Ausgabe 01/2019

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 01

Java aktuell

iJUG Verbund
www.ijug.eu

Next Level Java
Erste Eindrücke von der Version 11

Eclipse MicroProfile
Neue Features und Erweiterungen

Blockchain
Implementierung in der IDE seiner Wahl

Java im Aufwind

Für unseren Standort in Frankfurt am Main oder irgendwo auf der Welt suchen wir... Dich!

Du musst kein „Rockstar“ sein, um bei uns zu arbeiten.
Es reicht völlig „Du selbst“ zu sein.

Java-Entwickler (m/w/d)

mit oder ohne Berufserfahrung

Es wartet ein engagiertes Team auf Dich, um mit Dir zusammen unser Kernprodukt weiter zu optimieren und jede Menge anspruchsvolle IT-Projekte für unsere Kunden zu entwickeln.

Egal, ob Backend, Frontend oder Full-Stack: Bei uns bist Du von der Konzeption bis zur Umsetzung und Begleitung intensiv an allen Prozessen beteiligt.

Es lohnt sich:

- Ein fester Arbeitsplatz und kurze Entscheidungswege
- Ergebnisorientiertes Arbeiten und eine steile Lernkurve
- Viel Abwechslung, Freiraum und Eigenverantwortung
- Flexible Arbeitszeiten und Home-Office-Regelung
- Regelmäßige Team-Events

Weitere Infos: **www.inxire.com**

inxire – digital first.

INXIRE®

DOAG Legal Council: Erste Einschätzung zu kostenpflichtigen Java SE-Updates

Die öffentlichen Updates für Java SE 8 sollen nach Januar 2019 nur noch kostenpflichtig zur Verfügung stehen. Dann benötigen Unternehmen eine kommerzielle Lizenz. Das DOAG Legal Council nimmt sich aktuell des Themas an und prüft rechtliche Möglichkeiten, Oracle in dieser Angelegenheit zu begegnen.

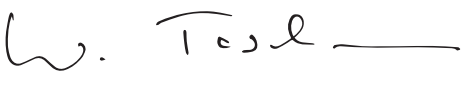
Dr. Jana Jentzsch, Mitglied des DOAG Legal Council, sagt: „Die Kunden sollten nun schnell auf diese Ankündigung reagieren und verschiedene Fragestellungen im Unternehmen klären, insbesondere wie viele Java-Installationen im Einsatz sind, wo sich diese befinden und ob es sich um eine eingebettete Nutzung handelt. Vor dem Hintergrund der möglicherweise hohen Kosten, die ab Januar anfallen könnten, werden einige Unternehmen gegebenenfalls sogar erwägen müssen, ob Java durch eine andere Technologie ersetzt werden kann."

Das DOAG Legal Council beschäftigt sich aktuell mit dem Thema und prüft, ob eventuell rechtliche Möglichkeiten bestehen, dass die Updates auch weiterhin kostenfrei zur Verfügung stehen oder zumindest Kosten reduziert werden können. In diesem Zusammenhang stellen sich laut Dr. Jentzsch beispielsweise Fragen nach dem auf das „Oracle Binary Code License Agreement" (BCLA) anwendbare Recht und dem Einräumen angemessener Übergangszeiträume durch Oracle.

Für die deutschen Oracle-Anwender erscheint es insoweit zumindest nach einer ersten Analyse nicht positiv, dass auf das BCLA das Recht des Staates Kalifornien Anwendung finden soll und als ausschließlicher Gerichtsstand San Francisco festgelegt ist.

Ich hoffe, dass sich die Situation noch zum Vorteil der Community wendet.

Ihr



Wolfgang Taschner
Chefredakteur Java aktuell

Inhalt

Pünktlich sechs Monate nach Java 10 ist Java 11 erschienen

Eine Blockchain inklusive Kryptowährung selbst erstellen

Visualisierung ist ein wichtiges Werkzeug für jeden Software-Entwickler

Wie Java-Programme in die Oracle-Datenbank kommen

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java.

30. August 2018

Eclipse Foundation wächst dank Jakarta EE und IoT

Die Eclipse Foundation hat sechzehn neue Mitglieder bekanntgegeben – das klingt zugegebenermaßen erst mal nicht so beeindruckend, es handelt sich jedoch um Organisationen: In der Regel Firmen – kleine und große wie Fujitsu oder Toyota, zudem sind unter Entwicklern bekannte Namen wie „Cloudbees" und „Lightbend" dabei; aber auch der iJUG e.V. gehört jetzt dazu, ebenso wie die London Java Community (LJC) und die Istanbul JUG. Schön, dass die Community sich hier wieder aktiv einbringt – iJUG und LJC haben als „Solutions Member" auch Stimmrechte in der Foundation, können den Kurs von Jakarta EE und anderen also unmittelbar mitbestimmen. Die Foundation begründet den zuletzt starken Anstieg von Mitgliedschaften mit zwei Projekten, die besonders viel Interesse wecken: Jakarta EE und die IoT Working Group.

http://www.eclipse.org

10. September 2018

Microvogel: Oracles Schwalbe

Oracle hat ein neues Open-Source-Framework veröffentlicht, das Projekt Helidon. Das ist griechisch für Schwalbe, als Symbol für den leichtgewichtigen Microservice-Ansatz. Das Framework kommt in zwei Varianten: Helidon SE ist, wie der Name vermuten lässt, komplett auf Java SE basierend, mit integriertem Reactive HTTP API (Netty) Web Server sowie Komponenten für Konfiguration und Security; Helidon MP implementiert aktuell Microprofile 1.1 plus die Metrics und Health Check APIs aus Microprofile 1.2.

https://helidon.io

11. September 2018

Java 11 – Unterschiede zwischen Oracle JDK und OpenJDK

Mit Java 11 sollen ja die Unterschiede zwischen dem Oracle JDK und OpenJDK verschwinden. Ein paar Unterschiede (neben den Lizenzen – GPL v2 mit „Classpath Exception" beziehungsweise „Binary Code License for Oracle Java SE technologies") gibt es allerdings doch, wie der oberste Produktmanager Donald Smith im Oracle Blog erklärt. Sie betreffen Feinheiten wie den Versionsstring und die Reaktion auf das praktisch obsolete „-XX:+UnlockCommercialFeatures"-Flag (das OpenJDK wirft dann einen Fehler). Es gibt auch tatsächliche funktionale Unterschiede, etwa dass „third party Cyptography Provider" beim Oracle JDK weiterhin mit einem bekannten Zertifikat signiert sein müssen und dass es Logdaten für Oracles kommerzielle „Advanced Management Console" erzeugen kann (ob diese Funktion auch im OpenJDK sinnvoll wäre, soll diskutiert werden). Also im Großen und Ganzen tatsächlich keine wichtigen Unterschiede, aber manchmal kommt es ja auf die Feinheiten an (wie den Versionsstring).

https://blogs.oracle.com/java-platform-group/oracle-jdk-releases-for-java-11-and-later

17. September 2018

„Java is still free"

Die Java Champions haben gemeinsam an einem Dokument geschrieben, das noch einmal in aller Ausführlichkeit die Themen „Releasezyklus", „Support", „LTS Releases" und unterschiedliche JDK-Distributionen beleuchtet. Das Dokument, verfügbar als Google Doc und auf medium.com, sollte so ziemlich alle Fragen dazu beantworten.

https://tinyurl.com/yap7h3wt

18. September 2018

SE 10 für Raspberry Pi – Liberica

Kurz vor dem Java-11-Release ist OpenJFX 11 freigegeben worden. JavaFX wird ja jetzt unter der Schirmherrschaft von Gluon, aber noch mit tatkräftiger Unterstützung von Oracle weiterentwickelt, als ein vom OpenJDK unabhängiges Projekt. Die Version 11 ist logischerweise mit einem JDK 11 lauffähig, beim OpenJDK funktioniert allerdings auch die Version 10. Gluon bietet für OpenJFX 11 jetzt ebenfalls „Long Term Support" an. Auch spezielle Hardware wird unterstützt: In der letzten Tagebuchausgabe hatte ich von Liberica JDK 10 geschrieben (Java SE 10 für den Raspberry Pi). OpenJFX 11 wird inzwischen mit Liberica JDK 11 getestet – mit der gleichen JavaFX-Codebasis wie für die Desktop-JDKs. Damit steht JavaFX 11 zumindest schon mal als „Early Access" auch für den Raspberry Pi zur Verfügung.

https://openjfx.io

19. September 2018

GlassFish-Code bei Eclipse angekommen

Der Code von GlassFish – die Referenz-Implementierung von Java EE – ist nun vollständig in den Git Repositories der Eclipse Foundation angekommen. Auch wenn GlassFish für Jakarta EE nicht mehr „die" Referenz-Implementierung sein wird, spielt er doch mindestens zur Demonstration der Kompatibilität eine wichtige Rolle. Damit ist also ein weiterer wichtiger Schritt für Jakarta EE geschafft – der eigentliche Aufwand ist ja nicht „git clone ...", sondern vor allem der Tanz der Juristen.

https://github.com/eclipse-ee4j/glassfish

20. September 2018

Eclipse IDE: Neues Release-Schema

Die Eclipse Foundation hat ihren Release-Zyklus für die IDE und verwandte Projekte umgestellt. Statt eines Haupt-Releases und drei-

er kleinerer Updates pro Jahr gibt es nun auch hier einen „Release Train", mit vier gleichwertigen „Simultaneous Releases". Auch das Schema für die Namen ändert sich: Auf „Neon" (Juni 2016), „Oxygen" (Juni 2017) und „Photon" (Juni 2018) folgen nun „2018-09" und im Dezember dann „2018-12" – wobei die zugrunde liegende Plattform-Nummerierung beibehalten wird; „2018-09" entspricht also „4.9" und „2018-12" dann schon „4.10".
https://www.eclipse.org/downloads/packages

22. September 2018

Spring-Framework 5.1 mit JDK-11-Unterstützung freigegeben

Das Spring-Framework 5.1 ist da und unterstützt Java von Version 8 bis zur erst in ein paar Tagen erscheinenden Version 11. Auch für die GraalVM gibt es ein paar spezielle Anpassungen. Der dazu passende „Entwicklungsbeschleuniger" Spring Boot 2.1 ist noch in Arbeit. Mitte Oktober soll ein erster Release-Kandidat herauskommen, basierend auf dem ersten Patch-Release 5.1.1 des Frameworks.
https://springoneplatform.io/2018/sessions/spring-framework-5-1-on-jdk-8-11

24. September 2018

JNoSQL in Jakarta EE

Jakarta EE soll das erste eigene Standardisierungsprojekt erhalten: JNoSQL als implementierungsunabhängige Schnittstelle beim Zugriff auf NoSQL-Datenbanken. Das Projekt ist bereits Anfang 2017 bei der Eclipse Foundation gestartet worden und schon weiter, als es die aktuelle Version 0.0.6 ausdrückt. Mit der Aufnahme in Jakarta EE, die aktuell diskutiert wird, könnte es dann die Lücke neben JPA ausfüllen. In Oracles ursprünglichem Plan für Java EE 9 war bereits eine Spezifikation mit dem zugehörigen Package „javax.persistence.nosql" vorgesehen – bevor EE an die Eclipse Foundation übergeben wurde. Die Package-Namen mit „javax" sind jetzt ja nicht mehr erlaubt, die Rechte an „javax" verbleiben bei Oracle, also wird es wohl etwas mit „jakarta.*" oder „org.eclipse.jakarta.*" werden – das wird gerade im EE4J-Projekt diskutiert.
https://www.tomitribe.com/blog/jnosql-and-jakarta-ee

25. September 2018

Java-11-Release

Java 11 ist veröffentlicht – das erste „Long Term Support Release" seit der Einführung des neuen Release-Zyklus, also das erste seit Java 8 mit langfristigem Support. Es ist kein großer Sprung von Release 10 mehr mit dem festen Sechs-Monats-Abstand; ein paar Features hatte ich ja schon mal erwähnt („NoOps" Garbage Collection, Flight Recorder, Standardisierung des HTTP-Client-API etc.).
http://openjdk.java.net/projects/jdk/11

26. September 2018

Freier Java-LTS-Support von Microsoft und Azul Systems

Azure und Azul ähneln sich zumindest in geschriebener Form schon mal. Aber jetzt kooperieren sie auch. Die Microsoft-Cloud hat ja schon lange Java-Unterstützung im Angebot. Jetzt soll den Kunden ohne zusätzliche Supportkosten ein Long Term Support für Java SE 7, 8 und 11 in Azure geboten werden – mithilfe des Supports von Azul und deren OpenJDK-Builds („Zulu Enterprise"). Hintergrund soll die Erkenntnis sein, dass viele Firmen mit den schnellen Release-Schritten nicht hinterherkommen und Projekte auf absehbare Zeit noch auf Java 7 und 8 festsitzen. Diesen will man in der Azure-Cloud ein möglichst behagliches Angebot machen.
https://azure.microsoft.com/de-de/blog/microsoft-and-azul-systems-bring-free-java-lts-support-to-azure

26. September 2018

JavaLand 2019: Programm online

Das Programm der nächsten JavaLand ist bereits online: Mehr als 100 Vorträge an zwei Tagen und jede Menge Community-Aktivitäten, plus der Schulungstag am Donnerstag – da sollte für jeden etwas dabei sein. Die Veranstaltung findet vom 19. bis 21. März 2019 im Phantasialand in Brühl statt.
https://programm.javaland.eu/2019/#/schedule

5. Oktober 2018

Kotlin Foundation

JetBrains und Google haben die Kotlin Foundation gegründet, um die Weiterentwicklung der Programmiersprache („as free software") sicherzustellen. Die Hauptverantwortlichkeiten sind das Verwalten der Marke, die Ernennung des „Lead Language Designers" und die Kontrolle über inkompatible Sprachänderungen. Die Foundation selbst bezahlt jedoch keine Entwicklerinnen und Entwickler, diese werden weiter von Google, JetBrains und anderen gestellt.
https://kotlinlang.org/foundation/kotlin-foundation.html

17. Oktober 2018

Vom JCP zum Eclipse Foundation Specification Process

Mike Milinkovich, Chairman der Eclipse Foundation, hat einen Entwurf des Eclipse Foundation Specification Process veröffentlicht. Das Dokument ist für Jakarta EE geschrieben worden, da die Eclipse Foundation einen Ersatz für den JCP benötigt, soll aber bei Bedarf auch anderen Working Groups dienen. Zentrale Anforderungen sind laut Milinkovich Leichtgewichtigkeit und die Nähe zur Open-Source-Entwicklung, wobei Letzteres eine Herausforderung sei, da das Schreiben von Spezifikationen sich deutlich von der Open-Source-Entwicklung unterscheidet. Eine „Code First"-Vorgehensweise und Experimente sollen vom Prozess unterstützt werden.
https://blogs.eclipse.org/blogs/mike-milinkovich

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

17. Oktober 2018

JVM Ecosystem Report 2018

Die Firma Snyk hat in Zusammenarbeit mit Oracle und dem Java Magazine eine Umfrage durchgeführt – nach eigener Aussage „the largest survey ever of Java developers": Mehr als 10.200 mal wurde der Online-Fragebogen ausgefüllt. Die Ergebnisse wurden jetzt veröffentlicht. Ein paar Zahlen daraus: Das Oracle JDK (70 Prozent) und das OpenJDK (21 Prozent) stellen den Löwenanteil, wenn es um den Produktionsbetrieb von Java-Anwendungen geht (genau gefragt war nach „Haupt-Anwendungen", um unkritische/Nischen-Applikationen auszuschließen, bei denen gerne mal experimentiert wird); selbst IBMs J9 kommt nur auf 4 Prozent. 79 Prozent der „Haupt-Anwendungen" laufen weiterhin auf Java 8, die Versionen 9 und 10 kommen auf je 4% (und Version 7 immer noch auf 9 Prozent). 28 Prozent wissen noch nicht, wie sie mit dem neuen Release-Zyklus umgehen sollen, während die (relative) Mehrheit von 34 Prozent sagt, dass sie auf Long Term Releases (und damit vermutlich auch den – kommerziellen – Support) setzen werden. Java EE 7 (27 Prozent) liegt immer noch vor EE 8 (22 Prozent). Bei der Nutzung von Cloud-Anbietern liegt Amazon AWS einsam an der Spitze (63 Prozent) vor Google (20 Prozent) und Azure (18 Prozent); dahinter: Red Hat Openshift (10 Prozent) und die Oracle-Cloud (6 Prozent).

https://snyk.io/blog/jvm-ecosystem-report-2018

25. Oktober 2018

CodeOne und EclipseCon

Zeitgleich finden diese Woche die Oracle CodeOne und die EclipseCon Europe statt. Die CodeOne in San Francisco, Nachfolger der JavaOne, aber als generelle Oracle–Entwicklerkonferenz gedacht, ist nach dem Wirbel um die (reine?) Umbenennung doch positiv in der Java-Community aufgenommen worden. Die meisten „üblichen Verdächtigen", die auch auf der JavaOne regelmäßig Vorträge gehalten oder die Community zusammengebracht haben, wollen auch die neue Konferenz nutzen, um Java voranzutreiben – und haben ordentlich „getrommelt" und Vorträge eingereicht. Das meiste ist dann auch wie zuvor auf der JavaOne, inklusive Community-Keynote – und es ist keine Rede mehr davon, dass Oracle Java sterben lassen möchte, wie manche gemutmaßt hatten. Was ja bei dem von Oracle immer noch betriebenen Aufwand rund um Java und bei den vielen neuen Projekten (FnProject, Helidon etc.) auch merkwürdig wäre. Wobei die Themenvielfalt natürlich größer wird. Aber sich nur mit Java beziehungsweise der JVM auseinanderzusetzen, reicht halt nicht mehr; und das spiegelt die CodeOne offensichtlich wider. Apropos FnProject: Oracle plant im Jahr 2019 eine darauf basierende „Oracle Functions"-Cloud; Open Source und Geldverdienen gehen Hand in Hand, das hat auch Oracle verinnerlicht. Das neue Supportmodell und der Release-Zyklus für Java sind natürlich immer noch ein heißes Thema bei der Java-Keynote. Mark Reinhold sagt dazu jedoch nur „Java is still free" (wie die Java Champions). Sprich: Wem das Oracle-Modell nicht passt, der findet Alternativen – oder kann es selbst angehen. Um die Zukunft des JDK geht es natürlich auch: Die Projekte „Amber" (Produktivitäts-steigernde Features wie „Switch Expressions") und „Panama" (Integration mit Non-Java/native APIs) laufen weiter, in diesem Jahr dazugekommen ist das Projekt „Loom" (leichtgewichtige „user-mode threads" aka „Fibers").

https://www.oracle.com/code-one

Die EclipseCon Europe in Ludwigsburg ist ebenfalls ein voller Erfolg. Die Themen „Jakarta EE" sowie „MicroProfile" ziehen viele zu Eclipse und natürlich zu den Konferenzen. Vieles dreht sich entsprechend um den Fortschritt bei Jakarta EE und den neuen Spezifikations-Prozess. Aber auch reine Community-Themen wie AdoptOpenJDK finden reichlich Beachtung (ebenso wie natürlich auch IoT).

https://www.eclipsecon.org/europe2018

Andreas Badelt
stellv. Leiter der DOAG Java Community
andreas.badelt@doag.org

Andreas Badelt ist stellvertretender Leiter der DOAG Java Community. Er ist seit dem Jahr 2001 ehrenamtlich in der DOAG Deutsche ORACLE-Anwendergruppe e.V. aktiv, zunächst als Co-Leiter der SIG Development und später der SIG Java. Seit dem Jahr 2015 ist er stellvertretender Leiter der neugegründeten Java Community innerhalb der DOAG. Beruflich hat er seit dem Jahr 1999 als Entwickler und Architekt für deutsche und globale IT-Beratungsunternehmen gearbeitet und ist seit dem Jahr 2016 als freiberuflicher Software-Architekt unterwegs („www.badelt.it").

Java-9-Standardbibliothek

gelesen von Dr. Stefan Pfeiffer

Das Buch stammt vom Autor des Klassikers „Java ist auch eine Insel" und setzt dort auf, wo grundlegende Kenntnisse über die Syntax der Programmiersprache Java und das objektorientierte Programmierparadigma aufhören. Dabei steht das Buch seinem Vorgänger in Sachen Umfang in nichts nach: Auf fast 1.500 Seiten beschreibt es die Dinge, die erforderlich sind, um die Sprache Java sinnvoll zum Lösen praktischer Probleme einsetzen zu können, und spannt den Bogen von grundlegenden Operationen mit Strings, Kollektionen, Datum und Zeit, Dateien und Threads über den Umgang mit XML oder JSON, Netzwerk-Programmierung, Web-Services, Reflection, JDBC, Swing und JavaFX bis hin zur Einbindung von Skript-Sprachen und das Java Native Interface (JNI). In diesem weiten thematischen Spektrum wird zwischen Kapiteln für Einsteiger und Kapiteln mit weitergehenden Details für Interessierte, die aber für das grundlegende Verständnis nicht relevant sind, unterschieden.

Die tiefergehenden Änderungen zwischen Java 8 und Java 9 – vor allem das Modulsystem – werden in einem gesonderten Kapitel zu Beginn behandelt. Die restlichen Kapitel sind deswegen auch noch für Java 8 oder für die Versionen nach Java 9 wertvoll. An den Stellen, an denen die Java-Standard-Bibliothek dann doch nicht ausreicht, macht der Autor sehr pragmatisch auch Abstecher zu externen Bibliotheken oder Tools, die in vielen realen Projekten wahrscheinlich sowieso genutzt werden. Hier seien Maven, Google Guava, XStream, Protocol Buffers, verschiedene Bibliotheken für den Umgang mit CSV-, HTML- und Officedaten, Jackson für den Umgang mit JSON-Daten und HTTP-Client/Server-Bibliotheken genannt.

Dass Java trotz seines neuen, schnellen Release-Zyklus weiterhin fest in der Legacy-Welt verankert ist, zeigen auch die Kapitel über Remote Method Invocation (RMI), SOAP-Webservices, JMS, Ant und sogar das Security-Konzept von Java-Applets. So versucht dieses Buch, eine umfassende Brücke zwischen einer antiquiert anmutenden Welt der Java-Entwicklung und den neuesten APIs und Konzepten aus Java 8+ zu schlagen. Die Entscheidung, weiterhin Swing ausführlich vorzustellen, gleichzeitig aber JavaFX nur ein Viertel des Seitenumfangs über Swing einzuräumen (also so viel wie AWT), zeigt deutlich die schwierige Situation auf, in der sich Java als Sprache zur Entwicklung von Desktop-Applikationen befindet.

Allein aufgrund seines Umfangs ist dieses Werk sicherlich nicht dafür gedacht, von vorne bis hinten in einem Rutsch gelesen zu werden. Es dient eher dazu, kapitelweise einen Zugang zu einem noch unbekannten Gebiet innerhalb der Standard-Bibliothek zu finden, um dann selbst zu entscheiden, ob die anstehenden Aufgaben mit

eventuell schwergewichtigen 3rd-Party-Bibliotheken gelöst werden sollen oder ob die Funktionalitäten in der mitgelieferten Standard-Bibliothek nicht bereits ausreichen.

Wer „Java ist auch eine Insel" bereits im Bücherregal stehen hat, ist sicherlich gut beraten, sich diese „zweite Insel" danebenzustellen. Auch wer feststellt, dass er aus Mangel an Kenntnissen über die umfangreichen Möglichkeiten der Standard-Bibliothek von Java zu oft zu externen Abhängigkeiten im Projekt greift, kann hier einen tiefen und ausführlichen Einblick erhalten.

Dr. Stefan Pfeiffer
stefan@fam-pfeiffer.de

Autor: Christian Ullenboom
Titel: Java SE 9-Standard-Bibliothek
Verlag: Rheinwerk Computing
Umfang: 1.447 Seiten
Preis: 49,90 Euro
ISBN 978-3-8362-5874-5

Markus' eclipse-Corner

Im September 2017 gab Oracle bekannt, Java EE an die Eclipse Foundation abzugeben [1]. Was die Community lange gefordert hat, sollte nun wahr werden: Oracle gibt die Alleinherrschaft auf, jeder kann sich einbringen, endlich eine Plattform von der Community für die Community. Ein Kommentar von Markus Karg von der Java User Group Goldstadt.

Doch klappt das auch alles so einfach, wie man es sich als Laie vorstellt? Wer die EclipseCon 2017 in Ludwigsburg *[2]* besucht hat, wird angesichts der zurückhaltenden Worte seitens Oracle erste Zweifel bekommen haben. In einem Jahr wolle man es schaffen, Java EE 8 unter neuer Flagge zu re-releasen. Wie, das soll alles sein? Für uns vom iJUG war klar, dass wir an der Sache dranbleiben wollten, kritisch berichten und, wenn es geht, aktiv mitarbeiten. Dies ist mit ein Grund, weshalb es nun diese Kolumne gibt – und einiges mehr. Doch der Reihe nach …

Die Eclipse Foundation (EF) war der Meinung, der Name „Java EE" sei negativ besetzt und daher müsse ein neuer her. Die Suche war nicht einfach, da aus verschiedenen juristischen Gründen eine ganze Menge von Namen und Logos nach Meinung der EF nicht akzeptabel waren. Entsprechend war die Auswahl bei der letztendlichen Wahl dann recht gering, um nicht zu sagen, bereits durch die vorgegebenen Kriterien zumindest deutlich gelenkt. Als Ergebnis kamen der Name „Jakarta EE" heraus *[3]*, den die Apache Foundation spendierte, und ein Segelschiff-Logo. Eine Verwechslungsgefahr des Gewinner-Logos mit dem der Fa. Norgine wurde seltsamerweise aber nicht bemängelt.

Als Nächstes brauchte das Schiff einen Kurs, denn auch wenn EF-Präsident Mike Milinkovich auf der ECE 2017 die EF als den Hort der Innovation bewarb, war (und ist) man bei der EF doch recht planlos in Bezug darauf, was man denn nun mit der Prise anfangen soll. So wurde zunächst eine Umfrage initiiert *[4]*, mit dem Erfolg, dass man die Zukunft der Plattform in der Cloud sieht. Was das wohl für die Rückwärtskompatibilität bedeutet? Einig sind sich hierbei die meisten, dass man zugunsten dieser Zukunft lieber früher als später alte Zöpfe abschneiden sollte. Je nach Provenienz gehen die Rufe in Richtung CORBA, EJB oder des nativen Komponentenmodells von JAX-RS. Klar ist, CDI soll neuer Kern sein.

Im Mai stellt sich bei der EF Ernüchterung ein *[5]*, als man merkt, dass die Zeit zur nächsten EclipseCon Europe knapp wird, aber erst ein Teil der Sourcen migriert ist, ohne jegliche Historie (die wird es auch nicht geben, da zu viel Arbeit), ohne TCK (das soll später kommen) und ohne Spezifikationen (die kommen vielleicht niemals). So hatten wir uns die Migration nicht vorgestellt. Ein harter Fork gegen den Willen von Oracle hätte kaum schlechter ausgesehen …

Die EF gibt sich ja gerne den Anschein einer demokratischen Vereinigung – faktisch stimmt das nicht. Sie organisiert vielmehr einen runden Tisch für Industrie-Unternehmen, die dann mehr oder weniger Beteiligungs- und Stimmrechte erhalten. Mit Anwendervereinigungen wie dem iJUG hat man da so seine liebe Not. So ist auch nach der Wahl zum neuen Jakarta EE Committee *[6]* klar, dass Oracle, IBM und Red Hat weiterhin das Sagen haben, während Ehrenamtliche eher als Gasthörer geduldet sind. Schade, hier wurde eine große Chance zur Bildung einer Community vertan – aber so ist die EF nun mal aufgestellt: Wes Brot ich ess, des Lied ich sing.

Daran ändert auch meine ständige Nörgelei im Forum nichts, daher beschließen wir in der iJUG-Arbeitsgruppe, unser Verband möge sich formell um Mitgliedschaft in der EF bemühen, sodass wir politisch den kapitalistischen Interessen der Konzerne etwas Einhalt gebieten können. Gesagt getan: Am 29. August teilt die EF mit *[7]*, dass der iJUG nun Mitglied ist. Umgekehrt hat der iJUG auch die EF aufgenommen, wobei Letztere kein Stimmrecht hat – sonst wäre unsere Unabhängigkeit ja ad absurdum geführt.

Eine eigene Pressemeldung war es der EF dann wert, dass endlich im Sommer der Quellcode von GlassFish da ist *[8]*. Schön, schön. Auch das TCK ist da. Aber noch immer als Monolith, und immer noch keine Spezifikationen. Dieses Jahr wird das vermutlich nichts mehr. Puh, enttäuschend. Dank der im September vom EF-Marketing durchgedrückten Konvention, dass alle APIs zwingend nun per „Jakarta" in Maven Central stehen müssen, was eine Höllenarbeit ist (alleine für Jersey dauerten die Anpassungen mehrere Tage), ist es fraglich, ob im Oktober auf der ECE 2018 wirklich das Ende der Migration verkündet werden kann. Zum Zeitpunkt des Schreibens (Mitte Oktober) bin ich der einzige, der überhaupt etwas auf Maven Central hochgeladen hat: JAX-RS 2.1.1 steht seit 10. September bereit, JAX-RS 2.1.2 seit 6. Oktober.

Der Grund dafür ist recht simpel: komplette Überforderung seitens der EF. Diese hat 330 aktive Projekte mit 1.120 Repositories *[9]* zu verwalten. Da die EF (verglichen mit der Apache Foundation) extrem bürokratisch ist, geht nichts mehr, wie man so schön sagt. Hinzu kommt, dass die Eclipse Foundation so clever war, Oracle zunächst alle Machtpositionen zu überlassen – doch Oracle hat nicht genug Personal. So gab es kürzlich einen Hilferuf des EE4J PMC, ob jemand bei einer ganzen Anzahl von Projekten nicht die Leitung übernehmen wolle, da sowohl die (Oracle-) Projektleiter als auch sämtliche (Oracle-) Committer dieser Projekte auf Kommunikationsversuche nicht reagierten. Es zeigt sich also, die lange Zeit, die Oracle für Java EE 8 brauchte, hatte Gründe, und diese Gründe werden nun angesichts öffentlicher Kommunikationskanäle und von jedermann einsehbarer Contribution-Statistiken offenbar: Personalmangel.

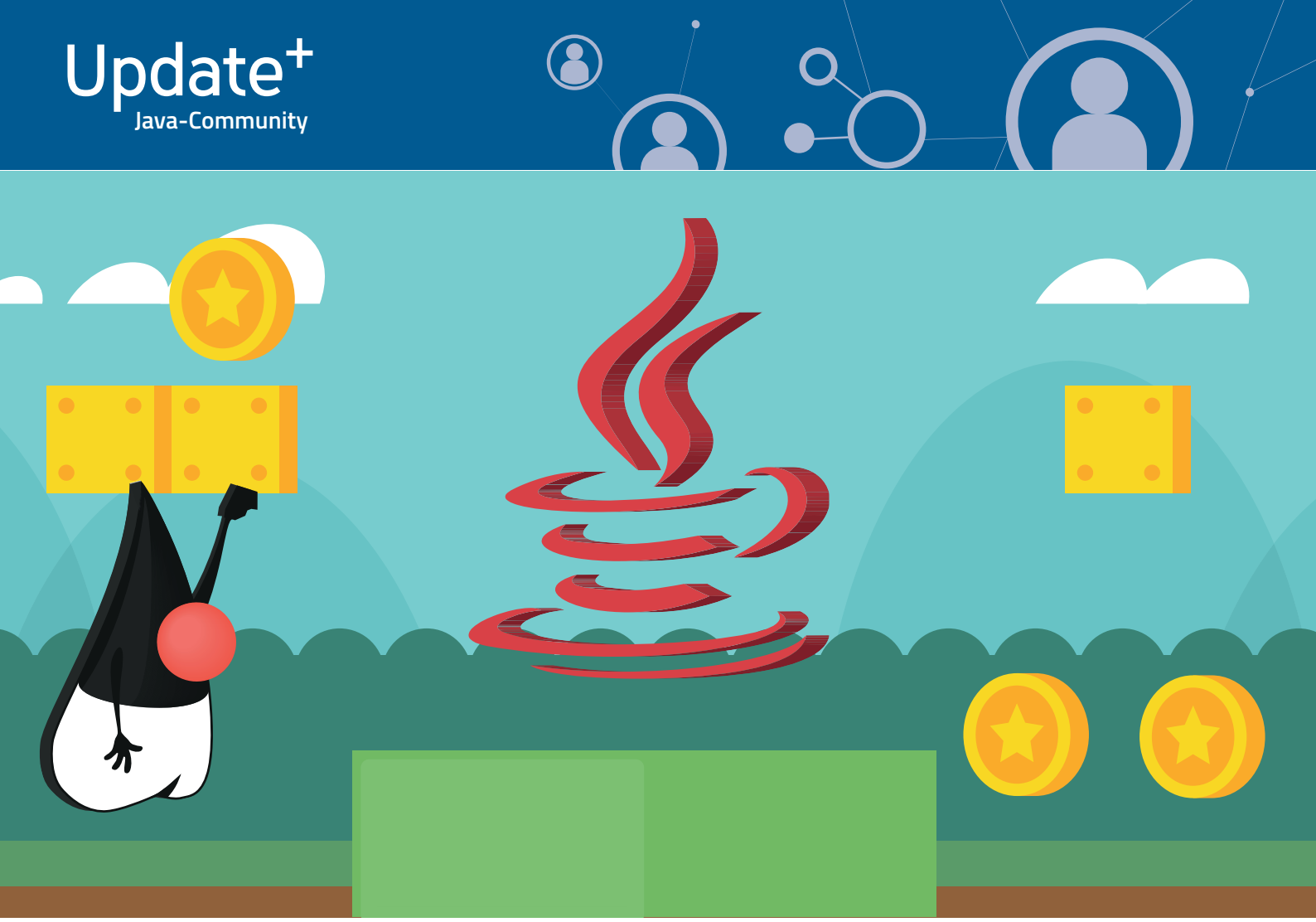
Schauen wir trotzdem zuversichtlich nach vorn: Ende Oktober ist der iJUG auf der EclipseCon Europe 2018 vertreten *[10]*; wer möchte, ist herzlich eingeladen, nach einer kritischen Session zum Thema „Jakarta EE Community With JAX-RS Team" mit dem Autor dieser Kolumne zu diskutieren: über das, was bisher geschah, über das, was noch kommt, und darüber, wie die Community die Zukunft aktiv mitgestalten kann.

Referenzen

[1] *https://mmilinkov.wordpress.com/2017/09/12/java-ee-moves-to-the-eclipse-foundation*
[2] *https://www.eclipsecon.org/europe2017/news/eclipsecon-ludwigsburg-great-partners*
[3] *https://mmilinkov.wordpress.com/2018/02/26/and-the-name-is*
[4] *https://jakarta.ee/news/2018/04/24/jakarta-ee-community-survey*
[5] *https://www.eclipse.org/community/eclipse_newsletter/2018/may*
[6] *https://www.eclipse.org/org/press-release/20180731-jakartaEE_committee_election.php*
[7] *https://globenewswire.com/news-release/2018/08/29/1558325/0/en/Eclipse-Foundation-Welcomes-16-New-Members-As-Jakarta-EE-and-Eclipse-IoT-Developer-Communities-Surge.html*
[8] *https://blogs.eclipse.org/post/tanja-obradovic/welcoming-glassfish-eclipse-foundation*
[9] *https://blog.benjamin-cabe.com/2018/09/04/how-many-lines-of-open-source-code-are-hosted-at-the-eclipse-foundation?PageSpeed=noscript*
[10] *https://eclipsecon.org/europe2018*

Markus Karg
markus@headcrashing.eu

Markus Karg ist Entwicklungsleiter eines mittelständischen Softwarehauses sowie Autor, Konferenzsprecher und Consultant. JAX-RS hat der Sprecher der Java User Group Goldstadt von Anfang an mitgestaltet, zunächst als freier Contributor, seit JAX-RS 2.0 als Mitglied der Expert Groups JSR 339 und JSR 370.



Version 11: Next Level Java

Falk Sippach, Orientation in Objects GmbH

Pünktlich sechs Monate nach Java 10 ist Ende September bereits Java 11 [1] erschienen. Allerdings waren die letzten Wochen und Monate stark geprägt von den Debatten zu Oracles neuer Support- und Lizenzpolitik sowie der Frage, ob Java beziehungsweise das JDK überhaupt kostenlos bleiben. Die Informationen zu den neuen Features und die Änderungen an der Syntax sowie der Klassen-Bibliothek gingen dabei etwas unter. Dieser Artikel beleuchtet beide Themen näher.

