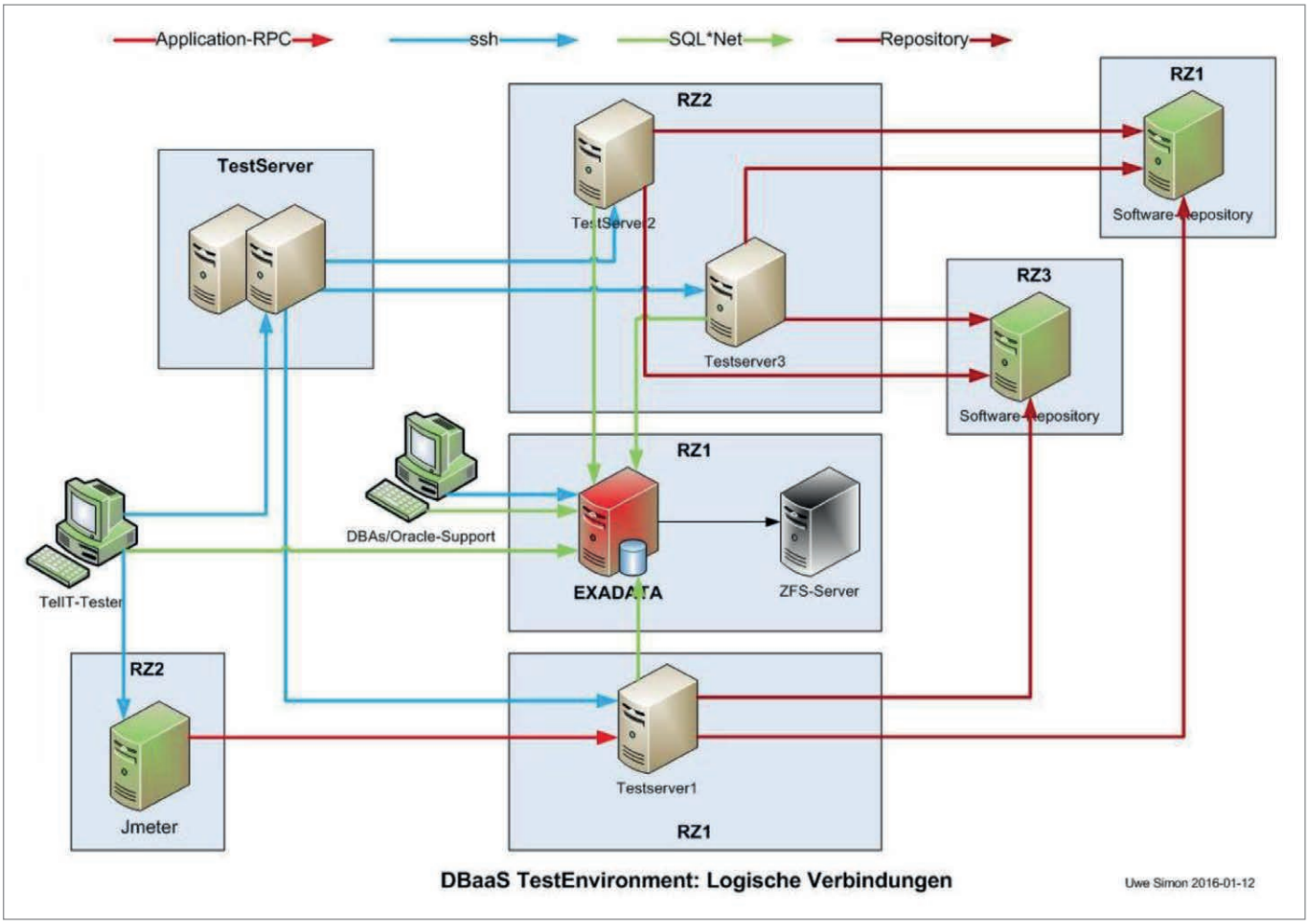


Abbildung 6: Grob-Übersicht PoC-Aufbau

Plattform bleibt, ist das bei Oracle alles mit RMAN, Data Guard, Physical-Standby etc. gut und einfach handhabbar. Sehr große Datenbanken (x-TB und hohe Transaktionsraten) verschiebt man allerdings auch damit nicht von einem ins andere Rechenzentrum. Ein einfacher NAS-Filer und eine Spedition sind hier manchmal immer noch der schnellste, kostengünstigste Weg.

Beim Wechsel der CPU macht einem Oracle das Leben nicht gerade einfach. RMAN kann leider nur Datenbanken automatisch von einer CPU-Plattform auf eine andere verschieben, solange die Endianess der beiden CPUs identisch ist. Das gilt nicht für Redo-Logs, sodass Data Guard nicht infrage kommt. Diese Konstellation existierte im PoC jedoch nicht (Power7/8 zu Intel x86-64Bit). Somit bleibt nur der mühsame Weg, über Transportable Tablespaces – mit den entsprechenden Nacharbeiten (zusätzliche Skripts für Accounts, Grants, Synonyms auf Quelle generieren etc.). Im Rahmen einer Konsolidierung von Hunderten Datenbanken muss das aber möglichst ohne manuelle Aktivitäten und natürlich fehlerfrei laufen. Um kurze Auszeiten zu erreichen, muss später die produktive Migration über Logical Stand-by erfolgen (besonders bei großen Datenbanken).

Die Daten-Migration über Transportable Tablespaces hat für den Performance-Test noch den positiven Effekt möglichst vergleichbarer Ergebnisse. Da eine physische Kopie der Daten existiert, entfallen aus Sicht der Exadata positive Effekte durch eine Reorganisation der Tabellen beziehungsweise Indizes (bei Export/Import oder anderen Datentransfer-Lösungen).

Bei dem für den Performance-Test ausgewählten CRM-System haben sich in der Vergangenheit Änderungen an der System-Architektur mehr oder weniger deutlich bemerkbar gemacht. Besonders Änderungen an der Storage-Architektur haben sehr starke Einflüsse. Das ist hauptsächlich darauf zurückzuführen, dass man pro Terrabyte Tablespace immer weniger Harddisks benötigt. Gerade bei Random-Access-Zugriffen macht sich das deutlich bemerkbar. Die Anzahl der Random-Access-Zugriffe pro Disk hängt direkt von deren Umdrehungszahl ab (15.000, 10.000 oder 7.200 RPM). Dies ergibt Werte zwischen 200 und 400 Random-Access-IOs/Sekunde je Disk. Eine Exadata in der High-Capacity-Konfiguration hat pro Full-Rack 168 Disks mit 7.200 RPM, lässt also maximal etwa 32.000 Random-Access-IOs/Sekunde von der Disk zu. Im ausgewählten CRM-System gab es aktuell mittags in einer Datenbank schon rund 20.000 Random-Access-IOs von Disks. Kritisch ist dies besonders dann, wenn auf Daten zugegriffen wird, die Wochen/Monate nicht in Benutzung waren und somit auch nicht im Cache sind. Auf der beste-

Datenbank-Funktionalität	Fachliche Funktionalität
Fullscan	Laden von externen Applikationscaches
RandomAccess	Kundendaten-Zugriff (Callcenter, Webportale etc.)
RandomAccess plus Fullscan	Normaler Betrieb
Fullscan plus Datenbank-Link	Laden aktueller Kundenstammdaten zur Rechnungsschreibung

Tabelle 1

henden Plattform musste der Einsatz von SSDs die geringere Random-Access-IO-Rate beim Wechsel der Harddisks im Storage von 15.000-RPM-Disks auf größere 10.000-RPM-Disks ausgleichen.

