www.ordix.de € 3,20

2|2017

ORDIX® news

einfach. besser. informiert.

Apache Kudu

SIND SIE STARTKLAR FÜR IHRE ZUKUNFT?

WIR GEBEN IHRER ENTWICKLUNG AUFTRIEB

NEUGIERIG?

Als neuem Mitarbeiter bieten wir Ihnen ein speziell auf Sie zugeschnittenes Start- und Entwicklungsprogramm. Die Themen Weiterbildung und Weiterentwicklung sind ein wesentlicher Teil unserer Unternehmensphilosophie.

STUDENTEN

Wir bieten in Kooperation mit unseren Partner-Fachhochschulen duale Studiengänge für die Standorte Paderborn, Wiesbaden, Köln und Gersthofen an.

Zum Beispiel:

Bachelor of Science
in Wirtschaftsinformatik

Im regelmäßigen Wechsel von Theorie und Praxis steht die Konzeption, das Design und die Entwicklung von modernen Anwendungen im Vordergrund.

BERUFSEINSTEIGER

Wir bieten Ihnen die Möglichkeit, sich in unserem Unternehmen zu entwickeln und ermöglichen Ihnen den optimalen Karrierestart.

Zum Beispiel:

Junior Software-Entwickler – Java

An unseren ORDIX Standorten arbeiten Sie mit modernen und innovativen Technologien und unterstützen unsere Experten in der Entwicklung von komplexen Applikationen.

BERUFSERFAHRENE

Sie suchen eine neue berufliche Herausforderung? Auf Sie warten spannende und anspruchsvolle IT-Projekte.

Zum Beispiel:

IT-Projektmanager

Die Planung, Organisation und Steuerung interessanter IT-Projekte bei unseren renommierten Kunden stehen im Fokus Ihres Aufgabenfeldes.

Weitere Informationen und Jobangebote finden Sie auf unseren Karriereseiten im Internet.

www.ordix.de/karriere
www.xing.com/companies/ordixag/jobs

kununu TOP COMPANY www.kununu.com VON MITARBEITERN EMPFOHLEN!

Paderborn, Dezember 2017

Jamaika-Aus

Kaum hatte ich mein Editorial mit der Überschrift Jamaika fertig, geht Lindner vor die Presse und knipst das Licht aus. Nun ist mein neuer, schwarz-weißer Lieblingspolitiker nicht nur für das Platzen von Jamaika verantwortlich, sondern auch dafür, dass es knapp wird mit der ORDIX® news 2/2017 vor Weihnachten in Ihrem Briefkasten.

Und dann wird „Jamaika-Aus“ auch noch das Wort des Jahres. Dabei müsste es jetzt eher eine Wiederholung von 2013 geben, da war nämlich „GroKo“ das Wort des Jahres. Aber auch „Schwarzgeldaffäre“ (2000), „Bundeskanzlerin“ (2005), „Klimakatastrophe“ (2007) oder „Reformstau“ (1997) hätten dieses Jahr mindestens genauso gut gepasst.

Aber spüren Sie eigentlich, dass wir zurzeit keine Regierung haben? Mir fehlt da momentan gar nichts. Am Ende des Jahres oder besser der Legislaturperiode hoffe ich jedoch darauf, dass irgendein Journalist oder der Bund der Steuerzahler die offizielle Rechnung macht, um wie viel höher die Kosten für unsere Bundestagsabgeordnete der vier Jahre 2017-2021 im Vergleich zu der Periode 2013-2017 sind. 709 Abgeordnete das sind fast 18,6 % mehr als die gesetzlich vorgeschriebenen 598, wobei das wegen potentieller Pattsituationen nicht sehr sinnig ist. Aber es gibt ja immer genügend Überhangmandate.

Würden Sie freiwillig 18,6 % mehr Miete zahlen oder die Erhöhung Ihrer Stromrechnung um diesen Prozentsatz in Kauf nehmen? Sicher nicht! Aber hier werden uns Steuerzahlern monatlich gut 60.000 EUR pro MdB mehr abverlangt. Das sind bei zusätzlich 111 Abgeordneten knapp 6,7 Millionen EUR im Monat oder lockere 320 Millionen während der Legislaturperiode. Von zukünftigen Pensionsverpflichtungen will ich gar nicht reden.

Davon kann man einiges in den vier Jahren finanzieren, was mehr Sinn macht, zumal die Kollegen MdB gerade jetzt, wo es noch keine großartigen parlamentarischen Auseinandersetzungen gibt, ein sehr lockeres Leben führen können - und jetzt also GroKo die Dritte. Das wird dann dazu führen, dass 2021 auch die GroKo nicht mehr die nötigen Stimmen haben wird. Oder vielleicht eine KoKo*)? Da kommen einem schon „Weimarer Verhältnisse“ in den Kopf.

Das wollen wir jetzt aber so kurz vor dem Fest des Friedens nicht heraufbeschwören. Dann sollten Sie lieber doch etwas zu und über Daten lesen: Die Fortsetzungen zu den Themen Data Mining und Big Data oder was Neues von IBM zu Db2 bzw. von uns zu Oracle 12.2 - da beginnen zwei neue Artikelreihen. Auch ganz interessant sind unsere Artikel zum Stichwort Automatisierung (Ansible, Continuous Integration, Automatisierte Installation mit SCCM und dem Thema Enterprise Job Scheduler), denn damit erleichtert sich Ihr Leben.

Java 9 ist neu und bietet damit auch den Start in eine neue Artikelreihe. Zu Beginn geht es um Sicherheit, wie auch in unserem Artikel zu PCI-DSS (Payment Card Industry Data Security Standard).

Egal ob jetzt GroKo, KoKo oder vielleicht auch noch Kenia (CDU/SPD/Grüne), bis Weihnachten und Neujahr wird es keine Antwort auf die Frage(n) geben. Genießen Sie einfach das Weihnachtsfest, selbst wenn am 23.12. das Horrorszenario in den Supermärkten abläuft und kommen Sie gut in 2018 an. Ich wünsche Ihnen nur das Beste.

Herzlichst Ihr

Wolfgang Kögler

*) KoKo: Kooperationskoalition

6

Lineare Regression

10

Apache Kudu

Big Data

Automatisierung

Oracle

Microsoft

IBM Datenbanksysteme

Temporales Datenmanagment in IBM Db2

Build-Prozess mal anders

IT-Security

Entwicklung

Aktuell

Impressum

Herausgeber: ORDIX AG Aktiengesellschaft für Softwareentwicklung, Beratung, Schulung und Systemintegration, Paderborn

Redaktion/Layout: Jens Pothmann, Isabell Rosenblatt

V.i.S.d.P.: Christoph Lafeld, Wolfgang Kögler

Anschrift der Redaktion: ORDIX AG | Karl-Schurz-Straße 19a | 33100 Paderborn
Tel.: 0 52 51 10 63 -0 | Fax: 0180 1673490

Auflage: 7.000 Exemplare

Druck: Druckerei Bösmann, Detmold

Bildnachweis: © iStockphoto | flyfloor | Diagrams projection
© pexels.com
© pixabay.com | HansenHimself | Impala
© flickr.com | Jim, the Photographer | Elephant
© istockphoto.com | anankkml | Greater Kudu in the dark ...
© freepik.com | @graphictwister | White wood texture
© pixabay.com | uhr-uhrwerk-zeit-uhrzeit
© pixabay.com | auto-ferrari-karriere-rot

Autoren: Dr. Hubert Austermeier, Dr. Uwe Bechthold, Veronica Eckstedt, Makus Fiegler, Carsten Griese, Frank Heimerzheim, Olaf Hein, Christopher Herclik, Sebastian Herd, Stefan Hummel, Wolfgang Kögler, Phillip Kürsten, Alexander Stondenko

Data Mining in der Praxis (Teil III)

Lineare Regression

Wie wird der Umsatz nächsten Monat, wenn wir 10 % mehr für Werbung ausgeben? Welche Einflussgröße hat die größte Auswirkung auf das Ergebnis? Mithilfe der Regression lassen sich derartige Fragen beantworten. Die lineare Regression erlaubt es, aus vorhandenen Daten Zusammenhänge zu identifizieren und in Form eines Modells für Prognosen nutzbar zu machen. In diesem Artikel erhalten Sie einen Überblick über die theoretischen Grundlagen der Regression sowie Beispiele zur Umsetzung mit Python.

Was ist lineare Regression?

In untersuchten Daten existiert eine Menge an unabhängigen Variablen, die den Wert einer abhängigen Variablen beeinflussen. Zum Beispiel beeinflussen die unabhängigen Variablen Einnahmen und Ausgaben die von ihnen abhängige Variable Gewinn nach dem Prinzip Gewinn = Einnahmen – Ausgaben. Der Zusammenhang im Beispiel ist trivial. Bei der Analyse von nicht trivialen Zusammenhängen mit potenziell hunderten von unabhängigen Variablen kann die lineare Regression behilflich sein. Von Interesse ist hierbei die konkrete Prognose einer abhängigen Zielvariable in Abhängigkeit der unabhängigen Quellvariablen. Weiterhin erlaubt die Regression die Analyse des Zusammenspiels der Quellvariablen. Nicht alle sind gleich gewichtet und die lineare Regression kann helfen, diejenige Einflussgrößen zu identifizieren, welche die größte Auswirkung auf das untersuchte Ziel haben.

Wie funktioniert lineare Regression?

Im einfachsten Fall wird der Zusammenhang zwischen einer einzelnen unabhängigen Variablen x und einer abhängigen Variablen y untersucht. In diesem Fall ergibt sich der Wert der Zielvariablen aus folgender mathematischen Funktion:

$$y = \beta_0 + \beta_1 * x + error$$

β_0 ist hierbei ein von x unabhängiger Wert. β_1 bestimmt die Stärke des Einflusses von x auf die Zielvariable y. Die lineare Regression ist ein statistisches Verfahren und wird somit nie 100 % korrekte Vorhersagen liefern. Der Wert $error$ trägt dieser Tatsache Rechnung und repräsentiert einen statistischen Fehler.

Komplexere Fälle von linearer Regression betrachten nicht nur eine unabhängige Variabel (einfache lineare Regression)

sondern sehr viele unabhängige Variablen (multiple lineare Regression):

$$y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n + error$$

Außerdem kann eine unabhängige Variable nicht nur einfach, sondern zum Beispiel auch quadratisch oder in einer höheren Potenz in die Berechnungsformel der Zielvariablen y eingehen:

$$y = \beta_0 + \beta_1 * x + \beta_2 * x^2 + ... + \beta_n * x^n + error$$

Erstellung eines Modells

Die blauen Punkte in Abbildung 1 stehen für 10 bekannte Ausprägungen der Quellvariablen in Form von 10 Wertepaaren (x, y). Mit Hilfe dieser Information wurde das Modell

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 * x$$

erzeugt. Abbildung 1 zeigt die sich ergebende Gerade mit dem Achsenabschnitt $\hat{\beta}_0$ und der Steigung $\hat{\beta}_1$.

$\hat{\beta}_0$ und $\hat{\beta}_1$ sind Vorhersagewerte die, basierend auf den untersuchten Quelldaten, ermittelt wurden. Da Quelldaten immer einen endlichen Umfang haben, können $\hat{\beta}_0$, $\hat{\beta}_1$ und daraus resultierend auch $\hat{y}$ nie perfekt sein. Die Kunst besteht nun darin, die Parameter $\hat{\beta}_0$ und $\hat{\beta}_1$ so zu bestimmen, dass der über alle zur Verfügung stehenden Quelldaten hinweg summierte Fehler möglichst klein wird.

In Abbildung 2 sind die Einzelfehler als rote Linien dargestellt. Es sind die jeweiligen Abweichungen des von der Regression prognostizierten Wertes auf der grünen Gerade vom tatsächlichen Wert eines blauen Punktes.

Eines der am häufigsten eingesetzten Verfahren zur Minimierung des Gesamtfehlers ist die Minimierung der Summe der quadratischen Fehler. Durch diesen kleinen Trick wird erreicht, dass größere Abstände vom Zielwert, also größere Fehler, stärker in den Gesamtfehler eingehen als kleine Abstände.

Auf die mathematischen Einzelheiten dieser nicht trivialen Aufgabe soll in diesem Artikel nicht näher eingegangen werden. In den Quellen ist ein sehr empfehlenswertes Einstiegswerk in die Materie angeführt [1].

Daten für ein einfaches Beispiel zur einfache linearen Regression

Die British Doctors Study [2] hat über 50 Jahre hinweg unter anderem die Sterblichkeit von Rauchern untersucht. Einige der Ergebnisse sollen im Folgenden in einem Beispiel analysiert werden. Hierzu dienen die Daten aus Tabelle 1.

Angaben wie „über 25“ lassen sich nicht mit einem analytischen Verfahren verwenden, dass Zahlen als Eingabeparameter erwartet. Daher wird im Folgenden mit einem vereinfachten Wert gearbeitet, der den tatsächlichen Wert ersetzt.

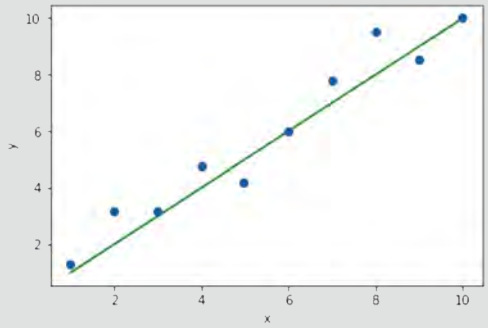

Abb. 1: Beispieldaten und die dazu gehörende Regressionsgerade

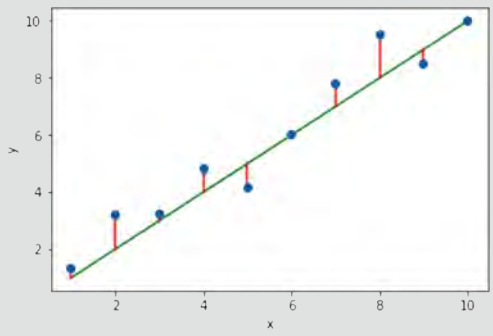

Abb. 2: Statistischer Fehler als Abweichung der tatsächlichen von den vorhergesagten Werten

Zigarettenkonsum (pro Tag)	Zigarettenkonsum (vereinfachter Wert)	Todesfälle durch Lungenkrebs (100 TSD Männer/Jahr)
0	0	17
1-14	10	131
15-24	20	233
über 25	30	417

Tabelle 1: Auswertungen aus der British Doctors Study [2]

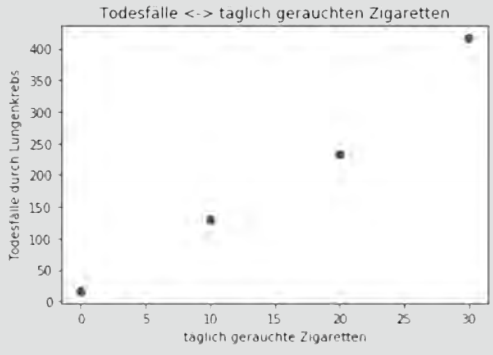

Abb. 3: Scatterplot von Daten zur Sterblichkeit von Rauchern

Beispielanalyse mit Python

Wie bereits im zweiten Artikel der Reihe Data Mining [3] wird auch für dieses Beispiel Python mit anaconda [4] und dem Jupyter Notebook [5] eingesetzt.

Es ist ratsam sich vor dem eigentlichen Beginn einer Analyse einen Überblick über die Daten zu verschaffen. Der in Abbildung 3 dargestellte Scatterplot der Daten bildet hierzu eine gute Grundlage. Der Plot wurde mit dem Code aus Abbildung 4 erzeugt. Nach notwendigen Imports werden die Daten aus Tabelle 1 in ein Pandas DataFrame geladen und mit der Bibliothek **`matplotlib`** ausgegeben.

```
%matplotlib inline
import matplotlib.pyplot asplt
import pandas as pd

df = pd.DataFrame ({'zigaretten':[0,10,20,30],
                    'lungenkrebs': [17,131,233,417]})

df.plot('zigaretten', 'lungenkrebs', kind='scatter')
plt.title('Todesfälle <-> täglich gerauchte Zigaretten')
plt.xlabel('täglich gerauchte Zigaretten')
plt.ylabel('Todesfälle durch Lungenkrebs')
plt.plot()
```

Abb. 4: Datenerzeugung und Ausgabe als Scatterplot

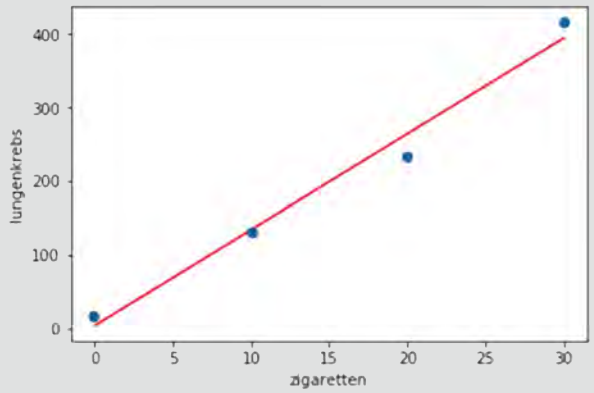

Abb. 5: Daten zu Lungenkrebs bei Rauchern mit dazugehörender Regressionsgerade

```
import numpy as np
from scipy import stats

df.plot ('zigaretten', 'lugenkrebs', kind='scatter')

beta_1, beta_0, r_value, p_value, std_err = stats.
linregress(df.zigaretten, df.lungenkrebs)

X = np.linspace(df.zigaretten.min(), df.zigaretten.max(), 100)
Y = beta_0 + beta1 * X

plt.plot(X; Y; color='red')
plt.show()
```

Abb. 6: Modellermittlung und Plot der Ergebnisse

Bei so wenigen Daten lässt sich schon durch reines Hinschauen ein Zusammenhang zwischen der Zahl der täglich gerauchten Zigaretten und der Sterblichkeit durch Lungenkrebs erkennen.

In Abbildung 5 wird dieser Zusammenhang belegt nachdem das Ergebnis einer Regressionsanalyse (zu sehen in dem Code der Abbildung 6) mit in den Scatterplot eingefügt wurde.

In diesem Beispiel wird die Methode **`stats.linregress`** aus dem Paket SciPy [6] verwendet. In vielen anderen Bibliotheken existieren vergleichbare Methoden.

Der Methode **`stats.linregress`** werden mit dem Befehl **`df.zigaretten`** und **`df.lungenkrebs`** die zu analysierenden Daten übergeben. Als Ergebnis besonders interessant sind die beiden Parameter **`beta_0`** und **`beta_1`**. Diese werden verwendet, um zu 100 x-Werten zwischen dem minimalen und maximalen Zigarettenkonsum entsprechende y-Werte zu berechnen. Die Zahl von 100 ist willkürlich gewählt, um einen möglichst glatten Plot zu erreichen. Relevant wird diese Zahl beim Plot von Ergebnissen einer multiplen linearen Regression, die nicht durch eine einfache Gerade gezeichnet werden kann.

Von den weiteren Rückgabewerten der Methode **`stats.linregress`** ist insbesondere der **`r_value`** von Interesse. Quadriert wird er zu r^2 und ist eine in der Statistik weit verbreitete Maßzahl für die Qualität eines Modells. Ein Wert von 1 steht hierbei für ein perfektes Ergebnis ohne existierenden statistischen Fehler. Umso kleiner der Wert wird, desto schlechter ist das Modell. Der absolute Wert von 0,98 bescheinigt unserem Modell eine gute Qualität. Der Blick auf die Gerade aus Abbildung 5 bestätigt dies, da sie die Datenpunkte gut repräsentiert.

Interessanter als der absolute Wert ist jedoch der Vergleich der r^2-Werte von verschiedenen in Frage kommenden Modellen. Die Maßzahl kann in einem solchen Fall genutzt werden, um gute von schlechten Modellen zu unterscheiden.

Ausblick multiple lineare Regression

Im Code-Beispiel wurde nur eine Quellvariable betrachtet. In der Praxis gibt es natürlich sehr viele verschiedene Einflussfaktoren auf eine Zielvariable. Auch wenn die zu Grunde liegende Mathematik etwas komplexer wird, unterscheiden sich uni- und multiple lineare Regression von den Verfahren her nicht wesentlich.

Interessant ist vielmehr ein ganz neuer Aspekt, den es zu beachten gibt. Und zwar gilt es aus der häufig sehr großen Menge an Quellvariablen diejenigen zu identifizieren, die den größten Einfluss auf die Zielvariable haben. Ein gutes Modell kann eine Zielvariable mit möglichst wenigen, dafür tatsächlich relevanten, Quellvariablen vorhersagen.

Um ein gutes Modell zu finden, müssen in der Praxis viele unterschiedliche Modelle betrachtet werden. Die mögliche

Zahl der verschiedenen Modelle steigt exponenziell mit der Anzahl der Quellvariablen. Bei der Modellauswahl helfen mathematische Verfahren, aber zu einem großen Teil auch die Erfahrung des Menschen, der die Analysen vornimmt.

Einige warnenden Worte zum Ende

Vorsicht bei Extrapolation. Im ersten Beispiel wurden Werte von x zwischen 1 und 10 zur Erzeugung des Modells verwendet. Nur in diesem Bereich kann das Modell sinnvolle Vorhersagen treffen, da es auch nur in diesem Wertebereich trainiert wurde.

Vorsicht vor Ausreißern. Eine Tageshöchsttemperatur von 1.000° C ist für einen Menschen als klarer Messfehler erkennbar und sollte somit bei der Modellermittlung nicht beachtet werden. Gerade bei wenigen verfügbaren Trainingswerten kann ein solcher Ausreißer einen deutlichen Einfluss auf ein Modell haben und es im Zweifel sogar gänzlich unbrauchbar machen. Es gilt also geeignete Verfahren zu verwenden, die Ausreißer zu erkennen und diese gegebenenfalls zu entfernen.

Fazit

Mithilfe der linearern Regression lassen sich aus vorhandenen Daten Modelle erstellen, die in der Lage sind für neue Datensätze Vorhersagen zu treffen. Die eingesetzten Algorithmen sind mathematisch recht komplex, verdichten sich aber mit geeigneten Python-Bibliotheken auf wenige Zeilen Code.

Um gute Ergebnisse zu erzielen, müssen in der Praxis sehr viele mögliche Modelle miteinander verglichen werden und Ausreißer zuverlässig als solche erkannt werden. Hierzu ist spezielles Wissen erforderlich, welches aber nur im engen Zusammenspiel mit eingehender Kenntnis der Daten seine vollen Möglichkeiten ausspielt. Aus dieser Sicht macht es Sinn, bereits früh in einem Projekt Fachabteilung und Experten aus dem Bereich Data Mining an einen Tisch zu bringen.

Links/Quellen

[1] Einführung in das statistische Lernen
http://www-bcf.usc.edu/~gareth/ISL/

[2] British Doctors Study zur Sterblichkeit bei Rauchern
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC437139/

[3] ORDIX® news Artikel 1/2017 –
„Data Mining in der Praxis (Teil II) - Klassifikation“:
https://ordix.de/ordix-news-archiv/1-2017.html

[4] Anaconda Cloud
https://anaconda.org/

[5] Project Jupyter
http://jupyter.org/

[6] Tools für Python
https://www.scipy.org/

Bildnachweis

Frank Heimerzheim
(info@ordix.de)

Big Data – Informationen neu gelebt (Teil VII)

Apache Kudu

Im September 2016 wurde die Version 1.0 von Apache Kudu veröffentlich. Seit Januar 2017 ist Kudu vollständig in die Hadoop-Distribution von Cloudera integriert und für den produktiven Einsatz freigegeben. Somit gibt es jetzt eine weitere Datenbank im Hadoop Zoo. Aber warum wird überhaupt eine weitere Datenbank benötigt? Was ist das Besondere an Kudu? Welche Probleme können damit gelöst werden? Diese und weitere Fragen werden in diesem Artikel beantwortet.

Warum wird eine weitere Datenbank benötigt?

Im Umfeld von Hadoop gab es bisher zwei gängige Arten für die Speicherung strukturierter Daten. Zum einen als Datei im HDFS und zum anderen in der NoSQL-Datenbank HBase. HDFS ist sehr gut für die Speicherung und Verarbeitung riesiger Datenmengen geeignet. Allerdings ist es in HDFS nicht möglich, effizient einzelne Datensätze zu lesen (wahlfreier Zugriff) und einzufügen. Weiterhin können einmal geschriebene Datensätze nicht wieder verändert oder gelöscht werden. Das ganze System ist für die Verarbeitung großer Dateien im Batch-Betrieb optimiert. HBase auf der anderen Seite ist im Kern eine Key-Value-Datenbank. Das System speichert seine Daten sortiert nach dem Primary Key ab. Einzelne Datensätze können in HBase schnell eingefügt, geändert, selektiert und gelöscht werden. Range-Scans von mehreren tausend Datensätzen (Short Scans) sind auch sehr gut möglich. Sollen aber Millionen oder gar Milliarden von Datensätzen analysiert werden (Large Scans), dann ist HBase häufig zu langsam.

Mit HDFS und HBase gibt es somit zwei sehr gegensätzliche Systeme, die für sehr unterschiedliche Anwendungsfälle und Zugriffsmuster optimiert wurden. In vielen Fällen wird aber weder das eine noch das andere Extrem benötigt. Stattdessen werden häufig sowohl die Batch-Fähigkeiten von HDFS, als auch die Realtime-Fähigkeiten von HBase benötigt. Bisher wurden solche Systeme durch den Einsatz beider Technologien realisiert. Das führt allerdings oft zu relativ komplexen Architekturen der Anwendungen und zu redundanter Datenhaltung. Wie in Abbildung 1 dargestellt gibt es hier eine Lücke, die mit Kudu geschlossen werden kann. Kudu wurde als universell einsetzbares Storage-System konzipiert. Mit dem Einsatz von Kudu lassen sich die

Architekturen vieler Anwendungen deutlich vereinfachen, sodass die Daten nur noch in einem System gespeichert werden müssen.

Was ist Kudu?

Kudu ist ein Top Level Apache-Projekt. Die Entwicklung wird aktuell hauptsächlich von Cloudera vorangetrieben. Die Version 1.0 wurde im September 2016 veröffentlicht. Im Januar 2017 wurde die Version 1.2 vollständig in die Cloudera Enterprise Plattform integriert und für den produktiven Einsatz freigegeben. Zurzeit ist das Projekt sehr aktiv und ca. alle drei Monate wird eine neue Version freigegeben.

Die wichtigsten Eigenschaften von Kudu sind:

- Speicherung der Daten in Tabellen
- Tabellen haben ein vordefiniertes Schema
- Jeder Datensatz wird durch seinen eindeutigen Primary Key identifiziert
- Datensätze werden sortiert nach dem Primary Key gespeichert
- Verteilung der Daten im Cluster durch Partitionierung
- Hochverfügbare Architektur durch Replikation der Daten und Metadaten
- Niedrige Latenz beim wahlfreien Zugriff
- Hohe Geschwindigkeit beim sequentiellen Zugriff
- CRUD-Operationen werden vollständig unterstützt
- Datenbanktreiber für C++, Java und Python
- Integration in Spark, MapReduce und Impala
- Transportverschlüsselung mit TLS
- Authentifizierung mit Kerberos

Welches Datenmodell wird verwendet?

Das Datenbankschema von Kudu ähnelt sehr stark dem Schema einer relationalen Datenbank. Daten werden in Tabellen gespeichert, die wiederum aus Spalten mit vorgegebenen Datentypen und einem Primary Key bestehen. Bereits beim Einfügen findet eine strenge Typprüfung der Daten statt (Schema on write). Als Datentypen werden aktuell die wichtigsten elementaren Typen unterstützt (siehe Abbildung 2). Datentypen für Dezimalzahlen, Maps oder Arrays fehlen zurzeit.

Die Datenbank arbeitet spaltenorientiert. Die einzelnen Spalten werden physikalisch in getrennten Dateien abgelegt. Dabei kann für jede Spalte einzeln definiert werden, wie die Daten kodiert und komprimiert werden.