Mit Java 11 führt Oracle eine neue Lizenz- und Support-Strategie ein. Freie Updates für die letzte Long-Term-Support-Version (LTS-Version) 8 wird es ab Januar 2019 nicht mehr geben. Zudem ist ab Java 11 das Oracle JDK nur noch in der Entwicklung kostenlos nutzbar, in Produktion muss ein Support-Vertrag *[2]* mit Oracle gegen entsprechende Gebühren abgeschlossen werden.

Allerdings ist das Oracle JDK ab sofort binär kompatibel mit dem OpenJDK, das als Open Source unter der GNU General Public License v2 (with the Classpath Exception - GPLv2+CPE) veröffentlicht ist. Um das JDK weiterhin in Produktion kostenlos einsetzen zu können, kann man also ab Java 11 das OpenJDK verwenden. Allerdings bietet Oracle immer nur Updates für die aktuelle OpenJDK-Version; mit dem nächsten halbjährlichen Major-Release muss man also potenziell die Anwendung bereits auf die nächste Version updaten.

Alternative Anbieter wie Azul und IBM haben im Rahmen des AdoptOpenJDK-Projekts allerdings bereits angekündigt, die LTS-Versionen (etwa Java 11) auch bis zu vier Jahre mit kostenlosen Updates zu versorgen. Auch für Kunden von diversen Linux-Distributionen (wie Red Hat Enterprise Linux) wird das OpenJDK 8 noch mindestens vier Jahre im Rahmen der Betriebssystem-Support-Verträge kostenlos mit Updates versorgt. Für Anwender der Java-Plattform gibt es also genügend Optionen zwischen kommerziellen Lösungen von Oracle (und auch Azul Zulu, IBM SDK etc.) sowie weiterhin freien Varianten mit dem OpenJDK und dem verlängerten Support des AdoptOpenJDK-Projekts *[3]* oder im Rahmen anderer Lizenzvereinbarungen (Red Hat Enterprise Linux *[4]*, IBM SDK *[5]* etc.). Allerdings gehört et-

was Mut dazu, Oracle den Rücken zu kehren und alternativen Java-Plattformen eine Chance zu geben.

Neue Features

Die Neuerungen von Java 11 fallen relativ übersichtlich aus. Das verwundert wegen des kurzen Zeitraums seit der Veröffentlichung der vorangegangenen Version nicht weiter. Trotzdem wurden insgesamt mehr als 2.400 Tickets geschlossen *[6]*. Fast 80 Prozent davon hat natürlich Oracle bearbeitet, an den restlichen 20 Prozent waren viele andere Firmen wie SAP, Red Hat, Google oder IBM beteiligt. Ein Großteil der Neuerungen wurde im Rahmen der folgenden Java Enhancement Proposals (JEPs) umgesetzt:

- 181: Nest-Based Access Control
- 309: Dynamic Class-File Constants
- 315: Improve Aarch64 Intrinsics
- 318: Epsilon: A No-Op Garbage Collector
- 320: Remove the Java EE and CORBA Modules
- 321: HTTP Client (Standard)
- 323: Local-Variable Syntax for Lambda Parameters
- 324: Key Agreement with Curve25519 and Curve448
- 327: Unicode 10
- 328: Flight Recorder
- 329: ChaCha20 and Poly1305 Cryptographic Algorithms
- 330: Launch Single-File Source-Code Programs
- 331: Low-Overhead Heap Profiling
- 332: Transport Layer Security (TLS) 1.3
- 333: ZGC: A Scalable Low-Latency Garbage Collector
- 335: Deprecate the Nashorn JavaScript Engine
- 336: Deprecate the Pack200 Tools and API

Aus Entwicklersicht sind nur einige wenige Punkte wirklich relevant. So wurde im JEP 323 eine Erweiterung der in Java 10 eingeführten Local-Variable Type Inference umgesetzt. Type Inference ist das Schlussfolgern der Datentypen aus den restlichen Angaben des Quellcodes und den Typisierungsregeln heraus. Das spart Schreibarbeit und bläht den Quellcode nicht unnötig auf, wodurch sich wiederum die Lesbarkeit erhöht. Seit Java 10 können lokale Variablen mit dem Schlüsselwort „var" folgendermaßen deklariert werden *(siehe Listing 1)*.

Neu in Java 11 ist, dass man nun auch Lambda-Parameter mit „var" deklarieren kann. Das mag auf den ersten Blick nicht sonderlich sinnvoll erscheinen, da man den Typ von Lambda-Parametern sowieso weglassen und über die Typ-Inferenz ermitteln lassen kann. Nützlich wird die Erweiterung aber für die Verwendung von Type Annotations wie „@Nonnull" und „@Nullable" *(siehe Listing 2)*.

Die nächste interessante Neuerung ist die Standardisierung des bisher noch experimentellen neuen HTTP-Client-API, das mit JDK 9 eingeführt und in JDK 10 aktualisiert wurde (JEP 110). Neben HTTP/1.1 werden nun auch HTTP/2, WebSockets, HTTP/2 Server Push, synchrone und asynchrone Aufrufe sowie Reactive Streams unterstützt. Garniert mit einem gut lesbaren Fluent-Interface wird die Verwendung von anderen HTTP-Clients (wie Apache) dann in Zukunft voraussichtlich obsolet sein *(siehe Listing 3)*.

Durch den JEP 330 (Launch Single-File Source-Code Programs) lassen sich jetzt Klassen starten, die noch nicht kompiliert wurden. Programme mit einer einzigen Datei sind heutzutage beim Schreiben kleiner Hilfsprogramme üblich und insbesondere die Domäne von Skript-Sprachen. Nun kann man sich auch in Java die unnötige Arbeit sparen und das verringert zugleich die Einstiegshürde für Java-Neulinge *(siehe Listing 4)*. Auf unixoiden Betriebssystemen können Java-Dateien als Shebang-Files sogar direkt ausgeführt werden *(siehe Listing 5)*.

```
// Funktioniert seit Java 10
var zahl = 5; // int
var string = "Hello World"; // String
var objekt = BigDecimal.ONE; // BigDecimal
```

Listing 1

```
// Inference von Lambda Parametern
Consumer<String> printer = (var s) -> System.out.println(s); // statt s -> System.out.println(s);

// aber keine Mischung von "var" und deklarierten Typen möglich
// BiConsumer<String, String> printer = (var s1, String s2) -> System.out.println(s1 + " " + s2);

// Nützlich für Type Annotations
BiConsumer<String, String> printer = (@Nonnull var s1, @Nullable var s2) ->
System.out.println(s1 + (s2 == null ? "" : " " + s2));
```

Listing 2

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
  .uri(URI.create("http://openjdk.java.net/"))
  .build();
client.sendAsync(request, asString())
  .thenApply(HttpResponse::body)
  .thenAccept(System.out::println)
  .join();
```

Listing 3

```
# java HelloWorld.java
// statt
# javac HelloWorld.java
# java -cp . hello.World
```

Listing 4

```
#!/path/to/java --source version
[...]
```

Listing 5

Weitere erwähnenswerte Änderungen sind die Unterstützung des Unicode-10-Standards und die Integration der bisher nur mit einer kommerziellen Lizenz verwendbaren Profiling-Tools „Mission Control" und „Flight Recorder" in das OpenJDK (sie wurden bisher nur mit dem Oracle JDK ausgeliefert). Ziel des Flight Recorder ist das möglichst effiziente Aufzeichnen von Anwendungsdaten, um bei Problemen die Java-Anwendung und die JVM analysieren zu können.

Etwas verrückt scheint zunächst der JEP 318 (Epsilon: A No-Op Garbage Collector) zu sein. Dabei handelt es sich um einen neuen Garbage Collector, der aber gar keine Garbage Collection durchführt (deshalb No-Op, also No-Operation). Interessant ist dieses Verhalten aber für Serverless Functions im Rahmen des Oracle-fn-Projekts. Da es sich hier um sehr kurz laufende Java-Anwendungen handelt, wäre ein zwischenzeitlicher Garbage-Collector-Lauf eher kontraproduktiv und sinnlos, wenn die Anwendung kurz darauf sowieso wieder beendet wird.

API-Änderungen

An der Java-Klassenbibliothek gab es natürlich auch unzählige kleine Änderungen. Besonders viel hat sich bei String und Character getan *(siehe Listing 6)*.

Was entfernt wurde

Die Ankündigungen in Form von Deprecations in den Versionen 9 und 10 sind nun in Java 11 Wirklichkeit geworden. Im JEP 320 wurden diverse Java-Enterprise-Packages aus Java SE entfernt, darunter JAX-WS (XML-basierte SOAP-Webservices inklusive der Tools „wsgen" und „wsimport"), JAXB (Java XML Binding inklusive der Tools „schemagen" und „xjc"), JAF (Java-Beans-Activation-Framework), Common Annotations („@PostConstruct", „@Resource" etc.), CORBA und JTA (Java-Transaction-API).

Neu ist auch, dass das Oracle JDK kein JavaFX mehr enthalten wird (mit dem OpenJDK wurde es übrigens noch nie ausgeliefert). Stattdessen wird JavaFX über OpenJFX *[7]* als separater Download angeboten und kann wie jede andere Bibliothek in beliebigen Java-Anwendungen verwendet werden.

Neben JavaFX ist auch der Support für Applets und Java Web Start eingestellt. Die Open-Source-Community plant allerdings bereits Nachfolge-Projekte *[8]*. Wenn man im Moment noch Java Web Start nutzen möchte, muss man zunächst beim Oracle JDK 8 bleiben und entweder ohne Sicherheits-Updates leben oder für den kommerziellen Support ab 2019 Geld ausgeben.

```
| Welcome to JShell -- Version 11
| For an introduction type: /help intro
// Unicode zu String
jshell> Character.toString(100)
$1 ==> "d"
jshell> Character.toString(66)
$2 ==> "B"

// Zeichen mit Faktor multiplizieren
jshell> "-".repeat(20)
$3 ==> "--------------------"

// Enthält ein Text keine Zeichen (höchstens Leerzeichen)?
jshell> String msg = "hello"
msg ==> "hello"
jshell> msg.isBlank()
$5 ==> false
jshell> String msg = "  "
msg ==> "  "
jshell> msg.isBlank()
$7 ==> true

// Abschneiden von führenden oder nachgelagerten Leerzeichen
jshell> " hello world   ".strip()
$8 ==> "hello world"
jshell> "hello world    ".strip()
$9 ==> "hello world"
jshell> "hello world    ".stripTrailing()
$10 ==> "hello world"
jshell> "hello world    ".stripLeading()
$11 ==> "hello world    "
jshell> "      ".strip()
$12 ==> ""

// Texte zeilenweise verarbeiten
jshell> String content = "this is a multiline content\nMostly obtained from some file\rwhich we will break into
lines\r\nusing the new api"
content ==> "this is a multiline content\nMostly obtained fro ... ines\r\nusing the new api"
jshell> content.lines().forEach(System.out::println)
this is a multiline content
Mostly obtained from some file
which we will break into lines
using the new api
```

Listing 6

Neu in Java 11 als „Deprecated" markiert wurde die JavaScript-Engine Nashorn. Es ist davon auszugehen, dass sie in zukünftigen Java-Versionen verschwinden wird. Nashorn hat sich allerdings nie so richtig als serverseitige JavaScript-Implementierung gegenüber Node.js durchsetzen können. Mit der GraalVM geht Oracle mittlerweile alternative Wege, um andere Programmiersprachen nativ auf der JVM auszuführen.

Übrigens wird es ab Java 11 die Java-Laufzeitumgebung (JRE) nur noch in der Server-Variante und nicht mehr für Desktops geben. Allerdings kann man für Desktop-Anwendungen mit dem Modulsystem und dem Werkzeug „jlink" mittlerweile selbst sehr einfach in der Größe angepasste Laufzeit-Umgebungen erstellen.

Fazit und Ausblick

Die großen Überraschungen sind ausgeblieben; vielmehr finalisiert Java 11 angefangene Arbeiten aus den beiden vorangegangenen Versionen, damit es als LTS-Release für die nächsten drei Jahre gut gewappnet ist. Dazu wurden auch einige alte Zöpfe abgeschnitten und zum Beispiel JavaFX und diverse Java-EE-Packages aus dem JDK entfernt.

Java 12 steht bereits in den Startlöchern und wird voraussichtlich im März 2019 erscheinen. Die Liste der Neuerungen wächst noch, interessant für Entwickler sind aber insbesondere die folgenden beiden; weitere werden sicher in den nächsten Wochen folgen:

- JEP 325: Switch Expressions
- JEP 326: Raw String Literals

Beide Sprach-Features stammen aus sogenannten „Inkubator-Projekten" (Amber *[9]*, Valhalla *[10]*, Loom *[11]*), in denen kleinere, die Entwicklerproduktivität steigernde Sprach- und VM-Features ausgebrütet und dann als Java Enhancement Proposals (JEPs) akzeptiert werden. Switch Statements können demnach in Zukunft auch als Expression verwendet werden, die das Ergebnis des entsprechenden Case-Zweigs direkt zurückliefert *(siehe Listing 7)*.

Mit Raw String Literals wird Java endlich die Möglichkeit bekommen, mehrzeilige Zeichenketten zu definieren, die zusätzlich Escape-Sequenzen (..) ignorieren. Damit lässt sich viel einfacher mit regulären Ausdrücken und Windows-Dateipfaden umgehen. Einzig das Ersetzen von Variablen (String-Interpolation) ist im Moment noch nicht geplant, ein Feature, das alternative Sprachen wie Groovy, Ruby und JavaScript schon länger unterstützen *(siehe Listing 8)*.

Java-Entwickler können sich also auch in den nächsten Jahren auf viele interessante neue Features freuen. Jetzt gilt es allerdings erstmal, die neuen Funktionen von Java 11 auszuprobieren und sich mit den veränderten Bedingungen bei den Lizenzen und beim Support auseinanderzusetzen.

Referenzen

[1] JDK-11-Projektseite: *http://jdk.java.net/11/*

[2] Java-SE-Support-Verträge: *https://blogs.oracle.com/java-platform-group/a-quick-summary-on-the-new-java-se-subscription*


```
int numLetters = switch (day) {
     case MONDAY, FRIDAY, SUNDAY  -> 6;
     case TUESDAY                 -> 7;
     case THURSDAY, SATURDAY      -> 8;
     case WEDNESDAY               -> 9;
};
```

Listing 7

```
Runtime.getRuntime().exec(`"C:\Program Files\foo" bar`);
// statt
Runtime.getRuntime().exec("\"C:\\Program Files\\foo\" bar");

String script1 = `function hello() {
                  print('"Hello World"');
}

hello();`;
// statt
String script2 =  "function hello() {\n" +
                  "   print(\'\"Hello World\"\');\n" +
                  "}\n" +
                  "\n" +
                  "hello();\n";
```

Listing 8

[3] AdoptOpenJDK: *https://adoptopenjdk.net*

[4] RHEL OpenJDK: *https://access.redhat.com/articles/1299013*

[5] IBM SDK: *https://developer.ibm.com/javasdk/2018/04/26/java-standard-edition-ibm-support-statement*

[6] OpenJDK-Community-Zusammenarbeit: *https://blogs.oracle.com/java-platform-group/building-jdk-11-together*

[7] OpenJFX: *http://openjdk.java.net/projects/openjfx*

[8] Opensource Java Webstart: *https://dev.karakun.com/webstart*

[9] Project Amber: *http://openjdk.java.net/projects/amber*

[10] Project Valhalla: *http://openjdk.java.net/projects/valhalla*

[11] Project Loom: *http://openjdk.java.net/projects/loom*

Falk Sippach

falk.sippach@oio.de

Falk Sippach hat mehr als fünfzehn Jahre Erfahrung mit Java und ist bei der Mannheimer Firma OIO Orientation in Objects GmbH als Trainer, Software-Entwickler und Projektleiter tätig. Er publiziert regelmäßig in Blogs, Fachartikeln und auf Konferenzen. In seiner Wahlheimat Darmstadt organisiert er mit Anderen die örtliche Java User Group. Falk Sippach twittert unter @sippsack.

Blockchain – selber machen

Thomas Deniffel, Skytala GmbH

Mit Code können wir uns unmissverständlich ausdrücken. Dies ermöglicht, das Thema „Blockchain" in diesem Artikel klar an einem Beispiel zu erläutern und eine Blockchain inklusive Kryptowährung zu erstellen. Nach den Grundlagen entdecken wir während der Implementierung, welche Komponenten eine Blockchain ausmachen, was sie so besonders und fehlerresistent macht und wie genug Vertrauen in einem anonymen Netzwerk so hergestellt werden kann, dass sich sogar Währungen umsetzen lassen. So kann man während des Artikels mittels Git die Implementierung in der IDE seiner Wahl mit verfolgen.

„Blockchain = Bitcoin". Die Begriffe werden oft fälschlicherweise als Synonym verwendet – die Unterscheidung fällt jedoch einfach: Die Kryptowährung Bitcoin wird mithilfe einer Blockchain implementiert. Das Besondere an solchen Blockchains ist, dass Parteien, die sich gegenseitig nicht vertrauen, sicher miteinander interagieren und handeln können. Im anonymen Internet, das unsere globale Wirtschaft vernetzt, ist das eine vielversprechende Eigenschaft. Kryptographie spielt für die sichere Interaktion anonymer Akteure eine entscheidende Rolle, wodurch der Begriff „Kryptowährung" entstanden ist.

Seit dem Jahr 2009 hinterfragt Bitcoin, was wir unter Geld verstehen. Viele assoziieren Geld mit Banken, Geldnoten und Überweisungen, wobei Bitcoin gleich die ersten beiden Punkte streicht. Doch warum brauchen wir Geld überhaupt? Jeder Mensch arbeitet (vereinfacht) acht Stunden am Tag an einem speziellen Produkt, möchte aber eine Vielzahl verschiedener Dinge kaufen. Daher muss er seine Arbeitszeit gegen andere Produkte eintauschen, die er nicht selbst herstellt. Das machen wir, indem wir unsere Arbeitszeit gegen Geld tauschen, das wir wiederum in Arbeitszeit in Form von Produkten Anderer tauschen.

Das Prinzip funktioniert, sobald alle ihre geleistete Arbeitskraft (das Einkommen) und die verbrauchte Arbeitskraft (die Ausgaben) zuverlässig dokumentieren – ohne zu betrügen. Dazu nutzen wir seit langer Zeit Konten. Ein Konto ist eine unveränderbare Liste, der neue Buchungen – positiv wie negativ – durch Transaktionen hinzugefügt

werden. Eine Transaktion beschreibt immer zwei Kontoeinträge: Abbuchung und Einzahlung in der gleichen Höhe, wodurch das Gesamtsystem konsistent bleibt. Ist die Anzahl der Konten klein genug, kann dazu ein einfaches Blatt Papier dienen, auf dem wir die Transaktionen notieren *(siehe Listing 1)*.

Es funktioniert, solange es wenige Konten gibt und niemand betrügt. Im Großen brauchen wir jedoch Banken, die diese Liste verwalten und überwachen. Wir sehen diese nach wie vor auf unseren Kontoauszügen, gefiltert auf für uns relevante Einträge. Eine Währung lässt sich also durch eine einfache lineare Liste umsetzen. Sie muss nur folgende Eigenschaften erfüllen:

- Unveränderbar
- Erweiterbar
- Invalide Operationen werden erkannt und verworfen

So wie Banken erfüllen Blockchains diese drei Eigenschaften; sie eignen sich daher als Währung. Das Besondere: Statt mit zentralem Transaktionsmanagement funktioniert eine Blockchain dezentral – selbst in einem Netzwerk, in dem sich die Teilnehmer gegenseitig nicht vertrauen. Die Datenstruktur Blockchain kann aber für mehr genutzt werden, etwa für Echtheitsnachweise von Dokumenten oder die Verfolgung von Transporten. Sobald Transaktionen zwischen Parteien stattfinden, die sich nicht kennen und/oder vertrauen, bietet sich ein Szenario für Blockchains.

Die grundlegende Idee

Damit eine Blockchain eine unveränderbare, sequenzielle und erweiterbare Kette von Blöcken bildet, ketten wir die Blöcke, die beliebige Nutzdaten beinhalten, durch Hashes aneinander *(siehe Abbildung 1)*. Möchten wir eine Währung implementieren, befindet sich im Block ein Teil der Liste von Transaktionen (wie im obigen Beispiel). Durch die Verkettung der Blöcke ist es nicht möglich, Elemente zu verändern, zu löschen oder einzufügen, denn jeder Bock speichert den Hash des Vorgänger-Blocks zur Zeit des Hinzufügens. Wird der Vorgänger-Block verändert, ändert sich der Hash, was dann erkannt und verworfen wird. Dazu später mehr beim Konsens-Algorithmus.

Als Use Case für eine Implementierung erschaffen wir nun unsere eigene Währung: JCoin. Die Klassenstruktur sieht wie in *Abbildung 2* aus. Es ist weder möglich noch sinnvoll, den kompletten Code abzudrucken. Es besteht allerdings die Möglichkeit, parallel mitzuentwickeln, indem man das vorbereitete Repository mit „git clone https://github.com/tom-010/jcoin.git" klont. Initial enthält es nur Boilerplate-Code. Man kann sich durch Checkouts im Artikel synchronisieren und die Entwicklung des Codes im Laufe des Artikels beobachten.

```
Alice -> Bob:    100 (Alice: 900, Bob: 1100)
Bob   -> Carol: 50 (Carol: 1050, Bob: 1050)
Carol -> Alice: 500 (Carol: 550, Alice: 1400)
...
```

Listing 1

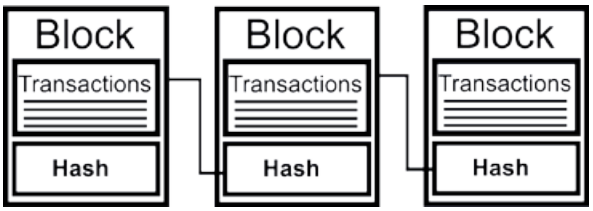


Abbildung 1: Blockchain

Block

Blöcke sind die Grundlage jeder Blockchain. Neben dem Hash sind hier die Transaktionen sowie andere Felder abgelegt, die wir entweder im Laufe des Artikels erkunden oder die als Implementierungshilfe dienen *(siehe Listing 2)*.

Jeder Block (als JSON serialisiert) beinhaltet den Hash des Vorgänger-Blocks, jeder Block besitzt also rekursiv die Hashes aller seiner Vorgänger. Um zu verifizieren, ob eine gegebene Kette nicht verändert wurde, traversieren wir die Liste an Blöcken (die Blockchain), hashen den aktuellen Block und prüfen, ob der berechnete Hash mit dem im nächsten Block übereinstimmt. Die Methode „validChain" zeigt später, wie das in Form von Code aussieht. Man sollte sich an dieser Stelle unbedingt die Zeit nehmen zu verstehen, wie das Hashing die Unveränderbarkeit garantiert.

Trotz Hashes könnte man einen Block ändern, einfügen oder löschen und die folgenden Hashes neu berechnen und aktualisieren. Diesen Angriff verhindert allerdings der Proof of Work, der zusammen mit dem Gossip-Protokoll dafür sorgt, dass manipulierte Ketten verworfen werden. Das funktioniert, weil der Proof of Work mitgehasht wird. Diese Aussage sollte man beim ersten Lesen überspringen und nach dem Artikel nochmal hierher zurückkehren.

Proof of Work

Könnte man ohne Aufwand Blöcke erzeugen, würde das die Synchronisierung im dezentralen Netz unmöglich machen. Zusätzlich

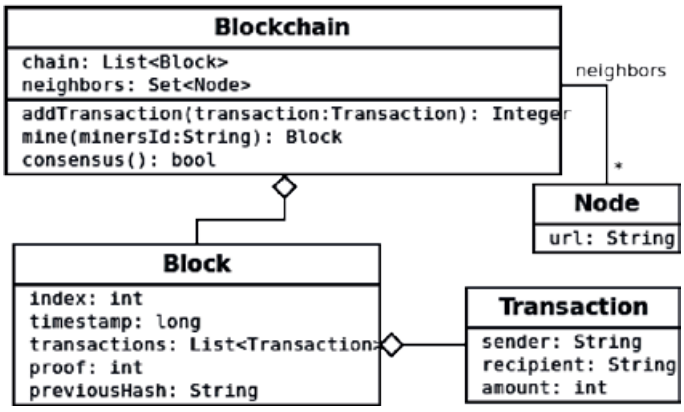


Abbildung 2: Die Klassenstruktur

```
{
  "index": 1,
  "timestamp": 1536418771131,
  "transactions": [...],
  "proof": 196759,
  "previousHash": "74234e...982b90b"
}
```

Listing 2

käme es innerhalb von Sekunden zu einer Inflation, die eine Währung wertlos macht. Der benötigte „Proof of Work" (PoW) verhindert das einfache Erzeugen eines Blocks. Der PoW ist eine Zahl, die zwei Kriterien erfüllt:

- Sehr aufwendig zu erstellen
- Sehr einfach zu kontrollieren

Der Algorithmus, der einen solchen PoW liefert, bedarf viel Rechenarbeit. Das Ergebnis wird dann jedoch von jedem Teilnehmer im Netz überprüft. Beim PoW entdecken wir die Analogie zum antiken Zahlungsmittel Gold: Es ist schwer zu finden (vor allem früher), man kann jedoch einfach durch einen Blick verifizieren, dass es wirklich Gold ist. Aus dieser Analogie stammt die Metapher „minen", was die Ausführung des PoW-Algorithmus meint. Wird ein PoW gefunden, kann ein neuer Block erstellt werden; ein neuer Block wurde also „gemint". Der PoW-Algorithmus ist leicht umzusetzen. Man spezifiziert einen PoW als Zahl x, die mit einer Zahl y (aus dem Vorgänger-Block) multipliziert wird. Der Hash des Ergebnisses muss mit einer 0 enden. Bei gegebenem y suchen wir also ein x, sodass „hash(x * y) = ...0" erfüllt ist *(siehe Listing 3)*.

Lesen wir etwa „y=4" aus dem Vorgänger-Block, ist die Lösung hier „x=37", da der erzeugte Hash mit 0 endet: „hash(4 * 37) = 12....0". Gängige Hashing-Algorithmen sorgen dafür, dass die Wahrscheinlichkeit, einen neuen PoW zu finden, über alle Versuche gleichverteilt ist. Gleichverteilung ist unabhängig von Raum und Zeit gültig, was zu einem fairen System führt. Damit ist die Wahrscheinlichkeit, einen neuen Block zu finden, im Netz gleichverteilt und jeder hat die gleichen Chancen – ohne dass eine Zentrale diese Fairness über ein Protokoll erzwingt.

Um Teilnehmer dazu zu bringen, die Rechenleistung für das Minen bereitzustellen, belohnen wir bei JCoin wie Bitcoin (dessen PoW-Algorithmus heißt übrigens „HashCash" und funktioniert ähnlich) das Mining mit einer Geldeinheit, was die Inflation der Währung an die Schwierigkeit koppelt. Um die Geschwindigkeit der Inflation anzupassen, wird die Schwierigkeit des PoW geändert, indem mehr Nullen (bei JCoin vier) am Ende des Hash gefordert sind. Das lässt die Schwierigkeit exponentiell ansteigen und die Inflation exponentiell verringern (siehe „git checkout pow").

Mining

Beim Mining geschieht die Magie einer Kryptowährung, denn hier wird der Wert der Währung erzeugt. Es besteht aus drei Schritten *(siehe Listing 4)*.

- Den Proof of Work berechnen
- Sich belohnen, indem man sich ein JCoin zuweist
- Einen neuen Block der Blockchain hinzufügen

Der Miner bekommt ein JCoin als Belohnung. Das kann nur über eine Transaktion geschehen, es gibt jedoch keinen Sender, da das Geld erschaffen wird. Deshalb setzt man „0" als Sender, was das Erzeugen der Geldeinheit darstellt: „git checkout mining". Ist ein PoW gefunden, ist man in der Lage, einen neuen Block zu erschaffen. Dazu wird alles bis dato Gesammelte zusammengefügt *(siehe Listing 5)*.

```
Integer proofOfWork(Integer lastProof) {
  Integer proof = 0;
  while(!validProof(lastProof, proof))
     proof += 1;
  return proof;
}

boolean validProof(Integer lastProof, Integer proof) {
  String guessHash = sha256(lastProof.toString()+proof.
toString());
  return guessHash.endsWith("0"); // 0000 später
}
```

Listing 3

```
Block mine(String minersNodeId) {
  // Sync mit dem rest des
  // Netzwerks. Später mehr
  consensus();

  Block lastBlock = lastBlock();
  // proof of first block is 0;
  Integer lastProof =(lastBlock != null) ?
                     lastBlock.getProof() : 0;

  Integer proof = proofOfWork(lastProof);

  // Belohnung für uns!
  addTransaction("0", minersNodeId, 1);

  String previousHash = hash(lastBlock);
  Block block = addBlock(proof, previousHash);

  // Nachbarn benachrichtigen -> Block übernehmen
  for(Node neighbor : getNeighbors())
     neighbor.resolveConflicts();
  return block;
}
```

Listing 4

```
Block addBlock(int proof, String previousHash) {
  // ...
  Block block = new Block();
  // ...
  block.setProof(proof);
  block.setPreviousHash(previousHash);
  block.setTransactions(currentTransactions);
  currentTransactions = new LinkedList<>();

  chain.add(block);
  return block;
}
```

Listing 5

```
Integer addTransaction(Transaction transaction) {
  currentTransactions.add(transaction);
  return lastIndex();
}

class Transaction {
  String sender;
  String recipient;
  Integer amount;
  // ...
}
```

Listing 6

esentri

IT'S IN ALL OF US *TO CREATE*

Jetzt bewerben unter
career@esentri.com

#inallofus
www.esentri.com

```
@RestController
public class BlockchainApi {

  Blockchain blockchain = new Blockchain();
  // Zufälligen Name für uns
  String ownNodeID = randomName();

  @GetMapping("/mine") // Mining anstoßen
  MiningMessage mine() { ... }

  @PostMapping("/transactions/new")
  TransactionAddedMessage newTransaction(Transaction transaction) { ... }
  @GetMapping("/chain") // zum kontrollieren/kopieren für andere
  List<Block> fullChain() { ... }

  @PostMapping("/nodes/register")
  String registerNodes(Node node) { ... }

  @GetMapping("/nodes/resolve") // Konsens-Algorithmus
  ResolvedMessage consensus() { ... }

  @GetMapping("/nodes/all")
  Set<Node> allNodes() { ... }
}
```

Listing 7

```
$ curl -X POST -H "Content-Type: application/json" -d ,{
 "sender": "my-id",
 "recipient": "others-id",
 "amount": 2
}‘ "http://localhost:8080/transactions/new"
```

Listing 8

Neue Transaktion

Die Transaktionen, die einem der Benutzer sendet, werden so lange zwischengespeichert (in „currentTransactions“), bis man einen neuen Block „mint“ *(siehe Listing 6)*. Damit ist eine Transaktion so lange transient, bis ein neuer Block gefunden wurde. Danach wird sie im Block in der Blockchain persistiert.

Weil eine Blockchain eine dezentrale Datenstruktur ist, existiert ein Netzwerk mit gleichberechtigten Nodes. Sie kommunizieren im Beispiel mittels REST-Schnittstellen. Implementiert ein Node diese, ist er Teil des Netzwerks. Dazu kommt Spring zum Einsatz *(siehe Listing 7)*. Nun lassen sich neue Transaktionen anlegen *(siehe Listing 8)*. Um die Transaktion zur Blockchain hinzuzufügen (persistieren), wird ein neuer Block „gemint“ *(siehe Listing 9)*.

Mining bringt uns ein JCoin. Deshalb können wir auch rund um die Uhr in einem Extra-Thread minen. Haben wir zwei Blöcke „gemint“, erhalten wir die Blockchain wie in *Listing 10*.

Konsens

Nun haben wir eine funktionstüchtige Blockchain: Transaktionen werden akzeptiert und neue Blöcke können „gemint“ werden, dazu ist sie unveränderbar und sequenziell. Eine grundlegende Idee fehlt jedoch: Das Ganze soll nicht zentral auf einem Rechner laufen, sondern dezentral. Dazu werden invalide Ketten noch nicht verworfen. Es gibt keine Zentrale wie einen Server, der als „Source of Truth“ dient. Wie können wir erreichen, dass jeder Node die gleiche und korrekte Blockchain repräsentiert – und das

```
$ curl "http://localhost:8080/mine"
{
"message":"New Block Forged",
"Index":0,
"Transactions":[
  {"sender":"my-id","recipient":"others-id","amount":2},
  {"sender":"0","recipient":"my-id","amount":1}],
"proof":4,
"previousHash":"74..90b"
}
```

Listing 9

```
$ curl "http://localhost:8080/chain"
[{
  "index":0,
  "timestamp":1537730860721,
  "transactions":[...]
}]

git checkout rest
```

Listing 10

in einem Netzwerk, in dem sich keiner vertraut? Wir brauchen einen Algorithmus, der Konsens herstellt.

Minen verschiedene Nodes unkoordiniert neue Blöcke, kommt es zwangsläufig zu Konflikten. Denn immer dann, wenn zwei Nodes gleichzeitig einen Block finden, ist zu entscheiden, welcher Block aufgenommen und welcher verworfen wird. Ohne zentralen Server gibt es keine Möglichkeit, pessimistisches Locking zu koordinieren, wodurch zwangsläufig im Zuge des optimistischen Locking gemergt werden muss.

Beim Mergen lassen sich die Blöcke nicht verändern, man muss also Blöcke entweder verwerfen oder aufnehmen. Um zu entscheiden, welche Blöcke übernommen werden, definiert man eine einfache Regel: Die längste valide Blockchain wird übernommen und der Rest

```
boolean consensus() {
  List<Block> newChain = new LinkedList<>();
  Integer maxLength = chain.size();

  for(Node node : neighbors) {
     List<Block> ownChain = node.readChain();
     if(ownChain == null)
        continue; // ignore

     Integer length = ownChain.size();

     // Andere Blockchain länger? Übernehmen!
     // -> Die Idee des Konsens-Algorithmus
     if(validChain(ownChain) && length > maxLength) {
        maxLength = length;
        newChain = ownChain;
     }
  }

  if(!newChain.isEmpty()) {
     this.chain = newChain;
     return true;
  }
  return false;
}

validChain(List<Block> chain) {
  for(int idx = 1; idx < chain.size(); idx++) {
     Block lastBlock = chain.get(idx-1);
     Block block = chain.get(idx);

     // Hash korrekt?
     if(!block.getPreviousHash().
equals(hash(lastBlock)))
        return false;