Die Performance-Tests gliedern sich in Datenbank- und fachliche Funktionalität *(siehe Tabelle 1)*. Um möglichst vergleichbare Messwerte zu erhalten, fanden die ausgewählten Performance-Tests als Erstes auf der produktionsnahen Performance-Testumgebung statt. In der bestehenden Umgebung sind schon die kritischen Tabellen auf SSDs gespeichert. Für den ersten Test wurden diese kritischen Tabellen „1:1" auf den Extreme-Flash-Storage der Exadata übertragen. In einem zweiten Test-Durchlauf wurden die Tabellen auf die normalen Disks (High-Capacity-Storage) abgelegt und ein entsprechender Anteil des FlexCache der Exadata konfiguriert. So hatte man einen Vergleich, inwieweit sich mit dem Cache die Performance der langsameren Harddisks ausgleichen ließ.

Für den Kompressionstest wurden die großen Objekte ausgewählt, bei denen wenige Tabellen den Großteil des Storage belegen. Generell gilt: Was man mit den großen Objekten nicht erreicht, kann man mit den kleinen Objekten nicht mehr herausholen. Da diese ausgewählten großen Tabellen sehr viele IDs (aus Sequenzen) und Zeitstempel haben (und somit die Zeilen nicht sehr ähnlich sind), war die Erwartung an die Kompressionsfaktoren von HCC deutlich niedriger, als die Oracle-Werbung suggeriert.

Das beste System ist nur halb so gut, wenn es sich nicht in das bestehende Environment integrieren lässt. Betriebliche Tests sollten das überprüfen. Die wesentlichen Punkte sind „Datensicherung", „Einbinden in die bestehende Betriebsüberwachung", „Bereitstellen von OVMs und Datenbanken", „Messen der Auslastung", Planen der Kapazitäten". Die Datensicherung erfolgt per RMAN auf Disk auf einen ZFS-Filer, der direkt über InfiniBand an die Exadata angeschlossen ist. Das ermöglicht schnelles Backup und Restore gerade von großen Datenbanken.

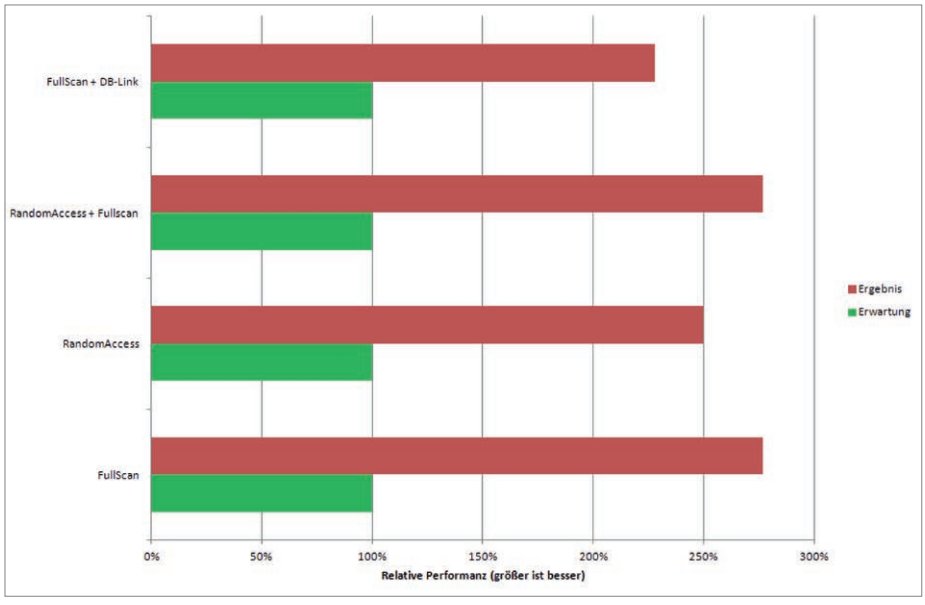


Abbildung 7: Zusammenfassung der Ergebnisse der Performance-Tests

Das PoC-Ergebnis

Wie erwartet hat sich bei der Exadata die beste Performance bei „DWH-like"-Abfragen gezeigt. Hier kann das System alle seine Vorteile ausspielen. Bei Applikationen mit Einzelsatz-Verarbeitung kann aber auch eine Exadata die Physik nicht überlisten (eine 8-TB-Harddisk mit 7.200 RPM kann nicht beliebig viele Random-Access-Zugriffe/Sekunde). ExtremeFlash und FlexCache können diese Nachteile beim Random-Access-IO jedoch kompensieren. ExtremeFlash-Storage steigerte die Performance zwischen 220 und 280 Prozent. FlexCache verlangsamte die Prozesse beim ersten Starten deutlich (Daten kommen direkt von der Disk). Je länger der Cache läuft, desto schneller wurden sie dann, um nach einigen Minuten fast an die ExtremFlash-Konfiguration heranzukommen *(siehe Abbildung 7)*.

Mögliche Storage-Einsparungen durch Hybrid Columnar Compression (HCC) hängen wesentlich von den Daten-Inhalten und den Zugriffen auf diese Daten ab. Hier verspricht die Werbung deutlich mehr, als in diesem Fall möglich ist. Die guten Werte lassen sich hauptsächlich bei „DWH-like"-Datenbanken erreichen. Bei einer Konsolidierungsstrategie auf Exadata wird es immer einige große Datenbanken geben, die von einer Kompression profitieren. Die Zahl solcher Datenbanken ist allerdings eher klein; da lassen sich kaum kompressi-