Jede Tabelle hat einen Primary Key. Dieser kann aus einer oder aus mehreren Spalten bestehen. Die Daten einer Tabelle werden nach dem Primary Key sortiert abgespeichert. Ein weiterer Index kann nicht angelegt werden. Die Modellierung des Primary Keys ist, wie auch bei HBase, entscheidend für die spätere Scan Performance.

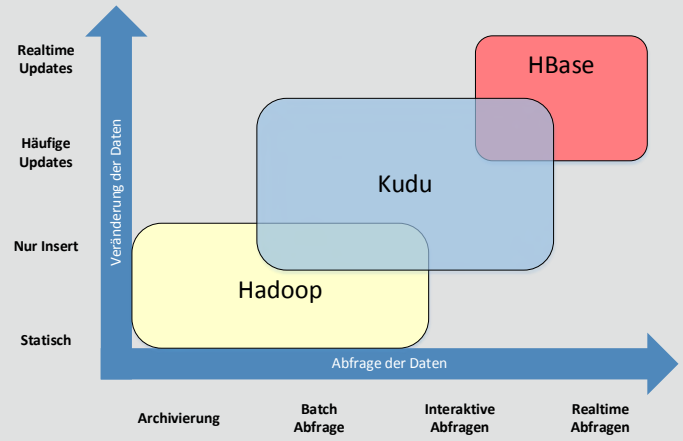

Abb. 1: Hadoop-Storage-Systeme für strukturierte Daten

Datentyp	Beschreibung
INT8, INT16, INT32, INT64	Nummerische Datentypen für ganze Zahlen
FLOAT, DOUBLE	Nummerische Datentypen für Fließkommazahlen
BOOL	Boolscher Datentype
STRING	UTF-8 kodierte Zeichenketten
BINARY	Binäre Daten mit max. 64k je Feld
UNIX_MICROS	Unix Zeitstempel in Microsekunden

Abb. 2: Datentypen

Bei einer Leseanfrage können als Selektionskriterium beliebige Attribute angegeben werden. Für eine optimale Performance müssen allerdings die Attribute des Primary Key in der Reihenfolge verwendet werden, in der sie auch bei der Definition des Schemas verwendet wurden. Dann kann die Datenbank sehr schnell zum ersten Datensatz springen und die Datensätze sequentiell lesen.

Können Tabellen partitioniert werden?

Um die Skalierbarkeit und die Performance zu erhöhen, können Tabellen partitioniert und dadurch in mehrere sogenannte Tablets aufgeteilt werden. Diese Tablets entsprechen Partitionen in anderen Datenbanksystemen. Eine nicht partitionierte Tabelle besteht aus genau einem Tablet.

Bei partitionierten Tabellen entspricht jede Partition einem Tablet. Ein Datensatz gehört immer zu genau einem Tablet. Durch die Partitionierung werden die Daten im

Cluster verteilt. Die folgenden Partitionierungsarten können verwendet werden:

- Hash
- Range

Bei der Hash-Partitionierung werden die Spalten, nach denen partitioniert wird, und die Anzahl der Partitionen (Buckets) definiert. Die Daten werden anschließend anhand eines Hash-Wertes auf die Tablets verteilt. Die Partitionierung muss beim Anlegen der Tabelle durchgeführt werden und kann nachträglich nicht verändert werden. Typische Attribute für eine Hash-Partitionierung sind zum Beispiel Kundennummer oder Kontonummer.

Um beispielsweise eine Tabelle mit Umsatzdaten einer Bank zu partitionieren, bietet sich die IBAN des Kontos als Partitionierungsschlüssel an. Durch diese Partitionierung wird sichergestellt, dass aktuelle Umsätze beim Einfügen auf viele Partitionen verteilt werden. Dadurch werden

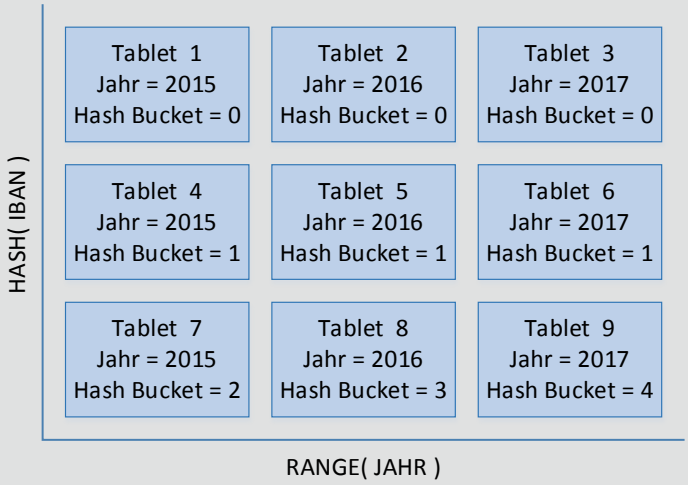

Abb. 3: Partitionierung

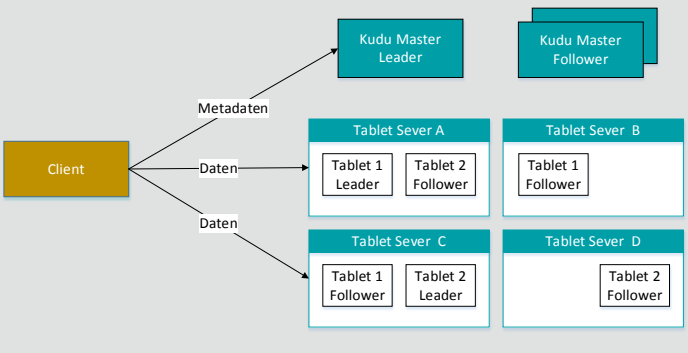

Abb 4: Architektur

„hot-blocks“ vermieden und es können sehr viele Daten sehr schnell eingefügt werden. Weiterhin können die Umsätze eines einzelnen Kunden schnell gelesen werden, da alle Daten für eine IBAN immer auf einem Knoten im Cluster gespeichert werden und nicht umständlich zusammengesucht werden müssen. Allerdings ist der Zugriff auf die Umsätze aller Kunden in einem bestimmten Zeitraum nicht sehr effizient. Weiterhin wird das System über die Zeit nicht gut skalieren, da die Anzahl der Hash-Partitionen nicht erhöht werden kann.

Für die Range-Partitionierung werden die Daten anhand eines Wertebereichs auf die Tablets verteilt. Im Gegensatz zur Hash-Partitionierung, können einzelne Partitionen nachträglich erzeugt und auch wieder gelöscht werden. Oft wird die Range-Partitionierung anhand der Zeit durchgeführt. Für das Beispiel der Umsatzdaten bietet sich das Jahr der Buchung für die Range-Partitionierung an. Wird zusätzlich zum Jahr das vollständige Buchungsdatum als Teil des Primary Key definiert, sind alle Datensätze innerhalb einer Partition nach dem Datum sortiert. Dadurch können die Umsätze aller Kunden in einem definierten Zeitraum sehr schnell selektiert werden. Allerdings wird das Einfügen neuer Umsätze langsamer, da alle neuen Daten in dieselbe Partition geschrieben werden müssen. Weiterhin können die Umsätze eines einzelnen Kunden nicht mehr effizient gelesen werden, da sie in der Regel über mehrere Partitionen verteilt sind.

Die beiden Partitionierungsarten dürfen auch kombiniert werden. Eine oder mehrere Hash-Partitionierungen können zusammen mit einer Range-Partitionierung verwendet werden. Insbesondere bei der Speicherung von Time-Series-Daten wird sehr oft die Hash-Partitionierung zusammen mit der Range-Partitionierung genutzt. Bei den Umsatzdaten ist zum Beispiel die Hash-Partitionierung über die IBAN und die Range-Partitionierung über das Jahr des Buchungsdatums eine sinnvolle Partitionierung. In Abbildung 3 wird das nochmals verdeutlicht.

Die Kombination von Hash- und Range-Partitionierung sorgt dafür, dass die aktuellen Daten auf mehre Tablets verteilt werden. Insbesondere beim Einfügen von Time-Series-Daten kann dadurch verhindert werden, dass alle Daten in das gleiche Tablet eingefügt werden und dadurch Performance-Probleme verursachen. Durch die Range-Partitionierung nach dem Jahr wird sichergestellt, dass das System auf Dauer skaliert und nicht im Lauf der Jahre langsamer wird.

Wie sieht die Architektur aus?

Kudu besteht aus zwei Arten von Server-Prozessen. In Abbildung 4 wird der grundsätzliche Aufbau dargestellt:

- Kudu Master
- Kudu Tablet Server

Der Kudu Master verwaltet die Metadaten des Systems. Das sind unter anderem Informationen zu den Tabellen,

Tablets und Tablet Servern. Weiterhin koordiniert der Master alle Metadaten Operationen wie zum Beispiel das Anlegen, Ändern und Löschen von Tabellen. Es gibt immer genau einen aktiven Master. Dieser wird „Leader“ genannt. In einfachen Umgebungen reicht ein Master Server häufig aus. Soll die Verfügbarkeit des Systems erhöht werden, dann können mehrere Master auf unterschiedlichen Hosts gestartet werden. Die weiteren Master werden in diesem Fall „Follower“ genannt. Beim Ausfall des Leaders bestimmen die verbleibenden Master automatisch einen neuen Leader, sodass das System weiter verfügbar ist. Um einen Leader zu bestimmen, arbeitet das System intern mit dem „Raft Consensus Algorithmus“ [Q3]. Bei diesem Algorithmus muss immer die Mehrzahl der Server verfügbar sein. Für eine Erhöhung der Verfügbarkeit wird eine ungerade Anzahl an Servern benötigt. In hochverfügbaren Umgebungen sind drei Master Server üblich.

Die Kudu Tablet Server speichern die eigentlichen Daten. Wie auch im HDFS, werden die Daten auf lokalen Festplatten des Server und nicht in einem hochverfügbaren Storage System abgelegt. Dadurch können sehr große Datenmengen relativ günstig gespeichert werden. Um sich vor Datenverlust zu schützen werden die Daten in der Regel repliziert. Ein Replikationsfaktor von 3 oder 5 ist üblich. Die Tablet Server verwenden ebenfalls den Raft Consensus Algorithmus, um einen Leader zu bestimmen. Dabei wird für jedes einzelne Tablet (d.h. für jede Partition einer Tabelle) der Leader ausgehandelt. Daten können nur über den Leader verändert werden. Leseanfragen können dagegen von allen Tablet Servern beantwortet werden.

In der Praxis werden die Master- und Tablet-Server-Prozesse auf unterschiedlichen Knoten im Cluster installiert. Client-Applikationen verbinden sich zum einen mit den Master Servern, um Metadaten auszutauschen. Dazu muss der Client alle Master Server kennen und es liegt in der Verantwortung des Client, sich mit dem aktuell aktiven Leader zu verbinden. Der Master teilt dem Client mit, auf welchen Tablets und somit auf welchen Tablet Servern welche Daten gespeichert werden. Für das Lesen und Schreiben von Daten verbindet sich der Client anschließend direkt mit den einzelnen Tablet Servern. Der Datenaustausch findet immer direkt zwischen dem Client und den Tablet Servern statt.

Wie kann Kudu integriert werden?

Für die Integration von Kudu in ein bestehendes Hadoop-System oder aber in eine neue Anwendung stehen verschiedene Schnittstellen und Treiber bereit.

Datenbanktreiber gibt es für die folgenden Programmiersprachen:

- C++
- Python
- Java

Kudu selbst wird in C++ implementiert. Der C++-Treiber und dessen Dokumentation sind sehr gut gepflegt. Der Python-Treiber stellt ein Interface für den C++-Treiber zur Verfügung. Beim Einsatz von Python muss zusätzlich der C++-Treiber installiert werden. Im Gegensatz dazu ist der Java-Treiber vollständig in Java implementiert und hat keine weiteren Abhängigkeiten. Der C++-Treiber wird in diesem Fall nicht benötigt. Bei der Programmierung mit Java und Python kann es sinnvoll sein, einen Blick in die C++-Dokumentation zu werfen, da diese in einigen Fällen etwas detaillierter ist.

Weiterhin ist Kudu sehr gut in die folgenden Komponenten des Hadoop-Ökosystems integriert:

- Impala
- Spark
- MapReduce

Kudu selbst hat keine Shell, um auf die Daten in der Datenbank zuzugreifen. Allerdings ist Kudu sehr gut in Impala integriert. Über Impala können SQL-Statements direkt gegen Kudu-Tabellen abgesetzt werden. Weiterhin können DDL-Statements zum Anlegen, Ändern und Löschen von Tabellen und Partitionen mit Impala ausgeführt werden.

Die Spark-Integration erfolgt über einen speziellen Kudu-Treiber. Das strenge Kudu-Tabellenschema ist für die Verarbeitung mit Spark und insbesondere mit Spark SQL sehr gut geeignet. Neben den üblichen CRUD-Operationen können mit Spark SQL auch DDL-Operationen ausgeführt werden.

Auch wenn MapReduce häufig durch Spark-Jobs ersetzt wird, so hat es für einige Randfälle immer noch seine Daseinsberechtigung. Der Kudu-Java-Treiber enthält alle notwendigen Klassen, um mit MapReduce Jobs auf Kudu-Daten zugreifen zu können.

Gibt es auch Lücken und Schwächen?

Kudu ist ein sehr junges System. Somit hat es noch nicht alle Features, die man von ausgereifteren Systemen, wie zum Beispiel HBase kennt. Erst mit der Version 1.3 wurden elementare Security Features integriert. Diese decken die wesentlichen Anforderungen wie zum Beispiel TLS-Verschlüsselung und Authentifizierung über Kerberos ab. An weiteren Features wie zum Beispiel der Zugriffskontrolle für einzelne Tabellen wird noch gearbeitet.

Eine weitere offene Baustelle ist Backup und Recovery. Durch die Replikation der Daten ist das System vor Datenverlust durch den Defekt einzelner Server sehr gut abgesichert. Vor dem Verlust von Daten durch den Ausfall eines ganzen Rechenzentrums oder durch das Löschen von Daten innerhalb der Datenbank gibt es zurzeit keinen hinreichenden Schutz. Der regelmäßige Export der Kudu-Tabellen in HDFS-Dateien ist ein üblicherweise angewendeter Workaround für dieses Problem. Das kann zum

Glossar

ACID
ACID steht für die folgenden Eigenschaften eines Datenbanksystems: Atomar (Atomicity), Konsistenz (Consistency), Abgegrenzt (Isolation) und Dauerhaft (Durability).

Cloudera
Cloudera ist ein Hadoop Distributor. Neben einer kostenlosen Version bietet Cloudera auch Support und zusätzliche Funktionalitäten als Teil von kostenpflichtigen Distributionen an.

CRUD
Das Akronym CRUD umfasst die elementaren Datenbankoperationen Create, Retrieve, Update und Delete.

HBase
NoSQL-Datenbank, die ihre Daten im HDFS speichert.

HDFS
Das Hadoop Distributed File System ist ein verteiltes Dateisystem zur Speicherung sehr großer Datenmengen.

Impala
System zur Ausführung von SQL-Statements im Hadoop-Ökosystem. Unterstützt werden zum Beispiel HDFS, HBase und Kudu.

NoSQL
Steht für „Not only SQL“ und bezeichnet Datenbanken, die nicht relational sind.

Raft Consensus Algorithmen
Verfahren, um in einem verteilten System die Konsistenz und Verfügbarkeit zu garantieren.

Schema on write
Verfahren im Datenbankumfeld, bei dem vor dem Schreiben geprüft wird, ob die Daten zum vordefinierten Schema der Tabelle passen.

Spark
Framework für die verteilte Verarbeitung von Daten in einem Cluster. Vergleichbar mit MapReduce, aber in vielen Fällen deutlich schneller.

Links/Quellen

[1] Big Data Seminar der ORDIX AG
https://seminare.ordix.de/seminare/big-data-und-data-warehouse.html

[2] ORDIX® news Artikel 02/2013 –
„NoSQL vs. SQL (Teil IV): HBase“
https://www.ordix.de/ordix-news-archiv/1-2013.html

[3] ORDIX® news Artikel 3/2015 –
„Big Data: Informationen neu gelebt (Teil III) - Apache Hadoop“
https://www.ordix.de/ordix-news-archiv/3-2015.html

[Q1] Apache Kudu Homepage
http://kudu.apache.org/

[Q2] Kudu-Dokumentation von Cloudera
https://www.cloudera.com/documentation/kudu/latest.html

[Q3] The Raft Consensus Algorithm
https://raft.github.io/

[Q4] GitHub-Seite des Kudu-Projektes
https://github.com/apache/kudu

[Q5] Jira-Seite des Kudu-Projektes
https://issues.apache.org/jira/browse/KUDU

Beispiel durch ein einfaches **`INSERT INTO ... SELECT ...`** Statement in Impala umgesetzt werden.

Das Projekt Apache Kudu und auch das Projekt Apache Impala werden massiv von Cloudera vorangetrieben. Entsprechend gut ist auch die Integration in die Hadoop-Distribution von Cloudera. In den anderen großen Distributionen von Hortonworks und MapR wird Kudu aktuell nicht unterstützt. Somit muss beim Einsatz von Kudu in einer dieser Distributionen mit zusätzlichem Installations- und Integrationsaufwand gerechnet werden. Weiterhin fehlen in diesem Fall auch komfortable Werkzeuge für die Überwachung und Konfiguration des Clusters.

Fazit

Kudu ist eine sehr interessante Datenbank. Insbesondere, wenn die Vielseitigkeit wichtiger ist als die Spezialisierung auf einige wenige Anwendungsfälle, kann Kudu seine Vorteile ausspielen. In der Praxis wird in vielen Fällen zusätzlich ein HDFS eingesetzt. Dieses speichert dann zusätzliche Daten oder dient als Backup System. Ein weiterer großer Vorteil von Kudu ist die sehr gute SQL-Unterstützung durch Impala. Kudu ist noch sehr jung und verfügt in einigen Bereichen noch nicht über alle üblichen Features einer ausgereiften Datenbank. Insbesondere gibt es noch Lücken rund um Backup & Recovery und Security. Zurzeit ist das Projekt sehr aktiv und es ist damit zu rechnen, dass viele dieser Lücken im Lauf der Zeit geschlossen werden. Auf den Github [Q4] und Jira [Q5] Seiten des Projektes, kann sich jeder Interessierte über den Status der aktuellen Entwicklung informieren.

Olaf Hein
(info@ordix.de)

Bildnachweise

Open Source Software Ansible

Continous Ansible mit Jenkins

Die Open Source Software Ansible bietet Konfigurationsmanagement, Orchestrierung und Provisionierung für unterschiedliche Plattformen. Mithilfe von Jenkins können Builds, Deployments und Automatisierungen mittels einer Open Source Software realisiert werden. Damit bietet sich Jenkins auch für die Automatisierung von Ansible Playbooks an. So erhält der Nutzer eine Weboberfläche, die den Zustand aller konfigurierten Ansible Playbooks oder ad-hoc-Kommandos grafisch darstellt und deren Kern die Operationen auch automatisiert ausführen kann.

Die Werkzeuge

Ansible [1] ist eine Open Source Software, lizensiert unter der GPL v3. Mit Ansible können Systeme konfiguriert und orchestriert werden. Provisionierung und sogenannte „Zero-Downtime"-Deployments sind ebenfalls möglich. Mittlerweile ist Ansible ein Teil von Red Hat, welches die dahinter stehende Firma AnsibleWorks 2015 nur 2 Jahre nach deren Gründung 2013 [2] übernommen hat. Als Zielsysteme werden Linux/Unix, Solaris, Windows und MacOS unterstützt.

Jenkins [3] ist ein Fork von Hudson, welches ursprünglich bei SUN Microsystems entwickelt wurde. Nach der Übernahme durch Oracle wurde mit Jenkins die offene Weiterentwicklung fortgeführt. Jenkins ist in Java implementiert und bietet ein durch Plugins erweiterbares webbasiertes System für Continous Integration, Continous Delivery oder automatisiertes Testen. Durch die Erweiterbarkeit bietet Jenkins viele Möglichkeiten, nahezu beliebige Aufgaben zu automatisieren, beispielsweise Softwarebuilds mit **`ant`** oder **`make`**. Die dazu notwendigen Quellen können aus bestehenden SCMs wie Git, SVN etc. bezogen werden. Sämtliche Aufgaben können zeitlich gesteuert oder von extern ausgelöst werden. Zusätzlich existieren nach der Beendigung der Aufgabe weitere Optionen zur Verarbeitung oder lediglich Benachrichtigungen der Anwender.

Nun liegt es nahe, diese beiden Werkzeuge eine Beziehung eingehen zu lassen. So besteht die Möglichkeit, die Ansible-Konfigurationen in einem Git-Repository zu hinterlegen. Bei jeder Änderung im Repository sollen diese dann auf die Zielsysteme angewandt werden. Dies kann später auch noch mit dem anschließenden Deployment auf Produktivsysteme gekoppelt werden, wenn das Deployment und diverse Funktionstests in der Testlandschaft erfolgreich sind. Dieser zweite Zusatz ist allerdings nicht Bestandteil dieses Artikels, stellt aber eine Möglichkeit zur weiteren Verwendung der Werkzeuge dar.

Ansible – ein kurzer Überblick

Ansible verarbeitet wahlweise einzelne Kommandos oder komplexe Abläufe, die in sogenannten Playbooks definiert werden. Die anzusteuernden Zielsysteme werden in Inventories konfiguriert. Dabei handelt es sich um einfache Textdateien im Stil einer **`INI`**-Datei (siehe Abbildung 1). Ansible verbindet sich per SSH mit allen Zielsystemen. Dabei ist es möglich, unterschiedliche User Accounts auf verschiedenen Systemen in einem Lauf zu verwenden. Zusätzlich stehen unterschiedliche Optionen zur Verfügung, um sich auf dem Zielsystem weiterführende Rechte (üblicherweise per **`sudo`**) zu verschaffen. Folgendes Kommando führt den Befehl **`uptime`** auf allen Hosts aus:

```
ansible –i hosts all –m command –a uptime
```

Die notwendigen Hosts werden aus dem Inventory **`hosts`** gelesen. Statt dem Bezeichner **`all`** kann hier auch eine Gruppe genannt werden, beispielsweise **`ntp-clients`** oder **`webserver`**.

```
[ntp-clients]
192.168.33.100
192.168.33.101

[webserver]
192.168.33.100
```

Abb. 1: Ein Inventory für Ansible

```
192.168.33.101 | SUCCESS | rc=0 >>
 09:55:38 up 9 min, 1 user, load average: 0.00, 0.04, 0.05

192.168.33.100 | SUCCESS | rc=0 >>
 09:55:37 up 10 min, 1 user, load average: 0.08, 0.07, 0.06
```

Abb. 2: Resultat des ad-hoc-Kommandos `uptime`

```
hosts: webserver
become: true
tasks:
    - name: install httpd
      package: name=httpd state=latest
    - name: install template
      template: src=index.html.j2 dest=/var/www/html/index.html
    - name: start and activate httpd service
      service: name=httpd state=started enabled=yes
```

Abb. 3: webserver.yml – ein Ansible Playbook

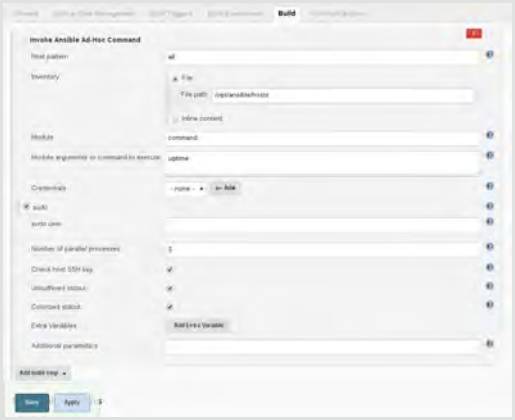

Abb. 4: Ansible ad-hoc „uptime“ in Jenkins

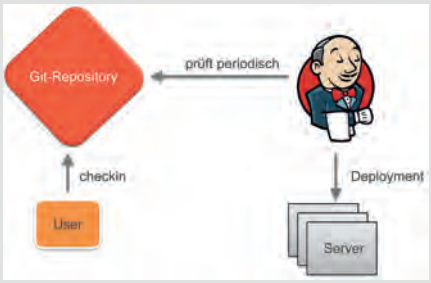

Abb. 5: Ablauf des Playbook webserver.yml mit Git und Jenkins

Abb. 6: Konfiguration von Git sowie der Intervalle unter Jenkins

Das von Ansible verwendete Modul ist **`command`**, das Argument für das Modul lautet **`uptime`**. Das Resultat der Verbindung zeigt Abbildung 2.

Durch die Playbooks besteht die Möglichkeit, mehrere Tasks aneinanderzureihen und in Abhängigkeit zu vorangegangenen Ergebnissen oder Optionen auf dem Zielsystem auszuführen. Solche Playbooks werden in YAML-Dateien gespeichert, eine Ausführung wird kurz als „Play“ bezeichnet. Das Playbook **`webserver.yml`** (siehe Abbildung 3) stellt vereinfacht die Installation, den Start von **`httpd`** (Apache2 unter Red Hat) sowie die Bereitstellung der Datei **`index.html`** mittels eines Template dar. Solch ein Playbook wird mittels **`ansible-playbook`** ausgeführt:

```
ansible-playbook –i hosts webserver.yml
```

Hier wird auch wieder das Inventory übergeben, eine Definition der Hosts im Playbook selbst ist auch möglich.

Ausgehend davon, dass der Inhalt in der **`index.html`**, bzw. im entsprechenden Template in einem Git-Repository gepflegt wird, stellt sich die Frage: Wie kommen Änderungen aus dem Repository automatisch auf alle konfigurierten Webserver?

Entweder wird hier der entsprechende Aufruf von Ansible per **`cron`** geplant oder als manueller Schritt vorgesehen. Solch ein manueller Vorgang wird selbstverständlich gerne vergessen. Eine Automatisierung per **`cron`** erlaubt aber keine einfach zu konfigurierenden komplexen Abläufe, wie das Deployment auf Produktivsysteme im Erfolgsfalle. Genau hier wird eine Integration in Jenkins interessant.

Jenkins herrichten

Die Nutzung von Ansible via Jenkins kann auf zwei Arten erfolgen. Entweder wird jeder entsprechende Job per Build-Schritt **`Execute shell`** geplant oder das komfortablere Plugin **`Ansible`** wird genutzt.

Für die Integration von Git und Ansible sollten folgende Plugins in Jenkins installiert werden:

- Git [4]
- Ansible [5]
- AnsiColor [6]

Das Plugin **`Git`** übernimmt den Bezug der Playbooks aus einem Git-Repository.

Das Plugin **`Ansible`** stellt zwei neue Build-Schritte zur Verfügung:

- Invoke Ansible ad-hoc command
- Invoke Ansible Playbook

Ersteres wird genutzt um z.B. das Ausführen einmaliger Kommandos per Ansible durchzuführen (analog dem oben genannten **`uptime`**). Der Schritt für die Playbooks hingegen erlaubt dann das Ausführen komplexer Abläufe.

Die farbliche Aufbereitung der Ausgabe wird dann vom Plugin **`AnsiColor`** übernommen. Dadurch werden erfolgreiche Schritte Grün, fehlerhafte Rot und Warnungen in Gelb ausgegeben. Dieses Plugin sollte auch genutzt werden,

wenn Ansible als **Execute shell** ausgeführt wird, hier sollte direkt zu Beginn der Shell-Schritte noch die Angabe von **export ANSIBLE_FORCE_COLOR=true** gesetzt werden, damit Ansible die farbliche Hervorhebung realisiert. Bei Verwendung des Plugin für Ansible geschieht dies automatisch.

Ein Ansible ad-hoc-Kommando in Jenkins definieren

Soll nun das erwähnte **uptime** per Jenkins ausgeführt werden, wird das entsprechende **ad-hoc**-Kommando konfiguriert (siehe Abbildung 4). Dazu wird in der Weboberfläche von Jenkins ein **New Item** erstellt, der Typ des Projektes ist ein **Freestyle project** und der Name **uptime**. Zusätzlich muss sich das übergebene Inventory (hier **/opt/ansible/hosts**) auf dem Jenkins selbst befinden.