     // Proof of Work korrekt?
     if(!validProof(lastBlock.getProof(), block.get-
Proof()) )
        return false;
  }
  return true;
}
```

Listing 11

verworfen; neu gefundene Blöcke werden sofort einer Version der Blockchain angehängt, es entstehen also zwei verschiedene valide Blockchains. So schafft man Konsens unter den Knoten im Netzwerk. Der offensichtliche Nachteil: Sobald eine Transaktion in einem Block untergebracht ist, ist nicht mehr sicher, dass der Block auch Teil der Blockchain bleibt, denn es kann passieren, dass der Block irgendwann (eventuell auch erst Jahre später) verworfen wird, sobald sich eine längere Kette mit einem synchronisiert. Konkret sieht das wie in *Listing 11* aus.

„validChain" überprüft, ob eine Blockchain gültig ist. Dazu prüft sie den Hash (die Verkettung ist korrekt) sowie den Proof of Work (die Arbeit wurde auch wirklich geleistet) jedes Blocks. So entsteht die Unveränderbarkeit und invalide Ketten werden verworfen, indem der Node mit der zerbrochenen Kette schlicht von allen Nachbarn ignoriert wird und damit nicht mehr Teil des Netzwerks ist. Hat jedoch ein Nachbar eine valide längere Kette, wird diese übernommen.

Dass es Nachbarn gibt, ist der Schlüssel zum dezentralen System: Es existiert kein zentraler Server, bei dem man sich registrieren muss. Man fügt einfach einen Nachbarn hinzu und fängt an, neue Blöcke zu minen.

Gossip-Protokoll – sag es weiter!

„Mint" man einen Block, ist dies den anderen im Netz mitzuteilen, damit er nicht verworfen wir. Man kennt jedoch nur seinen direkten Nachbarn, weshalb man ihn nicht einfach allen Teilnehmern schicken kann. Dies löst das Gossip-Protokoll: Findet man einen neuen Block, wächst die Blockchain um ein Element; sie ist damit länger als die der Nachbarn. Man benachrichtigt sie, woraufhin diese unsere Blockchain mit dem Konsens-Algorithmus prüfen und übernehmen. Ist das geschehen, benachrichtigen sie wiederum ihre Nachbarn. So findet unser Block seinen Weg durch das Netz, wobei die Geschwindigkeit zeitweise exponentiell steigt.

Das inhärente Problem ist jedoch, dass wir nicht feststellen können, ob jeder den Block erhalten hat. Befindet sich irgendwo ein Subnetz, dessen Nodes in der gleichen Zeit mehr Blöcke findet, wird der Block verworfen, sobald sich das Subnetz mit dem Netz verbindet. Deshalb kann man nie ganz sicher sein, dass eine Transaktion tatsächlich gespeichert ist. Es gibt Workarounds für dieses Problem, doch das Wichtigste ist, dass der Proof of Work schwer genug ist, dass zwei Blockchains nicht besonders weit divergieren können. Das Gossip-Protokoll wurde bereits im Konsens-Algorithmus implizit mit umgesetzt (siehe „git checkout distributed").

Fazit

Das war es: Es gibt eine dezentrale Blockchain, die alle nötigen Eigenschaften erfüllt, um eine Währung zu implementieren. Der Autor empfiehlt, mit Freunden Transaktionen durchzuführen und danach zu versuchen, die Blockchain zu löschen. Sobald nur ein Node nicht heruntergefahren ist, überlebt sie. Diese Fehlertoleranz ist eine der vielversprechenden Fähigkeiten einer Blockchain und wir werden die nächsten Jahre viele interessante Anwendungen entdecken.

Thomas Deniffel
tdeniffel@acm.org

Thomas Deniffel ist CTO bei der Skytala GmbH, die durch neueste Technologien individuelle Software-Projekte umsetzt. Dort ist er für die Auswahl der richtigen Werkzeuge zuständig und evaluiert in diesem Zuge die neuesten Trends. Dabei ist er weder Fan von Hypes noch vom religiösen Bestehen auf Programmiersprachen oder Tools, denn den eigenen Kopf zu verwenden ist gefragt. In seiner Freizeit hält er regelmäßig Präsentationen und Workshops und ist Organisator zweier Meetup-Gruppen mit den Schwerpunkten „Software-Entwicklung", „Software Craftsmanship" und „Clean Code".

Eclipse MicroProfile

Thilo Frotscher, Freiberufler

Die bislang unter „Java EE" bekannte Plattform ist mitten im Umbruch. Nach der Entscheidung von Oracle, die alleinige Kontrolle aufzugeben und die Plattform an die Eclipse Foundation zu übertragen, warten Entwickler nun gespannt auf ein erstes Release unter dem neuen Namen „Jakarta EE". Während der Umfang dieser initialen Version identisch mit Java EE 8 sein soll, arbeitet das Eclipse-MicroProfile-Projekt in rasantem Tempo an potenziellen neuen Features, die schon bald in die Plattform einfließen könnten.

Das Jahr 2016 haben Java-EE-Entwickler in keiner guten Erinnerung. Die Arbeiten an Java EE 8 waren praktisch zum Stillstand gekommen und es gab berechtigte Befürchtungen, dass Oracle das Interesse an der Plattform verloren habe und die Weiterentwicklung einstellen könnte. Es wäre das Ende der beliebten und weitverbreiteten Plattform gewesen und hätte zahllose Unternehmen und Entwickler weltweit in erhebliche Schwierigkeiten gebracht. Glücklicherweise ist dies nicht eingetreten. Im Gegenteil, die Übertragung von Java EE an die Eclipse Foundation wird gemeinhin positiv bewertet und als große Chance aufgefasst.

Als Jakarta EE wird die Plattform nun als Open Source weiterentwickelt. Die Entscheidungsprozesse sind offener, eine Beteiligung der Community ist explizit erwünscht und sogar die Technology Compatibility Kits (TCK) sind inzwischen als Open Source verfügbar. Insbesondere letzterer Schritt gilt als bahnbrechender Meilenstein in der Historie der Plattform.

Einziger Wermutstropfen: Während des (noch laufenden) Übertragungsprozesses zur Eclipse Foundation, der neben allerlei technischen vor allem auch viele juristische Herausforderungen mit sich bringt, ist die Weiterentwicklung der Plattform erneut zum Stillstand gekommen. Faktisch ist seit einigen Jahren technologisch nur wenig Fortschritt zu erkennen, wodurch das erste Release von Jakarta EE einen recht angestaubten Eindruck machen wird.

Zwar soll dieses Initial-Release zunächst im Wesentlichen nur den Nachweis erbringen, dass die Plattform auf der Infrastruktur der Eclipse Foundation gebaut werden kann und dass die Kompatibilität zu Java EE 8 erhalten bleibt. Dennoch wird es im Anschluss ganz entscheidend darauf ankommen, ob zeitnah ein weiteres Release veröffentlicht werden kann, das notwendige Erweiterungen und Modernisierungen enthält.

Die MicroProfile-Initiative

Tatsächlich gibt es bereits eine ganze Reihe potenzieller Neuerungen, die sich zur Integration in Jakarta EE anbieten, denn aus der eingangs geschilderten Ungewissheit über die Zukunft von Java EE ist im Herbst 2016 die sogenannte „MicroProfile-Initiative" entstanden. Dabei handelte es sich um den Zusammenschluss mehrerer Hersteller von Application-Servern, die nicht länger passiv bleiben und die weitere Entwicklung bei Oracle abwarten wollten – letztlich ist das Geschäftsmodell dieser Hersteller untrennbar mit dem Fortbestehen der Java-EE-Plattform verbunden. Ziel der Initiative war daher die Erarbeitung von Vorschlägen für neue Features und Erweiterungen, die die Plattform modernisieren und insbesondere für die Entwicklung von Microservices attraktiver machen sollten. Darauf aufbauend soll ein neues Java EE Profile speziell für diesen Einsatzzweck vorgeschlagen und standardisiert werden. Durch die Kombination der Begriffe „Microservice" und „Java EE Profile" kam somit der Name „MicroProfile" zustande.

In einem ersten Schritt wurden jene bestehenden Java-EE-Technologien identifiziert, die als absolutes Minimum oder kleinster gemeinsamer Nenner für die Entwicklung von Microservices anzusehen sind. Somit bestand das MicroProfile 1.0 lediglich aus JAX-RS, JAX-P und CDI. Da diese Technologien ohnehin bereits in den jeweiligen Application-Servern für Java EE 7 enthalten waren, konnten die Hersteller im Anschluss recht schnell spezielle MicroProfile-Editionen ihrer Server bereitstellen. Dabei handelte es sich schlicht um schlankere Versionen der bekannten Produkte.

Dennoch war ein wichtiger, zukunftsweisender Schritt getan: Die Abkehr vom monolithischen Application-Server wurde spätestens zu diesem Zeitpunkt eingeläutet. Die Zukunft gehört dagegen schlanken und modularen Servern, die nur genau jene Technologien enthalten, die für eine jeweilige Anwendung zur Laufzeit auch wirklich benötigt werden. In diesem Zusammenhang entstanden Lösungen, bei denen Anwendungen und die von ihnen benötigten Application-Server-Module während des Build-Prozesses gemeinsam in ein einziges ausführbares JAR gepackt werden, und somit ein Deployment-Modell, das sich vor allem im Kontext von Docker und Cloud erheblich besser einfindet.

Im Anschluss an dieses erste MicroProfile-Release wurde zu-

Abbildung 1: Die APIs des MicroProfile 2.0

nächst eine Reihe von Anforderungen gesammelt, die ein modernes Framework erfüllen sollte, um eine gute Unterstützung für die Entwicklung von „Cloud Native"-Anwendungen zu gewährleisten. Unter anderem wurden eine Unterstützung für den Zugriff auf Konfigurations-Parameter aus unterschiedlichen Quellen, Fehlertoleranz und Widerstandsfähigkeit, Monitoring und Sicherheit als besonders dringlich erachtet. In der Folge entstanden Vorschläge für entsprechende APIs, die von den Mitgliedern der MicroProfile-Initiative in (für Java-EE-Verhältnisse) bislang ungekannter Geschwindigkeit implementiert und zum Download bereitgestellt wurden.

In weniger als zwölf Monaten erschienen so die MicroProfile-Versionen 1.1 bis 1.4. Das aktuellste Release im Sommer 2018 ist MicroProfile 2.0, das inzwischen aus insgesamt zwölf APIs besteht. Dazu zählen weiterhin die aus Java EE übernommenen JAX-RS, JAX-P und CDI. Sie wurden auf die jeweils neuesten Releases aus Java EE 8 aktualisiert. Hinzu kam JSON-B 1.0 für das Binding zwischen JSON und fachlichen Java-Klassen. Die verbleibenden acht neuen APIs werden im weiteren Verlauf kurz vorstellt *(siehe Abbildung 1)*.

Konfiguration

Für eine weitverbreitete Plattform wie Java EE, die seit vielen Jahren stetig weiterentwickelt wird, ist es erstaunlich, dass so ein zentraler Aspekt wie das Einlesen von Konfigurations-Informationen bislang nicht durch ein standardisiertes API unterstützt wird. Aus diesem Grund werden häufig Bibliotheken wie Apache Commons Configuration oder Apache Delta Spike eingesetzt. Gerade beim Betrieb von Anwendungen in Containern oder in der Cloud ist es jedoch üblich, unterschiedliche Konfigurationsquellen wie Umgebungsvariablen, System-Properties und Dateien mit Default-Konfigurationen einzusetzen. So kam das Thema gleich zu Beginn der MicroProfile-Initiative auf die Tagesordnung: Das Config-API war das erste neu entwickelte API.

Mit dem Config-API ist es nun möglich, Konfigurationswerte mittels CDI in den Anwendungscode injizieren zu lassen. Dabei spielt es keine Rolle, über welche der unterschiedlichen Konfigurationsquellen der Wert letztlich bezogen werden kann. Das Auffinden des Wertes geschieht durch das API und ist daher transparent für den Anwendungsentwickler.

Hinter den Kulissen werden die unterschiedlichen Konfigurationsquellen in festgelegter Reihenfolge untersucht. Natürlich ist es auch möglich, Default-Werte zu definieren, die zum Einsatz kommen, falls ein Konfigurationswert in keiner der Quellen gefunden werden kann. Alternativ können nicht vorhandene Konfigurationswerte in einem „Optional<T>" gekapselt sein *(siehe Listing 1)*. Anstelle des Einsatzes von CDI kann über einen ConfigProvider auch programmatisch auf die Konfiguration zugegriffen werden *(siehe Listing 2)*.

Sollten die vom Config-API unterstützten Konfigurationsquellen nicht ausreichen, lassen sich eigene Konfigurationsprovider mit einem Plug-in-Mechanismus sehr leicht implementieren. Diese werden zur Laufzeit mithilfe des Service-Loader-Mechanismus erkannt und automatisch in Betrieb genommen.

```
@Inject
private Config config;

@Inject
@ConfigProperty(name="service.url")
private String serviceUrl;

@Inject
@ConfigProperty(name="service.username", defaultValue="Alice")
String username;

@ConfigProperty(name="service.password")
private Optional<String> password;
```

Listing 1

```
Config config = ConfigProvider.getConfig();
String serverUrl =
  config.getValue("service.url", String.class);
Optional<String> username =
  config.getOptionalValue("service.username ", String.class);
String[] codes =
  config.getValue("countryCodes", String[].class);
```

Listing 2

```
@CircuitBreaker
@Fallback(QuoteFallbackHandler.class)
public Quote getQuoteFromSupplier() { … }

@Retry(maxRetries = 3, retryOn = {MyRuntimeException.class})
public Person findPersonWithId(Long personId) { … }

@Timeout(500)
@Fallback(fallbackMethod = "getQuotesFallback")
public Response getQuotes() throws InterruptedException { … }
```

Listing 3

Fault Tolerance

Das Fault-Tolerance-API soll es Entwicklern ermöglichen, auf einfache und standardisierte Weise die Widerstandsfähigkeit ihrer Anwendungen zu verbessern. Eine auf HTTP basierende Kommunikation erfolgt offensichtlich über das Netzwerk – viele kommunizierende Services bilden ein verteiltes System, und da kann bekanntermaßen einiges schiefgehen. Es werden also Strategien benötigt, wie mit Verbindungsproblemen, Timeouts und Fehlerfällen umgegangen werden soll, ob gescheiterte API-Aufrufe etwa zu wiederholen sind und falls ja, wie häufig. Zu diesem Zweck definiert das Fault-Tolerance-API unterschiedliche Annotationen und Interfaces, mit deren Hilfe entsprechende Strategien im Code hinterlegt werden können.

Unterstützt werden die Patterns „Circuit Breaker" und „Bulkhead" sowie Strategien für Timeouts, Retries und Fallbacks. Zudem kann mit einer einzigen Annotation erreicht werden, dass Methoden in einem separaten Thread ausgeführt werden. Als Inspiration für das Fault Tolerance API dienten die bekannten Frameworks Hystrix und Failsafe *(siehe Listing 3)*.

JWT Propagation und Rest Client

Im Bereich der HTTP-APIs und somit auch der Microservices haben sich JSON Web Tokens (JWT) als Mittel zur Authentifizierung etabliert. Das JWT-Propagation-API des MicroProfile schlägt daher eine standardisierte Unterstützung von Open ID Connect auf Basis von JWT vor, mit der sich eine rollenbasierte Zugangskontrolle für Microservices umsetzen lässt. Konkret geschieht dies durch die Einführung der Interfaces „JsonWebToken" und „Claim" zur Repräsentation von Token und der darin enthaltenen Informationen. Zusätzlich wird die Annotation „LoginConfig" vorgeschlagen, die in JAX-RS-Ressourcen-Klassen eingesetzt werden soll, um anzuzeigen, welche Form der Authentifizierung für den Zugriff auf die jeweilige Ressource erforderlich ist. Eine dieser möglichen Authentifizierungsformen wäre dann „MP-JWT", also die vom MicroProfile vorgeschlagene Authentifizierung mittels JSON Web Token.

Das MicroProfile-Rest-Client-API zielt darauf, die Implementierung von JAX-RS-Clients zu vereinfachen. Bisher müssen Anwendungsentwickler recht viel technischen Code schreiben, um mithilfe des JAX-RS-Client-API einen Request zu versenden. Insbesondere müssen sie dafür Sorge tragen, dass zu sendende oder zu empfangene Daten korrekt zwischen eigenen fachlichen Klassen und dem für die Kommunikation verwendeten Format – typischerweise JSON – umgewandelt werden.

Mit dem Rest-Client-API soll nun mehr Typsicherheit erzielt und der Anteil technischen Codes reduziert werden. Hierzu würden Entwickler zunächst ein Java-Interface erstellen, um die Schnittstelle des API-Providers auf Client-Seite zu repräsentieren. Die Parameter und Rückgabewerte der einzelnen Interface-Methoden definieren dabei die fachlichen Klassen für die Datenkonvertierung. Weiterhin wird das Interface mit den (bislang nur serverseitig verwendeten) Annotationen wie „@Path", „@GET", „@POST" etc. versehen, um URL-Pfade und Request-Typen zu definieren *(siehe Listing 4)*.

Zur Laufzeit lassen sich dann auf Client-Seite Proxy-Objekte erzeugen, die das zuvor erstellte Interface implementieren, den technischen JAX-RS-Code aber verbergen und somit den Aufruf der API-Operationen durch fachlich anmutende Methoden ermöglichen. Die Proxy-Objekte können entweder mit CDI injiziert oder mithilfe des ebenfalls vorgeschlagenen „RestClientBuilder" programmatisch erzeugt werden. Zudem lassen sich clientseitige „ResponseExceptionMapper" registrieren, die bestimmte Status-Codes empfangener HTTP-Responses in fachliche Exceptions umwandeln.

OpenAPI

Mit dem Trend zu HTTP-APIs entstanden sehr bald auch unterschiedliche Beschreibungssprachen für die API-Spezifikation. Die populärsten sind sicherlich Swagger, RAML und API-Blueprint. In diesem Kontext formierte sich mit der OpenAPI-Initiative ein Verbund namhafter Hersteller, die sich zum Ziel setzten, eine Beschreibungssprache unter dem Namen „OpenAPI" zu standardisieren. Als Startpunkt für den neuen Standard wurde Swagger 2.0 gewählt und darauf aufbauend inzwischen OpenAPI 3.0 veröffentlicht.

Eine API-Spezifikation kann grundsätzlich auf zwei unterschiedlichen Wegen entstehen. Zum einen kann sie manuell erstellt werden – typischerweise unter Zuhilfenahme eines speziellen Editors. Ein alternativer Weg besteht darin, mit der Implementierung des API-Providers zu beginnen und die API-Spezifikation anschließend aus der Implementierung abzuleiten. In diesem Fall werden Werkzeuge eingesetzt, die den bestehenden Code analysieren, also beispielsweise im Falle des Einsatzes von JAX-RS die jeweiligen Annotationen auswerten, um eine OpenAPI-Datei zu generieren.

In der Regel sind jedoch die Informationen, die allein aus dem Code des API-Providers gewonnen werden können, für eine vollständige API-Spezifikation nicht ausreichend; es ist eine Reihe von Zusatz-Informationen anzureichern. Im Umfeld von Swagger entstand dabei für Java-Entwickler eine Sammlung proprietärer Annotationen, die in JAX-RS-Anwendungen zusätzlich eingebaut werden können, um solche Zusatz-Informationen zu hinterlegen.

Dieser Ansatz erscheint vielen Entwicklern nicht ideal, da die vielen zusätzlichen Annotationen den Blick auf das Wesentliche – den eigentlichen Anwendungscode – doch sehr versperren. Ungeachtet dessen hat MicroProfile OpenAPI zum Ziel, diesen Ansatz zu standardisieren, also Annotationen vorzuschlagen, die künftig zur Jakarta-EE-Plattform gehören könnten und die dazu dienen, API-Spezifikationen nach dem OpenAPI-Standard zu erzeugen.

Health Check, Metrics und Open Tracing

Nicht zuletzt sollen gleich drei unterschiedliche MicroProfile-APIs dazu dienen, Anwendungen im laufenden Betrieb zu überwachen. Das Health-Check-API ermöglicht es Entwicklern, einer Anwendung auf einfache Weise spezielle Endpunkte hinzuzufügen, mit denen ihre grundsätzliche Erreichbarkeit und der Status einzelner System-Komponenten geprüft werden kann. Hier ist insbesondere an den Einsatz in Cloud-Infrastrukturen gedacht, bei denen der Zustand einzelner Knoten automatisiert geprüft wird, um diese gegebenenfalls außer Betrieb zu nehmen oder neu zu starten.

Das Metrics-API richtet sich dagegen eher an den Wunsch, kontinuierlich bestimmte Messwerte wie etwa Antwortzeiten, die Anzahl bearbeiteter Requests oder die Auslastung von System-Ressourcen zu überwachen, um daraus bei Bedarf Statistiken zu erstellen oder Trends für künftige Lastspitzen abzulesen. Zu diesem Zweck werden spezielle Annotationen wie „@Timed", „@Metered" oder „@Counted" definiert, mit denen entsprechende Metriken im Code markieren werden können.

Mit dem Open-Tracing-API wird schließlich eine Unterstützung des Open-Tracing-Standards umgesetzt. Dabei handelt es sich um einen Mechanismus, mit dem sich Vorgänge oder Anwendungsfälle in einem verteilten System, wie etwa einer Microservices-Architektur, verfolgen lassen. Dabei können alle beteiligten Services sogenannte „Trace-Informationen" erzeugen, die anschließend gesammelt werden, um zu erkennen, wie sich die Verarbeitung eines Requests über unterschiedliche Knoten ausgebreitet hat. Open Tracing definiert hierzu herstellerneutrale APIs; die Trace-Informationen lassen sich von entsprechenden Tools verarbeiten. Das MicroProfile-Open-Tracing-API ermöglicht es, auf JAX-RS basierenden Anwendungen an solchen Tracing-Szenarien teilzunehmen.

Aktueller Stand

Mittlerweile ist auch die MicroProfile-Initiative der Eclipse Foundation beigetreten. Die Entwicklung der vorgestellten APIs erfolgt somit innerhalb des neu gegründeten „Eclipse MicroProfile"-Projekts. Dies ist ein sinnvoller Schritt, da sämtliche im MicroProfile entstandenen APIs potenzielle Kandidaten für die Aufnahme in künftige Releases von Jakarta EE sind. Eine Übergabe von Features ist sicherlich leichter zu bewerkstelligen, wenn beide Projekte unter dem Dach der Eclipse Foundation beheimatet sind.

Bislang ist noch unklar, in welcher Beziehung die beiden Eclipse-Projekte mittelfristig zueinander stehen sollen und ob etwa Jakar-

```
@Path("/movies")
public interface MovieReviewService {
  @GET
  Set<Movie> getAllMovies();

  @POST
  @Path("/{movieId}/reviews")
  String submitReview( @PathParam("movieId") String movieId, Review review );

  @GET
  @Path("/{movieId}/reviews")
  Set<Review> getAllReviews( @PathParam("movieId") String movieId );
 }
```

Listing 4

ta EE und MicroProfile zusammengeführt werden. Ein alternativer Vorschlag besteht darin, Eclipse MicroProfile als eigenständiges Projekt beizubehalten. Es könnte als „Incubator" dienen, in dem neue Technologien losgelöst von den organisatorischen Prozessen einer großen Plattform in pragmatischer Code-First-Mentalität entstehen. Sobald eine gangbare Lösung implementiert und ausreichend Feedback eingeholt wurde, sollen formale Spezifikation, Standardisierung und das Einfließen in Jakarta EE erst im letzten Schritt erfolgen.

Obwohl also noch nicht geklärt ist, ob und wann die einzelnen MicroProfile-APIs in die Jakarta-EE-Plattform einfließen, können sie dennoch schon heute eingesetzt werden. Hersteller wie Red Hat, IBM, Payara oder Tomitribe bieten Implementierungen des MicroProfile zum Download an. Entwickler, die erste Experimente mit den neuen APIs starten möchten, sollten daher einen Blick auf die aktuellen Releases von Thorntail, Open Liberty, Payara Micro oder Apache TomEE werfen.

Fazit

Während der Fokus im Jakarta-EE-Projekt aktuell noch darauf liegt, den Übergang zur Eclipse Foundation sowohl technisch als auch organisatorisch abzuschließen und ein erstes, zu Java EE 8 funktional identisches Release zu erstellen, entstehen im MicroProfile gleichzeitig eine ganze Reihe von APIs, die potenzielle Neuerungen für Jakarta EE darstellen.

Es bleibt nun abzuwarten, welche davon tatsächlich in die Plattform übernommen werden. Kritische Stimmen bemerken, dass auch im MicroProfile nur wenig Innovation stattfindet, sondern überwiegend Features kopiert werden, die andere Frameworks schon lange bieten. Dem ist entgegenzuhalten, dass Innovation auch gar nicht im Fokus der MicroProfile-Initiative stand. Stattdessen besteht das momentane Ziel darin, die Plattform derart zu erweitern, dass sie moderne Einsatzgebiete wie Microservices, Cloud und Container besser unterstützt. In diesem Kontext erscheint es sinnvoll, Konzepte zu übernehmen, die sich andernorts bewährt haben und Entwicklern daher in ähnlicher Form bekannt sind.

Es wird spannend sein zu beobachten, wie sich Jakarta EE und MicroProfile weiterentwickeln und ob das Momentum des Neuanfangs genutzt werden kann. In jedem Fall bietet der Übergang in die Open-Source-Welt eine große Chance für die Plattform. Es wird nun auf die Community ankommen, sich einzubringen und die lange geforderte Möglichkeit der Beteiligung und Einflussnahme auch zu nutzen.

Thilo Frotscher
feedback@frotscher.com

Thilo Frotscher arbeitet als freiberuflicher Entwickler und Trainer. Als Experte für Enterprise Java, APIs und System-Integration unterstützt er seine Kunden überwiegend durch Projektarbeit, Reviews oder die Durchführung von Schulungen. Thilo Frotscher ist (Co-) Autor mehrerer Bücher über Java EE, (Web) Services und System-Integration, hat zahlreiche Fachartikel verfasst und spricht regelmäßig auf Fachkonferenzen und Schulungsveranstaltungen oder bei Java User Groups.

JavaLand 2019: Größere Beteiligung als je zuvor

Dass die Java-Community absolut fantastisch ist, ist kein Geheimnis und wurde bereits mehrmals auf der JavaLand im Phantasialand unter Beweis gestellt. Doch in diesem Jahr hat sie sich mit mehr als 600 Vortragseinreichungen selbst übertroffen. Damit konnte die Veranstaltung wieder eine überwältigende Beteiligung und erneutes Wachstum verzeichnen. Sie findet vom 19. bis 21. März 2019 statt.

Für das Konferenzprogramm gab es insgesamt 637 Einreichungen, das sind ein Viertel mehr als im vergangenen Jahr. Auch das Newcomer-Programm für Referenten ohne Bühnenerfahrung fand reges Interesse mit 60 Einreichungen (zum Vergleich: 23 in 2018). Beim Schulungstag haben die Stream-Leiter dieses Mal die Qual der Wahl zwischen 37 Einreichungen von 23 Partnern. Das fertige Programm steht unter *„https://programm.javaland.eu/2019/#/schedule"* bereit.

Kafka als IoT-Daten-Plattform

Dr. Ralph Guderlei, eXXcellent solutions GmbH

Viele Firmen setzen auf das Internet of Things, um neue und innovative Produkte zu erstellen. Neben der Entwicklung von Hardware und Sensorik ist das Sammeln und Verarbeiten der anfallenden Sensordaten ein wesentlicher Bestandteil eines IoT-Projekts.

Das Unternehmen des Autors kooperiert seit gut zwei Jahren mit einem lokalen Energieversorger. Dieser will digitale Stromzähler mit Sendern ausstatten, um Verbrauchswerte übertragen zu können. Wenn das Vorhaben vollständig umgesetzt ist, senden rund 150.000 Stromzähler in 15-Minuten-Intervallen Daten. Die gesendeten und aufbereiteten Daten werden dann über Schnittstellen weiteren Systemen zur Verfügung gestellt. Der Energieversorger verspricht sich davon unter anderem eine deutliche Optimierung des Ableseprozesses und eine Verbesserung der Prognosen für den Strom-Einkauf.

Betrachtet man diesen Use Case etwas abstrakter, lassen sich allgemeine Anforderungen an ein System zur Verarbeitung von IoT-Daten ableiten: Eine potenziell große Anzahl von Sensoren versendet mehr oder weniger regelmäßig Daten, die mit geringen Latenzen verarbeitet werden sollen. Oft müssen die gesammelten Daten weiteren Systemen zur Verfügung stehen. Das System sollte in der Lage sein, neue Sensor-Typen einfach integrieren zu können. In diesem Use Case müssen Stromzähler unterschiedlicher Hersteller mit jeweils unterschiedlichen Übertragungswegen unterstützt werden. *Abbildung 1* zeigt die grobe System-Architektur, die sich aus diesen Anforderungen ergibt.

Sensoren übertragen die Daten auf unterschiedlichen Wegen wie LoRaWAN, BLE, WiFi oder GSM. Diese werden zunächst vorverarbeitet, indem man sie etwa in einheitliche Datenstrukturen überführt. Dann werden die eigentlichen Nutzdaten extrahiert und zur weiteren Verwendung bereitgestellt. In unserem Fall kommen die Daten über ein sogenanntes „LoRaWAN-Netzwerk", ein Funk-Netzwerk mit großer Reichweite, das sich besonders für die Übertragung von Sensordaten mit niedriger Datenrate eignet. Diese Art von Funknetz darf von jedermann betrieben werden – vergleichbar mit einem WLAN. Da momentan noch wenig Erfahrung mit dem Betrieb und dem Ausbau dieses Netz-Typs besteht, werden die anfallenden Metadaten gesammelt und zur weiteren Netzplanung herangezogen.

Für die Umsetzung des Systems hat man sich für Apache Kafka entschieden. Neben der weiten Verbreitung und der aktiven Open-Source-Community rund um Kafka war ausschlaggebend, dass mit Kafka Connect und Kafka Streams zwei Teilprojekte existieren, die die Aufgabenfelder „Datenaufnahme und –weitergabe" (Kafka Connect) sowie „Stream Processing" (Kafka Streams) abdecken.

Kafka als Pub/Sub-Queue

Apache Kafka ist zunächst nur eine Pub/Sub-Queue und dient dazu, Daten zwischen den einzelnen Bearbeitungsschritten auszutauschen. Innerhalb von Kafka sind Daten in sogenannten „Topics" organisiert. Diese bilden eine unveränderliche und persistente Sequenz von Nachrichten. Producer fügen neue Nachrichten an das Ende des Topics an, Consumer lesen die Nachrichten sequenziell aus dem Topic. Unterschiedliche Consumer können sich dabei an unterschiedlichen Stellen (Offset) im Topic befinden. Das Format der Nachrichten ist beliebig, typischerweise wird JSON oder Apache Avro als Datenformat verwendet *(siehe Abbildung 2)*.

Kafka ist von Grund auf als verteiltes System ausgelegt. Das bringt zwar bezüglich Skalierbarkeit und Verfügbarkeit deutliche Vorteile, sorgt aber gleichzeitig an der einen oder anderen Stelle für erhöhte Komplexität. Ein wichtiger Aspekt sind sogenannte „Partitionen". Sie dienen dazu, Daten eines Topics über mehrere Kafka-Instanzen zu verteilen.

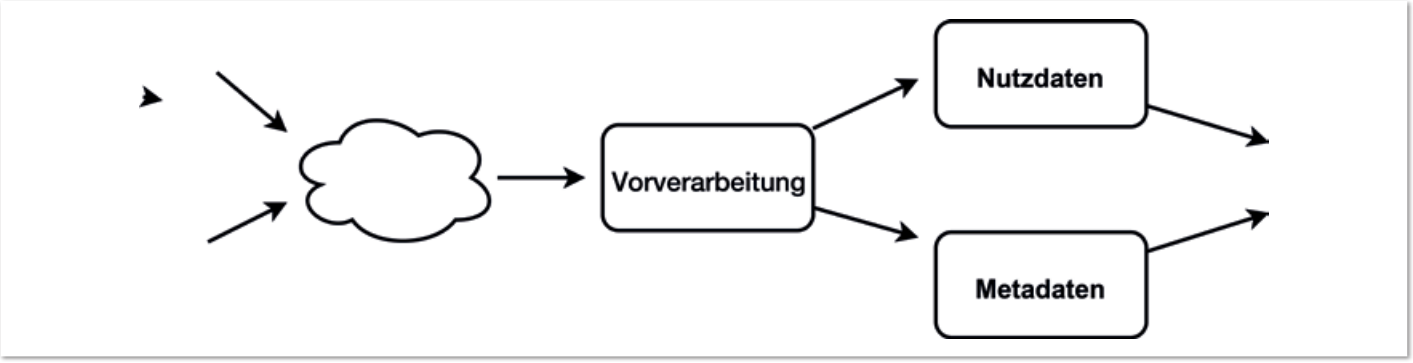


Abbildung 1: System-Architektur

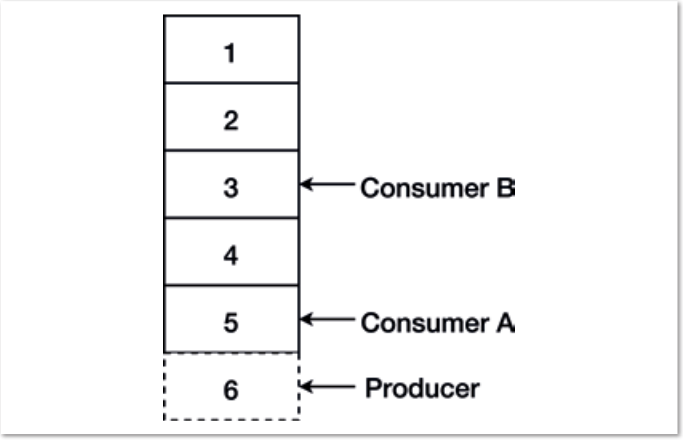


Abbildung 2: Nachrichten-Sequenz

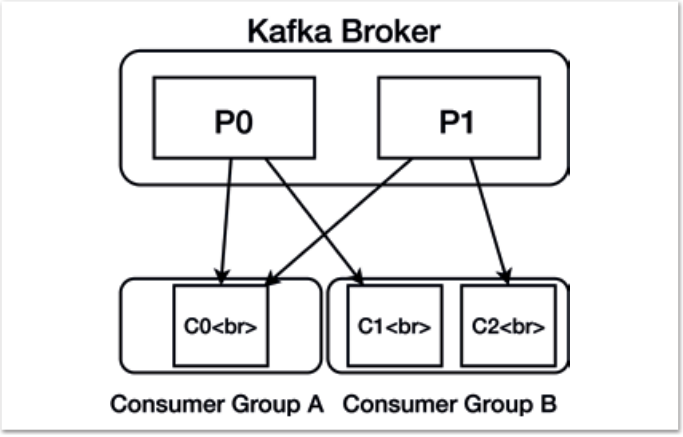


Abbildung 3: Kafka-Cluster

Gleichzeitig bestimmt die Anzahl der Partitionen den Parallelisierungsgrad der Datenverarbeitung in Consumern. Neben dem normalem Publish/Subscribe bietet Kafka die Möglichkeit, mehrere Consumer in einer Consumer Group zusammenzufassen. Die Nachrichten werden dann gleichmäßig über die Consumer einer Consumer Group verteilt und können so parallel bearbeitet werden.

Die maximal mögliche Anzahl der Consumer in einer Consumer Group entspricht der Anzahl der Partitionen eines Topics. Die Anzahl der Partitionen kann zwar im Nachhinein noch geändert werden, das hat aber unter Umständen negativen Einfluss auf die Performance, während der Kafka-Cluster sich neu synchronisiert *(siehe Abbildung 3)*.

Daten rein und raus: Kafka Connect

Eine Möglichkeit, um Daten nach Kafka hinein- oder aus Kafka herauszubekommen, ist Kafka Connect. Dessen Kernstück sind sogenannte „Konnektoren", die es in zwei Ausprägungen gibt: SourceConnectors für die Übernahme von Daten nach Kafka und SinkConnectors für die Übertragung von Daten in andere Systeme. Es existiert bereits eine große Anzahl von Konnektoren, oft als Open-Source-Projekte. Mit diesen lassen sich Datenbanken, andere Message Queues oder andere Systeme rein konfigurativ anbinden. Das Beispiel in *Listing 1* zeigt die Konfiguration für einen Konnektor, der alle Nachrichten des Topics „orders" in eine Tabelle „orders" in einer Sqlite-Datenbank schreibt.

Sollte noch kein Konnektor für ein Drittsystem vorhanden oder man mit der Funktionalität oder der Qualität eines bestehenden Konnektors nicht zufrieden sein, lassen sich eigene Konnektoren mit überschaubarem Aufwand auch selbst erstellen. Ein Konnektor besteht im Wesentlichen aus drei Klassen: dem Rahmen des Konnektors, der Beschreibung der Konfiguration und einer Task-Klasse, die die eigentliche Arbeit übernimmt *(siehe Listing 2)*.

Ein Task besteht aus drei zu implementierenden Methoden, zwei davon werden beim Starten und Beenden des Tasks ausgeführt, die dritte erledigt die eigentliche Verarbeitung der Daten. Im Code-Beispiel ist erkennbar, dass Kafka Connect ein eigenes Format für Daten verwendet. Dieses besteht aus einem Schema zur strukturellen Beschreibung der Daten und den eigentlichen Daten. Durch das Datenschema kann die Logik des Tasks sehr abstrakt und generisch programmiert werden. So wird im Fall des JDBC-Konnektors beispielsweise das notwendige SQL-Statement komplett aus dem Schema abgeleitet.

Kafka Connect stellt dann die Laufzeitumgebung für die Konnektoren zur Verfügung. Unter anderem bietet Kafka Connect auch ein REST-API an. Über dieses können Status-Informationen zu Konnektoren abgefragt und Konnektoren zur Laufzeit hinzugefügt oder entfernt werden. Im sogenannten „Distributed Mode" lässt sich Kafka Connect ebenfalls in einem Cluster betreiben. Kafka Connect verteilt in diesem Modus dann die Konnektoren im Cluster.

Verarbeitung mit Kafka Streams

Nachdem die Daten nun im System sind, geht es im nächsten Schritt darum, den Datenstrom zu verarbeiten. Dazu bietet sich Kafka Streams an, ein API ähnlich dem Java-Streams-API, das speziell

```
{
  "name": "jdbc-sink",
  "config": {
     "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
     "tasks.max": "1",
     "topics": "orders",
     "connection.url": "jdbc:sqlite:test.db",
     "auto.create": "true",
     "name": "jdbc-sink"
  },
  "tasks": [],
  "type": null
}
```

Listing 1

für die Arbeit mit Kafka-Topics entwickelt wurde. Entwickler, die mit dem Streams-API vertraut sind, dürften schnell damit zurechtkommen. Da Kafka Streams nur eine einfache Java-Bibliothek ist, lassen sich Streams-Anwendungen einfach als „executable jar" verpacken und ausführen. Das Beispiel in *Listing 3* zeigt eine einfache Kafka-Streams-Anwendung.

Über ein Fluent-API werden das Quell-Topic und die Art der Deserialisierung der Daten gewählt. Im Beispiel werden die JSON-Nachrichten des Topics automatisch in ein Java-POJO transformiert. Im folgenden Schritt werden dann aus den Rohdaten die eigentlichen Nutzdaten extrahiert. Pro Rohdaten-Objekt entsteht dabei eine Liste von Nutzdaten-Objekten, die mittels „flatMapValues" bearbeitet werden, sodass am Ende eine Sequenz von Nutzdaten-Objekten entsteht. Diese werden am Ende wieder als JSON-Nachrichten im Ziel-Topic publiziert.

Bei spezialisierten Bibliotheken oder Frameworks wie Kafka Streams, die eng mit anderen Systemen verwoben sind, ergeben sich oft Probleme, die mit diesen Werkzeugen erstellten Lösungen mit vertretbarem Aufwand automatisiert zu testen. Für Kafka Streams existiert seit Kafka 1.1.0 eine eigene Test-Bibliothek. Mit deren Hilfe lassen sich einfach Unit-Tests für Streams-Applikationen schreiben. Diese können dann mit den üblichen Mitteln in den normalen Build-Prozess integriert werden *(siehe Listing 4)*.

Im Test werden ein eingebetteter Kafka-Broker gestartet und die zu testende Streaming-Anwendung damit verbunden. Für die Tests können dann Nachrichten in das Quell-Topic der Streaming-Anwendung publiziert werden. Der eigentliche Test besteht aus der Verifikation der Nachrichten im Ziel-Topic der Streaming-Anwendung.

Die reine „1:1"-Verarbeitung von Sensordaten reicht in vielen Use Cases nicht aus. Oft sind in der Verarbeitung Daten aus vorherigen Übertragungen erforderlich. Ein einfaches Beispiel ist die Ermittlung von Paketverlusten. In unserem Fall wird bei jeder Übertragung ein fortlaufender Zähler mitgeführt. Um den Verlust von einem oder mehreren Paketen feststellen zu können, ist der aktuelle Zählerstand mit dem letzten bekannten Zählerstand zu vergleichen.

Normalerweise sind solche Informationen in Datenbanken abgelegt. Kafka Streams bietet dazu ein eigenes Konstrukt: KTables. Dabei

```
public Topology topology(Properties config) {
  final StreamsBuilder builder = new StreamsBuilder();
  final String IN = ...
  final String OUT = ...