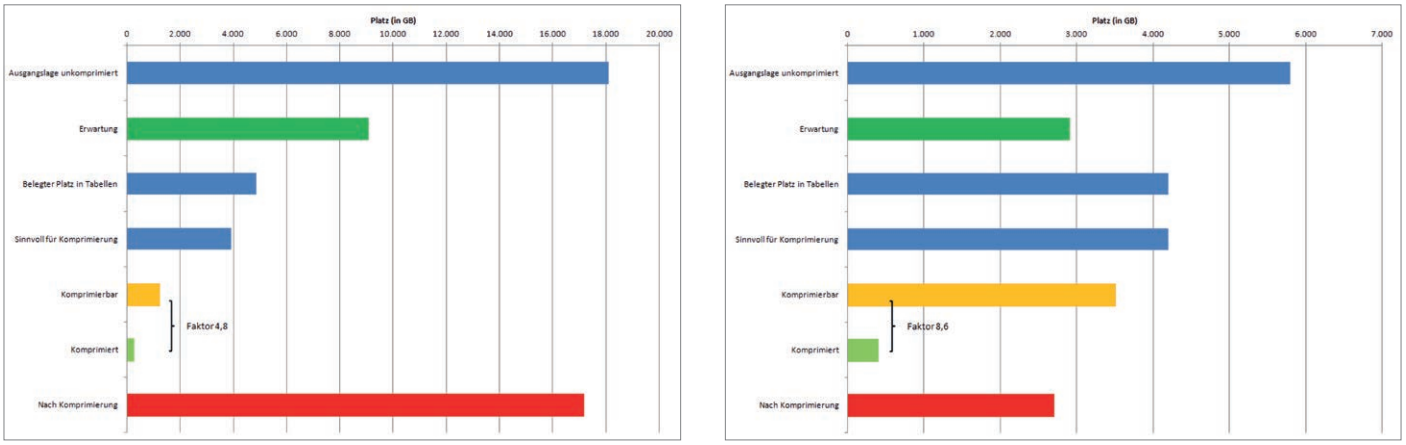


Abbildung 8: Ergebnis Kompressionstest OLTP (links) und DSS (rechts)

onswürdige Daten finden beziehungsweise der Aufwand, dies auszunutzen, steht in keinem Verhältnis zur potenziellen Einsparung. In den Test-Datenbanken ließen sich nur rund sieben Prozent der Daten komprimieren. Die Kompressionsrate lag bei einem Faktor von 4,8. Somit ergab sich eine Einsparung von nur rund fünf Prozent. Auch bei anderen Datenbanken zeigten sich entsprechende Faktoren bei komprimierbaren Tabellen. Generell sollte man bei der Komprimierung im Zuge von Datenbank-Konsolidierungen nicht zu optimistisch sein *(siehe Abbildung 8)*.

Eine rechnerische Ersparnis beim Storage im Vergleich zur bestehenden Infrastruktur ergibt sich aber doch: Im „current state" werden sowohl Betriebssystem und Software als auch Datenbanken auf SAN-Storage abgelegt und entsprechend abgerechnet. Auf der Exadata liegen Betriebssystem, Software etc. auf den lokalen Platten der Datenbank-Server und somit steht die Kapazität der Storage-Nodes komplett für die Datenbanken zur Verfügung. Nach dem neuen Verrechnungsmodell wird hier nur noch die allokierte Menge abgerechnet („pay what you use").

Der Grad einer möglichen Reduzierung bei den CPU-Cores hängt wesentlich von den Datenbank-Inhalten und deren Nutzung ab. Die Reduktion ergibt sich (prinzipbedingt) nur, wenn entsprechend viele SQL-Abfragen vom Query-Offloading (Smart-Scan) der Storage-Server profitieren. Dies ist bei vielen relativ kleinen Datenbanken oder den großen CRM-Systemen meist nicht der Fall. Hier darf man auch nicht zu optimistisch herangehen.

Für den Betrieb wird eine einheitliche Konfiguration der Exadata-Systeme, der OVMs und der Datenbanken angestrebt. Nur so kann man einen hohen Automatisierungsgrad erreichen. Mit sehr wenigen Konfigurationsparametern müssen sich möglichst viele Datenbanken „out of the box" komplett mit einem Click bereitstellen lassen können (OVM, Oracle-Software, Datenbank-Instanz, RMAN-Backup etc.). Jede Abweichung in den Konfigurationen verursacht manuelle Eingriffe.

Die Erwartungshaltung für den Produktionsbetrieb ist, dass bei steigender Auslastung der Exadata die Performance der einzelnen Applikationen zwar nachlassen wird, aber weiterhin über dem Niveau der bestehenden Plattformen bleibt. Ziel ist ja, die Exadata-Systeme − bei gleicher Performance wie bisher − möglichst gut auszulasten. Um dies zu erreichen, ist im laufenden Betrieb gegebenenfalls mit weiteren ExtremFlash-Modulen zulasten des Business-Case nachzurüsten.

Aus Sicht des Technikers ist es immer gut, wenn man Reserven hat und damit die Performance noch steigern kann. Das reduziert das technische Risiko. Hier muss man allerdings immer zwischen Business Case und Performance abwägen, es gibt eben nichts umsonst.

Fazit

Nach Abschluss des PoC mit positivem Ergebnis und einem ebenfalls sehr positiven Gesamt-Business-Case hat die Telekom-IT entschieden, mit dieser Plattform für Oracle-Datenbanken produktiv zu gehen. Es wurden die Betriebsabläufe weiter optimiert und die Bestellprozesse entsprechend angepasst. Die Migrationsverfahren, um Datenbanken von AIX/Solaris zu migrieren, wurden weiter verfeinert.

Das OVM-Handling ist noch nicht so, wie man es sich vorstellen könnte. So deckt der OEM noch nicht alles ab, was mit Virtualisierung auf der Exadata möglich ist. Unter anderem ist das Handling von mehreren VLANs an einem Datenbank-Server noch manuell. Es gibt einen Kontakt zur Oracle-Entwicklung, um hier eine Lösung zu finden.

Die nächsten Schritte

Der Plan ist nun, Anfang 2017 die komplette Plattform über zwei Data-Center-Standorte aufzubauen. An jedem Standort werden dann zwei physisch getrennte Security-Zonen aufgebaut, deren Datenbanken bei Anforderung über Data Guard an den zweiten Standort repliziert werden.

Gregor Büchner
gregor.buechner@t-systems.com

Datenmodellierung in Cassandra und die Cassandra Query Language (CQL)

Jan Ott, Trivadis AG

Apache Cassandra ist eine NoSQL-Datenbank und konkurriert somit nicht nur mit anderen NoSQL-Datenbanken wie zum Beispiel MongoDB, sondern auch mit traditionellen SQL-Datenbanken wie der Oracle-Datenbank. In der letzten Ausgabe haben wir das Prinzip vorgestellt, in diesem Artikel geht es darum, wie die Daten hereingebracht sowie herausgeholt werden und wie man Strukturen für die Daten erstellt.

Basis für eine erfolgreiche Nutzung von Cassandra ist neben dem Einsatz der passenden Hardware einschließlich Storage-System das richtige Datenmodell. Bei Cassandra handelt es sich um eine sogenannte „Wide-Row-Datenbank", die nicht komplett Schema-frei ist, aber ein flexibles Schema unterstützt.