Zu den Optionen gehört die Aktivierung von **sudo**. Der von Ansible genutzte User wird dann auf dem Zielsystem per **sudo** die notwendigen erweiterten Rechte anfordern. Die erweiterten Einstellungen erlauben dann auch die Anzahl der parallel abzuarbeitenden Hosts zu verändern, 5 ist hier der Standard.

Die Einstellung **Check host SSH key** weist den von Ansible genutzten SSH-Client an, den Host Key des Zielsystems mit den ihm bekannten Host Keys zu vergleichen. Nach Möglichkeit sollte diese Option aktiviert sein, allerdings müssen die Keys dafür auf den Jenkins Host importiert werden. So kann aber das Verbinden via SSH auf unbekannte/unberechtigte Hosts verweigert werden und entsprechend die Sicherheit gesteigert werden.

Zusätzlich sollte noch die Option **Colorized stdout** aktiviert werden, dies sorgt dafür, dass Ansible die Schritte farbig hervorhebt. In Kombination mit der Option **Color ANSI Console Output** im Abschnitt **Build Environment** wird dann für jeden Build eine grafische Ausgabe erzeugt.

Nach dem Speichern kann der Build erstmalig ausgelöst werden, die Ausgabe sollte analog dem Aufruf in Abbildung 2 sein, allerdings eingebettet in die Weboberfläche von Jenkins.

Let´s play ansible

Ebenso einfach ist die Konfiguration eines Jobs in Jenkins, der ein Ansible Playbook ausführt. In dieser Konfiguration selbst wird dann auch direkt Git als SCM verwendet.

Einen Ablauf stellt Abbildung 5 dar. Der oder die User ändern den Inhalt eines Playbook oder auch nur den Inhalt eines Template und checken die Änderungen in ein Git Repository ein. Dieses Repository wird in regelmäßigen Abständen von Jenkins auf Änderungen geprüft. Haben sich Änderungen ergeben, so wird der Build-Prozess ausgelöst und Ansible versorgt (Deployment) die definierten Hosts mit dem neuen Inhalt. Haben sich keine Änderungen im Repository ergeben wird nichts unternommen.

In der Beispielkonfiguration (siehe Abbildung 6) findet die zeitliche Prüfung alle 15 Minuten statt. Zusätzlich ist Git als SCM ausgewählt, die Repository URL entspricht einem regulären Zugriff per SSH auf den Host, der das Repository anbietet.

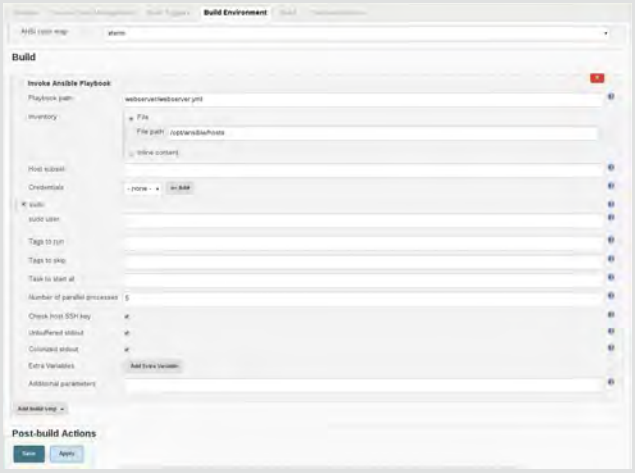

Abb. 7: Der Aufruf des Ansible Playbook unter Jenkins

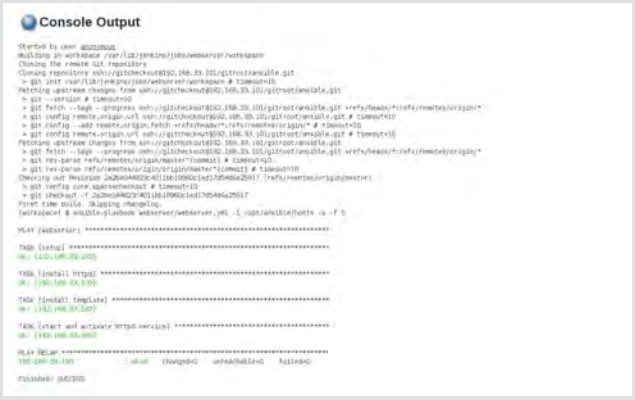

Abb. 8: Die Konsolenausgabe des Playbook webserver.yml

Abb. 9: Neues zur Wetterlage: Build-Übersicht in Jenkins

Als Build-Schritt wird nun **`Invoke Ansible Playbook`** ausgewählt. Der Pfad zum Playbook kann relativ angegeben werden, normalerweise ist die Basis für den Job unter **`/var/lib/jenkins/jobs/<JOBNAME>/workspace`**. Somit ist der Checkout des Repository auch direkt in diesem Verzeichnis. Das Git-Repository enthält ein Unterverzeichnis namens **`webserver`**, darin enthalten ist das eigentliche Playbook **`webserver.yml`**. Somit ergibt sich der vereinfachte Pfad zum Playbook:

`webserver/webserver.yml`.

Glossar

Ansible
Ansible ist eine Open Source Software von Red Hat zur IT-Automatisierung.

Continous Delivery
Dauerhafte Auslieferung einer Software bei Erreichen einer bestimmten Qualität.

Continous Integration
Stetiges Integrieren von Softwareteilen zu einem Ganzen, stellt bereits beim Einpflegen mögliche Probleme fest.

Fork
Die Abspaltung eines Softwareprojektes von einem bestehenden Projekt.

Git
Verteilte Versionsverwaltung von Dateien, mit dieser Software wird u.a. der Quellcode des Linux-Kernels verwaltet.

Inventory
Eine Datei die definiert, zu welchen Zielen per Ansible eine Verbindung aufgebaut werden soll. Die Hosts werden gruppiert. Eine Angabe weitere Verbindungsparameter (Benutzernamen, Ports, etc.) ist möglich.

Jenkins
Webbasierte Open Source Software für Continous Integration, Delivery und automatisiertes Testen.

Playbook
Die Zusammenfassung von mehreren Aufgaben für Ansible, können u.a. mit Abhängigkeiten gesteuert werden.

Play
Die Ausführung eines Playbook mit Ansible.

SCM
Source Code Management, eine Software zur Verwaltung von Quellcode (z.B. Git, Subversion).

YAML
Eine (für Menschen) leicht lesbare Sprache für Datenserialisierung.

Links

[1] Produktseite Ansible
https://www.ansible.com

[2] Pressemitteilung RedHat
https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible

[3] Produktseite Jenkins
https://jenkins.io

[4] Git Plugin
https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin

[5] Ansible Plugin
https://wiki.jenkins-ci.org/display/JENKINS/Ansible+Plugin

[6] AnsiColor Plugin
https://wiki.jenkins-ci.org/display/JENKINS/AnsiColor+Plugin

In der eigentlichen Build-Sektion (siehe Abbildung 7) werden nun die Einstellungen für die Ausführung des Plays vorgenommen. Das Inventory muss wieder angegeben werden, sofern im Playbook selbst die Hosts nicht definiert sind. Dieses Inventory zeigt auf die Datei **`/opt/ansible/hosts`**.

Hier ist Obacht geboten: eine Änderung am Inventory (bspw. Hinzufügen eines Host) löst keinen neuen Build aus. Erst ein neuer Commit in das Git-Repository würde auch den **`httpd`** auf dem hinzugefügten Host installieren!

Entweder wird das entsprechende Inventory auch im Git-Repository abgelegt oder über ein **`git submodul`** mehreren Git-Repositories hinzugefügt. Dies würde die Pflege der vollständigen Hosts an zentraler Stelle ermöglichen. Für die Benutzeranmeldung wird wieder auf **`sudo`** zurückgegriffen, dies ist aber konfigurierbar.

Die Ausführung des entsprechenden Build erfolgt dann unter der Ausgabe der einzelnen Schritte, dies beinhaltet den Git-Checkout sowie das Ausführen des Playbook (siehe Abbildung 8).

Die Wetteraussichten

Die in die Weboberfläche integrierte Übersicht (siehe Abbildung 9) von Jenkins erlaubt einen schnellen Blick auf die Liste der Builds. Dabei erleichtern Symbole wie Sonnenschein, bewölkt oder Gewitter das Erfassen der Zustände. Sind die letzten 5 Builds eines Jobs fehlgeschlagen, so wird hier eine Gewitterwolke dargestellt. Schlägt ein Build in den letzten 5 Anläufen fehl, so ist der Himmel bewölkt, die erfolgreiche Ausführung hingegen verheißt Sonnenschein.

Zeiten für den letzten Fehlschlag, den letzten erfolgreichen Build sowie die Dauer des Build werden in der Standardansicht dargestellt. Zusätzlich kann auch direkt die Konsolenausgabe des letzten Build kontrolliert werden.

Fazit

Die Integration von Ansible als Buildstep in Jenkins eröffnet neue Möglichkeiten für Continous Delivery. Durch die Gestaltung von abhängigen Schritten können erfolgreich geprüfte Versionen in die nächste Umgebung, beispielsweise die Produktion, gehoben werden. Die Optionen zur Benachrichtigung im Fehlerfall und die einfache Übersicht erlauben eine komfortable Administration. Somit können Entwickler durch das Einchecken in ein Repository ihre Neuerungen über nachvollziehbare Schritte durch mehrere Stufen in das Produktivsystem bringen.

Carsten Griese
(info@ordix.de)

SCCM Deployment (Teil II)

Automatisierte Oracle-Installation mittels des System Center Configuration Manager

Durch einen Umzug oder Neuaufbau eines Rechenzentrums müssen mehrere Installationen auf diversen Servern durchgeführt werden. So auch zum Beispiel bei Datenbankservern. Mit dem System Center Configuration Manager können Server mit einem Betriebssystem und zusätzlicher Software (z.B. Datenbanksoftware) automatisiert installiert werden.

Einleitung

Im zweiten Teil dieser Artikelreihe wird beschrieben, wie eine unbeaufsichtigte und vollautomatisierte Ausführung von mehreren Oracle-Installationen durchgeführt wird. Hierbei handelt es sich um die reine Installation und Konfiguration von Oracle und nicht um die Migration von Datenbanken. Zusätzlich wird hier pro Oracle Home nur eine Datenbank installiert.

Für eine grundlegende Einführung in das Thema Softwareverteilung mittels dem System Center Configuration Manager empfehlen wir den ersten Teil dieser Reihe [1].

Vorbereitungen

Zunächst werden die Installationsmedien von Oracle benötigt. Diese können auf der Oracle Internetseite [2] heruntergeladen werden. Anschließend werden die Setup-Dateien auf der SCCM-Freigabe abgelegt und somit dem SCCM zur Verfügung gestellt. Im nächsten Schritt wird sowohl eine globale Response-Datei für die Software-Only-Installation, als auch ein globales Datenbank-Template für die Datenbankerzeugung benötigt und ebenfalls auf der Freigabe abgelegt.

Innerhalb der Response-Datei und des Template gibt es dynamische Parameter, die während der Installation festgelegt werden. Dies geschieht mittels einer Batch-Datei, welche die Konfigurationsbefehle für die Oracle-Installation beinhaltet und die entsprechenden Parameter an den Befehl **`setup`**, sowohl für die Software-Only-Installation als auch die Datenbankerzeugung, dynamisch übergibt.

Oracle-Installationsdatei

Damit die Installation von verschiedenen Oracle-Datenbanken auf unterschiedlichen Servern stattfinden kann, wird eine zusätzliche Parameterdatei (z.B. **`Servernames.txt`**) benötigt. Innerhalb dieser Datei sind folgende Werte kommagetrennt einzutragen:

- Servername
- Oracle Home
- SID
- DB_UNIQUE_NAME
- Zeichensatz
- Nationaler Zeichensatz

```
For /F "tokens=1,2,3,4,5,6,7 delims=," %%A in
(ServerNames.txt) DO (
   SET srv=%%A
   SET home=%%B
   SET SID=%%C
   SET DB_UNIQUE=%%D
   SET Chr=%%E
   SET NChr=%%F
   SET oracle_base=D:\oracle
   call :ProcessNext
)
```

Abb. 1: Parameterdatei auslesen

```
IF /i %Computername% == %srv%( […] )
```

Abb. 2: IF-Abfrage Servername

```
%~dp0setup.exe -silent "ORACLE_HOSTNAME=%srv%"
"ORACLE_HOME=%home%" "ORACLE_BASE=%oracle_base%"
"ORACLE_HOME_NAME=%SID%"
"INVENTORY_LOCATION=D:\oracle\Inventory"
-force -nowait -noconsole -waitforcompletion -response-
File %~dp0oracle_sw_only.rsp
```

Abb. 3: Software-Only-Installationsbefehl

Parameter	Bedeutung
silent	unbeaufsichtigte Installation
ORACLE_HOSTNAME	Servername
ORACLE_HOME	ORACLE_HOME-Verzeichnis
ORACLE_BASE	ORACLE_BASE-Verzeichnis
nowait	Nach Abschluss wird das CMD-Fenster der Oracle-Installation automatisch beendet
force	Installation in vorhandenes Verzeichnis möglich
waitforcompletion	Installer wartet auf die Beendigung der Installation
responseFile	Parameterdatei

Abb. 4: Software-Only-Installationsparameter

```
start /wait cmd /k "copy %~dp0Global_Template.dbt %home%\
assistants\dbca\templates && exit"
```

Abb. 5: Datenbank-Template kopieren

```
start /wait cmd /k "%home%\BIN\dbca -silent -createDatabase
-templatename %home%\assistants\dbca\templates\Global_Tem-
plate.dbt -gdbName %SID% -sid %SID% -Characterset %Chr%
-nationalCharacterSet %NChr% -sysPassword P@ssw0rd -system-
Password P@ssw0rd -SERVICEUSERPASSWORD P@ssw0rd -initparams
db_name="%SID%" -initparams db_unique_name="%DB_UNIQUE%" -in-
itparams db_recovery_file_dest="E:\oradata\FRA\%SID%" -initpa-
rams db_recovery_file_dest_size=2560M && exit"
```

Abb. 6: Datenbank Installation & Konfiguration

Hier eine Parameterdatei mit Beispielinhalt:

```
SRV_01,D:\oracle\product\OraTest\12.1.0.2,
OraTest,OraTest1,WE8MSWIN1252,AL16UTF16

SRV_02,D:\oracle\product\ORAProd\12.1.0.2,
OraProd,OraProd1,WE8MSWIN1252,AL16UTF16
```

Die Werte werden zu Beginn durch eine Batch-Datei, welche alle Installationsbefehle enthält, in einer Schleife ausgelesen (siehe Abbildung 1) und anschließend in den entsprechenden Befehl **`setup`** für die Oracle-Installation eingefügt.

Damit die korrekte Oracle-Installation auf dem richtigen Server durchgeführt wird, ist innerhalb der Batch-Datei nach dem Auslesen der Parameterdatei eine einfache **`IF`**-Abfrage (siehe Abbildung 2) auf den Servername eingefügt.

Nur wenn der Parameter **`srv`** aus der Parameterdatei mit dem eigentlichen Servernamen übereinstimmt, werden die Installationsbefehle der Batch-Datei durchgeführt. Somit sind alle Installations- und Konfigurationsbefehle für Oracle innerhalb der **`IF`**-Abfrage einzufügen.

Software-Only-Installation

Die Software-Only-Installation wird mit dem Befehl aus Abbildung 3 durchgeführt. Wichtigster Parameter ist hierbei **`-silent`** (für die unbeaufsichtigte Installation). Alle weiteren Parameter sind in der Tabelle der Abbildung 4 aufgeführt. Der letzte Parameter ist die zuvor genannte Response-Datei. Bei dieser Datei handelt es sich um eine gewöhnliche Response-Datei von Oracle.

Der Parameter **`oracle.install.option`** muss innerhalb dieser Datei auf **`INSTALL_DB_SWONLY`** gesetzt sein. Alle dynamischen Parameter, welche durch den Befehl **`batch`** gesetzt werden, sind in der Response-Datei leer zu lassen (z.B. **`ORACLE_HOME=`**).

Datenbankerstellung

Bevor die eigentliche Datenbankerstellung durchgeführt werden kann, muss das Datenbank-Template in das durch die Software-Only-Installation erstellte Verzeichnis für den Database Creation Assistent (DBCA) kopiert werden. Dies geschieht mit dem Befehl aus Abbildung 5.

Nachdem das Template in das korrekte Verzeichnis kopiert wurde, kann anschließend der DBCA im **`silent`**-Modus ausgeführt werden (siehe Abbildung 6). Auch hier werden dynamische Parameter über die Batch-Datei an den Befehl **`setup`** übergeben. Die Parameter für die Datenbankerzeugung sind in der Tabelle der Abbildung 7 aufgelistet.

Abschluss

Folgende Dateien sind somit für die Installation notwendig und auf der SCCM-Freigabe abzulegen:

- Oracle-Installationsmedien
- Batch-Datei für die Installation
- Response-Datei
- Datenbank-Template
- Parameterdatei (Textdatei)

Für die Oracle-Installation mittels SCCM wird zunächst ein SCCM-Paket erstellt, welches alle hier beschriebenen Dateien enthält und als Programm die Batch-Datei ausführt.

Anschließend wird das erstellte Paket auf dem Distribution Point hochgeladen. Wichtig ist vor allem die Parameterdatei, da diese die Ausführung der korrekten Installation auf den unterschiedlichen Servern steuert.

Nun kann das erstellte Paket über eine Device Collection auf die gewünschten Server verteilt und ausgeführt werden. Hierbei sollte sichergestellt sein, dass in der Device Collection die Server enthalten sind, welche in der Parameterdatei aufgeführt sind. Das Erstellen und Verteilen eines SCCM-Paketes kann im ersten Teil dieser Artikelreihe nachgelesen werden [1].

Sebastian Herd
(info@ordix.de)

Parameter	Bedeutung
silent	Unbeaufsichtigte Installation
CreateDatabase	Datenbankerstellung
Templatename	Pfad inkl.Template
gdbName	Globaler Datenbankname
sid	SID
Characterset	Zeichensatz
nationalCharacterset	Nationaler Zeichensatz
sysPassword	Passwort: sys
systemPassword	Passwort: system
SERVICEUSERPASSWORD	Passwort: Dienstkonto
db_name	Datenbankname
db_unique_name	Unique Datenbankname
db_recovery_file_dest	Speicherort: Flash Recovery Area (FRA)
db_recovery_file_dest_size	Größer der FRA

Abb. 7: Datenbankerstellungsparameter

Links

[1] ORDIX® news Artikel –
„SCCM Deployment (Teil1): Automatisiertes Verteilen
von Software mittels System Center Configuration Manager“
https://www.ordix.de/ordix-news-archiv/1-2017.html

[2] Oracle Installationsmedien
http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

Enterprise Job Scheduler

Einführung in die Automation Engine

Heutzutage existiert wohl kaum noch ein Unternehmen, das ohne den Einsatz von IT auskommt. Das führt zwangsläufig dazu, dass immer wiederkehrende Aufgaben besonders in großen Unternehmen enorme Ressourcen verschlingen. Genau hier setzen Job Scheduler an – sie haben zum Ziel, solche Aufgaben in Prozessen abzubilden und automatisiert auszuführen. In diesem Artikel wird die Automation Engine von Automic vorgestellt, die besonders für den Einsatz in komplexen IT-Landschaften gedacht ist.

Einführung in das Job Scheduling

Die Idee eines Job Scheduler ist sicherlich nicht neu. Administratoren kennen seit jeher dutzende Routinetätigkeiten und lagern diese in Skripte aus. Unix und Windows haben auch entsprechende Werkzeuge im Bauch, um solche Skripte zeitgesteuert auszuführen. Während unter Unix der Cron-Daemon dafür zuständig ist, bildet die Aufgabenplanung das Gegenstück in der Windows Welt.

Die klassischen Dienste sind für einfache Wartungsjobs in jedem Fall ausreichend. Versucht man aber größere Geschäftsprozesse in Jobplänen abzubilden, so wird man meistens feststellen, dass hierfür viele Teilprozesse benötigt werden. Diese Teilprozesse müssen natürlich in einer bestimmten Abfolge abgearbeitet werden. Je komplexer der betrachtete Prozess ist, desto mehr Abhängigkeiten bestehen auch zwischen den Teilprozessen.

Erschwerend kommt hinzu, dass größere IT-Landschaften sehr heterogen sein können. So sind zum Beispiel verschiedene Betriebssysteme und Derivate, unterschiedliche Datenbanken und eine Vielzahl an Applikationen in einem Rechenzentrum keine Seltenheit.

Diese Abhängigkeiten machen es nötig, dass der eingesetzte Job Scheduler die verschiedenen Teilprozesse auf abgeschotteten Systemen über das ganze Rechenzentrum hinweg ausführen, auf verschiedene Ereignisse reagieren und den gesamten Geschäftsprozess in einem Jobplan abbilden kann [Q1].

Automic bietet mit der Automation Engine ein Produkt, in dem Jobpläne sehr einfach erstellt werden können. Durch eine Vielzahl an Objekten können die unterschiedlichen Geschäftsprozesse modelliert werden. Innerhalb des Jobplans können die verschiedenen Abhängigkeiten und Reaktionen auf Jobereignisse mit diversen Mitteln sehr einfach implementiert werden. Weiterhin liefert die Automation Engine Agenten für alle gängigen Betriebssysteme sowie RA- und SAP-Agenten, wodurch eine unternehmensweite Automatisierung über das gesamte Rechenzentrum hinweg realisiert werden kann. Zudem ist die Automation Engine mandantenfähig. Dies bringt den Vorteil, dass beispielsweise voneinander unabhängige Bereiche im System als eigene gekapselte Umgebungen eingerichtet werden können.

Architektur der Automation Engine

Die Automation Engine besteht aus verschiedenen Bestandteilen und Diensten, die das gesamte System abbilden. Je nach Umgebung und Lizenz können natürlich noch weitere Agenten angebunden werden (z.B. SAP Agent). Um die Funktionsweise der Automation Engine zu verstehen, werden in den folgenden Abschnitten die wichtigsten Komponenten näher beschrieben.

Datenbank

Die Automation Engine kennt eine Vielzahl an Objekten, die in der Automation Engine verwendet werden können. Möchte man beispielsweise einen Standard Jobplan anlegen, dann braucht die Automation Engine Informationen darüber, wie dieses Objekt auszusehen hat. Diese Objektdefinitionen werden in der Datenbank angelegt. Die vom Benutzer generierten Objekte werden ebenfalls in dieser Datenbank gespeichert.

Ein weiteres Merkmal der Automation Engine ist, dass jeder Job und Jobplan bei der Ausführung Reports und Statistiksätze generiert und diese in die Datenbank schreibt.

Die Datenbank kann auf verschiedenen Plattformen installiert werden (Oracle, DB2, Microsoft SQL Server). Allerdings sollten die Ressourcen bereits im Vorfeld gut geplant werden, da besonders für große Umgebungen genügend Arbeitsspeicher und Festplattenkapazität bereitgestellt werden muss. Eine Empfehlung für die richtige Dimensionierung ist in der offiziellen Dokumentation zu finden [Q2].

Automation Engine Server

Für die eigentliche Arbeit und Steuerung im System ist der Automation Engine Server zuständig – er bildet sozusagen den Kern des ganzen Systems. Dieser Kern wird durch verschiedene Server-Prozesse abgebildet. Der Automation Engine Server kennt dabei zwei Arten von Prozessen – den Communication Process (CP) und den Work Process (WP). Auf jedem System laufen mehrere CPs und WPs.

Die Ports und die Anzahl der Prozesse werden in der Konfigurationsdatei des Automation Engine Server definiert. Während die CPs für die Kommunikation verantwortlich sind, wie beispielsweise die Anmeldung über das User Interface, übernehmen die WPs die eigentliche Arbeit im System. Der zuerst gestartete WP hat eine besondere Bedeutung im System und wird primärer Arbeitsprozess genannt. Da es sich bei dem Automation Engine Server um das Herzstück des ganzen Systems handelt, wird in Abbildung 1 die Konfigurationsdatei **`ucsrv.ini`** examplarisch dargestellt. Diese Datei ist wie jede Konfiguration innerhalb der Automation Engine in Sektionen gegliedert. Weiterführende Informationen zu bestimmten Parametern können der offiziellen Dokumentation entnommen werden [Q3].

Agenten

Damit die Automation Engine Jobs auf verschiedenen Systemen ausführen kann, müssen auf jedem Zielsystem spezielle Agenten installiert und eingerichtet werden. Automic liefert Agenten für alle gängigen Betriebssysteme und zusätzlich weitere Agenten für spezielle Applikationen wie SAP. Web Services können über RA-Agenten angebunden werden. Diese zusätzlichen Agenten ermöglichen die Kommunikation zwischen der Automation Engine und dem angebundenen Server. Während der Jobplan innerhalb der Automation Engine erstellt wird, findet die eigentliche Ausführung auf dem Betriebssystem oder der Applikation des Agenten statt. Hierfür nimmt der entsprechende Agent den Job entgegen, führt diesen aus und liefert eine Rückmeldung.

User Interface

Das User Interface ist eine Java-basierte Schnittstelle, die dem Benutzer das Arbeiten mit der Automation Engine ermöglicht. Über diese GUI kann der Benutzer beispielsweise alle zur Verfügung stehenden Objekte nutzen und Geschäftsprozesse in Jobplänen abbilden. Weiterhin können diese über das Interface einmalig oder periodisch ausgeführt werden.

Wenn für die Ausführung noch weitere Kriterien, wie zum Beispiel der Wochentag, berücksichtigt werden müssen, kann der Benutzer auch ein spezielles Schedule-Objekt erzeugen. Das Schedule-Objekt ermöglicht eine periodische Ausführung der zusammengefassten Jobs anhand der festgelegten Kriterien.

Doch das User Interface wird nicht nur von den normalen Anwendern genutzt, sondern bildet auch für die Administratoren der Automation Engine ein hilfreiches Werkzeug. Hierüber haben sie die Möglichkeiten, Mandanten zu administrieren und systemrelevante Einstellung vorzunehmen.

Allerdings wird das User Interface seit der Version V11 nicht mehr weiterentwickelt. Mit der Version V12 liefert Automic eine Web Interface als neue Benutzerschnittstelle. Für diese Schnittstelle muss ein Webserver als zusätzliche Komponente installiert werden.

Die neue Webschnittstelle kann den vollen Funktionsumfang des alten User Interfaces allerdings noch nicht abdecken, weshalb das alte User Interface auch noch mit der Version V12 ausgeliefert wird. Neue Features können im alten User Interface jedoch nicht mehr verwendet werden.