  // capture and transform each incoming message
  final KStream<String, DataPoint> values = builder
        .stream(IN, Consumed.with(Serdes.String(),
           new JsonSerde<>(TTNMessage.class)))
        .flatMapValues(ElectricityMessageParser::parse)

  // send data to topic
  values.to(OUT, Produced.with(Serdes.String(),
     new JsonSerde<>(DataPoint.class)));
  return builder.build();
}
```

Listing 3

```
public class MySinkTask extends SinkTask {
  @Override
  public void start(Map<String, String> props) {
        // ...
  }

  @Override
  public void put(Collection<SinkRecord> records) {
     records.forEach(record -> {
        Struct payload = (Struct) record.value();
        Schema schema = record.valueSchema();
        // ... move data out of kafka
     };
  }

  @Override
  public void stop() {
     // ...
  }
}
```

Listing 2

```
Topology topology = ...

Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "test");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "dummy:1234");
testDriver = new TopologyTestDriver(topology, config);
ConsumerRecordFactory<String, String> recordFactory
  = new ConsumerRecordFactory<>(new StringSerializer(), new StringSerializer());

String payload = ...
testDriver.pipeInput(
  recordFactory.create("ttn", "foo", payload)
);

OutputVerifier.compareKeyValue(
  testDriver.readOutput("ttn-metadata", stringDeserializer,
        new JsonDeserializer<>(TtnConnectionInfo.class)),
     "003E94C6AD72F129",
     new TtnConnectionInfo("exxcellent-uno",
           "2017-04-28T11:30:40.62428417Z",
           "eui-0000024b0b03020e"));
```

Listing 4

```
final KStream<String, TtnDeviceInfo> uplinkMessages = ...
  .selectKey((k, v) -> v != null ? v.id : "no key")
  .filterNot((k, v) -> Objects.equals(k, "no key"));

final KTable<String, TtnDeviceInfo> storedValues = builder.table(
  TABLE,
  Consumed.with(Serdes.String(), new ConnectJsonSerde<>(TtnDeviceInfo.class))
);

final KStream<String, TtnDeviceInfo> updated = uplinkMessage
  .leftJoin(
     storedValues,
     TtnDeviceInfo::updatePacketsDropped,
     Joined.with(
        Serdes.String(),
        new ConnectJsonSerde<>(TtnDeviceInfo.class),
        new ConnectJsonSerde<>(TtnDeviceInfo.class))
  );
updated.to(TABLE,
  Produced.with(Serdes.String(),
     new ConnectJsonSerde<>(TtnDeviceInfo.class)));
```

Listing 5

handelt es sich um Topics mit einem Schlüssel, vergleichbar mit dem Primärschlüssel einer Tabelle einer relationalen Datenbank. Über diesen Schlüssel können dann KTables mit normalen Topics oder mit anderen KTables gejoint werden; fast so, wie man das von relationalen Datenbanken kennt *(siehe Listing 5)*.

Das Beispiel zeigt, wie eine Tabelle mit Daten aus einem Topic aktualisiert wird. Das Joinen findet mit der „leftJoin"-Methode statt. Das zweite Argument dieser Methode ist eine Funktion, die die Logik für das Aktualisieren enthält.

KTables sind damit ein bequemes Mittel, um zustandsbehaftete Daten in Kafka abzulegen. Kafka erledigt aber nicht nur die Speicherung, sondern auch die Replikation der KTables in einem Kafka-Cluster. Damit steht die Tabelle auf allen Knoten des Clusters zur Verfügung und kann von mehreren Instanzen einer Streaming-Anwendung verwendet werden.

Lessons learned

Mit Kafka, Kafka Connect und Kafka Streams lässt sich eine Microservices-Architektur entwickeln. Deshalb sollte man von Anfang an auf ein adäquates Monitoring achten. Sämtliche Bestandteile des Kafka-Ökosystems stellen via JMX Metriken zur Verfügung. Diese lassen sich dann von einem Monitoring-System (etwa die zurzeit beliebte Kombination von Prometheus und Grafana) verwenden.

Eine Metrik ist insbesondere für Streaming-Anwendungen interessant: der sogenannte „Consumer Lag". Sie gibt an, wie weit ein Consumer-Offset vom aktuellen Offset entfernt ist. Damit ist der Consumer Lag der wichtigste Hinweis auf eine zu langsame Datenverarbeitung in einem Stream.

Fazit

Das Unternehmen des Autors beschäftigt sich inzwischen seit gut zwei Jahren mit der Entwicklung und dem Betrieb einer Daten-Plattform auf Kafka-Basis. Kafka selbst hat sich dabei überwiegend als stabile und verlässliche Technologie gezeigt. Einzelne Bugs (etwa im Zusammenspiel von Kafka und Kafka Streams) wurden schon in der nächsten Version behoben.

Bei Kafka Connect ist das Bild gemischt. Einerseits lässt sich Kafka Connect bequem einrichten und einfach durch eigene Konnektoren erweitern, andererseits sind Fehler im Betrieb nur schwer festzustellen. In den neuesten Versionen wurde das Monitoring durch die Bereitstellung von Metriken zwar deutlich verbessert, ist aber immer noch eine Herausforderung.

Kafka Streams ist eine sehr gelungene Lösung, um Stream Processing in Verbindung mit Kafka zu machen. Im bisherigen Projektverlauf konnten alle Anforderungen an die Verarbeitung der Sensordaten mit Kafka Streams abgebildet werden.

Dr. Ralph Guderlei
ralph.guderlei@exxcellent.de

Dr. Ralph Guderlei ist Technology Advisor und Projektleiter bei der eXXcellent solutions GmbH in Ulm. Neben der Arbeit in unterschiedlichen Kundenprojekten berät er Teams in technologischen und methodischen Fragestellungen. Zurzeit beschäftigt er sich intensiv mit Lösungen im IoT- und Smart-City-Umfeld. Seine Erfahrungen gibt er gerne auf Konferenzen und in Fachartikeln weiter.

Microservices und Makro-Architektur: drei zentrale Entwurfsfragen bei vertikalen Anwendungs-Architekturen

Stefan Zörner, embarc

Moderne Architektur-Stile wie Microservices oder Self-contained Systems lassen Teams, die einzelne Teile entwickeln, viel Freiheit bei Technologie-Entscheidungen. Drei Themen entpuppen sich jedoch regelmäßig als Kandidaten, um übergreifend adressiert zu werden, damit die Anwendung wie aus einem Guss wirkt oder andere Architekturziele nicht verfehlt. Dieser Artikel stellt die Fragestellungen vor und zeigt Antworten auf.

Die großen IT-Trends der letzten Jahrzehnte – seien es Objektorientierung, SOA, Model-Driven Architecture, Cloud oder zuletzt Microservices – versprechen im Grunde nur zwei Dinge: Kosteneffizienz und/oder Flexibilität. Kosteneffizienz heißt Kosten sparen in Entwicklung und Betrieb, erreicht beispielsweise durch effizientere Werkzeuge, Methoden oder (wie bei SOA und OO) Wiederverwendung. Flexibilität bedeutet, schnell auf Veränderungen reagieren zu können, etwa auf neue fachliche Anforderungen, technologische Trends oder Schwankungen in der Last. Microservices versprechen dabei in erster Linie Flexibilität. Wie genau erfüllen Microservices diesen Wunsch? Wie erreicht man Flexibilität durch Anwendung des Architektur-Stils?

Facette von Flexibilität	In Microservices unterstützt durch ...
Neue fachliche Anforderungen schnell umsetzen können	▪ Kleinteiligkeit („Micro") als Gegenmodell zum Monolithen ▪ Lose Kopplung der Services, leichte Austauschbarkeit
Technologische Trends schnell aufgreifen können	▪ Hoher Freiheitsgrad bei Technologieauswahl ▪ Services mit unterschiedlichem Technologie-Stack möglich
Schwankungen in der Last gut auffangen können	▪ Services einzeln skalierbar ▪ Services schnell start- und wegwerfbar

Tabelle 1: Flexibilität erreichen

Ein Spannungsfeld

Microservices verfolgen die Idee, eine einzelne Anwendung in kleine, lose gekoppelte Services zu zerlegen. Für diese Services gelten charakteristische Eigenschaften *[1]*, die positiv auf unterschiedliche Aspekte oder Fähigkeiten von Flexibilität wirken. *Tabelle 1* stellt diese gegenüber.

Wie bei Trends üblich, klingt der Ansatz verlockend. Doch schon in der zitierten Definition *[1]* schlummert ein Spannungsfeld. Denn der möglichen technologischen Vielfalt der Services (unterschiedliche Paradigmen, Programmiersprachen, Bibliotheken etc.) steht die Anforderung nach einer einzelnen Anwendung gegenüber *(siehe Abbildung 1)*. Ein gewisser Grad an Einheitlichkeit ist dafür unabdingbar.

Themen für die Makro-Architektur

Software-Architektur wird gerne als Summe wichtiger Entscheidungen verstanden, die im weiteren Verlauf schwer zurückzunehmen sind. Auch in anderen Kontexten als Microservices sehen wir unterschiedliche Ebenen von Entscheidungen. Eine Systemlandschaft und einzelne Anwendungen zum Beispiel oder eine Produktfamilie und einzelne Ausprägungen oder Varianten sowie eben wie hier eine Anwendung und einzelne Services. Die Frage lautet also: Welche Aspekte sind für alle Teile (Anwendungen, Ausprägungen, Services) des Ganzen (Systemlandschaft, Produktfamilie, Anwendung) gleich und wo haben einzelne Teile (oder besser: die dafür verantwortlichen Teams) Spielraum?

Die erste Ebene bezeichnen wir jeweils als Makro-Architektur. Dinge in der Makro-Architektur einer Microservices-Lösung zu vereinheitlichen, verspricht gewisse Vorteile; Freiheiten zu gewähren ebenso. Letzteres macht den Microservices-Architekturstil mit zu dem, was er ist. In *Tabelle 2* sind häufig genannte Argumente für Standardisierung und Individualisierung aufgezählt.

Die Vielfalt von Themen, bei denen Teams diskutieren können, ob sie diese in der Makro-Architektur für alle Services einheitlich adressieren wollen, ist groß. Sie zerfallen in diese (nicht 100% trennscharfen) Kategorien:

- Interaktion mit Benutzern und Anbindung von Fremdsystemen, also UI- und Integrationsthemen
- Technische Aspekte unter der Haube, wie zum Beispiel Programmiersprache oder Sicherheit

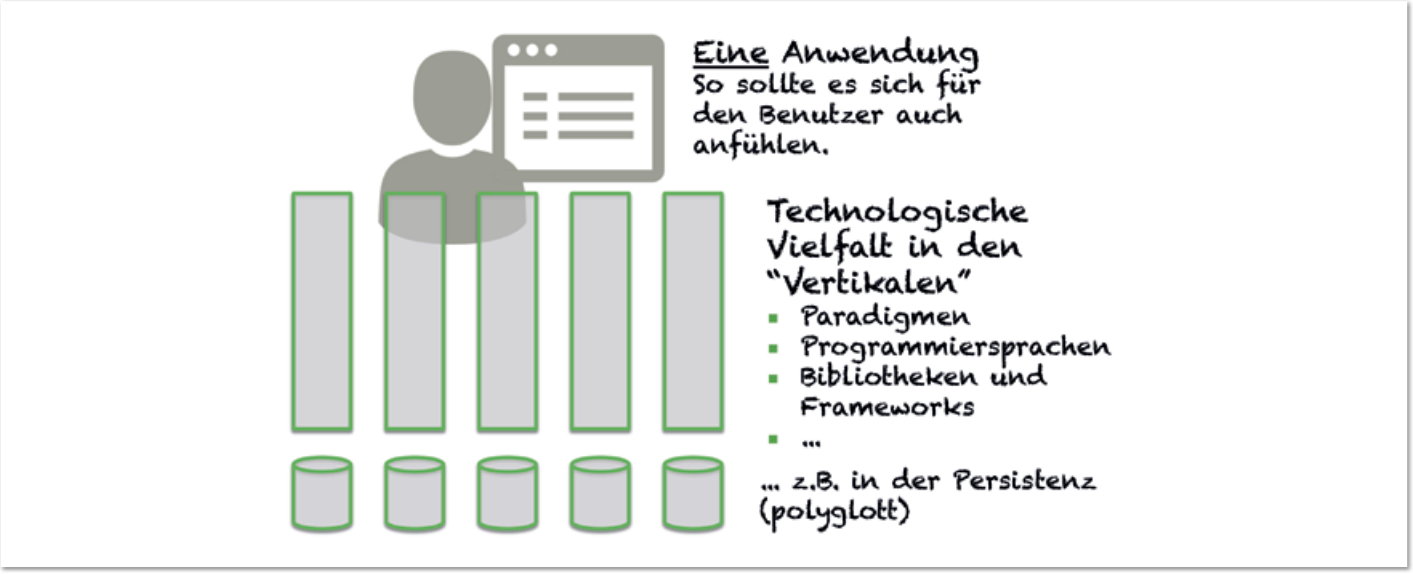


Abbildung 1: Technologische Freiheiten in den Vertikalen vs. Eine Anwendung

Pro Vereinheitlichung	Pro Individualisierung
▪ Entwickler wechseln leicht zwischen Teams und Projekten ▪ Konzentration auf Fachlichkeit leichter möglich ▪ Wiederverwendung technischer Lösungen ▪ Fehlervermeidung in kritischen Bereichen durch erprobte Konzepte	▪ Einsatz optimaler Lösungen für spezifische Probleme möglich ▪ Neue Trends lassen sich schneller aufgreifen ▪ Fehlentscheidungen haben geringere Relevanz ▪ Geringere Abhängigkeit von einzelnen Lieferanten

Tabelle 2: Häufig genannte Vorteile

Abbildung 2: Ein bunter Kandidatenzoo für übergreifende Themen

- Entwicklung und Weiterentwicklung, also alles rund um das Vorgehen und die Vorgehensweise (etwa Entwerfen, Quelltext verwalten, Testen, Bauen etc.)
- Zielumgebung und Betriebsaspekte, hierzu zählen Infrastruktur und Middleware, Monitoring, Disaster-Recovery etc.

Abbildung 2 zeigt eine Tag-Cloud mit vielen konkreten Themen zur Illustration. Sie sind nach den vier aufgezählten Kategorien farblich kodiert.

Bei welchen dieser zahllosen Themen sollten Teams vereinheitlichen? Wo sollten sie Spielraum haben, um spezifische Lösungen und Innovation zu ermöglichen? Vielleicht reichen für bestimmte Themen auch Vorschläge, die den Start erleichtern, sich mit der Zeit entwickeln und durch neue ersetzt werden.

Im Einzelnen hängt das vom Vorhaben und vom Kontext ab. Gleichzeitig gibt es zu bestimmten Themen nach Erfahrung des Autors sehr regelmäßig Diskussionen darüber, „ob und, wenn ja, wie wir das zentral machen …“. Er hat drei besonders häufige Fragestellungen herausgepickt, um sie im Folgenden zu diskutieren.

Thema 1: Die UI-Frage

Eine häufige Anforderung an Microservices-Lösungen: Trotz mehrerer Teile soll sich die Anwendung dem Benutzer „wie aus einem

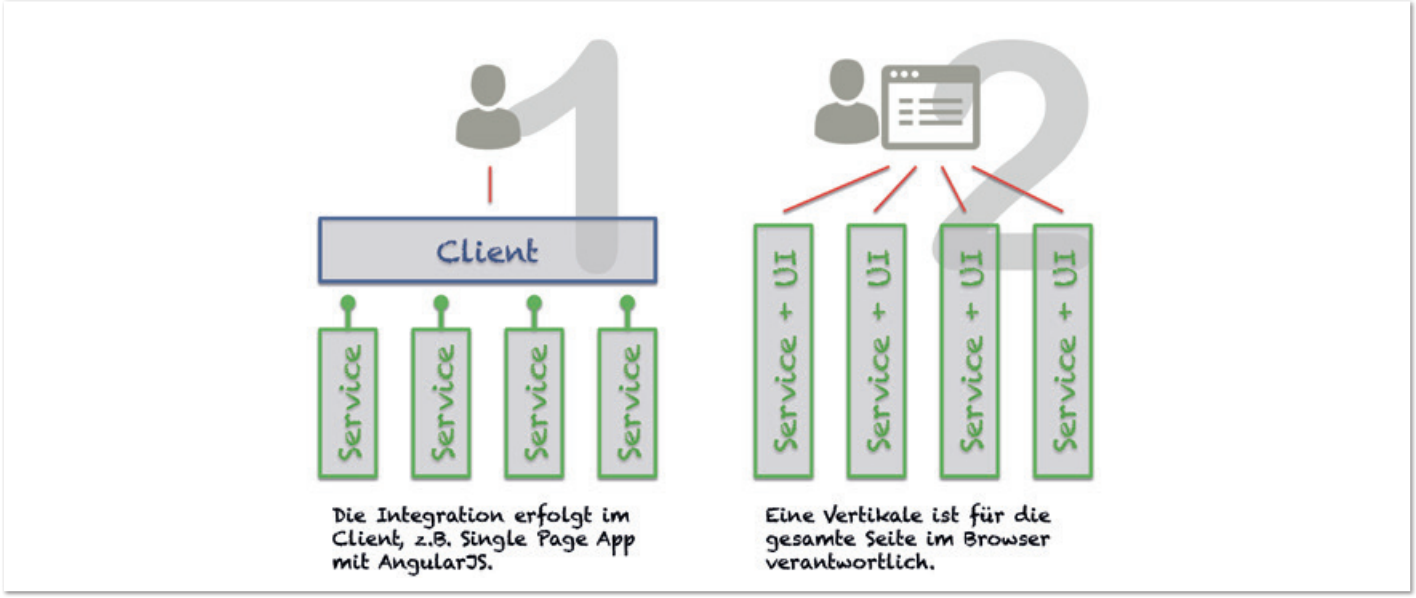


Abbildung 3: Zwei extreme Lösungsoptionen für die UI-Frage

Option 1: Gemeinsames UI für UI-lose Vertikalen	Option 2: Separate UIs für die Vertikalen
▪ Inhalte und Funktionen aus verschiedenen Themen lassen sich nahtlos integrieren – keine Brüche ▪ Einheitliches User Interface leicht erreichbar ▪ Spezialisiertes Team für optimale UX denkbar	▪ Teams vollumfänglich für Thema verantwortlich (Cross-funktionale Teams) ▪ Technologische Freiheiten ▪ Unabhängiges Arbeiten leicht möglich

Tabelle 3: Jeweilige Stärken der beiden UI-„Extrem"-Optionen

Guss" präsentieren. Die Frage: Wie realisieren wir mit mehreren Teilen (und Teams) ein UI? Darauf gibt es zwei extreme Antworten (und viele dazwischen).

Die erste Extremposition: Die einzelnen Microservices haben keinen UI-Anteil, sondern nur eine (REST-)Schnittstelle. Ein gemeinsamer Client greift auf alle Microservices zu. *Abbildung 3* zeigt links die Idee schematisch.

Die zweite Extremposition: Vertikalen (z.B. Microservices) bringen jeweils die komplette UI für „ihr Thema" mit. Die Integration erfolgt beispielsweise über Links im Browser. *Abbildung 3* zeigt diese Idee rechts.

.

Beide Positionen haben ihre Berechtigung und sind nicht nur akademischer Natur. Netflix hat beispielsweise die erste Option gewählt und entwickelt gleich mehrere übergreifende Clients in jeweils auf die entsprechende UI-Technologie spezialisierten Teams. Xing hingegen verfolgt bei seiner Webseite einen Ansatz nach Variante zwei.

Die unterschiedlichen Antworten auf die gleiche Frage verwundern nicht: Beide Optionen haben unbestreitbare Stärken, skizziert in *Tabelle 3*. Deswegen sind Lösungen wie so oft in der Software-Architektur ein Ausbalancieren und Kompromissefinden. Die perfekte User Experience lässt sich typischerweise mit der ersten Option (gemeinsames UI) erreichen. Der Wunsch, Teams unabhängig entwickeln und releasen zu lassen, führt zu Lösungsansätzen in Richtung der zweiten Option.

Thema 2: Kommunikation

Eine weitere häufige Anforderung an Microservices-Lösungen: Ein Service benötigt Funktionalität und/oder Daten eines anderen Service, um seine Aufgabe zu erledigen. Die Frage: Dürfen Services miteinander reden, und wenn ja, wie? Wenn nein, wie realisieren wir dann obige Anforderung?

Die Relevanz für Microservices-Architekturen liegt im Wunsch nach loser Kopplung, etwa um Teile leicht austauschen zu können, und auch, um Teams möglichst unabhängig voneinander arbeiten zu lassen. Technisch gesehen zerfällt die Frage streng genommen in zwei detailliertere Entscheidungen: direkte vs. indirekte Kommunikation sowie synchrone vs. asynchrone.

Bei synchroner Kommunikation wartet ein Client (etwa ein Service) auf die Antwort des genutzten Service und blockiert. Bei asynchroner Kommunikation wartet der Client nicht auf eine Antwort. Er erhält diese falls nötig später, gegebenenfalls über einen anderen Kanal.

Bei direkter Kommunikation kennt der Client den genutzten Service und spricht ihn direkt an. Im indirekten Fall kommuniziert der Client über eine Middleware, die ihn vom angesprochenen Service entkoppelt. Client und Server kennen sich also nicht. In der Makro-Architektur einer Microservices-Lösung ist zu klären, welche Kommunikationsmuster in welchen Situationen zulässig sind und wie sie umzusetzen sind.

Die engste Kopplung entsteht durch direkte, synchrone Kommunikation. Hier nutzt ein Service direkt Funktionalität von einem anderen Service und blockiert während des Aufrufs. Für die Implementierung ergeben sich zwei Herausforderungen. Zum einen ist das Auffinden („Service Discovery") zu klären. Die klassische Netflix-Architektur beispielsweise sieht hier Eureka als Service Registry vor. Auch moderne Plattformen zur Container-Orchestrierung (wie Kubernetes) bieten direkt Lösungen hierzu an.

Die zweite Herausforderung: Was, wenn ein Service fehlerhaft ist, gar nicht oder langsam antwortet? Typische Lösungen sind Resilience-Muster wie Circuit Breaker, bei Netflix etwa implementiert durch Hystrix. Eine gute Quelle für Lösungen rund um diese Herausforderungen bietet Chris Richardson auf seiner Webseite *[2]* und in seinem Buch „Microservices Patterns" an. Service Registry und Circuit Breaker sind Beispiele für Muster dort – die Diskussion erfolgt Technologie-neutral und nennt am Ende Beispiel-Implementierungen für verschiedene Plattformen wie Java.

Da die direkte, synchrone Kommunikation zu einer engen Kopplung führt, erfreuen sich andere Konstellationen großer Beliebtheit. Hier kommen dann Messaging und Events zum Einsatz. Die beteiligten Kommunikationspartner akzeptieren mitunter „eventual consistency" als Preis für geringere Abhängigkeiten.

Thema 3: Security

Kevin Hoffman führt in „Beyond the Twelve Factor App" *[3]* aus: „Security sollte niemals ein nachträglicher Einfall bei Deiner Anwendungsentwicklung sein." Die Frage ist: Welche Themen rund um Security adressieren wir in der Makro-Architektur? (und wie?) Tatsächlich ist Security ein weites Feld. *Abbildung 4* zeigt häufige Aufgaben, denen sich Teams im Microservices-Umfeld stellen.

Im Einzelnen geht es um folgende Themen:

- Die Benutzer der Anwendung müssen sich anmelden (Authentifizierung), deren Berechtigung muss überprüft werden (Autorisierung). Der Zugriff vom Browser erfolgt in der Regel verschlüsselt.
- Stellt die Anwendung ein API bereit, ergibt sich hier für die Clients die gleiche Aufgabenstellung wie für (menschliche) Benutzer. Die Lösung kann gleich aussehen, muss aber nicht.
- Falls Services mit anderen Services kommunizieren (siehe Thema 2), ergeben sich auch hier Fragen der Authentifizierung, Autorisierung, Verschlüsselung.
- Schließlich greifen Services mitunter auf Datenbanken oder generell Persistenz-Lösungen zu oder nutzen andere Fremdsysteme, in denen Berechtigungen überprüft werden. Wo und wie speichert man beispielsweise nötige Zugangsdaten (Zertifikate, Kennwörter etc.)?

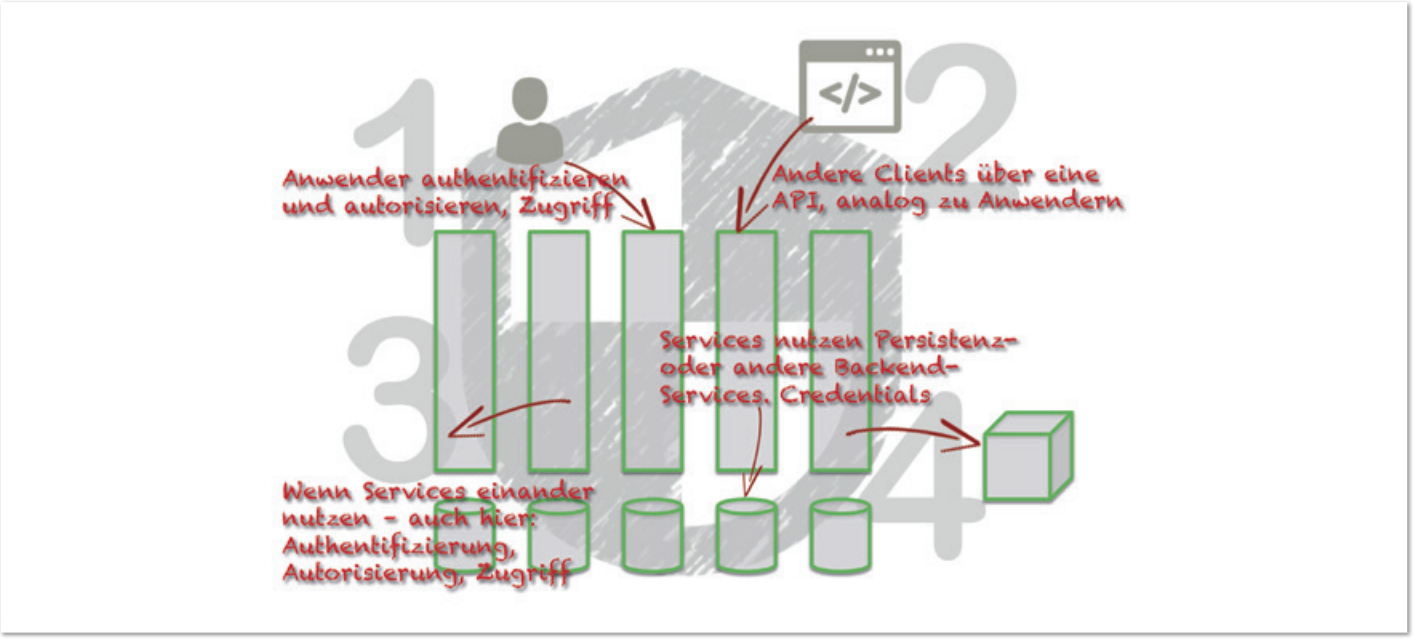


Abbildung 4: Zentrale Aufgaben rund um Security bei Microservices

Alle Punkte mit allen Optionen auszudiskutieren, sprengt hier den Rahmen. Zumindest eine häufige Antwort im ersten und zweiten Themenfeld soll allerdings kurz vorgestellt werden, da sie Verantwortlichkeiten zwischen Makro- und Mikro-Architektur interessant ausbalanciert.

Die Authentifizierung erfolgt dort zentral in der Makro-Architektur durch einen gemeinsamen Service. Single Sign-on ist erforderlich, damit die Anwendung wie aus einem Guss erscheint. Die einzelnen Services erhalten ein Authentifizierungs-Token, etwa via JSON Web Token (JWT). Die Autorisierung, also die Überprüfung der Berechtigungen, obliegt dann jedoch den einzelnen Services.

Ein Service erhält lediglich die Identität des Aufrufers und gegebenenfalls weitere Eigenschaften (wie globale Rollen). Hintergrund dieser Strategie ist zum einen der Wunsch, dass Teams neue Berechtigungen in ihren Services einführen und überprüfen können, ohne eine zentrale Komponente dafür anpassen zu müssen. Diese obliegen oftmals einem anderen Team und man könnte nicht mehr eigenständig liefern. Weiterhin sind Berechtigungen oftmals fachlich getrieben, die einzelnen Services (Vertikalen) fachlich geschnitten und somit der natürliche Ort dafür.

Weitere Informationen und Fazit

Ein Architektur-Stil, der klarere Präferenzen für die genannten Themen parat hat, sind die Self-contained Systems (SCS) *[4]*. Vertikalen (dort: Systeme) haben in SCS stets eine Web-UI. Kommunikation hat wo immer möglich asynchron zu erfolgen. Geteilte Infrastruktur sollte wo immer möglich vermieden werden. Ein zentraler Authentifizierungsdienst ist hier vielleicht eine der zulässigen Ausnahmen. Die ISA-Principles *[5]* bündeln Erfahrungswissen zu Microservices und SCS in insgesamt neun Prinzipien. Eines davon (das dritte) fordert die explizite Bearbeitung von Makro- und Mikro-Architektur.

Zusammenfassend ist ein gutes Verständnis von Makro- und Mikro-Architektur zentral für den Einsatz moderner Architektur-Stile. UI-, Kommunikations-, und Security-Themen sind häufige Brennpunkte für eine frühe, initiale Bearbeitung in der Makro-Architektur. Entscheidungen dort bergen oftmals Kompromisse, die entlang der Qualitätsziele ausbalanciert gehören. Dazu müssen diese natürlich für das Vorhaben bekannt sein.

Links und Literatur

[1] Martin Fowler, James Lewis, Microservices: a definition of this new architectural term: *https://martinfowler.com/articles/microservices.html*

[2] Chris Richardson: *http://microservices.io/patterns*

[3] Kevin Hoffman, Beyond the Twelve Factor App, O'Reilly Media 2016

[4] Self-contained Systems (SCS): *http://scs-architecture.org*

[5] Independent Systems Architecture: *https://isa-principles.org*

[6] Architektur-Spicker zu Microservices, Kurzreferenz: *https://www.embarc.de/architektur-spicker*

Stefan Zörner
stefan.zoerner@embarc.de

Stefan Zörner unterstützt Kunden von embarc in Architektur- und Umsetzungsfragen mit dem Ziel, gute Architektur-Ansätze wirksam in der Implementierung zu verankern. Sein Wissen und seine Erfahrung teilt er regelmäßig in Vorträgen, Artikeln und Workshops. Stefan ist Apache Committer, aktives Board-Mitglied im iSAQB und Autor des Buchs „Software-Architekturen dokumentieren und kommunizieren".


Lorem ipsum dolor sit amet cns ect
etuer leo interdum pretium elit In.
Et eros a pellentesque volutpat
fermentum odio eleifend.
Ein Bild
sagt mehr als
tausend
Worte
Daniel Rosowski

Visualisierung ist ein wichtiges Werkzeug für jeden Software-Entwickler, um komplexe Sachverhalte für sich und seine Kollegen zu veranschaulichen. Viele von uns kennen leider nur die allgemein üblichen WYSIWYG-Tools, bei denen Handarbeit gefragt ist. Aber gerade wenn es darum geht, Datenstrukturen aus laufenden Programmen zu visualisieren, ist es hilfreich, wenn man sein Diagramm beschreiben kann, anstatt es selber zu zeichnen. Das Layout wird daraufhin algorithmisch ermittelt und der Graph gerendert. Dieser Artikel stellt DOT vor, eine Beschreibungssprache für Graphen, die genau für diesen Anwendungsfall konzipiert wurde und die in keinem Werkzeugkasten eines Entwicklers fehlen sollte.

In der Informatik liegen die meisten Informationen in strukturierter Form vor und können in der Regel als Graphen dargestellt werden. Häufig sind beispielsweise Computer-Netzwerke in einer Netzwerk-Topologie dargestellt oder die verschiedenen Arbeitsschritte eines Algorithmus werden in einem Flussdiagramm veranschaulicht. Selbst unsere heißgeliebten UML-Diagramme sind nichts anderes als gerichtete Graphen. Kurz gesagt, Graphen eignen sich hervorragend, um komplexe Sachverhalte zu visualisieren, da es den meisten Menschen leicht fällt, den Sinn hinter einer Handvoll miteinander verbundenen Kästchen zu erkennen.

Es gibt viele gute Werkzeuge, um Graphen selbst zu zeichnen, etwa Dia, xfig, Inkscape oder Skencil, um nur einige freie Varianten zu nennen. Wenn es sich um umfangreiche Graphen handelt oder um Graphen, die sich häufig ändern, wäre es gut, wenn man die Graphen beschreiben könnte und das Layout automatisch erzeugt würde. Hier kommt die Sprache DOT ins Spiel.

DOT wurde bereits gegen Ende der 1990er Jahre von AT&T Labs entwickelt und umfasst eine vollständige Grammatik, um gerichtete und ungerichtete Graphen (dazu später mehr) zu beschreiben. Ganz im Sinne der Unix-Philosophie („do one thing and do it well", „handle text streams, because that is a universal interface" *[1]*) erledigt DOT genau eine Sache, nämlich das Layout eines Graphen, der in einem Textformat beschrieben ist. Das hat den Vorteil, dass diese Beschreibungen ganz einfach und ohne Zuhilfenahme eines Tools erstellt und (wenn nötig) auch verarbeitet werden können. Die freie Software-Suite Graphviz interpretiert das DOT-Format und bietet verschiedene Algorithmen für das Layouten der Graphen an.

Eins nach dem anderen: Wer Graphen in DOT beschreiben möchte, sollte sich erst einmal mit den Grundlagen der Graphen-Theorie vertraut machen. Aber keine Sorge, es folgt keine seitenlange theoretische Abhandlung. Tatsächlich reicht eine Handvoll grundlegende Konzepte, um DOT bereits sinnvoll einsetzen zu können.

Etwas Theorie

Ein Graph besteht aus einer Menge von Knoten und Kanten. Wenn ein Knoten auf einen anderen Knoten (oder auch auf sich selbst) verweist, nennt man die Verbindung zwischen den beiden Knoten eine Kante. Es wird zwischen ungerichteten und gerichteten Graphen unterschieden. Bei einem ungerichteten Graphen ist die Richtung der Kante ohne Bedeutung. „A – B" ist genau das Gleiche wie „B – A" *(siehe Abbildung 1)*. Anders verhält es sich bei dem gerichteten Graphen. A → B ist ein anderer Graph als B → A, da die Richtung entscheidend ist *(siehe Abbildung 2)*.

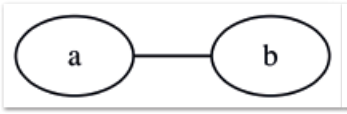


Abbildung 1: Ungerichteter Graph

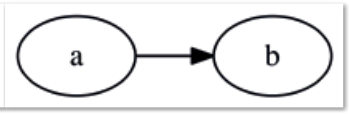


Abbildung 2: Gerichteter Graph

Ein Baum ist ein spezieller azyklischer Graph, der von einem Ursprungsknoten (Wurzel) ausgeht und 1 bis n Kindsknoten besitzt. Jeder Kindsknoten besitzt dann wiederum 1 bis n Kindsknoten etc., bis die Kindsknoten am Ende keine weiteren Knoten referenzieren. Diese Kindsknoten am Ende des Baums werden auch „Blätter" genannt *(siehe Abbildung 3)*.

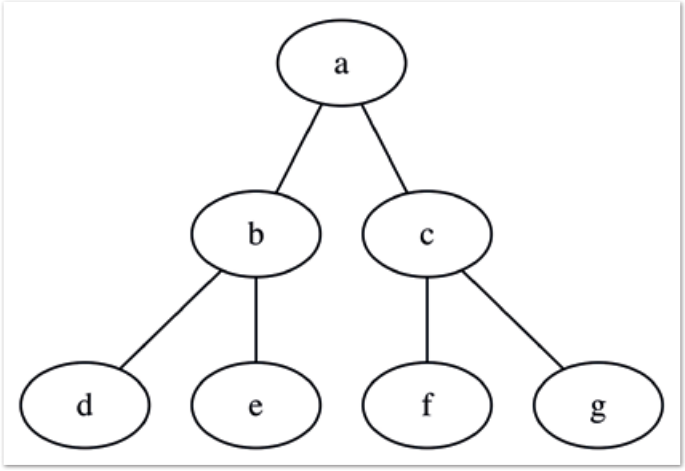


Abbildung 3: Baum

Ein Netzwerk ist ein gerichteter Graph mit gewichteten Kanten. Anders als bei einem Baum dürfen in einem Netzwerk Zyklen auftreten. Ein Zyklus ist eine kreisförmige Beziehung zwischen verschiedenen Knoten *(siehe Abbildung 4)*.

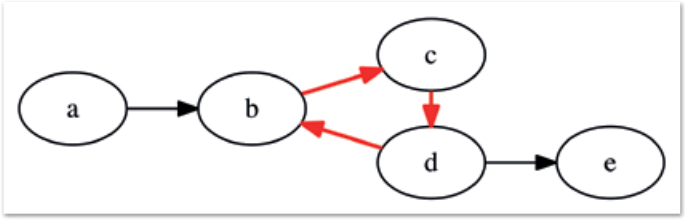


Abbildung 4: Netzwerk

Die Syntax

Damit liegt die Theorie vor, um erste Graphen zu beschreiben. Als Nächstes folgt die Syntax oder Grammatik der DOT-Sprache. Der einfachste anzunehmende Graph besteht aus zwei Knoten sowie einer verbindenden Kante und ist ungerichtet *(siehe Listing 1)*. Tipp:

Um die angegebenen Beispiele nachzuvollziehen, kann viz-js.com verwendet werden, um die Graphen live im Browser zu rendern.

Knoten werden mit einem Schlüssel deklariert, der dann auch bei der Definition einer Kante verwendet wird. Die Deklaration der Knoten ist optional, wenn dem Knoten keine Attribute zugewiesen sind (mehr dazu später). Um einen gerichteten Graphen zu beschreiben, verwendet man das Schlüsselwort „digraph" und einen Pfeil anstatt der beiden Gedankenstriche *(siehe Listing 2)*.

Formen und Farben

DOT bietet eine Vielzahl von Attributen, um das Aussehen der Knoten und Kanten anzupassen. Ein sehr gebräuchliches Attribut ist die Beschriftung eines Knotens. Im obigen Beispiel wird der Knoten „a" auch mit der Beschriftung „a" angezeigt, da keine andere Beschriftung angegeben wurde. Dem Knoten lassen sich bei der Deklaration verschiedene Attribute zuweisen, so auch eine Beschriftung. Diese Zuweisung folgt immer dem Schema „knoten [attribute1=wert1, attribut2=wert2]". In *Listing 3* wird dem Knoten „a" eine etwas längere Beschriftung zugewiesen, die auch Zeilenumbrüche enthalten darf. Referenziert wird der Knoten weiterhin über den Schlüssel, in diesem Fall „a".

Neben der Beschriftung lassen sich auch Font sowie Schriftgröße und -farbe der Beschriftung definieren oder dem Knoten eine andere Form geben. Eine Liste aller möglichen Attribute steht in der Dokumentation von Graphviz *[2]*. Natürlich gibt es neben der Beschriftung auch eine Fülle an Darstellungsoptionen für die Kanten. So kann man beispielsweise die Farbe oder Dicke einer Kante festlegen, um diese im Graphen hervorzuheben *(siehe Listing 4)*. Damit könnte man zum Beispiel den Pfad in einem Graphen markieren. Das Beispiel ließe sich auch in einer vereinfachten Notation beschreiben, in der man die Kanten einfach hintereinander definiert *(siehe Listing 5)*.

Wie bereits erwähnt, ist die Gewichtung der Kanten bei einem Netzwerk zwingende Voraussetzung. In DOT lässt sich mit dem Schlüsselwort „weight" das Gewicht der Kante festlegen. Die Gewichtung findet beim Layout des Graphen Beachtung und beeinflusst die Länge der Kante. Der Algorithmus zielt darauf ab, dass die Kanten mit der höchsten Gewichtung den kürzesten Weg haben.

Die Entwickler der DOT-Sprache haben selbstverständlich auch daran gedacht, eine Formatierung auf viele Knoten anzuwenden. Angenommen, man möchte die Form und die Füllfarbe der Knoten verändern. Anstatt bei jedem einzelnen Knoten die Formatierung zu ändern, lässt sich mit „node [attribut1=wert1, attribut2=wert2, ...]" die Formatierung sämtlicher Knoten im Graphen anpassen. Analog kann man auch die Formatierung sämtlicher Kanten mit „edge [attribut=wert]" anpassen *(siehe Listing 6)*. Die globalen Werte werden vererbt und lassen sich zu einem späteren Zeitpunkt erweitern *(siehe Listing 7)*.

Records

Der Record, eine spezielle Knotenform, soll nicht unerwähnt bleiben. Records dienen dazu, die Beschriftung eines Knotens in tabellarischer Form darzustellen. Das Schlüsselwort „shape=record" definiert einen Knoten als Record. Das Label eines Records folgt einem bestimmten Format. So werden die verschiedenen Spalten mit einem Pipe-Zeichen („|") voneinander getrennt. Innerhalb einer Spalte kann zwischen der vertikalen und der horizontalen Ausrichtung mit geschweiften Klammern gewechselt werden. Mit einem gerichteten