Auch wenn Cassandra Objekte hat, die an eine relationale Datenbank (RDBMS) erinnern (Tabellen, Primary-Key, Index), sollte man Daten nicht nach den etablierten Best Practices von RDBMS modellieren. Es

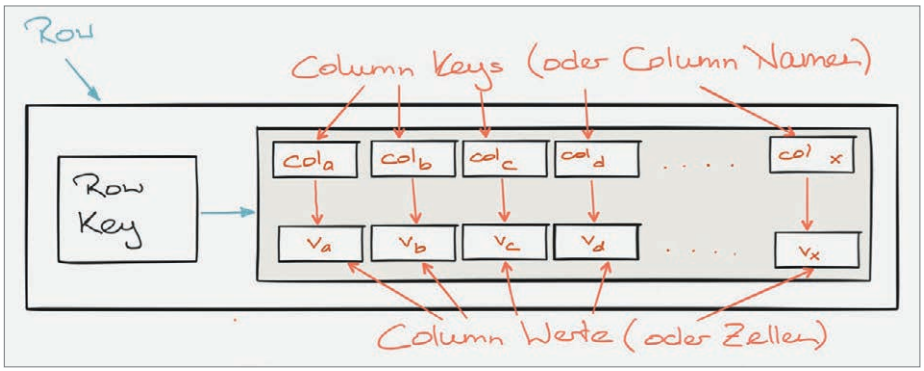


Abbildung 1: Cassandra wide Row

```
CREATE KEYSPACE IF NOT EXISTS my_space
          WITH replication = {'class': 'SimpleStrategy',
                              'replication_factor' : 1};
```

Listing 1

gibt bei Cassandra beispielsweise keine Foreign-Keys, keine referenzielle Integrität und auch keine Joins. Tabellen in Cassandra sind im Wesentlichen nichts anderes als eine riesige Map von sortierten Columns: „Map<RowKey, SortedMap<ColumnKey, ColumnValue>>".

Die Map erlaubt einen effizienten Zugriff über den Key und die sortierten Columns lassen effiziente Scans zu. In Cassandra werden dazu der Row-Key und der Cluster-Key/Column-Key verwendet. Die maximale Anzahl unterschiedlicher Column-Keys je Row ist dabei riesig, mit einer theoretischen Obergrenze von zwei Milliarden Column-Keys. Eine Cassandra-Tabelle kann also nicht nur in die Tiefe, sondern auch in die Breite wachsen. Eine Row ist immer komplett auf einem Node gespeichert und die Platzierung im Cluster wird wie beschrieben über den Row-Key bestimmt. Daher heißt die Row auch „Partition" *(siehe Abbildung 1)*.

Auch wenn eine Cassandra-Row grundsätzlich beliebig und dynamisch in die Breite wachsen kann, ist dies nicht in jedem Fall gewünscht und gefordert. Cassandra lässt es auch zu, Tabellen zu definieren, die statisch immer die gleichen Werte besitzen, analog zu RDBMS-Tabellen. Diese Tabellen nennt man „skinny Row". Im Gegenzug heißen die Tabellen, die dynamisch in die Breite wachsen, auch „wide Row".

Wie lässt sich das in der Praxis nutzen und wie geht man mit den fehlenden Joins und Foreign-Keys um? Die Lösung heißt „Denormalisierung". Das richtige Datenmodell zu finden, ist die schwierigste Aufgabe bei der Nutzung von Cassandra. Auch wenn die Cassandra-Tabellen stark den relationalen Tabellen ähneln, ist deren Nutzung stark unterschiedlich. Die wichtigsten zwei Regeln bei der Modellierung in Cassandra sind:

- Daten sollten sich möglichst gleichmäßig über den Cluster verteilen.
- Die Anzahl der Partitionen beziehungsweise Rows, die gelesen werden müssen, soll minimiert werden. Im besten Fall ist das genau nur eine Partition.

Um das bestmögliche Datenmodell zu finden, muss bei Cassandra zwingend von den Abfragen ausgegangen werden, die es zu beantworten gilt. Die Modellierung nach Beziehungen oder nach Objekten bietet absolut keinen Mehrwert. Wie sieht das in der Praxis aus? Wir verwenden zur weiteren Illustration das altbekannte EMP/DEPT-Schema, wie wir es von der Oracle-Datenbank kennen, und modellieren es in der Welt von Cassandra.

Als ersten Schritt gilt es, genau zu bestimmen, welche Abfragen Cassandra unterstützen soll. Dabei sind Dinge wie Gruppierung von Attributen, Sortierung, Filterung aufgrund von Bedingungen etc. wichtig. In unserem Fall beschränken wir uns auf folgende drei Abfragen:

```
CREATE TABLE emp(
  empno    INT,
  ename    VARCHAR,
  job      VARCHAR,
  mgr      INT,
  hiredate TIMESTAMP,
  sal      FLOAT,
  comm     FLOAT,
  deptno   INT,
  PRIMARY KEY (empno)
);
```

Listing 2

- Zugriff auf einzelne Mitarbeiter über seine Mitarbeiternummer („EMPNO")
- Zugriff auf eine Abteilung über ihre Abteilungsnummer („DEPTNO")
- Zugriff auf die zuletzt eingestellten Mitarbeiter (nur „EMPNO" und „ENAME" notwendig) einer bestimmten Abteilung, absteigend sortiert nach

```
INSERT INTO emp (empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES( 7839, 'KING', 'PRESIDENT', null, '1981-11-17', 5000, null, 10);
INSERT INTO emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES( 7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, null, 10);
INSERT INTO emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES( 7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, null, 20);
INSERT INTO emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES( 7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, null, 30);
. . .
```

Listing 3

```
SELECT empno, comm, job FROM emp WHERE empno = 7900;

 empno | comm | job
-------+------+-------
  7900 | null | CLERK
```

Listing 4

```
SELECT empno, comm, job FROM emp WHERE ename = 'JAMES';

InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data filtering
and thus may have unpredictable performance. If you want to execute
this query despite the performance unpredictability, use ALLOW FILTER-
ING"
```

Listing 5

Einstellungsdatum („HIREDATE“) mit optionaler Einschränkung auf einen bestimmten Zeitbereich

Im nächsten Schritt gilt es, die Tabellen zu bestimmen, die diese Abfragen unterstützen, und zwar im besten Fall über einen Zugriff auf genau eine Partition. Cassandra verwendet für die Interaktion mit der Datenbank die Cassandra Query Language (CQL). Damit lassen sich einerseits die Datenstrukturen verwalten (Data Definition Language – DDL), andererseits Daten manipulieren und abfragen (Data Manipulation Language – DML). CQL ist dabei stark an die Structured Query Language (SQL) angelehnt, die wir von den RDBMS-Datenbanken kennen und schätzen. Wer schon mit SQL gearbeitet hat, wird sich schnell zu Hause fühlen.

Als Erstes braucht es in Cassandra einen Keyspace, analog zu dem Schema bei der Oracle-Datenbank. Neben dem Namen werden in Cassandra zusätzlich der Replikations-Faktor (wie viele Kopien) und die Replikations-Strategie (einzelner Cluster oder Multi-Datacenter) angegeben *(siehe Listing 1)*.