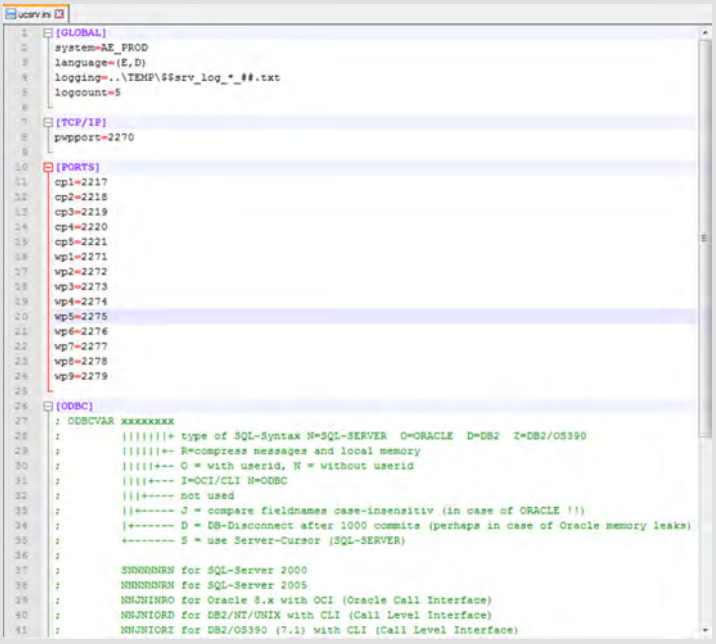

Abb. 1: Konfigurationsdatei ucsrv.ini des Automation Engine Server

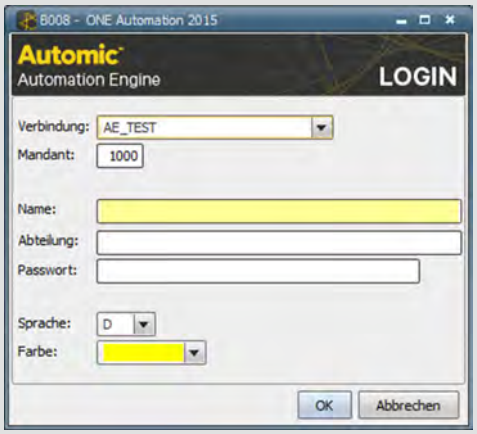

Abb. 2: Login am User Interface

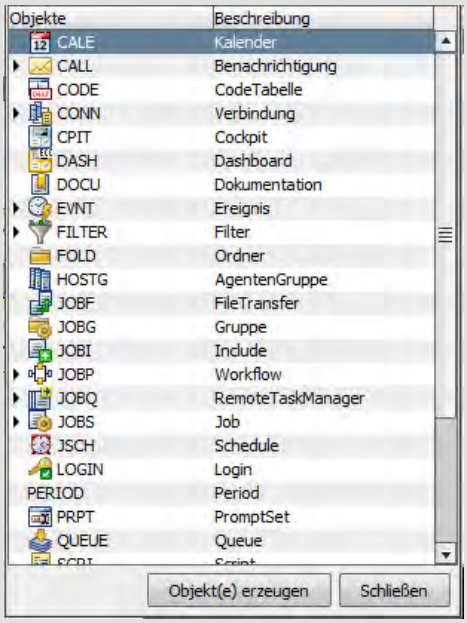

Abb. 3: Objekte der Automation Engine

Service Manager

Bei dem Service Manager handelt es sich um ein sehr nützliches Programm, mit dem definierte Prozesse und Agenten der Automation Engine gestartet werden können. Über den Service Manager des Automation Engine Server kann beispielweise festgelegt werden, in welcher Reihenfolge die Serverprozesse zu starten sind.

Der Service Manager kann und sollte auch auf jedem Server mit einem Agenten installiert und eingerichtet werden. Wird der Service Manager konsequent auf allen Agenten eingerichtet, können diese Agenten über das Automation Engine System ferngesteuert werden. Beispielsweise lässt sich bei richtiger Konfiguration jeder Betriebssystemagent über das User Interface starten und stoppen.

Dienstprogramme

Automic liefert zusätzlich diverse Dienstprogramme. Diese Programme sind zwar kein direkter Bestandteil des Automation Engine Systems, erleichtern die Installation und Administration der Automation Engine jedoch ungemein.

Funktionsweise der Automation Engine

Nachdem nun alle Komponenten der Automation Engine vorgestellt wurden, werden noch einmal die Funktionsweise des gesamten Systems und die Abläufe im Hintergrund anhand eines simplen Beispiels vorgestellt. In diesem Beispiel meldet sich ein Benutzer an der Automation Engine an, erstellt einen Unix Job und führt diesen anschließend aus.

Anmeldung

Nach dem Starten des User Interface muss sich der Benutzer zunächst über eine Eingabemaske (siehe Abbildung 2) am System anmelden. Damit sich ein Anwender überhaupt an der Automation Engine anmelden und dort Jobpläne erstellen kann, sind zwei Voraussetzungen erforderlich:

- Auf dem Client des Anwenders wurde das User Interface installiert und konfiguriert. In der Konfigurationsdatei **`uc4config.xml`** müssen zwingend die Verbindungsparameter eingestellt werden. Diese teilen dem User Interface mit, wie das Automation Engine System erreicht werden kann
- Der Anwender kennt seinen Mandanten im System und besitzt einen Benutzer mit ausreichenden Berechtigungen in diesem Mandanten

Für die Kommunikation zwischen dem User Interface und dem Automation Engine Server wird der konfigurierte CP genutzt. Nach erfolgreicher Authentifizierung wird eine aktive Verbindung durch diesen Prozess aufrechterhalten.

Joberstellung

Die Automation Engine verfügt über eine Vielzahl an Objekten. Eine kleine Auswahl der verfügbaren Objekte kann der Abbildung 3 entnommen werden. Ein Beispiel für eines dieser Objekte ist der Job, in dem genau definiert ist, wie die Verarbeitung auf dem Zielserver auszusehen hat. Entsprechend existieren für verschiedene Agenten auch unterschiedliche Typen (Unix Job, Windows Job, …).

Die genaue Definition des Unix Job befindet sich in der Datenbank. Der Automation Engine Server lädt mit Hilfe eines WP die genaue Definition aus der Datenbank und ermöglicht unserem Benutzer nun, einen Unix Job mit der GUI anzulegen. In diesem Job muss der Anwender eben-

falls den Unix Agenten und die Zugangsdaten hinterlegen. Diese Informationen stellen Attribute des Job Objekts dar und können entweder statisch im Job oder dynamisch mit Skriptsprachmitteln in der Automation Engine festgelegt werden. Über einen WP speichert der Automation Engine Server diesen Job anschließend in der Datenbank des Systems.

Jobausführung

Agenten werden in der Automation Mandaten zugeordnet. Damit der erstellte Job also auf dem ausgewählten Unix-Agenten ausgeführt werden kann, muss sich dieser Agent im gleichen Mandanten befinden und mit der Automation Engine verbunden sein. Ähnlich wie bei dem User Interface, müssen die Verbindungsparameter auch auf dem Agenten eingestellt werden. Die Zuordnung zu einem Mandanten obliegt jedoch dem Administrator. An dieser Stelle sei darauf hingewiesen, dass ein Agent durchaus in mehreren Mandanten verwendet werden kann.

Sofern diese Bedingungen gegeben sind, kann der Benutzer den erstellten Job ausführen. Während die Verbindung zwischen dem Agent und dem Automation Engine Server durch einen CP aufrechterhalten wird, sorgt ein WP für die Aktivierung und Ausführung des Jobs. Dieser Job wird an den Agenten übergeben und anschließend ausgeführt. Solange der Job noch aktiv ist, wird er durch den WP zusätzlich überwacht. Der Agent meldet das Ergebnis nach der Ausführung wieder an den Automation Engine Server, der wiederum den Jobreport und Statistiksatz zu diesem Job in der Datenbank speichert.

Fazit

Automic bietet mit der Automation Engine einen Job Scheduler, mit dem komplexe Geschäftsprozesse über die gesamte IT-Landschaft im Unternehmen abgebildet und automatisiert werden können. Da die Automation Engine eine Vielzahl an Agenten und vordefinierten Objekten mitbringt, kann sie ohne weiteres auch in heterogenen Server-Umgebungen eingesetzt werden.

Trotz aller Ressourceneinsparung durch die Automatisierung sollte nicht vergessen werden, dass die Einführung und der Betrieb eines solchen Systems durchaus sehr zeitintensiv sein kann. Personelle Ressourcen werden nicht nur für die Installation und Konfiguration benötigt, sondern vor allem für die Administration. Der administrative Aufwand hängt natürlich von der Größe des Automation Engine System ab. Während kleinere Umgebungen sicherlich mit einem Administrator auskommen, bedarf es in Umgebungen mit mehreren tausend Agent auch ganzen Abteilungen, die den Betrieb sicherstellen. Neben dem zusätzlichen Aufwand für den Betrieb, dürfen die Lizenzkosten für einen solchen Enterprise Job Scheduler nicht außer Acht gelassen werden.

Glossar

CP
Hinter der Abkürzung versteckt sich der Begriff Communication Process. Hierbei handelt es sich um die Kommunikationsprozesse des Automation Engine Server. Diese Prozesse benötigt die Automation Engine, damit sich sowohl Benutzer über das User Interface als auch alle Agenten mit dem Automation Engine Server verbinden können.

RA
Mit dem Begriff Rapid Automation beschreibt Automic eine Technologie, um weitere Services an die Automation Engine anzubinden. Der Einrichtungsaufwand hängt hierbei stark von dem anzubindenden Service ab. Mit dieser Funktionalität kann beispielsweise eine Anbindung an einen Web Service realisiert werden.

WP
Der Work Process, auch Arbeitsprozess genannt, ist für die eigentliche Arbeit in der Automation Engine zuständig. Wenn zum Beispiel ein Job ausgeführt oder der Report eines Jobs in die Datenbank geschrieben wird, erfolgt dies durch die Arbeitsprozesse.

Links/Quellen

[1] Online Dokumentation der Automation Engine
https://docs.automic.com/documentation/webhelp/german/ALL/components/AE/11.1/All%20Guides/help.htm

[2] Seminare
https://www.qskills.de/qs/workshops/automic/?PHP_Session=ugroahwh

[Q1] Admin Magazin
http://www.admin-magazin.de/Das-Heft/2010/02/Geplante-Tasks-automatisch-ausfuehren

[Q2] Unterstützte Plattformen
https://docs.automic.com/documentation/webhelp/german/ALL/components/AE/11.2/All%20Guides/help.htm#sys_requirements.htm%3FTocPath%3DAdministratorhandbuch%7CInstallation%7C_____1

[Q3] Aufbau der Automation Engine INI-Datei
https://docs.automic.com/documentation/webhelp/german/ALL/components/AE/11.2/All%20Guides/help.htm#ucaaqm.htm%3FTocPath%3DAdministratorhandbuch%7CKonfiguration%7CAufbau%2520der%2520Konfigurationsdateien%7CAutomation%2520Engines%7C_____1

[Q4] Online Dokumentation der Automation Engine
https://docs.automic.com/documentation/webhelp/german/ALL/components/AE/11.2/All%20Guides/help.htm

Alexander Stondenko
(info@ordix.de)

ORDIX® seminare

Seminarzentrum:
ORDIX AG
Kreuzberger Ring 13
65205 Wiesbaden
Tel.: 0611 77840-00

Online-Anmeldung, aktuelle Seminarinhalte und Termine unter:
https://seminare.ordix.de

Unser Seminarstandort ist Wiesbaden.
Die hier angegebenen Termine entsprechen dem ersten Seminartag.
Die Preise gelten pro Seminar pro Teilnehmer in Euro zzgl. ges. MwSt., Inhouse-Preise auf Anfrage.

Big Data und Data Warehouse

BIG Data				
DB-BIG-01	Big Data: Informationen neu gelebt	1 Tag	590,00 €	05.03. | 22.05. | 27.08. | 05.11.
DB-BIG-02	Big Data: Apache Hadoop Grundlagen	3 Tage	1.290,00 €	12.03. | 04.06. | 10.09. | 03.12.
Data Warehouse				
DB-DB-03	Data Warehouse Grundlagen	3 Tage	1.290,00 €	06.03. | 23.05. | 21.08. | 06.11.
DB-NSQL-01	Einführung in NoSQL-Datenbanken	2 Tage	1.090,00 €	15.03. | 07.06. | 13.09. | 06.12.

PostgreSQL

DB-PG-01	PostgreSQL Administration	5 Tag	2.150,00 €	12.02. | 23.04. | 30.07. | 24.09. | 03.12.

Oracle

Entwicklung				
DB-ORA-01	Oracle SQL	5 Tage	1.890,00 €	05.02. | 16.04. | 18.06. | 20.08. | 15.10.
DB-ORA-01A	Oracle SQL Power Workshop	3 Tage	1.290,00 €	19.03. | 11.06. | 17.09. | 26.11.
DB-ORA-02	Oracle Datenbankprogrammierung mit PL/SQL Grundlagen	5 Tage	1.890,00 €	12.02. | 23.04. | 03.09. | 19.11.
DB-ORA-34	Oracle Datenbankprogrammierung mit PL/SQL Aufbau	3 Tage	1.290,00 €	05.03. | 07.05. | 29.10. | 10.12.
DB-ORA-42	Oracle PL/SQL für Experten - Performance Analyse & Laufzeitopt.	3 Tage	1.290,00 €	26.02. | 02.07. | 08.10.
DB-ORA-53	Oracle Text	3 Tage	1.390,00 €	19.02. | 14.05. | 24.09. | 10.12.
DB-ORA-51	Oracle Spatial	3 Tage	1.290,00 €	05.02. | 02.05. | 03.09. | 19.11.
DB-ORA-46	Oracle APEX Anwendungsentwicklung Grundlagen	3 Tage	1.290,00 €	26.03. | 11.06. | 24.09. | 12.11.
DB-ORA-47	Oracle APEX Anwendungsentwicklung Aufbau	3 Tage	1.290,00 €	09.04. | 02.07. | 22.10. | 03.12.
Administration				
DB-ORA-03	Oracle Datenbankadministration Grundlagen	5 Tage	1.990,00 €	15.01. | 09.04. | 09.07. 24.09. | 12.11.
DB-ORA-04	Oracle Datenbankadministration Aufbau	5 Tage	1.990,00 €	22.01. | 23.04. | 30.07. 22.10. | 10.12.
DB-ORA-07	Oracle Tuning - Theorie und Interpretation von Reports	5 Tage	2.290,00 €	12.03. | 02.07. | 15.10. 03.12.
DB-ORA-11	Oracle Troubleshooting Workshop	3 Tage	1.390,00 €	Termine auf Anfrage
DB-ORA-08	Oracle 12c Real Application Cluster (RAC) und Grid Infrastructure	5 Tage	2.290,00 €	19.02. | 14.05. | 23.07. | 10.09. | 26.11.
DB-ORA-49	Oracle 12c Neuheiten	5 Tage	2.090,00 €	19.03. | 25.06. | 06.08. | 08.10.
DB-ORA-49E	Oracle 12c Neuheiten für Entwickler	3 Tage	1.390,00 €	26.03. | 04.06. | 17.09. | 05.11.
DB-ORA-52W	Oracle Lizenz Workshop Webinar	1 Tage	590,00 €	Termine auf Anfrage
DB-ORA-33	Oracle Security	3 Tage	1.290,00 €	08.01. | 22.05. | 27.08. | 05.11.
DB-ORA-35	Oracle Cloud Control	3 Tage	1.290,00 €	26.03. | 16.07. | 10.09. | 29.10.
DB-ORA-55	Oracle ASM für Single Instance	3 Tage	1.390,00 €	12.02. | 07.05. | 06.08. | 12.11.
DB-ORA-56	Oracle Tenant Technologie (Multi/Single Tenant)	3 Tage	1.390,00 €	05.02. | 02.05. | 27.08. | 29.10.
DB-ORA-57	Single Sign On mit Oracle	3 Tage	1.390,00 €	28.05. | 13.08. | 22.10.
Backup und Recovery				
DB-ORA-32	Oracle Backup und Recovery mit RMAN	5 Tage	1.990,00 €	29.01. | 16.04. | 30.07. | 08.10.
DB-ORA-31	Oracle Data Guard	4 Tage	1.690,00 €	26.02. | 14.05. | 20.08. | 15.10.
MySQL				
DB-MY-01	MySQL Administration	3 Tage	1.290,00 €	17.07. | 25.09. | 27.11.

IBM Datenbanksysteme

Informix				
DB-INF-01	IBM Informix SQL	5 Tage	1.790,00 €	05.03. | 18.06. | 24.09.2018
DB-INF-02	IBM Informix Administration	5 Tage	1.990,00 €	19.03. | 23.03. | 02.07.2018
DB2				
DB-DB2-01	IBM DB2 für Linux/Unix/Windows SQL Grundlagen	5 Tage	1.890,00 €	12.03. | 25.06. | 17.09.
DB-DB2-02	IBM DB2 für Linux/Unix/Windows Administration	5 Tage	1.990,00 €	09.04. | 09.07. | 08.10.
DB-DB2-05	IBM DB2 für Linux/Unix/Windows Monitoring und Tuning	3 Tage	1.290,00 €	02.05. | 06.08. | 26.11.
DB-DB2-06	IBM DB2 für Linux/Unix/Windows Backup und Hochverfügbarkeit mit HADR	3 Tage	1.390,00 €	23.04. | 30.07. | 12.11.

Microsoft

Entwicklung				
MS-SQL-01	Querying Data with Transact-SQL	5 Tage	1.990,00 €	26.02. | 04.06. | 03.09. | 19.11.
MS-SQL-07	Updating Your Skills to Microsoft SQL Server 2016	5 Tage	1.990,00 €	22.01. | 07.05. | 27.08. | 29.10.
Administration				
MS-SQL-02	Administering a SQL Database Infrastructure	5 Tage	1.990,00 €	19.02. | 11.06. | 10.09. | 26.11.
MS-SQL-05	Implementing a SQL Data Warehouse	5 Tage	1.990,00 €	05.03. | 22.05. | 13.08. | 15.10.
MS-SQL-11	Microsoft SQL Server for Oracle DBAs	4 Tage	1.790,00 €	15.01. | 16.04. | 20.08. | 22.10.
MS-SQL-17W	Microsoft SQL Server 2017 Upgrade Webinar	1 Tag	99,00 €	04.01.

Rechenzentrum

ANSIB-01	Konfigurationsmanagement mit Ansible	3 Tage	1.350,00 €	12.03. | 04.06. | 04.09. | 27.11.
E-Dock-01	Docker DevOps Workshop	1 Tag	405,00 €	09.04. | 29.06. | 03.09. | 26.11.
SM-NAG-01	Systemüberwachung mit Nagios - Workshop	3 Tage	1.190,00 €	29.01. | 16.04. | 09.07. | 15.10.

Web und Application-Server

INT-04	Apache HTTP Server Administration	3 Tage	1.190,00 €	19.02. | 28.05. | 03.09. | 19.11.
INT-07	Tomcat Konfiguration und Administration	3 Tage	1.290,00 €	05.03. | 11.06. | 17.09. | 26.11.
INT-08	WebSphere Application Server Installation und Administration	3 Tage	1.390,00 €	05.02. | 23.05. | 27.08. | 29.10.
INT-12	WildFly Application Server Administration	3 Tage	1.290,00 €	26.03. | 23.07. | 15.10. | 10.12.
DB-ORA-50	Oracle WebLogic Administration Grundlagen	3 Tage	1.390,00 €	26.02. | 18.06. | 10.09. | 05.11.

IT-Security

IT-SEC-01	IT-Sicherheit für Projektmanager und IT-Leiter - ein Überblick	3 Tage	1.590,00 €	22.01. | 04.04. | 30.07. | 24.09. | 03.12.
IT-SEC-02	Certified Information Systems Security Professional (CISSP)	5 Tage	4.240,00 €	12.03. | 02.07. | 27.08. | 22.10.
IT-SEC-03	Certified Information Security Manager (CISM)	3 Tage	1.820,00 €	14.05. | 03.12.
IT-SEC-04	Certified Information Systems Auditor (CISA)	4 Tage	2.130,00 €	05.03. | 06.08.
IT-SEC-05	Security Awareness für Mitarbeiter	1 Tage	1.000,00 €	09.03. | 17.05. | 10.08. | 06.12.

Projekt- und IT-Management

Klassisches Projektmanagement

PM-01	IT-Projektmanagement praxisorientiert	3 Tage	1.690,00 €	05.02. | 16.04. | 06.08. | 08.10.
PRINCE-01	PRINCE2® Foundation	3 Tage	1.225,00 €	05.02. | 05.03. | 09.04. | 14.05. | 04.06. | 02.07. | 06.08.
PRINCE-02	PRINCE2® Practitioner	3 Tage	1.560,00 €	10.01. | 07.02. | 07.03. | 11.04. | 16.05. | 06.06. | 04.07.
PRINCE-03	PRINCE2® kompakt	5 Tage	2.595,00 €	05.02. | 05.03. | 09.04. | 14.05. | 04.06. | 02.07. | 03.09.
PM-06	Projekte souverän führen - Systemisches Projektmanagement	4 Tage	1.850,00 €	12.02. | 09.07. | 29.10.
PM-05	Projektcontrolling in der IT	2 Tage	1.090,00 €	08.02. | 19.04. | 09.08. | 11.10.
PM-07	Krisenmanagement in Projekten - Projektkrisen vorbeugen & meistern	2 Tage	1.100,00 €	01.03. | 14.06. | 27.09.
PM-14	Anforderungsmanagement in IT-Projekten	2 Tage	1.090,00 €	15.02. | 12.04. | 23.07. | 15.11.
PM-T-01	Testmanagement für agile und klassische Projekte	2 Tage	1.190,00 €	01.02. | 07.05. | 05.07. | 18.10.

Agiles Projektmanagement

AGIL-01	Agil führen - Neue Konzepte für Ihre Führung im agilen Umfeld	3 Tage	1.190,00 €	19.03. | 23.07. | 03.12.
SCRUM-01	Agiles Projektmanagement mit Scrum - Mit agilem Vorgehen mehr ...	2 Tage	1.190,00 €	19.02. | 27.08. | 19.11.
SCRUM-02	Scrum Vorbereitung zur Zertifizierung - So einfach kann es klappen	1 Tage	690,00 €	21.02. | 29.08. | 21.11.
SCRUM-04	Scrum Product Owner - Produkte erfolgreich entwickeln	3 Tage	690,00 €	02.05. | 17.09.
KB-01	KANBAN in der IT - Prozesse & Projekte mit Hilfe von Kanban optimieren	2 Tage	1.190,00 €	22.02. | 30.08. | 22.11.

IT-Management, IT-Strategie und IT-Organisation

PM-29	Systemische Führung - Führung unter Berücksichtigung aller Aspekte	3 Tage	1.650,00 €	07.05. | 13.08. | 26.11.
MGM-07	IT-Strategie - strategische IT-Planungen	3 Tage	1.650,00 €	12.03. | 25.06. | 10.12.
MGM-02	IT-Architekturen	3 Tage	1.590,00 €	26.02. | 23.05. | 22.10.
PM-10	IT-Controlling	3 Tage	1.590,00 €	04.04. | 18.06. | 03.09.
MGM-04	Geschäftsprozessmanagement (BPM)	3 Tage	1.590,00 €	23.04. | 20.08. | 15.10.
ITIL-01	ITIL® V3 Foundation	3 Tage	922,50 €	15.01. | 12.02. | 12.03. | 16.04. | 07.05. | 11.06. | 09.07.
ITIL-02	ITIL® V3 Practitioner	3 Tage	1.380,00 €	18.01. | 15.02. | 15.03. | 19.04. | 14.06. | 12.07. | 23.08.
ITIL-03	ITIL® V3 kompakt	3 Tage	2.250,00 €	15.01. | 12.02. | 12.03. | 16.04. | 11.06. | 09.07. | 20.08.
PM-28	IT-Organisation	3 Tage	1.650,00 €	14.05. | 10.09.

Kommunikation und Selbstmanagement

K-03	Effektive Kommunikation in Projekten - Soft Skills Workshop	2 Tage	1.090,00 €	25.01. | 28.05. | 06.09. | 06.12.
K-04	Konfliktmanagement - Mehr Sicherheit in unsicheren Situationen	2 Tage	1.100,00 €	08.03. | 21.06. | 25.10.
K-05	Zeit- und Selbstmanagement - Mit weniger Stress mehr erreichen	2 Tage	1.090,00 €	15.03. | 28.06. | 08.11.
K-10	Coaching von IT Mitarbeitern - Führungskraft als Coach	3 Tage	1.090,00 €	09.04. | 28.05. | 17.12.
K-11	Kundenorientiertes Verhalten im Service	2 Tage	1.190,00 €	19.02. | 16.05. | 29.08. | 17.10.
K-12	Plötzlich IT-Führungskraft - Von Anfang an richtig führen	4 Tage	1.690,00 €	26.03. | 04.06. | 08.10.
K-20	Beratungs Know-How für IT-Experten - Kunden & Kollegen gut beraten	3 Tage	1.390,00 €	16.04. | 25.06. | 24.09.
MGM-03	IT-Management Einführung - Die IT nachhaltig zum Erfolg führen	3 Tage	1.590,00 €	05.03. | 11.06. | 05.11.
MGM-09	Leadership Skills - Reflektion des eigenen Führungsstils	2 Tage	1.190,00 €	14.02. | 14.05. | 27.08. | 15.10.
PM-CH-01	Change-Management in der IT - Veränderungen reibungslos einführen	3 Tage	1.590,00 €	23.04. | 16.07. | 12.11.

Storage

STORAGE01	Storage Grundlage	2 Tage	1.190,00 €	15.03. | 07.06. | 20.09. | 13.12.

Betriebssysteme & Monitoring

Unix/Linux

BS-01	Unix/Linux Grundlagen für Einsteiger	5 Tage	1.690,00 €	29.01. | 23.04. | 09.07. | 17.09. | 26.11.
BS-02	Linux Systemadministration	5 Tage	1.690,00 €	12.02. | 04.06. | 06.08. | 15.10. | 03.12.
BS-25	Unix Power Workshop für den Datenbank- & Applikationsbetrieb	5 Tage	1.890,00 €	26.02. | 18.06. | 10.09. | 19.11.
BS-27	Neuerungen SUSE Linux Enterprise Server 12	3 Tage	1.290,00 €	15.01. | 09.04. | 16.07. | 08.10.
BS-09	Linux Hochverfügbarkeits-Cluster	5 Tage	1.890,00 €	19.02. | 11.06. | 03.09. | 10.12.

Solaris

BS-03-11	Solaris 11 Systemadministration Grundlagen	5 Tage	1.990,00 €	05.03. | 25.06. | 27.08. | 22.10.
BS-04-11	Solaris 11 Systemadministration Aufbau	5 Tage	1.990,00 €	19.03. | 02.07. | 24.09. | 05.11.
BS-06-11	Solaris 11 für erfahrene Unix/Linux-Umsteiger	5 Tage	1.990,00 €	19.02. | 14.05. | 20.08. | 12.11.
BS-24	Solaris 11 Administration Neuheiten	3 Tage	1.290,00 €	04.04. | 30.07. | 29.10.

IBM AIX

AIX-01	IBM AIX Systemadministration Grundlagen	5 Tage	1.990,00 €	12.03. | 18.06. | 03.09. | 19.11.
AIX-04	IBM AIX Systemadministration Power Workshop	3 Tage	1.290,00 €	19.03. | 25.06. | 10.09. | 26.11.
AIX-02	IBM AIX Installation, Backup und Recovery mit NIM	3 Tage	1.290,00 €	26.03. | 02.07. | 17.09. | 10.12.

Entwicklung

Allgemeines

OO-01	Einführung in die objektorientierte Programmierung und UML	3 Tage	1.190,00 €	15.01. | 04.04. | 16.07. | 17.09.
E-SWA-01	Softwarearchitekturen	5 Tage	1.890,00 €	19.02. | 04.06. | 03.09. | 10.12.

Script-Sprachen

P-PERL-01	Perl Programmierung Grundlagen	5 Tage	1.690,00 €	29.01. | 09.04. | 30.07. | 08.10.
P-PERL-02	Perl Programmierung Aufbau	5 Tage	1.690,00 €	12.02. | 23.04. | 13.08. | 22.10.
P-UNIX-01	Shell, Awk und Sed	5 Tage	1.690,00 €	05.02. | 16.04. | 27.08. | 12.11.
P-PYTH-01	Python Programmierung	4 Tage	1.690,00 €	12.03. | 11.06. | 24.09. | 03.12.

XML

P-XML-01	XML Grundlagen	3 Tage	1.190,00 €	29.01. | 02.05. | 20.08. | 05.11.

Java

P-JAVA-01	Java Programmierung Grundlagen	5 Tage	1.690,00 €	05.03. | 18.06. | 10.09. | 05.11.
P-JAVA-03	Java Programmierung Aufbau	5 Tage	1.690,00 €	19.03. | 25.06. | 24.09. | 19.11.
P-JEE-08	Java Performance Tuning	3 Tage	1.290,00 €	12.03. | 30.07. | 15.10. | 03.12.
P-JAVA-11	Java 8 Neuheiten	2 Tage	990,00 €	01.02. | 19.04. | 23.07. | 18.10.