```
graph { ①
a;   ②
b;
a -- b ③
}
```

Listing 1: ① Definition eines ungerichteten Graphen, ② Deklaration der Knoten und ③ Definition einer Kante zwischen a und b.

```
digraph {
a -> b
}
```

Listing 2

```
graph {
a [label="Eine etwas längere\n Beschriftung"]; a -- b;
}
```

Listing 3

```
graph {
a -- b [penwidth=2, color="red"];
b -- c [penwidth=2, color="red"];
c -- d [penwidth=2, color="red"];
d -- e [penwidth=2, color="red"];
c -- a;
d -- a;
e -- b;
}
```

Listing 4

```
graph {
a -- b -- c -- d -- e [penwidth=2, color="red"];
c -- a;
d -- a;
e -- b;
}
```

Listing 5

```
graph {
node [shape=box, style=filled, fillcolor=lightblue];
edge [penwidth=2];
a -- b;
}
```

Listing 6

Graphen und Records lassen sich beispielsweise einfache Klassen-Diagramme erstellen *(siehe Listing 8)*.

HTML

Ein weiteres Gimmick von DOT ist die Formatierung von Knoten mit HTML-Elementen. Damit lassen sich beispielsweise komplexere Strukturen wie Tabellen in den Knoten darstellen *(siehe Listing 9)*. Der Autor hat sich dies erfolgreich zunutze gemacht, um Freemarker-Templates für verschiedene Knotenarten zu hinterlegen und diese dann entsprechend rendern zu lassen.

Es ist allerdings zu erwähnen, dass es sich hierbei nur um eine Teilmenge von HTML handelt, die Graphviz für die Formatierung der Knoten un-

```
graph {
node [shape=box]  ①
html;
node [style=filled]  ②
head;
body;
node [fillcolor=lightblue] ③
title;
meta;
h1;
p;
span;
html -- head;
html -- body;
head -- title;
head -- meta;
body -- h1;
body -- p;
body -- span;
}
```

Listing 7: ① Zuerst wird „shape=box" für alle Knoten definiert, ② zusätzlich zu „shape=box" gilt für alle Knoten, die nach dem Knoten „html" definiert werden, das Attribut „style=filled". ③ Die Blätter unseres HTML-Baums würden wir gerne mit fillcolor=lightblue versehen. Dieses und alle bisher genannten Attribute gelten dann für alle Knoten nach „head" und „body".

```
digraph {
rankdir="RL";
Fahrzeug [shape="box"]
Auto [
shape="record",
label="Auto|+ velocity : int\l|+ accelerate() : void\l"
];
Fahrrad[
shape="record"
label="Fahrrad|+ velocity : int\l|+ accelerate() : void\l"
]
Auto -> Fahrzeug [arrowhead="empty"]
Fahrrad -> Fahrzeug [arrowhead="empty"]
}
```

Listing 8

```
graph {
a [label=<  ①
<table>
<tr><td>Das</td><td>ist</td></tr>
<tr><td>ziemlich</td><td>cool!</td></tr>
</table>
>]   ②
a -- b;
a -- c;
}
```

Listing 9: ① Ein HTML-Label wird mit einer spitzen Klammer initiiert und eine spitze Klammer am Ende beendet den HTML-Code ②.

```
graph {
a -- {b c};
}
```

Listing 10

terstützt. Eine vollständige Beschreibung darüber, welche HTML-Tags unterstützt werden, steht in der Dokumentation *[3]*.

Neben der Formatierung lassen sich Knoten und Kanten auch mit Links versehen, die – ein unterstützendes Ausgabeformat (wie PostScript

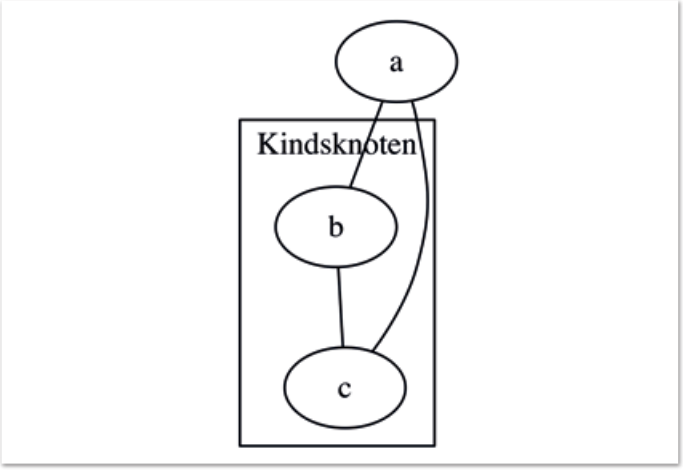


Abbildung 5: Das Beispiel aus Listing 11

oder SVG) vorausgesetzt – auch angeklickt werden können. Wie wir später sehen werden, ist dieses Feature besonders hilfreich, wenn man eine JavaScript-Bibliothek für das Rendering der Graphen verwendet.

Subgraphen

Ein weiteres wichtiges Feature ist die Unterstützung der Subgraphen. Sie erfüllen drei Funktionen in DOT: Zunächst dienen sie der semantischen Gruppierung von Knoten und Kanten, die in einem logischen Zusammenhang stehen. Ein Subgraph wird mit geschweiften Klammern initiiert und kann nebenbei auch dazu verwendet werden, um Kanten zusammenzufassen. *Listing 10* zeigt den anonymen Subgraphen, bestehend aus „b" und „c". Durch die verkürzte Schreibweise lässt sich für „a" jeweils eine Kante zu „b" und „c" anlegen.

Darüber hinaus dienen Subgraphen auch als Kontext für Attribute. So lassen sich etwa alle Knoten eines Subgraphen mittels „node [shape=box]" als Kasten darstellen. In dieser Funktion treffen wir den Subgraphen später noch an, wenn es um das Thema „Layout" geht. Subgraphen, die mit dem Schlüsselwort „subgraph" und dem Präfix „cluster" beginnen, werden mit einem Kasten umrandet dargestellt *(siehe Abbildung 5)*. Außerdem lassen sich diese speziellen Subgraphen mit dem Attribut „label" beschriften. Die *Abbildung 5* wird beispielsweise über eine Beschreibung wie in Listing11 erzeugt.

Layout

Um das Layout für Graphen automatisch zu erzeugen, führt Graphviz das Konzept der Ränge („ranks") ein. Jeder Knoten erhält einen höheren Rang als der höchstrangige Knoten, der auf diesen Knoten zeigt. Bisher wurden alle Knoten von oben nach unten ausgerichtet; die Standard-Ausrichtung, die Graphviz verwendet, wenn nichts explizit angegeben wurde. Je tiefer der Knoten, desto höher ist dessen Rang bei dieser Ausrichtung. Mit dem Schlüsselwort „rankdir" lässt sich der Graph beispielsweise von links nach rechts ausrichten. Der Rang der Knoten erhöht sich hier also horizontal von links nach rechts. Das Prinzip lässt sich am besten anhand eines Beispiels veranschaulichen *(siehe Listing 12)*.

Einer der grundlegenden Mechanismen, um auf das Layout Einfluss zu nehmen, besteht in der Manipulation der Rangfolge. Angenommen, man möchte bei dem obigen Beispiel die Knoten „B" und „C" auf einer Ebene darstellen. Dazu bietet uns Graphviz die Möglichkeit, den Knoten in einem Subgraphen mit dem Schlüsselwort „rank=same" in einem Rang zu gruppieren *(siehe Listing 13)*.

Die Kehrseite der Medaille eines algorithmisch generierten Layouts ist allerdings, dass das mit Graphiz erstellte Layout trotz aller Tricks und Kniffe von den Vorstellungen abweichen kann. Neben den Einstellungen zum Layout könnte auch ein anderer Layout-Algorithmus Abhilfe schaffen. Graphviz bietet eine Handvoll Layout-Algorithmen (oder „Engines"), die jeweils für eine bestimmte Art von Graphen optimiert sind. Der am häufigsten verwendete Algorithmus ist „dot", der sich gut für gerichtete, azyklische Graphen eignet.

Es gibt auch Algorithmen für ungerichtete Graphen, die die Knoten beispielsweise konzentrisch („circo") oder spiralförmig („twopi") anordnen. Da der Rahmen dieses Artikels mit der genauen Betrachtung jedes einzelnen Algorithmus gesprengt würde, sei hier auf die Mainpage von Graphviz („man dot") verwiesen. Wenn immer noch Probleme mit dem Layout bestehen, sollte man sich fragen, ob DOT wirklich das richtige Tool für diesen Graphen ist. Es ist keine Schande, auch mal einen Graphen von Hand zu zeichnen.

Rendering

Eine der Stärken der DOT-Sprache ist das einfache Textformat, um die Graphen zu beschreiben. Durch die Trennung von Beschreibung und Darstellung lässt sich ein Graph ganz einfach im Programmfluss generieren. Letztendlich handelt es sich um einen String, der in einem bestimmten Format erzeugt wird und hinterher in den Logs oder in einer Datenbank-Tabelle abgelegt werden kann. Zu einem späteren Zeitpunkt kann man den Graphen dann rendern lassen, um sich beispielsweise eine komplexe Datenstruktur zum jeweiligen Zeitpunkt anzusehen.

Neben diesem manuellen Prozess lässt sich DOT mithilfe verschiedener JavaScript-Bibliotheken eingebettet in eine Web-Anwendung zur Laufzeit rendern. Wenn ein Graph beschrieben und als Textdatei gespeichert ist, kann man mit dem „dot"-Befehl verschiedene Ausgabe-Formate daraus erzeugen wie:

- PostScript (und daraus dann PDF)
- SVG (skalierbare Vektorgrafik für das Web)
- Xfig-Format
- PNG
- GIF
- server- und clientseitige Imagemaps

Mit „dot -Tpng -odot.png dot.txt" wird aus der Beschreibung unseres Graphen in „dot.txt" eine Bilddatei im PNG-Format erzeugt.

viz.js

Man ist mit DOT nicht auf die Kommandozeile beschränkt. Viz.js *[4]* bietet eine komplette Graphviz-Umgebung im Browser. Bei dieser JavaScript-Bibliothek handelt es sich tatsächlich um ein lauffähiges Graphviz, das mit „Emscripten" in „Webassembly" crosskompiliert wurde. Was auf den ersten Blick ziemlich wild aussieht, ist überraschend einfach in der Benutzung.

Viz.js ist in zwei verschiedene Dateien aufgeteilt. Zum einen findet man das eigentliche API in der „viz.js"-Datei. Außerdem gibt es noch die beiden Dateien „full.render.js" und „lite.render.js". Die „full"-Rendering-Datei beinhaltet neben der Möglichkeit der HTML-Labels auch einige der weniger gebräuchlichen Rendering-Engines, auf die zuvor eingegangen wurde. Viz.js kann auf zwei Arten eingebunden werden: zum einen als Web Worker, bei dem der Browser das Rendering des Graphen im Hintergrund erledigt *(siehe Listing 14)*. Oder die Dateien werden einfach mit dem „<script>"-Tag eingebunden *(siehe Listing 15)*. Aber hier ist Vorsicht geboten, denn die Dateien für das Rendering sind recht groß (größer 1 MB).

```
graph {
subgraph cluster_0 {
label="Kindsknoten";
b -- c;
}
a -- {b c};
}
```

Listing 11

```
digraph {
rankdir="LR";
A -> B;          ①
A -> C;          ②
B -> C;
C -> D;          ③
}
```

Listing 12: ① „A" ist der Wurzelknoten, also hat „A" den Rang 1; „A" zeigt außerdem auf „B", damit erhält „B" den Rang 2. ② Sowohl „A" als auch „B" zeigen auf „C", also erhält „C" einen höheren Rang als „B". ③ „C" zeigt auf „D", deshalb erhält „D" einen höheren Rang als „C".

```
digraph {
rankdir="LR";
A -> B -> C -> D;
A -> C;
{rank=same B C};
}
```

Listing 13

```
const workerURL = 'path/to/full.render.js';
let viz = new Viz({ workerURL });
```

Listing 14

```
<script src="viz.js"></script>
<script src="full.render.js"></script>
```

Listing 15

Das Viz.js-API bietet verschiedene Funktionen, um die Graphen in unterschiedlichen Ausgabeformaten zu rendern. Ein sehr gebräuchliches Format, das von den meisten modernen Browsern unterstützt wird, ist das Vektorgrafik-Format SVG. Neben den offensichtlichen Vorteilen von Vektorgrafiken, etwa dass diese ohne Qualitätsverlust skaliert werden können, ist es möglich, Knoten und Kanten mit HTML-Links zu versehen. Gerade bei einer Web-Anwendung bietet das die Möglichkeit für den Benutzer, mit dem Graphen zu interagieren.

Die Funktion „renderSVGElement" liefert ein „Promise"-Objekt zurück, worauf mit „then" im Erfolgsfall und „catch" im Fehlerfall reagiert werden kann *(siehe Listing 16)*. Die Variable „element" beinhaltet das ge-