Wir sehen zudem, dass es in Cassandra die Option „if not exists” gibt. Der Keyspace wird nur angelegt, wenn er nicht schon existiert. Die Nutzung des Keyspace erfolgt über den USE-Befehl „USE my_space;“.

Nun sind wir soweit, die entsprechenden Tabellen definieren zu können. Wie erwähnt, orientieren wir uns dazu an den Abfragen, die es zu unterstützen gilt.

Für die erste Anforderung reicht eine „skinny Row“-Tabelle aus, da wir jeden Mitarbeiter mit einer fixen Anzahl von Columns beschreiben, die wir zum Designzeitpunkt fixieren können *(siehe Abbildung 2)*. Wir nutzen den Befehl „Create Table“, der sehr stark einer RDBMS-Tabelle ähnelt *(siehe Listing 2)*. Für das Einfügen der Daten nutzen wir den „Insert“-Befehl, der syntaktisch gleich ist wie das Pendant in SQL *(siehe Listing 3)*.

Cassandra unterstützt keine Transaktionen. Daher ist weder ein Commit notwendig noch ein Rollback möglich. Um unsere Anforderung zu erfüllen, müssen wir auf den Mitarbeiter über seine Mitarbeiternummer zugreifen können. Dies geschieht auch bei Cassandra mit der „SELECT“-Operation *(siehe Listing 4)*. Wir schränken dabei über den Row-Key (Primary-Key) ein, sodass Cassandra genau

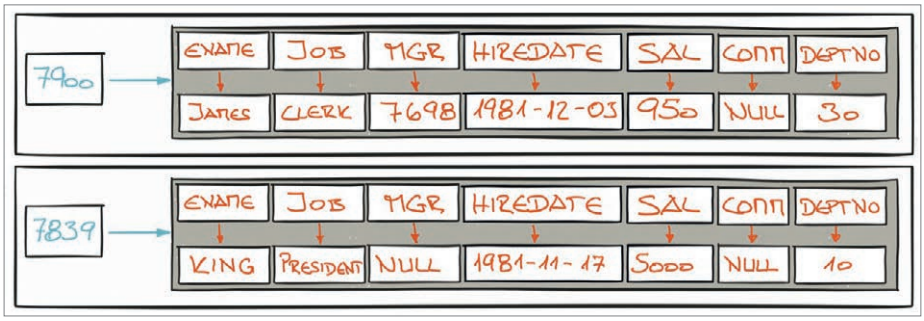


Abbildung 2: Struktur der Tabelle EMP in Cassandra

nur eine Partition lesen muss. Ein „SELECT“ mit einer „WHERE“-Klausel ohne Angabe des Primary-Key führt jedoch zu einem Fehler *(siehe Listing 5)*.

Cassandra hat hier eine Notbremse eingebaut, da so ein Statement zu einem „Full Cluster Scan“ führe würde. Daher ist „ALLOW FILTERING“ nur mit großer Vorsicht zu verwenden. Bei Netflix würde dies beispielweise bedeuten, dass 2.700 Nodes alle einen Full Scan auf die Tabelle machen, was die Performance des ganzen Clusters beeinträchtigen würde. Über die „IN“-Klausel können auch mehrere Mitarbeiter in einem Statement gelesen werden *(siehe Listing 6)*.

Hier gilt es zu berücksichtigen, dass wir damit nicht nur auf eine Partition zugreifen, sondern auf mehrere Partitionen, abhängig von der Anzahl der Werte in der

```
SELECT empno, comm, job FROM emp WHERE empno IN (7900,7902);

 empno | comm | job
-------+------+---------
  7900 | null |   CLERK
  7902 | null | ANALYST
```

Listing 6

```
CREATE TABLE dept(
  deptno INT,
  dname  VARCHAR,
  loc    VARCHAR,
  PRIMARY KEY (deptno)
);
```

Listing 7

```
INSERT INTO dept (deptno,  dname, loc)
VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept (deptno,  dname, loc)
VALUES(20, 'RESEARCH', 'DALLAS');
INSERT INTO dept (deptno,  dname, loc)
VALUES(30, 'SALES', 'CHICAGO');
INSERT INTO dept (deptno,  dname, loc)
VALUES(40, 'OPERATIONS', 'BOSTON');
```

Listing 8

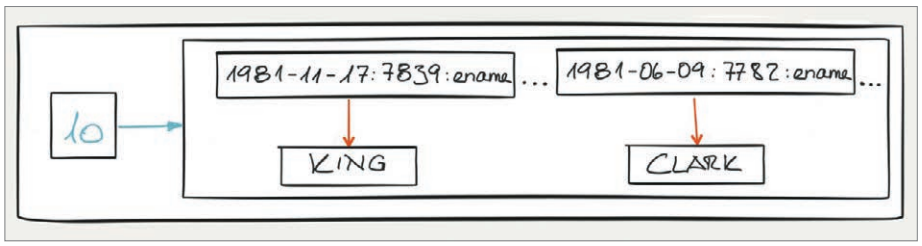


Abbildung 3: Row-Key „10“ mit den zwei „ENAME“

„IN“-Klausel. Für die zweite Anforderung nutzen wir ebenfalls eine „skinny Row“-Tabelle *(siehe Listing 7)*. Das Einfügen über „INSERT“ und die Auswertung per „SELECT“ sind dabei identisch zur EMP-Tabelle *(siehe Listing 8)*.

Die dritte Anforderung können wir aufgrund der fehlenden Joins nicht effizient über die beiden bestehenden Tabellen erfüllen. Wir könnten zwar bei der Tabelle „EMP“ einen Index auf „DEPTNO“ anlegen, es wäre damit aber nicht möglich, die Daten sortiert nach „HIREDATE“ und nur für einen bestimmten Zeitbereich zurückzuerhalten. Es müssten also alle Mitarbeiter in das Client-Programm eingelesen und der Join mit der „DEPT“-Tabelle im Client durchgeführt werden, was alles andere als ideal wäre. Cassandra kennt keine „ORDER BY“-Klausel, die über mehrere Rows hinweg sortiert. Diese wäre für Cassandra auch nicht effizient durchzuführen, da die Rows der Tabelle „EMP“ ja sinnvollerweise über den ganzen Cluster verteilt sind.

Um dies zu vermeiden, greifen wir bei Cassandra zum Mittel der Denormalisierung. Die Daten werden redundant in einer zusätzlichen Tabelle in anderer Struktur gespeichert; es werden also quasi benutzerbestimmte Indizes erstellt. Für unsere Problemstellung ist eine „wide Row“-Tabelle sinnvoll: „DEPTNO“ wird dabei als Row-Key verwendet, es gibt also eine Partition je Abteilung und die Daten der Mitarbeiter, die zur Abteilung gehören, werden als dynamische Columns der Row hinzugefügt, absteigend sortiert nach „HIREDATE“ *(siehe Abbildung 3)*.