Java EE

P-JAVA-12	Java EE Power Workshop	5 Tage	1.890,00 €	22.01. | 23.04. | 16.07. | 26.11.
J-HIB-01	Java Persistence API mit Hibernate	5 Tage	1.690,00 €	29.01. | 09.04. | 09.07. | 12.11.
INT-05	Java Web Services	3 Tage	1.190,00 €	05.02. | 07.05. | 06.08. | 29.10.
P-JEE-06	Spring Power Workshop	5 Tage	1.590,00 €	15.01. | 14.05. | 13.08. | 10.12.

Web- und GUI-Entwicklung

P-PHP-01	PHP Programmierung	5 Tage	1.690,00 €	29.01. | 23.04. | 30.07. | 12.11.
P-JEE-05	Rich Internet Application mit JSF und Primefaces	5 Tage	1.590,00 €	26.02. | 11.06. | 08.10. | 10.12.
P-JEE-05A	Webanwendungen mit JavaServer Faces (JSF)	5 Tage	1.590,00 €	19.02. | 14.05. | 20.08. | 22.10.
E-ANG-02	Webanwendungen mit Angular 2	3 Tage	1.590,00 €	31.01. | 26.03. | 25.07. | 24.10.
E-TYPSC-01	TypeScript Grundlagen	2 Tage	1.190,00 €	29.01. | 22.03. | 23.07. | 22.10.

Tools und Verfahren

P-CI-01	Continuous Integration (CI) Workshop	3 Tage	1.190,00 €	04.04. | 25.07. | 05.09. | 26.11.

Mit der Datenbank durch die Zeit reisen (Teil I)

Temporales Datenmanagement in IBM Db2 macht`s möglich

In diesem ersten Artikel der zweiteiligen Reihe werden die Grundkonzepte sowie die Verwendung von System Time dargestellt. In der nächsten Ausgaben werden dann die Benutzung von Business Time und die Kombination aus System- und Business Time gezeigt.

Einführung und System Time

Durch die Zeit reisen – ein Traum so alt wie die Menschheit. Zumindest auf Datenebene kann dieser verwirklicht werden: Db2 (für Linux, Unix und Windows) verfügt ab Version 10 über eine Zeitmaschine, mit der sich Daten aus der Vergangenheit, Gegenwart und Zukunft betrachten lassen.

Das in der Db2 integrierte temporale Datenmanagement spart Zeit und Geld. Neue Funktionen zur Auswertung historischer Datensätze und zur Verfolgung zeitlicher Änderungen erleichtern Datenbankadministratoren und Anwendungsentwicklern die Arbeit erheblich.

Statt eigenentwickelter Trigger oder komplexer Funktionen erleichtert Db2 solche Aufgaben mit Hilfe temporaler Tabellen sowie durch Abfragen und Semantiken auf der Grundlage von ANSI/ISO SQL:2011.

Anwendungsszenarien einer temporalen Datenhaltung

Temporale Datenhaltung ist in vielen verschiedenen Anwendungsszenarien nützlich:

1. Für die interne Revision muss ein Kreditinstitut einen Bericht über die in den vergangenen fünf Jahren vorgenommenen Änderungen an den Unterlagen eines Kunden vorlegen.
2. Im Rahmen eines anhängigen Rechtsstreits muss ein Krankenhaus seinen Kenntnisstand über den medizinischen Zustand eines Patienten vor der Umstellung auf eine neue Behandlung überprüfen.
3. Ein Kunde legt Widerspruch gegen die Regulierungsentscheidung einer Versicherung im Zusammenhang mit einem Autounfall ein. Die Versicherung muss nun überprüfen, welche Versicherungsbedingungen zum Zeitpunkt des Unfalls Bestand hatten.

4. Ein Online-Reisebüro möchte, dass das System bestimmte logische Widersprüche bei Reisebuchungen erkennt. Falls also ein Kunde für acht Tage ein Hotel in Rom bucht und für drei dieser Tage einen Mietwagen in New York reserviert, möchte das Reisebüro benachrichtigt werden, um dies prüfen zu können.
5. Ein Einzelhändler will sicherstellen, dass für jedes Produkt in seinem Sortiment innerhalb eines beliebigen Zeitraums höchstens ein einzelner Rabatt gewährt wird.
6. Wegen eines Fehlers bei der Dateneingabe im Zusammenhang mit einem dreimonatigen Sonderzinssatz für die Kreditkarte eines Neukunden muss die Bank nun eine Korrektur vornehmen - und einen neuen Saldo berechnen, falls erforderlich.

Bei jedem dieser Szenarien ist Zeit ein kritischer Faktor. Die Funktionen für temporale Datenhaltung in Db2 unterstützen mit relativ geringem Aufwand die Entwicklung zeitbezogener Anwendungen und Abfragen. Vor der Vertiefung dieser Fähigkeiten werden zunächst zeitbezogene Konzepte vorgestellt und dabei auch die Unterschiede zwischen System Time (Bearbeitungszeit) und Business Time (Gültigkeitszeit bzw. Anwendungszeit) erläutert.

Basiskonzepte

System Time ermöglicht das Nachverfolgen von Änderungen in einer Tabelle, etwa wenn eine Versicherungspolice geändert oder ein Kredit vergeben wird (Historisierung). Business Time ermöglicht die zeitlich korrekte Darstellung von Geschäftsvorfällen, beispielsweise die jeweils aktuellen Bedingungen einer Versicherungspolice oder der jeweilige Zinssatz eines Kredits. Bisweilen wird Business Time auch als Gültigkeits- oder Anwendungszeit bezeichnet. Sollen sowohl System Time wie auch Business Time dargestellt werden, müssen beide Arten temporaler Daten in einer Tabelle abgebildet sein. Solche Tabellen nennt man dann bitemporal.

Wie lassen sich diese Konzepte nun auf die oben beschriebenen Szenarien anwenden? Für die Szenarien 1 und 2 werden Informationen über die Bearbeitungszeit benötigt, um den historischen Zustand einer oder mehrerer Tabellen darstellen zu können. Für die Szenarien 3 bis 6 werden Informationen über die Gültigkeitszeit benötigt, um die Gültigkeitszeiträume verschiedener Geschäftsvorfälle darzustellen. Darüber hinaus wird für Szenario 6 möglicherweise auch die Bearbeitungszeit benötigt - das heißt bitemporale Daten - sofern die Bank den Dateneingabefehler rückwirkend korrigieren und gleichzeitig festhalten möchte, wann der Fehler korrigiert wurde.

Periodenkonzepte sind sowohl für die Gültigkeitszeit wie auch für die Bearbeitungszeit relevant. Eine Periode gibt den Startzeitpunkt und den Endzeitpunkt eines Zeitraumes an. Bei Db2 lassen sich die Start- und Endzeiten einer Periode mit zwei Spalten einer Tabelle festlegen. Die Unterstützung temporaler Daten für neue oder vorhandene Tabellen erfolgt mit Hilfe von Syntaxerweiterungen der Anweisungen **CREATE TABLE** und **ALTER TABLE**.

Vorteile temporaler Datenhaltung

Die in Db2 integrierten Funktionen für temporale Datenhaltung ermöglichen eine einfachere Anwendungsentwicklung und gewährleisten die konsistente Verarbeitung zeitbezogener Ereignisse für alle Anwendungen, die auf die Datenbank zugreifen – einschließlich kommerzieller Anwendungen. Mit Hilfe einfacher deklarativer Anweisungen kann man in Db2 eine automatische Historie der Änderungen an einer Datenbank anlegen oder die Termine bestimmter Geschäftsvorfälle verfolgen, ohne dass dafür eigene Trigger, Prozeduren oder Anwendungen entwickelt werden müssen. Das erleichtert den Unternehmen auch die Umstellung auf neue Compliance-Regeln. Ein konsistentes Konzept für temporale Datenhaltung ermöglicht darüber hinaus die Arbeit mit deutlich weniger komplexen Abfragen und eine effektivere Analyse zeitabhängiger Vorfälle.

Die temporale Unterstützung in Db2 reduzierte den Programmieraufwand gegenüber den beiden eigenentwickelten Implementierungen um über 90 Prozent. Allein die Darstellung grundlegender temporaler Funktionen mit Hilfe von Prozeduren in SQL oder Java erforderte 16 bzw. 45 Mal so viele Programmzeilen wie die einfachen SQL-Anweisungen, die auf die temporalen Funktionen von Db2 zurückgreifen (siehe Abbildung 1). Die Entwicklung und Prüfung dieser wenigen Db2/SQL-Anweisungen dauerte weniger als eine Stunde.

Die Entwicklung und Prüfung der eigenentwickelten Lösungen hingegen nahm vier bis fünf Wochen in Anspruch, und beide Alternativen boten dann nur einen Teil der temporalen Unterstützung, die Db2 zur Verfügung stellt. So fehlten bei den Eigenlösungen temporale Konzepte wie System Time, bitemporale Daten, Views und viele andere Db2-Funktionen. Die Eigenentwicklung eines vergleichbaren Umfangs an temporalen Funktionen würde also wahrscheinlich Monate in Anspruch nehmen. Das steht in keinem Verhältnis zu dem geringen Aufwand, der für die Formulierung entsprechender Db2/SQL-Anweisungen nötig ist!

Beispielszenario

Ein einfaches Beispiel mit Musterdaten zeigt, welche Möglichkeiten die Funktionen zur temporalen Datenhaltung in Db2 bieten. Im folgenden Beispiel geht es um Policen

Abb. 1: Entwicklungskosten für die Implementierung einer zeitlichen Spezifikation von Datensätzen (Business Time)

für Kfz-Versicherungen, die der Einfachheit halber in einer einzelnen Tabelle dargestellt sind. Die Tabelle enthält verschiedene Informationen, die in solchen Policen in der Regel erfasst sind: Versicherungsnummer (**id**), Fahrgestellnummer (**vin**), voraussichtliche jährliche Kilometerleistung (**annual_mileage**), Kostenübernahme für Leihwagen bei schadensbedingter Fahrzeugreparatur (**rental_car**) und die Deckungssumme (**coverage_amt**), einschl. Sachschäden, Heilkosten usw. Die Abbildung 2 zeigt den Aufbau der Tabelle **POLICY** ohne temporale Unterstützung.

Wie können die temporalen Funktionen von Db2 die Verwaltung solcher Versicherungspolicen erleichtern?

Datenverwaltung mit System Time

Das Konzept System Time in Db2 bietet die Möglichkeit, die zeitliche Entwicklung von Daten zu verfolgen und darzustellen. Wird eine Tabelle mit einem Bearbeitungszeitintervall (System Time Period) erzeugt, erfasst Db2 sämtliche Änderungen an dieser Tabelle automatisch und speichert „alte" Sätze in einer separaten History-Tabelle mit identischer Struktur. Werden zeitbezogene Abfragen an die Tabelle gerichtet, greift Db2 bei Bedarf transparent auf die automatisch gepflegte History-Tabelle zu. Diese Funktion ermöglicht den einfachen Umgang mit historischen Daten und macht komplexe Abfragen mit **WHERE**-Klauseln, diversen Zeitstempeln und **JOIN**-Bedingungen überflüssig.

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt
1111	A1111	10000	Y	500000

Abb. 2: Die Ausgangstabelle POLICY

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end	trans_start

Abb. 3: Die Tabelle POLICY (mit aktuellen Daten)

POLICY_HISTORY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end	trans_start

Abb. 4: Die Tabelle POLICY_HISTORY (mit historischen Daten)

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2015-11-15	9999-12-31
1414	B7777	14000	N	750000	2015-11-15	9999-12-31

Abb. 5: Inhalt der aktuellen Tabelle nach den INSERTs am 15.11.2015

POLICY_HISTORY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end

Abb. 6: Inhalt der History-Tabelle nach den INSERTs am 15. November 2015

Definition einer Tabelle mit System Time

Das Definieren einer Tabelle mit System Time erfolgt in drei einfachen Schritten:

1. **Basistabelle für aktuelle Daten definieren**
   Zusätzlich zu den eigentlichen Spalten einer Tabelle werden drei **TIMESTAMP**(12)-Spalten angelegt, zwei für die Start-/Endzeitpunkte der Bearbeitungszeit und eine für den Zeitpunkt des Transaktionsstarts. Db2 nutzt diese Spalte um festzustellen, wann die Transaktion erstmals eine Anweisung zur Änderung der Daten in der Tabelle ausführte. Die drei **TIMESTAMP**-Spalten können mit **GENERATED ALWAYS** definiert werden, damit Db2 die Werte bei jedem **INSERT**, **UPDATE** und **DELETE** automatisch erzeugt. Damit ist sichergestellt, dass die Werte stets in die Datenbank eingetragen werden und dass die Zeitstempel korrekt sind. Definiert man diese Spalten zusätzlich noch als **IMPLICITLY HIDDEN**, werden sie bei **SELECT***-Anweisungen nicht angezeigt.
2. **History-Tabelle definieren**
   Erzeugen einer Tabelle mit identischer Struktur, wie sie die Tabelle mit den aktuellen Daten aufweist. Dies lässt sich einfach mit der Anweisung **CREATE TABLE . . . LIKE** bewerkstelligen.
3. **Versionierung für die Basis-Tabelle aktivieren**

An einem Beispiel wird deutlich, wie einfach es mit Db2 ist, mit System Time automatisch verschiedene Versionen von Daten zu pflegen.

1. Schritt:
Erzeugen einer Tabelle mit einer SYSTEM_TIME-Periode - Definition der Tabelle POLICY

```
CREATE TABLE policy (
  id              INT primary key not null,
  vin             VARCHAR(10),
  annual_mileage  INT,
  rental_car      CHAR(1),
  coverage_amt    INT,
  sys_start       TIMESTAMP(12) GENERATED
ALWAYS AS ROW BEGIN NOT NULL,
  sys_end         TIMESTAMP(12) GENERATED
ALWAYS AS ROW END NOT NULL,
  trans_start     TIMESTAMP(12) GENERATED
ALWAYS AS TRANSACTION START ID
                  IMPLICITLY HIDDEN,
  PERIOD SYSTEM_TIME (sys_start, sys_end)
 );
```

2. Schritt: Erzeugen einer zugehörigen History-Tabelle

```
CREATE TABLE policy_history LIKE policy;
```

3. Schritt: Versionierung aktivieren

```
ALTER TABLE policy ADD VERSIONING USE
HISTORY TABLE policy_history;
```

Die Abbildungen 3 und 4 zeigen das mit o.g. Anweisungen erzielte Ergebnis.

Bereits vorhandene Tabellen können mit der Anweisung **ALTER TABLE** für die Verwaltung historischer Daten genutzt werden. Hierzu müssen die oben erwähnten Spalten **sys_start** und **sys_end** eingefügt und **PERIOD SYSTEM_TIME** definiert werden.

Db2 unterstützt automatische Schemaänderungen für zugehörige History-Tabellen, sofern für die Basistabellen wie in Schritte 1-3 beschrieben die Versionierung aktiviert ist. Wird also eine Spalte in die Tabelle **POLICY** eingefügt, erweitert Db2 die Tabelle **POLICY_HISTORY** ebenfalls um die gleiche Spalte.

Dateneingabe in eine Tabelle mit System Time

Das Einfügen von Daten in eine Tabelle mit System Time unterscheidet sich nicht von der Dateneingabe in herkömmliche Tabellen. Beispielsweise müssen am 15. November 2015 zwei neue Datensätze für Kfz-Versicherungen in die Tabelle POLICY eingegeben werden. Das geschieht mit folgenden Anweisungen:

```
INSERT INTO policy(id, vin, annual_mileage,
rental_car, coverage_amt)
   VALUES(1111, 'A1111', 10000, 'Y', 500000);
INSERT INTO policy(id, vin, annual_mileage,
rental_car, coverage_amt)
   VALUES(1414, 'B7777', 14000, 'N', 750000);
```

Beim Einfügen der neuen Zeilen in die aktuelle Tabelle erzeugt Db2 die Zeitstempel für die Spalten **sys_start** und **trans_start**. Diese Zeitwerte waren in den **INSERT**-Anweisungen nicht angegeben. Db2 hat sie automatisch eingefügt. Die Abbildungen 5 und 6 zeigen, wie die Tabellen **POLICY** und **POLICY_HISTORY** nach den Einfügungen aussehen. Zur Vereinfachung zeigen die beiden Abbildungen nur die Datumswerte der Spalten **sys_start** und **sys_end** an. Die Daten werden im Format YYYY-MM-DD angezeigt. Da die Spalte mit der Transaktionsstartzeit als verborgen definiert wurde, wird sie hier in den Abbildungen 5 und 6 nicht angezeigt.

Die Werte in der Spalte **sys_start** in der Tabelle **POLICY** geben an, wann die Sätze eingefügt wurden, im Beispiel am 15. November 2015. Die Werte für **sys_end** wurden auf den 30. Dezember 9999 gesetzt, um anzuzeigen, dass die Daten dieser Sätze noch aktuell sind.

Aktualisieren von Daten in eine Tabelle mit System Time

Werden Daten in der Tabelle aktualisiert, sichert Db2 automatisch eine Vorversion der Daten in der zugehörigen History-Tabelle. Dieser Vorgang läuft transparent ab und erfordert weder Programmierung noch Benutzereingriff. Angenommen, die folgende Anweisung wird am 31. Januar 2016 ausgeführt, um die Deckungssumme für Police 1111 in 750000 zu ändern:

```
UPDATE policy
    SET coverage_amt = 750000
    WHERE id = 1111;
```

Wie in Abbildungen 7 und 8 gezeigt, aktualisiert Db2 den Wert in der Zeile der aktuellen Tabelle. Dann verschiebt Db2 eine Kopie des alten Satzes in die History-Tabelle. In beiden Tabellen speichert Db2 darüber hinaus die Start- und Endzeiten für die Werte in diesen Zeilen. So wird der Wert für diese Zeile in der Spalte **sys_end** in der History-

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	750000	2016-01-31	9999-12-31
1414	B7777	14000	N	750000	2015-11-15	9999-12-31

Abb. 7: Inhalt der aktuellen Tabelle nach dem UPDATE vom 31.01.2016

POLICY_HISTORY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2015-11-15	2016-01-31

Abb. 8: Inhalt der History-Tabelle nach dem UPDATE vom 31.01. 2016

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	5000	N	250000	2017-01-31	9999-12-31
1414	B7777	14000	N	750000	2015-11-15	9999-12-31

Abb. 9: Inhalt der aktuellen Tabelle nach dem UPDATE vom 31.01.2017

POLICY_HISTORY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2015-11-15	2016-01-31

Abb. 10: Inhalt der History-Tabelle nach dem UPDATE vom 31.01.2017

POLICY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	5000	N	250000	2017-01-31	9999-12-30

Abb. 11: Inhalt der aktuellen Tabelle nach dem Löschen am 31.03.2017

POLICY_HISTORY

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2015-11-15	2016-01-31
1111	A1111	10000	Y	750000	2016-01-31	2017-01-31
1414	B7777	14000	N	750000	2015-11-15	2017-03-31

Abb. 12: Inhalt der History-Tabelle nach dem Löschen am 31.03.2017

Abfragen an eine Tabelle mit System Time

Tabelle auf den Transaktionszeitpunkt der **UPDATE**-Anweisung gesetzt. All dies geschieht automatisch und transparent für den Anwender. Db2 speichert auch die Transaktionsstartzeit in den Abbildungen 7 und 8, auch wenn das hier nicht angezeigt wird.

Alle weiteren Aktualisierungen werden natürlich auf ähnliche Weise abgewickelt. Angenommen, dass Police 1111 am 31. Januar 2017 in einigen weiteren Punkten wie der jährlichen Kilometerleistung, der Kostenübernahme für Leihwagen während schadensbedingter Fahrzeugreparatur und der Deckungssumme aktualisiert wird. Hier ist die entsprechende **UPDATE**-Anweisung:

```
UPDATE policy
  SET annual_mileage = 5000, rental_car='N',
coverage_amt = 250000
  WHERE id = 1111;
```

Bei der Ausführung dieser Anweisung modifiziert Db2 automatisch die Tabellen **POLICY** und **POLICY_HISTORY**, wie in Abbildungen 9 und 10 gezeigt.

Löschen von Daten aus einer Tabelle mit System Time

Bei einem Löschvorgang entfernt Db2 die jeweiligen Daten aus der aktuellen Tabelle und überträgt eine Kopie der gelöschten Daten in die zugehörige History-Tabelle. Db2 setzt den Endzeitpunkt der gelöschten Daten in der History-Tabelle auf den Transaktionsstartpunkt der **DELETE**-Anweisung. Dieser Vorgang läuft transparent ab und erfordert weder Programmierung noch Benutzereingriff.

Angenommen die Zeile mit den Daten für Police 1414 wird am 31. März 2017 durch folgende Anweisung gelöscht:

```
DELETE FROM policy WHERE id = 1414;
```

Db2 löscht den Satz in der Tabelle **POLICY** und fügt diesen in die History Tabelle ein (siehe Abbildungen 11 und 12). Die Spalte **sys_end** wird automatisch auf das aktuelle Datum, also auf den Wert 31. März 2017 gesetzt.

Abfragen an eine Tabelle mit System Time

Abfragen an eine Tabelle mit System Time lassen sich ganz einfach formulieren. Syntax und Semantik herkömmlicher **SELECT**-Anweisungen bleiben unverändert. So gelten diese Anweisungen ohne Zeitraumangaben stets für aktuelle Daten in der Basis-Tabelle. Vorhandene Anwendungen, Prozeduren und Datenbankberichte können also auch nach einer Erweiterung alter Tabellen durch das temporale Konzept weiter genutzt werden. In die **SELECT**-Anweisung lassen sich nun zusätzlich für den transparenten Zugriff auf historische Daten (oder eine Kombination aus aktuellen und historischen Daten) drei Zeitraumangaben aufnehmen.

Einige Beispiele zeigen, wie einfach sich zeitbezogene Abfragen formulieren lassen.

Die Informationen über Kfz-Versicherungen in der aktuellen und der History-Tabelle entsprechen den in Abbildungen 11 und 12. Nun wird folgende Abfrage gestellt:

```
SELECT coverage_amt
    FROM policy
    WHERE id = 1111;
```

Wie zu erwarten, gibt Db2 einen Satz mit einer Deckungssumme von 250.000 zurück.

Für Abfragen nach älteren Daten muss eine von drei unterstützten Zeitraumangaben in die **FROM**-Klausel aufgenommen werden:

- **FOR SYSTEM_TIME AS OF ...**
  gibt Daten ab einem bestimmten Zeitpunkt zurück.

  ```
  SELECT coverage_amt
    FROM policy FOR SYSTEM_TIME AS OF '2015-
  12-01'
    WHERE id = 1111;
  ```

- **FOR SYSTEM_TIME FROM ... TO ...**
  gibt Daten für einen bestimmten Zeitraum zurück. Db2 nutzt hier ein inclusive-exclusive-Konzept für diese Zeitraumangabe. Das bedeutet: Der angegebene Startzeitpunkt liegt innerhalb des Zeitraums, der Endzeitpunkt nicht.

  ```
  SELECT count(*)
   FROM policy FOR SYSTEM_TIME FROM '2016-11-30'
                                TO '9999-12-30'
   WHERE vin = 'A1111';
  ```

- **FOR SYSTEM_TIME BETWEEN ... AND ...**
  gibt Daten zurück, die zwischen bestimmten Start- und Endzeiten liegen. Db2 nutzt hier ein inclusive-inclusive-Konzept für diese Zeitraumangabe. Das bedeutet: Der angegebene Startzeitpunkt und der Endzeitpunkt liegen innerhalb des Zeitraums.

Fazit

Die temporalen Funktionen von Db2 ermöglichen auf einfache Weise die differenzierte Auswertung historischer Datensätze. Auf Grundlage der temporalen Erweiterungen des SQL:2011-Standards bietet Db2 eine innovative Umgebung für effektive temporale Datenhaltung, die im Vergleich mit selbst entwickelten Lösungen zur Abbildung temporaler Konzepte in Form von Triggern, Prozeduren oder selbst entwickelten Anwendungen ganz erhebliche Zeit- und Kostenvorteile bietet.

Stefan Hummel, Certified IT Specialist
IBM Deutschland

Quellen

[Q1] Cynthia M. Saracco, Matthias Nicola, Lenisha Gandhi:
„A matter of time: Temporal data management in DB2“

[Q2] Db2 auf IBM developerWorks (Technische Artikel, Software, Forum)
http://www.ibm.com/developerworks/data/products/db2

Einfach. Einfacher. Ansible.

Automatisierung mit Ansible

Schon in früheren Ausgaben wurde über Tools berichtet, welche eine automatisierte Administration ermöglichen. Diese waren BMC BladeLogic, Puppet und Ansible. Der Fokus lag dabei aber immer auf Unix-Systemen. In diesem Artikel wollen wir nun zeigen, dass sich auch Windows-Systeme automatisieren lassen.

Konzept

Bei Ansible wird eine Kontrollmaschine zentral eingesetzt, welche auf die entfernten Rechner zugreift und dort remote administrative Aufgaben erledigt. Dabei bilden sogenannte Module die Basis. Jedes Modul hat einen bestimmten Verwendungszweck. Zum Beispiel gibt es Module für den Zugriff auf das Dateisystem, damit Dateien erstellt oder gelöscht werden können. Zurzeit existieren ca. 800 Module, welche sich in unterschiedliche Kategorien aufteilen. Davon sind ca. 40 für Windows-Systeme konzipiert.

Installation

Da Ansible in Python geschrieben ist, kann jedes System, welches Python 2.6 oder 2.7 installiert hat, als Kontrollmaschine verwendet werden. Ausgenommen ist Windows. Der Support für Python 3 ist noch nicht umgesetzt. Derzeit existiert nur eine technische Preview. Für die Installation auf der Kontrollmaschine gibt es mehrere Wege. Ein Weg geht über die Paket-Repositories der Linux- bzw. Unix-Systeme.

Für Max OSX oder Systeme, die Ansible in ihren Paketquellen nicht enthalten haben, wird Pip empfohlen. Pip ist ein Paketmanager für Python-Pakete. Ein weiterer Weg geht über den Quellcode. Dieser muss heruntergeladen werden und diverse Abhängigkeiten müssen selbst nachinstalliert werden.

Konfiguration

Auf der Kontrollmaschine gibt es drei Konfigurationsschritte, die durchgeführt werden müssen. Der erste ist das Eintragen der entfernten Rechner, die administriert werden sollen. Dies geschieht z.B. in der Datei **`/etc/ansible/hosts`** und werden Inventory genannt. Dort können einzelne IP-Adressen, Servernamen oder auch Gruppen eingetragen werden.

Gruppen werden durch die eckigen Klammern gekennzeichnet. Darunter werden dann die Rechner eingetragen. Neben der Gruppen ist es auch möglich durch sogenannte Pattern-Rechner zu adressieren (siehe Abbildung 1).

Der zweite Schritt ist das Hinterlegen der Nutzerdaten der Windows-Systeme. Diese braucht Ansible, damit es sich mit den Systemen verbinden kann. Dabei werden Benutzername, Passwort, Port und die Verbindungsmethode benötigt. Die Daten können in der Datei **`/etc/ansible/group_vars/windows.yml`** hinterlegt werden. Dabei muss die Datei den Namen der Gruppe mit den Windows-Systemen haben (siehe Abbildung 2).

Der dritte Schritt, sofern benötigt, ist die Konfiguration von Ansible selbst. In der Datei können zum Beispiel der Pfad des Inventory angepasst, die temporären Verzeichnisse geändert und der Pfad zu den Ansible-Bibliotheken bestimmt werden. Die Konfigurationsdatei von Ansible kann entweder unter **`/etc/ansible`** liegen oder im Home-Verzeichnis des Benutzerkontos, welches Ansible nutzen soll.