```
var viz = new Viz(); // or using webworker new Viz({ workerURL })
viz.renderSVGElement('digraph { a -> b }')
.then(function(element) {
document.body.appendChild(element);
})
.catch(error => {
Create a new Viz instance (@see Caveats page for more info) viz = new Viz();
Possibly display the error
console.error(error);
});
```

Listing 16

renderte SVG, das im Erfolgsfall etwa in ein „div" gepackt werden kann.

Tipp: Da es sich bei dem Ausgabeformat um reines SVG handelt, können auch weitere Bibliotheken eingebunden werden, um beispielsweise ein Zoom einzubauen *[5]*. Für eine komplette Demo-Anwendung auf Basis von Spring Boot und viz.js empfiehlt sich ein Blick auf „webviz" *[6]*.

D3

Bei d3-graphviz *[7]* handelt es sich um eine Erweiterung der populären JavaScript-Bibliothek D3, die für die Visualisierung von Daten für das Web entwickelt wurde. Die Erweiterung ist mithilfe von viz.js ebenfalls in der Lage, Graphen im DOT-Format zu rendern. Neben dem reinen Rendering kann d3-graphviz allerdings auch Übergänge im Graphen als Animationen darstellen. Das Prinzip ist recht einfach. Es werden zwei Graphen gerendert und D3 ist anhand der resultierenden SVG-Grafik in der Lage zu ermitteln, welche Knoten und Kanten hinzugekommen sind oder entfernt wurden. Die Veränderung lässt sich daraufhin als Animation darstellen. Außerdem bietet D3 die Möglichkeit, den Graphen on-the-fly zu verändern. Es können Attribute angepasst, neue Knoten oder Kanten hinzugefügt oder vorhandene gelöscht werden. Damit hat der Benutzer die Möglichkeit, mit dem Graphen zu interagieren und ganz neue Use Cases abzubilden.

Weitere Tools

Es gibt zahlreiche Tools, die auf DOT und Graphviz aufsetzen, um ihre Funktionalität zur Verfügung zu stellen. Einige davon sind gerade im Java-Umfeld sehr verbreitet. Mit PlantUML *[8]* lassen sich UML-Diagramme in einem Textformat beschreiben. Gerade im Hinblick auf agile Architektur-Dokumentation findet dieses Werkzeug eine immer größere Anhängerschaft. Die Diagramme können zusammen mit dem Quelltext in dem Versionskontrollsystem gepackt werden. Durch das Textformat können somit verschiedene Stände miteinander verglichen oder sogar zusammengeführt (gemergt) werden.

Das Maven-Dependency-Plug-in sollte jedem Java-Entwickler ein Begriff sein. Es besitzt eine Funktion, mit der die Abhängigkeiten eines Maven-Moduls als DOT-Graph generiert werden können. Dieser Mechanismus ist sehr praktisch, gerade wenn es um automatische Dokumentation beispielsweise während des CI geht. Mit „mvn dependency:tree -Dincludes=com.example -DappendOutput=true -DoutputType=dot -DoutputFile=/path/to/output.gv" werden alle Abhängigkeiten von „com.example" in einem Graphen angezeigt und in die Datei „output.gv" geschrieben.

Seit dem JDK 9 kommt Java jetzt auch mit einem nativen Modulsystem, Jigsaw. Ein Tool, das seitdem mitgeliefert wird, ist „JDeps". Es kann Abhängigkeiten zwischen Java-Modulen als DOT-Graph anzeigen. Mit „jdeps -dotoutput <dir> <classes/package>" werden alle Abhängigkeiten zu anderen Java-Modulen als Graph dargestellt.

Fazit

Der Artikel hat ein Werkzeug gezeigt, um Datenstrukturen als Graphen zu visualisiert. Durch die Trennung von Beschreibung und Layout ergeben sich zahlreiche Anwendungsfälle für DOT, von der Ausgabe in den Logs bis hin zur vollständigen Web-Anwendung mit viz.js. Da es eine Fülle verschiedener Optionen gibt, um die Darstellung der Knoten und Kanten anzupassen, wurde in diesem Artikel nur auf die Grundlagen eingegangen. Hier ist auf die sehr gute und umfangreiche Dokumentation von Graphviz verwiesen.

Natürlich gibt es auch Fälle, in denen DOT nicht das geeignete Werkzeug ist. Gerade wenn es darum geht, komplexe Layout-Ansprüche umzusetzen, kann Graphviz recht widerspenstig sein. In diesem Fall sollte man nicht versuchen, DOT und Graphviz auf seinen Anwendungsfall umzubiegen, sondern stattdessen lieber den manuellen Ansatz bevorzugen.

Weitere Informationen

[1] *https://en.wikipedia.org/wiki/Unix_philosophy*
[2] *https://graphviz.gitlab.io/_pages/doc/info/attrs.html*
[3] *https://graphviz.gitlab.io/_pages/doc/info/shapes.html#html*
[4] *https://github.com/mdaines/viz.js*
[5] *https://github.com/ariutta/svg-pan-zoom*
[6] *https://github.com/drosowski/webviz*
[7] *https://github.com/magjac/d3-graphviz*
[8] *http://plantuml.com*

Daniel Rosowski
daniel@rosowski.com

Daniel Rosowski ist Software-Entwickler mit langjähriger praktischer Erfahrung im Bereich der Enterprise-Systeme. Auch wenn sein Hauptinteresse der praktischen Umsetzung gilt, stehen immer mehr Architektur-Themen auf der Tagesordnung. Seit dem Jahr 2011 ist er Mitgründer und Geschäftsführer der Smartsquare GmbH. In seiner Freizeit gründete Daniel Rosowski die Java User Group Bielefeld und ist auch heute noch als deren Sprecher aktiv.

Digitalisierung: Alter Wein in neuen Schläuchen?

Brigitte Kötting und Frank Closheim, inxire GmbH

Die Digitalisierung in Unternehmen bestimmt als Trendthema seit geraumer Zeit die Medien. Doch wie neu ist dieser Digitalisierungs-Hype wirklich – und was haben die alten Römer damit zu tun? Am Beispiel eigener Use Cases aus dem Bereich „Enterprise-Digitalization-Software" beleuchtet der Artikel den Verlauf der digitalen Transformation seit den Zeiten, als es dafür noch keinen offiziellen Namen gab.

Das Zeitalter der Digitalisierung stellt Unternehmen vor immer komplexere Herausforderungen, die sie häufig nur durch tiefgreifende IT-Reformen bewältigen können. Ihre bestehenden Systemlandschaften sind zumeist heterogen gewachsen und werden den Ansprüchen der modernen Arbeitswelt aufgrund der steigenden technischen und fachlichen Anforderungen immer weniger gerecht.

Nicht ohne Grund zählen laut der Capgemini-Studie „IT-Trends 2018" die begrenzten Anpassungsmöglichkeiten der Altsysteme zu den größten Problemen bei der Digitalisierung *[1]*. Hinzu kommt, dass die Prozesse, die hinter dem digitalen Wandel stehen, deutlich mehr als nur reines Daten-Management erfordern. Stattdessen gewinnen flexible und kollaborative Enterprise-Digitalization-Plattformen stärker an Bedeutung, um die Unternehmen automatisiert und effizient bei der Umsetzung ihrer Digitalisierungsstrategien zu unterstützen. Vor diesem Hintergrund betrifft die digitale Transformation nicht nur vereinzelte Unternehmensbereiche, sondern ist immer als konzernweites, globales Thema zu betrachten. Von den Medien wird die Digitalisierung bereits seit Jahren immer wieder als neuer Trend dargestellt. Doch ist sie das wirklich?

Digitalisierung ist nicht gleich Digitalisierung

Ohne es zu ahnen, legten bereits die Römer den Grundstein für die heutige Digitalisierung. Sie definierten ihre Längeneinheiten anhand von Körperteilen, wie den Abmessungen eines Fußes, der Länge einer Elle, der Spanne einer Hand oder der Breite eines Fingers. Letzterer heißt in Latein „digitus" – von ihm leitet sich das Wort „digital" ab: „den Finger betreffend". Der Finger ließ sich nicht nur als Maßeinheit nutzen, er eignete sich auch hervorragend als Hilfsmittel zum Rechnen. So wandelte sich der „digitus" im Laufe der Zeit zum Wort für Ziffer und zog schließlich als „digit" in die englische Sprache ein.

Obwohl Rechenmaschinen das mühsame Zählen mit den Fingern schließlich ersetzten, übernahm man den Begriff „digital" im 20. Jahrhundert dennoch ins Deutsche. Hier bedeutet er im allgemeinen Sprachgebrauch „in Ziffern dargestellt". Während wir mit „Digitalisierung" zum einen die Umwandlung analoger Werte in digitale Formate und zum anderen den digitalen Wandel selbst beschreiben, stehen dem deutschen Wort zwei englische Definitionen gegenüber. „Digitization" steht für die Umwandlung von analogen in digitale Daten, „Digitalization" hingegen beschreibt den Einsatz digitaler Technologien in Unternehmen, den Aufbau digitaler Prozesse zur Nutzung und Optimierung von Daten bis hin zur Entwicklung neuer digitaler Geschäftsmodelle.

Heute, mehr als 1.500 Jahre nach dem Untergang des weströmischen Reichs, könnte der Stand der Digitalisierung in deutschen Unternehmen kaum unterschiedlicher sein. Die etventure-Studie „Digitale Transformation 2018 – Hemmnisse, Fortschritte, Perspektiven" zeigt, dass sich zwar 43 Prozent der befragten Großunternehmen „gut" oder „sehr gut" auf die digitale Transformation vorbereitet sehen. Umgekehrt fühlen sich jedoch 57 Prozent entsprechend schlechter für den digitalen Wandel aufgestellt *[2]*.

Wie verschieden das Feld der Digitalisierung bearbeitet wird, erleben die Autoren seit mehr als fünfzehn Jahren in der Zusammenarbeit mit ihren Kunden. Anhand von drei Use Cases vermitteln sie einen Einblick in den Verlauf der digitalen Transformation aus der Sicht eines Entwicklers von Digitalisierungssoftware.

Use Case 1: Zentrales Wissensmanagement statt unstrukturierter Daten

Der erste Anwendungsfall behandelt die europäische Tochtergesellschaft eines US-amerikanischen Automobilkonzerns. Schon im Jahr 2002 unternahm sie ihre ersten Schritte in Richtung Unterneh-

mensdigitalisierung – ein Begriff, der zu dieser Zeit allerdings noch nicht dafür verwendet wurde.

Die Globalisierung stellte das Unternehmen vor ein geschäftsrelevantes Problem: Die Entwicklungsteams arbeiteten europaweit verteilt und nutzten unterschiedliche Datenbestände, abgelegt in Datenbanken und Fileshares. Der Zugriff darauf war für die anderen Standorte stark eingeschränkt und der fehlende Kontext, beispielsweise bei technischen Zeichnungen, bedeutete einen erheblichen Aufwand bei der Suche nach den richtigen Informationen.

Um den Mitarbeitern ein effizientes Arbeiten über Landesgrenzen hinweg zu ermöglichen, musste das Unternehmen seine heterogenen Ablagestrukturen konsolidieren und ein zentrales Wissensmanagement aufbauen. Darin sollten sowohl alle Daten der Fahrzeugentwicklung als auch das entsprechende Wissen der Mitarbeiter gesammelt und verfügbar gemacht werden. Am Ende der fünfmonatigen Entwicklung erhielten rund 7.000 Computer-Arbeitsplätze Anbindung an eine einheitliche Plattform für Content Management und Collaboration.

Das Besondere an dem für den Konzern entwickelten System war die Volltext- und Metadatensuche über alle Inhalte, die zudem auf einer einheitlichen Ablagestruktur und einer standardisierten Klassifizierung basierten. Über die selbsterklärende Oberfläche wurden die Daten mittels Fenster-Technik, Kontext-Menüs und Drag & Drop schnell und sicher gespeichert, verwaltet und ausgetauscht. So konnte jeder Benutzer Informationen einpflegen, ohne spezielle Programmierkenntnisse zu erwerben. Der gesamte Content lag in einer objektrelationalen Datenbank und im medienneutralen XML-Format vor. Das schuf die idealen Voraussetzungen für eine direkte Weiterverwendung der Inhalte in Print- und elektronischen Medien.

Die crossfunktionalen Teams profitierten von den integrierten Funktionen, die es ihnen ermöglichten, in virtuellen Projekträumen miteinander zu kommunizieren, Daten auszutauschen sowie Projekte zu planen und durchzuführen. Jedes Teammitglied konnte zu jedem Zeitpunkt auf die Daten zugreifen und aktiv am Informationsaustausch teilnehmen, wobei die einzelnen Zugänge über vielfältige Berechtigungen gesteuert werden konnten.

Der Einsatz des neuen Wissensmanagement-Systems hatte sowohl auf die IT-Landschaft als auch auf das Business positive Auswirkungen. Es stand nun eine zentrale Informationsquelle zur Verfügung, die eine niedrige Total Cost of Ownership (TCO) bot und dabei skalierbar, performant, hochverfügbar und stabil war. Es gelang dem Automobilkonzern damit, den Aufwand für Abstimmungen deutlich zu optimieren und das Projektmanagement transparenter zu gestalten. So sparte das Unternehmen Zeit und Kosten und steigerte zudem die Motivation seiner Mitarbeiter. Durch den modularen Charakter der Software-Plattform konnte der Konzern zudem jederzeit Erweiterungen im laufenden Betrieb installieren und den funktionalen Umfang des Gesamtsystems weiter ausbauen.

Use Case 2: Datenströme verarbeiten und weltweit verteilen

Unternehmensdaten digital zu erfassen, zu vereinheitlichen und in einen Kontext zu bringen, ist ein erster wichtiger Schritt der Digitalisierung und schafft bereits einen klaren Mehrwert für das Unternehmen. Einen Schritt weiter geht der zweite Use Case: Eine US-amerikanische Forschungseinrichtung erfasst automatisiert große Mengen an Daten und reichert diese mit vorhandenen digitalen Informationen an. So ergänzen sich beide Datensets zu einem übergeordneten Datenstrom.

Die Wissenschaftler des Instituts beschäftigen sich seit Jahren mit der Frage, wie sich nachhaltig saubere Energie für die Menschheit gewinnen lässt. Um eine Antwort darauf zu finden, nutzen sie den größten und stärksten Laser der Welt für ihre Forschungen zur Trägheitsfusion. Auf einer Fläche von drei Football-Feldern fokussieren sie 192 Laserstrahlen auf ein stecknadelkopfgroßes Zielobjekt: eine zwei Millimeter große Kunststoffkapsel, gefüllt mit Deuterium und Tritium.

Die Energie des Lasers komprimiert die kleine Kugel um mehr als das Tausendfache, erhitzt sie und zündet die Fusionsreaktion – die Atomkerne des Gemischs verschmelzen. Die Spitzenleistung des Lasers beträgt zeitweise mehr als fünfhundert Billionen Watt. Wird das Zielobjekt von allen Strahlen gleichzeitig getroffen, erreicht es Temperaturen von bis zu hundert Millionen Grad Celsius.

Detailgenau registrieren unzählige Messsonden jeden Schuss des Lasers, der zudem durch wissenschaftliche Experten auf der ganzen Welt teils aufwendig durch eine manuelle Analyse vor- und nachbereitet werden muss. Der Datenstrom, erzeugt durch die gewonnenen Sensordaten, wird somit durch die Analysedaten aus Vor- und Nachbereitung der Experten angereichert. Pro Monat entstehen dadurch mehrere Terabyte an hochsensiblen Daten, die gespeichert und weiterverarbeitet werden müssen.

Um die enorm großen Datenmengen aus ihren Laser-Experimenten zu bewältigen, nutzt die Einrichtung seit dem Jahr 2007 eine skalierbare Digitalisierungsplattform und entwickelt auf deren Basis eine Vielzahl von Applikationen für die Planung und Terminierung der Laser-Experimente, die Instandhaltung der Anlage sowie die Speicherung, Verarbeitung und Verteilung der Daten.

Dreißig Jahre lang stellt die Plattform die aufbereiteten Informationen ausgewählten Spezialisten weltweit zur Verfügung. Diese können die Werte im Anschluss selbst analysieren und für ihre eigene Forschung nutzen, bestehende Verfahren optimieren oder neue Methoden entwickeln.

Use Case 3: Digitales Instandhaltungsmanagement

Wie die Digitalisierung komplette Arbeitsprozesse neugestalten und optimieren kann, zeigt der dritte Use Case. Ein deutsches Verkehrsunternehmen erstellt und verwaltet die Instandhaltungs-Dokumentationen seiner Schienenfahrzeuge in Form von Arbeitsanweisungen und Instandhaltungs-Handbüchern. Um einen reibungslosen Betrieb zu gewährleisten, muss das Unternehmen diese den Werken jederzeit in der aktuellsten Version zur Verfügung stellen. Zudem muss der komplette Erstellungsprozess lückenlos belegbar sein.

Dieser komplexe Prozess erfolgte früher in einem historisch gewachsenen IT-System, das den Bedürfnissen der mehreren Hundert Anwender jedoch aufgrund der steigenden technischen und fachlichen Anforderungen immer weniger gerecht wurde. Auch aus wirtschaftlichen Gründen waren Änderungen, Wartung und technischer Betrieb nicht mehr zweckmäßig.

Die Verantwortlichen führten deshalb im Jahr 2014 eine digitale Maintenance-Plattform ein, die ihre Erstellungs- und Verwaltungsprozesse nicht nur unterstützen, sondern durch digitalisierte und dynamische Workflows auch deutlich beschleunigen sollte. Seitdem profitiert das Unternehmen unter anderem von einer wesentlichen Reduzierung des Aufwands bei der Regelwerkserstellung.

Auch für die Logistik der Werkstattzuteilung lässt sich das System nutzen. So kann bei einem Arbeitsauftrag beispielsweise direkt angezeigt werden, in welcher Werkstatt er ausgeführt werden kann – unter Berücksichtigung freier Kapazitäten und baulicher Gegebenheiten vor Ort. Zudem legt die Plattform den Grundstein für neue Bereiche wie Predictive Maintenance: Die „vorausschauende Wartung" erkennt dank Echtzeit-Datenanalyse frühzeitig Abweichungen und meldet potenzielle Störungen, noch bevor sie auftreten.

Die hier vorgestellten Use Cases sind nur drei von unzähligen Beispielen dafür, wie vielfältig Unternehmen ihre digitale Transformation seit Jahren gestalten. Nach Erfahrung der Autoren nach haben sie alle eines gemeinsam: Sie besitzen noch viel Digitalisierungspotenzial, das darauf wartet, ausgeschöpft zu werden.

Die kontinuierlichen Veränderungen der Märkte erfordern eine schnelle Anpassung von Unternehmensstrukturen und Prozessen. Eine Neuausrichtung lässt sich in der nötigen Geschwindigkeit jedoch nur mit optimaler digitaler Unterstützung wirtschaftlich sinnvoll realisieren. Die eingesetzten Software-Systeme müssen sich nahtlos in die bestehende IT-Landschaft einfügen und dabei selbst flexibel genug sein, um notwendige Strategie-Anpassungen jederzeit aktiv mitzutragen.

Trotz allen Fortschritts stößt die heutige Digitalisierung aber auch an ihre Grenzen. Dies zeigt sich unter anderem am Beispiel von Mixed-Reality-Brillen. Diese könnten etwa die Arbeit in Produktionshallen oder Werkstätten deutlich effizienter, sicherer und bequemer gestalten. Doch bis sie wirklich zum beruflichen Alltag gehören, müssen sowohl die dafür eingesetzte Hardware als auch die vorhandene Infrastruktur teilweise deutlich verbessert werden. So reicht die Verbindungsgeschwindigkeit der Geräte häufig nicht für die benötigten Anwendungen aus und auch der niedrige Tragekomfort, die kurzen Akkulaufzeiten sowie eine geringe Widerstandsfähigkeit gegen Staub, Hitze oder Feuchtigkeit sprechen derzeit noch gegen einen dauerhaften Einsatz vor Ort.

Dies steht beispielhaft für die aktuellen Herausforderungen der Digitalisierung. Aufgrund von Ressourcen-Problemen werden viele Projekte entweder nur als Pilot durchgeführt oder die Umsetzung erfolgt wegen der teils hohen Kosten nur dort, wo es wirklich dringend ist. So kommt es nicht selten vor, dass Unternehmen auf dem Weg zum digitalen Business ins Stolpern geraten.

Fazit

Die Technologie ist ein wesentlicher Faktor der digitalen Transformation. Ebenso wichtig sind die Menschen, die sie täglich nutzen. Für sie müssen wir den Weg der Unternehmensdigitalisierung so einfach wie möglich gestalten. Die heute alltägliche Nutzung von Internet und sozialen Netzwerken hat die Anforderungen an den digitalen Arbeitsplatz deutlich geprägt. Interaktive Features wie Teilen, Liken oder Kommentieren sollten genauso selbstverständlich sein wie Newsfeeds und eine intelligente Suche, die auf Basis von Machine-Learning-Algorithmen aus jeder Benutzerinteraktion lernt.

Die digitale Transformation ist nicht von heute auf morgen zu bewältigen. Eines jedoch ist sicher: An der Digitalisierung führt kein Weg vorbei. Ist sie aber nun ein Hype oder „alter Wein in neuen Schläuchen"? Die Erfahrung der Autoren zeigt, dass die Digitalisierung schon vor langer Zeit Einzug in die Unternehmen gehalten hat. Geändert hat sich vor allem die Geschwindigkeit, mit der sie heute umgesetzt wird.

Verbesserungen bei der Hard- und Software, das Aufkommen von kostengünstigem Speicher und modernen Informationstechnologien wie Big Data, Blockchain oder Machine Learning sowie eine neue digitale Unternehmenskultur beschleunigen die Transformation in einem bisher nicht gekannten Maß. So bleibt das Thema auch in den kommenden Jahren spannend. Und mehr denn je gilt heute eine Weisheit, die der französische Wissenschaftler Louis Pasteur bereits im 19. Jahrhundert formulierte: „Veränderungen begünstigen nur den, der darauf vorbereitet ist."

Weitere Informationen

[1] Capgemini-Studie „IT-Trends 2018": *www.capgemini.com/it-trends*

[2] etventure-Studie „Digitale Transformation 2018 - Hemmnisse, Fortschritte, Perspektiven": *https://studie2018.etventure.de*

Frank Closheim
frank.closheim@inxire.com

Frank Closheim ist bei inxire als Head of Product Management tätig. Er ist Experte für Enterprise-Digitalization-Software, hat achtzehn Jahre Erfahrung in der Java-Programmierung und arbeitete mehr als zwölf Jahre in verschiedenen Rollen bei Oracle. Zudem war er Referent auf zahlreichen Konferenzen und ist Autor verschiedener Fachartikel und Produkt-Dokumentationen.

Brigitte Kötting
brigitte.koetting@inxire.com

Brigitte Kötting verantwortet bei inxire die Bereiche Marketing und Kommunikation. Umgeben von Entwicklern und im Umgang mit den Kunden begegnen ihr dort täglich die unterschiedlichen Ausprägungen der Digitalisierung in Unternehmen.

Die neue Oracle-Supportpolitik für Java im Detail

Michael Paege, DOAG Competence Center Lizenzierungsfragen

Bereits im April 2018 hat Oracle eine neue Supportpolitik für Java angekündigt. Diese ist mittlerweile im Markt angekommen und Verunsicherung macht sich breit. Kunden stehen vor Fragen wie: Kostet Java jetzt Geld? Gibt es Lizenzierungs- oder Compliance-Risiken? Nicht zuletzt: Darf Oracle das?

Sun hat auf der JavaOne im Mai 2006 verkündet, dass Java im Quelltext unter freier Lizenz als freie Software veröffentlicht werden soll und dies nachfolgend auch umgesetzt. Hieraus entwickelte sich das OpenJDK. Die Java-Entwicklung wurde, nachdem Oracle Sun und damit auch Java übernommen hat, im Jahr 2006 aufgeteilt. Oracle treibt seitdem die Weiterentwicklung des JDKs voran und stellt eine kommerzielle Version, das OracleJDK, zur Verfügung. Dieses basiert auf dem OpenJDK und war bisher kostenfrei, enthielt aber auch im Bundle kostenpflichtige Features (wie beispielsweise Flight Recorder, Mission Control, MSI Installer), die eine Lizenzierung von Java SE erfordern.

Traditionell existieren zwei Versionen, eine Laufzeit-Umgebung, das sogenannte „Java Runtime Environment" (JRE), und ein Java Development Kit (JDK), das die Laufzeit-Umgebung um Entwicklungswerkzeuge wie den Compiler erweitert. Für das Oracle JDK hat Oracle nun folgendes entschieden:

- Der Support (Updates etc.) für Java 8, das seit dem Jahr 2014 verfügbar und das aktuell am meisten eingesetzte Oracle JDK ist, soll ab Januar 2019 nur noch kostenpflichtig zur Verfügung stehen.

- Der Support für Zwischen-Versionen soll nur noch für sechs Monate zur Verfügung zu stehen.
- Der Long Term Support (LTS), also länger als sechs Monate, soll nur noch für das im Herbst 2018 erscheinende Java 11 sowie für das im Jahr 2021 erscheinende Java 17 angeboten werden.
- Ab Java 11 will Oracle das Oracle JDK nur noch für Entwicklung, Test, POC und Demo kostenfrei zur Verfügung zu stellen. Für den produktiven Einsatz des Oracle JDK 11 und höher sollen Kunden dann die Java SE Subscription kaufen müssen.
- Ab Java 11 will Oracle kein separates JRE mehr zur Verfügung stellen. Somit ist eine kostenfreie Production-Nutzung mit Oracle JDK11 und höher nicht mehr möglich.

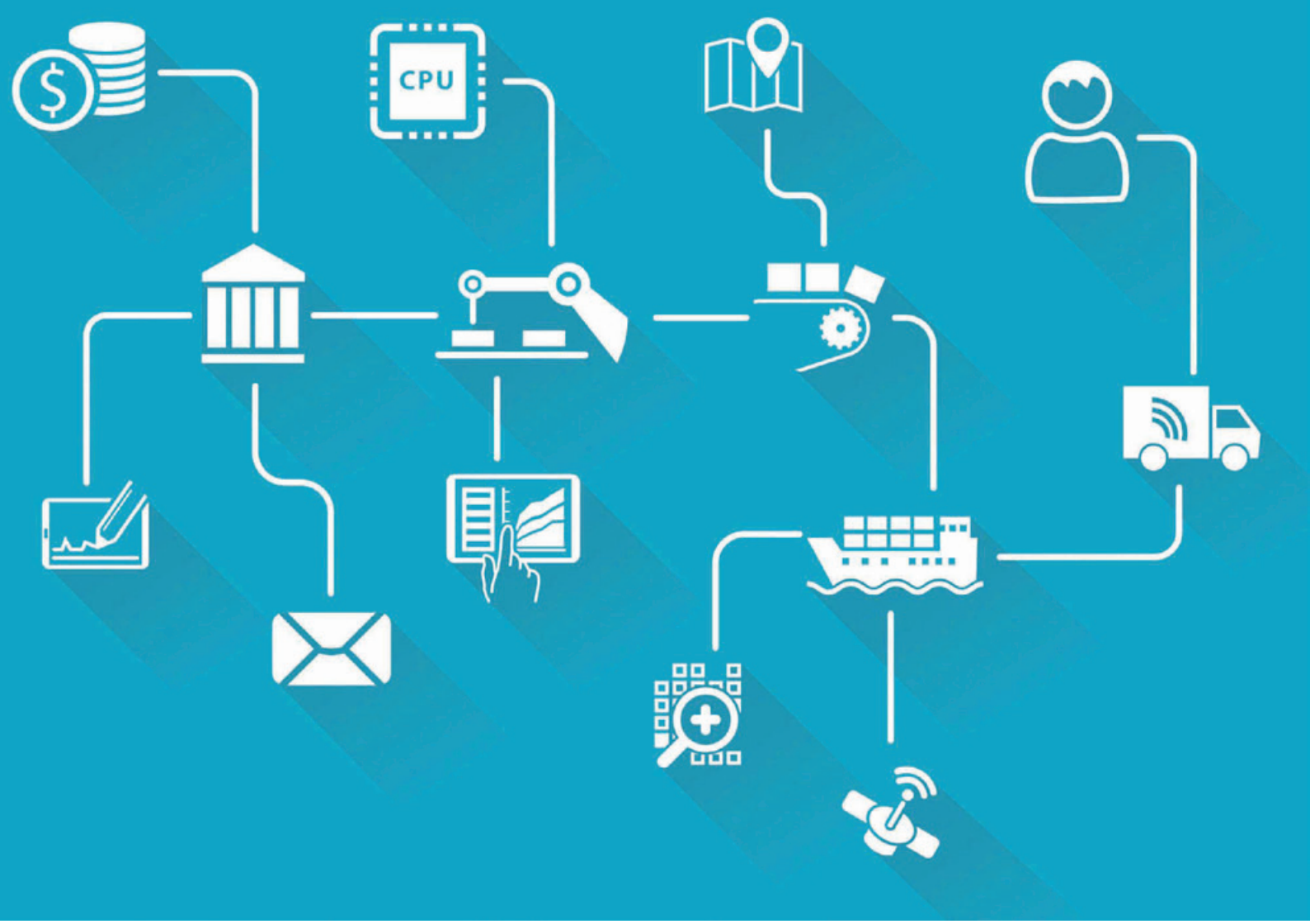
Die neue Java Release Timeline steht unter *„https://www.ijug.eu/fileadmin/images/2018/New_Java_Release_Timeline.png".* Da das Oracle JDK auf OpenJDK basiert, bedeutet dies für OpenJDK-Nutzer, dass sie sich an einen sechsmonatigen Release-Zyklus gewöhnen müssen. Aus Lizenzsicht ändert sich aber nichts. Für Kunden, die Oracle-Middleware-Produkte einsetzen, die Java SE enthalten, und die für diese Middleware-Produkte einen gültigen Support-Vertrag haben, ändert sich nichts. Diese sind aktuell: WebLogic Standard Edition, WebLogic Enterprise Edition, WebLogic Suite, Internet Application Server Enterprise Edition, GlassFish Server, Coherence Standard Edition, Coherence EE, Coherence Grid Edition, WebCenter Content, WebCenter Universal Content Management. Auch Oracle Forms ist laut Aussage des Product Managers Michael Ferrante nicht betroffen. Wie die Situation bezüglich der Oracle-Datenbank aussieht, die ebenfalls Java enthalten kann, wird aktuell noch geklärt.

Kunden, die Oracle JDK 8 einsetzen, können dies auch weiterhin kostenfrei tun, bekommen aber keine Updates mehr, was aus Security-Gesichtspunkten kritisch ist, denn Java-Umgebungen waren in der Vergangenheit oft Ziel von Angriffen. Die hohe Anzahl an sicherheitskritischen Schwachstellen (Severe 5 und höher) zeigt, wie hoch hier die Sicherheit bewertet werden muss. Kunden, die Oracle JDK 8 einsetzen und dafür Sicherheitsupdates benötigen, sind gezwungen, die Java SE Subscription zu kaufen. Oracle gewährt Premier Support bis März 2022, Extended Support erfolgt bis März 2025. Kunden, die zukünftig auf Oracle JDK 11 migrieren wollen, benötigen die Java SE Subscription für den Produktiv-Einsatz, aber auch weil keine separate JRE mehr zur Verfügung gestellt wird. Kunden, die auf OpenJDK wechseln, können Java weiterhin kostenfrei einsetzen. AdoptOpenJDK will für Java 8 kostenfreien Support bis September 2022 gewährleisten.

Bei den Compliance-Risikenmuss man zwischen Lizenzrisiken und Security-Risiken unterscheiden: Lizenz-Risiken entstehen durch die beschriebenen Ankündigungen aus Sicht der DOAG eher nicht, sie bestanden seit geraumer Zeit und bestehen weiterhin durch das Bundling von freien und kostenpflichtigen Features im Oracle JDK. Dies ist aber nicht neu. Security-Risiken werden steigen, wenn sich Kunden entscheiden, weiterhin Oracle JDK 8 zu verwenden und nicht die Java SE Subscription zu kaufen, um die Updates zu erhalten.

Für Unternehmen gibt es Kostensteigerungen, sofern sie Oracle JDK einsetzen. Der bequemste Weg, weiter Java 8 zu nutzen und Security-Updates zu erhalten, bedeutet den Kauf der Java SE Subscription. Die Alternativen, wie der Umstieg auf OpenJDK, bedeuten viel Recherche-, Migrations- und Testaufwand.

Oracle darf das vermutlich, denn das freie Java besteht in Form von OpenJDK weiterhin. Dieser Aspekt sowie weitere rechtliche Fragestellungen in diesem Umfeld werden aktuell im DOAG Legal Council geprüft und bewertet.



Effiziente Delivery mit APIs, Microservices und DevOps

Sven Bernhardt, OPITZ CONSULTING Deutschland GmbH

Traditionelle IT-Systemlandschaften bestehen oft aus monolithischen Applikationen, die häufig langwierigen, formalisierten Release-Zyklen unterliegen. Aus Sicht der Gesamtstabilität ist das auch sinnvoll, da monolithische Applikationen in der Regel eine Sammlung von eng miteinander verzahnten Business Capabilities abbilden – fachlich bedingte Änderungen sind so allerdings nur innerhalb der Release-Zyklen möglich – und somit wenig agil sind. Konträr dazu steht heute die zentrale Herausforderung der IT, Agilität im Unternehmen sicherzustellen. Änderungen an bestehenden Komponenten sollen schnell durchgeführt und bereitgestellt werden, ohne dass dies Auswirkungen auf bestehende Systeme und Services hat. Definitiv eine Herausforderung! Mit der richtigen Architektur-Idee und geeigneten Werkzeugen jedoch keine unlösbare ...

Moderne Ansätze wie Microservices-Architekturen *[1]* setzen auf die strikte Trennung unterschiedlicher Business Capabilities, die unabhängig voneinander implementiert, bereitgestellt und betrieben werden können. Ist eine Kommunikation zwischen den Services erforderlich, erfolgt diese in der Regel eventbasiert über einen Event-Hub. Einzelne Microservices sind somit vollständig voneinander entkoppelt. Änderungen, bedingt durch neue oder geänderte Anfor-

derungen, können also ohne Beeinträchtigung bereits vorhandener Funktionalitäten erfolgen; soweit die Theorie.

In der Praxis geht es allerdings nicht ohne ein Umdenken in der Organisation. Betrieb und Entwicklung müssen enger zusammenrücken. Die konsequente Umsetzung eines DevOps-Ansatzes ist unabdingbar für den Erfolg von Microservices, um Vorteile wie die Steigerung von Agilität und Effizienz realisieren zu können. Die Einführung eines DevOps-Ansatzes bedingt neue Denk- und Arbeitsweisen innerhalb der IT-Organisation. Dazu zählen unter anderem:

- Das Aufbrechen getrennter Entwicklungs- und Betriebsbereiche
- Die Formung neuer, heterogener Teams
- Die Automatisierung großer Teile des Entwicklungsprozesses

Das neue Mantra, das die neu geformten Teams in diesem Zusammenhang verinnerlichen müssen, lautet: „You build it, you run it!" Ein Prozess, der nicht von heute auf morgen abgeschlossen ist. Die erfolgreiche Einführung von Microservices ist also mit nicht zu unterschätzenden organisatorischen Herausforderungen verbunden, insbesondere für gewachsene IT-Organisationen. Aber auch technologisch beziehungsweise architektonisch ergeben sich diverse Herausforderungen – denn in den seltensten Fällen startet ein Unternehmen auf der grünen Wiese.

Referenz-Architektur für flexible Anwendungs-Architekturen

Das Projekt der Open Modern Enterprise Software Architecture (OMESA) *[2]* beschäftigt sich mit verschiedenen Fragestellungen, zum Beispiel damit, wie bewährte architektonische Grundprinzipien und Architekturmuster in modernen Software-Architekturen zu verankern sind. Ziel ist es, alte und neue Welt sinnvoll miteinander zu kombinieren. Anstelle kompletter Restrukturierung und Refaktorisierung heißt die Devise „sinnvolle Koexistenz". Ein solches Vorgehen ist gerade in Bezug auf langjährig gewachsene IT-Systemlandschaften sinnvoll.

Eine der zentralen Botschaften von OMESA lautet: „Microservices are no silver bullets!" OMESA definiert zu diesem Zweck eine Referenz-Architektur sowie ein mehrstufiges Capability Model *(siehe Abbildung 1)*. Sie zeigt die zentralen Ebenen, wobei Microservices im Bereich der „Service Implementation" *[3]* einzuordnen sind, als sogenannte „Fully-decoupled Services". Auf der Ebene der „Persistence" befinden sich die bestehenden und zumeist monolithischen IT-Systeme, die über „Semi-decoupled Services" in die Gesamt-Architektur eingebunden sind.

Oberhalb der Ebene „Service Implementation" ist die API-Ebene verortet, die sich wiederum in die Bereiche „Single-Purpose API" und „Multi-Purpose API" aufteilt. Auf der obersten Ebene, der „Delivery Experience", geht es schließlich um die Interaktion mit der Außenwelt; in vielen Fällen handelt es sich hierbei um eine Mensch-Maschine-Interaktion. Dabei geht es vor allem um Themen wie moderne Client-Applikationen oder die Möglichkeit, über verschiedene Kanäle wie Chatbots mit den Services eines Unternehmens interagieren zu können.

APIs verbinden Microservices und UIs

In der OMESA-Referenz-Architektur ist die API-Ebene ein grundlegender Baustein moderner Software-Architekturen. Aber warum sind APIs so essenziell? Um möglichst unabhängig voneinander zu bleiben, interagieren Microservices untereinander hauptsächlich asynchron beziehungsweise eventbasiert. Für die Kommunikation mit der Außenwelt, beispielsweise über Benutzeroberflächen, ist dieser Kommunikationsstil allerdings im Sinne einer guten User Experience (UX) nicht geeignet. Ein synchrones Kommunikationsverhalten, das sich durch zeitnahe Reaktionen auszeichnet, fühlt sich für menschliche Akteure natürlicher an und sollte deshalb auch hier das bevorzugte Kommunikationsmuster sein. Das bedeutet, dass Microservices, deren Funktionalitäten extern verfügbar gemacht werden, ein synchrones Service-Interface, also ein API, bereitstellen müssen.

Der in OMESA propagierte, zweischichtige API-Ansatz, der Single- und Multi-Purpose-APIs unterscheidet, macht die Gesamt-Architektur flexibler und agiler *[4]*. Multi-Purpose-APIs sind allgemeiner, bieten einen breiteren Funktionsumfang und sind somit potenziell wiederverwendbar; Single-Purpose-APIs hingegen können auf den

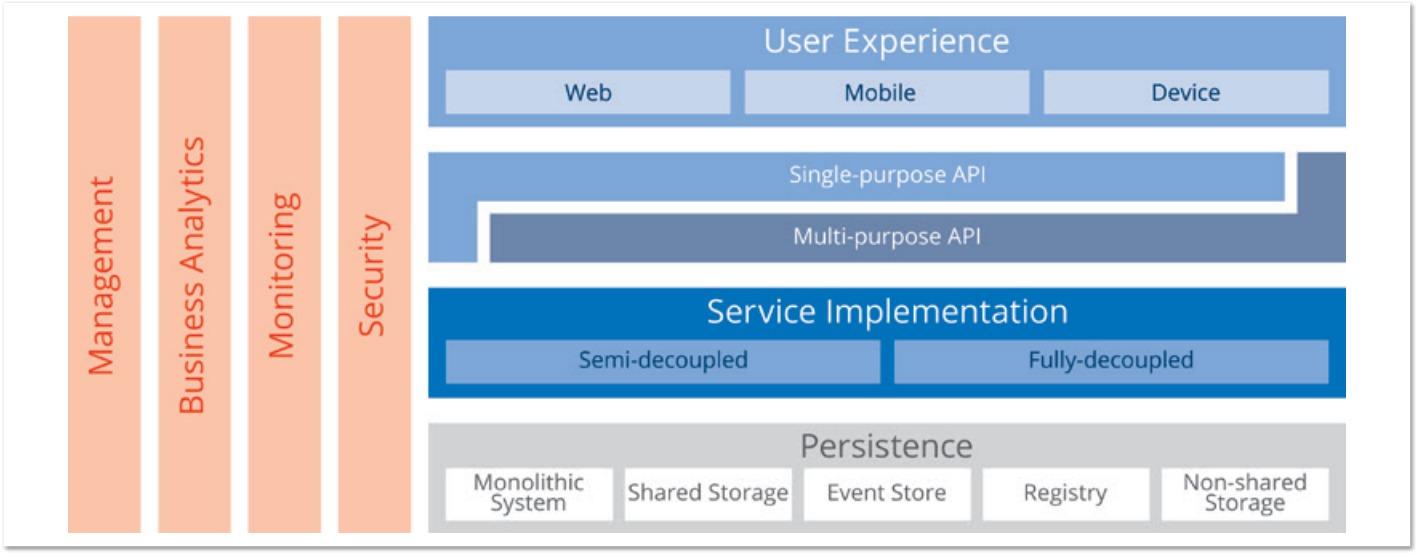


Abbildung 1: OMESA-Referenz-Architektur

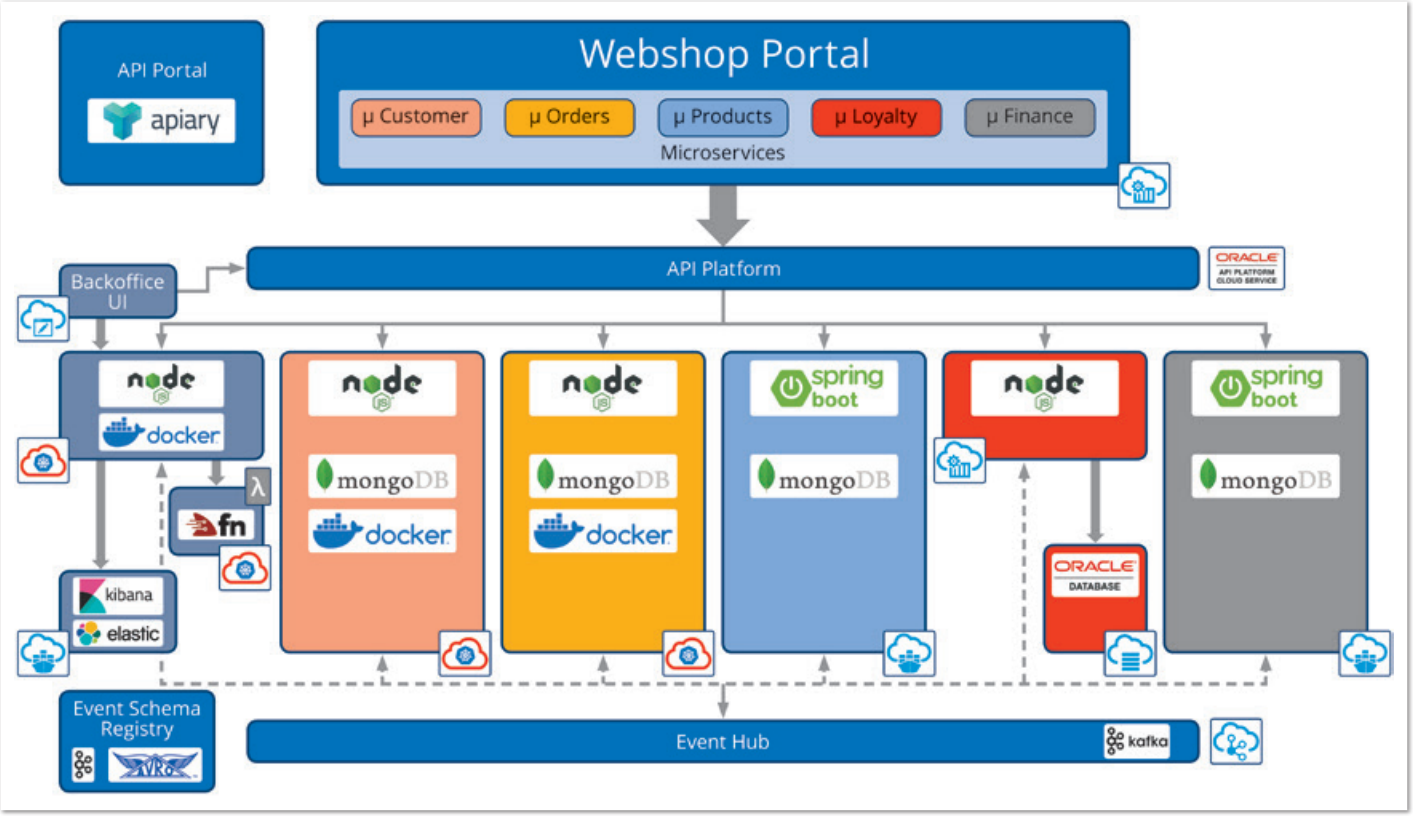


Abbildung 2: Gesamt-Architektur der Webshop-Lösung

Multi-Purpose-APIs aufbauen und bilden dabei UI- oder Device-spezifische Logik ab. Single-Purpose-APIs stellen im Grunde eine Implementierungsvariante des „Backends For Frontends"-Pattern (BFF) *[5]* dar.

Die API-Ebene dient also vor allem dazu, die Service-Implementierung von der Benutzeroberfläche zu abstrahieren und die von den Services bereitgestellten Funktionalitäten sicher nach außen zu exponieren. „Sicher" bedeutet hier, dass die API-Ebene übergreifende Aspekte wie grundlegende Sicherheitsmechanismen (wie Authentifizierung und Autorisierung), Origin-Controls (Cross-Origin Sharing Resources, CORS *[6]*) oder Threat-Protection-Maßnahmen (wie Rate Limits) zentral und konsistent definiert werden, ohne diese explizit in jedem Service ausimplementieren zu müssen. Das hat den Vorteil, dass sich Backend-Entwickler voll und ganz auf die Implementierung der Geschäftslogik konzentrieren können.

Zwischenfazit der wichtigsten Fakten

Zusammenfassend kann bislang festgehalten werden, dass die Umsetzung moderner Architekturen auf der Basis von Microservices sowohl organisatorische als auch architektonische sowie technische Herausforderungen mit sich bringt. Zudem ist die Etablierung von DevOps essenziell für den Erfolg von Microservices. APIs sind in diesem Zusammenhang wichtig, um Service-Funktionalitäten nach außen anbieten zu können und so Mehrwerte wie die Etablierung neuer digitaler Economies zu generieren.

Im zweiten Teil dieses Artikels sollen die angesprochenen Aspekte kurz anhand eines praktischen Beispielprojekts näher beleuchtet werden. Es handelt sich dabei um eine gemeinschaftliche Entwicklung der Oracle-ACEs Lonneke Dikmans, Lucas Jellema, Luis Weir, Guido Schmutz, José Rodrigues und Sven Bernhardt, die sich in Vorbereitung auf einen Vortrag für das jährlich stattfindende PaaS-Forum im März 2018 als Team zusammenfanden, um einen Showcase auf Basis der Oracle Cloud zu entwickeln. Die Ideen der OMESA-Referenz-Architektur dienten dem Team dabei als architektonische Grundlage für die Implementierung.

Beispielszenario: eine Webshop-Lösung

Als Beispielszenario diente dem Team ein fiktiver Webshop, der auf einer Microservices-Architektur basiert. Jedes Teammitglied war für die Umsetzung einer bestimmten Business Capability, also für jeweils einen Microservice, zuständig. *Abbildung 2* zeigt die Gesamt-Architektur sowie die verwendeten Technologien im Überblick.

Wie der *Abbildung 2* entnommen werden kann, sind bei der Umsetzung viele verschiedene Technologien zum Einsatz gekommen, um den Webshop mit seinen sechs Microservices „Logistics", „Customers", „Orders", „Products", „Loyalty" und „Finance" umzusetzen. Als Laufzeit-Umgebung für die Microservices dienten verschiedene Cloud Services: Oracle Application Container Cloud Service, Oracle Container Cloud Classic und Oracle Container Engine for Kubernetes. Die Interaktion zwischen den Microservices erfolgt eventbasiert über den Oracle Event Hub Cloud Service. Um die Verbindlichkeit der Event-Definitionen sicherzustellen und die Abstimmungen zwischen den Teams zu vereinfachen, kommt eine Avro Event Schema Registry zum Einsatz *[7]*.

Da die Microservices von UIs verwendet werden sollen, müssen sie REST-APIs anbieten, die über den Oracle API Platform Cloud Service (APIP CS) nach außen exponiert werden. Der Einfachheit halber und da im ersten Schritt nur eine Web UI bedient werden musste, wird

auf API-Ebene nicht zwischen Multi- und Single-Purpose-APIs unterschieden.

Beim Webshop-Portal, das auf Oracle JET basiert, handelt es sich um keine klassische, monolithische Web-Applikation. Vielmehr stellt es nur den Rahmen zur Verfügung, der die übrigen Microservice-spezifischen UIs einbindet, was maximale Flexibilität bei Änderungen bedeutet. Wenn beispielsweise aufgrund technischer Anpassungen ein Deployment des Microservice „Finance" notwendig wird, kann dieses jederzeit durchgeführt werden, ohne die übrigen Services zu beeinträchtigen. Benutzer können also weiterhin den Produktkatalog durchsuchen oder Bestellungen durchführen; nur im „Finance"-Bereich kommt es kurzfristig zu Einschränkungen. Die Gesamt-Architektur ist also sehr flexibel aufgebaut. Sie besteht aus Einzelkomponenten, die auf horizontaler Ebene voneinander unabhängig sind. Auf diese Weise kann sehr agil auf sich ändernde Fachanforderungen reagiert werden.

„API first"-Entwicklung

Intuitive APIs sind kritische Erfolgsfaktoren moderner Software-Architekturen. Sie sollten einfach zu bedienen, schwer zu missbrauchen, benutzer- sowie wartungsfreundlich und konsistent definiert sein. Kurzum: Gutes API-Design ist wichtig für die Akzeptanz der Anwender und damit äußerst relevant für die Nutzung eines API. Wie bei der UX für UIs muss man sich also auch hier genauer ansehen, wie das API vom Konsumenten verwendet wird. Deshalb starten wir die Entwicklung nicht etwa mit der Implementierung der Backend-Logik, sondern mit dem Design des API.

Ein „API first"-Design-Ansatz ist für die Flexibilität und Agilität während des gesamten API-Lebenszyklus unerlässlich. Darüber hinaus ermöglicht „API first" die kollaborative Zusammenarbeit verschiedener Stakeholder an einer API-Definition und entkoppelt so API-Implementierung, UI- und Backend-Service-Entwicklung. Dieser Zusammenhang wird in *Abbildung 3* gezeigt.

Für die Microservices des Webshops wurde also zunächst das API beschrieben. Das Team setzte dafür die Oracle Apiary Plattform ein, über die das zugehörige API wahlweise mit Swagger oder API Blueprint beschrieben werden kann. Der Vorteil von Apiary ist, dass das API nach Fertigstellung der Beschreibung durch die Plattform direkt in einer Mock-Variante zur Verfügung gestellt wird und verwendet werden kann.

Wie in *Abbildung 3* dargestellt, können nun App Developer, API Developer und Backend Developer direkt und völlig unabhängig voneinander mit der Implementierung der Web-UI, des API und der Backend-Service-Implementierung starten. Da die API-Beschreibung so allen Stakeholdern bereits zu einem sehr frühen Zeitpunkt zur Verfügung steht, sind Änderungen am API einfach und ohne großen Aufwand möglich. Dank der kurzen Feedbackzyklen kommt man schnell zu einer guten API-Usability.

Test-Automatisierung

Das Thema „Test-Automatisierung" spielt im Kontext von DevOps eine wichtige Rolle. Um Änderungen möglichst schnell produktiv zur Verfügung stellen zu können, ist eine hohe Test-Abdeckung notwendig. Bei der Entwicklung der Backend-Logik der Microservices schrieb das Team Unit-Tests, die dann zum Build-Zeitpunkt mithilfe entsprechender Build-Management-Tools wie Apache Maven automatisiert ausgeführt wurden. In diesem Zusammenhang stellte sich allerdings die Frage, ob und wie es möglich sein würde, Tests für API-Definitionen zu automatisieren. Denn schließlich soll ja auch sichergestellt werden, dass sich ein Backend Service konform zur API-Definition verhält.

Genau solche Tests lassen sich mit Dredd *[8]* automatisieren, einem HTTP-API-Testing-Framework, das Tests gegen ein API ausführen

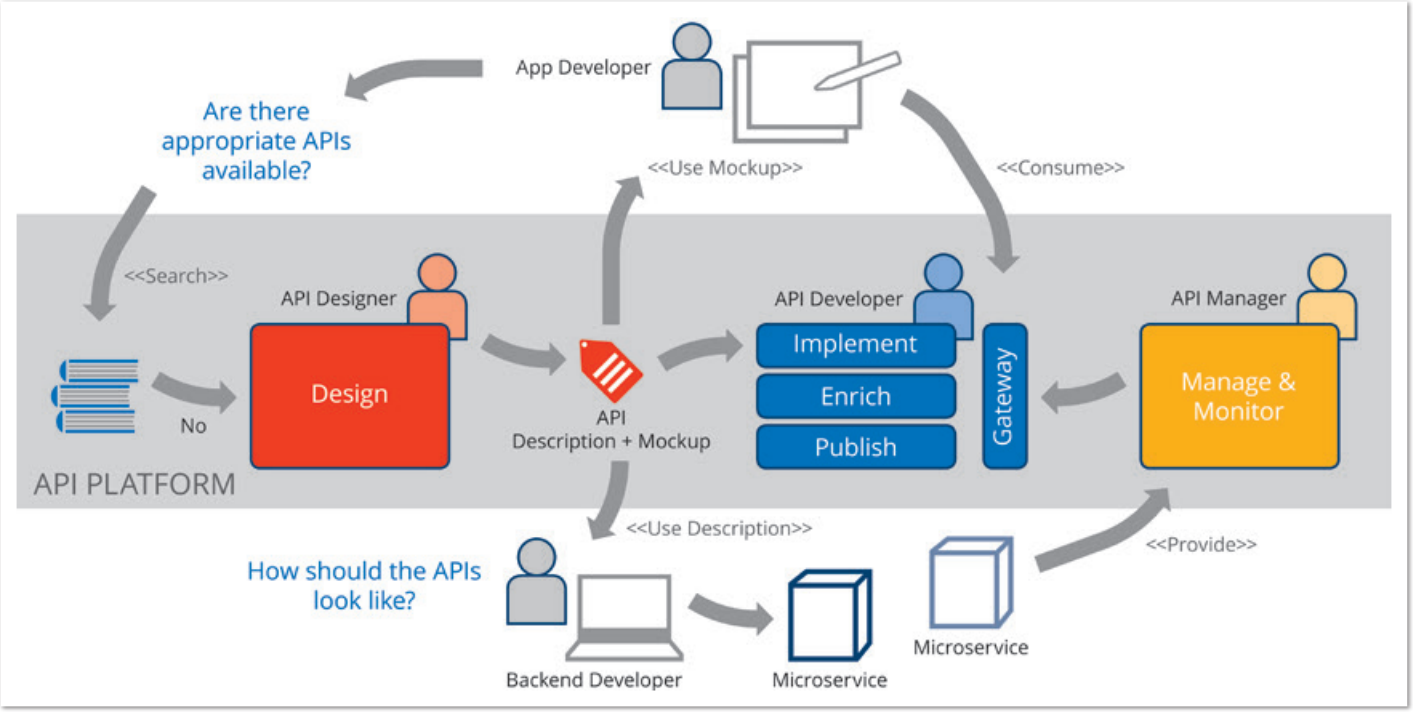


Abbildung 3: „API first"-Design-Ansatz

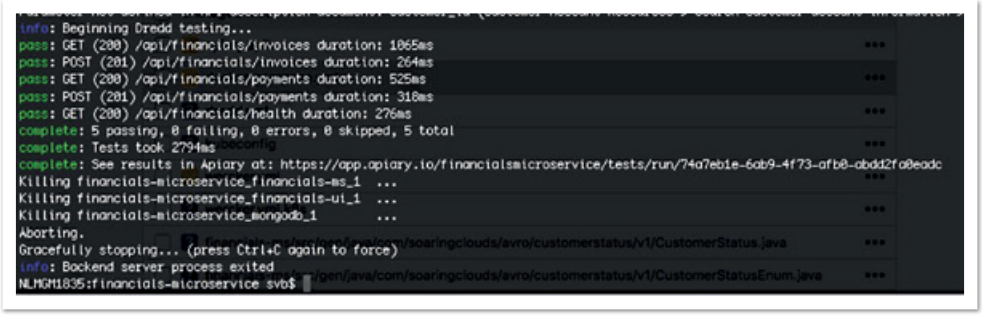


Abbildung 4: Dredd-Testreport

kann. Die lokale Installation des Frameworks erfolgt über NPM. Im Anschluss daran können per „dredd init"-Kommando eine „dredd.yml" -Datei erzeugt und darin die Test-Details für das entsprechende API definiert werden, etwa der HTTP-Endpunkt des zu testenden API. Output ist dann ein entsprechender Test-Report auf der Kommandozeile. Darüber hinaus können die Test-Ergebnisse auch in der Apiary Console eingesehen werden *(siehe Abbildung 4)*.

Build- und Delivery-Automatisierung

Als Build- und Delivery-Plattform kam bei der Webshop-Lösung Oracle Wercker *[9]* zum Einsatz, eine Automatisierungsplattform für den Build und die Bereitstellung von Microservice- und Container-basierten Applikationen. Um Wercker für den Build und das Deployment einer Container-basierten Applikation nutzen zu können, muss die Plattform mit dem GitHub-Repository der zu bauenden Applikation verbunden sein. Im Anschluss daran kann der Build und Delivery Workflow über eine Sequenz sogenannter „Pipelines" definiert werden.

Was innerhalb der einzelnen Pipelines passiert, wird in einer YAML-Datei definiert, der „wercker.yml", die im Git-Repository abzulegen ist. *Abbildung 5* zeigt einen Workflow, der zunächst den Container baut, diesen dann in einer Registry registriert und ihn im Anschluss einrichtet. Darüber hinaus können noch weitere Pipelines in den Workflow aufgenommen werden, zum Beispiel um die Dredd-Tests auszuführen, bevor der Service ausgerollt wird.

Fazit

Wie der Artikel zeigt, sind die Herausforderungen bei der Entwicklung von Microservices-basierten Applikationen mannigfaltig. Eine zentrale Rolle spielen APIs, um die externe Kommunikation abzubilden, sowie ein konsistenter DevOps-Ansatz. Die Vorgehensweise bei der Entwicklung ist ein entscheidender Faktor; der „API first"-Ansatz hilft dabei, die Entwicklung möglichst effizient zu gestalten.

Anhand einer beispielhaften Webshop-Entwicklung wurde skizziert,

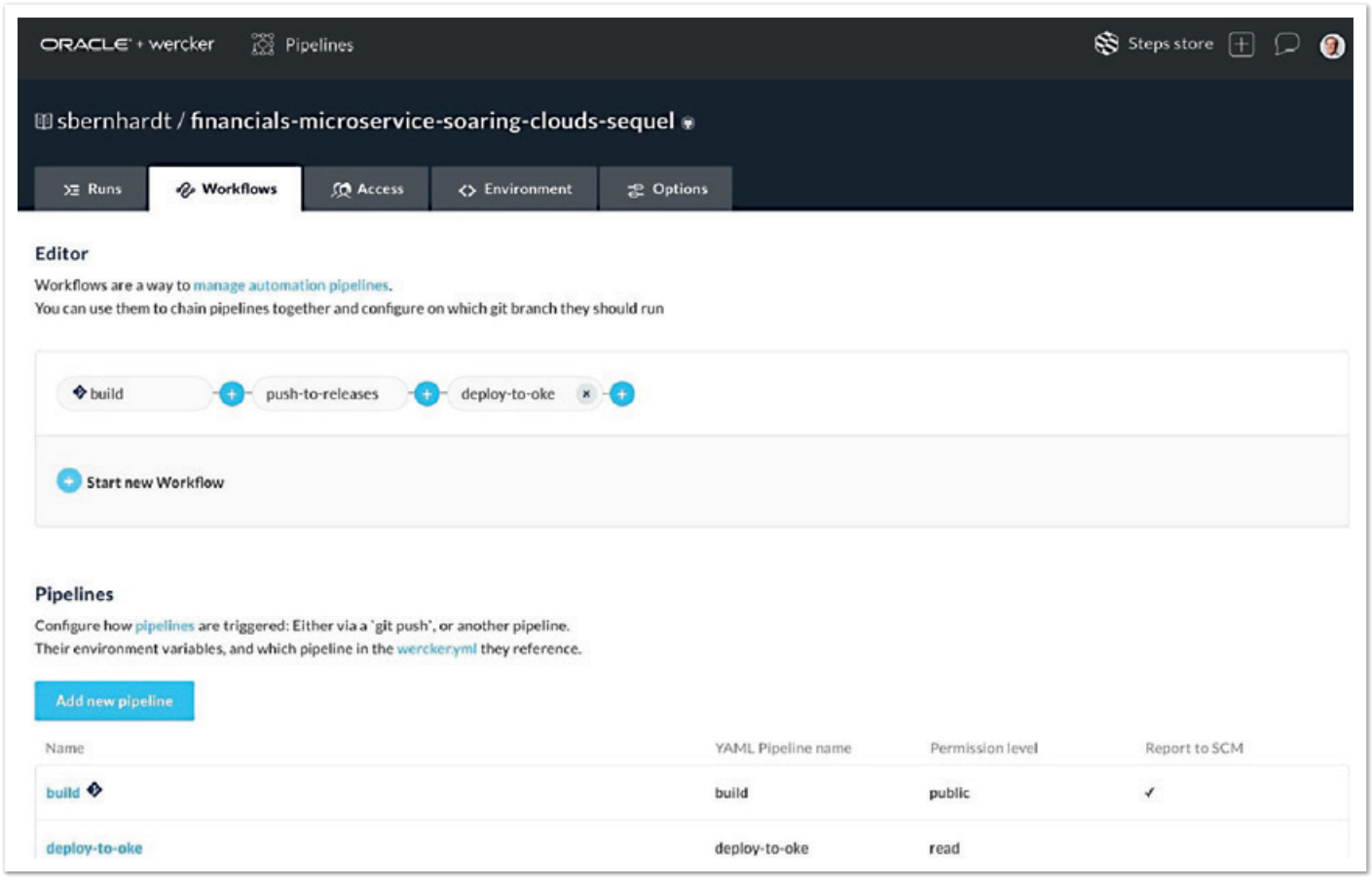


Abbildung 5: Wercker Workflow

worauf es bei der Umsetzung moderner Applikationen in Bezug auf Microservices ankommt. Da die Teammitglieder über fünf verschiedene Länder verteilt waren, konnten wir zudem einen ersten Eindruck bezüglich der organisatorischen Aspekte bekommen, die mit einer Microservices-Implementierung einhergehen. Die Team-Kommunikation lief teilweise über Slack; im Laufe des Projekts, das sich über zwei Monate erstreckte, wurden mehr als dreitausend Nachrichten ausgetauscht! Ein Indikator dafür, wie groß der Abstimmungsbedarf selbst in einem solch kleinen Use Case sein kann.

Weitere Herausforderungen technischer Natur werden uns zukünftig noch weiter beschäftigen. Eine Frage, die intensiv diskutiert wurde, war zum Beispiel: „Wie gestalte ich eine Choreographie für die unterschiedlichen Microservices, um einen Bestellprozess abbilden zu können?" Gedanken hierzu finden sich unter *[10]*.

Quellen

[1] James Lewis, Martin Fowler: „Microservices – A Definition of this New Architectural Term", 2014: *https://www.martinfowler.com/articles/microservices.html*

[2] OMESA Group: „Open Modern Software Architecture Project", OMESA Website, 2017: *http://omesa.io*

[3] OMESA Group: „Capabilities Service Implementation", OMESA Website, 2017: *http://omesa.io/serviceimplementation*

[4] OMESA Group: „Capabilities API", OMESA Website, 2017: *http://omesa.io/apilayer*

[5] Sam Newman: „Pattern: Backends For Frontends", Blog des Autors, 11/2015: *https://samnewman.io/patterns/architectural/bff*

[6] Anne van Kesteren: „Cross-Origin Resource Sharing", W3C Recommendation, 1/2014: *https://www.w3.org/TR/cors*

[7] Apache Software Foundation: „Welcome to Apache Avro!", Projektdokumentation, 2012: *https://avro.apache.org*

[8] „Dredd – HTTP API Testing Framework", Projektdokumentation 5/18, *http://dredd.readthedocs.io/en/latest*

[9] Oracle + Wercker: „Increase developer velocity …", Oracle 2018: *http://www.wercker.com*

[10] Luis Weir: „Is BPM Dead, Long Live Microservices?", Blog des Autors, 2/2018: *http://www.soa4u.co.uk/2018/02/is-bpm-dead-long-live-microservices.html*

Sven Bernhardt
sven.bernhardt@opitz-consulting.com

Sven Bernhardt ist Software-Architekt mit mehr als zehn Jahren Erfahrung in Planung, Design, Implementierung und Einsatz individueller Software-Lösungen für verschiedene Kunden in verschiedenen Branchen. Derzeit arbeitet er als Senior Solution Architect für die OPITZ CONSULTING Deutschland GmbH. Neben seiner Tätigkeit in Kunden-Projekten ist er regelmäßig als Referent auf zahlreichen IT-Konferenzen unterwegs und als Autor von Blogs sowie Artikeln in diversen Fachzeitschriften tätig. Sven Bernhardt ist ein aktives Mitglied der Oracle Developer Community und hat derzeit den Status eines Oracle ACE (Acknowledged Community Expert).

Java-Programme in der Oracle-Datenbank? Na klar!

Matthias Schulz, Schulz IT Services GmbH

Stored Procedures bieten eine Reihe bekannter Vorteile, aber warum sollte man Java dafür einsetzen? Der Artikel zeigt, welche Eigenschaften Java Stored Procedures haben und wie Java-Programme in die Oracle-Datenbank kommen.

Stored Procedures bieten bei datennahen Prozessen eine bessere Kapselung, höhere Datensicherheit, Trennung von Data- und Business-Logik, kürzere Programmausführungszeiten, geringere Netzwerklast und vieles mehr. In Oracle-Datenbanken sind sie meistens in der Sprache PL/SQL geschrieben.

Java Stored Procedures, also in der Datenbank gespeicherte und ausführbare Java-Programme, eignen sich zur Lösung vielfältigster Aufgaben. Das Spektrum reicht von einfachen Einzel-Klassen (wie Filehandling, Mailversand, Datei-Zipper etc.) bis hin zu großen komplexen Systemen (wie ETL-System, Workflow, Datamart-Beladung etc.).

Der Einsatz von Java zum Erstellen von Stored Procedures ermöglicht die Nutzung der unzähligen Bibliotheken und Frameworks, die für Java verfügbar sind. So bieten bereits die Bordmittel von Java einen deutlich besseren Zugriff auf Filesystem und andere Server, als dies mit PL/SQL alleine möglich wäre. Sollen weitere Frameworks und Bibliotheken verwendet werden, so lassen sich deren „.jar"-Files mit dem Tool „loadjava" (siehe unten) direkt in die Oracle-Datenbank laden.

Nicht zuletzt ermöglichen Java Stored Procedures die Wiederverwendung von bereites vorhandenen Programmen sowie Know-how und erlauben es, Programme zu erstellen, die im Gegensatz zu PL/

SQL unabhängig vom Datenbank-Hersteller sind und auch in anderen Datenbank-Systemen als Oracle eingesetzt werden können.

Sicherheit

Wie sicher und leistungsfähig sind Java Stored Procedures? Die nationalen Behörden der USA und Großbritanniens verwenden diese zur Abwicklung der Einreisekontrollen. Da ein Ausfall dieser Systeme jegliche Ein- und Ausreise zum Erliegen brächte, wurde eine Technologie gewählt, die Sicherheit, Stabilität, hohe Performance und langfristige Produktverfügbarkeit gewährleistet.

Ein einfaches Beispiel

Java-Klassen können mit dem Befehl „CREATE JAVA SOURCE" direkt als Source-Code in die Oracle-Datenbank geladen werden, ähnlich wie ein PL/SQL-Source. Die Option „AND RESOLVE" kompiliert den Source-Code und erstellt ein Java Class Object. Es kann nahezu jede Java-SE-Klasse erstellt werden, die Verwendung von Grafik- oder UI-Methoden ist jedoch nicht möglich *(siehe Listing 1)*.

Ein PL/SQL-Wrapper ermöglicht den Zugriff auf die Methoden von Java-Klassen. Jede „static"-Java-Methode kann über eine passende PL/SQL-Funktion oder Prozedur aufgerufen werden, deren Rückgabewert, Parameter-Anzahl und -Datentyp kompatibel sind *(siehe Listing 2)*.

Um die Konsolenausgabe des Beispiels sichtbar zu machen, müssen sowohl die Konsolenausgabe aktiviert als auch die Ausgabe der Java-Konsole umgeleitet werden *(siehe Listing 3)*. *Listing 4* zeigt den Aufruf der Java-Klasse mit dem PL/SQL-Wrapper. Mit „DROP JAVA SOURCE "TestClass";" löscht man das Java-Source-Objekt und die zugehörige Java-Klasse wieder aus der Datenbank.

Das „loadjava"-Tool

Mit „loadjava" *(siehe „https://docs.oracle.com/database/122/JJDEV/loadjava-tool.htm")* werden einzelne Files (Java-Source, Java-Class, Resource-File etc.) und ganze „.jar"-Files in die Oracle-Datenbank geladen. Beim „.jar"-File wird der komplette Inhalt als Objekte des aktuellen Users angelegt, wobei Package-Strukturen erhalten bleiben. Das Tool ist ein Bestandteil des Oracle-Clients, nicht jedoch des Instant-Clients. Es ist zu beachten, dass Pfad-Angaben vor dem zu ladenden Datei-Namen als Package-Informationen mit in der Datenbank abgelegt werden *(siehe Listing 5)*.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE
NAMED"TestClass" AS
public class TestClass {
  public static void hello (String name) {
     System.out.println ("Hello " + name + "!");
  }
}
/
```