Die „EMPNO“ nehmen wir zusätzlich zu „HIREDATE“ in den Clustering-Key auf, um die Eindeutigkeit des Primary-Key sicherzustellen *(siehe Listing 9)*. Das Einfügen über „INSERT“ und die Auswertung per „SELECT“ sind dabei identisch zur EMP-Tabelle *(siehe Listing 10)*.

Somit kann die Anforderung „Zugriff auf die zuletzt eingestellten Mitarbeiter (nur „EMPNO“ und „ENAME“ notwendig) einer bestimmten Abteilung, absteigend sortiert nach Einstellungsdatum („HIREDATE“) mit optionaler Einschränkung auf einen bestimmten Zeitbereich“ effizient unterstützt werden, wie dies der „SELECT“ in *Listing 11* zeigt.

Hier sehen wir, dass die Sortierung auch ohne Angabe von „ORDER BY“ korrekt ist, da die Columns auch physisch in dieser Reihenfolge abgespeichert sind. Ein „ORDER BY“ über Columns, die nicht zum Clustering-Key gehören, führt zu einem Fehler *(siehe Listing 12)*. Über den Clustering-Key lassen sich auch Wertebereiche einschränken *(siehe Listing 13)*.

Diese Form der Speicherung als „wide Row“-Tabelle eignet sich auch hervorragend, um Zeitreihen zu speichern und effizient auszuwerten, was gerade bei IoT-Use-Cases oft notwendig ist. In diesem Fall werden eine Sensor-Instanz zum Row-Key und der zu einem bestimmten Zeitpunkt ermittelte Messwert als dynamische Column an die Partition angehängt.

```
CREATE TABLE emp_by_dept(
   deptno    INT
  ,empno     INT
  ,ename     VARCHAR
  ,hiredate TIMESTAMP
  ,PRIMARY KEY ((deptno), hiredate, empno)
  )
WITH CLUSTERING ORDER BY (hiredate DESC, empno ASC);
```

Listing 9

```
INSERT INTO emp_by_dept (deptno, empno, ename,  hiredate)
                 VALUES (10,     7839,  'KING', '1981-11-17');

INSERT INTO emp_by_dept (deptno, empno, ename,   hiredate)
                 VALUES (10,     7782,  'CLARK', '1981-06-09');
```

Listing 10

CQL ist nicht gleich SQL

Auch wenn die Syntax von CQL sehr stark an SQL angelehnt ist, sollte man sich nicht irreführen lassen und das Gefühl haben, es handle sich bei Cassandra einfach um eine andere SQL-Datenbank. Wir haben schon gesehen, dass es viele Klauseln in CQL gar nicht gibt, etwa den „JOIN“. Es gibt allerdings auch keine Foreign-Keys und sonstige Datenbank-Constraints. Dies zeigt ein einfaches Beispiel in *Listing 14*.

Wir haben einen neuen Mitarbeiter mit dem Salär 5.000 eingefügt. Nun setzen wir das gleiche „INSERT“ nochmals ab, in diesem Fall aber mit einem Salär von 6.000. In einer Oracle-Datenbank-Tabelle mit „PRIMARY KEY“-Constraint auf „EMPNO“ würden wir eine Constraint-Verletzung erhalten. In Cassandra gibt es jedoch keinen Fehler *(siehe Listing 15)*.

Wir sehen, dass das Salär auf 6.000 geändert worden ist. Was geschieht aber, wenn wir einen „UPDATE“ auf einen „PRIMARY KEY“ machen, den es gar nicht gibt *(siehe Listing 16)*?

Die Row wird eingefügt. Die Felder, die im „UPDATE“ nicht angegeben wurden, sind leer. Ein „INSERT“ und „UPDATE“

führt immer zu einer Merge-Operation, also zu einem Upsert. Cassandra lässt „UPDATE“ und „DELETE“ nur über den „PRIMARY KEY“/„PARTITION KEY“ zu *(siehe Listing 17)*.

Fazit

Cassandra ist eine schöne neue Welt. Die neuen Ansätze sind genial. Die Nutzung bedingt allerdings einen ganz anderen Denk-/Design-Ansatz. In Cassandra müssen die zu unterstützenden Abfragen bekannt sein, bevor man mit der Datenmodellierung starten kann. Die Tabellen werden dann passend dazu angelegt.

Ein Datenmodell in der dritten Normalform ist nicht möglich beziehungsweise nicht sinnvoll. Dies war für den Autor mit seinen zwanzig Jahren Erfahrung in relationaler Datenmodellierung das wohl Schwierigste.

Durch die Denormalisierung werden die Daten verdoppelt, verdreifacht oder mehr. Für jede Abfrage muss eine passende Tabelle gefunden, erstellt und bewirtschaftet werden. Das hat entsprechende Folgen:

- Die Datenmenge steigt
- Es muss sichergestellt sein, dass all diese „Kopien“ synchron sind
- Es gibt keine Ad-hoc-„JOIN“-Abfragen

Bei der Nutzung von Cassandra müssen Einschränkungen in Kauf genommen werden:

- Kein „JOIN“ zwischen Tabellen möglich
- „UPDATE“ nur über den „PARTITION KEY“
- „WHERE“-Klausel nur eingeschränkt verfügbar
- Aggregationen, aber kein „GROUP BY“
- Kein „ORDER BY“ über mehrere Partitionen hinweg

Diese Einschränkungen sind aber durchaus sinnvoll und stellen sicher, dass Cassandra die extreme Skalierbarkeit und Leistungsfähigkeit erreichen kann.

Netflix hat mit einer Oracle-Datenbank angefangen und ist an dessen Grenzen gestoßen. Sie haben den Schritt zu Cassandra gemacht und konnten nur so die Expansion bewältigen. Es muss jedoch kein 2.700-Node-Cluster sein. 3.000 bis 5.000 Operationen pro Sekunde und Core sind allerdings schon eine ganze Menge. Das bedeutet, dass schon ein relativ kleiner Cassandra-Ring riesige Datenmengen verarbeiten kann.

Kosten sind immer ein Thema. Diese sind um Faktoren niedriger als für eine Oracle-Datenbank oder einen SQL-Server. Eigentlich ist Cassandra gratis. Die Software kann von der Apache Webseite heruntergeladen und installiert werden.

Will man Cassandra jedoch nicht ohne Support nutzen, dann empfiehlt sich die Nutzung einer Distribution von Cassandra. DataStax bietet mit DataStax Enterprise ein Paket an, das eine gehärtete Version von Cassandra inklusive Support enthält. DataStax stellt auch einen sehr guten Online-Kurs für Cassandra zur Verfügung *(siehe „https://academy.datastax.com“)*. Dieser ist gratis. Zudem gibt es eine Zertifizierung für Cassandra.

Cassandra gibt dem Anwender Optionen. Er kann klein anfangen und, wenn nötig, wachsen. Er kann es inhouse oder in der Cloud betreiben und dies zu einem interessanten Preis.

Hinweis: Die Listings 11 bis 17 finden Sie unter *www.doag.org/de/go/red_stack/201702/ott/listings*

Jan Ott
jan.ott@trivadis.com

Termine

28.04.2017
DOAG-Leitungskräfteforum 2017
Vereinsinterna und Treffen der Community-Leitungsteams, Berlin

29.04.2017
DOAG Ordentliche Delegiertenversammlung 2017
Berlin

09.05.2017
APEX Connect 2017
Berlin

10.05.2017
Regionaltreffen Würzburg
Oracle Database 12.2 - New Features

12.05.2017
Regionaltreffen Rhein-Neckar
Security: permanent testing, PSU und PatchBundle, Mannheim

17.05.2017
SOUG Training Day „Hochverfügbarkeit“
Olten

18.05.2017
Regionaltreffen Nürnberg/Franken
Oracle Database Read Only Tablespaces 11g vs. 12c, Nürnberg

18.05.2017
SOUG Training Day „Hochverfügbarkeit“
Morges

20.06.2017
AOUG Anwenderkonferenz
Wien

27.06.2017
30 Jahre SOUG - Große Jubiläumsfeier
Zürich

Wir begrüßen unsere neuen Mitglieder

Persönliche Mitglieder

- Sabine Hunsicker
- Thies Wellpott
- Markus Hohloch
- Ozren Pestic
- Francesca Meyer
- Rene Rousseau
- Wolfgang Gruber
- Jochen Geiss
- Florian Umlauf
- Sandro Ferreira
- Michael Ludewig
- Mohamed Taouri
- Sinan Petrus Toma
- Dennis Giese
- Manfred Schmid

Firmenmitglieder DOAG

- targens GmbH, Peter Zweifel
- Bechtle Logistik & Service GmbH, Manuel Cintean

Impressum

Red Stack Magazin wird gemeinsam herausgegeben von den Oracle-Anwendergruppen DOAG Deutsche ORACLE-Anwendergruppe e.V. (Deutschland, Tempelhofer Weg 64, 12347 Berlin, *www.doag.org*), AOUG Austrian Oracle User Group (Österreich, Lassallestraße 7a, 1020 Wien, *www.aoug.at*) und SOUG Swiss Oracle User Group (Schweiz, Dornacherstraße 192, 4053 Basel, *www.soug.ch*).

Red Stack Magazin ist das User-Magazin rund um die Produkte der Oracle Corp., USA, im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Oracle-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Red Stack Magazin wird verlegt von der DOAG Dienstleistungen GmbH,Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

Die DOAG Deutsche ORACLE-Anwendergruppe e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Stefan Kinnen. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH (Anschrift s.o.)
Chefredakteur (ViSdP): Wolfgang Taschner
Kontakt: redaktion@doag.org
Weitere Redakteure (in alphabetischer Reihenfolge): Gaetano Bisaz, Mylène Diacquenod, Marina Fischer, Klaus-Michael Hatzinger, Sebastian Höing, Fried Saacke

Titel, Gestaltung und Satz:
Caroline Sengpiel, DOAG Dienstleistungen GmbH (Anschrift s.o.)

Fotonachweis:
Titel: © Timur Arbaev/123RF
Foto S. 11: © ag visuell/Fotolia
Foto S. 14: © Igor Zhuravlov/123RF
Foto S. 20: © Artem Egorov/123RF
Foto S. 26: © kebox/123RF
Foto S. 30: © Bakhtiar Zein/123RF
Foto S. 36: © Alexander Pokusay/123RF
Foto S. 43: © yuriy2design
Foto S. 50: © Jakub Jirsak/123RF
Foto S. 58: © Scott Betts/123RF
Foto S. 62: © rawpixel/123RF
Foto S. 69: © https://svn.apache.org/repos/asf/cassandra/logo/cassandra.svg

Anzeigen:
Simone Fischer, DOAG Dienstleistungen GmbH (verantwortlich, Anschrift s.o.)
Kontakt: anzeigen@doag.org
Mediadaten und Preise unter:
www.doag.org/go/mediadaten

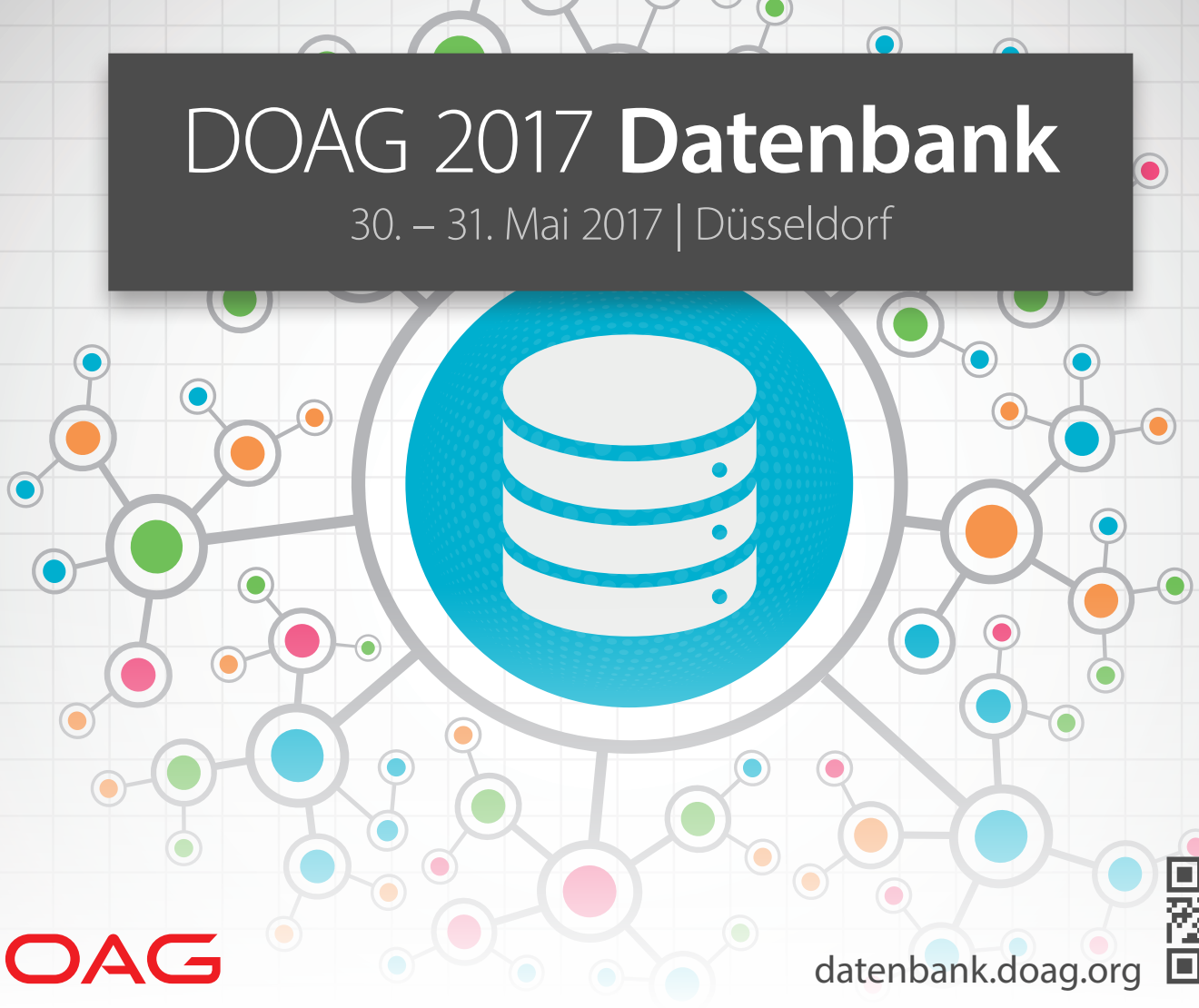
Druck:
adame Advertising and Media GmbH,
www.adame.de

Inserentenverzeichnis

dbi services ag *www.dbi-services.com*	S. 13	MuniQsoft GmbH *www.muniqsoft.de*	S. 3	TÜV Rheinland Akademie GmbH *www.akademie.tuv.com*	S. 7
DOAG e.V. *www.doag.org*	U 3	Oracle *www.oracle.com*	U 2		
E-3 Magazin *www.e-3.de*	S. 39	Trivadis AG *www.trivadis.com*	U 4		

DOAG 2017 **Datenbank**

30. – 31. Mai 2017 | Düsseldorf



DOAG

datenbank.doag.org

DOAG 2017 Exaday

20. Juni 2017 in Frankfurt

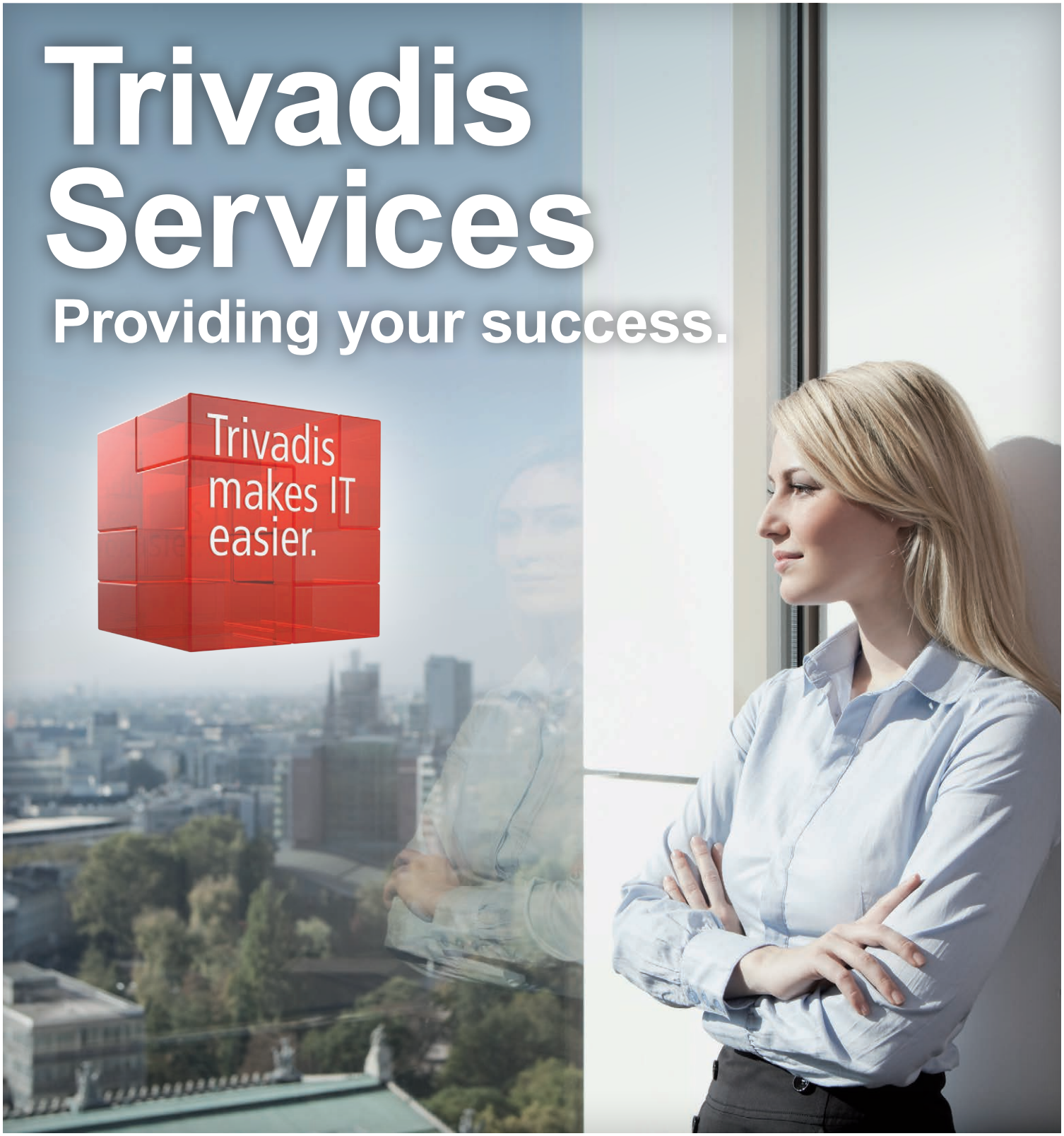
Early Bird bis
8. Mai 2017

DOAG

exaday.doag.org

Trivadis Services

Providing your success.



Gestalten Sie Ihr Leben sorgenfreier. Und Ihre IT leistungsfähiger. Trivadis Services stellt den Lebensnerv heutiger Unternehmen in den Mittelpunkt der Servicebetreuung: die IT-Anwendungen und -Systeme sowie deren Management. Dafür stehen Ihnen verschiedene flexible IT-Service-Modelle zur Verfügung – basierend auf modernem IT-Service-Management (ITIL). Unser Spektrum reicht von standardisierten Managed-Service-Produkten (SystemCare) bis hin zu individuell anpassbaren SLA-Leistungen. Das alles sowohl für den Betrieb bzw. Lifecycle Ihrer Applikationen als auch für Ihre Infrastruktur (Datenbanken, Middleware und Betriebssysteme). Als Service in der Cloud oder On Premise in Ihrem Rechenzentrum. Sprechen Sie mit uns. www.trivadis.com/services | info@trivadis.com

BASEL ▪ BERN ▪ BRUGG ▪ DÜSSELDORF ▪ FRANKFURT A.M. ▪ FREIBURG I.BR. ▪ GENF
HAMBURG ▪ KOPENHAGEN ▪ LAUSANNE ▪ MÜNCHEN ▪ STUTTGART ▪ WIEN ▪ ZÜRICH

trivadis
makes IT easier.