Auf den Windows-Systemen, welche administriert werden sollen, muss ein Powershellskript ausgeführt werden, welches von Ansible bereitgestellt wird [3]. Dieses Skript richtet Windows für die Administration mit Ansible ein. Dabei wird der Windows Remote Management Dienst (WinRM) aktiviert und die notwendige Version der

```
192.168.56.101
email.example.com
server[01:50].example.org

[windows]
192.168.56.110
```

Abb. 1: Eintrag in der /etc/ansible/hosts

```
ansible_user: ordix
ansible_password: ordix9876
ansible_port: 5986
ansible_connection: winrm
ansible_winrm_server_cert_validation: ignore
```

Abb. 2: Eintrag in der /etc/ansible/group_vars/windows.yml

```
192.168.56.110 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Abb. 3: Ausgabe des Moduls win_ping

```
- name: Some tasks with ansible
  hosts: windows
  tasks:
    - name: Create a temp directory
      win_file:
         path='C:\temp' state=directory
    - name: Create a registry entry
      win_regedit:
         key: HKCU:\Software\ORDIX
         value: Windows
         data: 10
    - name: Restart spooler service
      win_service:
         name: spooler
         state: restarted
```

Abb. 4: Inhalt von example.yml

```
PLAY [Some tasks with ansible] ************************
TASK [Create a temp directory] ************************
changed: [192.168.56.110]
TASK [Create a registry entry] ************************
changed: [192.168.56.110]
TASK [Restart spooler service] ************************
changed: [192.168.56.110]

PLAY RECAP ********************************************
192.168.56.110  : ok=3 changed=3 unreachable=0 failed=0
```

Abb. 5: Ausgabe von example.ymlk

Links/Quellen

[1] ORDIX® news Artikel 02/2013 –
„BMC BladeLogic Server Automation – Homo(gen) oder Hetero(gen)?“
https://www.ordix.de/ordix-news-archiv/2-2013.html

[2] ORDIX® news Artikel 01/2016:
„Konfigurationsmanagement mit Puppet“
https://www.ordix.de/ordix-news-archiv/1-2016.html

[3] Powershellskript zur Einrichtung von Ansible-Remote-Management
https://github.com/ansible/ansible/blob/devel/examples/scripts/ConfigureRemotingForAnsible.ps1

Christopher Herclik
(info@ordix.de)

Powershell geprüft. Zusätzlich werden Zugriffe via WINRM auf den Windows Host erlaubt.

Kommandozeile

Um Ansible zu nutzen kann die Kommandozeile verwendet werden. Als Beispiel sei hier ein Aufruf gezeigt, welcher überprüft, ob die Remote-Systeme online sind:

```
ansible windows –m win_ping
```

Der Aufruf beginnt mit dem Kommando, gefolgt von der IP-Adresse oder auch hier die Gruppe **windows** aus Abbildung 1. Danach wird das Modul angegeben, welches genutzt werden soll. Der Aufruf gibt in diesem Fall eine positive Rückmeldung (siehe Abbildung 3).

Ein anderes Beispiel sei hier das Erstellen einer Datei:

```
ansible windows –m win_file –a "path=C:\temp\file.txt"
```

Dieser Aufruf erstellt eine Datei in einem Ordner **temp** unter dem Laufwerk C. Es können den Modulen auch Argumente übergeben werden. In diesem Fall der Pfad der Datei mit dem Dateinamen.

Playbooks

Für die automatisierte Ausführung einzelner administrativer Schritte existieren sogenannte Playbooks. Diese werden in der YAML-Syntax erstellt. In Abbildung 4 ist ein Beispiel zu sehen. Es wird ein Verzeichnis erstellt, die Registry bearbeitet und der Spooler-Dienst neu gestartet. Der Aufruf, sofern der Inhalt in der Datei **example.yml** abgespeichert ist, lautet:

```
ansible-playbook example.yml
```

Bei dem Aufruf werden die einzelnen Schritte nacheinander abgearbeitet. Bei der Ausgabe wird der Endstatus von jeder Aufgabe ausgegeben. Es gibt drei mögliche Ausgaben: **changed**, **ok** und **failed**. **Changed** bedeutet, dass etwas geändert wurde. Zum Beispiel beim Erstellen des Ordners **temp**. **Ok** tritt auf, wenn nichts geändert wurde und kein Fehler entstanden ist. Dies wäre der Fall, wenn der Ordner **temp** schon existieren würde. **Failed** wird ausgegeben, wenn ein Fehler aufgetreten ist. Nachdem alle Aufgaben abgearbeitet wurden, wird eine Zusammenfassung pro Rechner ausgegeben (siehe Abbildung 5).

Fazit

Wie gezeigt werden konnte, ist das Einrichten und Bedienen von Ansible kein großer Aufwand und nicht schwer. Auch wenn der Anteil der Windows-Module noch nicht sonderlich hoch ist, so kann schon viel mit Ansible automatisiert werden. Mehr erfahren Sie in unserem Blog und in unserem Seminar zu Ansible.

Neuheiten Java 9 (Teil I)

Java 9 - Was lange währt

Wer kann sich noch erinnern? Im März 2014 kam das letzte Java Major Release Java 8 und es gab sehr viel Neues, wobei sich vor allem Lambda-Ausdrücke und die mächtige Stream-API hervortaten. Am 21. September 2017 kam nun endlich das mit Spannung erwartete Java 9 heraus, um mit der Hauptneuerung „Modularisierung" aufzuwarten.

Was bringt Java 9, warum hat es so lange gedauert und wie können wir am besten vom neuen Release profitieren? Das wollen wir hier und in weiteren Artikeln zum Thema machen. Zunächst fokussieren wir uns auf das folgenreichste neue Feature: Die Modularisierung (läuft bei Oracle unter dem Codenamen „Jigsaw"). Sie geht verschiedene Problematiken im Java-Umfeld an, die sich grob in die Kategorien Sicherheitslücken, JAR-Hell und Spaghetti-Projekte einsortieren lassen.

Modularisierung – Gab es das nicht schon vorher?

Modularisierung in Java gab es in gewisser Weise schon immer, aber diese beschränkt sich vorwiegend auf Quellcode-Ebene. Jeder kennt die Einteilung in Klassen, die – grob gesprochen – aus einem öffentlich zugänglichen Teilbereich (**`public`**) und einen verborgenen, privaten (**`private`**) bestehen. Klassen und Interfaces sind ihrerseits wieder in Packages angeordnet, die ebenfalls bezogen auf Sichtbarkeit steuerbar sind (**`package-visibility`**). Das ist soweit schön und gut, in der Praxis fehlen aber noch ganz wichtige Zutaten. Wir kommen später noch genauer auf **`public`** und die neuen, damit einhergehenden Einschränkungen zu sprechen, die durch Jigsaw Einzug in das Java-Ökosystem halten.

Leider müssen wir nämlich schmerzlich akzeptieren, dass diese Aufteilungs- und Strukturierungsmöglichkeiten mit dem **`build`**-Vorgang schlagartig verloren gehen, denn typischerweise gelangt compilierter Bytecode in Archivdateien (**`*.jar`**, **`*.war`**). Deren Namensgebung ist weitgehend beliebig. Diese Dateien werden deployed oder allgemeiner gesprochen ausgeliefert und sie repräsentieren häufig die eigentliche Applikation. Zur Laufzeit werden nun die Klassen und Interfaces aus den Archivdateien vom Classloader geladen und die Gesamtheit aller geladenen Java Konstrukte machen die lauffähige Anwendung aus.

Die Problematik resultiert aus mangelnden Vorgaben und Regelungsmöglichkeiten zum Aufbau und zur Struktur von solchen Archivdateien. In einer JAR-Datei können prinzipiell beliebig viele und beliebig abhängige Java-Klassen untergebracht sein. Sobald sie geladen sind, geht die Information, aus welcher JAR-Datei sie stammen, verloren.

Die JAR-Hell- und Security-Einfallstore

Worin besteht denn nun genau das Problem und was haben JAR-Dateien damit zu tun? Es kommt zu einem großen Bruch beim Übergang von Quellcode zum Binärformat, weil es die Eigenschaft der Kapselung so nicht bei Java-Artefakten gibt (wie man JAR-Dateien auch nennt). Daraus folgt zum Beispiel, dass sich sensibler Code nur schlecht schützen lässt sobald er im Binärformat vorliegt. Und das Binärformat kommt praktisch überall vor, selbst das JDK (Java Devolpment Kit) ist im Wesentlichen nichts anderes als eine Ansammlung von Archivdateien (z.B. **`$JRE_HOME/lib/rt.jar`**, welches die Java-Kernklassen beinhaltet).

Diese mangelnde Schutzmöglichkeit musste selbst Oracle schmerzlich erfahren, als mit dem Erwerb von Java und der Auslieferung von Java 7 eine Vielzahl von Sicherheitslücken offenkundig wurden und es zahlreicher Patches bedarf, diese zu schließen. Denn sicherheitskritische Codestellen waren in den JDK-Archiven mit relativ einfachen Mitteln (Stichwort **`Reflection`**) lesbar, untersuchbar und schlimmstenfalls manipulierbar. Schadsoftware hatte dadurch relativ leichtes Spiel und leichten Zugang zu Java-Kernklassen. Das lag u.a. an der einfachen Möglichkeit, die **`$JRE_HOME/lib/rt.jar`** und andere JAR-Dateien mit einem prinzipiellen Zugriffsschutz auszustatten, was dann auch von der kriminellen Hackerszene weidlich ausgenutzt wurde.

Module weisen den Weg ins Glück

Das konnte so nicht weitergehen, und Oracle hat einen sehr langen Weg hin zu einer Modularisierung des

gesamten JDK beschritten. Dieser dauerte 5 Jahre bis zur Version Java 9. Das Ergebnis ist eine komplett geänderte Dateistruktur des JRE (Java Runtime Environment), was z.B. an dem Wegfall der Datei **`$JRE_HOME/lib/rt.jar`** erkennbar wird. Statt einer Handvoll übervoller JAR-Dateien, haben wir es ab Java 9 mit einem Modulsystem von über 100 Modulen zu tun.

Zur erheblichen Verzögerung von Java 9 hat auch die durchaus kontrovers geführte Diskussion um das richtige Konzept zur Modularisierung beigetragen. Kurz vor dem eigentlich geplanten Releasetermin Juli 2017 verweigerten Red Hat und IBM ihre Zustimmung zum Jigsaw-Vorschlag von Mark Reinhold, dem Java-Chefarchitekten von Oracle. Das Konzept wurde nochmals dahingehend überarbeitet, um einen „weicheren" Migrationspfad von alten Java-Anwendungen zu Java 9 zu ermöglichen, wodurch dem Erscheinungstermin am 21. September 2017 nichts mehr im Wege stand.

Was hat es denn nun mit Jigsaw auf sich? Der Vorgang der Modularisierung erzeugt Module, und diese zeichnen sich durch drei Eigenschaften aus, welche die Praxistauglichkeit ausmachen. Ein Modul hat:

- Einen Namen oder eine eindeutige ID
- Abhängigkeiten: Was braucht mein Modul, wovon bin ich abhängig?
- Nutzbare API: Was kann ich anbieten, wie sieht die öffentliche Schnittstelle aus?

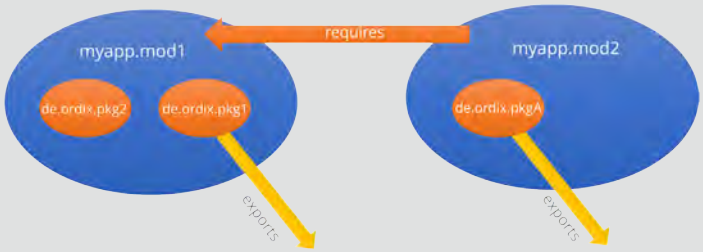

Abb. 1: Grundzüge der Modulabhängigkeiten. Packages stellt man per exports zur Verfügung. Mittels requires drückt man den Bedarf an anderen Modulen aus

```
module myapp.mod1 {
    requires mod2;
    exports de.ordix.pkg1;
    exports de.ordix.pkg2;
}
module myapp.mod1 {
    exports de.ordix.pkgA;
}
```

Abb. 2: Die Inhalte von 2 Moduldeskriptoren, abgelegt in den jeweiligen Verzeichnissen in den Deskriptor Dateien modul-info.java

```
src/myapp.mod1/
   module-info.java
   de.ordix.pkg1/
      ClassA.java
      ClassB.java
...
   de.ordix.pkg2/
      ClassC.java
```

Abb. 3: Verzeichnis eines Moduls mit der Deskriptor Datei auf oberster Ebene und den Packages, die sich im Modul befinden

Wichtige Konzepte des JPMS

Das Modulsystem von Java 9 (JPMS – Java Platform Module System) eröffnet den Entwicklern die Möglichkeit, Artefakte (JAR-Dateien) als Module und die oben erwähnten drei Eigenschaften zu definieren. Module stellen damit eine vollkommen neue, zusätzliche Strukturierungsmöglichkeit in Java 9 zur Verfügung. Bislang waren es Packages, die die größtmögliche Klammer bildeten, unter der sich zusammengehörige Klassen, Interfaces, Enums, etc. gruppierten.

Im neuen Java 9 kommen noch Module dazu, die viele alte Ordnungsprinzipien und Regeln neu ausrichten. Ein Modul ist ein abgeschlossener „Behälter", in dem sich zusammenhängende Klassen, Properties und nativer Code versammeln und Abhängigkeiten von Codeteilen drücken sich auf Modulebene aus (siehe Abbildung 1). Ein Modul-Deskriptor beschreibt diese Abhängigkeiten, gehört damit ebenfalls zum Modulinhalt und befindet sich im **`root`**-Verzeichnis des Modul-Directory.

Beispielhaft zu sehen ist das im kurzen Codeblock in der Abbildung 2. Dort ist das Modul mit dem Namen **`myapp.mod1`** beschrieben (**`identity`**, Eigenschaft 1), zusammen mit der Abhängigkeit zu einem anderen Modul **`myapp.mod2`** (**`requires`**, Eigenschaft 2) und mit einem eigenen zur Verfügung gestellten Paket **`de.ordix.pkg1`** (**`exports`**, Eigenschaft 3). Das ist im Übrigen kein Rechtschreibfehler, **`requires`** erfordert Module, **`exports`** stellt Packages zur Benutzung bereit.

Wo aber hat denn so ein Codeblock zu stehen? Es handelt sich schließlich um nichts Geringeres als eine Moduldefinition. Genau das ist es auch und dafür wird eine spezifische Source-Datei mit dem Namen **`module-info.java`** vorgeschrieben (siehe Abbildung 3). Es handelt sich um den zuvor erwähnten Modul-Deskriptor, den der Compiler wie jede andere Java-Source transformiert und zwar zu **`module-info.class`**.

Die Wirkungsweise von JPMS

Um die Vorteile von JPMS erkennen zu können, müssen wir einen genauen Blick auf dessen Bedeutung und Semantik werfen. Das Schlüsselwort **`public`** hat früher signalisiert: Die Klasse oder das Interface ist global im Anwendungskontext nutzbar, sofern das zugehörige Package per **`import`** herangezogen wurde.

Im neuen Java 9 heißt **`public`**: Eine so spezifizierte Klasse ist nur innerhalb des Moduls von überall ansprechbar. Von

einem anderen Modul nur genau dann, wenn das zugehörige Package mit der Klasse per **`exports`** freigegeben wird und das andere Modul es per **`requires`** anfordert. Das ist revolutionär, denn es bringt – endlich muss man sagen – echten Zugriffsschutz. Eine **`public`**-Klasse, die nicht exportiert wird, kann von außerhalb des Moduls nicht angesprochen werden, auch nicht über Reflection-Mechanismen!

Fazit

Java 9 ist nach langer Zeit nun endlich da und wir haben die wichtigste Neuerung unter die Lupe genommen: Modularisierung auf Artefaktebene. Diese Möglichkeit erlaubt dem Entwickler, auf JAR-Dateien bzw. auf dem Auslieferungspaket einer Java-Anwendung die wichtigen Prinzipien der „Strong Encapsulation“ und „Reliabale Configuration“ anzuwenden. Dadurch gelingt es zum einen aus der JAR-Hell auszubrechen und zum anderen, potenzielle Sicherheitslücken zu vermeiden. Wie schon angesprochen umfasst Java 9 eine Vielzahl von Neuerungen, auf die wir in zukünftigen Artikeln eingehen wollen. Selbstverständlich wird es in Kürze eine Seminarveranstaltung zu „Java 9 Neuheiten“ bei der ORDIX AG geben. Dort vermitteln wir Ihnen als Seminarteilnehmer sämtliche Neuerungen zu Java 9.

Dr. Hubert Austermeier
(info@ordix.de)

Glossar

JPMS
Das Java Platform Modular System ist die neue Strukturierungsmöglichkeit auf Ebene von JAR-Dateien.

Links/Quellen

[1] Java-Seminare bei der ORDIX AG
https://seminare.ordix.de

[2] Download des Java 9 JDK
http://openjdk.java.net/projects/jdk9/

[3] Modulare Java-Zukunft
https://www.heise.de/developer/artikel/Modulare-Java-Zukunft-Das-Java-Platform-Module-System-erklaert-3700766.html

Neuerungen in der Oracle Database 12.2 (Teil I)

SQL- und PL/SQL-Neuerungen in Oracle 12.2

In diesem ersten Artikel der neuen Reihe stellen wir Ihnen die wesentlichen Neuerungen von Oracle 12.2 im SQL- und PL/SQL-Bereich vor. Wie auch in den vergangenen Releases hat Oracle den Funktionsumfang von SQL und PL/SQL um neue Funktionalitäten mit dem Ziel erweitert, die Programmierung zu vereinfachen, weniger fehleranfällig zu machen und die PL/SQL-Programme sicherer zu gestalten. In diesem Release ist unter anderem die Aufhebung der fast schon historischen Grenze von 30 Bytes bei Objektnamen hervorzuheben, mit deren Hilfe sprechende Namen bei Datenbankobjekten ohne Abkürzungen möglich sind. Darüber hinaus hat Oracle einige Erweiterungen eingeführt, mit deren Hilfe bei einigen Anwendungsfällen weniger Code individuell programmiert werden muss.

Lange Objektnamen

Datenbankobjektnamen wie Tabellen, Spalten, Stored Procedures oder Variablen in PL/SQL waren lange Zeit bis Oracle 12.1 auf 30 Bytes begrenzt. Ab Oracle 12.2 ist die Begrenzung auf 128 Bytes angehoben worden. Der wesentliche Vorteil der langen Schreibweise liegt vor allem in der Verwendung von aussagekräftigen Namen und in der Verbesserung der Lesbarkeit und Verständlichkeit des Code. So können z.B. sprechende Namen mit Beschreibung des Zweckes für Datenbankobjekte wie Fremdschlüssel, Indizes und Constraints verwenden werden, ohne diese abkürzen zu müssen.

Eine Ausnahme bildet hier immer noch der Datenbankname (8 Bytes) und folgende Objektnamen mit 30 Bytes Begrenzung:

```
create table tb_company (
  name    varchar2(30),
  name_ci varchar2(30) collate binary_ci
);
insert into tb_company values('ORDIX','ORDIX');
insert into tb_company values('Ordix','Ordix');
insert into tb_company values('ORdiX','ORdiX');
commit;
SQL> select * from tb_company where name_ci='ordix';
NAME       NAME_CI
---------- ----------
ORDIX      ORDIX
Ordix      Ordix
ORdiX      ORdiX
SQL> select count(distinct name)    cnt_name,
  2         count(distinct name_ci) cnt_name_ci
  3  from tb_company;
  CNT_NAME CNT_NAME_CI
---------- -----------
         3           1
```

Abb. 1: Groß- /Kleinschreibung Insensitivität auf Tabellenspaltenebene

- Disk Group
- Pluggable Database (PDB)
- Rollback Segment
- Tablespace
- Tablespace Set

Werden lange Objektnamen (> 30 Bytes) verwendet, so sollten vor allem bestehende Anwendungen bezüglich eventuellem Werteüberlauf der Variablen bei der Protokollierung, dynamischem SQL und festkodierten Variablenlängen überprüft werden. Um den Übergang auf die langen Objektnamen zu vereinfachen, hat Oracle zum einen eine Konstante **`ORA_MAX_NAME_LEN`** und eine Funktion **`ORA_MAX_NAME_LEN_SUPPORTED`** implementiert. Beide liefern die maximal unterstützte Länge für Objektnamen in einer Oracle-Version und können in einem PL/SQL-Programm abgefragt werden, um einen Variablenwerteüberlauf zu vermeiden.

Case insensitive Datenbank

Für den Vergleich und Sortierung von Zeichen (engl. collation) stellt Oracle zwei Vergleichstypen zur Verfügung: binär und linguistisch. Bei dem binären Vergleich werden die Zeichen nach der numerischen Reihenfolge in der entsprechenden Zeichensatztabelle sortiert bzw. miteinander verglichen. Der linguistische Zeichenvergleich dagegen sortiert die Zeichen nach der alphabetischen Reihenfolge der entsprechenden Sprache und ist in der Regel aufwändiger als ein binärer Zeichenvergleich. Zusätzlich kann der Vergleich der Zeichen bzgl. Groß- und Kleinschreibung und Akzent unempflindlichkeit (Insensitivität) beeinflusst werden, z.B.:

- **`BINARY_CI`** - binary case insensitive
- **`BINARY_AI`** - binary accent and case insensitive
- **`BINARY_CS`** - binary case and accent sensitivity

Bis Oracle 12.1 konnten die Collation-Einstellungen nur auf der Session- und SQL-Ebene eingestellt werden, z.B. über die Initialisierungsparameter **NLS_COMP** und **NLS_SORT**. Ab Oracle 12.2 kann der Vergleich von Zeichen zusätzlich auf der Schema- und Objektebene wie Tabellen, Views und Stored Procedures und sogar auf der Spaltenebene festgelegt werden (siehe Abbildung 1). Die Voraussetzung dafür ist die Aktivierung der Extended Datatypes (**MAX_STRING_SIZE=EXTENDED**), die allerdings nicht rückgängig gemacht werden kann und aus diesem Grund nur nach ausführlichen Tests zu empfehlen ist.

Durch die Möglichkeit der Anpassung von Collation zum Beispiel auf der Tabellenspaltenebene, kann das Verhalten einer Applikation dediziert pro Tabellenspalte bzgl. Groß- und Kleinschreibung und Akzent Insensitivität verändert werden, ohne den Code inkl. der SQL-Anweisungen anpassen zu müssen. Die Anpassungen von Collation auf der Tabellenebene sind für die Applikation dabei völlig transparent und eine Anpassung der Anwendung wegen der nicht-sensitiven Datenverarbeitung mit den typischen Funktionen wie **UPPER** oder **LOWER** ist hierbei nicht notwendig. Zu berücksichtigen ist allerdings, dass dabei die **WHERE**-Klausel der SQL-Anweisungen um den Ausdruck **nlssort** (**<column>,'nls_sort=<collation_type>'**) erweitert wird und dadurch in der Regel ein entsprechender funktionsbasierter Index notwendig wird.

Fehlertoleranz

Die Konvertierungsfunktionen wie **TO_NUMBER**, **TO_DATE** oder **TO_TIMESTAMP** wurden um die Übergabe eines Default-Wertes erweitert, der im Fehlerfall statt einer Fehlermeldung zurückgeliefert wird. Damit kann eine Verarbeitung trotz eines Konvertierungsfehlers fortgesetzt werden, ohne dass eine Fehlermeldung geworfen und die Abarbeitung eines SQL-Statements abgebrochen wird (siehe Abbildung 2). Individuelle Lösungen, die in der Praxis häufig zu finden sind, werden damit überflüssig. Soll lediglich nur überprüft werden, ob eine Datentypkonvertierung erfolgreich sein wird, kann die neue Funktion **VALIDATE_CONVERSION** verwendet werden. Diese Funktion liefert den Wert 1 falls eine Konvertierung möglich ist und den Wert 0 falls eine Konvertierung fehlschlägt (siehe Abbildung 3).

Auch die Funktion **LISTAGG**, mit der eine mit Trennzeichen getrennte Liste von Werten einer Spalte erzeugt werden kann, wurde um einen Default-Wert mit einer Überlaufbehandlung erweitert. Der Default-Wert tritt bei dieser Funktion dann in Kraft, wenn die maximale Länge von 4.000 Bytes überschritten wird. Zusätzlich zu dem Default-Wert kann auch noch ein Zähler ausgegeben werden, der die Anzahl überzähliger Zeichen beinhaltet. Die überschüssigen Zeichen können allerdings nicht ausgegeben und auch nicht gespeichert werden. Da ein Werteüberlauf bei der Funktion **LISTAGG** standardmäßig zu einem Fehler und somit zu einem Abbruch der Verarbeitung führt, ist die Erweiterung um den Default-Wert mit der Überlaufbehandlung sehr hilfreich, weil man damit die potenziellen Fehlerquellen in der Verarbeitung eliminieren kann.