Listing 1

```
CREATE OR REPLACE PROCEDURE test_java_hello(name VAR-
CHAR2)
AS LANGUAGE JAVA NAME 'TestClass.hello(java.lang.
String)';
/
```

Listing 2

Die verwendeten Parameter haben folgende Bedeutung:

- *example.jar*
  Name des „.jar"-Files
- *thin*
  JDBC-Thin-Client verwenden
- *force*
  Vorhandene Klassen überschreiben
- *resolve*
  Sourcen sofort kompilieren
- *verbose*
  Detaillierte Ausgaben
- *user*
  Datenbank-Verbindung: Username/Passwort@Datenbank

Die Performance

Sind Java Stored Procedures schneller als Java-Programme, die außerhalb der Datenbank laufen? Ein einfaches Testprogramm, das sowohl stand-alone als auch in der Datenbank läuft, beantwortet diese Frage. Es soll folgende Funktionen ausführen:

1. Datenbank-Verbindung aufbauen
2. Tabelle „BASIC_LOB_TABLE", falls vorhanden, entfernen und erstellen
3. Einfügen von zwei Datensätzen
4. Selektieren der Datensätze und Abarbeiten des Cursors in einer Schleife
5. Lesen der Clob- und Blob-Felder der Tabelle und Ermitteln der Größen
6. Truncate auf das Clob- und das Blob-Objekt und Ermitteln der Größen

Tabelle 1 und *Abbildung 1* zeigen das Ergebnis. Betrachtet man nur die Laufzeit der DML-Operationen (Schritte 3 bis 6), so ergibt sich bei den hier verwendeten Systemen eine um den Faktor 12 schnellere Programmausführung bei der Java Stored Procedure.

Ein primärer Vorteil von Java Stored Procedures und Stored Proce-

```
-- Konsolenausgabe aktivieren:
set serveroutput on size 1000000 ;

-- Java-Ausgaben auf die Konsole umleiten:
exec dbms_java.set_output(1000000) ;
```

Listing 3

```
BEGIN
  test_java_hello('DOAG');
END;
/
```

Listing 4

```
D:\Oracle\product\12.1.0\client_1\bin\loadjava.bat
example.jar -thin -force -resolve -verbose -user MY_
USER/my_password@MYDB
```

Listing 5

Lauf	Stand-alone	JServer	schneller
1	80 ms	10 ms	8 mal
2	94 ms	8 ms	12 mal
3	93 ms	8 ms	12 mal
4	93 ms	8 ms	12 mal
5	93 ms	7 ms	13 mal

Tabelle 1: Vergleich der Laufzeiten – DML-Operationen (Schritte 3 bis 6)

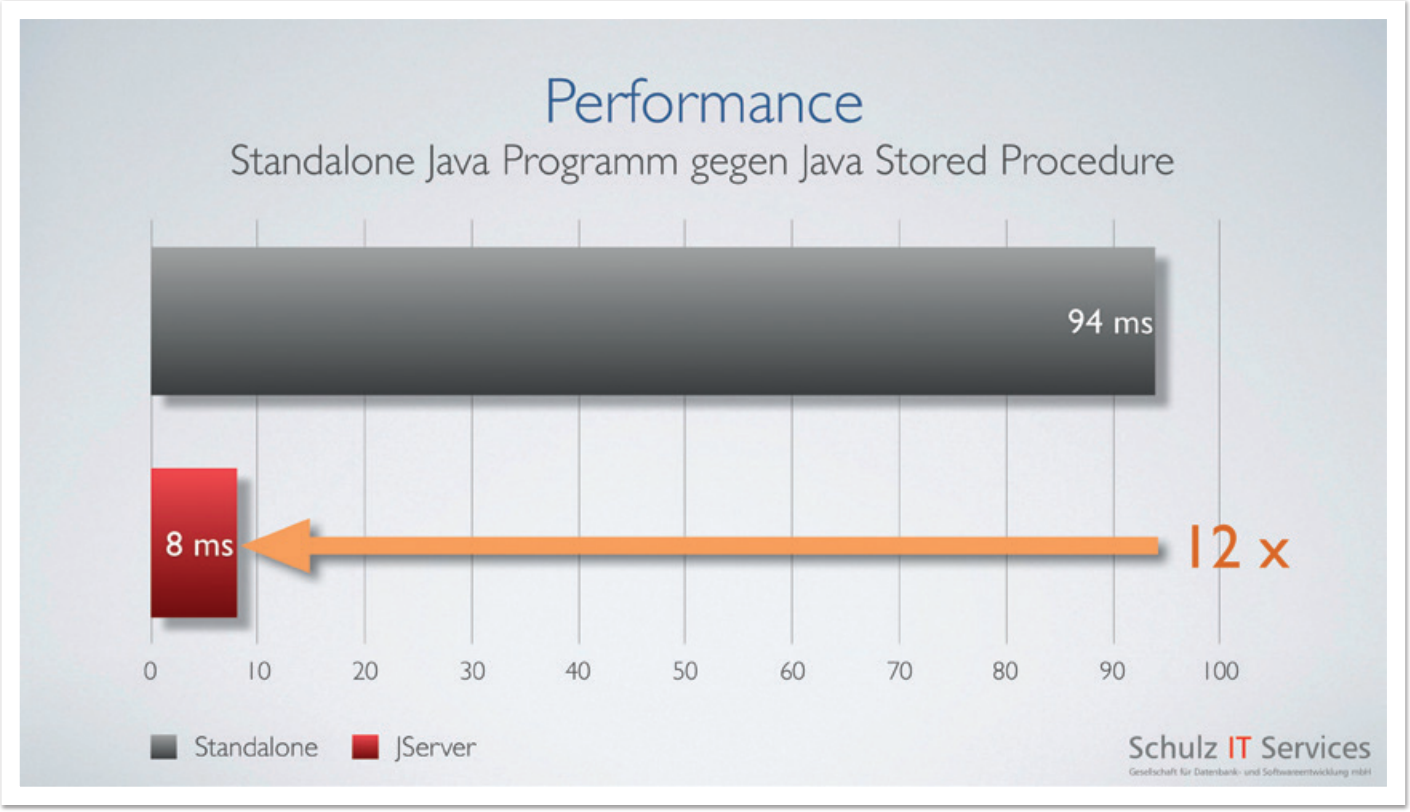


Abbildung 1: Performance – Java-Stand-alone-Programm gegen Java Stored Procedure

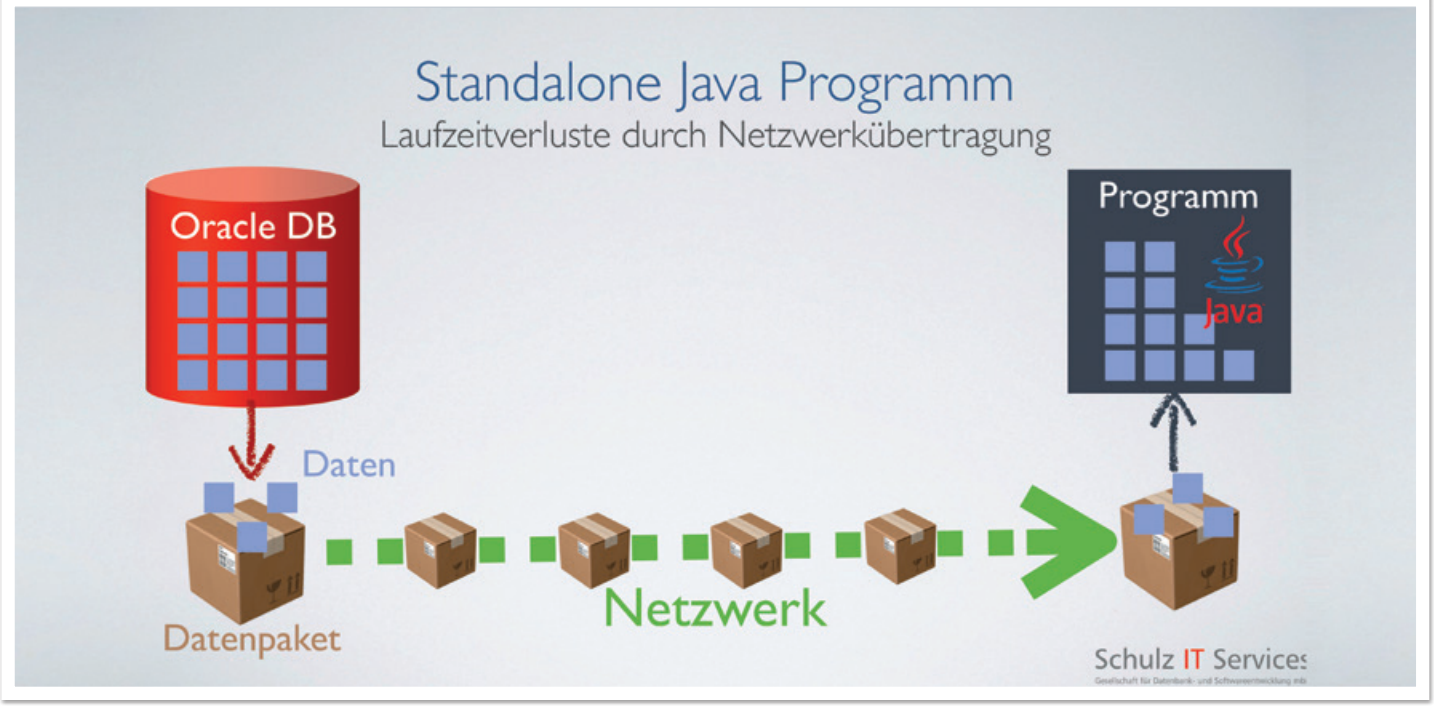


Abbildung 2: Laufzeit-Verluste durch Netzwerk-Übertragung

dures im Allgemeinen ist der Wegfall des Umwegs der Daten über das Netzwerk. Einpacken, Übertragen und Wiederauspacken der Daten erzeugt einen beachtlichen Zeitverlust, der hier einfach wegfällt *(siehe Abbildung 2)*. Der Source-Code für das Testprogramm steht unter *„https://oracledeli.wordpress.com/2017/12/23/java-stored-procedures-performance-test".*

Java-Versionen in der Oracle-Datenbank

Oracle liefert seit der Datenbank-Version 8.1.5 eine integrierte Java Virtual Machine (JVM) namens „JServer" mit aus *(siehe Tabelle 2)*. Die View „SELECT * FROM all_registry_banners;" gibt an, ob und wenn ja, welche Version des JVM installiert ist. Die installierte Java-Version erhält man mit „SELECT dbms_java.get_ojvm_property(propstring=>'java.version') AS java_version FROM DUAL;".

Für den Zugriff auf die eigene Datenbank existiert eine immer geöffnete JDBC-Verbindung „Connection con = DriverManager.getConnection(" jdbc:default:connection:");". Diese sollte nicht geschlossen werden. Auf andere Datenbanken kann, wie gewohnt, via JDBC zugegriffen werden.

Stand-alone oder JServer?

Um festzustellen, ob das Programm innerhalb einer Oracle-Datenbank (JServer) läuft, lässt sich mit „System.getProperty(„oracle.jserver.version")" eine System-Property abfragen. Liefert die Methode einen Wert zurück, läuft das Programm innerhalb einer Oracle-Datenbank, ansonsten wird „null" zurückgegeben *(siehe Listing 6)*.

Session und Multi-Threading

Alle Java Stored Procedures laufen innerhalb einer Oracle-Datenbank-Session und sind völlig voneinander isoliert, als ob jedes Programm eine eigene separate JVM und einen eigenen Garbage Collector hätte. Um Daten zwischen Programmen auszutauschen, sollten daher Alternativen wie Tabellen oder Messages (Oracle Advanced Queuing) verwendet werden. Multi-Threading-

Oracle-Datenbank Version	Java-Version Standard	Java-Version unterstützt
11g	JRE 1.5	-
11g, ab 11.2.0.4	JDK 6	-
12c	JDK 6	JDK 7
12c, ab 12.2.0.1	Java 8	Nashorn (JavaScript Engine)

Tabelle 2: Java-Versionen in der Oracle-Datenbank

```
if (System.getProperty("oracle.jserver.version") != null){
  /* program runs inside the Oracle database */
}
else{
  /* program runs as standalone */
}
```

Listing 6

```
SELECT dbms_java.longname(o.object_name) as long_ob-
ject_name,
        o.*
    FROM USER_OBJECTS o
  WHERE o.object_type LIKE 'JAVA%'
  ORDER BY dbms_java.longname(o.object_name);
```

Listing 7

```
SELECT dbms_java.longname(e.name) as long_object_name,
        e.*
    FROM USER_ERRORS e
  WHERE e.TYPE LIKE 'JAVA%'
  ORDER BY dbms_java.longname(e.name);
```

Listing 8

View	Description
USER_JAVA_ARGUMENTS	argument information of stored java class owned by the user
USER_JAVA_CLASSES	class level information of stored java class owned by the user
USER_JAVA_COMPILER_OPTIONS	native compiler options provided by the user
USER_JAVA_DERIVATIONS	this view maps java source objects and their derived java class objects and java resource objects for the java class owned by user
USER_JAVA_FIELDS	field information of stored java class owned by the user
USER_JAVA_IMPLEMENTS	interfaces implemented by the stored java class owned by user
USER_JAVA_INNERS	list of inner classes refered by the stored java class owned by user
USER_JAVA_ARGUMENTS	argument information of stored java class owned by the user
USER_JAVA_CLASSES	class level information of stored java class owned by the user
USER_JAVA_COMPILER_OPTIONS	native compiler options provided by the user
USER_JAVA_DERIVATIONS	this view maps java source objects and their derived java class objects and java resource objects for the java class owned by user
USER_JAVA_FIELDS	field information of stored java class owned by the user
USER_JAVA_IMPLEMENTS	interfaces implemented by the stored java class owned by user
USER_JAVA_INNERS	list of inner classes refered by the stored java class owned by user

Tabelle 3: Views für Java-Objekte im Data Dictionary

Programme sind zwar lauffähig, verwenden aber nur eine CPU. Sind mehrere Threads erforderlich, lässt sich dies durch Oracle-Scheduler-Jobs realisieren.

Data Dictionary

Wie bei Oracle üblich, finden sich auch sämtliche Java-Objekte im Data Dictionary *(siehe Tabelle 3)*. Die View „USER_OBJECTS“ ent-

```
SET SERVEROUTPUT ON ;

DECLARE
     p_resourcefile_name VARCHAR2(4000 CHAR) := 'messagetransmitter.properties';
     l_content           CLOB;
     l_max_chars         INTEGER := 32767;
     l_amount            INTEGER := l_max_chars;
     l_text              VARCHAR2(32767 CHAR);
BEGIN
     DBMS_LOB.createtemporary(l_content, FALSE);
     DBMS_JAVA.EXPORT_RESOURCE(p_resourcefile_name, l_content);
     DBMS_LOB.READ(l_content, l_amount, 1, l_text);
     DBMS_OUTPUT.put_line(l_text);
     IF l_amount >= l_max_chars THEN
        DBMS_OUTPUT.put_line('[...]');
     END IF;
END;
/
```

Listing 9

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED  "SocketTest " AS
import java.net.Socket;
public final class SocketTest
{
  public static void connectTest(final String pHost, final int pPort)
  {
     Socket socket;
     System.out.println( "testSourceSocketConnection: host:  " + pHost
               +  ", port: " + pPort);
     try {
        socket = new Socket(pHost, pPort);

        if (socket.isConnected()) {
           System.out.println( "socket.isConnected() = true\n ");
        }
        else {
           System.out.println( "socket.isConnected() = false !!!\n ");
        }
        socket.close();
     }
     catch (Exception e) {e.printStackTrace();}
  }
}
/
```

Listing 10

```
CREATE OR REPLACE PROCEDURE
  SOCKET_TEST(p_host VARCHAR2, p_port NUMBER)
AS LANGUAGE JAVA
  NAME 'SocketTest.connectTest(java.lang.String, int)';
/
```

Listing 11

```
SET SERVEROUTPUT ON ;
BEGIN
  DBMS_JAVA.set_output(1000000);
  SOCKET_TEST('myserver.mycompany.com', 61616);
END;
/
```

Listing 12

```
testSourceSocketConnection: host: myserver.mycompany.com, port:123
java.security.AccessControlException: the Permission (java.net.SocketPermission myserver.mycompany.com resolve) has
not been granted to DISTRIBUTOR_PROCESSING.
The PL/SQL to grant this is
dbms_java.grant_permission( 'DISTRIBUTOR_PROCESSING', 'SYS:java.net.SocketPermission', 'myserver.mycompany.com',
'resolve' )
  at java.security.AccessControlContext...
```

Listing 13