```
with data (tag) as (
  select '10.01.2017' from dual union all
  select '30.02.2017' from dual union all
  select 'text'       from dual
) select tag, to_date(
    tag default '01.01.2017' on conversion error,
    'dd.mm.yyyy') as tag_def
from data;
TAG        TAG_DEF
---------- ----------
10.01.2017 10.01.2017
30.02.2017 01.01.2017
text       01.01.2017
3 Zeilen ausgewählt.
```

Abb. 2: Beispiel mit Default bei Konvertierungsfunktionen

```
with data (tag) as (
  select '10.01.2017' from dual union all
  select '30.02.2017' from dual union all
  select 'text'       from dual
) select tag,
  validate_conversion(tag as date, 'dd.mm.yyyy') as conv_ok
from data;
TAG           CONV_OK
---------- ----------
10.01.2017          1
30.02.2017          0
text                0
```

Abb. 3: Beispiel mit der Funktion VALIDATE_CONVERSION

```
SQL> -- Befehlshistorie für max. 100 Befehle aktivieren...
SQL> set hist on
SQL> -- SQL-Anweisung ausfuehren...
SQL> select 'test1' from dual;
'TEST
-----
test1
SQL> -- Inhalt der Befehlshistorie anzeigen...
SQL> hist
  1  select 'test1' from dual;
SQL> -- 1. Befehl aus der Befehlshistorie ausführen...
SQL> hist 1 run
'TEST
-----
test1
```

Abb. 4: Beispiel mit SQL*Plus-Befehlshistorie

```
-- Warnungen aktivieren...
alter session set plsql_warnings='enable:all';
-- Stored Procedure mit pragma deprecate anlegen...
create or replace procedure p_test is
  pragma deprecate( p_test,
    'p_test is deprecated please use p_test_2 instead.' );
begin
   dbms_output.put_line('p_test');
end p_test;
/
show err
LINE/COL   ERROR
--------   -----------------------------------------------------
3/3        PLW-06019: Entity P_TEST ist veraltet
create or replace procedure p_test2 is
begin
  p_test;
end p_test2;
/
show err
LINE/COL   ERROR
--------   -----------------------------------------------------
3/3        PLW-06020: Referenz auf eine veraltete Entity: P_TEST
           deklariert in Einheit P_TEST[1,11]. p_test is depreca-
           ted please use p_test_2 instead.
```

Abb. 5: Beispiel mit Pragma Deprecated

```
declare
  c1  constant varchar2(2) := 'AB';
  c2  constant number( length(c1) + 2 ) := 1234;
  v_proc_name  varchar2( (ORA_MAX_NAME_LEN*2)+1 );
begin
  v_proc_name := 'ora00.pro_mitarbeiter';
end;
/
PL/SQL-Prozedur erfolgreich abgeschlossen.
```

Abb. 6: Beispiel mit statischen Ausdrücken in Variablendeklarationen

```
create or replace package pck_ma as
  function  f_create(p_name varchar2) return number
    accessible by (p_main);
  procedure p_delete(p_nr number)
    accessible by (p_main_priv);
end pck_ma;
/
Package wurde erstellt.
```

Abb. 7: Beispiel mit der ACCESSIBLE-BY-Klausel

SQL*Plus-Befehlshistorie

Das SQL*Plus-Programm wurde um die langersehnte Befehlshistorie erweitert. Befehle die innerhalb einer Session eingegeben und ausgeführt wurden, werden innerhalb der Session zwischengespeichert und können über die Angabe der Nummer aus der Befehlshistorie einfach erneut ausgeführt werden (siehe Abbildung 4).

Standardmäßig ist die Befehlshistorie allerdings nicht aktiv. Erst nach der Angabe von **SET HIST[ORY] ON** wird die Befehlshistorie für 100 (maximal 100.000) Befehle aktiv geschaltet. Da jede SQL*Plus Session eine eigene Befehlshistorie hat, können Befehle von einer vorangegangenen SQL*Plus Session nicht verwendet werden.

Pragma Deprecated

Soll ein PL/SQL-Objekt nicht mehr verwendet werden oder durch ein anderes PL/SQL-Objekt ersetzt werden, so kann dieses mit der neuen Pragma Deprecated (zu Deutsch: veraltet) gekennzeichnet werden. Die Pragma Deprecated sorgt dafür, dass ein Nutzer einer mit der Pragma Deprecated markierten PL/SQL-Einheit bereits schon zum Kompilierzeitpunkt eine Warnung bekommt, dass diese PL/SQL-Einheit veraltet und somit nicht mehr genutzt werden soll. Darüber hinaus kann die Pragma Deprecated noch mit einem Meldungstext versehen werden, um den Nutzer z.B. auf ein anderes PL/SQL-Objekt hinzuweisen, welches stattdessen verwendet werden soll. Das Pragma Deprecated kann für Stored Procedures, aber auch für Variable, Exceptions und Cursor innerhalb von Stored Procedures genutzt werden. Nicht anwendbar ist die Pragma allerdings auf anonymen PL/SQL-Blöcken und verschachtelte PL/SQL-Prozeduren oder Funktionen (siehe Abbildung 5).

Statische Ausdrücke in Variablendeklarationen

Auch im Bereich von Variablendeklarationen in einem PL/SQL-Programm hat Oracle eine Verbesserung eingeführt. Im Wertebereich von Variablen-, Konstanten- und **SUBTYPE**-Deklarationen können ab diesem Release statische Ausdrücke verwendet werden. Ein statischer Ausdruck ist ein Ausdruck, der zum Kompilierzeitpunkt bestimmt sein muss, wie es z.B. bei einer Konstanten oder einem Ausdruck wie „1+1“ der Fall ist (siehe Abbildung 6). Variablen und Funktionsaufrufe sind keine statischen Ausdrücke und können im Wertebereich von Variablendeklarationen nach wie vor nicht verwendet werden. Mit Hilfe dieser Erweiterung können festkodierte Längenangaben bei Variablen und Konstanten vermieden werden.

Feingranulare PL/SQL-ACCESSIBLE-BY-Klausel

Die **ACCESSIBLE-BY**-Klausel für PL/SQL-Objekte ist mit der Oracle 12.1 eingeführt worden. Mit der Klausel **ACCESSIBLE BY** kann festgelegt werden, dass ein PL/SQL-Objekt wie Prozedur, Funktion oder Package, nur von bestimmten PL/SQL-Objekten aufgerufen werden kann. Ein typischer Anwendungsfall sind Hilfsfunktionen, die nur von ausgewählten PL/SQL-Objekten aufgerufen werden können oder sogar dürfen, mit dem Ziel eventuelle Seiteneffekte zu verhindern.

Bis Oracle 12.1 war die Zugriffsbeschränkung für PL/SQL-Packages allerdings nur auf der Package-Ebene möglich. Eine Zugriffsbeschränkung für einzelne Prozeduren oder Funktionen eines PL/SQL-Package war nicht möglich. Mit Oracle 12.2 ist die Einschränkung nun entfallen (siehe Abbildung 7).

Fazit

Zusammenfassend kann gesagt werden, dass die Oracle-Version 12.2 einige interessante und hilfreiche Neuerungen in den Bereichen SQL und PL/SQL mit sich bringt. Die Erweiterung der Konvertierungsfunktionen um einen Default-Wert, der im Fehlerfall statt einer Fehlermeldung geliefert wird, ersetzt individuelle Implementierungen, die in der Regel komplizierter und aufwändiger in der Wartung sind. Ebenso ist die Möglichkeit der transparenten Funktionserweiterung bzgl. der Groß- und Kleinschreibung und Akzent Insensitivität mit Collation auf der Tabellen- bzw. Spaltenebene zu erwähnen. Mit dieser Erweiterung muss in der Anwendung weniger Code implementiert werden, was ebenso zu einer besseren Wartbarkeit und einer geringeren Fehleranfälligkeit führt.

Die in diesem Artikel vorgestellten Neuerungen im SQL- und PL/SQL-Bereich behandeln die wesentlichen Aspekte der Oracle 12.2 Erweiterungen. Falls wir Ihr Interesse an den Neuerungen der Oracle-Version 12c aus Entwicklersicht geweckt haben, dann empfehlen wir Ihnen das Seminar „12c Neuheiten für Entwickler“.

Markus Fiegler
(info@ordix.de)

Service Registry & Discovery

Kein ewiges Suchspiel in der Microservices-Landschaft

Um stabile und hochwertige Software schnell ausliefern zu können, müssen die Unternehmen auf Veränderungen flexibel reagieren. Dazu gehört auch die IT des Unternehmens und damit die eingesetzten Tools und Technologien. Die Microservices-Architektur ermöglicht bereits durch die Struktur der Software, dass neue oder geänderte Anforderungen schnell umzusetzen und auszuliefern sind. Die Kommunikation zwischen den einzelnen Microservices spielt bei dieser Architektur eine große Rolle. Dementsprechend muss es einen Weg geben, um in der Microservices-Landschaft einzelne Microservices zu finden und somit auch ansprechen zu können.

Was sind Microservices?

Viele große Unternehmen wie Netflix, Amazon oder Otto nutzen erfolgreich Microservices. Aber was versteht man überhaupt darunter? Microservices sind ein Architekturmuster, bei welchem die Anwendung aus vielen kleinen Diensten zusammengesetzt wird.

Diese Dienste setzen jeweils eine bestimmte Aufgabe um, sind alleine für diese verantwortlich und sollen leicht zu verstehen sein. Funktional zusammengehöriger Programmcode wird so zusammengehalten. Jeder Microservice ist ein eigenständiges Gebilde und kommuniziert mit den anderen Services durch Aufrufe über das Netzwerk, sodass diese nicht fest miteinander gekoppelt sind.

Vorteile von Microservices

Die Verwendung von Microservices bringt viele verschiedene Vorteile mit sich. Unter anderem sorgt die Eigenständigkeit der Dienste dafür, dass jeder Microservice eine andere Technologie einsetzen kann. Anstatt sich für die gesamte Anwendung für beispielsweise eine Programmiersprache zu entscheiden, kann bei jedem Service – und somit bei jeder zu erledigenden Aufgabe – die Technologie ausgewählt werden, die dafür am besten geeignet ist. Auch die Speicherung der benötigten Daten kann auf verschiedene Arten umgesetzt werden. Zusätzlich können die einzelnen Dienste unabhängig voneinander deployed werden, sodass neue Funktionen unabhängig vom restlichen System zur Verfügung gestellt werden.

Der modulare Aufbau sorgt dafür, dass jeder Dienst beliebig wiederverwendet werden kann. Die in den Microservices gekapselten Funktionalitäten können auf verschiedene Weisen im System eingesetzt werden. Zusätzlich ist der Aufwand für das Ersetzen eines Services ziemlich gering, da eine neue Implementierung durch dessen geringe Größe relativ schnell durchgeführt ist. Eine verbesserte Umsetzung der Funktionalität oder der Einsatz neuer bzw. anderer Technologien, kann so schnell realisiert werden. Außerdem können die Microservices aufgrund ihrer Eigenständigkeit ohne weitere Probleme entfernt oder ausgetauscht werden, wenn einzelne oder sogar alle Dienste nicht mehr zu gebrauchen sind.

Service Registry & Discovery

Eine Microservices-Architektur besteht aus vielen kleinen, eigenständigen Diensten, die miteinander über das Netzwerk kommunizieren. Damit diese Kommunikation gelingt, müssen die Dienste andere Microservices finden und somit ansprechen können. Demnach wird eine zentrale Adressverwaltung für die Microservices benötigt. Zudem kann es praktisch sein, alle vorhandenen beziehungsweise laufenden Services zu kennen. Es erleichtert unter Umständen die Arbeit des Entwicklers, wenn dieser weiß, welche Funktionalitäten bereits implementiert sind und wo diese zu finden sind.

Die Adressverwaltung der Microservices ist nur möglich, wenn dieser Dienst aus zwei Teilen besteht: zuerst muss es einen Mechanismus geben, um einen Dienst anzumelden bzw. zu registrieren. Dies wird als Service Registry bezeichnet und enthält ein Verzeichnis aller laufenden Services mit deren jeweiligen Netzwerkadresse. Dann

muss es außerdem einen Weg geben, um einen bestimmten Microservice mithilfe dieses Verzeichnisses zu finden. Dies wird als Service Discovery bezeichnet.

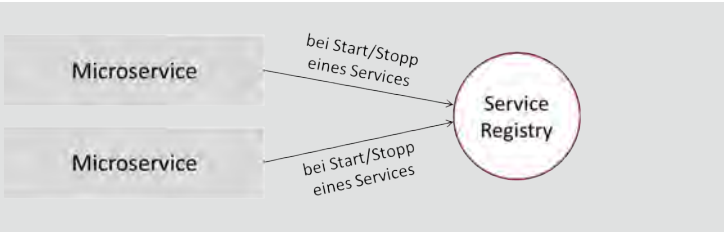

Abb. 1: Self Registration

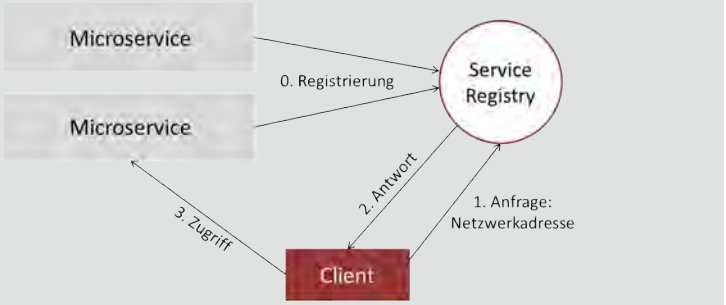

Abb. 2: Client-Side Service Discovery

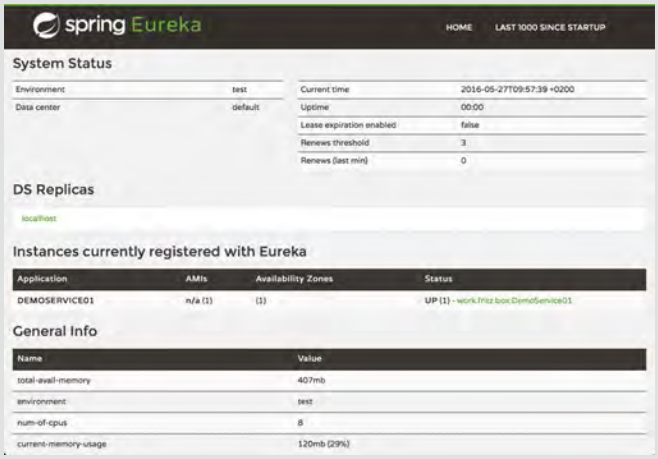

Abb. 3: Eureka-Dashboard

```
@EnableEurekaServer
@SpringBootApplication
public class EurekaApplication {
	public static void main(String[] args) {
	    SpringApplication.run(EurekaApplication.class, args);
	}
}
```

Abb. 4: Aktivierung des Eureka Server

Es gibt verschiedene Möglichkeiten diese Mechanismen umzusetzen. Bei der sogenannten Self Registration sind die Services selbst für die Registrierung zuständig. Beim Starten eines Services muss dieser sich mit seiner Netzwerkadresse an der Service Registry anmelden. Die Registrierung muss in periodischen Abständen erneuert werden, damit die Service Registry weiß, dass der Service noch aktiv ist. Auch beim Stoppen eines Services meldet sich dieser eigenständig von der Registrierungseinheit ab. Dieses Verfahren ist in Abbildung 1 dargestellt.

Eine andere Variante ist die 3rd-Party-Registration, bei der ein Service Manager für die Verwaltung der Services an der Service Registry zuständig ist. Beim Starten eines Services wird dieser angemeldet, beim Stoppen abgemeldet. Außerdem erkennt der Service Manager, wenn ein Service abstürzt.

Service Discovery kann client- oder serverseitig geschehen. Bei der Client-Side Service Discovery fragt der Client die Netzwerkadresse des gewünschten Services bei der Service Registry an und kann nach Erhalt der Adresse mit dem Service kommunizieren (siehe Abbildung 2). Bei der Server-Side Service Discovery wiederum kommuniziert der Client mit einem ihm bekannten Router, wenn er einen Service benutzen möchte. Dieser fragt die Service Registry nach der Netzwerkadresse des Service und ruft diesen danach auf.

Beispiel Eureka

Netflix als einer der Vorreiter der Microservices-Architektur hat in den letzten Jahren einige Open Source Frameworks veröffentlicht. Das Spring-Team hat daraufhin Spring Boot mit einigen Netflix-Bibliotheken in dem Projekt „Spring Cloud Netflix“ integriert. Dort wird die bekannte Autokonfiguration von Spring Boot mit eingebunden und ermöglicht durch einfache Annotationen die entsprechenden Komponenten von Netflix zu integrieren, konfigurieren und zu nutzen. Im Netflix-Stack übernimmt Eureka die Aufgabe der Service Registry & Discovery. Dabei werden Self Registration und Client-Side Service Discovery verwendet.

Eureka-Server

Der Eureka-Server ist die zentrale Anlaufstelle für die Eureka-Clients – also die Microservices – und wird als eigenständiger Dienst innerhalb einer Microservices-Architektur betrieben. Da dieser die Service Registry bereitstellt, kennt der Eureka-Server alle angemeldeten Clients. Auskunft über deren Status und einige Serverdetails gibt das mitgelieferte Dashboard, welches in Abbildung 3 dargestellt ist.

Um den Eureka-Server nutzen zu können, muss die Abhängigkeit **`spring-cloud-starter-eureka-server`** in der Maven **`pom.xml`** angegeben werden. Durch die Annotation **`@EnableEurekaServer`** wird die Service Registry in einer regulären Spring-Applikation aktiviert. Die Umsetzung dessen ist in Abbildung 4 gezeigt.

Die Service Registry muss nun noch konfiguriert werden, da dieser als Standard versucht sich selbst auch zu registrieren. Außerdem muss ein HTTP-Port angegeben werden, auf welchem der Server laufen soll. Die Konfiguration im YAML-Format ist in Abbildung 5 dargestellt.

Eureka-Client

Eureka-Clients können sowohl als Application Clients als auch als Application Services auftreten. Application Clients sind Dienste, die andere Dienste kontaktieren möchten und dafür die Netzwerkadresse beim Eureka-Server anfragen. Application Services bieten selbst Dienste an und werden daher vom Client am Eureka-Server registriert. Nach der Registrierung ist ein Heartbeat-Prozess dafür zuständig, in regelmäßigen Abständen dem Eureka-Server den aktuellen Status zu übermitteln. Fehlen die Heartbeats eine längere Zeit über, so wird der Service als inaktiv markiert und danach aus der Serviceliste entfernt. Oftmals ist es so, dass ein Dienst beide Funktionen (Client und Service) bereitstellt. Daher werden beim Eureka-Client diese beiden Typen nicht voneinander getrennt.

Um den Eureka-Client nutzen zu können, muss die Abhängigkeit **`spring-cloud-starter-eureka`** in der Maven **`pom.xml`** angegeben werden. Die Registrierung des Client am Eureka-Server erfolgt durch die Annotation **`@EnableDiscoveryClient`** in der Spring-Applikation, die den Dienst implementiert. Dies ist in Abbildung 6 dargestellt.

Der Eureka-Client muss nun noch konfiguriert werden. Zum einen benötigt dieser einen Namen mit dem er angesprochen werden kann und einen Port auf dem die Applikation laufen soll. Zum anderen muss angegeben werden, unter welcher URL der Eureka-Server zu finden ist. Die Konfiguration im YAML-Format ist in Abbildung 7 gezeigt.

Fazit

Um in einem System voller kleiner und eigenständiger Microservices die Kommunikation unter den verschiedenen Diensten zu ermöglichen, muss es eine Adressverwaltung geben. An der sogenannten Service Registry melden sich die Microservices an und bei Bedarf auch wieder ab. Service Discovery wird der Mechanismus des Auffindens eines bestimmten Microservice anhand des Adressverzeichnisses genannt. Eureka zum Beispiel übernimmt die Aufgaben von Service Registry & Discovery und kann in Spring Boot relativ leicht eingebunden und genutzt werden. Eureka ist aber nur eine von vielen Möglichkeiten – eine Alternative hierzu ist beispielsweise Consul von HashiCorp [1].

Veronika Eckstedt
(info@ordix.de)

```
eureka:
   instance:
      hostname: localhost
      client: # Not a client, don't register with yourself
            registerWithEureka: false
            fetchRegistry: false
server:
   port: 1111 # HTTP port
```

Abb. 5: Konfiguration des Eureka-Server

```
@EnableDiscoveryClient
@SpringBootApplication
public class ClientApplication {
   public static void main(String[] args) {
      SpringApplication.run(ClientApplication.class, args);
   }
}
```

Abb. 6: Aktivierung des Eureka-Client

```
# Spring Properties
spring:
      application:
         name: myService
# Discovery Server Access
eureka:
      client:
         serviceUrl:
         defaultZone: http://localhost:1111/eureka/
# HTTP port
server:
      port: 2000
```

Abb. 7: Konfiguration des Eureka-Client

Links

[1] Produktseite Consul:
https://www.consul.io/

[Q1] Newman, Sam (2015): „Microservices: Konzeption und Design“, MITP-Verlags GmbH & Co. KG

[Q2] Thönes, Johannes (2015): „Microservices: Software Engineering, In: IEEE Software, Januar/Februar, S. 113-116

[Q3] Pelka, Carsten und Michael Plöd (2017):
„Microservices à la Netflix: Die Bestandteile von Spring Cloud Netflix“, In: Java Magazin, 1, S. 110–113

[Q4] Richardson, Chris (2016):
„Microservices: From Design to Deployment“

[Q5] Richardson, Chris (2017): „Pattern: Self Registration“
http://microservices.io/patterns/self-registration.html

[Q6] Richardson, Chris (2017): „Pattern: 3rd Party Registration“
http://microservices.io/patterns/3rd-party-registration.html

[Q7] Richardson, Chris (2017): „Pattern: Client-side service discovery“
http://microservices.io/patterns/client-side-discovery.html

[Q8] Richardson, Chris (2017): „Pattern: Server-side service discovery“
http://microservices.io/patterns/server-side-discovery.html

[Q9] Long, Josh (2015): „Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka“, Hrsg. Pivotal Software
https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka

Software Development Kit (SDK)

Build-Prozess mal anders

Wird eine Software vom Kunden eingekauft anstatt diese selbst zu entwickeln, wird sie in vielen Fällen zunächst angepasst, bevor sie in den Betrieb geht. Die Softwarehersteller bieten hierfür meist die Möglichkeit mittels des Software Development Kit (SDK) einen Teil der Software zu personalisieren, um die Anwendung an die Wünsche des Kunden anzupassen. Dieser Artikel erläutert eine Möglichkeit, wie solche Software versioniert, automatisiert, gebaut und deployed werden kann.

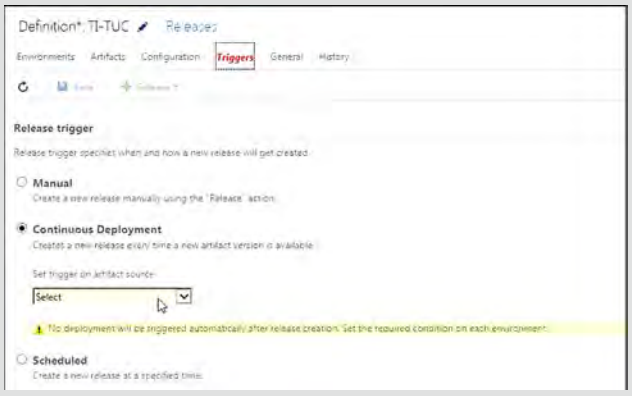

Abb. 1: Continuous-Integration-Einstellung des Team Foundation Server

Die Situation

Der Kunde möchte durch den Einkauf einer Standardsoftware Zeit und Kosten sparen. Die eingekaufte Software unterstützt die Personalisierung verschiedener Softwareteile mittels des Software Development Kit (SDK). Durch dieses SDK wird es dem Kunden ermöglicht, die eingekaufte Software an die vorhandenen Prozesse des Unternehmens anzupassen.

Bei einem solchen Vorgehen werden zwei Softwarestände zu einem Softwarerelease zusammengeführt. Die zwei Softwareteile lassen sich wie folgt unterscheiden:

- Kernsoftwareteile:
  Software, die vom Softwarehersteller geliefert wird
- Personalisierte Softwareteile:
  Eigene Softwareentwicklung die mittels SDK vorgenommen wurde

Hierbei können Seiteneffekte entstehen, wenn der personalisierte Softwareteil nicht kompatibel mit der Kernsoftware ist. Um dieses Problem frühzeitig identifizieren zu können, werden statt einer eigenen Entwicklungsumgebung mit dem aktuellen Entwicklungsstand, mehrere Versionen der Software für Smoke-Tests zur Verfügung gestellt, um die grundsätzliche Kompatibilität zu überprüfen.

Damit die Bereitstellung der verschiedenen Softwareversionen keinen übermäßigen Aufwand für das Projektteam erfordert, muss der Build-Prozess soweit wie möglich automatisiert werden. Nur so ist es möglich die Bereitstellung der Releases für die Entwicklungsumgebung und gleichzeitig für die Testumgebung zu ermöglichen.

Das nachfolgende Beispiel basiert auf einem aktuellen Kundenszenario. Die Verwendung vereinzelter Komponenten, wie beispielsweise dem Team Foundation Server, kann somit in anderen Projekten variieren.

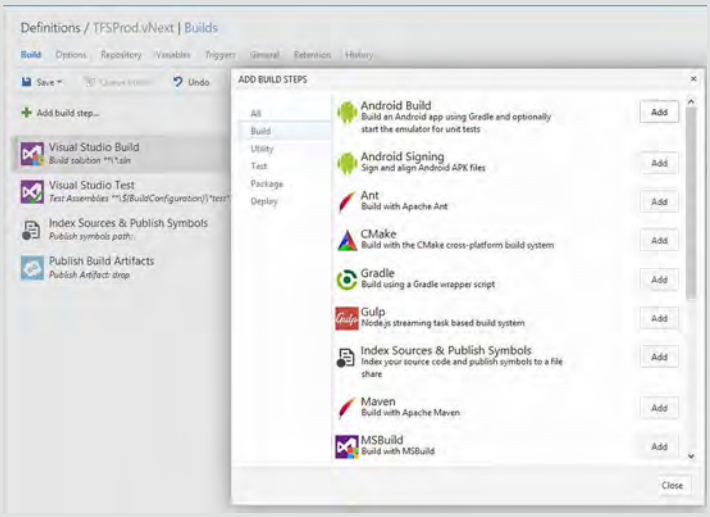

Abb. 2: Beispiel einer Build-Definition im Team Foundation Server

Ein Repository, zwei Softwareteile

Die Grundlage zur Automatisierung des Build-Prozesses ist eine zentrale Quellcodeverteilung, die in diesem Fall mit Hilfe von Git bewerkstelligt wird. Innerhalb des Git-Repository ist hierbei nur der personalisierte Softwareteil versioniert.

Die Kernsoftware wird nicht im Git gepflegt, da es hierbei meist um vorkompilierte Softwareteile handelt. Stattdessen wird dieser Teil der Software lokal auf einem Netzlaufwerk gespeichert, der im späteren Verlauf des Deployment-Prozesses eine wichtige Rolle spielen wird.

Das Repository besteht im Grunde aus zwei verschiedenen Branches:

- **Entwicklung**
  Innerhalb des Entwicklungs-Branch werden die verschiedenen personalisierten Softwareteile entwickelt. Jede Änderung der Software wird innerhalb dieses Branch, bzw. eines Unterzweigs der Entwicklung, vorgenommen.

- **Release**
  Wurden genügend Features für ein neues Release entwickelt, wird der aktuelle Stand der Entwicklung in den Release-Branch zusammengeführt. Dieser Branch wird somit nur dann aktualisiert, wenn die vorhandene Menge an neuen Funktionen ein neues Release bilden kann.

Build on Commit

Sobald innerhalb des Git-Repository ein neuer Commit im Release Branch entstanden ist, wird automatisch ein neuer Build angestoßen. Dies wird ermöglicht durch die Verwendung des Continuous Integration Plattform Team Foundation Server (TFS).

Der TFS bietet die Möglichkeit verschiedene Build-Regeln zu angeschlossenen Repositories zu definieren. So ist es möglich, bei jedem Commit innerhalb eines bestimmten Branch einen neuen Build anzustoßen oder wiederholend zu einem definierten Zeitpunkt. Diese Funktionalität wird auch Trigger genannt. Konfiguriert wird die gesamte Konfiguration mittels einer Weboberfläche, wie die Abbildung 1 am Beispiel der Continuous-Integration-Einstellung beweist.

Ein solcher Trigger wird innerhalb einer Build-Definition konfiguriert. Diese Build-Definition hält zudem alle Aufgaben, die währen des Build-Prozesses anfallen können. Die Abbildung 2 zeigt hierbei einen beispielhaften Ablauf, der wie folgt beschrieben werden kann:

1. Ausgabe aller Parameter für diesen Build
2. Verbindung zum Netzlaufwerk herstellen
3. Kopieren der Kernsoftware in das Arbeitsverzeichnis
4. Aufruf des Ant-Skriptes zur Paketierung der Anwendung
5. Speichern der Build-Artefakte im TFS
6. Verbindung zum Netzlaufwerk trennen

Die verschiedenen Build-Schritte werden vom TFS bereitgestellt. Des Weiteren ist es möglich eigene Erweiterungen für den TFS zu entwickeln, beziehungsweise den Erweiterungsmanager zu verwenden.

Ebenso wäre es natürlich möglich als Continuous Integration Plattform beispielsweise den Apache Jenkins oder Atlassian Bamboo zu verwenden, da diese einen ähnlichen Funktionsumfang besitzen.

Gebaut wird mit Ant

Innerhalb des Ant-Skriptes wird der eigentliche Build gesteuert. Die verschiedenen personalisierten Softwareteile werden innerhalb des Arbeitsverzeichnisses in die Kernsoftware integriert. Hierbei werden verschiedene Teile der Kernsoftware überschrieben, bzw. neue Teile hinzugefügt. Neue Softwareteile werden der Kernsoftware durch verschiedene Property-Dateien bekannt gemacht.

Nach der vollständigen Integration aller personalisierten Softwareteile, wird das gesamte Arbeitsverzeichnis als Java-Enterprise-Applikation gepackt.

Glossar

Apache Ant
Software zur automatisierten Erstellung von installierbaren Softwarepaketen aus existierendem Quellcode. Gesteuert wird Ant durch eine Build-Datei im XML-Format, in welchem die einzelnen Build-Schritte definiert werden.

Continuous Integration
Continuous Integration beschreibt den Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung. Das Ziel der kontinuierlichen Integration ist die Steigerung der Softwarequalität.

Git
Eine Software zur verteilten Versionsverwaltung von Dateien. Git ist ein sehr beliebtes Tool innerhalb von Softwareentwicklungsprojekten, da hierdurch sowohl die Versionsverwaltung als auch die parallele Arbeit im Team ermöglicht wird.

Smoke-Test
Ein funktionaler Test, der in diesem Fall die grundsätzliche Funktionalität der Software sowie die Verbindung mit den nach- bzw. vorgelagerten Systemen prüft.

Team Foundation Server
Der Team Foundation Server von Microsoft ist eine Continuous Integration Plattform, die zur Verwaltung von Repositories, von Arbeitspaketen und Testfällen sowie dem Build und Release Management genutzt werden kann. Ähnliche Produkte sind beispielsweise der Apache Jenkins und Bamboo von Atlassian.

Links

[1] ORDIX Blog - Git
https://blog.ordix.de/tags/git.html

[2] ORDIX Blog - Softwareentwicklung auf einem höheren Level – yContinuous Integration
https://blog.ordix.de/technologien/softwareentwicklung-auf-einem-hoeheren-level-continuous-integration-warum-und-wofuer-1.html

[3] ORDIX® seminare - Continuous Integration Workshop
https://seminare.ordix.de/seminare/entwicklung/course/1855-P-CI-01

[4] Flower, Martin (2006): Continuous Integration
http://martinfowler.com/articles/continuousIntegration.html

Bildnachweis

Ein erfolgreicher Build führt zum Deployment

Wurde der Build erfolgreich durchgeführt, kann der TFS hierauf automatisch reagieren und ein Deployment auf dem WebSphere Application Server durchführen. Ermöglicht werden kann das durch die Verwendung von entsprechenden Erweiterungen, der einzelnen Java Enterprise Application Server oder durch die Verwendung eines weiteren Build- bzw. Deployment-Skriptes durch Maven oder Ant.

Ausblick

In der aktuellen Konfiguration wird das gesamte Deployment automatisiert. Allerdings zieht eine Änderung der Software nicht selten auch eine Änderung an der Datenbank mit sich.

Dieser Teil des Build- und Deployment-Prozesses ist momentan manuell zu tätigen. Um die Struktur der Datenbank ebenfalls automatisiert anpassen zu können, kann beispielsweise auf Tools wie Liquibase zurückgegriffen werden.

Tools zur automatischen Anpassung von Datenbanken sollten allerdings nicht in der Produktiv- oder Prelive-Umgebung eingesetzt werden, da hier immer die Anpassungen an der Datenbank manuell erfolgen müssen.

Fazit

Durch die Verwendung des Team Foundation Server und der Continuous-Integration-Funktionalität ist es nun möglich, den gesamten Build-Prozess von einem neuen Commit im Git-Branch bis zum Deployment auf dem jeweiligen Server stark zu reduzieren.

In unserem Beispiel wurde somit die Zeit für ein Deployment von ungefähr 40 Minuten manuellen Arbeitsaufwands auf insgesamt fünf Minuten automatisierte Verarbeitung reduziert.

Durch diesen Ansatz ist sehr früh ersichtlich, ob der aktuelle Stand der Entwicklung ein wirklicher Kandidat für die Produktionsumgebung ist oder nicht. Die gesamte Softwarequalität kann durch einen solchen automatisierten Prozess erheblich verbessert werden.

Durch eine weitere Erhöhung des Automatisierungsgrades, gerade im Bereich der Datenbankaktualisierung, kann so in Zukunft noch mehr Zeit eingespart werden.

Phillipp Kürsten
(info@ordix.de)

Sicherheit für Kreditkartendaten

Business need to know: PCI-DSS in der Datenverarbeitung

Der Payment Card Industry Data Security Standard (PCI-DSS) ist ein verpflichtender Branchenstandard der kreditkartenausgebenden zur Datensicherheit für kreditkartendatenverarbeitende Institute. Der vorliegende Artikel behandelt, was dieser Standard in der Praxis für die Datenverarbeitung bedeutet.

Das PCI-DSS Council

Die Standards werden seit 2006 vom PCI-DSS Council, dem Zusammenschluss von American Express, Discover, JCB International, MasterCard und Visa erstellt. Sie gelten für Händler, die Kreditkartenlesegeräte nutzen und für die Hersteller von Hard- und Software, die die globale Infrastruktur für die Verarbeitung von Zahlungsdaten herstellen und betreiben. Dieser Artikel behandelt die Auswirkungen auf die Finanzinstitute, die diese Zahlungsdaten innerhalb ihrer IT-Infrastruktur verarbeiten. Das Council veröffentlicht die Standards auf seiner Webseite [1] Stand Juli 2017 liegen sie in der aktuellen Version 3.2 vor.

Neben der Aufstellung und Erläuterung der Standards finden sich auf der Webseite auch weitere hilfreiche Dokumente wie Glossare, Quick Reference Guides, Mustervorlagen und Verfahrensweisen zur priorisierten Implementierung der Standards. Anhand der Verfahrensweisen sind die Standards durch die Finanzinstitute umzusetzen. Dabei sind die Ergebnisse vorgegeben, während der Weg der Umsetzung dem einzelnen Unternehmen offen bleibt. Dieser Prozess wird von PCI-DSS zertifizierten Unternehmen (den sog. Qualified Security Assessors: QSA) bei Wunsch beratend unterstützt und innerhalb eines umfangreichen Assessments abgenommen.

Des Weiteren ist natürlich zu unterbinden, dass sich anhand gestohlener Authentisierungsdaten Karten fälschen lassen. Diese Daten nutzt man üblicherweise nur am Verkaufspunkt, sie dürfen nicht gespeichert werden und unterliegen nicht der PCI-Betrachtung in diesem Artikel.

Eine Nicht-Umsetzung der Compliance mit den Standards kann zum Entzug der Lizenz für die Verarbeitung der Daten führen. Dadurch entstandene Betrugsfälle werden mit empfindlichen Strafen geahndet.

Welche Daten sind zu schützen?

Grundsätzlich lassen sich die Kundendaten einer Kreditkarte in zwei Bereiche aufspalten: den Karteninhaberdaten (Cardholder Data: CHD) wie PAN (Primary Account Number), den Namen des Karteninhabers, den Service Code und das Verfallsdatum der Karte sowie die „empfindlichen“ Authentisierungsdaten (Sensitive Authentication Data: SAD), d.h. Daten des Magnetstreifens oder des Chips, die CAV2/CVC2/CVV2/CID-Daten (die drei- oder vierstellige Zahl auf Vorder- oder Rückseite der Karte) und die PIN. Da es an dieser Stelle nur um die in Transaktionen verarbeiteten Karteninhaberdaten geht, können die Authentisierungsdaten, die üblicherweise nur beim Bezahlen am Verkaufspunkt Verwendung finden, in unserer Betrachtung vernachlässigt werden.

Da sich schon allein anhand der Karteninhaberdaten Transaktionen (z.B. Interneteinkäufe) realisieren lassen, ist die Schutzbedürftigkeit der Daten klar: es gibt für sie einen illegalen Markt, in dem der einzelne komplette Datensatz im Wert von derzeit ungefähr einem Euro gehandelt wird. Insbesondere in Rechenzentren, wo Millionen dieser Datensätze in der Verarbeitung sind, ist es unabdingbar die Sätze zu schützen.

Die Standards kümmern sich allein um die Vertraulichkeit der Daten. Richtlinien zu Verfügbarkeit und Integrität werden nicht getroffen.

Scoping

Der erste Schritt zur Einführung des PCI-Datensicherheitsstandards ist die Analyse des Umfangs, des „Scope“ seiner Umsetzung. Letztendlich ist jedes Gerät (Rechner, Netzwerkkomponenten, Storage, Arbeitsplatz-PC etc.) mit dem die Daten transportiert, gespeichert, verarbeitet oder angesehen werden und jede Person, die auf eines dieser Geräte Zugriff hat, im Scope des Standards. Ohne geeignete Segmentierung, also Eingrenzung des Aufent-

Erstellung und Wartung sicherer Netzwerke und Systeme	**1.**	Installation und Wartung einer Firewall-Konfiguration zum Schutz von Karteninhaberdaten
	2.	Keine vom Anbieter gelieferten Standardeinstellungen für Systemkennwörter und andere Sicherheitsparameter verwenden
Schutz von Karteninhaberdaten	**3.**	Schutz gespeicherter Karteninhaberdaten
	4.	Verschlüsselung bei der Übertragung von Karteninhaberdaten über offene, öffentliche Netzwerke
Unterhaltung eines Anfälligkeits-Managementprogramms	**5.**	Schutz sämtlicher Systeme von Malware und regelmäßige Aktualisierung von Antivirensoftware und Programmen
	6.	Entwicklung und Wartung sicherer Systeme und Anwendungen
Implementierung starker Zugriffskontrollmaßnahmen	**7.**	Beschränkung des Zugriffs auf Karteninhaberdaten je nach Geschäftsinformationsbedarf
	8.	Identifizierung und Authentifizierung des Zugriffs auf Systemkomponenten
	9.	Physischen Zugriff auf Karteninhaberdaten beschränken
Regelmäßige Überwachung und regelmäßiges Testen von Netzwerken	**10.**	Verfolgung und Überwachung des gesamten Zugriffs auf Netzwerkressourcen und Karteninhaberdaten
	11.	Regelmäßiges Testen der Sicherheitssysteme und -prozesse
Pflege einer Informationssicherheitsrichtlinie	**12.**	Pflegen Sie eine Informationssicherheitsrichtlinie für das gesamte Personal

Abb. 1: Überblick über den PCI-DSS [2]

haltsbereichs der Daten innerhalb der verarbeitenden Infrastruktur, wäre der Standard auf alle Hardwarekomponenten und das komplette Personal anzuwenden, das damit in Berührung kommt. Dies ist offensichtlich ein nicht vertretbarer Aufwand an anzuwendender Sicherheitstechnologie und bspw. spezieller, geforderter PCI-Schulung des Personals. Die Beschreibung der genutzten Umgebung zur Verarbeitung der Daten geschieht in der Definition des Cardholder Data Environments (CDE), die detailliert dokumentiert und infolge eines Audits geprüft und abgenommen wird.

Neben der CDE selbst sind allerdings auch alle Geräte, die an ihr angeschlossen sind, im Scope der PCI-DSS-Betrachtung: Backup-Server, Mail-Server, Proxies, NTP und DNS-Server und dergleichen.

Ohnehin schreiben die Standards vor, dass der Zugriff auf die schützenswerten Daten nur dem Personal mit der geschäftlichen Notwendigkeit, dem „Business need to know“, gestattet ist. Dies umfasst Mitarbeiter, die z.B. Zahlungen prüfen oder Fragen zu Buchungen beantworten müssen, jedoch nicht die Administratoren der genutzten Datenbanken und Verarbeitungs- und Transportsysteme. Ein zentraler Punkt der Standards ist somit die Vermeidung von Angriffen auf die Datensicherheit, von innerhalb der datenverarbeitenden Unternehmen und folglich auch Verhinderung über gehackte, administrative Nutzerkonten, die sich von außen Zugang zu den Daten verschaffen.

Neben dem angeführten detaillierten Architekturdiagramm der CDE inklusive angeschlossener Systeme, ist auch ein Datenflussdiagramm zu erstellen, in dem aufgeführt ist, an welcher Stelle die relevanten Daten die CDE betreten, wo sie gespeichert, transportiert und verarbeitet werden und wo sie gegebenenfalls die CDE wieder verlassen. Anhand dieser primären Dokumente lässt sich der Aufwand einer Umsetzung der Standards diskutieren, planen und abschätzen.

Weitere Best Practices

PCI-DSS dient nicht nur der einmaligen Bereitstellung einer geeigneten Umgebung zum sicheren Verarbeiten der relevanten Daten. Es müssen zudem geeignete Prozesse etabliert werden, die eine tägliche Sicherheitskontrolle, eine kontinuierliche Dokumentation von sicherheitsrelevanten Ereignissen und regelmäßige Schulungen des Personals gewährleisten, da die PCI-DSS Audits wiederholt stattfinden. Zu den geeigneten Prozessen der täglichen Kontrolle gehört ein engmaschiges Monitoring der Sicherheitskontrollen wie Firewalls, Intrusion Detection und Präventionssysteme, Überwachung der Dateiintegrität, Antivirussoftware, Zugangskontrollen und dergleichen.

Jegliche Änderungen an der Umgebung (Changes) sind auf ihre Auswirkungen in Richtung der PCI-DSS-Anforderungen zu prüfen. Periodisch (viertel-/halb-/jährlich) sind Reviews der Umgebung anzufertigen, die alle Anlagen und Lokationen, Rechenzentren, Komponenten und Software umfasst und sogar Dateien wie Auditlogs und Firewallreviews mit einschließen. Ohne auch bisher eine PCI-DSS-Anforderung wörtlich zu kennen, soll klargestellt sein, dass eine Einführung der Standards einen hochkomplexen und tiefgreifenden, und ebenso kostspieligen wie aufwändigen Eingriff in die Unternehmensstruktur bedeutet.

Die Anforderungen im Einzelnen

Die Abbildung 1 zeigt die in sechs Gruppen aufgeteilten 12 Grundanforderungen von PCI-DSS. Diese Grundanforderungen spalten sich in gut 400 einzelne Anforderungen auf. Diese Anzahl reduziert sich allerdings dadurch, dass eine einzelne fachliche Anforderung („setze dies und jenes um"), oftmals gleichlautende, weitere Anforderungen nach sich zieht: Dokumentation zum Status der Umsetzung der Anforderungen, Schulung des Personal zu den Anforderungen, regelmäßige Überprüfung der Anforderungen. Somit bleiben im Endeffekt „nur" rund 120 unterschiedliche Anforderungen übrig.

Selbst diese reduzierte Anzahl an Anforderungen können nicht alle im Rahmen dieses Artikels besprochen werden. Das Dokument [3] beschreibt jedoch alle Anforderungen im Detail und gibt Hinweise auf Prüfverfahren und Anweisungen zur Umsetzung.

Erwähnenswert ist, dass nicht jede Anforderung zwingend umzusetzen ist. In Fällen, in denen eine Stelle eine explizite PCI-DSS-Anforderung aufgrund von legitimen, technischen oder dokumentierten geschäftlichen Einschränkungen nicht exakt erfüllen kann, können sog. Kompensationskontrollen in Erwägung gezogen werden. Voraussetzung hierfür ist jedoch, dass der mit der Nichterfüllung verbundene Risikozuwachs durch die Implementierung von Kontrollen an anderer Stelle kompensiert wird. Die Kompensationskontrollen werden vom Anbieter vorgeschlagen, bedürfen allerdings der Abnahme durch den QSA.

PCI-DSS kümmert sich somit um:

- Vertraulichkeit der PAN-Daten im Netzwerk und in Dateien, Datenbanken (3.4 Render PAN unreadable anywhere it is stored (including on portable digital media, backup media, and in logs))
- Maßnahmen zur Sicherung von Metadaten (in Logs etc.)
- Sicherung der Daten in Archiven
- Vorhandensein und Einhaltung von Changemanagement- und SIEM-Prozessen
- Entdeckung und Abwehr von Angriffen
- Abschottung der kreditkartendatenverarbeitenden Umgebung
- Physische Zugangskontrolle
- Logische Zugangskontrolle (need-to-know-Prinzip)
- Logging und Monitoring von Zugriffen
- Schulung von Mitarbeitern
- Dokumentation der Richtlinien
- Ausführen von Reviews
- Einhaltung von „Best-Practices"-Prozessen

Vorgehen in einem Kundenprojekt

Ziel des Kundenprojektes war die Umsetzung der PCI-DSS-Standards zur Auditierung durch einen QSA für eine kreditkartendatenverarbeitende Anwendung in Version 3.1. Zu den Aufgaben zur Realisierung konnten frühzeitig identifiziert werden:

- Konsolidierung der Teildatenbanken für CHD auf eine Datenbank
- Konsolidierung der Transaktionsverarbeitung (Anwendung) auf ein System
- Tokenisierung und Verschlüsselung der PAN
- Entwurf und Umsetzung eines stringenten Verschlüsselungsmanagements
- Umsetzung einer starken Verschlüsselung entlang aller Lieferwege
- Entwurf und Umsetzung eines Betriebs- und Benutzerkonzepts für Datenbank, Anwendung und Transaktionsverarbeitung
- Entwurf und Umsetzung eines konformen Konzepts für Logging, Monitoring, Reporting und Alerting

Konsolidierung

Ein wichtiger Schritt in Richtung Verkleinerung des CDE war die Konsolidierung der Speicherorte und der Systeme zur Transaktionsverarbeitung. Die Vorteile der Parallelisierung (schnellere Verarbeitung, erhöhte Ausfallsicherheit) wurden dabei beschränkt. Allerdings konnte die Anzahl der beteiligten Systeme und die Anzahl der beteiligten Administratoren gedrittelt werden. Die Anzahl der angeschlossenen Systeme reduzierte sich deutlich, die Verwaltung der relevanten Nutzer konnte durch die Beschränkung des Systems auf die kreditkartendatenverarbeitende Anwendung extrem vereinfacht werden, der Scope von PCI-DSS war somit auf einen Bruchteil der gesamten IT-Landschaft heruntergebrochen.

Tokenisierung und Verschlüsselung

Ein zentraler Aspekt für die Umsetzung der Anforderung 3.4 war die Einführung einer Tokenisierung in der Anwendung, d.h. in der sechzehnstelligen PAN wird eine gewisse Anzahl an Stellen durch einen alphanumerischen Code (Token) ersetzt. In der Datenbank entsteht eine Tabelle, in der die stark verschlüsselte PAN und die tokenisierte PAN gegenübergestellt sind. Den Schlüssel zur Entschlüsselung der PAN darf außer der Anwendung selbst kein Nutzer anwenden. Eine Verarbeitung z.B. von Bulkdaten, d.h. Dateien mit tausenden Abrechnungen mit Klartext-PAN-Daten, läuft somit wie folgt ab:

- Entschlüsselung der ankommenden verschlüsselten Datei
- Tokenisierung: Ersetzen der PAN durch tokenisierten PAN in der Datei

- Verarbeitung anhand der tokenisierten PAN
- Bei Ausgabe von Verarbeitungsdaten in Datei anschließend Detokenisierung, Verschlüsselung und Weiterversand

Das Betriebssystem unterstützt dabei die Verschlüsselung und Verarbeitung fälschungs- und abbruchsicher mit eigenen Cryptokarten. Einmalig mussten dazu sämtliche in der Datenbank gespeicherten PAN-Daten durch die tokenisierten PANs ersetzt werden. Dieses Verfahren bietet mehrere Vorteile:

- Klartext-PAN-Daten sind vom System verschwunden, Administratoren von Storage, DB und System sehen nur tokenisierte Daten
- Die zentrale Anforderung, keine PAN im Klartext zu speichern, ist erfüllt
- In Backups und Archiven existieren ebenfalls keine Klartext-PANs
- Die Schlüsselnutzung ist auf den Applicationuser begrenzt
- Klartext-PAN-Daten existieren maximal temporär im Hauptspeicher

PCI-DSS schreibt die Mindeststärke der genutzten Schlüssel vor, für die Verschlüsselung der PANs wurde ein symmetrischer Schlüssel der Stärke 256 bit und Methode AES gewählt.

Verschlüsselung der Kommunikation mit Partnern

Für den Datenaustausch mit Partnern werden hybride Verschlüsselungsmethoden eingesetzt, d.h. ein asymmetrisches Schlüsselpaar wird mit dem RSA-Algorithmus erzeugt und der öffentliche Anteil (Public Key) mit dem Partner ausgetauscht. Hybride Verschlüsselungsverfahren werden genutzt, um mit einem symmetrischen Schlüssel die Nutzlast/Dateninhalt der Datei auf effiziente und schnelle Weise zu verschlüsseln. Der symmetrische Schlüssel wird dabei für jede Datenübertragung neu generiert. Mit dem öffentlichen Schlüssel des Partners wird dieser symmetrische Schlüssel verschlüsselt und an die Datei angehängt. Somit umgeht man die technologisch aufwändigere und langsamere Verschlüsselung großer Dateien mit Schlüsseln großer Bitlänge.

PCI-DSS erlaubt nur bestimmte Kombinationen von asymmetrischen und symmetrischen Schlüsseln, so z.B. die Verschlüsselung von AES 128 bit Sessionkeys nur mit RSA 3072 bit [2] Schlüsseln. Die Verschlüsselung läuft über Kryptomodule der Hardware. Diese Module benutzen eigene, abgegrenzte Kryptoprozessoren und Speicherbereiche und stellen höchste Sicherheitsanforderungen zufrieden.

Kryptoschlüssel besitzen nur eine begrenzte Gültigkeit und werden nach Ablauf der Periode oder wenn der Verdacht einer Kompromittierung besteht ausgetauscht. Ein Prozess des Schlüsselmanagements (Verantwortlichkeiten, Kommunikationswege, Notfallprozeduren, Verwaltung von Master Keys etc.) ist daher zu definieren und wird im Rahmen des Audits überprüft.

Betriebskonzepte

Zur Erstellung der Betriebskonzepte der Anwendung, der genutzten Datenbank und des Transaktionssystems wurde ein pragmatischer Ansatz gewählt. In Workshops mit den beteiligten Administratoren prüften wir die über 400 Anforderungen auf ihre Relevanz für das einzelne Sachgebiet. Die Anzahl der zu erfüllenden Anforderungen reduzierte sich dadurch stark. Für jede übrig gebliebene Anforderung wurde ein Soll-Ist-Vergleich angestellt und diskutiert, wie eine Umsetzung stattfinden kann, womit ein Aktionsplan für das Einführungsprojekt entstand. Durch die nahe Orientierung am Wortlaut des Standards ergab sich durch die Stellungnahme zu jedem relevanten Punkt ein direkter und prüfbarer „Report on Compliance“ für den folgenden Audit.

Logging, Monitoring, Reporting und Alarmierung

Auch wenn die Anforderungen zu diesem Thema in der Zahl recht gering sind, ist der technische und organisatorische Aufwand zur konformen Abhandlung enorm.

Über Audittrails muss die Nachvollziehbarkeit des Zugangs von individuellen Nutzern zu kritischen Ressourcen, ebenso wie ihre eigene Unveränderlichkeit, sichergestellt sein. Alle Zugriffe zu Karteninhaberdaten und alle Aktionen von administrativen Nutzern sind zu erfassen. Logs sind in regelmäßigen Abständen nach Anomalien und verdächtigen Tätigkeiten auszuwerten.

Logging bedeutet dabei, dass sicherheitsrelevante Aktivitäten auf dem Rechner in verarbeitbarer Form festgehalten werden.

Monitoring sorgt für eine Filterung relevanter Aktivitäten, die aus dem Logging stammen. Mittels geeigneter Filter (z.B. White- oder Blacklists) erfolgt eine sinnvolle Reduktion der anfallenden Datenmengen und kann durch Setzen von Schwellwerten (bspw. sehr häufiges Eingeben eines falschen Passworts für einen Benutzer) einen Alarm auslösen.

Das Reporting stellt die überwachten Tätigkeiten in geeigneter Form zusammen und generiert Berichte, die vom vorgesehenen Personal gesichtet werden. Bei verdächtigen Tätigkeiten erfolgt eine Alarmierung zur näheren Untersuchung und Bewertung des Alarmgrundes.

Alarmierungen können auf zwei Wegen ausgelöst werden:

- **Realtime Alert**
  Eingehende Meldungen aus dem Logging bzw. aus dem Monitoring werden von einem Überwachungstool

in Echtzeit auf sicherheitstechnische Relevanz hin gefiltert und an ein Security Incident and Management System (SIEM) weitergegeben. Dort wird sofort ein Alarm ausgelöst, auf den innerhalb einer angegebenen Frist reagiert werden muss.

- **Reports**
  Aus der Gesamtheit der Meldungen wird täglich ein Report generiert, der auch zeitliche Zusammenhänge sicherheitsrelevanter Ereignisse aufdecken kann. Der Report ist täglich zu reviewen und gegebenenfalls ist ein Alarm auszulösen.

Funktionstrennung

Im Unternehmen sind für einzelne Tätigkeitsfelder Rollen zu definieren, so dass Sicherheitsleitlinien eingehalten werden können. Dazu gehört eine Trennung von überwachenden und implementierenden Tätigkeiten oder die Bewertung und Lösung von sicherheitsrelevanten Ereignissen.

Operative und überwachende Tätigkeiten sind demnach auf Rollen in unterschiedlichen Abteilungen der Organisation aufgeteilt. Die finanziellen, fachlichen und organisatorischen Konsequenzen aus den Anforderungen sind nicht zu unterschätzen: geeignete Tools sind zu suchen, zu evaluieren und zu implementieren, Regelwerke (Black-/Whitelists etc.) zu entwickeln, testen und einzusetzen, Prozeduren und Personal in der Organisation einzuführen.

Fazit

Dieser Artikel behandelt die hervorstechendsten Aspekte einer Umsetzung von PCI-DSS-Vorgaben [3] in einem kreditkartendatenverarbeitenden Institut. Auf dem Weg dorthin sind meist noch technologische Randthemen (Einschränkung der Zahl der Adminnutzer, Absicherung von Netzwerkstrecken durch TLS innerhalb des RZ, Einbindung von Partnern in die Sicherheitstechnologie etc.) abzudecken. Die Anforderungen sind hoch, aber aufgrund der Angriffswahrscheinlichkeit und des hohen Risikos an Ruf- und finanziellem Schaden sinnvoll.

Dr. Uwe Bechthold
(info@ordix.de)

Glossar

AES
Der Advanced Encryption Standard ist vom National Institute of Standards and Technology (NIST) als Standard definiert worden. Es handelt sich um ein symmetrisches Verschlüsselungsverfahren.

CHD
Cardholder Data sind personenbezogene Daten, die mit einer Person verbunden sind, die eine Kredit- oder Debitkarte besitzt.

CDE
Cardholder Data Environment ist ein Computersystem oder eine vernetzte Gruppe von IT-Systemen, die Karteninhaberdaten oder empfindliche Zahlungsauthentifizierungsdaten verarbeitet, speichert und/oder überträgt.

Kompromittierung
Ein System, eine Datenbank oder auch nur ein einzelner Datensatz wird als kompromittiert betrachtet, wenn Daten manipuliert sein könnten und wenn der Eigentümer (oder Administrator) des Systems keine Kontrolle über die korrekte Funktionsweise oder den korrekten Inhalt mehr hat, beziehungsweise ein Angreifer ein anderes Ziel der Manipulation erreicht hat.

PAN
Die Primary Account Number identifiziert eindeutig Kreditkarten und Debitkarten, sie ist eine Zahlungskartennummer. Die PAN ist auf der Spur 2 einer Kreditkarte/Debitkarte codiert.

PCI-DSS
Payment Card Industry Data Security Standard (PCI-DSS) ist ein Regelwerk im Zahlungsverkehr, das sich auf die Abwicklung von Kreditkartentransaktionen bezieht.

QSA
Ein Qualified Security Assessor (QSA) ist eine Person, die vom PCI Security Standards Council zertifiziert wurde, um Händler für die Einhaltung des PCI-DSS-Standards zu auditieren.

RSA
Der Rivest-Shamir-Adleman-Algorithmus ist ein System für die Verschlüsselung und Authentifizierung im Internet und gilt als der am häufigsten verwendete Algorithmus zur Verschlüsselung und Authentifizierung.

SAD
Sensitive Authentication Data sind Daten, die von Kartenausstellern zur Autorisierung von Transaktionen verwendet werden.

SIEM
Security Incident and Event Management ist ein Ansatz des Sicherheitsmanagements, der darauf abzielt, eine ganzheitliche Sicht auf die Sicherheit der Informationstechnologie eines Unternehmens zu haben.

TLS
Transport Layer Security ist ein Protokoll zum Schutz persönlicher Daten bei der Kommunikation von Nutzern mit Anwendungen im Internet.

Links

[1] PCI-Standards
www.pcisecuritystandards.org/documents/PCI_DSS_v3-2_3_de-DE.pdf

[2] Empfehlung für die Schlüsselverwaltung -
National Institute of Standards and Technology
https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-3/archive/2012-07-10

[3] PCI-DSS Council
www.pcisecuritystandards.org

Manufaktur IT
Information Security Training for Business

ORDIX® seminare

Werden Sie Top Experte in der IT-Security!

Wir bereiten Sie erfolgreich und schnell auf die internationalen Security-Zertfikate in deutscher Sprache vor.

Jetzt zum Training anmelden!

Die GISWS prognostiziert für Europa einen Mangel von 350.000 Cybersicherheits-fachleuten bis 2022.
(Global Information Security Workforce Study 2017)

CISSP
CISM CISA

Mit uns erfolgreich zum Examen!

Unsere Seminare zur IT-Security finden Sie unter:
seminare.ordix.de/seminare/it-security.html