```
dbms_java.grant_permission( 'DISTRIBUTOR_PROCESSING', 'SYS:java.net.SocketPermission', 'myserver.mycompany.com',
'resolve' )
```

Listing 14

Thema	PL/SQL	Java	Vorteil
SQL-Befehle	•		Keine zusätzliche JDBC-Schicht
SQL-Datentypen	•		Keine Umwandlungen
Berechnungen		•	Schneller
Moderne Sprachelemente		•	Vererbung, Closures etc.
Filehandling		•	Mehr Möglichkeiten/Frameworks
Mailversand		•	Mehr Möglichkeiten/Frameworks
Zugriff auf Web-Server		•	Mehr Möglichkeiten/Frameworks
Zugriff auf andere Systeme		•	JDBC, REST etc.

Tabelle 4: Java oder PL/SQL?

hält auch alle Java-Objekte, erkennbar an einem mit „JAVA" beginnenden Object Type. Die Namen der Java-Objekte werden verkürzt im Data Dictionary abgelegt, jedoch kann mit der Funktion „dbms_java.longname" wieder der volle Name angezeigt werden *(siehe Listing 7)*. Fehlerhafte Java-Objekte finden sich in der View „USER_ERRORS" *(siehe Listing 8)*.

Um den Inhalt eines Java-Objekts, etwa den eines Resource-Files, auszugeben, dient das Skript in *Listing 9*, hier für das File „message-transmitter.properties".

Berechtigungen

Oracle sichert Java Stored Procedures sorgsam ab. Das bringt in der Praxis zunächst einige Beschränkungen mit sich und erfordert beispielsweise für Zugriffe auf das Dateisystem oder einen fremden Server die Vergabe expliziter Rechte. Dazu ein Beispiel: „SocketTest" ist ein einfaches Programm, um zu testen, ob eine Verbindung zu einem bestimmten Server mit einem bestimmten Port aufgebaut werden kann *(siehe Listing 10)*. *Listing 11* zeigt den PL/SQL-Wrapper. Wir testen nun den Zugriff auf einen Server. Die Werte für Host *(„myserver.mycompany.com")* und Port (61616) sollten durch Daten eines realen Servers ersetzt werden *(siehe Listing 12)*. *Listing 13* zeigt das Ergebnis. Wir erhalten eine Fehlermeldung, weil das Programm (noch) keine Java-Permission besitzt, um auf den gewünschten Server (hier *„myserver.mycompany.com")* zuzugreifen. Die Vergabe der fehlenden Rechte gestaltet sich bei Java Stored Procedures jedoch recht einfach, da mit der Fehlermeldung bereits ein fertiger Befehl zur Vergabe der fehlenden Rechte mitgeliefert wird *(siehe Listing 14)*.

Java oder PL/SQL?

Wann sich welche der beiden Sprachen besser eignet, hängt von verschiedenen Faktoren ab und bedarf einer individuellen Abwägung – je nach Aufgabe eignet sich entweder PL/SQL oder Java besser *(siehe Tabelle 4)*. Letztendlich sollte in die Entscheidung auch das Know-how der Entwickler, die Verfügbarkeit von entsprechenden Entwicklern im Markt sowie die Wiederverwendbarkeit der Programme eine Berücksichtigung finden. Bleibt abschließend nur noch die Frage, ob man nun Java oder PL/SQL zum Erstellen von Stored Procedures einsetzen sollte. Die beste Antwort lautet: PL/SQL und Java, je nach Aufgabe.

Weiterführende Links

- Performance Test Source Code: *https://oracledeli.wordpress.com/2017/12/23/java-stored-procedures-performance-test*
- Database Java Developer's Guide: *https://docs.oracle.com/database/122/JJDEV/toc.htm*
- The loadjava Tool: *https://docs.oracle.com/database/122/JJDEV/loadjava-tool.htm*
- DBMS_JAVA Package: *https://docs.oracle.com/database/122/JJDEV/DBMS-JAVA-package.htm*

Matthias Schulz
schulz@schulz-it-services.de

Matthias Schulz ist seit dem Jahr 2002 Geschäftsführer der Schulz IT Services GmbH und dort als Consultant, Trainer und Konferenzsprecher im Datenbank-Umfeld tätig. Seine Schwerpunkte sind Data Vault, High-End-Systeme (Exadata, Exasol) und die Oracle-Datenbank (Entwicklung, Tuning, Architektur, ETL, Testing und Java in der Datenbank). Zudem ist er Dozent für Informatik (Datenbanken) an der Dualen Hochschule Baden-Württemberg.

Einführung in RxJS

Michael Ruttka, buschmais GbR

Den ersten Kontakt mit den sogenannten „Reactive Extensions for JavaScript“ (RxJS, siehe „http://reactivex.io/rxjs“) hat der Autor beim Umstieg von Angular.js auf Angular 2 gemacht. Wurden in Angular.js asynchrone Prozesse noch mit Promises abgebildet, so hat man sich in Angular 2 für Observables von RxJS entschieden. Es war also an der Zeit, mal einen genaueren Blick auf dieses Framework zu werfen. Ziel dieses Artikels ist es, die grundlegenden Konzepte von RxJS kennenzulernen und eine solide Wissensbasis zu erlangen, um seine ersten Schritte mit RxJS zu wagen.

Promises und Observables sind eine Art Platzhalter für noch nicht bekannte Ergebnisse. Ein einfaches Beispiel stellt die HTTP-Schnittstelle dar. Sendet der Client eine HTTP-Anfrage, ist deren Ergebnis erst nach der Antwort des Servers bekannt. Der Aufrufer erhält von der Schnittstelle einen solchen Platzhalter, kann auf dessen Ergebnis lauschen und unterdessen beispielsweise eine Lade-Animation anzeigen.

Was aber ist an Observables anders als an Promises? Der entscheidende Unterschied besteht darin, dass Observables über die Möglichkeit verfügen, abgebrochen werden zu können und mehr als nur ein Ergebnis zu verarbeiten. Dies kann beispielsweise bei der Event-Behandlung in einem Client-Framework wie Angular sehr von Vorteil sein.

Einrichtung

Bevor man mit RxJS loslegen kann, muss es zunächst einmal eingerichtet sein. In diesem Artikel wird „npm“ verwendet, um die Abhängigkeiten zu verwalten. Also wird auf der Kommandozeile „npm install rxjs –save“ ausgeführt, um RxJS in der neuesten Version in die „package.json“ aufzunehmen.

Da in diesem Artikel TypeScript zum Einsatz kommt, ist außerdem

noch der TypeScript-Compiler erforderlich. Er wird mit „npm install typescript -g" global auf unserer Entwicklungsmaschine installiert. Das Code-Beispiel in *Listing 1* zeigt, ob die Installation erfolgreich war.

Um den TypeScript-Code ausführen zu können, muss er zuerst mit „tsc hello-rxjs.ts" vom TypeScript-Compiler in JavaScript kompiliert werden. Danach sollte im Dateisystem eine Datei „hello-rxjs.js" auftauchen. Diese kann nun mit „node hello-rxjs.js" von Node.js ausgeführt werden. Es erscheint „hello RxJS" auf der Kommandozeile.

In diesem kurzen Beispiel ist bereits ein erster Fallstrick versteckt. Es gibt zwei Varianten, wie RxJS importiert werden kann. Innerhalb dieses Artikels wird ausschließlich die einfachere Variante, das „Komplettpaket", verwendet. Die Alternative bindet nur die verwendeten Komponenten in die Anwendung ein. Dies empfiehlt sich vor allem dann, wenn es auf die Auslieferungsgröße ankommt. In dieser Variante sind auch alle verwendeten Operatoren (dazu später mehr) einzeln zu importieren.

Observables

Das Herz von RxJS bilden die Observables. Ein Observable ist die Repräsentation einer beliebigen Menge von Werten, die über eine beliebige Zeitdauer verteilt sein können. Mit einer Subscription kann man diese Werte beobachten. Werte sind oft auch Ereignisse (etwa ein Mausklick oder eine Navigation), können aber auch Werte (wie 1, 2, 3, A, B,C) im engeren Sinne sein.

Das einführende Beispiel zeigt bereits ein solches Observable. Die dort verwendete Klasse „Subject" implementiert das Interface „Observable" und erweitert dieses um zusätzliche Methoden (wie „next") zur Auslösung des Observable. Aufgrund dieser Möglichkeit eignet sich Subject zur einfachen und nachvollziehbaren Demonstration der Funktionsweise von RxJS.

Die Klasse „Observable" besitzt die Methode „subscribe", mit deren Hilfe man eine Subscription erhält. Observables haben immer einen Zustand. Solange keine Subscription für ein Observable aktiv ist, hat es den Zustand „cold". In diesem Zustand interessiert sich niemand für die Werte des Observable, also werden sie auch nicht verarbeitet.

Mit dem Aufruf von „subscribe" und der damit verbundenen Erzeugung einer Subscription geht das Observable in den Zustand „hot" über. Die Werte des Observable werden nun überwacht und verarbeitet. Werte, die vor der Subscription in das Observable gereicht wurden, wird der neue Subscriber nie erfahren. Das Beispiel in *Listing 2* würde also lediglich die Werte „2" und „3" ausgeben.

Da zum Zeitpunkt der Subscription der Wert „1" bereits gesendet wurde, wird die Subscription lediglich über die nachfolgenden Werte benachrichtigt. Das Beispiel zeigt außerdem, dass die an „subscribe" übergebene „Arrow"-Funktion jeweils einmal pro Wert des Subject ausgeführt wird. Doch die Methode „subscribe" verfügt noch über zwei weitere Parameter, um auf Ereignisse des Observable zu reagieren. Das zweite Argument ist ein Callback für Fehler und das dritte Argument wird ausgeführt, wenn das Observable „completed" ist. *Listing 3* zeigt ein Beispiel, das diese beiden Callbacks demonstriert.

Die Ausgabe dieses Code-Schnipsels würde die Werte „1" und „2" sowie einen Fehler liefern, nicht aber die „finished"-Meldung. Das

```
import {Subject} from 'rxjs/Rx';
const subject = new Subject<string>();
subject.subscribe((value) => console.log(`hello
${value}`));
subject.next('RxJs');
```

Listing 1

```
import {Subject} from 'rxjs/Rx';

const subject = new Subject<string>();
subject.next('1');
subject.subscribe((value) => console.log(value));
subject.next('2');
subject.next('3');
```

Listing 2

```
import {Subject} from 'rxjs/Rx';

const subject = new Subject<string>();
subject.subscribe(
  (value) => console.log(value),
  (error) => console.error(error),
  () => console.log('finished'));

subject.next('1');
subject.next('2');
subject.error(new Error('something went wrong'));
subject.complete();
```

Listing 3

liegt daran, dass ein Observable nach einem Fehler nicht mehr beendet werden kann – es endet also entweder in dem Zustand „Error" oder „completed". Würde man also die Zeile weglassen, die den Error auslöst, würde neben „1" und „2" auch die Ausgabe „finished" erscheinen. An dieser Stelle eine weitere Besonderheit: Fehler, die im Value-Callback (erstes Argument der „subscribe"-Methode) auftreten, werden nicht vom Error-Callback behandelt.

Hat man erst einmal eine Subscription in der Hand, ist man auch in der Verantwortung, diese wieder aufzulösen; ansonsten riskiert man ein Speicherleck. Es ist also wichtig sicherzustellen, dass die Methode „unsubscribe" aufgerufen wird, wenn die Werte nicht mehr überwacht werden sollen. In Angular wird hierfür gerne die Lifecycle-Methode „ngOnDestroy" verwendet. Eine Ausnahme bilden Observables, bei denen eine Beendigung sichergestellt werden kann. Eine HTTP-Anfrage gehört beispielsweise zu dieser Klasse von Observables, da sie nach der Antwort durch den Server keine weiteren Ergebnisse liefern können und somit abgeschlossen sind.

Operatoren

Bis jetzt sind Observables noch nicht sonderlich spektakulär; erst mit Einführung der Operatoren werden Observables wirklich interessant. Denn neben dem Unterschied zu Promises, dass mehr als nur ein Wert verarbeitet werden kann, wird dem Entwickler mit den Operatoren eine große Werkzeugkiste in die Hand gegeben. Sie ermöglicht es, die Werte eines Observable zu verändern, sie mit anderen Observables zu vereinen, zu filtern, zu limitieren und noch vieles mehr.

Operatoren werden in aller Regel auf Observables ausgeführt und

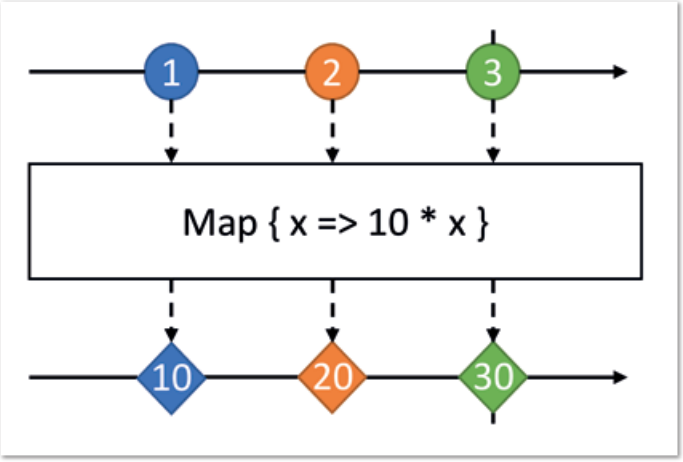


Abbildung 1: Einfaches Marble-Diagramm mit einem Mapping der Werte auf ihr Zehnfaches

geben ein neues Observable zurück. Dadurch können mehrere dieser Aufrufe aneinandergekettet werden. Es entsteht eine Observable-Sequenz, wodurch eine Notation ähnlich wie beim Builder-Pattern entsteht. Nachfolgend stelle ich die verschiedenen Kategorien, in die sich Operatoren aufteilen lassen, mit ihren wichtigsten Vertretern vor.

Marble-Diagramme

Es hat sich etabliert, die Funktionsweise von Observables durch Marble-Diagramme zu veranschaulichen. Ein Marble-Diagramm zeigt für ein einzelnes Observable immer einen Zeitstrahl, auf dem sich bunte „Murmeln" befinden, die die Werte symbolisieren. *Abbildung 1* zeigt ein einfaches Beispiel für ein Marble-Diagramm, das eine Observable-Sequenz darstellt, in der die eingehenden Werte „1:1" in einen anderen Wert konvertiert werden. *Tabelle 1* erklärt die Symbolik von Marble-Diagrammen.

Aus Platzgründen wird in diesem Artikel nicht für jeden hier genannten Operator zusätzlich ein Marble-Diagramm aufgeführt. Eine ausführliche Liste mit Marble-Diagrammen steht auf der Seite *„http://rxmarbles.com"*. Ein Besuch lohnt sich, denn die interaktiven Diagramme erleichtern den Einstieg in RxJS ungemein.

Erstellung

Die folgenden Operatoren erzeugen neue Observables. Sie können in den meisten Fällen statisch auf der Klasse „Observable" aufgerufen werden. „from" erzeugt ein Observable, das jeden Wert eines Arrays einzeln in die Verarbeitungskette reicht. Im obigen Beispiel werden also die Werte 1, 2 und anschließend 3 ausgegeben. Nachdem alle Werte verarbeitet wurden, wird das Observable automatisch „completed" *(siehe Listing 4)*.

„timer/interval" erzeugt ein Observable, das nach einer Verzögerung (etwa 500 ms, *siehe Listing 5)* den Wert „0" in die Sequenz reicht. Wird wie im Beispiel auch der zweite Parameter angegeben, so wird zusätzlich nach dieser Anzahl an Millisekunden je ein weiterer inkrementierter Wert zurückgegeben. Dieses Observable läuft demnach unendlich lang. Es ist also wichtig, dass ein „unsubscribe" die Beendigung dieser Subscription sicherstellt. Das obige Beispiel wird die Werte „0", „1", „2", „3" und so weiter ausgeben, bis die Anwendung beendet ist. Ist lediglich die Funktionalität des zweiten Parameters gewünscht, so kann man dies mit dem Operator „interval" erreichen,

Symbolik	Bedeutung
1	Einzelner ausgestoßener Wert
10	Transformierter Wert in der gleichen Farbe des Quellwerts
	Terminierung der Observable-Sequenz durch einen Fehler (im Beispiel nicht verwendet)
2	Normale Terminierung der Observable-Sequenz ("complete")

Tabelle 1

```
Observable.from([1, 2, 3])
  .subscribe((value) => console.log(value));
```

Listing 4

```
Observable.timer(500, 100)
  .subscribe((value) => console.log(value));
```

Listing 5

```
Observable.from([1, 2, 3])
  .map((x) => x * 10)
  .subscribe((value) => console.log(value));
```

Listing 6

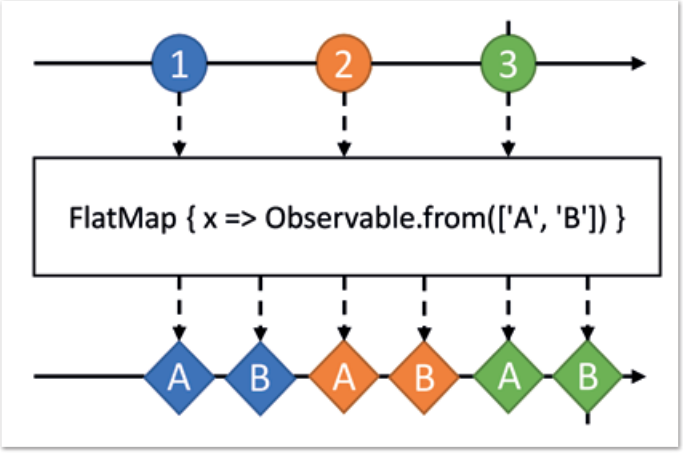


Abbildung 2: Marble-Diagramm zum Codebeispiel des „flatMap"-Operators

```
Observable.from([1, 2, 3])
  .flatMap((value) => Observable.from('A', 'B'))
  .subscribe((value) => console.log(value));
```

Listing 7

```
Observable.timer(500, 100)
  .filter((value) => value % 2 === 0)
  .take(5)
  .subscribe((value) => console.log(value));
```

Listing 8

```
Observable.interval(100)
  .take(5)
  .subscribe((value) => console.log(value));
```

Listing 9

```
const delayed = Observable.timer(1000);

Observable.interval(500)
  .takeUntil(delayed)
  .subscribe((value) => console.log(value));
```

Listing 10

```
const nameObservable = Observable.from(['Jim', 'Johnny',
'Jack']);
const surnameObservable = Observable.from(['Beam',
'Walker', 'Daniels', 'Jameson']);

Observable.zip(nameObservable, surnameObservable)
  .map((values) => `${values[0]} ${values[1]}`)
  .subscribe((value) => console.log(value));
```

Listing 11

der sich abgesehen von dem anfänglichen Delay identisch verhält.

Transformation

Es kommt sehr häufig vor, dass die Werte eines Observable in gänzlich neue Objekte umgewandelt werden müssen. Transformations-Operatoren übernehmen diese Aufgabe und erzeugen basierend auf den Eingangswerten ein oder mehrere neue Objekte für die weitere Sequenz.

„map" transformiert den eingehenden Wert in einen beliebigen anderen Wert. Im *Listing 6* wird der eingehende Wert mit „10" multipliziert. Die Konsole wird also die Werte „10", „20" und „30" anzeigen. Das Marble-Diagramm zu diesem Operator wurde übrigens als Beispiel zur Erklärung von Marble-Diagrammen verwendet.

„flatMap" transformiert einen einfachen Eingangswert in ein Observable, dessen Sequenzwerte anschließend jeweils einzeln in der Sequenz weiterverarbeitet werden *(siehe Abbildung 2)*. Im *Listing 7* wird ein simples Array mit den Werten 1, 2 und 3 mit flatMap aufgerufen. Für jeden dieser Werte wird im flatMap-Callback ein neues Observable zurückgegeben, das die Werte „A" und „B" beinhaltet. Die Ausgabe lautet demnach „A", „B", „A", „B", „A" und „B".

Filter

Filter-Operatoren verringern die Menge an verarbeiteten Werten. Sie können einzelne Werte aus der Sequenz entfernen oder die Sequenz gänzlich beenden. „filter" ist der wohl naheliegendste Operator der Filter-Kategorie. Er begrenzt die Elemente einer Verarbeitungskette anhand einer Funktion, die für jeden Wert einen Wahrheitswert zurückgibt. Ist dieser wahr, wird das Element in der Sequenz weitergereicht; ist er falsch, wird es gefiltert. Im *Listing 8* werden alle ungeraden Werte gefiltert, sodass am Ende nur die geraden Zahlen „0", „2", „4", „6" und „8" ausgegeben werden.

„take" begrenzt die verarbeiteten Werte auf eine übergebene Anzahl. Im *Listing 9* wird das unendliche Intervall, das alle 100 ms einen numerischen Wert liefert, nach fünf Ausführungen beendet. Die Ausgabe ist also auf „0", „1", „2", „3" und „4" begrenzt. Durch diese Begrenzung wird die eigentlich unendliche Subscription endlich und kann so automatisch beendet werden. Der Aufruf von „unsubscribe" ist demnach nicht notwendig, außer es kann nicht sichergestellt werden, dass mindestens fünf Werte durch das Observable erzeugt werden.

Eine andere Variante von „take" stellt „takeUntil" dar, die so lange Werte verarbeitet, bis ein anderes Observable einen Wert liefert. Im *Listing 10* werden also die Werte „0" und „1" nach je 500 ms ausgegeben. Da zu diesem Zeitpunkt eine Sekunde vergangen ist, liefert das zweite Observable einen Wert „0" und beendet somit die Sequenz. Beide Subscriptions sind dann automatisch beendet.

Kombination

Hat man RxJS erst einmal im Einsatz, wird auch schnell die Anforderung entstehen, die Werte zweier Observables zu mischen und in einer gemeinsamen Sequenz zu verarbeiten. Die folgenden beiden Operatoren ermöglichen ein solches Zusammenführen zweier Observables.

Möchte man die Werte von zwei Observables „1:1" miteinander kombinieren, so greift man zum „zip"-Operator. Dieser kann beliebig viele Observables entgegennehmen. Es wartet so lange, bis von allen ein Wert eingetroffen ist, und reicht diese dann in einem Array weiter.

Im *Listing 11* wird „zip" verwendet, um ein Observable von Vornamen mit einem Observable von Nachnamen zu kombinieren. Die Ausgaben sind demnach „Jim Beam", „Johnny Walker" und „Jack

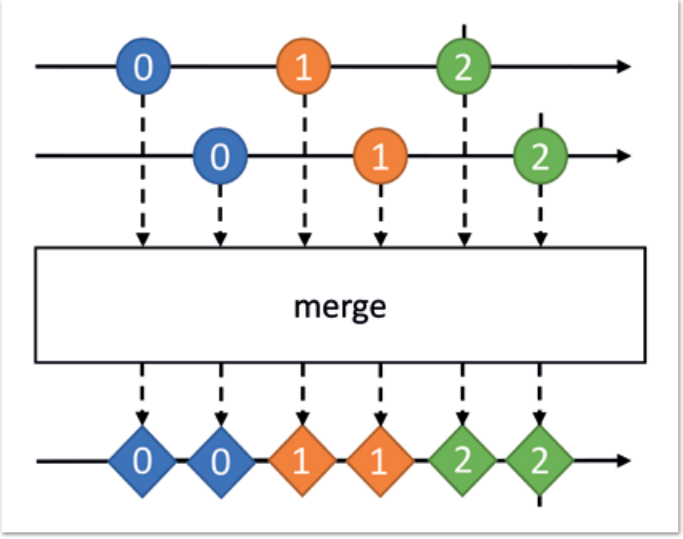


Abbildung 3: Marble-Diagramm zum Codebeispiel des „merge"-Operators

```
Observable.interval(150)
  .merge(Observable.interval(200))
  .take(6)
  .subscribe((value) => console.log(value));
```

Listing 12

```
Observable.from(['Jim', 'Johnny', 'Jack'])
  .do((name) => console.log(name))
  .subscribe();
```

Listing 13

```
Observable.interval(100)
  .delay(300)
  .take(5)
  .subscribe((value) => console.log(value));
```

Listing 14

Daniels". „Jameson" wird nicht ausgegeben, da hier der Vorname fehlt.

„merge" erweitert die Werte eines Observable um die Werte eines anderen. Im Gegensatz zu „zip" wartet „merge" nicht auf die Werte des anderen Observable; die Werte werden sofort weitergereicht, egal von welchem Observable sie stammen (siehe *Abbildung 3* und *Listing 12*).

„Utility" sind praktische Hilfs-Operatoren, die keiner der anderen Kategorien genau zugeordnet werden konnten. „do" ist ein sehr praktischer Operator für Seiteneffekte. Er führt lediglich eine Funktion aus, ohne die Werte der Sequenz in irgendeiner Art zu verändern. Im *Listing 13* wurde die Konsolen-Ausgabe von der „subscribe"-Methode in das „do" verlagert, was keinen spürbaren Unterschied macht.

„delay" verzögert die Weitergabe eines Werts um die übergebene Anzahl an Millisekunden und reicht anschließend alle bis dahin aufgetretenen Werte weiter. Nach Ablauf des Delay gehen alle folgenden Werte sofort weiter. *Listing 14* zeigt nach 300 ms die Werte „0", „1", „2" und „3" und nach weiteren 100 ms den Wert „4" an.

Fehlerbehandlung

Neben der Observable-weiten Fehlerbehandlung der „subscribe"-Methode existieren auch mehrere Operatoren, die auf vorhergehende Errors in der Observable-Sequenz reagieren. Operatoren dieser Kategorie ermöglichen eine Wiederaufnahme der Sequenz. Der Error-Handler der „subscribe"-Methode wird also unter Umständen nicht benachrichtigt.

Eine Fehlerbehandlung durch den Error-Handler der „subscribe"-Methode (zweiter Parameter) hat nicht die Möglichkeit, die Subscription weiter aufrechtzuerhalten. Nach der Ausführung der Fehlerbehandlung ist die Subscription zwangsweise beendet, obwohl das Observable möglicherweise noch weitere Werte liefern könnte.

Anders verhält sich der „catch"-Operator. Dieser ruft die übergebene Funktion auf, sofern irgendwo vor ihm in der Sequenz ein Fehler auftreten ist. Der Rückgabewert der Funktion wird anschließend in der weiteren Sequenz verarbeitet. Somit gibt das Beispiel den Wert „fixed it!" mehrfach zurück, obwohl der Wert ursprünglich ein numerischer Zähler des Operators „interval" war *(siehe Listing 15)*.

Vor allem bei Netzwerkanfragen kann es vorkommen, dass fehlgeschlagene Anfragen wiederholt werden müssen. Für diese Aufgabe

```
Observable.interval(100)
  .do(() => {throw Error('something went wrong')})
  .catch((error) => 'fixed it!')
  .subscribe((value) => console.log(value));
```

Listing 15

```
Observable.from(['Jim', 'Johnny', 'Jack'])
  .do((name) => console.log(name))
  .do(() => {throw Error('something went wrong')})
  .retry(2)
  .subscribe((value) => console.log(value));
```

Listing 16

eignet sich der „retry"-Operator. Er wiederholt im Fehlerfall die vorhergehende Sequenz, bis diese erfolgreich ist oder die übergebene Anzahl an Versuchen erreicht wurde. Das *Listing 16* gibt also dreimal „Jim" und anschließend einen Stacktrace des Fehlers aus. „Johnny" und „Jack" kommen gar nicht erst zum Zuge.

Fazit

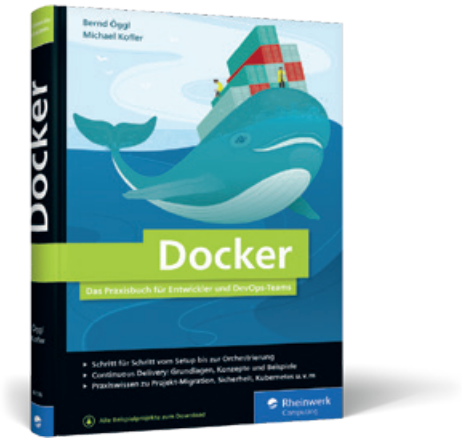
RxJS ist eine hochinteressante Bibliothek, die – gut in die Anwendung integriert – ein mächtiges Werkzeug darstellt. Angular hat hier bereits gute Arbeit geleistet. Die HTTP-Kommunikation, das Data Binding und die Eventbehandlung setzen auf Observables. Hat man die Funktionsweise erst einmal richtig verstanden, gelingt einem das Umdenken von Promises auf Observables recht schnell.

Weiterführende Links

- *http://reactivex.io/rxjs*
- *https://www.learnrxjs.io*
- *http://rxmarbles.com*
- *https://channel9.msdn.com/blogs/j.van.gogh/reactive-extensions-api-in-depth-marble-diagrams-select–where*

Michael Ruttka
michael.ruttka@buschmais.com

Michael Ruttka ist Consultant bei der buschmais GbR. Seine fachlichen Schwerpunkte liegen in der Entwicklung und Konzeption von Web-Anwendungen.



Docker – Das Praxisbuch für Entwickler und DevOps-Teams

gelesen von Georg Zilly

Nachdem sich die Virtualisierung kompletter Systeme fest in unserer IT-Landschaft etabliert hat, ist die Verwendung von Virtualisierung auf Container-Basis, speziell von Docker, schon einige Zeit auf dem Vormarsch. Nach dem schnellen Wandel in der Anfangsphase dieser Technologie stabilisiert diese sich zusehends. Zahlreiche, darunter auch deutschsprachige Bücher zu Docker sind bereits erschienen, nun kommt dieses neue Werk vom Rheinwerk Verlag hinzu, das besonderen Wert auf die praktische Anwendung legt. Daneben gibt es im gleichen Verlag bereits seit einiger Zeit ein allgemeineres Buch „Skalierbare Container-Infrastukturen" zu diesem Themenbereich.

Wer sich nur für die Docker-Technologie interessiert, dürfte mit dem neuen Buch besser bedient sein. Neben den fünf Varianten von „Hello World", die in diesem Buch zu finden sind, enthält es auch viele weitere Beispiele von Anwendungen und deren Konfiguration. So lernt der Leser auch vieles über die Vielfalt moderner IT-Systeme und über Herausforderungen, wie sie in der Praxis anzutreffen sind.

Im ersten Kapitel wird zügig mit drei Varianten von „Hello World" in Form einer Web-Anwendung eingestiegen. Dabei gehen die Autoren davon aus, dass der Leser bereits gut mit seinem Betriebssystem und dessen Werkzeugen umgehen kann. Selbstständig eine Powershell zu starten, einen Editor zu verwenden und Dateien anzulegen, wird dabei vorausgesetzt. In den praxisnahen Beispielen wird hauptsächlich Linux eingesetzt und auch wenn viele der einzelnen Details erklärt werden, ist hier etwas Hintergrundwissen von Vorteil.

Unerfahrene Anwender, vor allem von Windows 7, werden daher eher so ihre Schwierigkeiten mit dem Werk haben, zumal laut Aussage der Autoren Docker erst ab Windows 10 Professional möglich ist. Die Verwendung der Docker-Toolbox, wie sie auf der Docker-Webseite zu finden ist, wird nicht einmal erwähnt. Eventuell hatten die Autoren auch gute Gründe dafür. Schade aber, dass diese Möglichkeit nicht einmal genannt wird. Diesen Anwendern wird einfach die Verwendung von Linux in einer virtuellen Maschine empfohlen.

Nach dem einleitenden Kapitel folgen vier Kapitel mit den eigentlichen Grundlagen zu Docker, die wieder mit praxisnahen Beispielen erklärt sind. Dabei werden häufig Themen angerissen und dann für deren genauere Erläuterung auf später verwiesen, was leider dazu führt, dass man einen klaren roten Faden nicht erkennen kann.

Die folgenden neun Kapitel beschäftigen sich mit Praxisbeispielen. Hier fand der Autor durchaus den einen oder anderen guten Tipp, etwa zum Einsatz eines Reverse Proxy. Im nächsten Kapitel dreht es sich um Continuous Integration und Continuous Delivery. Es handelt sich um eine schöne Einleitung zu diesem Thema mit Tipps dafür, wie Docker auch gut für Entwicklungssysteme eingesetzt werden kann.

Während in den anderen Kapiteln immer wieder Bezug auf den Betrieb von Docker unter anderen Betriebssystemen genommen wird, findet man bei den Tipps zu Sicherheitsthemen fast ausschließlich Hinweise zum Betrieb unter Linux. Die Autoren haben hier sicher einen riesigen Erfahrungsschatz, der so im Zusammenhang mit Docker kaum woanders zu finden ist.

Die letzten beiden Kapitel beschäftigen sich mit dem Einsatz von Docker als skalierbares System und in Cloud-Umgebungen. Hier findet man ei-

Autoren: Bernd Öggl, Michael Kofler

Titel: Docker – Das Praxisbuch für Entwickler und DevOps-Teams

Verlag: Rheinwerk Computing

Umfang: 426 Seiten

Preis: Buch 39,90 Euro, eBook 35,90 Euro, Buch und eBook 44,90 Euro

ISBN 978-3-8362-6176-0

nen Einstieg in die Docker-eigenen Swarm-Funktionalitäten und das ursprünglich von Google entwickelte Kubernetes. Beide Technologien werden in lokaler Umgebung und auf Cloud-Systemen wie Hetzer, Amazon, Microsoft und Google vorgestellt. Die Erklärungen beschreiben hier aber mehr die grundsätzliche Vorgehensweise und gehen weniger ins Detail.

Wer sich Informationen dazu erhofft, wie man mit Programmier-Werkzeugen eigene Tools zur Docker-Engine bauen kann, etwa mit einem API, wird in diesem Buch nicht fündig. Trotz vieler praxisbezogener Anwendungen findet man ebenso wenig Hilfe zum Betrieb eines eigenen Repository oder zur Datensicherung von Containern.

Fazit

Das Praxisbuch ist tatsächlich ein solches. Die vielen praxisbezogenen Beispiele und der große Erfahrungsschatz der Autoren, vor allem im Bereich Linux, liefern gute Hilfestellung und wertvolle Tipps für den Alltag. Es ist eher für Leute geeignet, die schon Erfahrung mit Linux besitzen und erste Erfahrungen mit Docker. Anfänger werden es eher schwer haben und eine klare Linie im Aufbau der Themen wäre hilfreich.

Trotz des Versuchs, immer wieder auf andere Plattformen hinzuweisen, hat das Buch einen eindeutigen Schwerpunkt auf Linux; kein einziges der neun Praxiskapitel setzt eine Windows-Anwendung in einem Windows-Container ein. Das ist zugegebenermaßen auch der Plattform geschuldet und dem Umstand, dass der Leser keine Geldbeträge in irgendwelche Lizenzen stecken sollte, um die Beispiele nachzubauen. Das Werk nimmt an einigen Stellen Bezug auf aktuelle Themen und Entwicklungen. Für die Kapitel zu den Cloud-Themen hätte sich der Autor mehr und ausführlichere Informationen gewünscht.

Das Buch ist als Druck, eBook und auch in einer Kombination aus beidem erhältlich. Die Druckausgabe hat ein integriertes Lesezeichen; Beispiele und Projektdateien werden zusätzlich auf der Webseite des Verlags und über GitHub zur Verfügung gestellt.

Georg Zilly
zilly@jonasoft.de

Helidon Takes Flight – ein neues Open-Source Java-Microservice-Framework von Oracle

Dmitry Kornilov, Oracle-Mitarbeiter und unter anderem mit der Helidon-Projektleitung betraut, stellt auf der Plattform der Oracle Developers am 7. September 2018 unter dem Namen „Helidon Project" ein neues Java-Microservice.Framework aus der Familie der MicroProfiles vor. Kornilov gibt dazu einen Überblick über die Entstehung und den technischen Background der neuen Oracle-Entwicklung.

Helidon ist griechisch und bedeutet Schwalbe, die laut Wikipedia ein Vogel ist, „der einen schlanken, stromlinienförmigen Körper und lange spitze Flügel hat, die eine große Manövrierfähigkeit und … einen sehr effizienten Flug ermöglichen". Perfekt, um durch die Wolken zu fliegen.

Als mit dem Eintritt in die Cloud-Welt Microservice-Architekturen zur Entwicklung von Cloud-Diensten immer populärer wurden, stellte sich heraus, dass auch die Entwickler sich umorientieren mussten. Laut Kornilov sei es zwar möglich, Microservices mit Java EE zu erstellen, aber es ist besser, ein Framework von Grund auf für den Aufbau von Microservices zu haben. „Wir wollen ein leichtgewichtiges Set von Bibliotheken erstellen, das keinen Applikationsserver benötigt und das in Java-SE-Anwendungen verwendet werden kann." Diese Bibliotheken könnten getrennt voneinander verwendet werden, aber wenn man sie zusammen einsetzt, bilden sie die Grundlage, die ein Entwickler benötige, um einen Mikroservice zu erstellen: Konfiguration, Sicherheit und einen Webserver.

Es gibt bereits einige Bemühungen, Standard-Microservice-Frameworks aus MicroProfiles zu erstellen. MicroProfile sei in der Java EE/Jakarta EE Community sehr beliebt und bieten eine ähnliche Entwicklungserfahrung wie Java EE. „Wir mögen die Idee und unterstützen die Initiative. Helidon implementiert MicroProfile 1. 1.", so Kornilov.

Helidon gibt es in den zwei Varianten Helidon SE und Helidon MP. Es umfasst damit die zwei Kategorien Micro-Frameworks und MicroProfile. Entwickler können sich entscheiden, welche sie in ihren Anwendungen verwenden wollen.

Kornilov kündigt an: „Wir werden weiterhin an der Implementierung neuer Versionen von MicroProfile arbeiten und beabsichtigen, relevante Jakarta EE-Standards in diesem Bereich so zu unterstützen, wie sie definiert sind."

Zahlreiche Pläne seien bereits geschmiedet. Unser kurzfristiges Ziel ist es, laut Kornilov, Helidon in der Java-Community bekannt zu machen. Es sei geplant, auf einigen Konferenzen über Helidon zu sprechen. Darüber hinaus finden vier Helidon-bezogene Gespräche auf der Code One 2018 statt. Vorträge zur EclipseCon Europe 2018 wurden eingereicht und werden dort am Jakarta EE/MicroProfile Community Day teilnehmen. Schulungsmaterial wie Videos, Muster und Artikel sind in der Vorbereitung. Weitere Informationen unter *„https://medium.com/oracledevs/helidon-takes-flight-fb7e9e390e9c".*

Data Analytics 2019

Die Datenexplosion meistern

26. & 27. März | Phantasialand bei Köln

analytics.doag.org

ORACLE Cloud

DOAG

iJUG Verbund

Alle Mitglieder auf einen Blick

Der iJUG möchte alle Java-Usergroups unter einem Dach vereinen. So können sich alle Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne unter office@ijug.eu melden.

www.ijug.eu

Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, www.ijug.eu) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

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

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

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
Chefredakteur (ViSdP): Wolfgang Taschner
Kontakt: redaktion@doag.org

Redaktionsbeirat:
Andreas Badelt, Melanie Feldmann, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, André Sept

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Fotonachweis:
Titel: Original © Vecteezy.com
S. 9: © Rheinwerk Computing
S. 12: © Freepik/freepik.com
S. 16: © aimage/123RF
S. 26: © Schuchrat Kurbanov/DOAG
S. 31: © Kirill Makarov/123RF
S. 36: © orson/123RF
S. 42: © Dario Lo Presti/123RF
S. 46: © aimage/123RF
S. 63: © Rheinwerk Computing

Anzeigen:
Simone Fischer, DOAG Dienstleistungen GmbH
Kontakt: anzeigen@doag.org

Mediadaten und Preise unter:
www.doag.org/go/mediadaten

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

Inserentenverzeichnis


ORAWORLD
Das e-Magazine für alle Oracle-Anwender!
EOUC
EMEA ORACLE USERGROUP COMMUNITY
• Spannende Geschichten aus der Oracle-Welt
• Technologische Hintergrundartikel
• Leben und Arbeiten heute und morgen
• Einblicke in andere User Groups weltweit
• Neues (und Altes) aus der Welt der Nerds
• Comics, Fun Facts und Infografiken
Jetzt Artikel einreichen oder Thema vorschlagen!
Jetzt e-Magazine herunterladen
www.oraworld.org

JavaLand

Early Bird
bis 15. Jan. 2019

19. - 21. März 2019 in Brühl bei Köln

Ab sofort Ticket & Hotel buchen!

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu