Ausgabe 04/2019

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 04

Java aktuell

iJUG Verbund
www.ijug.eu

Java 12
Die neuen Features im Überblick

Jakarta EE
Neuigkeiten und Ergebnisse der Verhandlungen

Web-APIs
Diese Programmierschnittstellen erleichtern Ihnen die Arbeit

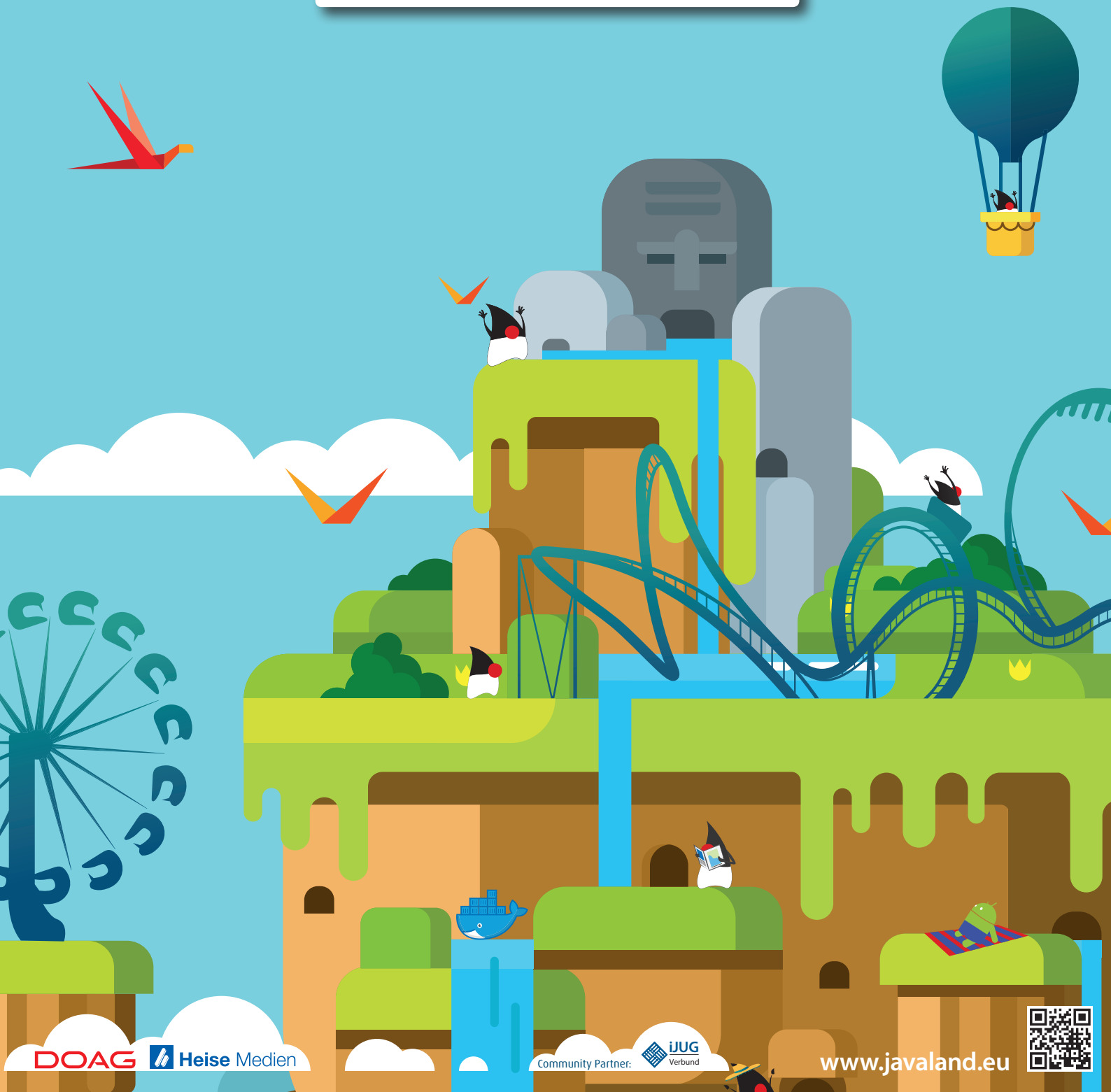
Java im Wandel

JavaLand

Save the Date

16. - 19. März 2020

in Brühl bei Köln



DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu

Das Ende von Java EE

Endlich gibt es Neuigkeiten aus den mittlerweile eineinhalbjährigen Verhandlungen zwischen Oracle und der Eclipse Foundation zu vermelden. Alle Details hat Markus Karg in seiner Eclipse Corner auf Seite 8 für Sie zusammengefasst. So viel sei schon einmal gesagt: Es kommt einiges an Arbeit auf die Foundation im Hinblick auf das Projekt "Jakarta EE" zu.

Doch nun zu etwas Erfreulicherem: Pünktlich zur JavaLand im März 2019 wurde das langersehnte Java 12 veröffentlicht. Daher gab es auf der Community-Konferenz viel Neues zu berichten und auszutauschen. Auf den Seiten 9 bis 12 gibt Falk Sippach einen Einblick in die neuen Features von Java 12. Dabei geht er insbesondere auf die neuen Switch Expressions ein. In der vergangenen Ausgabe zeigte Michael Paege Risiken und Handlungsalternativen für Java-Nutzer in Bezug auf Oracles neue Lizenzpolitik auf – seitdem hat sich einiges getan. Ein aktuelles Update zur Thematik finden Sie auf Seite 13. Außerdem lassen wir in dieser Ausgabe Bernd Müllers „Unbekannte Kostbarkeiten des SDK" wiederaufleben! Langzeit-Java-aktuell-Fans dürfte diese Rubrik noch ein Begriff sein: von 2011 bis 2015 war sie regelmäßig vertreten. In dieser Ausgabe widmet er sich dazu der Just-In-Time-Compilation. Weiterhin erwarten Sie spannende Artikel zu Hadoop, Web-APIs, SCS-Architekturen und vieles mehr.

Wir wünschen Ihnen viel Spaß beim Lesen!

Ihre

Lisa Damerow
Redaktionsleitung Java aktuell

Inhalt

Die neuen Features von Java 12 im Detail

Was es bei der Containerisierung zu beachten gibt

Diese APIs sollte jeder Entwickler kennen

Praxisbericht der Migration auf eine SCS-Architektur

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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

10. Februar 2019

Wie soll MicroProfile mit „breaking changes" umgehen?

Ende Januar haben die MicroProfiler entschieden, wie mit „breaking changes" umzugehen ist. Der MicroProfile „Release Train" sieht drei Plattform-Releases pro Jahr vor, jeweils im Abstand von vier Monaten. Im Juni-Release eines jeden Jahres können die Einzel-Spezifikationen nun immer inkompatible Änderungen unterbringen. Bei den anderen Releases ist dies nur möglich, wenn die Mehrheit der Committer dafür stimmt. Damit wollen sie sich in entscheidenden Situationen die nötige Flexibilität bewahren, die MicroProfile auszeichnet. Wenn in einem Plattform-Release kein „breaking change" enthalten ist, führt es auch im Juni nicht automatisch zu einer neuen Hauptversion. MicroProfile 28 soll es damit so schnell nicht geben *[1]*.

11. Februar 2019

JUnit 5.4

JUnit 5.4 ist da. Neben vielen kleinen Neuerungen lässt sich jetzt die Reihenfolge von Tests per @TestMethodOrder steuern (klingt nach einem Anti-Pattern, kann aber bei Integrationstests das Entwicklerleben erleichtern). Außerdem lassen sich Abhängigkeiten für Maven und Gradle jetzt auf das Bundle „junit-jupiter" reduzieren.

15. Februar 2019

JavaFX 12

Wieder kurz vor dem Erscheinen des Java-Releases mit gleicher Nummer ist das herausgetrennte, von Oracle und Gluon getriebene JavaFX in Version 12 fertig. Aber genau wie beim OpenJDK ist auch JavaFX 12 kein Long-Term Support Release – Fixes werden also nur bis zum Erscheinen von Version 13 geliefert. Die Änderungen sind aber eh überschaubar *[2]*.

15. Februar 2019

Helidon 1.0

Oracles Open-Source-Microservice-Framework Helidon („Schwalbe") ist jetzt in Version 1.0.0 offiziell freigegeben worden. Es implementiert das vergleichsweise alte MicroProfile 1.2 (Helidon MP; Support für MicroProfile 2.2 ist in Arbeit) bzw. ein eigenes „functional style" API (Helidon SE). Letzteres enthält z.B. auch ein OpenTracing API, das in MicroProfile erst ab Version 1.3 vorkommt; es hat auch noch einige Überarbeitungen für Version 1.0.0 erfahren, aber: „From this point on, we will have much greater API stability" – heißt es in den Release Notes.

28. Februar 2019

WildFly 16 mit Galleon-Preview

WildFly 16 ist draußen und mit dem Java 12 Release Candidate getestet (Java 8 wird auch noch bis voraussichtlich Version 18 offiziell unterstützt). Die zentrale Neuheit ist das Provisionierungstool Galleon, allerdings noch als „Tech Preview", das unter Nutzung von Maven Repositories einen für die jeweilige Anwendung maßgeschneiderten Anwendungsserver erstellt. Dazu werden die benötigten „Layers" (ein Feature oder eine Kombination mehrerer Features) per Kommandozeile oder in einer Provisioning.xml-Datei definiert, um daraus dann die Konfiguration des Anwendungsservers (standalone.xml) zu erstellen; im Unterschied zu Red Hats hauseigener Alternative Thorntail, die ein „uber-jar" erzeugt.

28. Februar 2019

Jakarta EE News?

Die neue Ausgabe des Eclipse-Newsletters hat den Schwerpunkt Jakarta EE. Release 8 zieht sich weiter hin. Im Wesentlichen arbeiten die Juristen hinter den Kulissen – oder vielleicht auch nicht, weil weder Oracle noch Eclipse die aufwendigen Prüfungen stemmen können oder wollen? Package-Namen mit javax und die juristisch abgesicherte Übergabe der Spezifikationen inklusive des geistigen Eigentums daran lauten die zentralen Punkte. Aber im Newsletter wagt Arjan Tijms von Payara einen Ausblick auf das Folge-Release Jakarta EE 9. Bei JSF 3.0 sollen alte Zöpfe abgeschnitten werden – unter Brechung der Abwärtskompatibilität für ganz(!) alte Applikationen. Auf der Plus-Seite stehen die stärkere Nutzung von CDI und die Einbindung eines Action-Frameworks. Im Bereich Security soll sich einiges tun, z.B. in Bezug auf Rollen-Annotations, „custom authorization rules" oder die Unterstützung für Standardprotokolle wie OpenID Connect. JAX-RS soll in vollständig mit CDI „verheiratet" werden. Auch bei Interceptors und Concurrency soll sich einiges tun. Unter dem Arbeitstitel „Requestlets" soll das Zusammenspiel zwischen CDI und Servlets verbessert werden. Arjans vollständiger Beitrag ist hier zu finden *[3]*.

8. März 2019

Quarkus.io

Das nächste Microservices-Framework kommt, diesmal von Red Hat und ganz bescheiden angekündigt mit „Supersonic Subatomic Java". Quarkus heißt es (laut Urban Dictionary: „an insult used when little children are around." Aha…). „A Kubernetes Native Java stack tailored for GraalVM and OpenJDK HotSpot", verspricht die Website. Extrem kleine Memory Footprints und Startup-Zeiten im Millisekundenbereich (durch „compile time boot" – das Ausführen von Startup Code zur Compile Time und Serialisieren ins Image) sollen Java in hochskalierbaren Kubernetes-Umgebungen bis hin zu Serverless-Ansätzen wieder ganz nach vorn bringen. Dabei werden reaktive und imperative Entwicklung nahtlos miteinander verbunden.

19. März 2019

Java SE 12

Java 12 ist da. Entsprechend dem neuen halbjährlichen „Release Train" sind die Änderungen zu Version 11 überschaubar. Sharat Chander, Chef-Produktmanager für Java SE, betont im Blog, dass die Änderungsgeschwindigkeit nicht nachgelassen hat, sondern Änderungen nur früher verfügbar sind. Manchmal auch Schritt für Schritt, wie die neuen Switch Expressions, die erstmal als Preview eingeführt werden. Weitere Änderungen betreffen insbesondere die Garbage Collection – Verbesserungen am G1 und der neue, auf kurze Pausen ausgelegte Collector „Shenandoah". Startup Performance Tuning über „Default Class Data Sharing (CDS) Archives" ist dabei, sowie ein „JVM Constants API", das einen vereinfachten, standardisierten Umgang mit dem Constant Pool bietet. Auch eine Microbenchmark-Suite für das JDK gibt es. Außerdem werden alle Source Files des arm64-Ports entfernt, um nicht neben aarch64 eine zweite Implementierung pflegen zu müssen. Die Liste der aktuellen und für zukünftige Releases geplanten Features ist unter *[4]* zu finden.

20. März 2019

JavaLand

Es war wieder soweit: Die JavaLand hat ihre Tore geöffnet. Am Vortag gab es die JavaLand 4 Kids: Die kleineren und größeren Kinder hatten viel Spaß, u.a. mit dem Bau und der Programmierung von Robotern basierend auf Lego WeDo, Mindstorms und Calliope. Die eigentliche Konferenz wurde wieder mit einem frühen Vortragsslot gestartet. Statt Frühstücksquark Quarkus um 8:30 Uhr. Nach der offiziellen Begrüßung zeigte Ed Burns einen etwas anderen Blickwinkel: Was ist das Geheimnis einer erfolgreichen Programmier-Plattform? Danach ging es auf zum „Community Tent" zu den Community-Aktivitäten. Die großen Themen waren nicht überraschend: Microservices, die aktuellen Hype-Frameworks und Spezifikationen wie MicroProfile. Von Helidon war nicht viel zu sehen, dafür aber von der GraalVM. Podiumsdiskussionen zu Jakarta EE sind auch wichtig, um sich auszutauschen. Auch Lizenzfragen sind weiterhin ein Thema – ebenfalls mit einem Vortrag plus Podiumsdiskussion abgedeckt. Wieder einmal zwei vollgepackte Konferenz-Tage. Zum Karussellfahren bleibt auch keine Zeit, weil erstmal sportliche Betätigung angesagt ist – ein Kung-Fu-Workout mit Markus Karg und danach die Party mit Live-Band im STOCK'S. Es hat sich auf jeden Fall gelohnt, war aber, wie immer, zu kurz – also bis zum nächsten Jahr!

21. März 2019

Kotlin in Top 20 im RedMonk Ranking

Im letzten RedMonk-Ranking *[5]* der (auf StackOverflow und in GitHub-Projekten) beliebten Sprachen hat sich Kotlin erstmals unter die Top 20 geschoben – und steht damit im Java-Ökosystem nur noch hinter Scala (Platz 14) und Java (2) selbst und, genau genommen, JavaScript (1). Eine Momentaufnahme, aber der Trend scheint für Kotlin zu sprechen.

25. März 2019

Thorntail 2.4

Thorntail ist in Version 2.4 verfügbar. Es unterstützt MicroProfile 2.2 und Java SE 11, wobei bei Letzterem beachtet werden muss, dass nur der konventionelle Classpath, nicht der Modulepath des mit Java 9 gekommenen Modulsystems unterstützt wird – Abhängigkeiten werden weiter per „jboss-deployment-structure.xml" definiert.

25. März 2019

OpenJDK von Alibaba

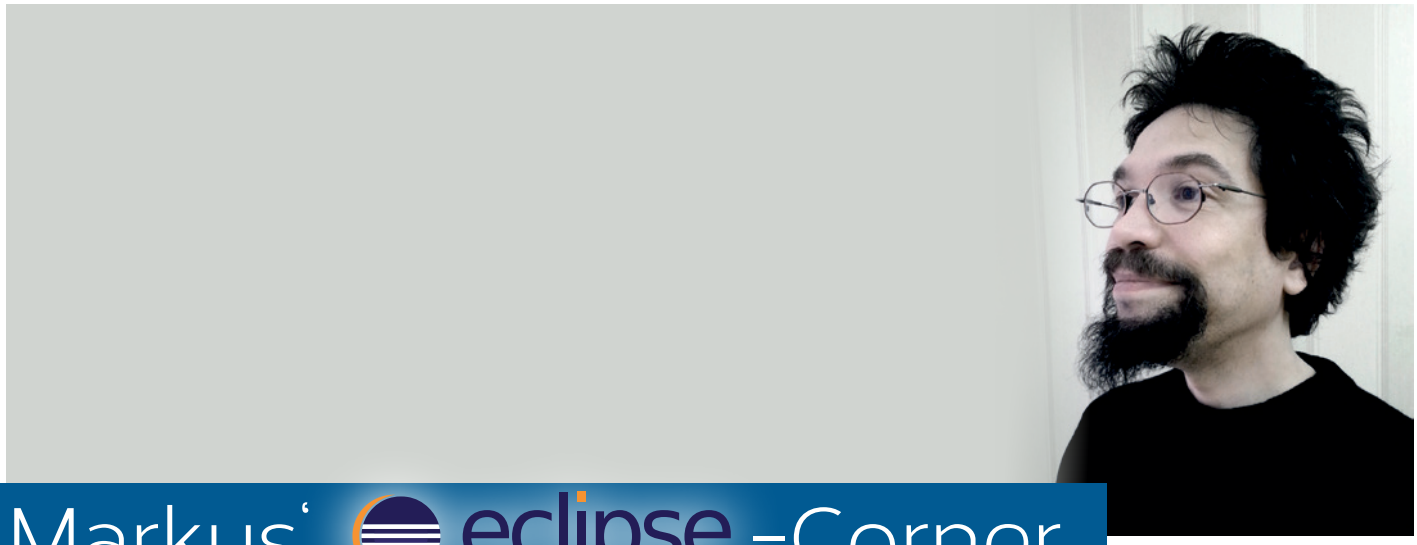
Der chinesische eCommerce- und Cloud-Gigant Alibaba hat seinen eigenen Fork des OpenJDK 8 unter unveränderten Lizenzbedingungen wieder veröffentlicht, unter dem Namen „Dragonwell"*[6]*. Warum auch nicht, auch Konkurrent Amazon lässt ja die Community an seinem Corretto teilhaben. Laut GitHub-Doku ist das JDK bei Alibaba auf über 100.000 Servern im Einsatz – es ist allerdings nur für Linux ausgelegt. Fun Fact: Laut heise.de-Artikel ist Dragonwell ein gerösteter Grüntee (deutsch „Drachenbrunnentee") – ob das in der Java-Welt ankommt?

Referenzen

[1] *https://docs.google.com/document/d/16v3jVkcDzVz9BVU5aGEzPVbK-a8Blx7S1gbqToVUaLs/edit#*
[2] *https://github.com/javafxports/openjdk-jfx/blob/jfx-12/doc-files/release-notes-12.md#release-notes-for-javafx-12*
[3] *https://www.eclipse.org/community/eclipse_newsletter/2019/february/Jakarta_EE_9.php*
[4] *https://openjdk.java.net/jeps/0*
[5] *https://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/*
[6] *https://github.com/alibaba/dragonwell8*

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 im DOAG e.V. aktiv, zunächst als Co-Leiter der SIG Development und später der SIG Java. Seit 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").



Markus' eclipse-Corner

Das war's dann also: Java EE ist endgültig tot! Die Eclipse Foundation hat 18 Monate mit Oracle verhandelt – mit dem Ergebnis, dass keinerlei Oracle-Marken von der Stiftung verwendet werden dürfen. Das Verhandlungsergebnis ist jedoch nicht "nichts", immerhin wurde erreicht, dass das bislang lizenzgebührenpflichtige Java EE TCK zu Open Source erklärt wurde (eine langjährige Forderung der Apache Foundation) und mit viel Glück folgen noch die Quelltexte der Spezifikationen, wenn man das Einverständnis alle Autoren erhalten hat. Es ist aber ein dramatisches Ergebnis für die Anwender. Denn unter das Verbot fallen nicht nur die Projektnamen wie "Java API for Restful WebServices" und entsprechende Abkürzungen wie "JAX-RS", sondern auch die zugehörigen Packages wie "javax.ws.rs". Oracle erlaubt die Nutzung dieser Namen nur dann, wenn die Schnittstellen zu 100 Prozent kompatibel bleiben und in keinster Weise erweitert werden – oder garantiert ist, dass nur JREs damit gebündelt werden, die von Oracle selbst stammen oder für die Oracle Lizenzgebühren erhält. Die Eclipse Foundation hat nicht vor, für ihre Laufzeitumgebungen Lizenzgebühren zu bezahlen, womit das Thema Features de facto vom Tisch ist. Würde die EF Lizenzzahlungen an Oracle (und eben nur an Oracle) leisten, wäre sie laut Juristen nicht mehr gemeinnützig. Fällt die Gemeinnützigkeit, fällt die Steuerbefreiung von Spenden. In der Folge spendet kein Unternehmen mehr und die Stiftung wäre in ihrer Existenz bedroht. Kurzum, Fehlerbereinigungen für die von Sun/Oracle gemachten Bugs soll die EF nach Oracles Wunsch weiterhin entwickeln, neue Features aber bitte nicht! Das Hinzufügen eines einzigen Parameters zu einer bestehenden Methode würde Oracle als Verstoß gegen das Markenrecht ahnden. Insofern ist die Weiterentwicklung der bestehenden, bislang unter Java EE bzw. Jakarta EE subsummierten APIs faktisch verboten.

Als Lösungsweg erlaubt Oracle, alle bestehenden Projekte, Abkürzungen und Packages umzubenennen. Mit ein paar Tagen Aufwand ist das zu schaffen. JAX-RS beispielsweise hat dies bereits vorbereitet und wartet nur noch auf das OK des Project Management Committees (PMC). Dieses tut sich jedoch schwer, denn für bestehende Anwendungen hat diese Lösung fatale Folgen: Die Binärkompatibilität wird gebrochen, alle Importe müssen im Quellcode der konsumierenden Anwendung umgeschrieben und die Anwendung neu kompiliert werden. „Write once, run anywhere" (kurz WORA) sieht anders aus. Im Endeffekt wäre die Anwendung nicht mehr Java EE kompatibel, sondern "nur noch" Jakarta EE kompatibel. Die einen sehen darin kein Problem, andere sehen die Chance, alte Zöpfe abzuschneiden, wieder andere prophezeien den Untergang des Abendlands. Aus der Microprofile-Ecke werden Rufe laut, einfach alles in die Tonne zu treten und auf Microprofile-Basis alle Anwendungen neu zu schreiben. Daher wird derzeit heftig diskutiert, ob ein "Big Bang" – also die sofortige Umbenennung aller Packages – oder nur der veränderten Packages erfolgen soll, man also einen Mix aus alten und neuen Packages pflegt, oder das Ganze gar über einen zehnjährigen Zeitraum streckt, was ständiges Nachpflegen der Anwendungen erfordert. Wie auch immer: Altanwendungen müssen entweder umgebaut oder in den Wartungsmodus gebracht werden.

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.

Java 12 ist da

Falk Sippach, Orientation in Objects

Mitte März erfolgte wie geplant die Veröffentlichung des JDK 12 mit den folgenden Java Enhancement Proposals (JEP) [1]:

- *JEP 189: Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)*
- *JEP 230: Microbenchmark Suite*
- *JEP 325: Switch Expressions (Preview)*
- *JEP 334: JVM Constants API*
- *JEP 340: One AArch64 Port, Not Two*
- *JEP 341: Default CDS Archives*
- *JEP 344: Abortable Mixed Collections for G1*
- *JEP 346: Promptly Return Unused Committed Memory from G1*

Auf den ersten Blick sieht es nach wenig aus. Neben internen Änderungen und Performanceverbesserungen sticht aus Entwicklersicht hauptsächlich ein Punkt ins Auge: Switch Expressions. Im Rahmen dieses Artikels wollen wir das neue Feature näher beleuchten und auch einen Blick auf die weiteren Neuerungen werfen.

Probleme mit dem Switch Statement

Das Konstrukt „Switch" ist eigentlich eine Idee aus der prozeduralen Programmierung und ein Überbleibsel aus der Tatsache, dass die Syntax von Java initial stark an C angelehnt wurde. Für den erfahrenen Programmierer ist das Switch Statement jedoch ein Code Smell und kann im Sinne der Objektorientierung besser mit dem Strategie-Entwurfsmuster ausgedrückt werden. Nichtsdestotrotz gibt es Situationen, in denen man mit einem Switch sehr effizient

und übersichtlich Fallunterscheidungen umsetzen kann. Allerdings hatte die bisherige Implementierung einige Schwächen. Werfen wir darum zunächst einen Blick auf das klassische Switch Statement *(siehe Listing 1)*.

Diese Variante wirkt etwas unhandlich und wird bei komplexeren `case`-Blöcken schnell unübersichtlich. Zudem können kleine Flüchtigkeitsfehler oder Unwissenheit über die genaue Funktionsweise zu schlecht nachvollziehbaren Bugs führen. Ein möglicherweise unabsichtlicher Fall-Through entsteht, wenn man einen `case`-Block nicht mit einem `break` beendet. So springt die Ausführung weiter in den nächsten Block, auch wenn dessen Bedingung gar nicht zur Switch-Variablen passt. In unserem Beispiel würde bei der Ausführung mit `i = 1` als Ergebnis `two` zurückgegeben. Sinnvoll kann dieses Verhalten wiederum sein, wenn mehrere Switch-Bedingungen auf das gleiche Ergebnis abgebildet werden sollen, wobei dann die durchfallenden `case`-Zweige explizit leer bleiben *(siehe Listing 2)*.

Wenn man vergisst, einen bestimmten Fall abzudecken, wird der Switch scheinbar nicht ausgeführt. Daher empfiehlt es sich immer, einen `Default`-Zweig zu definieren. Dieser kann dann wenigstens zur Laufzeit einen Fehler für nicht definierte Input-Parameter werfen. Denn leider warnt uns der Compiler nicht, wenn wir den Switch mit Werten kleiner als eins oder größer als drei aufrufen. In den unterschiedlichen `case`-Blöcken wird typischerweise redundant immer wieder die gleiche temporäre Variable mit natürlich unterschiedlichen Werten befüllt. Das widerspricht dem DRY-Prinzip ("Don't Repeat Yourself") und erschwert die Nachvollziehbarkeit, insbesondere wenn mit dem Fall-Through gearbeitet wird. Temporäre Variablen sind nicht umsonst ein Code Smell. Idealerweise kann man komplett auf sie verzichten.

Zu guter Letzt führt der globale Scope des Switch Statement dazu, dass in den einzelnen Blöcken möglicherweise unterschiedliche Variablen definiert werden müssen, um ungewollte Nebeneffekte bei der Verwendung der gleichen Namen zu vermeiden.

Alles neu macht die Switch Expression

Ab Java 12 können die Case-Zweige jetzt mehrere Labels haben. Das ermöglicht eine redundanzfreie und kompakte Schreibweise, um einen Fall-Through zu verwenden *(siehe Listing 3)*.

Mit der Pfeil-Syntax existiert eine Variante, die der Definition von Lambda-Ausdrücken ähnelt. Jeder Zweig wird auf eine Zeile reduziert und das `break` entfällt. Das spart wieder Boiler Plate Code und macht Switch Statements kompakter und übersichtlicher. Ein unabsichtlicher Fall-Through ist so nicht mehr möglich. Es wird immer nur genau das eine Statement ausgeführt. Sobald mehr als eine Anweisung auf der rechten Seite benötigt wird, muss der Code innerhalb eines Blockes stehen. Damit wird ein eigener Scope geschaffen, der zugleich eine kollisionsfreie Variablendeklaration ermöglicht *(siehe Listing 4)*.

In der funktionalen Programmierung gibt es typischerweise keine Zustandsänderungen in Form von Variablenzuweisungen. Vielmehr wird

```
public String describeInt(int i) {
  String str = "not set";
  switch(i) {
    case 1:
      str = "one";
    case 2:
      str = "two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```

Listing 1

```
public String describeInt(int i) {
  String str = "not set";
  switch(i) {
    case 1:
    case 2:
      str = "one or two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```

Listing 2

```
public String describeInt(int i) {
  String str = "not set";
  switch (i) {
     case 1, 2:
       str = "one or two";
       break;
    case 3:
       str = "three";
       break;
  }
  return str;
}
```

Listing 3

mit Ausdrücken gearbeitet, die als Inputparameter wiederum anderen Funktionen zur weiteren Bearbeitung übergeben werden können. Die Switch Expression ist jetzt auch ein Ausdruck, der ein Ergebnis zurückliefert. So kann man das Resultat zum Beispiel direkt einer Variablen (besser Konstanten) zuweisen oder einem Methodenaufruf als Parameter übergeben beziehungsweise per `return` zurückgeben *(siehe Listing 5)*.

Übrigens muss jeder Zweig mit dem `break` jeweils genau einen Wert zurückliefern („break-with-value"-Semantik). Dadurch kann der Compiler prüfen, ob wir alle möglichen Fälle behandeln oder wenigstens einen `Default`-Block als Fallback angegeben haben. Damit ist sichergestellt, dass immer genau ein Zweig abgearbeitet und dessen Ergebnis zurückgeliefert wird. Switch Statements ohne

```
public String describeInt(int i) {
  String str = "not set";
  switch (i) {
    case 1, 2 -> str = "one or two";
    case 3 -> {
    var i = complexComputation();
    str = "three" + i;
  }
  }
  return str;
}
```

Listing 4

```
public static String describeInt(int i) {
  return switch (i) {
    case 1, 2:
      break "one or two";
    case 3:
      break "three";
    default:
      break "smaller than one or bigger than three";
  };
}
```

Listing 5

```
public static String describeInt(int i) {
  return switch (i) {
    case 1, 2 -> "one or two";
    case 3 -> "three";
    default -> "smaller than one or more than three";
  };
}
```

Listing 6

sichtbaren Output gehören somit der Vergangenheit an. Durch die schon angesprochene alternative Pfeil-Notation verkürzt sich die Syntax der Switch-Expression nochmals *(siehe Listing 6)*.

Die Switch Expression arbeitet natürlich auch nahtlos mit der in Java 10 eingeführten Local Variable Type Inference zusammen. Bei der Zuweisung zu einer mit „var" deklarierten Variable wird der spezifischste gemeinsame Typ aller `case`-Zweige (kleinster gemeinsamer Nenner) ausgewählt. Ein Nachteil von Switch Expressions ist aber, dass ein Ausdruck letztendlich immer zu einem Wert aufgelöst werden muss. Dadurch ist es nicht möglich, innerhalb der Expression `return` oder `continue` zu benutzen. Das wurde häufig verwendet, um direkt aus einer Methode beziehungsweise Iteration zu springen. Durch das Werfen einer Exception kann man allerdings weiterhin den Ausdruck vorzeitig beenden.

Als Datentypen sind bisher `byte`, `short`, `char`, `int`, deren Wrapper-Klassen sowie Enums und Strings in Switch Statements erlaubt. Es gibt bereits Pläne, zusätzlich `float`, `double` und `long` zu unterstützen. Mit dem JEP 305 soll in zukünftigen Versionen zudem das Pattern Matching – eine Art Switch on Steroids – Einzug in das JDK halten. Die Intention des Pattern Matching ist eigentlich die Destrukturierung von Werten, also viel mehr als das reine Unterscheiden beliebiger Datentypen. Nichtsdestotrotz kann man die Switch Expressions in der heutigen Form bereits als eine Art Vorgeschmack auf das Pattern Matching sehen. Nicht vergessen sollten wir allerdings, dass es sich aktuell im JDK 12 nur um eine Preview handelt. Bis zum nächsten LTS-Release (JDK 17) kann sich durch Feedback aus der Community an der Umsetzung nochmal einiges ändern.

Weitere Neuerungen

Werfen wir noch einen Blick auf einige der anderen Punkte aus den Release Notes *[2]*. So wird ab dem JDK 12 im Sourcecode eine Suite von etwa 100 Microbenchmarks ausgeliefert, die auf dem bekannten Java Microbenchmark Harness (JMH) basieren. Die Suite soll die Ausführung von existierenden und die Erstellung von neuen Microbenchmarks erleichtern.

Das JVM Constants API ermöglicht das typsichere und damit weniger fehleranfällige Laden von Werten aus dem Java Constants Pool (`int`, `float`, `String`, `Class`). Dieses API ist vor allem für Werkzeuge hilfreich, die Klassen und Methoden manipulieren. Bei den Garbage Collectors gibt es ebenfalls wieder einige Neuerungen. So hat der Default-GC G1 diverse Performance-Verbesserungen erfahren. Mit Shenandoah wurde zudem ein neuer Garbage Collector eingeführt. Er wurde ursprünglich von Red Hat entwickelt und ist eine Alternative zum bereits existierenden ZGC *[3]*, verwendet jedoch einen anderen Algorithmus. Dabei sind die STW-Pausen (Stop-the-World) sehr kurz und zudem können große Mengen an Hauptspeicher (mehrere Terabyte Heap) effizient verwaltet werden. Dazu werden die Aufräumarbeiten konkurrierend zur eigentlichen Anwendung erledigt, was diese dann allerdings etwas langsamer macht. Dieses Feature ist noch experimentell und darum nicht im Standard (Oracle) OpenJDK enthalten.

Um beim Starten der JVM nicht immer alle Metadaten der Klassen erneut laden zu müssen, können diese Informationen in sogenannten Class-Data-Sharing-Archiven abgelegt werden. Das verkürzt die Startzeit, verringert den Memory Footprint und verbessert die Performance der Garbage Collection. Für 64-Bit-Builds ist ab Java 12 das Abspeichern dieser Informationen der Standard, um die Startup-Zeit von Java-Anwendungen out of the box verbessern zu können.

In der JDK-Klassenbibliothek gab es auch ein paar Änderungen *[4]*. Die Klasse String hat zum Beispiel zwei neue Methoden spendiert bekommen, um einen Text entweder einzurücken (indent) oder umzuwandeln (transform). Die Methode „mismatch" in Files findet das erste Byte, das sich zwischen zwei Dateien unterscheidet *(siehe Listing 7)*.

Die neue Klasse "CompactNumberFormat" formatiert Zahlen abhängig vom Locale basierend auf dem Unicode Common Locale Data Repository (CLDR) *[5] (siehe Listing 8)*.

Ursprünglich sollten neben den Switch Expressions auch die Raw-String-Literals in Java 12 erscheinen. Allerdings wurde die Veröffentlichung dieses Features auf unbestimmte Zeit verschoben, weil noch der nötige Feinschliff gefehlt hat: „... in reviewing the feedback we have received, I am no longer convinced that we've yet got to the

```
// "     foobar"
System.out.println("foobar".indent(5));

// java.lang.Integer
var clazz = "42".transform(Integer::valueOf).getClass();

final long mismatch = Files.mismatch(Path.of("a.out"),
Path.of("b.out"));
```

Listing 7

```
import java.text.*;
var cnf = NumberFormat.getCompactNumberInstance(
Locale.GERMAN, NumberFormat.Style.LONG );

cnf.format(1L << 10); // ==> 1 Tausend
cnf.format(1920); // ==> 2 Tausend
cnf.format(1L << 20); // ==> 1 Million
cnf.format(1L << 30); // ==> 1 Milliarde
```

Listing 8

right set of tradeoffs between complexity and expressiveness, or that we've explored enough of the design space to be confident that the current design is the best we can do." – Brian Goetz *[6]*.

Trotzdem können wir bereits jetzt einen Blick darauf werfen, was uns in einem der nächsten Java Releases erwarten wird. Die Hauptprobleme bei den klassischen Java Strings sind der nicht intuitive Einsatz von Zeilenumbrüchen („1\n2\n3") und das schlecht lesbare „Escapen" des Backslash, was insbesondere Windows-Dateipfade und reguläre Ausdrücke unnötig verkompliziert *(siehe Listing 9)*.

Raw-String-Literale interpretieren abgesehen von CRLF beziehungsweise LF keinerlei Escape-Sequenzen. Als Trennzeichen fungiert der Backtick *(siehe Listing 10)*. Übrigens kann ein Raw-String-Literal auch mit mehrfachen Backticks umschlossen sein *(Listing 11)*. Das ermöglicht die Verwendung von Backticks im Text, ohne sie „escapen" zu müssen. Weitere Informationen finden sich in einem Blog-Post *[7]*.

```
var dir = "c:\\tmp";
var regexp = "\\d{2}\\.\\d{3}");
```

Listing 9

```
var string = `1. Zeile \d{2}\.\d{3}
2. Zeile: c:\windows`
```

Listing 10

```
var textWithBackticks = ``Enthält ` im Inhalt``
```

Listing 11

Fazit

Möglicherweise ist man weiterhin auf Java 8 beschränkt oder möchte bis 2021 zunächst die aktuelle LTS (Long Term Support) Version JDK 11 verwenden. Trotzdem empfiehlt es sich, das JDK 12 bereits heute herunterzuladen *[8]* und sich mit den Neuerungen, insbesondere den Switch Expressions, vertraut zu machen. Alternativ kann man auch die neuen halbjährlichen Releases des Oracle OpenJDK mitgehen. Dann muss man zwar häufig migrieren, vermeidet jedoch die aufwendigen Big-Bang-Umstiege auf eine neue Java-LTS-Version alle drei Jahre. Zudem erhält man so für das freie Oracle OpenJDK noch für ein halbes Jahr kostenlose Updates. Denn das Oracle JDK 11 darf in Produktion nur noch kostenpflichtig eingesetzt werden. Auch für das Oracle JDK 8 gibt es seit Januar 2019 keine freien Updates mehr. Möchte man lieber über mehrere Jahre eine aktuelle, aber freie JDK-LTS-Version mit regelmäßigen Updates einsetzen, muss man auf alternative OpenJDK-Distributionen wie AdoptOpenJDK oder Amazon Corretto ausweichen. Auf jeden Fall scheint Oracles Idee mit den halbjährlichen Major-Releases und damit regelmäßigen – wenn auch wenigen – Änderungen aufzugehen. Bisher wurden diese kleinen Releases sehr pünktlich geliefert und so dürfen wir bereits auf die nächsten Neuerungen von Java 13 im Oktober 2019 gespannt sein.

Referenzen

[1] JDK-12-Projektseite: *https://openjdk.java.net/projects/jdk/12/*
[2] Release Notes: *http://jdk.java.net/12/release-notes*
[3] ZGC: *https://wiki.openjdk.java.net/display/zgc/Main*
[4] API-Änderungen: *https://github.com/AdoptOpenJDK/jdk-api-diff*
[5] Unicode Common Locale Data Repository: *https://unicode.org/reports/tr35/tr35-numbers.html#Compact_Number_Formats*
[6] Drop Raw String Literals: *https://mail.openjdk.java.net/pipermail/jdk-dev/2018-December/002402.html*
[7] Raw String Literals: *https://blog.oio.de/2019/03/04/raw-string-literals-geplant-nach-java-12/*
[8] JDK 12 Download: *http://jdk.java.net/12/*

Falk Sippach
falk.sippach@oio.de

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

Java: Was ändert sich? Neuerungen im April 2019

Michael Paege, Opitz Consulting

So schnell kann's gehen: Kaum ist mein Artikel „Java: Was ändert sich? Risiken und Handlungsalternativen für Java-Nutzer" in der „Java Aktuell" 3/2019 erschienen, sind einige wichtige Änderungen durch Oracle zu vermelden, die ich dem Leser nicht vorenthalten möchte. Daher dieser Nachtrag mit den aktuellen Änderungen.

Wie im vorherigen Artikel beschrieben, hatte Oracle zunächst zwei unterschiedliche Kriterien für eine Subskriptionspflicht bei Oracle JDK 8 und Oracle JDK 11 formuliert:

- Bei Oracle JDK 8 hieß es, private Nutzung bleibe bis 31. Dezember 2020 frei, dagegen werde kommerzielle Nutzung ab April 2019 mit dem dann erscheinenden Build von Oracle JDK 8 und allen folgenden Builds kostenpflichtig.
- Für das Oracle JDK 11 jedoch hatte Oracle formuliert, dass die Nutzung für Entwicklung, Test, Demo und Proof of Concept kostenfrei sei, für alles andere aber die Subskription benötigt werde. Somit wäre eigentlich auch eine private produktive Nutzung von Oracle JDK 11, z. B. für Banking-Applikationen, subskriptionspflichtig geworden.

Wenn man nun das Oracle JDK 8, Build 211, herunterladen möchte, wird man auf ein Lizenz-Update hingewiesen, das besagt, dass das „Oracle Technology Network License Agreement for Oracle Java SE" *[1]*

zu akzeptieren sei. Auf diesen OTN-Java-Lizenzvertrag verweist ein ähnlich formulierter Hinweis beim Download des Oracle JDK 11. Somit verweisen sowohl Oracle JDK 8 ab Build 211 als auch Oracle JDK 11 für Downloads ab dem 16. April 2019 auf denselben Lizenzvertrag. Die oben beschriebenen Unterschiede und Interpretationsmöglichkeiten sind somit erstmal beseitigt. Das ist gut.

Was steht nun im neuen OTN-Java-Lizenzvertrag *[3]*? Die entscheidenden Formulierungen lauten:

„Oracle grants you a nonexclusive, nontransferable, limited license to use the Programs, subject to the restrictions stated in this Agreement and Program Documentation, only for:

(i) Personal Use,
(ii) Development Use,
(iii) Oracle Approved Product Use, and/or
(iv) Oracle Cloud Infrastructure Use. "

Auch diese Formulierungen sind nicht vollständig selbsterklärend, werden aber im Vertrag im Rahmen der Begriffsdefinitionen näher erläutert:

"**Personal Use refers** to an Individual's use of the Programs solely on a desktop or laptop computer under such Individual's control only to run Personal Applications. **Personal Applications** refers to Applications designed for individual personal use only, such as games or personal productivity tools." Dieser Bereich ist meiner Ansicht nach ausreichend erläutert. Die kostenfreie Nutzung des Oracle JDK für private/persönliche Zwecke auf einem Client Computer (Desktop/Laptop) ist dadurch abgedeckt.

"**Development Use refers** to Your internal use of the Programs to develop, test, prototype and demonstrate Your Applications. For purposes of clarity, the "to develop" grant includes using the Programs to run profilers, debuggers and Integrated Development Environments (IDE Tools) where the primary purpose of the IDE Tools is profiling, debugging and source code editing Applications." Auch diese Formulierung lässt meines Erachtens keine Fragen offen.

"**Oracle Approved Product Use** refers to Your internal use of the Programs only to run: (a) the product(s) identified as Schedule A Products at *[3]*; and/or (b) software Applications developed using the products identified as Schedule B Products at *[3]* by an Oracle authorized licensee of such Schedule B Products. If You are unsure whether the Application You intend to run using the Programs is developed using a Schedule B Product, please contact your Application provider."

"**Oracle Cloud Infrastructure Use** ("OCI Use") refers to Your use of the Programs on Oracle's Cloud Infrastructure with the Oracle Cloud Infrastructure products identified in the Oracle PaaS and IaaS Universal Credits Service Descriptions available at *[4]* during the period in which You maintain a subscription for such Oracle Cloud Infrastructure products." Hierdurch ist die Java-Nutzung bei PaaS (Platform as a Service) und IaaS (Infrastructure as a Service) inkludiert, wenn der PaaS/IaaS-Service Java benötigt.

Fazit

Auch wenn sie spät vorgenommen wurden, durch diese Änderungen im OTN-Vertragstext hat Oracle für Java SE etliche Möglichkeiten für Fehlinterpretationen beseitigt und diverse noch offene Fragen der letzten Monate zur Java-SE-Subskriptionierung beantwortet. Vor allem ist zu begrüßen, dass die Nutzung von Oracle JDK 8 und Oracle JDK 11 ab April 2019 auf demselben Vertragstext basiert und kein Java-Kunde durch diese Änderungen im Vergleich zu den bisherigen Ankündigungen oder Vertragstexten schlechter gestellt wird.

Quellen

[1] *https://www.oracle.com/technetwork/java/javase/terms/license/javase-license.html*
[2] *https://www.oracle.com/technetwork/java/javase/terms/license/javase-license.html*
[3] *https://java.com/oaa*
[4] *http://oracle.com/contracts*

Michael Paege
michael.paege@opitz-consulting.com

Michael Paege studierte BWL mit Schwerpunkt Wirtschaftsinformatik an der Westfälischen Wilhelms-Universität in Münster. Bereits während des Studiums hat er Anwendungssysteme mit den relationalen Datenbanken Informix und vor allem Oracle entwickelt. Nach dem Studium 1993 begann er bei OPITZ CONSULTING als PL/SQL- und Forms/Reports-Entwickler und übernahm bald Projektleitungsaufgaben. 1999 gründete er die Hamburger Niederlassung von OPITZ CONSULTING, die er bis 2010 leitete. Parallel dazu wurde das Thema Lizenzvertrieb und Lizenzberatung zu einem immer größeren Aufgabenbereich, den er in 2010 zu seinem Hauptaufgabenfeld machte. Ehrenamtlich ist Michael Paege seit langem in der DOAG aktiv und gründete das Competence Center Lizenzfragen, das er bis heute leitet.

Unbekannte Kostbarkeiten des SDK Heute: Just-in-Time-Compilation

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des SDK vorstellen: die unbekannten Kostbarkeiten.

Der Just-in-Time-Compiler leistet seit zwei Dekaden Unglaubliches: Wir müssen uns über performanten Code (fast) keine Gedanken machen und haben trotzdem eine meist hoch performante Ausführung unserer Systeme garantiert. Der Just-in-Time-Compiler (kurz JIT) arbeitet im Verborgenen und erspart uns aufgrund der JVM-Ergonomics eine manuelle Konfiguration. Unter Ergonomics versteht man die von der JVM an Anwendung und Umgebung angepassten, automatisch durchgeführten Optimierungen zur Performanzsteigerung. Dazu gehören Einstellungen für die Garbage Collection, die Heap-Größe, aber eben auch der JIT-Compilation. Dieser Artikel gibt einen kleinen Einblick in die sehr unübersichtliche Welt der JIT-Compilation und hinterlässt (hoffentlich) das gute Gefühl, eine sehr reife Ablaufumgebung für unsere Applikationen zu haben.

Unbekannte Kostbarkeiten reloaded

Von 2011 bis 2015 haben wir in unserer Kolumne *Unbekannte Kostbarkeiten* insgesamt 14 Artikel in der Java aktuell veröffentlicht [1]. Mit dem Release von Java 8 änderte sich sowohl das Leserinteresse als auch die Publikationslandschaft: Features wie Lambdas und Streams dominierten die Medien und Konferenzen, der Rest war

Datentyp	Flag-Name	Flag-Wert	Kategorie
bool	C1ProfileInlinedCalls	true	C1 product
bool	DebugInlinedCalls	true	C2 diagnostic
intx	FreqInlineSize	325	pd product
bool	IncrementalInline	true	C2 product
bool	Inline	true	product
ccstr	InlineDataFile	-	pd product
intx	InlineSmallCode	2000	pd product
bool	InlineSynchronized-Methods	true	C1 product
intx	MaxInlineLevel	9	product
intx	MaxInlineSize	35	product
intx	MaxRecursiveInlineLevel	1	product
intx	Tier23InlineeNotify-FreqLog	20	product
bool	UseInlineCaches	true	product
bool	UseInlineDepthFor-SpeculativeTypes	true	C2 diagnostic
bool	UseOnlyInlinedBimorphic	true	C2 product

Tabelle 1: VM-Flags für das Inlining

eher außen vor. Auch mit Java 9 war eine Konzentration der Medien auf das Thema *Modul-System* klar zu erkennen. Andere Themen waren von weniger Interesse. Wir haben daher mit unseren *„Unbekannten Kostbarkeiten"* eine kleine Pause eingelegt. Jetzt, mit den schnellen Versionswechseln von Java 10 und danach, ist die Zeit reif, sich wieder mit den unbekannten Kostbarkeiten zu beschäftigen: Zu groß ist die Gefahr, dass wir tolle Features, die uns Java zur Verfügung stellt, nicht verwenden, weil wir sie nicht kennen.

Unser heutiger, erster Beitrag in den *„Unbekannten Kostbarkeiten reloaded"* besitzt eine Sonderstellung. Wir müssen nichts tun, um den JIT-Compiler zu verwenden, er ist immer da und funktioniert auch ohne unsere Mitwirkung ausgesprochen gut. Dieser Artikel ist also eher ein Blick hinter die Kulissen als ein Tipp oder eine Anleitung für die explizite Verwendung.

Die Anfänge des JIT

Als Java im Jahr 1995 das Licht der Welt erblickte, veröffentlichte Sun das Whitepaper *„The Java Language: An Overview"*, in dem man die folgende Aussage lesen konnte: *„The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on"*. Obwohl die Java-Implementierung 1.0 des Jahres 1995 den Java-Quellcode in Byte-Code kompilierte und dieser dann in der JVM interpretiert wurde, existierte die Idee der Just-in-Time-Kompilierung bereits. Im Dezember 1998 war die Zeit dann reif. Mit der Version 1.2 erhielt die JVM einen JIT-Compiler und Sun veröffentlichte folgende Pressemitteilung: *„Java HotSpot Performance Engine turbocharges the Java 2 platform – performance is no longer an issue"*. Die JVM wurde mit dem Zusatz „Hot Spot" versehen, da die JIT-Kompilierung nur für „heiße", also besonders häufig aufgerufene Code-Stücke erfolgte. Verglichen mit heute war das damalige Wissen um die effiziente Implementierung von JIT-Mechanismen geradezu prähistorisch. Was der JIT-Compiler heute alles leistet, schauen wir uns nun an. Wir können allerdings nur an der Oberfläche kratzen und beschränken uns auf ein paar wenige, wie wir hoffen aber interessante Themen: Inlining, On-Stack Replacement und Escape Analysis.

Inlining

Unter Inlining versteht man das Ersetzen des Codes eines Methodenaufrufs durch den Methodenrumpf. Dies erspart zur Laufzeit den Verwaltungsaufwand für die Methodenaufrufe (Stichwort Call Stack) und ermöglicht weitere Optimierungen, da der neu kombinierte Code eventuell Vereinfachungen zulässt, die sonst nicht möglich gewesen wären. Moderne JVMs unterstützen das Inlining in verschiedenen Ausprägungen, die durch Kommandozeilen-Flags parametrisiert sind.

Die *Tabelle 1* zeigt die Ausgabezeilen eines VM-Aufrufs mit den Parametern `-XX:+UnlockDiagnosticVMOptions` und `-XX:+PrintFlagsFinal`, die „Inline" enthalten. Also die VM-Flags, die etwas mit Inlining zu tun haben.

Die erste Spalte gibt den Datentyp innerhalb der VM an, die zweite Spalte den Namen des Flag. Die dritte Spalte nennt den aktuellen Wert des Flag und die vierte Spalte dessen Kategorie. Offizielle VM-Optionen, die also auf allen Java-VMs definiert sind, sind mit "product" gekennzeichnet, plattformabhängige Flags mit "pd" (platform dependent), da sie sich zum Beispiel auf Intel/AMD und ARM unterscheiden. Debugging-relevante Flags sind mit "diagnostic" gekennzeichnet. Die Unterscheidung, zu welchem JIT-Compiler das Flag gehört, erfolgt über C1 (Client) und C2 (Server). Dazu später mehr.

Wie ist das zu interpretieren beziehungsweise zu nutzen? Das Inlining häufig aufgerufener Methoden wird bis zu einer Größe von 325 Bytes vorgenommen (Flag `FreqInlineSize`). Für selten aufgerufene Methoden wird das Inlining nur durchgeführt, falls sie kleiner als 35 Bytes (Flag `MaxInlineSize`) sind. Soll dies auch bei größeren Methoden durchgeführt werden, so muss die VM zum Beispiel mit der Option `XX:MaxInlineSize=70` aufgerufen werden, was die Methodengröße verdoppelt. Wie wir am Anfang des Artikels bereits erwähnt haben, ist die Verwendung der JIT-Optionen für die meisten Entwickler nicht empfehlenswert, da die JVM-Ergonomics in der Regel besser sind als unsere vermeintlichen Verbesserungsversuche. Hunt und John *[2]* raten sogar explizit von der Verwendung der Optionen ab.

On-Stack Replacement

Anders als beim Inlining werden ganze Methoden vollständig in Maschinen-Code übersetzt, wenn sie „hot" sind, also besonders häufig aufgerufen wurden. Im Client-Compiler ist der Schwellenwert 1.500, beim Server-Compiler 10.000 Aufrufe. Wird eine Methode derart übersetzt, erfolgt der *nächste* Aufruf der Methode über den Maschinen-Code, nicht den Byte-Code. Was passiert aber bei sehr lange laufenden Schleifen? Hier kann nicht auf den nächsten Aufruf des Schleifen-Codes gewartet werden, der eventuell in sehr weiter Zukunft liegt oder sogar nie erfolgt. Der Übergang von der Ausführung vom Byte-Code einer Schleife zum Maschinen-Code wird *„On-Stack Replacement"* genannt. Neben einem Zähler für die Aufrufhäufigkeit einer Methode gibt es auch einen Zähler, der die Rücksprünge innerhalb einer Schleife zählt, den sogenannten *„BackEdge Counter"*.

Wird dieser übersprungen, kompiliert der JIT-Compiler nicht eine gesamte Methode, sondern nur die Schleife, und beginnt die Ausführung des kompilierten Codes. Die Schwellenwerte für Methodenaufrufe und Schleifenrücksprünge lassen sich mit `XX:CompileThreshold=N` und `XX:BackEdgeThreshold=N` setzen.

Escape Analysis

Ein alloziertes Objekt *„escaped"*, wenn es nach seiner Allokation in einem Thread A durch einen anderen Thread B gesehen (zugegriffen) werden kann. Falls dies nicht der Fall ist, kann der Compiler eine oder mehrere der folgenden Optimierungen durchführen:

- Object Explosion: Anlegen der Felder des Objekts an verschiedenen Stellen und potenzieller Verzicht auf die Objektallokation selbst
- Scalar Replacement: Speichern der Felder des Objekts in CPU-Registern
- Thread Stack Allocation: Speichern der Felder des Objekts in einem Stack-Frame
- Synchronisierung eliminieren
- GC Read/Write-Barriers eliminieren

Die Escape Analysis wurde in Java 6u14 eingeführt und ist seit Java 7u4 der Default. Gesteuert wird sie über das VM-Flag `XX:+DoEscapeAnalysis`. Das Flag ist boolesch und wird mit `+` ein-, mit `-` ausgeschaltet.

C1, C2, Graal

Nachdem wir uns drei zentrale Optimierungsmöglichkeiten des JIT-Compilers angeschaut haben, wollen wir einen Blick auf den JIT-Compiler selbst werfen. Es gibt nicht nur einen, es gibt drei: C1, C2 und Graal. C1 und C2 sind der sogenannte Client- und der Server-Compiler und seit vielen Jahren in der VM enthalten. Die zentrale Idee der Differenzierung ist, dass Client-Anwendungen – zu der Zeit der Einführung zum Beispiel AWT-("Abstract Window Toolkit"), später Swing-Anwendungen auf dem Client – schnell hochfahren müssen, um den Benutzer nicht warten zu lassen. Der Schwellenwert für die Methoden-Compilation beträgt daher 1.500 Methodenaufrufe und findet ohne weiteres Monitoring der Aufrufe statt. Bei Server-Anwendungen ist die Startzeit nicht so kritisch, sodass zunächst Methodenaufrufe beobachtet und analysiert werden und der JIT darauf basierend dann besseren Code erzeugen kann als ohne diese Monitoring-Informationen. Hier beträgt der Schwellenwert 10.000 Methodenaufrufe. Der Client-Compiler war zunächst der Client-VM, also der 32-Bit-VM vorbehalten. Genauso verhielt es sich mit dem Server-Compiler, der in Server-VMs mit 64 Bit enthalten war. Später wurden beide Compiler zugleich verbaut und die sogenannte *Tiered Compilation* eingeführt, bei der zunächst der Client-, später der Server-Compiler denselben Code noch einmal übersetzt hat. Dieses Verhalten ist seit Java 8 der Default.

Mit dem JEP 317 *[3]* wurde ein zunächst experimenteller JIT-Compiler, genannt *Graal*, eingeführt. Motivation für Graal war zum einen die schlechte Wartbarkeit des in C++ geschriebenen C2-Compilers, zum anderen die Realisierung eines Ahead-of-Time-Compilers mit JEP 295 *[4]*. Graal ist seit JDK 9 in Linux, seit JDK 10 auch in Windows verfügbar. Er ist jedoch als experimentell gekennzeichnet und muss explizit freigeschaltet werden. Hierzu wurde mit JEP 243 *[5]* ein VM-Compiler-Interface realisiert, um beliebige JIT-Compiler über einen Plug-in-Mechanismus der VM zur Verfügung zu stellen. Der JEP ist mit *JVM Compiler Interface* überschrieben, sodass die entsprechende Option `XX:+UseJVMCICompiler` lautet. Durch den experimentellen Charakter des Compilers müssen die experimentellen Optionen zusätzlich durch `XX:+UnlockExperimentalVMOptions` verfügbar gemacht werden.

Zusammenfassung

Wir haben uns die drei Just-in-Time-Compiler C1, C2 und Graal angeschaut. Man muss allerdings gestehen, dass dies nur sehr oberflächlich und exemplarisch an den drei Themenkomplexen Inlining, On-Stack Replacement und Escape Analysis erfolgt ist. Javas JIT-Compiler sind sehr hochentwickelte Komponenten des Java-Laufzeitsystems, die, basierend auf Ergonomics, ohne unser Zutun sehr performanten Code erzeugen. In der Regel müssen wir uns daher keine Gedanken über performanten Code machen und können uns auf performante Algorithmik und gute Architektur konzentrieren.

Referenzen

[1] *https://www.pdbm.de/unbekannte-kostbarkeiten.html*

[2] Charlie Hunt und Binu John. Java Performance. Addison-Wesley, 2012.

[3] JEP 317: Experimental Java-Based JIT Compiler. *http://openjdk.java.net/jeps/317*

[4] JEP 296: Ahead-of-Time Compilation. *http://openjdk.java.net/jeps/295*

[5] JEP 243: Java-Level JVM Compiler Interface. *http://openjdk.java.net/jeps/243*

Bernd Müller
bernd.mueller@ostfalia.de

Nach seinem Studium der Informatik und der Promotion arbeitete Bernd Müller für die IBM und die HDI Informationssysteme. Er ist Professor, Geschäftsführer, Autor mehrerer Bücher zu den Themen JSF und JPA, sowie Speaker auf nationalen und internationalen Konferenzen. Er engagiert sich im iJUG und speziell in der JUG Ostfalen.

Mit Spring, Docker und Kubernetes nach Produktion schippern

Timm Hirsens, Atlas Dienstleistungen für Vermögensberatung

Beispiele, um eine Spring-Boot-Anwendung in einem Docker-Container zu deployen, gibt es wie Sand am Meer. Aber reicht das auch für einen entspannten Produktionsbetrieb in Kubernetes? Lassen sich Docker und Java so ohne Weiteres kombinieren? Welche Fallen lauern hier? Wie bauen wir verschiedene Umgebungen mit Kubernetes auf? Wie konfigurieren wir unsere Anwendung am besten für diese?

Gründe dafür, Spring-Boot-Anwendungen auf einem Kubernetes-Cluster zu betreiben, gibt es reichlich. Damit es bei diesem Vorhaben keine bösen Überraschungen gibt, sollten einige Dinge jedoch unbedingt beachtet werden.

Das richtige Docker-Image

Damit wir mit unserer Spring-Boot-Anwendung mithilfe von Kubernetes nach Produktion schippern können, müssen wir sie zunächst in einen Container verpacken. Nach einer kurzen Google-Suche ist die vermeintliche Lösung auch schnell gefunden.

Das Ergebnis dieses Dockerfiles *(siehe Listing 1)* ist durchaus ein funktionierendes und korrektes Docker-Image. Allerdings gibt es hier auch noch einiges an Verbesserungspotenzial, wenn unser Ziel ein möglichst reibungsloser Betrieb in Produktion ist. Beginnen wir mit der ersten Zeile des Dockerfiles. Im Beispiel erben wir von dem Docker-Image `openjdk:8-jdk-alpine`. Bei den Images mit dem Namen „openjdk" handelt es sich um offizielle Images aus dem Docker Hub. Grundsätzlich macht man hier also nicht allzu viel falsch. Das Basis-Image aus dem Beispiel verwendet „Alpine Linux" als Unterbau, hierbei handelt es sich um eine Linux Distribution, die gerne für den Betrieb als Container verwendet wird. Sie besticht besonders durch ihre geringe Größe von lediglich rund 7 MB. Diese Größe gibt es allerdings nicht umsonst; in einem Alpine-Image ist statt einer vollständigen GNU/Linux-Umgebung nur „busybox" enthalten und statt der C-Bibliothek „glibc" enthält Alpine Linux die Bibliothek „muslc". Besonders bei der Arbeit mit JNI (Java Native Interface) kann dieser Unterschied an Bedeutung gewinnen.

Wenn ein Container dann doch mal durch einen Debugger betrachtet werden soll, ist es durchaus nützlich, wenn man sich in ihm auch zurechtfindet und die nötigen Tools schon installiert sind. Daher ergibt es durchaus Sinn, das Container-Image etwas größer werden zu lassen. Als Basis eignet sich beispielsweise ein CentOS. CentOS ist im Vergleich zu Alpine Linux mit rund 75 MB zwar wesentlich größer, allerdings wiegt ja selbst die JRE/JDK 100 bis 250 MB. Daher sind die Einsparungen auf Betriebssystem-Ebene in den meisten Fällen eher zu vernachlässigen. Ein einfaches Basis-Image für unsere Java-Anwendungen könnte daher so aussehen *(siehe Listing 2)*.

Wir erben von einem aktuellen CentOS-Image und installieren dort die aktuellste Java-Version. Wir verwenden hier das JDK, zum reinen Betrieb reicht jedoch auch eine JRE. Anschließend räumen wir noch ein wenig auf, damit das Image nicht unnötig groß wird. Die Angabe der Versionsnummer beziehungsweise des Tags in der `FROM`-Klausel

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Listing 1

```
FROM centos:7.6.1810

LABEL maintainer="Timm Hirsens <mail@timmhirsens.de>"

RUN yum install java-11-openjdk-headless-11.0.2.7 java-11-openjdk-devel-11.0.2.7 --setopt=tsflags=nodocs -y \
    && rm -rf /var/cache/yum \
    && yum clean all
```

Listing 2

kann bei Dockerfiles weggelassen werden. Um fiese Überraschungen zu vermeiden, ist es sinnvoll, bei der Arbeit mit Docker-Images immer eine konkrete Versionsnummer anzugeben. Selbst bei der Verwendung eines Image als Build-Node sollte die Versionsnummer angegeben werden. Das bedeutet natürlich auch, dass hier regelmäßig manuelle Updates stattfinden müssen, um eventuelle Sicherheitslücken zu schließen.

Das Docker-Overlay-Dateisystem nutzen

Da wir jetzt ein gutes Basis-Image erstellt haben, wollen wir nun natürlich auch unsere Anwendung in einen Container verfrachten. In unserem Beispiel wurde dafür die einfachste Variante gewählt, es wurde unser Spring Boot Fat-JAR mithilfe des `COPY`-Befehls in das Docker-Image kopiert. Grundsätzlich spricht auch gegen dieses Vorgehen nichts, trotzdem können wir hier noch optimieren.

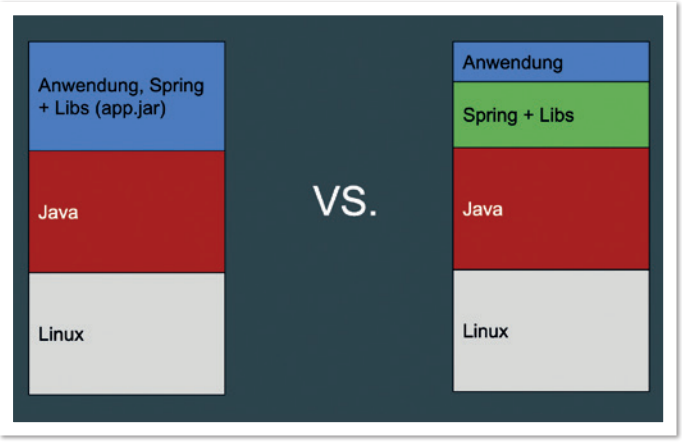


Abbildung 1: Vereinfachte Aufteilung des Container-Image in Schichten (Quelle: Timm Hirsens)

```
FROM mycompany.com/openjdk:11.0.2.7

VOLUME /tmp

EXPOSE 8080
EXPOSE 1099

ARG DEPENDENCY=build/dependency
COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/org /app/org
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app
```

Listing 3

Docker-Images werden in Schichten aufgebaut, wobei jede Zeile in unserem Dockerfile eine neue Schicht bildet. Hierbei handelt es sich vor allem um eine Optimierungsmaßnahme für den Transport von Docker-Images. Wird ein Docker-Image aus einer Docker-Registry heruntergeladen, müssen nur die Schichten geladen werden, die der Anfrager noch nicht lokal kennt. Starten wir einen Docker-Container in unserem Kubernetes-Cluster, muss hier auch das Image aus der Registry geladen werden, aber eben nur die Schichten, die auf dem Kubernetes-Node noch nicht bekannt sind. Teilen wir unser Image also so ein, dass die Schicht mit den meisten Änderungen möglichst klein ist, können wir sowohl unsere Build- als auch unsere Deployment-Zeiten verkürzen. Das können wir erreichen, indem wir das Spring Boot Fat-JAR wieder entpacken und folgende Schichten bilden *(siehe Abbildung 1)*:

- Die Spring-Boot-Bibliotheken und ihre Abhängigkeiten
- Unsere Metainformationen („META-INF“)
- Unsere statischen Ressourcen
- Unsere eigenen Klassen

Somit bildet der Part unseres Fat-JAR, der sich am häufigsten ändert, unsere eigenen Klassen, die letzte Schicht im Docker-Image *(siehe Listing 3)*.

Kubernetes Deployment konfigurieren

Wenn wir unseren Container gebaut haben, kommt der nächste Schritt in Richtung Produktion, unser Container wird „verschifft“. Damit ein Container innerhalb eines Pod in einem Kubernetes-Cluster gestartet wird, gibt es verschiedene Möglichkeiten, diesen zu konfigurieren. In den meisten Fällen ist die Verwendung eines „Deployments“ die sinnvollste Variante. In einem solchen wird unter anderem angegeben, wie viele Instanzen („Replicas“) eines Containers gewünscht sind, wie diese gestartet werden sollen, welche Ressourcen zur Verfügung stehen und vieles mehr *(siehe Listing 4)*.

Beschränkung von Ressourcen

Es gibt zwei wichtige Optionen bei Kubernetes-Deployments, die für einen produktiven Betrieb auf jeden Fall angegeben werden sollten.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-auf-kubernetes
  labels:
    app: spring-auf-kubernetes
spec:
  replicas: 2
  selector:
    matchLabels:
      app: spring-auf-kubernetes
  template:
    metadata:
      labels:
        app: spring-auf-kubernetes
    spec:
      containers:
        - name: spring-auf-kubernetes
          image: fr1zle/spring-auf-kubernetes:2cf65dbe
          ports:
            - containerPort: 8080
          args: [
            "/usr/bin/java",
            "-Djava.security.egd=file:/dev/./urandom",
            "-cp",
            "app:app/lib/*",
            "org.springframework.boot.loader.JarLauncher"
          ]
          readinessProbe:
            httpGet:
              path: /actuator/health
              port: 8080
            initialDelaySeconds: 30
            periodSeconds: 10
            timeoutSeconds: 3
          livenessProbe:
            httpGet:
              path: /actuator/info
              port: 8080
            initialDelaySeconds: 30
            periodSeconds: 10
            timeoutSeconds: 3
          resources:
            requests:
              cpu: 100m
              memory: 800Mi
            limits:
              cpu: 1
              memory: 800Mi
```

Listing 4

Dazu gehört als Erstes die Beschränkung der Ressourcen. Kubernetes bietet uns die Möglichkeit, im Gegensatz zu einem klassischen Application-Server, Ressourcen für einzelne Anwendungen sehr genau zu beschränken. Im Deployment geben wir an, wie viele CPU- und Arbeitsspeicherressourcen unserem Pod zur Verfügung stehen sollen. Hierzu stehen zwei Kategorien zu Verfügung: `Requests` und `Limits`. Für Ressourcen, die in `Requests` definiert werden, gibt der Kubernetes-Cluster eine Garantie, dass diese zum Start des Pod und zu seiner gesamten Lebensdauer auch zur Verfügung stehen.

Die Ressource-CPU wird in Tausenderschritten angegeben, „1000m" entspricht dabei einer CPU. Ein Pod, der „100m" CPU anfordert, erhält also ein Zehntel der CPU-Zeit eines Pod, der „1000m" oder „1" CPU anfordert. Die Anzahl der CPUs, die bei den `Requests` angegeben wird, berücksichtigt die JVM bei der Berechnung der verfügbaren CPU-Kerne. Hierbei wird auf den nächsten ganzen CPU-Kern gerundet. Die so errechneten CPU-Kerne werden von der JVM für die Berechnung der internen Threads (also für Just-In-Time Compilation, Garbage Collection usw.) herangezogen. Mit den `Limits` definieren wir, wie viel CPU unser Pod maximal verbrauchen darf. Wird mehr CPU verwendet, drosselt Kubernetes den Pod entsprechend. So ist auch im Fall eines Fehlers sichergestellt, dass eine Anwendung einer anderen nicht die CPU-Zeit „klauen" kann.

Für den Betrieb einer kleineren Spring-Boot-Anwendung reichen für gewöhnlich „100m" CPU. Der Start einer Spring-Boot-Anwendung kann allerdings, je nach verwendeten Bibliotheken, einiges an CPU benötigen. Um nicht unnötig viele Ressourcen zu verbrauchen, aber dennoch einen schnellen Start der Anwendung zu ermöglichen, geben wir unserem Pod ein `Request` von „100m" CPU und ein `Limit` von „1" CPU.

Die Berechnung der Arbeitsspeicherressourcen ist deutlich einfacher, sie werden einfach in Megabyte angegeben. Auch hier schaut die JVM seit neueren Versionen genauer hin; die Anzahl von Megabyte, die wir in den „Limits" definieren, werden für die Berechnung der einzelnen Speicherbereiche im Heap der JVM verwendet. Eine einfache Spring-Boot-Anwendung kommt mit etwa 800 MB gut aus. Die 800 MB werden sowohl bei `Requests` als auch bei den `Limits` eingetragen. Da die JVM die `Limits` zur Berechnung verwendet, müssen wir mit den `Requests` sicherstellen, dass der gewünschte Speicher auch zur Verfügung steht.

Java und cgroups

Die verfügbaren Ressourcen eines Docker-Containers werden über sogenannte „cgroups" (Control-Groups) gesteuert. In älteren Java-Versionen beachtet die JVM allerdings nicht die Ressourcen, die ihr durch eine „cgroup" zugeteilt werden, sondern die des Host-Computers. Hat also ein Kubernetes-Node 32 CPU-Kerne, berechnet die JVM anhand dieser Anzahl unter anderem, wie viele Kerne sie für die Garbage Collection verwenden kann. Damit das nicht passiert, sollte eine aktuelle Java-Version verwendet werden. Ab Java 11 oder Java 8 Update 191 verwendet die JVM standardmäßig die Einträge aus den „cgroups" für die Berechnung der CPU-Kerne und des verfügbaren Heap. Lässt sich der Einsatz einer älteren Java-Version nicht vermeiden, sollte unbedingt mindestens die Option `-XX:+UseCGroupMemoryLimitForHeap` gesetzt werden. Außerdem sollten die Threads des Garbage Collector entsprechend den verfügbaren Ressourcen konfiguriert werden.

Kubernetes-Probes konfigurieren

Die Kubernetes-Probes sind das zweite Feature, das in Produktion auf jeden Fall aktiviert werden sollte. Mit ihrer Hilfe wird sichergestellt, dass Anwendungen auch nach dem Deployment möglichst unterbrechungsfrei weiterlaufen. Die Aufgabe der `Liveness Probe` ist es, sicherzustellen, dass der in einem Container gestartete Prozess noch läuft und auf Anfragen reagiert. Ist dies nicht so, wird der entsprechende Pod gelöscht und ein neuer, identischer Pod aufgebaut. Die `Readiness Probe` hingegen stellt sicher, dass die Anwendung auch bereit ist, Anfragen entgegenzunehmen. An dieser Stelle wird sichergestellt, dass auch die externen Abhängigkeiten, wie eine Datenbank, verfügbar sind und erreicht werden können. Ist das nicht der Fall, wird der Pod aus dem Loadbalancing des Service genommen und erhält so lange keine Anfragen mehr, bis es wieder positive Antworten von der `Readiness Probe` gibt.

Beim Betrieb einer Spring-Boot-Anwendung in Kubernetes eignen sich besonders die `Actuator`-Endpunkte für die genannten Probes. Viele Beispiele verwenden hier den `Health`-Endpunkt für

beide Probes, was im produktiven Betrieb für die eine oder andere negative Überraschung sorgen kann. Da die beiden Probes ganz unterschiedliche Aufgaben erfüllen, sollte auch bei den `Actuator`-Endpunkten unterschieden werden. Für die `Liveness Probe` empfiehlt es sich, einen Endpunkt zu verwenden, der möglichst statisch eine positive Antwort (HTTP 200) liefert. Von den Spring-Boot-`Actuator`-Endpunkten eignet sich hier vor allem der Info-Endpunkt (`/actuator/info`); dieser liefert, sobald die Anwendung läuft, immer eine Antwort.

Für die Kubernetes „Readiness Probe" eignet sich der „Health"-Endpunkt. Dieser liefert für gewöhnlich einen Fehler, sobald eine Komponente, auf der eine Applikation eine Abhängigkeit hat, nicht mehr zur Verfügung steht. Üblicherweise zählen dazu Datenbank, Kafka-Cluster und externe Services, die für den Betrieb der Anwendung vonnöten sind. Steht eine der Komponenten nicht zur Verfügung oder ist für die Anwendung nicht erreichbar, erhält sie keinen weiteren Traffic mehr durch den Service. In den meisten Fällen ist dieses Verhalten auch so gewünscht, bei einigen Bibliotheken sollte allerdings darauf geachtet werden, ob diese einen eigenen Health Check registrieren. Wird beispielsweise ein Circuit Breaker aus der Bibliothek „resilience4j" eingesetzt, so wird standardmäßig auch ein Health Check hinzugefügt – mit der Folge, dass sobald ein Circuit geöffnet wird, die Applikation aufgrund des fehlerhaften Health Checks nicht mehr erreichbar wäre. Wenn hier ein Fallback-Szenario implementiert wurde, ist das natürlich nicht das gewünschte Verhalten. Daher sollten die angemeldeten Health Checks immer im Auge behalten werden.

Verschiedene Umgebungen mit Kubernetes

Für einen entspannten Produktionsbetrieb braucht es jedoch nicht nur eine stabile Produktionsumgebung, sondern auch noch mindestens eine zusätzliche Testumgebung, auf der Änderungen an den Applikationen getestet werden können. Bei der Verwendung von Kubernetes gibt es grundsätzlich zwei verschiedene Varianten, das Konstrukt „Umgebung" abzubilden. Die erste Variante ist die Lösung über sogenannte „Namespaces". In einem Kubernetes-Cluster können über Namespaces Ressourcen und Rechte eingeschränkt werden, außerdem können Services aus einem Namespace, je nach Konfiguration, nicht ohne Weiteres auf den Service eines anderen Namespace zugreifen. Eine Testumgebung ließe sich also durch einen zusätzlichen Namespace abbilden, hiervon ist allerdings eher abzuraten.

Sinnvoller ist es, für jede Umgebung einen eigenen Kubernetes-Cluster zu erstellen. Bei dieser Variante ist es quasi unmöglich, dass ein Fehler in der Testumgebung ernsthafte Auswirkungen auf die Produktionsumgebung hat. Nebenbei eignet sich die Testumgebung so auch für den Test eines Kubernetes-Updates oder dazu, um den Austausch einer Kubernetes-Komponente zu testen.

Staging und Konfiguration mit Spring Boot

Wenn unsere Anwendung nun auf mehreren Umgebungen laufen soll, muss sie sich natürlich auch entsprechend konfigurieren lassen. Glücklicherweise bringt auch hier Spring Boot schon einiges mit. Standardmäßig erfolgt die Konfiguration von Spring-Boot-Anwendungen über Konfigurationsdateien („application.properties" oder „application.yaml"). Über Profile können wir steuern, welche Konfigurationsdateien angezogen werden sollen. Die Datei „application.properties" wird immer angezogen, wenn wir das Spring-Profil `test` aktivieren. Indem wir beispielsweise die Umgebungsvariable `SPRING_PROFILES_ACTIVE` auf den Wert `test` setzen, wird zusätzlich auch die Datei „application-test.properties" angezogen. So können wir über das Setzen einer Umgebungsvariablen steuern, auf welcher Umgebung sich die Applikation aktuell befindet.

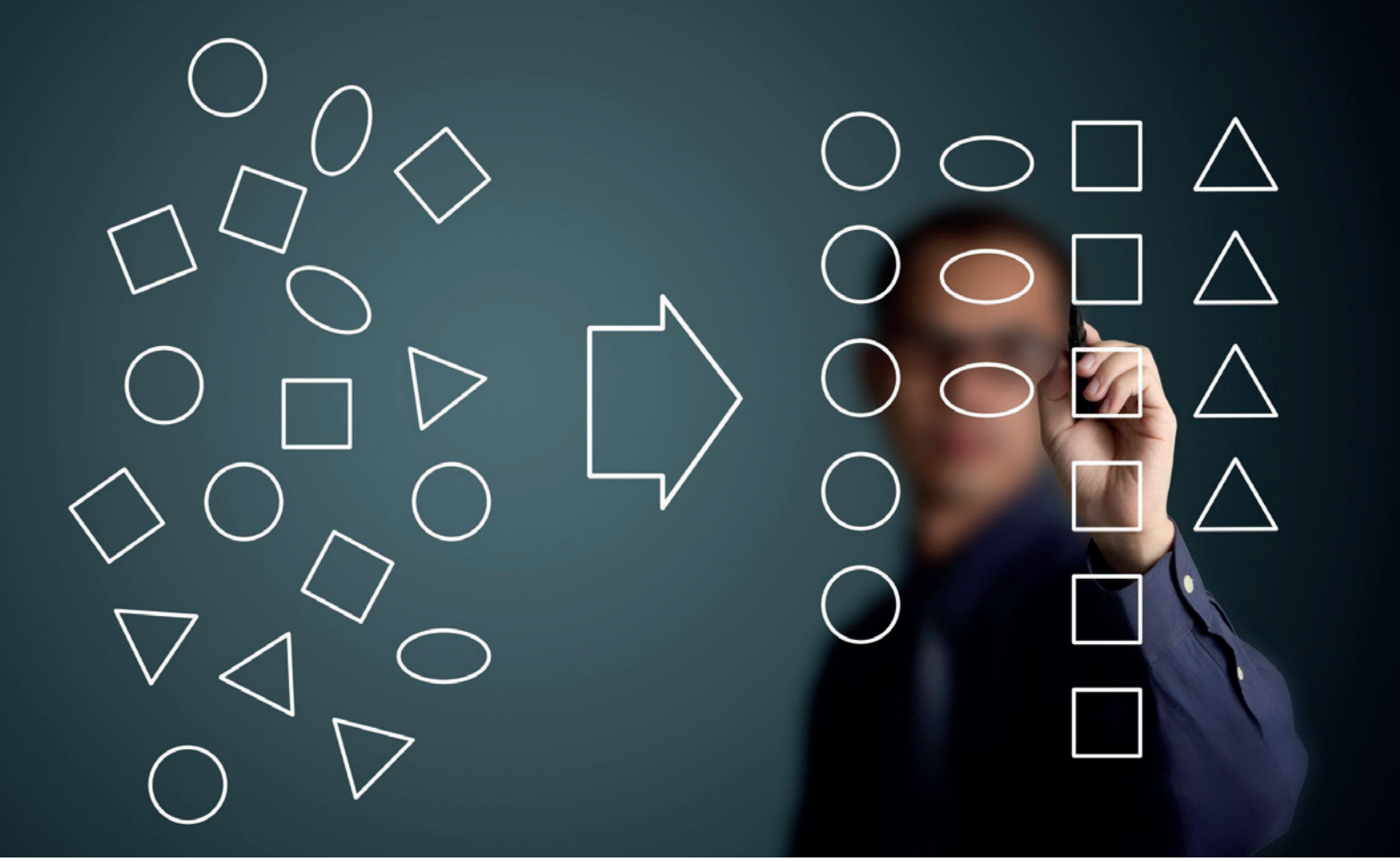
Wenn sich nur wenige Konfigurationen umgebungsabhängig ändern, reicht es unter Umständen völlig aus, diese einfach als Umgebungsvariable zu definieren. So wird aus der Property „spring.datasource.url" die Umgebungsvariable `SPRING_DATASOURCE_URL`. Die Daten aus Umgebungsvariablen haben dabei immer Vorrang vor den Daten aus der Konfigurationsdatei.

Die dritte Möglichkeit ist, dass die Konfigurationsdateien nicht in der Applikation enthalten sind, sondern über eine Kubernetes ConfigMap bereitgestellt werden. Diese ConfigMap kann dann als Volume in den Container als Mount geladen werden *(siehe „https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#add-configmap-data-to-a-volume")*. Über die Umgebungsvariable `SPRING_CONFIG_LOCATION` kann dann der Pfad zu der ConfigMap konfiguriert werden.

Fazit

Wir sehen also, mit dem reinen Befolgen einiger Tutorials erhalten unsere Spring-Boot- und Docker-Anwendungen nicht unbedingt Produktionsreife. Mit der Berücksichtigung einiger Tücken und dem Einhalten von Best Practices erreicht man diese jedoch recht schnell. Sind im Unternehmen einmal sinnvolle Standards gefunden worden, sollten diese in eine Dokumentation oder in ein Template, beispielsweise mit Helm *(siehe „https://helm.sh/docs/helm/#helm-template")*, gegossen werden, damit nicht jedes Team diese Erfahrungen erneut machen muss. Dann steht einem entspannten Segeln in Produktion nichts mehr im Wege. In meinem GitHub-Account habe ich ein Projekt, in dem einige dieser Standards in Form eines Beispiels festgehalten sind *(siehe „https://github.com/fr1zle/spring-auf-kubernetes")*.

Timm Hirsens
hi@timmhirsens.de



Eine moderne und konsistente Implementierung natürlicher Ordnung bei Java-Objekten – Teil 2

Christian Heitzmann, SimplexaCode

Eigene Java-Klassen sortierbar, also komparabel zu machen, ist eine elementare und wichtige Aufgabe. Gemessen an seiner Bedeutung wird dieses Thema in der Grundlagenliteratur jedoch nur sehr spärlich behandelt. Dieser zweiteilige Artikel zeigte in der letzten Ausgabe die Evolution der Methoden über die verschiedenen Java-Versionen hinweg auf und bot eine Code-Vorlage für eine saubere Implementierung ab Java 8 an. Nachdem der Leser darin mit der Theorie einer modernen und konsistenten Implementierung betraut wurde, wirft der hier vorliegende zweite Teil des Artikels einen Blick auf etwas realistischere Beispiele.

Die Beispielklasse, um die sich in diesem Artikel alles dreht, nennt sich `Customer` und repräsentiert einen Kunden aus einem Onlineshop. Natürlich müssen diese Kunden vergleichbar, also komparabel sein, denn sie werden ziemlich sicher irgendwo und irgendwann einmal sortiert werden müssen *(siehe Listing 1)*.

Natürlich sollen in einer solchen Klasse zu Demonstrationszwecken so viele verschiedene Attribute (englisch „fields“) wie möglich enthalten sein. Das Attribut `dateOfBirth` darf ausdrücklich `null` sein für den Fall, dass der Kunde sein Geburtsdatum nicht bekannt geben will (und der Onlineshop nicht davon abhängig ist). Das Attribut `orderCount` ist dazu gedacht, die Anzahl der Bestellungen des Kunden zu zählen, `averageOrderValue` sollte den durchschnittlichen Betrag aller Bestellungen angeben und der Boolean `premiumCustomer` ist genau dann `true`, wenn dieser Kunde aus irgendeinem Grund vom System als „Premium-Kunde“ angesehen wird, wobei die Geschäftslogik dahinter hier nicht zu interessieren

```
import java.util.*;
import java.time.*;

public final class Customer implements Comparable<Customer> {
    private String firstName;
    private String lastName;
    private LocalDate dateOfBirth;  /* May remain null. */
    private int orderCount;
    private double averageOrderValue;
    private boolean premiumCustomer;

    public Customer(String firstName, String lastName, LocalDate dateOfBirth,
                    int orderCount, double averageOrderValue, boolean premiumCustomer) {
        this.firstName = Objects.requireNonNull(firstName, "Parameter \"firstName\" is null.");
        this.lastName = Objects.requireNonNull(lastName, "Parameter \"lastName\" is null.");
        this.dateOfBirth = dateOfBirth;
        this.orderCount = orderCount;
        this.averageOrderValue = averageOrderValue;
        this.premiumCustomer = premiumCustomer;
    }
```

Listing 1

```
    public static void main(String[] args) {
        /* Create and initialize customer array. */
        Customer[] customerArray = {
                new Customer(>"Richard", >"Leblanc", LocalDate.of(1975, Month.OCTOBER, 6),
                             2, 498.30, false),
                new Customer(>"William", >"Robinson", null,
                             2, 737.35, true),
                new Customer(>"Adella", >"Wheeler", null,
                             7, 824.33, true),
                new Customer(>"Ronald", >"Hogan", null,
                             2, 297.99, false),
                new Customer(>"Joseph", >"Davis", LocalDate.of(1981, Month.NOVEMBER, 3),
                             9, 783.16, true),
                new Customer(>"Charles", >"Quintana", LocalDate.of(1989, Month.JANUARY, 2),
                             4, 748.35, true),
                new Customer(>"Graham", >"Reamer", LocalDate.of(1967, Month.MARCH, 16),
                             3, 255.96, false),
                new Customer(>"Suzanne", >"Murray", LocalDate.of(1975, Month.OCTOBER, 6),
                             2, 199.49, false),
                new Customer(>"Ronald", >"Hogan", LocalDate.of(1997, Month.NOVEMBER, 26),
                             2, 368.10, false),
                new Customer(>"Dolores", >"Falco", null,
                             3, 556.41, true)
        };
        Arrays.sort(customerArray);
        /* Print customer array. */
        for (Customer customerAct : customerArray) {
            System.out.println(customerAct);
        }
    }
    @Override
    public String toString() {
        return String.format(>"Name: %-7s %-8s | DoB: %10s | Orders: %d | AOV: %3.2f | Premium? %5b",
                             firstName, lastName, dateOfBirth,
                             Integer.valueOf(orderCount),
                             Double.valueOf(averageOrderValue),
                             Boolean.valueOf(premiumCustomer));
    }
```

Listing 2

hat. Im echten Leben würde man diese drei Eigenschaften ziemlich sicher nicht als direkte Klassenattribute darstellen, sondern vielmehr durch eine Geschäftslogik jedes Mal, wenn sie benötigt werden, bestimmen lassen. Das ist jedoch eine andere Geschichte und soll nicht vom Grundgedanken der Beispiele in diesem Artikel ablenken.

Der Konstruktor ist selbsterklärend. Es ist nur wichtig zu betonen, dass `firstName` und `lastName` auf `null` überprüft werden, weil sie niemals `null` sein dürfen, wohingegen `dateOfBirth` `null` sein darf.

Der nächste Code-Ausschnitt *(siehe Listing 2)* beinhaltet die main-Funktion, die ein Array mit zehn fingierten Kunden initialisiert, diese dann sortiert und abschließend formatiert auf der Konsole ausgibt. Zu Demonstrationszwecken haben zwei Personen den gleichen Namen (Ronald Hogan) und zwei andere Personen tei-

len sich das gleiche Geburtsdatum (Richard Leblanc und Suzanne Murray am 6. Oktober 1975).

Die `Customer`-Klasse ist `Comparable` und implementiert darum die `compareTo`-Methode. Seit Java 8 basiert die moderne Implementierung jedoch mit Vorteil auf `Comparator`, weswegen man seine `compareTo`-Methode einfach auf einen solchen `Comparator` verweisen lässt. Vor diesem Hintergrund wurden in diesem Artikel sieben verschiedene `Comparators` für die Beispiele implementiert, die allesamt folgend erläutert werden. Um nicht unnötige Compiler-Warnungen, die ungenutzten Code betreffen, zu generieren, gibt es noch einen „Comparator Switch", mit dem, wie in der fett gedruckten Zeile gezeigt *(siehe Listing 3)*, die jeweilige Sortierfunktionalität ausgewählt werden kann. Die `compareTo`-Methode schaut dann einfach auf den Switch und leitet zum ausgewählten `Comparator` weiter.

Sortieren nach Nachnamen

Der wahrscheinlich häufigste Fall besteht darin, die Kunden alphabetisch anhand ihres Nachnamens („last name") zu sortieren. Wenn sich zwei oder mehr Kunden den gleichen „last name" teilen, dann werden ihre „first names" als zweitwichtigstes Attribut behandelt. Wenn sowohl „last" als auch „first name" gleich sind, dann wird das nächstsignifikante Attribut „date of birth" herangezogen. Um es kurz zu halten, werden anschließend keine weiteren Attribute mehr untersucht *(siehe Listing 4)*. Die Ausgabe des Programms ist in *Ausgabe 1* zu sehen.

```
private static enum ComparatorSwitch {
        LAST_NAME, FIRST_NAME, DATE_OF_BIRTH,
        ORDER_COUNT, ORDER_COUNT_REVERSED, PREMIUM_CUSTOMER,
        LAST_NAME_LENGTH
    }
    private static final ComparatorSwitch COMPARATOR_SWITCH
            = ComparatorSwitch.LAST_NAME;
    @Override
    public int compareTo(Customer otherCustomer) {
        switch (COMPARATOR_SWITCH) {
        case LAST_NAME:
            return LAST_NAME_COMPARATOR.compare(this, otherCustomer);
        case FIRST_NAME:
            return FIRST_NAME_COMPARATOR.compare(this, otherCustomer);
        case DATE_OF_BIRTH:
            return DATE_OF_BIRTH_COMPARATOR.compare(this, otherCustomer);
        case ORDER_COUNT:
            return ORDER_COUNT_COMPARATOR.compare(this, otherCustomer);
        case ORDER_COUNT_REVERSED:
            return ORDER_COUNT_REVERSED_COMPARATOR.compare(this, otherCustomer);
        case PREMIUM_CUSTOMER:
            return PREMIUM_CUSTOMER_COMPARATOR.compare(this, otherCustomer);
        case LAST_NAME_LENGTH:
            return LAST_NAME_LENGTH_COMPARATOR.compare(this, otherCustomer);
        default:
            assert false;
            return 0;
        }
    }
}
```

Listing 3

```
private static final Comparator<Customer> LAST_NAME_COMPARATOR
            = Comparator.comparing((Customer c) -> c.lastName)
            .thenComparing(c -> c.firstName)
            .thenComparing(c -> c.dateOfBirth, Comparator.nullsLast(LocalDate::compareTo));
```

Listing 4

```
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
```

Ausgabe 1

Die beiden „Ronald Hogans" besitzen den gleichen Namen. In diesem Fall ist „date of birth" das Zünglein an der Waage. Allerdings ist eines dieser „date of birth" gleich `null` – was aber überhaupt kein Problem darstellt. `Comparator.nullsLast(LocalDate::compareTo)` ermöglicht einen einfachen Umgang mit `null`-Werten. Es behandelt `null`-Werte so, dass sie in der Reihenfolge zuletzt kommen (im Gegensatz zu `nullsFirst`, in dem sie in der Reihenfolge zuerst kommen).

Sortieren nach Vornamen

Durch einfaches Vertauschen von „first name" und „last name" wird ein `Comparator` erzeugt, der nach Vornamen („first name") sortiert, dann nach „last name" und letztlich nach „date of birth" *(siehe Listing 5)*. Die Beispielausgabe ist in *Ausgabe 2* zu sehen.

Sortieren nach Geburtsdatum

Kunden nach ihrem Geburtsdatum („date of birth") zu sortieren, ist ebenfalls keine Zauberei *(siehe Listing 6 und Ausgabe 3)*.

Es ist schön zu erkennen, wie die `null`-Geburtstage in der Reihenfolge zuletzt kommen. Wenn zwei Personen den gleichen Geburtstag teilen, so wie Richard Leblanc und Suzanne Murray, dann werden sie nach „last name" (und anschließend nach „first name") sortiert.

Sortieren nach Anzahl der Bestellungen

Es wird empfohlen, die Primitiv-Typ-Versionen `[then]comparingXXX` zu verwenden, wenn Zahlen verglichen werden *(siehe Listing 7)*. In diesem Beispiel werden die Kunden anhand ihrer Anzahl von Bestellungen („order count") – also einem Integer – sortiert *(siehe Ausgabe 4)*.

Das mag wahrscheinlich nicht das beste Ergebnis sein, für das sich der Onlineshop interessiert. Es würde mehr Sinn ergeben, die Kunden mit der größten „number of orders" zuerst anzuzeigen. Diese Kunden sollten anschließend nach ihrer „average order value" sortiert werden, erneut in absteigender Reihenfolge.

Umgekehrtes Sortieren nach Anzahl der Bestellungen

Das `Comparator`-Interface stellt eine Default-Methode `reversed()` zur Verfügung, die in die `[then]comparing[XXX]`-Aufrufkette eingebunden werden kann. Allerdings ist es dabei wichtig, nicht in die Falle zu tappen und anzunehmen, dass sich diese Methode nur auf den letzten `[then]comparing[XXX]`-Aufruf bezieht. Nein, `reversed()` dreht den gesamten `Comparator` um, inklusive aller `[then]comparing[XXX]`-Methodenaufrufe davor.

Wenn man im hier vorliegenden Fall `.reversed()` ganz am Ende der Methodenkette schreiben würde, wäre die Ausgabe zufällig richtig. Die letzte `reversed`-Methode dreht sowohl den „order count" als auch den „average order value" um, was eigentlich genau dem gewünschten Verhalten entspräche. Das Gleiche gilt auch, wenn man `.reversed()` nur nach der „order count"-`comparingInt`-Methode (das heißt, nach der mittleren Zeile) schreiben würde. Auch in diesem Fall ist das Ergebnis – wenn auch nur zufällig – richtig.

Selbst wenn die Ausgabe richtig ist, ist das Schreiben derartigen Codes trügerisch. Es verlässt sich auf einen Denkfehler, dessen Implementierung etwas – zufällig – richtig macht. Salopp ausgedrückt,

```
private static final Comparator<Customer> FIRST_NAME_COMPARATOR
            = Comparator.comparing((Customer c) -> c.firstName)
            .thenComparing(c -> c.lastName)
            .thenComparing(c -> c.dateOfBirth, Comparator.nullsLast(LocalDate::compareTo));
```

Listing 5

```
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
```

Ausgabe 2

```
private static final Comparator<Customer> DATE_OF_BIRTH_COMPARATOR
            = Comparator.comparing((Customer c) -> c.dateOfBirth,
                                   Comparator.nullsLast(LocalDate::compareTo))
            .thenComparing(c -> c.lastName)
            .thenComparing(c -> c.firstName);
```

Listing 6

```
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
```

Ausgabe 3

ist es nichts anderes, als darauf zu hoffen, dass „falsch und falsch" im Resultat wieder „richtig" ergibt. Ein korrekter Weg, einen einzelnen spezifischen „Zwischen-Comparator" umzudrehen, wird in *Listing 8* aufgezeigt.

Er sieht nicht ganz so elegant wie die vorherigen Code-Ausschnitte aus. Die obige Implementierung verwendet eine überladene Version `thenComparing(Function, Comparator)`, die nebst der eigentlichen Key-Extractor-`Function` noch einen zusätzlichen `Comparator` entgegennimmt. Dieser zusätzlich zur Verfügung gestellte `Comparator` ist `Comparator.reverseOrder()` (nicht zu verwechseln mit `Comparator.reversed()`), der ganz einfach die natürliche Ordnung des betreffenden Datentyps umdreht.

Leider existieren diese überladenen Methoden nicht für die Primitiv-Typ-Versionen `thenComparingInt`, `thenComparingLong` und `thenComparingDouble`, sodass man hier auf die allgemeinen Methoden zurückgreifen muss, die jeweils auch ein (Auto-)Boxing und Unboxing für primitive Typen durchführen. Es ist eine Frage der persönlichen Vorliebe, ob man seine Entwicklungsumgebung (IDE) so einrichtet, dass man bei jedem automatischen Boxing und Unboxing eine Warnung erhält. Wird man von der IDE gezwungen, alle Konvertierungen manuell vorzunehmen, so kann dies sehr hilfreich sein, um Stolperfallen oder Performance-Lecks zu erkennen, die andernfalls unerkannt blieben.

Ein weiterer Nachteil ist, dass man dem Compiler vermehrt „nachhelfen" muss, indem man ihm den spezifischen Typ (hier: `Customer`) innerhalb des Lambda-Ausdrucks angibt. Die Beispielausgabe ist in *Ausgabe 5* zu sehen.

Sortieren nach Premium-Kunden

Listing 9 zeigt die Sortierung der Kunden nach Premium-Kunden („premium customer", `true` soll vor `false` kommen), dann nach „order count" (in absteigender Reihenfolge) und letztlich nach „average order value" (erneut in absteigender Reihenfolge). Alle drei Sortierschritte sind explizit umgedreht. Sie könnten zwar durch ein einzelnes `.reversed()` ganz am Ende der `[then]comparing`-Kette ersetzt werden, doch sollte man die Warnung aus dem letzten Abschnitt im Hinterkopf behalten. Wer sich dennoch dafür entschei-

```
private static final Comparator<Customer> ORDER_COUNT_COMPARATOR
            = Comparator.comparingInt((Customer c) -> c.orderCount)
            .thenComparingDouble(c -> c.averageOrderValue);
```

Listing 7

```
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
```

Ausgabe 4

```
private static final Comparator<Customer> ORDER_COUNT_REVERSED_COMPARATOR
            = Comparator.comparing((Customer c) -> Integer.valueOf(c.orderCount),
                                   Comparator.reverseOrder())
            .thenComparing((Customer c) -> Double.valueOf(c.averageOrderValue),
                           Comparator.reverseOrder());
```

Listing 8

```
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
```

Ausgabe 5

```
private static final Comparator<Customer> PREMIUM_CUSTOMER_COMPARATOR
            = Comparator.comparing((Customer c) -> Boolean.valueOf(c.premiumCustomer),
                                   Comparator.reverseOrder())
            .thenComparing((Customer c) -> Integer.valueOf(c.orderCount),
                           Comparator.reverseOrder())
            .thenComparing((Customer c) -> Double.valueOf(c.averageOrderValue),
                           Comparator.reverseOrder());
```

Listing 9

```
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium? true
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium? true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium? true
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium? true
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium? true
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
```

Ausgabe 6

```
private static final Comparator<Customer> LAST_NAME_LENGTH_COMPARATOR
            = Comparator.comparingInt((Customer c) -> c.lastName.length())
            .thenComparing(c -> c.lastName)
            .thenComparing(c -> c.firstName);
```

Listing 10

```
Name: Joseph  Davis    | DoB: 1981-11-03 | Orders: 9 | AOV: 783.16 | Premium?  true
Name: Dolores Falco    | DoB:       null | Orders: 3 | AOV: 556.41 | Premium?  true
Name: Ronald  Hogan    | DoB:       null | Orders: 2 | AOV: 297.99 | Premium? false
Name: Ronald  Hogan    | DoB: 1997-11-26 | Orders: 2 | AOV: 368.10 | Premium? false
Name: Suzanne Murray   | DoB: 1975-10-06 | Orders: 2 | AOV: 199.49 | Premium? false
Name: Graham  Reamer   | DoB: 1967-03-16 | Orders: 3 | AOV: 255.96 | Premium? false
Name: Richard Leblanc  | DoB: 1975-10-06 | Orders: 2 | AOV: 498.30 | Premium? false
Name: Adella  Wheeler  | DoB:       null | Orders: 7 | AOV: 824.33 | Premium?  true
Name: Charles Quintana | DoB: 1989-01-02 | Orders: 4 | AOV: 748.35 | Premium?  true
Name: William Robinson | DoB:       null | Orders: 2 | AOV: 737.35 | Premium?  true
```

Ausgabe 7

det, sollte zumindest einen erklärenden Kommentar hinzufügen. Die Programmausgabe sehen Sie in *Ausgabe 6*.

Es ist wichtig zu beachten, dass die standardmäßige (natürliche) Ordnung für Booleans zuerst `false` und dann `true` ist. Da hier die Reihenfolge umgedreht wurde, erscheint korrekt zuerst `true` vor `false`.

Sortieren nach der Länge des Nachnamens

„Nun, das ist ja alles schön und gut, was mir da bis jetzt gezeigt wurde, aber was ist, wenn ich nicht nur einfach Attribute vergleichen möchte, sondern kompliziertere Dinge?" – Kein Problem! Im abschließenden Beispiel werden Kunden anhand der Länge ihres Nachnamens sortiert *(siehe Listing 10)*. Das Ergebnis ist in *Ausgabe 7* zu sehen.

Man muss einfach eine Key-Extractor-`Function` zur Verfügung stellen, die das macht, was man wünscht. Man sieht, dass sich an der Art, wie `Comparators` geschrieben sind, ansonsten wenig ändert.

Zusammenfassung und Ausblick

Im zweiten Teil dieses Artikels wurden viele Beispiele gezeigt, wie die Vergleichsfunktionalität in Java implementiert werden kann. Auch wenn das Umdrehen einzelner Sortierschritte (immer noch) etwas umständlich ist, sind die Konsistenz und der Stil solcher Vergleichsmethoden ansonsten beeindruckend.

In naher Zukunft gilt es, nach einer besseren Lösung Ausschau zu halten, was die umgedrehten Sortierschritte anbelangt. Denkbar wäre eine Hilfsklasse, die das Umdrehen einzelner Sortierschritte vereinfacht und darüber hinaus solche Methoden auch für die primitiven Datentypen anbietet. Die Frage, ob man nennenswerte Performance-Einbußen bei der Verwendung solcher Methodenketten in Kauf nehmen muss, wurde zwar nicht direkt, aber in einem vergleichbaren Fall im Rahmen von `equals`- und `hashCode`-Methoden in einem zweiteiligen Blogeintrag des Autors beantwortet *(https://link.simplexacode.ch/zxxx und https://link.simplexacode.ch/5gih)*. Kurz: Die Performance-Einbußen bewegen sich innerhalb eines unbedeutenden Rahmens und rechtfertigen es nicht, von der modernen und konsistenten Implementierung natürlicher Ordnung bei Java-Objekten abzukommen – sofern man nicht aufzeigen kann, dass die Zeit, die mit dem Vergleichen von Java-Objekten verbracht wird (namentlich beim Sortieren) einen unstreitig signifikanten Großteil der gesamten Laufzeit einer Applikation ausmacht. Das wiederum dürfte für die meisten realen Applikationen ganz klar nicht der Fall sein.

Hinweis: Die Quelltexte zu diesem Artikel findet der Leser unter *https://link.simplexacode.ch/txx9*. Die englische Version des Artikels steht unter *https://link.simplexacode.ch/gurm* zur Verfügung.

Christian Heitzmann
christian.heitzmann@simplexacode.ch

Christian Heitzmann ist Gründer und Geschäftsführer der SimplexaCode AG in Luzern, die sich auf Software-Entwicklung, -Schulung und -Beratung vor allem für MINT-Anwendungen und technische Implementierungsthemen in Java spezialisiert hat. Er ist seit fünfzehn Jahren mit Java vertraut und hat während vieler Jahre Algorithmen und Mathematik unterrichtet.

JUG SAXONY DAY

13.09.

PROGRAMM
1 Keynote und über 30 Sessions
jug-saxony-day.org

ORT
Radisson Blu Park Hotel & Conference Centre
Radebeul b. Dresden

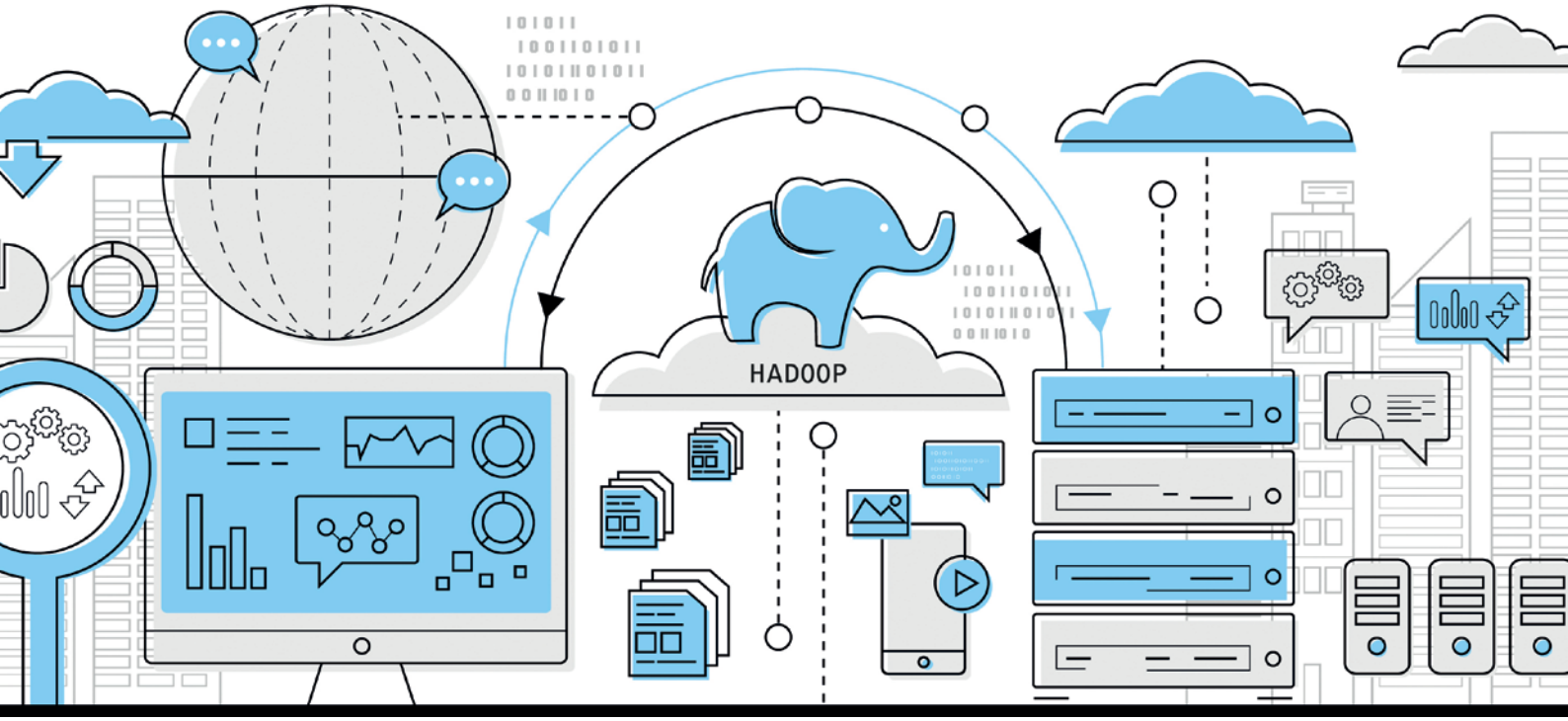
TICKETBUCHUNG
Early Bird bis 15. Juli 2019
Late Bird ab 16. Juli 2019
Freitickets für Studierende
(Solange der Vorrat reicht.)

backoffice.jugsaxony.org

Eine Veranstaltung des JUG Saxony e. V.

Kontakt
team@jugsaxony.org
jugsaxony.org

Folgt uns auf



Hadoop – Taming the Elephant (With a Whale)

Lisa Maria Moritz, Innoq

Dieser Artikel richtet sich an Personen, die bisher keine Erfahrung mit Hadoop haben und sich einen Einstieg basierend auf praktischen Übungen wünschen. Neben der Erläuterung der Kernkomponenten, werden mithilfe eines Docker-Image grundlegende Konzepte gezeigt.

Einleitung

Dieser Artikel basiert auf dem Vortrag „Hadoop – Taming the Elephant (With a Whale)". Er behandelt die Grundlagen von Hadoop *[1]* und bleibt dabei praxisnah und zielgerichtet. Der Aufbau des Artikels gleicht in weiten Teilen dem Aufbau des Vortrags, dessen Folien online verfügbar sind *[2] (siehe Abbildung 1)*.

Was ist Hadoop eigentlich?

Hadoop ist ein System zur Datenspeicherung und -analyse, das seit 2006 entwickelt wird und seit 2008 zu Apaches Kernprojekten gehört. Heutzutage wird Hadoop oft in Enterprise-Systemen für die Arbeit mit Big Data eingesetzt. Die Tochter von Doug Cutting, einem der Mitbegründer des Projektes, nannte ihren gelben Plüschelefant „Hadoop". Daher rühren sowohl der Projektname als auch das gelbe Elefantenlogo *[3]*. Hadoop besteht aus fünf Kernkomponenten: Hadoop Common, HDFS, MapReduce, YARN und Ozone. Hadoop Common beinhaltet die grundlegenden Bestandteile der Applikation. HDFS, das „Hadoop Distributed File System", dient zur Speicherung von Daten. MapReduce stellt ein Programmiermodell zur Verarbeitung großer Datenmengen dar. YARN, kurz für „Yet Another Resource Negotiator", ist das Cluster-Ressource-Management-System von Hadoop. Ozone ist erst seit Ende 2018 in einer Beta-Version Teil des Kerns, es handelt sich hierbei um einen Objektspeicher. Dieser Artikel befasst sich hauptsächlich mit HDFS und MapReduce.

Hadoop in Docker starten

Ein Vorgehen für einen einfachen und schnellen Start in die Thematik, das viele Möglichkeiten für praktische Fingerübungen bietet, ist die Verwendung eines Docker-Image *[4]*. So werden umfangreiche Installationen vermieden und die Übungen können schnell in Angriff genommen werden. Auch für die ersten Schritte in Hadoop kann ein Docker-Image eingesetzt werden. In Produktion bildet Hadoop in Docker die Ausnahme. Die Basis für die Beispiele in diesem Artikel ist das Docker-Image `sequenceiq/hadoop-docker:2.7.0` *[5]*, in dem alle für Hadoop notwendigen Bestandteile als Single-Node-Installation in einem einzigen Docker-Container gestartet werden. Mit dem Befehl in *Listing 1* wird das Docker-Image gestartet.

Beim Starten des Image werden die Ports für die Web-Benutzeroberfläche (*50070*), das Tracken von Jobs (*8088*) sowie der Port zum Herunterladen von Ergebnissen (*50075*) weitergeleitet. Dies ermöglicht später einen Zugriff auf diese Dienste via „localhost".

Nach dem Ausführen des *run*-Befehls wird man direkt zur Bash des gestarteten Containers weitergeleitet. Dies ist der Indikator für den erfolgreichen Start des Docker-Containers. Um zu kontrollieren, ob alle Hadoop-Services erfolgreich gestartet wurden, kann ein Beispiel-MapReduce-Job verwendet werden, der von Apache bereits in Hadoop inkludiert wurde. Zum Starten des Jobs, der in Java geschrieben ist, wird der Befehl *(siehe Listing 2)* in der Bash des Docker-Containers ausgeführt.

Es gibt zwei Wege zur Überprüfung der Ergebnisse. Zum einen können die Ergebnisse mithilfe der Web-UI kontrolliert werden. Hierfür wird in einem Web-Browser die URL „localhost:50070" geöffnet. In der Navigationsleiste ist der Punkt „Utilities" zu finden, der den Punkt „Browse HDFS" beinhaltet. Hier kann das gesamte HDFS erkundet werden. Unter anderem auch die zuvor entstandenen Ergebnisse. Diese sind unter „/user/root/output" zu finden. Es handelt sich um einen Ordner mit zwei Dateien: einem SUCCESS-Deskriptor – eine leere Datei, die anzeigt, dass die Durchführung des MapReduce-Jobs erfolgreich war – sowie einer Datei „part-r-00000", die die entstandenen Ergebnisse enthält *(siehe Abbildung 2)*.

Zum anderen können die Ergebnisse über die Kommandozeile angesehen werden. Gibt man den Befehl *(siehe Listing 3)* in der Bash des Docker-Containers ein, so werden die Ergebnisse des Jobs ausgegeben.

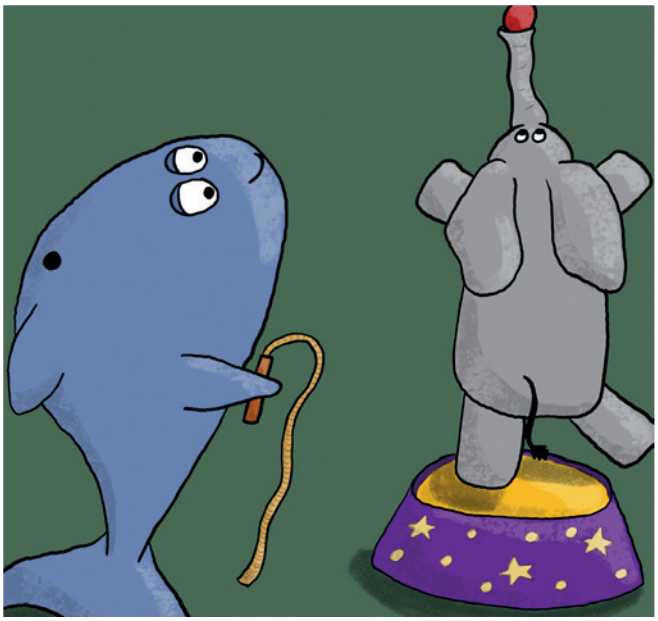


Abbildung 1: Titelbild des Vortrags „Hadoop - Taming the Elephant (With a Whale)" (Quelle: Lisa Maria Moritz)

```
docker run -it \
-p 50070:50070 \
-p 8088:8088 \
-p 50075:50075 \
sequenceiq/hadoop-docker:2.7.0 \
/etc/bootstrap.sh -bash
```

Listing 1: Befehl zum Starten des Hadoop-Docker-Containers

MapReduce

Ein guter Anfang, um erste praktische Erfahrungen mit Hadoop zu machen, ist, einen MapReduce-Job mithilfe von Java und Maven zu entwickeln. Aber was ist MapReduce? MapReduce ist, wie bereits erwähnt, eine der fünf Kernkomponenten von Hadoop. Es handelt sich um ein Programmiermodell zur Verarbeitung großer Datenmengen und gliedert sich in zwei Teilphasen: die Map-Phase und die Reduce-Phase *[6]*. Beide Phasen akzeptieren Schlüssel-Wert-Paare als Eingabe und liefern diese als Ausgabe. In der Map-Phase wird der Datensatz zerlegt, es werden ein neuer Schlüssel extrahiert und ein passender Wert zugeordnet. Wert und Schlüssel richten sich

```
$HADOOP_PREFIX/bin/hadoop jar \
share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar \ grep input output 'dfs[a-z.]+'
```

Listing 2: Befehl zum Ausführen des bereitgestellten MapReduce-Beispiels

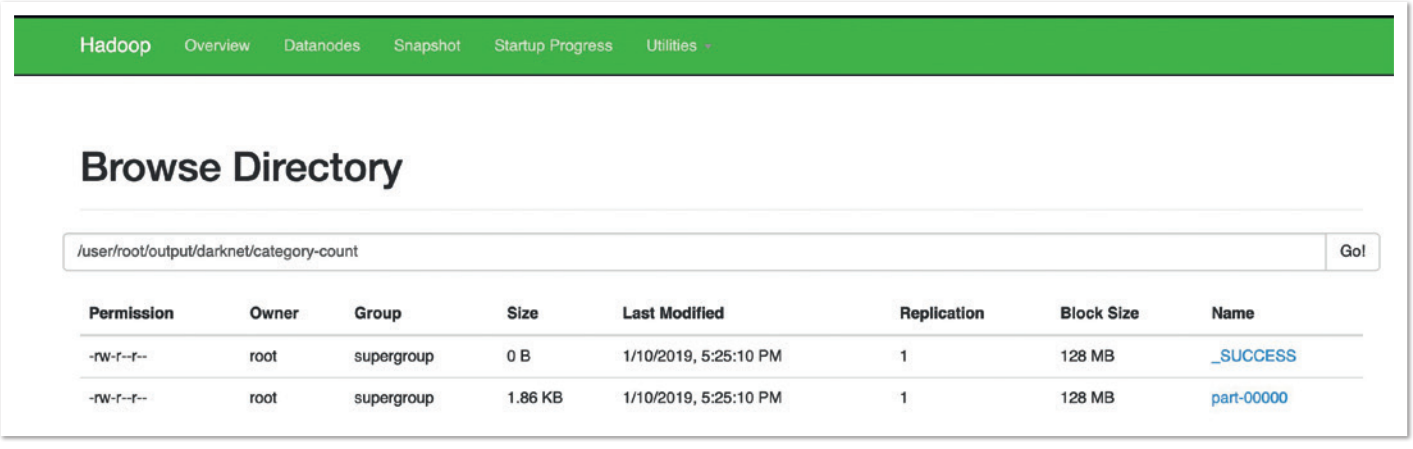


Abbildung 2: Erkunden des HDFS mithilfe der Web-UI (Quelle: Lisa Maria Moritz)

```
$HADOOP_PREFIX/bin/hdfs dfs -cat output/*
```

Listing 3: Befehl zum Abfragen der Ergebnisse über die Kommandozeile

hierbei nach dem Ziel des Jobs. Soll beispielsweise aus einem Haustier-Datensatz mit Identifikationsnummern, Tierarten und Tiernamen die Anzahl der verschiedenen Tierarten ermittelt werden, so ist es denkbar, die Tierart als Schlüssel und die Identifikationsnummer als Wert zu extrahieren, wie in *Abbildung 3* dargestellt.

In der Reduce-Phase werden die vom Mapper zerlegten Datensätze zur Erstellung des Ergebnisses wieder zusammengefügt. Für das Haustier-Beispiel würde das bedeuten, die Identifikationsnummern für eine bestimmte Tierart zu zählen und so die Anzahl der Haustiere dieser Tierart zu ermitteln *(siehe Abbildung 4)*.

Optional kann einem MapReduce-Job ein sogenannter "Combiner" zugefügt werden, der die Netzwerklast zwischen Mapper und Reducer reduziert. In den meisten Fällen kann der bereits vorhandene Reducer ebenfalls als Combiner eingesetzt werden. Für die nachfolgenden Beispiele ist ein Combiner nicht vonnöten, da keine Multi-Node-Installation von Hadoop vorliegt und die Netzwerkauslastung kein Problem darstellt.

YARN

Seit Hadoop 2, also seit 2013, gehört YARN zum Kern von Hadoop. Neben der Zuweisung der Systemressourcen für verschiedene An-

```
public class CategoryCountMapper
    extends Mapper<LongWritable, Text, Text, LongWritable> {
    private static final int CATEGORY = 1;
    private static final String SEPARATOR = ",";
    @Override
    public void map(LongWritable key, Text value,
            Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] lineData = line.split(SEPARATOR);
        context.write(new Text(lineData[CATEGORY]), key);
    }
}
```

Listing 4: Mapper in Java

```
public class CategoryCountReducer
    extends Reducer<Text, LongWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key,
            Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        int count = 0;
        for (LongWritable value : values) {
            count++;
        }
        context.write(key, new IntWritable(count));
    }
}
```

Listing 5: Reducer in Java

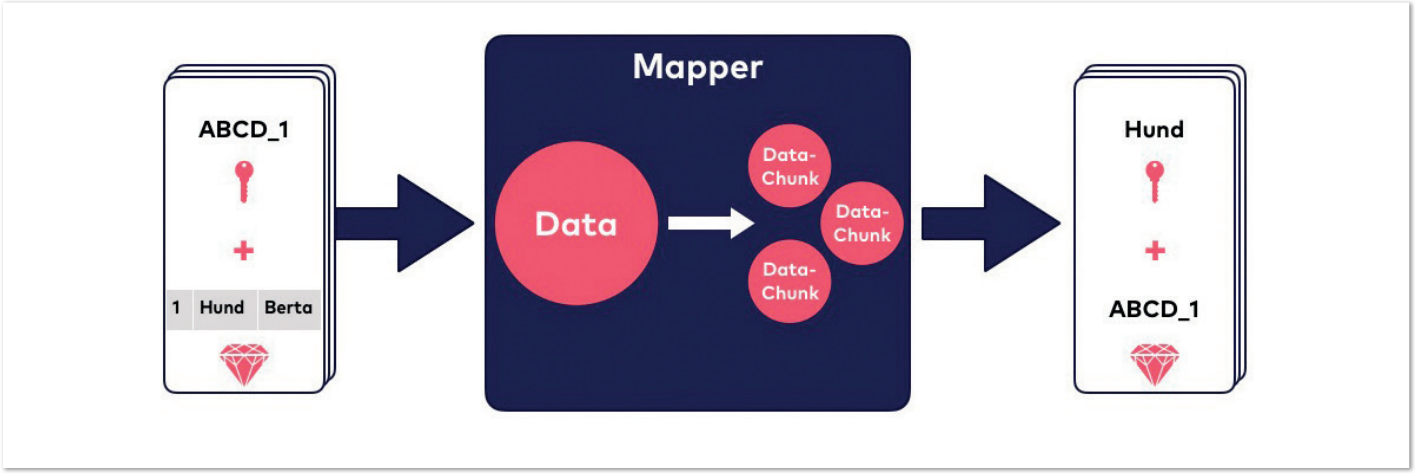


Abbildung 3: Veranschaulichung des Mapper-Beispiels (Quelle: Lisa Maria Moritz)

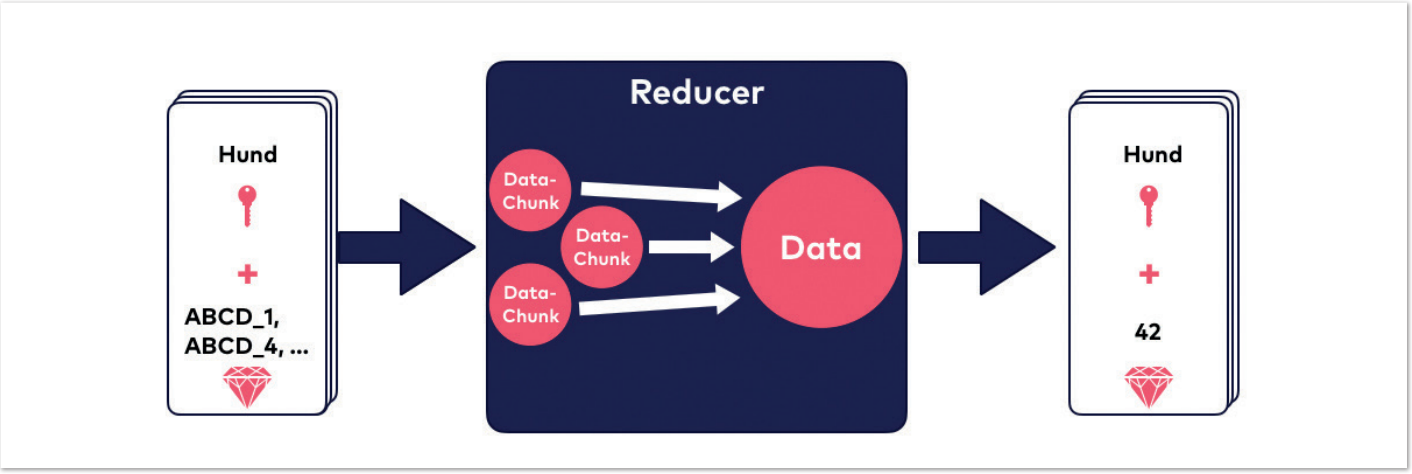


Abbildung 4: Veranschaulichung des Reducer-Beispiels (Quelle: Lisa Maria Moritz)

wendungen kümmert sich YARN außerdem um die Verteilung von Aufgaben auf den Knoten des Clusters *[7]*. Durch die Entkopplung von Ressourcenmanagement und Planung der MapReduce-Jobs gelang es mithilfe von YARN, die vorherige Implementierung von MapReduce zu verbessern. YARN ermöglicht außerdem die Integration von Alternativen zu MapReduce.

HDFS

Hadoop speichert die Daten im HDFS, dem „Hadoop Distributed File System". Das HDFS besteht aus zwei Node-Typen, die in einem Master-Worker-Muster zusammenarbeiten. Zum einen gibt es die DataNodes, die für die Speicherung der Daten zuständig sind. Zum anderen gibt es einen NameNode, der die Metadaten des HDFS beinhaltet und „weiß", wo welche Daten abgelegt sind. Die Datenspeicherung selbst erfolgt in sogenannten Blöcken. Eine Datei wird beim Speichern in einen oder mehrere Blöcke zerlegt, die einzelnen Blöcke können auf verschiedenen DataNodes abgelegt werden *[8]*. Das Dateisystem von Hadoop ist darauf ausgelegt, effizient wenige große Datensätze zu verarbeiten.

MapReduce mit Java

Zur Realisierung eines MapReduce-Jobs in Java werden zwei Abhängigkeiten via Maven eingebunden: „hadoop-common" *[9]* und „hadoop-mapreduce-client-core" *[10]*. Die nachfolgenden Implementierungen basieren auf einem CSV-Datensatz, in dem es unter anderem die Information „Category" gibt. Das Ergebnis des MapReduce-Jobs soll das Zählen des Auftretens der einzelnen Kategorien sein.

Im vorigen Abschnitt wurde bereits erwähnt, dass MapReduce-Jobs aus einem Mapper und einem Reducer bestehen. Beide werden mithilfe des von Hadoop bereitgestellten API umgesetzt.

Als Erstes wird der Mapper entwickelt, der von der Hadoop-Klasse `Mapper` erbt. Die `map`-Methode muss überschrieben werden. Beim Erben von der Basisklasse `Mapper` müssen die Variablentypen von Schlüssel und Wert der Eingabe und Ausgabe spezifiziert werden. Die `map`-Methode hat die Parameter `key` und `value`, wobei dieser bei einer CSV-Datei die gesamte Zeile darstellt, sowie einen `context`. Der

```
public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: <input> <output>");
        System.exit(-1);
    }

    String inputPath = args[1];
    String outputPath = args[2];

    Job job = Job.getInstance();
    job.setJobName("Category Count");

    FileInputFormat.addInputPath(job, new Path(inputPath));
    FileOutputFormat.setOutputPath(job, new Path(outputPath));

    job.setMapperClass(CategoryCountMapper.class);
    job.setReducerClass(CategoryCountReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setJarByClass(CategoryCount.class);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Listing 6: MapReduce-Einstiegspunkt in Java

```
$HADOOP_PREFIX/bin/hadoop jar \
/tmp/category-count-mapreduce.jar \
example/input \
example/category-count
```

Listing 7: Befehl zum Starten des Java-MapReduce-Jobs

Parameter `context` dient zur Weitergabe der Ergebnisse. Im folgenden Beispiel wird die Kategorie einer Zeile separiert und gemeinsam mit der Zeilen-ID `(key)` im `context` abgelegt. Hadoop verwendet eigene Datentypen, so wird `Text` statt `String` und `LongWritable` statt `Long` verwendet *(siehe Listing 4)*.

Auf dieser Basis wird der Reducer implementiert. Er wird die Ergebnisse des Mappers aufnehmen und die Vorkommen einzelner Kategorien zählen. Ein Reducer muss von der Hadoop-Klasse Reducer erben und die `reduce`-Methode überschreiben. Die `reduce`-Methode hat als Parameter einen `key`, ein `Iterable values` und den `context`. Als *key* erhält der Reducer die Kategorie, die im Mapper extrahiert wurde, die zugehörigen Zeilen-IDs werden ihm als `Iterable` übergeben. Die Aufgabe des Reducer liegt im Zählen der IDs und somit im Reduzieren der Ergebnisse des Mappers. Bei Iteration über die IDs wird das Auftreten der einzelnen Kategorien gezählt. Das Ergebnis wird als Wert vom Typ `IntWritable` zu der jeweiligen Kategorie gespeichert *(siehe Listing 5)*.

Neben einem Mapper und einem Reducer benötigt die Java-Implementierung des MapReduce-Jobs einen Einstiegspunkt. Dieser beinhaltet einige zusätzliche Informationen zum MapReduce-Job, wie einen Namen, die Angabe der Klasse des Mappers und des Reducer sowie die Informationen darüber, wo die Eingabedaten liegen und wohin die Ausgabedaten geschrieben werden sollen *(siehe Listing 6)*.

Für die Ausführung des MapReduce-Jobs wird ein `jar` benötigt, dieses kann mithilfe von Maven erstellt werden. Nach Ausführen des Maven-Befehls liegt das `jar` im *target*-Ordner und kann von dort in den Docker-Container verschoben werden.

Zum Starten des MapReduce-Jobs dient der gleiche Befehl wie zum Starten des MapReduce-Jobs von Apache, lediglich die Argumente des Befehls müssen angepasst werden *(siehe Listing 7)*.

```
$HADOOP_PREFIX/bin/hadoop jar \
$HADOOP_PREFIX/share/hadoop/tools/lib/hadoop-streaming-2.7.0.jar \
- files /tmp/mapper.py,/tmp/reducer.py \
- input example/input \
- output example/category-not-java-count \
- mapper /tmp/mapper.py \
- reducer /tmp/reducer.py
```

Listing 8: Befehl zum Starten eines MapReduce-Jobs über das Hadoop-Streaming-API

```
#!/usr/bin/env python
import sys

id = 0
for line in sys.stdin:
    val = line.strip()
    data = val.split(',')
    category = data[1]
    print(category + '\t' + str(id))
    id = id + 1
```

Listing 9: Mapper in Python

```
#!/usr/bin/env python
import sys

previous_key = None
count = 0

for line in sys.stdin:
    (key, val) = line.strip().split('\t')
    if previous_key is None:
        previous_key = key
    if key == previous_key:
        count = count + 1
    else:
        print(key + '\t' + str(count))
        count = 1
    previous_key = key
```

Listing 10: Reducer in Python

Wie im Beispiel zuvor können die Ergebnisse via Web-UI oder Kommandozeile betrachtet werden.

MapReduce ohne Java

MapReduce-Jobs können auch in anderen Sprachen als Java implementiert werden. Eine Möglichkeit dafür bietet das Hadoop-Streaming-API *[11]*. Es arbeitet mit der Unix-Standardeingabe und -ausgabe, dadurch ist die einzige Anforderung an die Programmiersprache, damit umgehen zu können. Das in Java vorhandene, nützliche Feature, dass die Werte für einen Schlüssel dem Reducer als `Iterable` übergeben werden, geht hierbei leider verloren, sodass es die Aufgabe des Programmierers ist, die Werte den richtigen Schlüsseln zuzuordnen. Glücklicherweise übergibt das Hadoop-Streaming-API dem Reducer die Schlüssel in geordneter Reihenfolge, es genügt also, den vorangegangenen Schlüssel zwischenzuspeichern. Durch die Verwendung von Standardeingabe und -ausgabe ergibt sich der Vorteil, dass Mapper und Reducer mithilfe von Pipes in der Kommandozeile getestet werden können. Für MapReduce-Jobs, die mit dem Streaming-API implementiert werden, ist es nicht nötig, einen dedizierten Startpunkt zu implementieren. Es ist ausreichend, einen Mapper und einen Reducer zu schreiben, die mit folgendem Befehl *(Listing 8)* über das in Java implementierte Hadoop-Streaming-API gestartet werden können.

In diesem Beispiel werden ein in Python implementierter Mapper und Reducer aufgerufen, die mit den gleichen Eingabedaten wie die Java-Realisierung arbeiten. Ein dediziertes Verzeichnis

```
curl -i -L \
“http://localhost:50070/webhdfs/v1/user/root/output/?op=LISTSTATUS”
```

Listing 11: Ordner durchsuchen über HttpFS

```
curl -i -L \
“http://localhost:50070/webhdfs/v1/user/root/output/example/category-count/part-r-00000?op=OPEN“
```

Listing 12: Dateien öffnen über HttpFS

speichert die Ausgabedaten. Bei der Verwendung des Hadoop-Streaming-API sollte daran gedacht werden, die Dateien mithilfe von `chmod` ausführbar zu machen. Der Mapper geht Zeile für Zeile die Standardeingabe durch, teilt die CSV-Zeile am Separator und schreibt die erhaltene Kategorie, die in der zweiten Spalte der CSV-Zeile steht, sowie eine selbst zugewiesene ID in die Standardausgabe *(Listing 9)*.

Der Reducer ist ähnlich aufgebaut. Er verarbeitet Zeile für Zeile die Standardeingabe, die der Ausgabe des Mappers entspricht. Die Besonderheit des Reducer besteht darin, dass der vorherige Zeilenschlüssel gespeichert werden muss, um nur die zu einem Schlüssel gehörenden Werte zu zählen. Das Ergebnis des Reducer wird in die Standardausgabe geschrieben *(siehe Listing 10)*.

Ergebnisse über HTTP erhalten

Die in den beiden vorangegangenen Abschnitten entstandenen Ergebnisse sollen möglicherweise in einer UI dargestellt werden. Wie kann man die Ergebnisse von außen abfragen? Auf diese Frage liefert HttpFS eine Antwort. Diese Abkürzung steht für „HDFS over HTTP“. Es ermöglicht, alle Operationen, die man auf dem HDFS ausführen kann, über HTTP auszuführen. Für die Erstellung einer UI, die die Ergebnisse veranschaulicht, genügen zwei Operationen des HDFS: das Browsen durch Ordnerstrukturen und das Öffnen von Dateien. Hier werden diese als Beispiele mithilfe von „curl“ gezeigt *[12]*. Das Browsen von Ordnerstrukturen erfolgt durch die Verwendung des Öffnungsmodus `LISTSTATUS` *(siehe Listing 11)*.

Die Antwort wird in JSON übertragen und enthält ein Array mit Dateien innerhalb des Ordners. Es werden unter anderem der Dateiname (*pathSuffix*) und der Typ der Datei (*type*) übertragen. Eine Datei kann vom Typ FILE oder *DIRECTORY* sein. Ein *DIRECTORY* kann erneut durchsucht werden, während ein *FILE* wie folgt geöffnet werden kann *(siehe Listing 12)*.

Resümee

Trotz des großen Umfangs von Hadoop ist es möglich, erste Schritte mit „dem Elefanten“ am heimischen Rechner zu wagen. Dabei hilft der „Wal“ und komprimiert den grauen beziehungsweise gelben Riesen in einen handlichen Container. Die eigentlichen Stärken von Hadoop, wie die Verarbeitung großer, unstrukturierter Datenmengen, können auf diese Weise zwar nicht demonstriert werden. Die Grundlagen von MapReduce und HDFS können allerdings leicht zugänglich weitergegeben werden.

Quellen

[1] *https://hadoop.apache.org/*
[2] *https://www.innoq.com/de/talks/2019/01/hadoop-einfuehrung-jug-hh-2019/*
[3] Tom White (2015): *Hadoop: The Definitive Guide (4th Edition)*. O’REILLY.
[4] *https://www.docker.com/*
[5] *https://hub.docker.com/r/sequenceiq/hadoop-docker/*
[6] Jeffrey Dean, Sanjay Ghemawat (2004): *MapReduce: Simplified Data Processing on Large Clusters*. *https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf.*
[7] *https://www.searchenterprisesoftware.de/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator*
[8] Alex Holmes (2015): *Hadoop In Practice (2nd Edition)*. Manning.
[9] *https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common*
[10] *https://mvnrepository.com/artifact/io.hops/hadoop-mapreduce-client-core*
[11] *https://www.tutorialspoint.com/hadoop/hadoop_streaming.htm*
[12] *https://curl.haxx.se/*

Lisa Maria Moritz
lisa.moritz@innoq.com

Lisa ist Consultant bei INNOQ. Ihre Schwerpunkte liegen in Web-Architekturen und der Programmierung mit Java. Sie interessiert sich auch für andere Gebiete und Sprachen, wie Python oder JavaScript. Sie ist sowohl im Backend als auch im Frontend unterwegs.

Ein Einblick in Web-APIs

Simon Skoczylas, Karakun

Web-APIs, als Programmierschnittstellen im Browser, bieten im Bereich der Web-Entwicklung immer mehr Möglichkeiten zur Interaktion mit dem Nutzer und der Welt außerhalb des Browsers. Hier möchte ich ein paar der bekannten und unbekannten Web-APIs vorstellen, um Entwickler auf verschiedene Web-APIs aufmerksam zu machen. Weiterhin möchte ich zeigen, dass die Entscheidung zwischen nativen, hybriden und Web-Anwendungen immer sorgfältiger getroffen werden muss.

Das Web entwickelt sich kontinuierlich weiter. Seit Jahrzehnten existiert der Wunsch, Anwendungen über das Web erreichbar und nutzbar zu machen. Auf diesem Weg verwendete man verschiedene Technologien, dazu gehörten in der Vergangenheit auch Java Applets und Flash, und in Zukunft dann WebAssembly *[1]* – wobei Letzteres als Standard für das Web entwickelt wurde. Unabhängig von diesen Technologien basiert das Web auf Standards wie HTML, CSS und einer Vielzahl durch das World Wide Web Consortium (W3C) standardisierten Web-APIs, die von modernen Browsern zur Verfügung gestellt werden. Diese Web-APIs zu kennen und korrekt einzusetzen, stellt eine neue Herausforderung dar. Kenntnisse über vorhandene Schnittstellen unterstützen und erleichtern Entscheidungen für oder gegen eine Technologie, weshalb das Wissen über die vorhandenen Web-APIs bei Entscheidungen für eine Web-Anwendung hilfreich ist. Hinzu kommt, dass dank Progressive Web Apps der Fokus von Anwendungen immer häufiger ins Web verlagert wird und das Wissen über bestehende Web-APIs zur Verbesserung der Web-Anwendung führt. Wo früher nur unzuverlässig bestimmt werden konnte, welche Funktionalität ein Browser bietet, ist es heutzutage mit Prinzipien wie der sogenannten „Feature Detection" problemlos möglich, die neuesten Web-APIs zu verwenden, ohne bestehen-

de Funktionalitäten einer Web-Anwendung in Gefahr zu bringen. Ganz im Sinne von Progressive Enhancement kann auf diese Weise dem Anwender dort, wo die Möglichkeit besteht, nach und nach ein Mehrwert geboten werden. Dank der Dokumentation seitens der Browser-Hersteller und dank der allgemeinen Dokumentation der Mozilla Foundation *[2]* sind viele Web-APIs verständlich beschrieben und müssen nicht erst mühevoll erarbeitet werden. Falls innerhalb der Dokumentation der Mozilla Foundation noch Lücken vorhanden sind oder Beiträge nicht in die benötigte Sprache übersetzt sind, ist jeder dazu aufgerufen, an der Dokumentation mitzuarbeiten. Des Weiteren können Entwickler über die Seite *http://www.caniuse.com* feststellen, in welchem Browser ein bestimmtes API zur Verfügung steht.

Feature Detection

Um eine bestimmte Funktionalität zu verwenden, muss vorher sichergestellt werden, dass diese im Browser des Anwenders vorhanden und einsatzbereit ist. Ohne Prüfung kann es innerhalb der Anwendung zu Fehlern kommen, wenn ein Nutzer mit einem Browser ohne diese geforderte Funktionalität auf die Web-Anwendung zugreift. In der Vergangenheit wurde bei dieser Prüfung oft auf den User-Agent-String *(siehe Listing 1)* zurückgegriffen und dann im Browser, oder sogar schon auf dem Server, eine Entscheidung über die vorhandenen Funktionalitäten getroffen.

Auf diese Art und Weise wurde versucht, den Browser, die Version und das Betriebssystem zu ermitteln. Verständlicherweise ist eine Prüfung über den User-Agent-String alles andere als zuverlässig, da dieser auf einen beliebigen Wert gesetzt werden kann und es in der Vergangenheit häufiger zu Missverständnissen kam. Hatte zum Beispiel der Internet Explorer bis Version 10 die Zeichenkette „MSIE" innerhalb des User-Agent-String verwendet, änderte sich dies mit Version 11. Dies führte innerhalb verschiedener Web-Anwendungen zu Problemen, da sie auf spezielle Funktionalitäten des Internet Explorers angewiesen waren und diesen nicht mehr erkannten.

In aktuellen Web-Anwendungen verzichtet man deshalb auf die Verwendung des User-Agent-String und prüft mittels Feature Detection, ob eine Funktionalität vorhanden ist *[3]*. Im Grunde basiert die Feature Detection darauf, im Browser zu prüfen, ob ein bestimmtes Objekt oder eine bestimmte Funktion vorhanden sind *(siehe Listing 2)*. Für erweiterte Prüfungen zu Funktionalitäten kann auf die JavaScript-Bibliothek Modernizr *[4]* zurückgegriffen werden.

Page Visibility API

Mit dem Page Visibility API *[5]* kann bestimmt werden, ob die aktuelle Web-Anwendung aktuell für den Nutzer sichtbar ist und der Fokus auf ihr liegt. Dies ist vor allem für das sogenannte „Tabbed Browsing" relevant, da die eigene Web-Anwendung sich in einem Tab im Hintergrund befinden kann. Währenddessen können ressourcenintensive Aufgaben, wie zum Beispiel das Polling nach Daten, gestoppt werden. Um den aktuellen Status zu ermitteln, wird

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/73.0.3683.86 Safari/537.36
```

Listing 1: Beispiel für einen User-Agent-String

```
if (navigator.bluetooth) {
// Funktionalität kann verwendet werden
} else {
// Alternativen verwenden
}
```

Listing 2: Feature Detection für das Web Bluetooth API

das `Document`-Objekt um zwei Attribute und ein Event erweitert. Das Attribut `hidden` repräsentiert einen booleschen Wert, der `true` zurückgibt, wenn die Anwendung nicht sichtbar, und `false`, wenn die Anwendung sichtbar ist. Das Attribut `visibilityState` liefert eine Zeichenkette abhängig vom aktuellen Status der Anwendung. Ist der Rückgabewert `visible`, dann ist die `Web`-Anwendung aktuell sichtbar, ist der Rückgabewert `hidden`, dann ist die Anwendung aktuell für den Nutzer nicht sichtbar. Da `document.hidden` aus historischen Gründen in der Spezifikation beibehalten wurde, sollten Entwickler vor allem auf `document.visibilityState` zurückgreifen, um den Status der Sichtbarkeit zu bestimmen. Das Event `visibilitychange` wird vom Browser bei Änderungen an der Sichtbarkeit gesendet *(siehe Listing 3)*.

NavigatorOnLine (oder Browser state)

Die HTML5-Spezifikation definiert über die `NavigatorOnline`-Schnittstelle *[6]* das boolesche Attribut `onLine` am `Navigator`-Objekt. Dieses Attribut dient zur Bestimmung des aktuellen Verbindungsstatus des Browsers. In anderen Worten, ob der Browser gerade online `(true)` oder offline `(false)` ist. Zusätzlich dazu werden die Events `online` und `offline` bereitgestellt, um über eine Änderung am Verbindungsstatus zu informieren. *Listing 4* zeigt, wie das `onLine`-Attribut ausgelesen und die Events verwendet werden können.

Notification API

Dank des Notification API *[7]* kann aus Web-Anwendungen heraus eine Benachrichtigung über das Benachrichtigungssystem des Betriebssystems an den Nutzer gesendet werden. Eine solche Benachrichtigung kann damit auch angezeigt werden, wenn die Web-Anwendung aktuell nicht im Vordergrund ist und der Nutzer zum Beispiel einen anderen Tab geöffnet oder eine andere Anwendung im Fokus hat. Bevor der Nutzer eine Benachrichtigung erhalten kann, muss er dies zuvor erlauben.

Für alle Aufgaben des Notification API wird das `Notification`-Objekt definiert. Dieses Objekt enthält im Attribut `permission` den aktuellen Status der Erlaubnis. Mögliche Werte hierfür sind:

```
if (document.visibilityState) {
  window.addEventListener('visibilitychange', (event) => console.log(document.visibilityState));
}
```

Listing 3: Beispiel für das Page Visibility API

```
if (window.navigator) {
  console.log('Status', navigator.onLine);
  window.addEventListener('online', (event) => console.log('Wir sind online'));
window.addEventListener('offline', (event) => console.log('Wir sind offline'));
}
```

Listing 4: Beispiel für NavigatorOnLine

```
if (window.Notification) {
  window.Notification.requestPermission((permission) => {
     if (permission === "granted") {
        const notification = new Notification('Hallo', {
           body: 'Welt',
           icon: 'images/world.jpg',
        });
        notification.addEventListener('close', (event) => console.log('Nachricht geschlossen'));
     }
  });
}
```

Listing 5: Beispiel für das Notification API

```
if (navigator.share) {
    navigator.share({
        title: 'Java aktuell',
        text: 'Einblick in Web-APIs',
        url: 'https://www.ijug.eu/de/java-aktuell/',
    })
      .then(() => console.log('Erfolgreich geteilt'))
      .catch((error) => console.log('Fehler', error));
  }
}
```

Listing 6: Beispiel für das Web Share API

- `denied`, wenn der Nutzer keine Erlaubnis erteilt hat
- `granted`, wenn der Nutzer die Erlaubnis erteilt hat
- `default`, wenn der Nutzer noch keine Erlaubnis erteilt hat

Um eine Erlaubnis für Benachrichtigungen beim Nutzer zu erfragen, muss die Funktion `Notification.requestPermission()` aufgerufen werden, die ein `Promise` zurückgibt. Dieses enthält den oben erwähnten Status der Erlaubnis, wenn sie erfüllt wird. Ist die Erlaubnis einmal erteilt, kann über das Erstellen einer neuen Instanz des `Notification`-Objekts eine Benachrichtigung erstellt und gleichzeitig auch angezeigt werden. Des Weiteren kann eine Benachrichtigung über ein `options`-Attribut angepasst werden. Auf diese Weise können zum Beispiel ein Text und ein Bild für die Benachrichtigung gewählt werden, wie im *Listing 5* dargestellt wird.

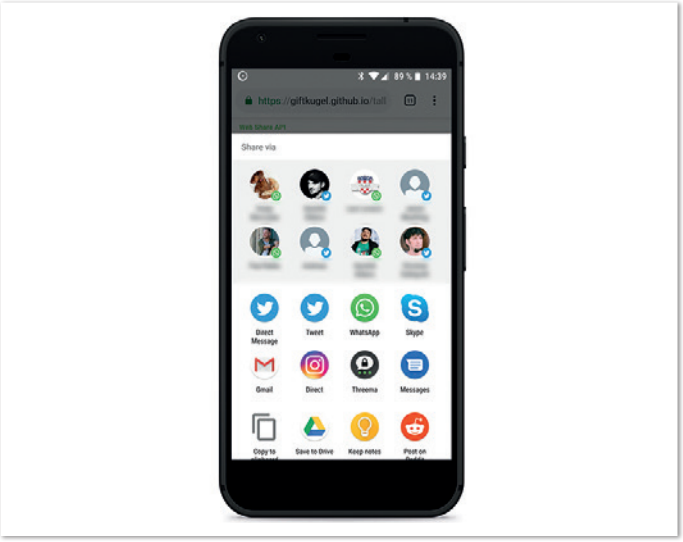


Abbildung 1: Teilen mit dem Web Share API auf Android (Quelle: Simon Skoczylas)

Web Share API

Mit dem Web Share API *[8]* ist es möglich, den nativen Mechanismus zum Teilen von Inhalten zu verwenden *(siehe Abbildung 1)*. Dazu wird das `Navigator`-Objekt im Browser um die `share`-Funktion erweitert. Als Parameter erwartet die `share`-Funktion ein Objekt mit den Attributen `url`, `text` und `title`. Der Rückgabewert ist ein `Promise`, der erfüllt wird, wenn der Nutzer die Operation zum Teilen abschließt und andernfalls zurückgewiesen wird. *Listing 6* zeigt den Aufruf der `share`-Funktion, wobei der Aufruf im Normalfall an eine Nutzeraktion geknüpft werden sollte, zum Beispiel den Klick auf einen Link oder Button.

Web Speech API

Mithilfe des Web Speech API *[9]* kann der Browser Texte in Sprache ausgeben (Text-to-Speech) und sogar Sprache verstehen (Speech Recognition). Zur Ausgabe von Sprache wird eine Instanz von `SpeechSynthesisUtterance` benötigt. Diese kann mit einem Text, der gesprochen werden soll, instanziiert werden. Weitere Attribute können dann optional direkt für die Instanz gesetzt werden. Dazu

```
if (window.speechSynthesis) {
    const utterance = new SpeechSynthesisUtterance('Hallo Welt!');
    utterance.rate = 0.8;
    utterance.lang = 'de';
    utterance.voice = speechSynthesis.getVoices()[0];
    speechSynthesis.speak(utterance);
}
```

Listing 7: Beispiel für das Web Speech API

```
if (navigator.vibrate) {
    navigator.vibrate([200, 50, 200]);
}
```

Listing 8: Beispiel für das Vibration API

Background Sync	Navigation Timing	Vibration
Battery Status	Network Information	Web Audio
Canvas	Notifications	Web Bluetooth
Clipboard	Page Visibility	Web Crypto
Credential Management	Payment Request	WebGL
Fetch	Presentation	WebRTC
Fullscreen	Push	Web Share
Gamepad	Selection	WebSocket
Generic Sensor	Service Worker	Web Speech
Geolocation	Server-sent events	Web Storage
IndexedDB	Shape Detection	WebUSB
Navigator OnLine	Streams	WebVR

Tabelle 1: Übersicht über die Web-APIs

zählen `lang` für die Sprache, `voice` für die Stimme sowie die Attribute `volume`, `rate` und `pitch`, um die Lautstärke, Geschwindigkeit und Tonhöhe zu verändern. Die Attribute `text` und `lang` sind Zeichenketten, `volume`, `rate` und `pitch` Gleitkommazahlen und `voice` eine Instanz von `SpeechSynthesisVoice`, die vom Browser ermittelt werden kann. Um die vorhandenen Stimmen zu erhalten, kann die Methode `getVoices` des `speechSynthesis`-Objekts verwendet werden. Wurden die gewünschten Werte gesetzt, kann die `SpeechSynthesisUtterance` an die `speak`-Funktion des `speechSynthesis`-Objekts übergeben werden, wie in *Listing 7* gezeigt wird.

Vibration API

Um auf die Vibrationshardware des Gerätes zuzugreifen, wurde das Vibration API *[10]* eingeführt. Diese Schnittstelle ermöglicht es dem Nutzer, durch Vibration ein physikalisches Feedback zu geben, zum Beispiel bei einer fehlerhaften Eingabe. Falls das Gerät keine Möglichkeit zur Vibration hat, passiert bei der Verwendung des Vibration API nichts. Für das Vibration API definiert man am `Navigator`-Objekt die Funktion `vibrate`. Diese Funktion erwartet als Parameter einen Wert für die Dauer der Vibration in Millisekunden oder ein Array mit der Dauer für eine Vibration und der Dauer für eine Pause im Wechsel. Zum Beispiel 200, 50, 200 für eine Abfolge von 200ms Vibration, 50ms Pause und 200ms Vibration, wie das *Listing 8* zeigt.

Überblick der vorhandenen Web-APIs

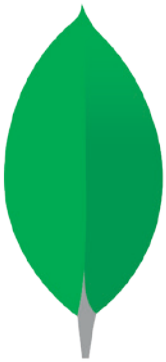
Neben den hier erwähnten Web-APIs bietet das Web noch eine Vielzahl weiterer Schnittstellen, die darauf warten, genutzt zu werden. *Tabelle 1* soll einen Überblick über die Möglichkeiten geben und vielleicht dazu animieren, sich das eine oder andere Web-API noch mal im Detail anzuschauen. In den nächsten Jahren wird die Liste mit Sicherheit noch etwas länger, es bleibt spannend.

Quellen

[1] *https://webassembly.org/*
[2] *https://developer.mozilla.org/en-US/docs/Web/API*
[3] *https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Feature_detection*
[4] *https://modernizr.com/*
[5] *https://www.w3.org/TR/page-visibility-2/*
[6] *https://www.w3.org/TR/html5/browsers.html#browser-state*
[7] *https://www.w3.org/TR/notifications/*
[8] *https://wicg.github.io/web-share/*
[9] *https://w3c.github.io/speech-api/*
[10] *https://w3c.github.io/vibration/*

Simon Skoczylas
simon@skoczylas.net

Simon Skoczylas ist Senior Software Engineer bei der Karakun AG am Standort Dortmund. Er ist leidenschaftlicher Softwareentwickler und Softwarearchitekt. Seine Erfahrungen sammelte er vor allem in der Entwicklung von Software auf Basis von Java und JavaScript. Sein Schwerpunkt liegt auf der Erstellung von Web-Anwendungen.



MongoDB: Paradigmenwechsel im Arbeiten und Verwalten von Daten

Dr. Christian Kurze, MongoDB

Performante, elastisch skalierbare und global verfügbare Applikationen entscheiden heute über Erfolg oder Misserfolg eines Unternehmens. Doch wie lassen sich neue digitale Produkte und moderne Applikationen am besten entwickeln? Während DevOps und Microservices eine sehr gute Antwort liefern, bereitet die Arbeit mit Daten oft Kopfschmerzen – die klassischen relationalen Konzepte stoßen an ihre Grenzen. Dieser Artikel gibt einen Überblick zu MongoDB und zeigt, wie mithilfe des Dokumentenmodells, nativer Hochverfügbarkeit und horizontaler Skalierbarkeit eine Deployment-unabhängige, global verteilbare Datenplattform bereitgestellt werden kann.

Einleitung

Warum profitieren 88 Prozent der CIOs *[1]* trotz agiler Methoden, Microservices, Cloud und neuen Technologien wie Machine Learning, IoT und Blockchain noch nicht von ihrer Digitalisierungsstrategie? Unsere Erfahrung aus der Arbeit mit Tausenden von Unternehmen – Startups bis hin zu Fortune-100-Unternehmen – zeigt, dass die Arbeit mit Daten als Herz jeder Applikation schwerfällig ist:

- Starre Datenmodelle und Wasserfall-ähnliche Entwicklungsmodelle erschweren kurze Release-Zyklen;
- Datenmengen wachsen massiv mit neuen und sich ständig verändernden Datentypen;
- Schwierigkeiten in der Nutzung verteilter Systeme und Cloud-Computing versperren den Zugang zu on demand und hoch skalierbarer Rechen- und Speicherkapazität;
- Hohe Anforderungen existieren bezüglich Datensouveränität und -lokalität, nicht zuletzt durch die neue Europäische Datenschutzgrundverordnung.

MongoDB adressiert diese Herausforderungen: Das JSON-basierte Dokumentenmodell erhöht mit seiner Schemafreiheit die Produktivität und nimmt Daten in beliebiger Struktur auf. Als verteiltes Datenbanksystem stehen Hochverfügbarkeit, horizontale Skalierbarkeit, Workload Isolation sowie global verteilte Cluster als Funktionalität out of the box zur Verfügung. Der Plattform-Gedanke erlaubt es, beliebige Deployment-Umgebungen zu kombinieren: vom Laptop über Mainframe, virtuelle Maschinen, Docker/Kubernetes sowie Cloud- und hybriden Modellen bis hin zum komplett gemanagten Service (DBaaS) in der Cloud auf Basis von AWS, GCP und Azure. MongoDB Mobile deckt zusätzlich mobile Geräte und Edge-Devices ab.

Entwicklungsproduktivität: Das Dokumentenmodell

Nachdem sich relationale Datenbanken in den letzten 40 Jahren als Standard zur Datenhaltung, -nutzung und -anreicherung etabliert haben, ist es an der Zeit, den schemagebundenen Ansatz abzulösen. Das JSON-basierte Modell kombiniert vier Vorteile:

- **Einfachheit** durch die Arbeit mit Daten in einer intuitiven Art und Weise, inklusive Strong Consistency und Transaktionen über mehrere Dokumente;
- **Flexibilität:** Anpassungen an Schemata während der Laufzeit ermöglichen Code-only Deployments;
- hohe **Performance** durch Bündelung von Strukturen bei gleichzeitiger Reduktion von Code; sowie
- **Vielseitigkeit** durch die Unterstützung einer breiten Palette an Datenmodellen, Beziehungen und Abfragen.

Ein Beispiel zur Kundenverwaltung soll die Unterschiede verdeutlichen: Spaltet das relationale Modell die Entität „Kunde" in ver-

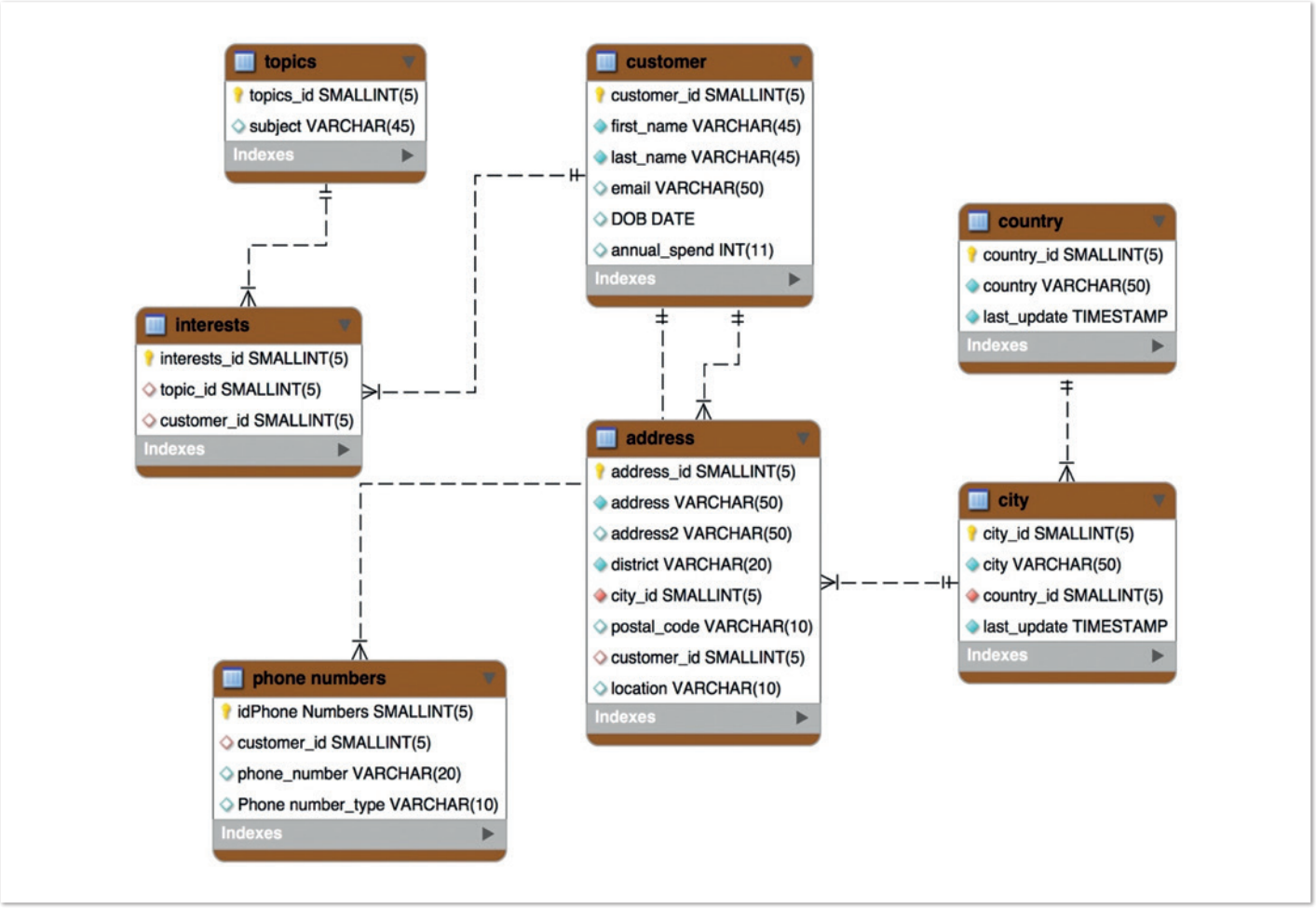


Abbildung 1: Modellierung eines Kunden im relationalen Modell – Daten sind über mehrere Tabellen verteilt (Quelle: Dr. Christian Kurze)

schiedene Tabellen auf *(siehe Abbildung 1)* – und begründet somit die Notwendigkeit von ORM-Layern zur Überwindung des „object-relational impedance mismatch" –, repräsentiert im Dokumentenmodell eine einzige Datenstruktur ein Objekt entsprechend den Anforderungen der Applikation. In Beziehung stehende Strukturen werden als Subdokumente und Arrays eingebettet. Das JSON-Dokument in *Abbildung 2* zeigt einen exemplarischen Kunden.

Die physische Speicherung als Binary JSON (BSON) erlaubt erweiterte Datentypen, wie int, long, date, floating point und decimal128. Sie sind insbesondere für verlustfreie Finanz- und wissenschaftliche Berechnungen sowie zum Vergleich und Sortieren von Daten notwendig.

Idiomatische Treiber für alle populären Programmiersprachen wie Java, JavaScript, C/C++, C#/.NET, Python, PHP, Scala (und über 30 weitere) sowie für analytische Zwecke, zum Beispiel Spark und R (siehe auch Workload Isolation weiter unten), machen Daten intuitiv zugänglich. Entwickler können also ihre üblichen Werkzeuge verwenden – Neuentwicklung und Einarbeitung in bestehende Projekte erfolgen somit viel schneller. Über SQL greifen Reporting-Werkzeuge direkt auf Dokumente, inklusive automatischen Mappings auf Tabellen und Views, zu.

Garantien zur Konsistenz von Strong Consistency über Eventual Consistency und Causal Consistency (read your own writes) bis hin zu strikter Linearisierbarkeit sind Cluster-weit oder bis auf Einzelabfrage pro Use Case konfigurierbar und werden durch die Datenbank forciert (siehe auch Hochverfügbarkeit weiter unten).

Cross-Document ACID-Transaktionen, einschließlich Commit- und Rollback-Funktionen sowie der notwendigen Isolations- und All-Or-Nothing-Garantien, stehen für Veränderungen an mehreren Dokumenten, etwa einem Banktransfer, bei dem Debitor und Kreditor unbedingt den gleichen Betrag atomar abgebucht beziehungsweise gutgeschrieben bekommen sollen, zur Verfügung.

Die **Flexibilität des schemafreien Ansatzes** ermöglicht das Hinzufügen neuer Daten und weiterer Attribute ohne jegliches `CREATE`- oder `ALTER TABLE`-Statement. Eine Reihe von Schema-Design-Patterns beschreiben Best-Practices *[2]*. Der sogenannte „Polymorphismus" erlaubt beispielsweise, dass Kunden unterschiedlich strukturiert sein können: Alle Kunden haben eine ID, optional je nach Kundengruppe einen Social-Media-Account oder Geodaten aus der mobilen App.

JSON-Schema dient der **Schema-Governance**: DevOps und DBA-Teams machen klare Vorgaben für jede Collection, inklusive der gerade beschriebenen Mischung aus Pflichtwerten und freien Attributen. Die Datenbank weist Dokumente ab oder gibt eine Warnung, sollten diese den Regeln widersprechen.

Verbesserte **Performance** entsteht durch das Speichern von zusammengehörigen Daten in einem Dokument. Physisch ist im Kunden-

beispiel also nur ein einziger, gegenüber herkömmlich sieben, Festplattenzugriff notwendig. Verknüpfungen über mehrere Collections erlaubt der `$lookup`-Operator.

Die **Vielseitigkeit des Dokumentenmodells** erlaubt unter anderem die Modellierung von tabellenartigen Strukturen, Key-Value-Paaren, Geodaten sowie Knoten und Kanten von Bäumen und Graphen innerhalb einer einzigen Technologie.

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John",
    "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : {
        "street" : "16 Hatfields",
        "city" : "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
        51.5065752,-0.109081]}},
+   {...}
  ],
  "phone" : [
    { "location" : "work",
      "number" : "+44-1234567890"},
+   {...}
  ],
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")
}
```

Abbildung 2: Modellierung eines Kunden als JSON-Dokument (Quelle: Dr. Christian Kurze)

Ausdrucksstarke Abfragen	Finde alle Kunden einer bestimmten Stadt, die nicht auf der „Do not Call"-Liste stehen
Geodaten	Finde das beste Angebot für einen Kunden, der sich gerade auf den Koordinaten der Maximilianstraße in München befindet
Volltextsuche	Finde alle Tweets, die einen bestimmten Firmennamen innerhalb der letzten beiden Jahre aufweisen
Facettierte Navigation	Filtere alle Produkte < 50,00 Euro, Größe L und hergestellt durch ExampleCo
Aggregation	Zähle und sortiere die Kunden nach Stadt, berechne minimalen, maximalen und durchschnittlichen Umsatz
Nativer Support für Binary JSON	Füge eine weitere Telefonnummer zu einem Kunden hinzu, ohne das Dokument auf Client-Seite neu zu schreiben; Update von zwei aus zehn Telefonnummern; Sortierung nach Modifikationsdatum
Feingranulare Array-Operationen	In den Scorings eines Kunden: Update jeden Score, der < 70 ist, auf 0
JOIN ($lookup)	Finde alle Kunden aus München, suche alle Transaktionen und summiere den Betrag pro Kunde
Graph-Abfragen ($graphLookup)	Finde alle Personen, die über max. drei Stufen mit einem Kunden in Beziehung stehen.

Tabelle 1: Beispiele für umfassende Abfragen

Die **ausdrucksstarke Abfragesprache** (Beispiele in *Tabelle 1*), ergänzt um **Indexierung sekundärer Attribute** *(siehe Tabelle 2)*, unterstützt einfache CRUD-Operationen („Create, Read, Update, Delete") bis hin zu anspruchsvollen Pipelines für Analysen und Transformationen.

Verteilte Datenhaltung: Hochverfügbarkeit, Skalierung, Workload Isolation und Datenlokalität

Applikationen werden heute für Tausende oder Millionen von Nutzern entwickelt; 24/7-Erreichbarkeit von jedem Gerät aus und elastische Skalierbarkeit sind selbstverständlich geworden. Nutzer erwarten konsistente und minimale Antwortzeiten unter Wahrung von Datenschutz und Datensicherheit. Zur **Hochverfügbarkeit** erstellen die meisten relationalen Systeme eine Spiegelung des Datenbestands. Infolgedessen werden zusätzliche Werkzeuge zur Datenreplikation oder zum Erkennen von Fehlern benötigt, die ein Umschalten der Datenbanken im Fehlerfall initiieren. Im Gegensatz dazu arbeitet eine verteilte Datenplattform mit eingebauten Mechanismen zur Replikation. Diese sogenannten „Replica Sets" stellen einen Verbund von Servern dar, sind selbstheilend und führen ein Failover vollautomatisch durch *(siehe Abbildung 3)*. Zudem wird mit dieser Architektur auch eine rollierende Wartung möglich, etwa das Einspielen von Upgrades ohne Downtime.

Um Strong Consistency zu gewährleisten, erhält ein Member die Rolle des Primary Servers, gegen den alle Schreiboperationen stattfinden (zur horizontalen Skalierung siehe weiter unten). Die weiteren Mitglieder des Replica Set arbeiten als Secondary Server und replizieren alle Veränderungen des Primary Servers auf Basis des Operations Log. Es besteht aus einer geordneten Menge idempotenter Operationen, die auf die Replicas kopiert werden.

Im Falle des Fehlers eines Primary Servers wird einer der Secondary Server vollautomatisch innerhalb weniger Sekunden zum neuen Primary Server gewählt; Applikationen werden automatisch auf den neuen Primary Server umgeleitet. Tritt der ehemalige Primary Server wieder dem Cluster bei, geht er automatisch in den Secondary Status und synchronisiert seine Daten vom (neuen) Primary Server. Schreibverfügbarkeit ist auch während des Failover-Vorgangs durch Retryable Writes gegeben: Nicht ausgeführte Schreiboperationen werden automatisch gemäß dem „exactly once"-Prinzip wiederholt.

Die bis zu 50 Server innerhalb eines Replica Set erlauben die Verteilung von Daten über Rechenzentren hinweg und automatischen Failover im Falle des Totalausfalls eines Rechenzentrums, aber auch globale Verteilung von Daten für schnelle Lesezugriffe. Entwickler können die Replica Sets dabei flexibel konfigurieren:

- Sicherstellen von Schreiboperationen auf spezifische Server eines Replica Set – lokal und in entfernten Standorten. Der „write concern" konfiguriert Regeln zur Bestätigung einer Datenbankoperation, beispielsweise das Schreiben auf mindestens zwei Servern in einer Region sowie mindestens einem Server einer weiteren Region.
- Sicherstellen von garantierten Latenzen von Leseoperationen, zum Beispiel basierend auf der physischen Lokation eines Servers. Die „read preference" vom Typ „nearest" wählt automatisch den Server mit der geringsten Ping-Zeit aus, im Fehlerfall automatisch einen anderen Server in der Nähe. Explizite Nodes können über Tags angesprochen werden.

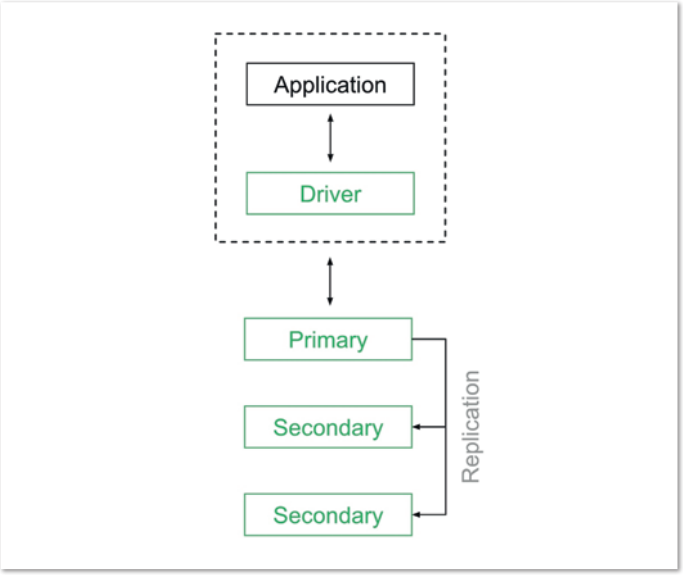


Abbildung 3: Selbstheilende Replica Sets (Quelle: Dr. Christian Kurze)

Index-Typen	Index-Features
Primary Index: Jede Collection hat einen Primary Key Index	TTL Index: Index auf einem Datumsfeld; sobald die Gültigkeit abgelaufen ist, lösche das Dokument
Compound Index: Index auf mehreren Attributen im Dokument	Unique Index: Stellt sicher, dass alle Werte nur einmal vorkommen
MultiKey Index: Index auf Arrays	Partial Index: Basierend auf einem Ausdruck, erlaubt Indizes auf einem Subset von Daten
Text Index: Volltextsuchen	Case Insensitive Index: Textsuche ohne Unterscheidung der Groß- und Kleinschreibung
GeoSpatial Index: 2d & 2dSphere Index für Geodaten	Sparse Index: Nur Dokumente mit einem bestimmten Feld im Index
Hashed Index: Hash-basierte Werte für Sharding	

Tabelle 2: MongoDB unterstützt vollwertige sekundäre Indizes

Replica Sets erlauben zusätzlich **Workload Isolation zwischen operativen und analytischen Aufgaben innerhalb desselben Clusters** *(siehe Abbildung 4)*. Explorative Abfragen oder das Training von Machine-Learning-Modellen erfolgen ohne Einfluss auf operative Applikationen. Zusätzliche Secondary Server werden vollautomatisch mit allen Daten der operativen Applikation ETL-frei und in Echtzeit versorgt. Zur Realisierung von **Echtzeit-Data-Pipelines** lauschen Applikationen über sogenannte „Change Streams" auf Veränderungen in der Datenbank und triggern Aktionen. Dieser reaktive Programmiermodus eröffnet diverse Use Cases: Trading-Applikationen, die in Echtzeit auf steigende/fallende Kurse reagieren müssen; Aktualisierung von Dashboards in analytischen Systemen oder Suchmaschinen; IoT-Data Pipelines, die auf den Zustand der physischen Geräte reagieren; Synchronisation von Daten in Serverless- oder Microservice-Architekturen.

Horizontale Skalierung (alias Sharding oder Partitionierung) auf Basis von Standard-Hardware oder Cloud-Infrastruktur ist die Antwort auf Anforderungen zu hohen Datenvolumina und Durchsatz bei geringen Kosten. Die Verteilung von Daten auf Shards (jeweils ein hochverfügbares Replica Set) erfolgt automatisch über einen frei definierbaren Shard Key. Shards können, transparent für die Applikation und je nach Anforderung, hinzugenommen, aber auch wieder aus dem Cluster entfernt werden: Egal ob ein oder Hunderte von

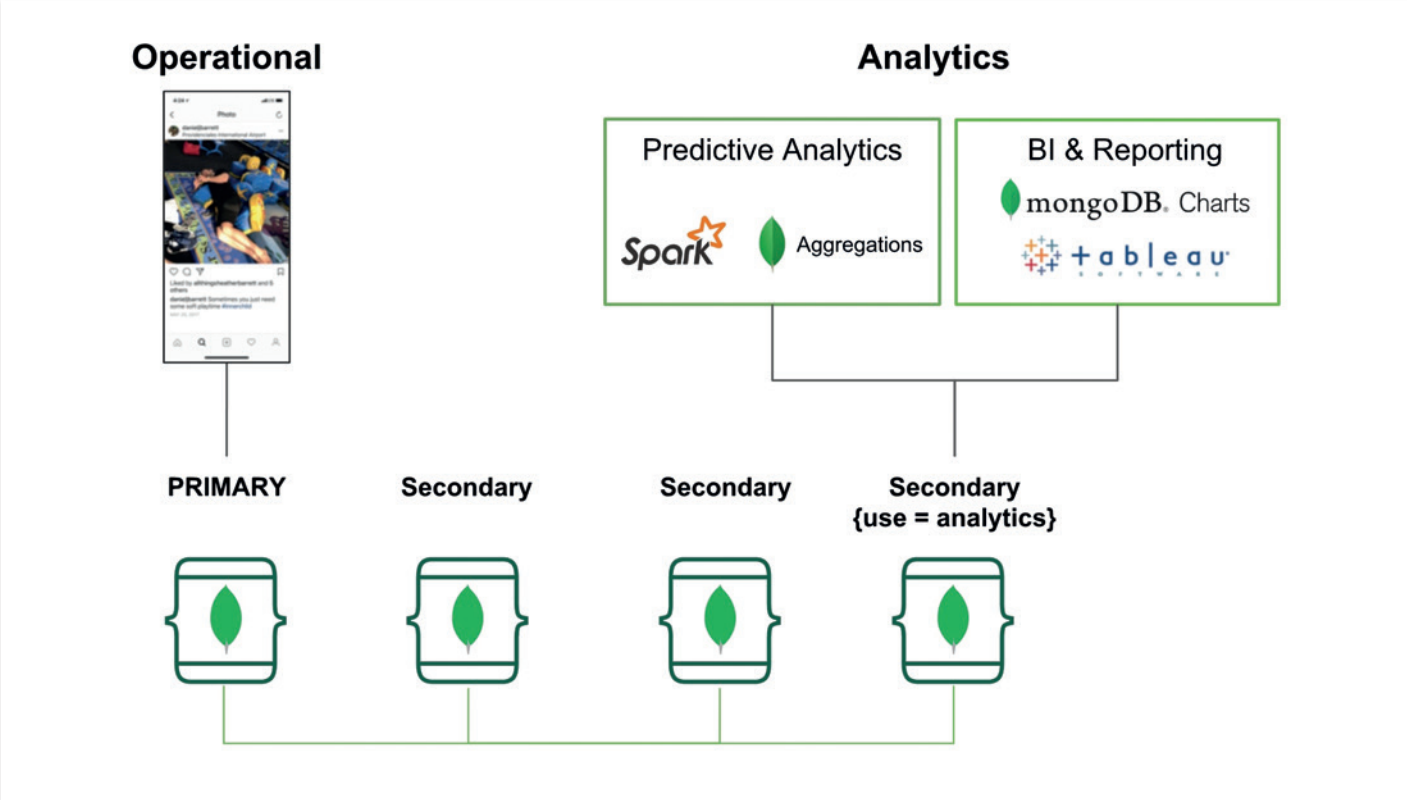


Abbildung 4: Kombination von operativen und analytischen Workloads (Quelle: Dr. Christian Kurze)

Shards, der Quellcode sieht exakt identisch aus; es ist keine zusätzliche Logik in der Applikation notwendig; ebenso wenig spezielle Cluster-Software oder verteilte Storage-Systeme. Abfragen werden über Query Router (deren Anzahl sich aus den Anforderungen zu Performance und Verfügbarkeit ergibt) an die eigentlichen Shards verteilt *(siehe Abbildung 5)*.

Die meisten Datenbanksysteme bieten nur ein Hash-basiertes Verteilen von Daten auf Partitionen. In Abhängigkeit von der Anwendung sollten jedoch mehrere Mechanismen zur Verfügung stehen:

- **Ranged Sharding:** Dokumente mit ähnlichen Schlüsseln liegen sehr wahrscheinlich auf demselben Shard. Dieser Ansatz eignet sich für Range-based Queries sowie zur gemeinsamen Ablage von Kunden einer spezifischen Region in einem Shard;
- **Hashed Sharding:** Dokumente werden anhand eines MD5-Hash des Shard Key verteilt. Dieser Ansatz garantiert eine uniforme Verteilung von Schreiboperationen;
- **Zone Sharding** *(siehe Abbildung 6)*: Dieser Ansatz erlaubt die Definition von spezifischen Regeln zur Ablage von Daten im gesharded Cluster. Zones sind ideal, um ein **Datenlokalitätsprinzip** umzusetzen: Entweder nach geographischer Region (Zonen für Nordamerika, Europa und Asien) oder auch nach Serviceklassen (schnelle Hardware für Premium-Kunden oder heiße Daten, langsamere für lauwarme/kalte Daten). Jede Zone kann dabei mehrere Shards enthalten, was wiederum zu einer unabhängigen Skalierbarkeit der einzelnen Zonen führt.

Umfassende **Sicherheitsmechanismen** runden die Möglichkeiten ab: Authentifikation (LDAP/Active Directory, Kerberos, x.509-Zertifikate, IP-Whitelisting), Autorisierung (auf Basis von rollenbasierter Zugriffskontrolle mit granularen Berechtigungen pro User/Rolle), Auditierung (für regulatorische Compliance; DDL, DML, DCL) sowie Verschlüsselung (at Rest und in Flight).

Vorteile einer einheitlichen Datenplattform

MongoDB kann in diversen Umgebungen eingesetzt werden: Laptop, Mainframe, Docker/Kubernetes, Private und Public Cloud sowie in hybriden Szenarien. MongoDB Mobile erlaubt weiterhin die automatische Synchronisation von mobilen und Edge-Devices. Entwickler finden überall die gleiche Umgebung vor; ebenso stehen Ops-Teams identische Werkzeuge für Monitoring, Optimierung, Automatisierung und Backup zur Verfügung. Die Kombination verschiedener Umgebungen erlaubt es Unternehmen, Schritt für Schritt in die Cloud zu gehen. So können Workloads in hybriden Umgebungen unerwartete Lastspitzen ausgleichen oder die Cloud wird für geographische Regionen verwendet, in denen keine eigenen Rechenzentren zur Verfügung stehen. Ein Lock-in auf einen bestimmten Infrastruktur- oder Cloud-Provider besteht nicht.

Im Rahmen eines gemanagten Service kann der Aufwand für DevOps-Teams minimiert werden. Die Vorteile liegen klar auf der Hand: Automatisierter Self-Service und Elastizität; Hochverfügbarkeit; Globale Cluster; Security by Default; umfassendes Monitoring und Performance-Optimierung sowie Mechanismen zur Live-Migration von Daten aus On-Premises-Rechenzentren oder anderen Cloud-Anbietern.

Zusammenfassung

Kritische Erfolgsfaktoren für die Digitale Transformation sind Geschwindigkeit und Qualität: Wie schnell ist ein Unternehmen in der Lage, neue Applikationen zu entwickeln, sie zu skalieren und Erkenntnisse aus den generierten Daten zu gewinnen? Diese Elemente sind der Schlüssel für bessere Kundenerfahrungen, tiefere und datengetriebene Einsichten in das Kundenverhalten sowie neue Produkte und Geschäftsmodelle.

Um dies zu gewährleisten, sollte eine moderne Datenplattform drei wesentliche Punkte erfüllen:

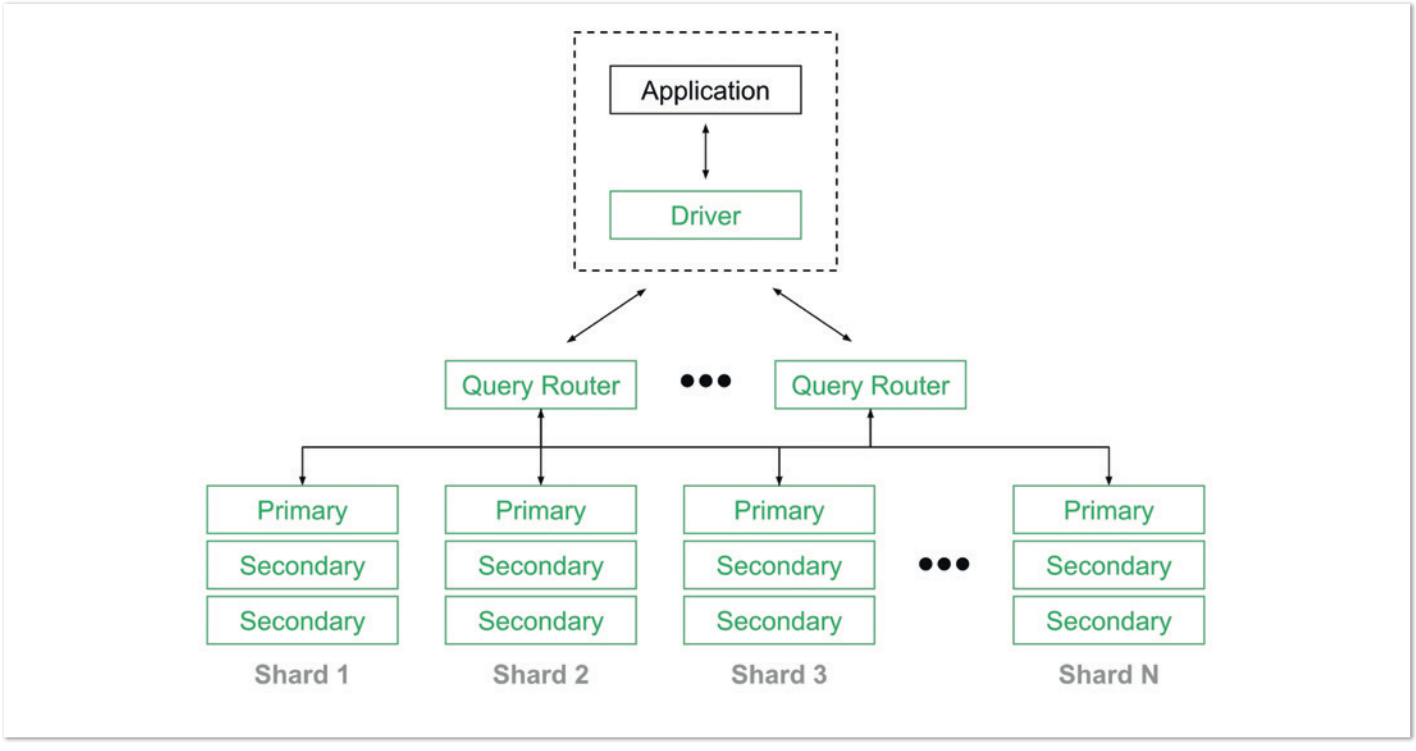


Abbildung 5: Automatisches Sharding für horizontale Skalierung (scale-out und scale-in) (Quelle: Dr. Christian Kurze)

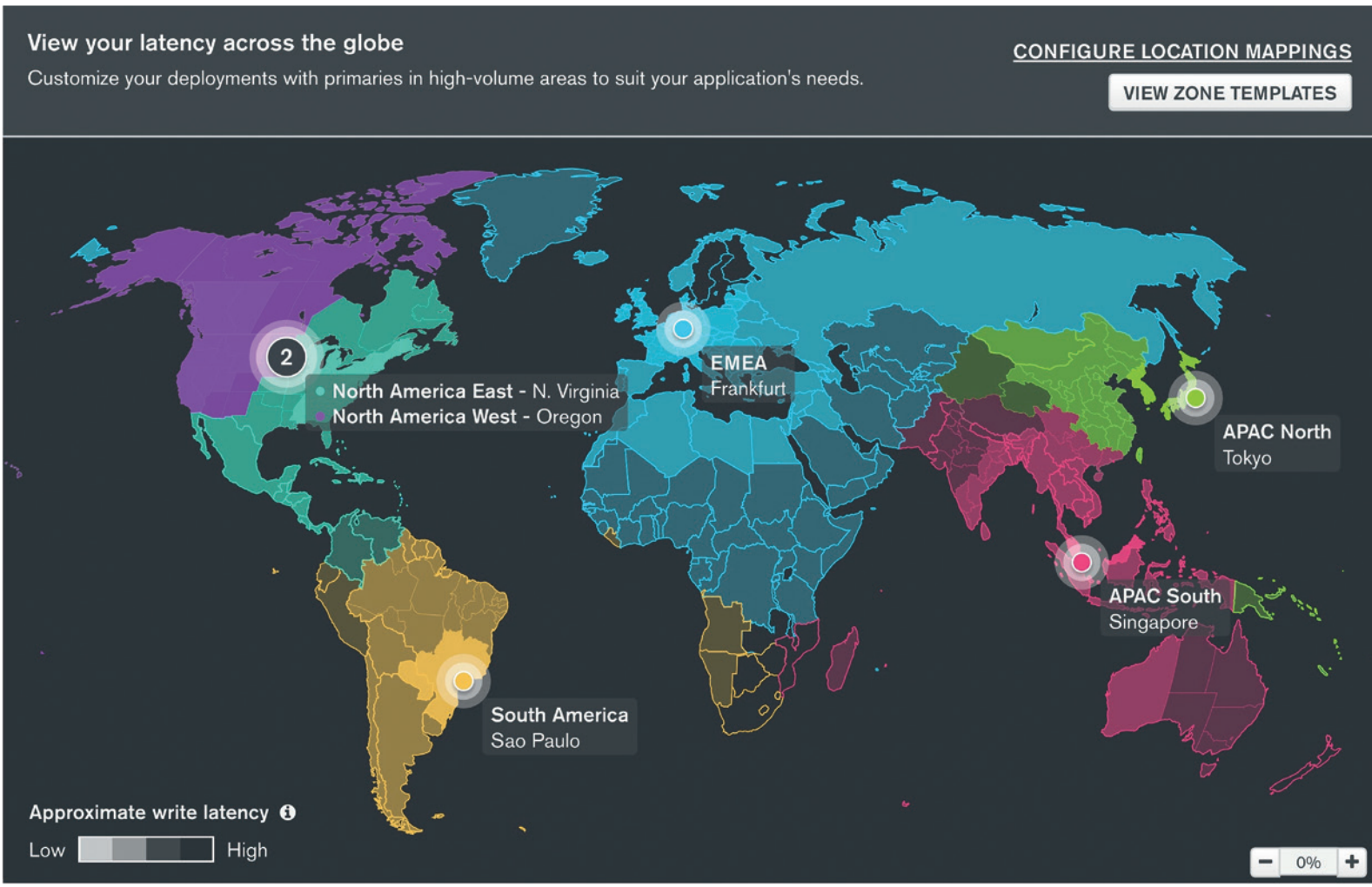


Abbildung 6: Grafischer Assistent zur Erstellung eines global verteilten Clusters mit minimalen Lese- und Schreib-Latenzen auf Basis von Zone Sharding (Quelle: Dr. Christian Kurze)

- Schnelle, einfache, flexible und vielseitige Arbeit mit Daten – das Dokumentenmodell erweist sich als ideal und entspricht der natürlichen Denk- und Arbeitsweise von Fachabteilungen und Entwicklern;
- Verteiltes Systemdesign – Hochverfügbarkeit, automatische Skalierung sowie Kombination von operativen und analytischen Workloads in Echtzeit unter der Beachtung von Datenlokalität;
- Einheitliche Nutzung für Entwickler auf allen Plattformen, um zukunftssichere Applikationen zu erstellen und einen Vendor Lock-in zu vermeiden.

Dieser Artikel hat die fundamentalen Konzepte von MongoDB in diesen drei Säulen dargelegt und soll eine Anregung zum Paradigmenwechsel in der Arbeit mit und der Verwaltung von Daten sein.

Quellen

[1] Gemäß einer Studie von Harvey Nash und KPMG *(https://home.kpmg.com/xx/en/home/insights/2017/05/harvey-nash-kpmg-cio-survey-2017.html)*

[2] Siehe u.a. *https://www.mongodb.com/blog/post/building-with-patterns-the-polymorphic-pattern*

Dr. Christian Kurze
christian.kurze@mongodb.com

Dr. Christian Kurze arbeitet als Senior Solutions Architect bei MongoDB. Als Experte für Datenmanagement und Datenintegration liegt sein Fokus in der Generierung von Mehrwert aus Daten. Vor seiner Tätigkeit bei MongoDB war er in den Feldern Datenvirtualisierung, Business Intelligence und aktivem Metadatenmanagement aktiv. Herr Kurze promovierte zur Automatisierung von Data-Warehouse-Systemen und ist Autor von Fachbeiträgen sowie regelmäßiger Sprecher auf Konferenzen.

Agiles Requirements Engineering – Klassische und agile Methoden im Anforderungsmanagement erfolgreich vereinbaren

Lars Möller, Pentasys

Agile Methoden setzen sich nach und nach in Unternehmen durch. Dennoch findet man auch heute noch Prozesse, die sich nach der Wasserfall-Methode richten. Dies ist unter anderem dadurch zu verstehen, dass agile Methoden einerseits Ungewissheit in Bezug auf das Endprodukt mit sich bringen, andererseits kann und darf aufgrund regulatorischer sowie geschäftlicher Anforderungen nicht auf feste Meilensteine verzichtet werden.

In diesem Dilemma entscheidet sich das Management häufig für die „konservative" Variante: Nämlich weiterhin auf die klassischen Methoden des Anforderungsmanagements zu setzen. Dabei dominiert die Sicht auf Agilität als Zusatzrisiko, denn iterative Änderungen und noch nicht feststehende Ergebnisse lassen sich in Modellen schlecht darstellen.

Wird ein solches Vorgehen gewählt, bremst der klassische Ansatz jedoch die volle Anwendung der agilen Vorgehensweise aus – und damit auch ihre Vorteile. Aus dieser Herausforderung entsteht daher die Notwendigkeit, klassisches Requirements Engineering und agiles Projektmanagement in Einklang zu bringen. Im Folgenden erfahren Sie, wie dies gelingen kann.

Wo klassisches Requirements Engineering und Agilität auseinandergehen

Das Anforderungsmanagement gehört zu den elementarsten Bereichen der Softwareentwicklung und ist dementsprechend stark standardisiert. Dies findet sich beispielsweise in den ISO-Normen ISO/IEC 12207 und ISO/IEC 15504 wieder, sowie in der Anwendung formalisierter Methoden zur Anforderungsspezifikation, beispielsweise Lastenhefte oder Templates zur Beschreibung der Anforderungen. Daneben haben sich eigene Dateiformate etabliert, wie etwa das ReqIF (Requirements Interchange Format) zum Austausch von Daten zwischen verschiedenen Softwarewerkzeugen.

Auch heute noch werden diese Methoden in einem Prozess implementiert, der an das Wasserfall-Modell erinnert. Dieser schreibt vor, dass Auftraggeber und Auftragnehmer die Anforderungen an das Projekt „Up-Front", also bereits vor Projektbeginn, definieren müssen. Über den Projektverlauf hinweg werden sie dann Phase für Phase abgearbeitet. Dass dieses Vorgehen nach wie vor zum Einsatz kommt, ist insofern verständlich, als hiermit eine sehr granulare und präzise Festlegung des Projektablaufs und seiner Resultate im Vorab möglich ist. Insbesondere aus der geschäftlichen Perspektive kann die klassische Methode attraktiv erscheinen, weil sie die Allokation von Personal und Ressourcen zunächst vereinfacht und Unwägbarkeiten definitorisch ausschließt.

Das agile Modell hingegen ist darauf ausgelegt, die Anforderungen kontinuierlich zu prüfen und gegebenenfalls anzupassen. Dies betrifft auch die Validierung der zugrunde liegenden Annahmen der definierten Anforderungen, zum Beispiel bestimmte Vorlieben der Endnutzer. Anforderungsmanagement findet daher laufend statt, von Beginn bis Ende des Projekts. Hieraus ergibt sich allerdings eine gewisse Offenheit in der Entwicklungsrichtung und bei den zu erwartenden Resultaten. Dies steht der Vorstellung von Requirements Engineering als a priori definiertem „Masterplan" eindeutig entgegen!

Auch agile Projekte brauchen Requirements Engineering

Wie lassen sich die beiden unterschiedlichen Vorgehensweisen also auf einen gemeinsamen Nenner bringen? Diese Frage sollte nicht unbeantwortet bleiben. Denn einerseits würde eine Rückkehr zum Projektmanagement nach der Wasserfall-Methode auch deren klare Nachteile gegenüber agilen Methoden zum Vorschein bringen: Eine verzögerte Time-to-Market, mehr Unsicherheit bei der Auslieferung des Endprodukts sowie ein mögliches Vorbei-Entwickeln an den Nutzeranforderungen wären in einem solchen Szenario der „Preis" für eine umfassendere und transparentere Up-Front-Planung. Die Nachteile überwiegen dabei allerdings die Vorteile. Grundsätzlich gilt daher, dass moderne IT-Entwicklung stets agile Methoden zugrunde legen sollte.

Andererseits wären agile Projekte ohne ein dediziertes Requirements Engineering unvollständig. Gerade in komplexen Projekten mit einem hohen Investitionsaufwand stellt das Requirements Engineering Planungssicherheit und eine Grundlage für das Risikomanagement her. In bestimmten Fällen ist es sogar regulatorische Pflicht. Zu guter Letzt braucht es das Requirements Engineering auch für ein gemeinsames Verständnis unter den beteiligten Stakeholdern. Insbesondere, wenn externe Dienstleister am Projekt beteiligt sind, spielt es eine wichtige Rolle bei der vertraglichen Vereinbarung mit dem Auftraggeber.

Damit wird die „Mission" des agilen Requirements Engineering deutlich: Agiles Projektmanagement in Kombination mit Anforderungsmanagement ermöglichen und dabei die Vorteile beider zur Geltung bringen. Dies kann realistisch gesehen nur funktionieren, indem das agile Denken im Requirements Engineering den Vorrang erhält – wenn also das handwerkliche Rüstzeug des Anforderungsmanagements zwar weitgehend gleichbleibt, die dahinterliegende Philosophie sich aber wandelt.

Das Mindset nach den Prinzipien des Agilen Manifests

Um den (scheinbaren) Widerspruch zwischen Requirements Engineering und agilen Methoden aufzulösen, ist es zunächst die Überlegung wert, auf die Grundprinzipien der Agilität zurückzukommen – wie sie im allseits bekannten Agilen Manifest niedergelegt sind. Hieraus lässt sich ein spezifisch agiles Verständnis für Requirements Engineering entwickeln, aus dem sich in einem zweiten Schritt dann konkrete Tools und Prozesse für die Praxis ableiten. Was Agilität für das Requirements Engineering bedeuten kann, soll anhand der folgenden vier Punkte aus dem Manifest für Agile Softwareentwicklung dargelegt werden:

1. **Individuen und Interaktionen mehr als Prozesse und Werkzeuge:** Auch im Anforderungsmanagement muss das Verständnis für die Bedürfnisse und Wünsche der Anwender und Stakeholder an erster Stelle stehen. Dies macht die zahlreichen definierten Prozesse und Werkzeuge keineswegs obsolet – denn sie spielen eine wichtige Rolle als unterstützende Elemente, um das genannte Verständnis zu erreichen.

2. **Funktionierende Software mehr als umfassende Dokumentation:** Das inhaltliche Augenmerk beim Anforderungsmanagement sollte stets auf der Funktionsfähigkeit der Software liegen. Wichtig dabei ist, diese aus der Anwendersicht zu verstehen. Denn nur, was beim User auch ankommt, funktioniert tatsächlich. Das Requirements Engineering erfüllt in diesem Zusammenhang die Aufgabe, verteiltes Wissen zusammenzuführen und im Projektablauf zu konsolidieren. Dies entbindet allerdings nie von der Pflicht, eine korrekte Dokumentation zu führen, die alle regulatorischen Anforderungen erfüllt. Über die formalen Anforderungen der Dokumentation sollte jedoch die Funktionsfähigkeit der Software nie vergessen werden.

3. **Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung:** Agile Methoden leben vom Informationsaustausch sowie vom gegenseitigen Verständnis der Stakeholder. Dies gilt ganz besonders für das Requirements Engineering, das bereits vor dem Projektstart beginnt. In dieser Phase sind die Informationsbasis und das gegenseitige Verständnis noch am geringsten und müssen erst hergestellt werden. Sehr hilfreich dabei ist beispielsweise die Definierung von User Stories, entlang derer das Produkt über den Projektzeitraum hinweg erarbeitet wird.

4. **Reagieren auf Veränderung mehr als das Befolgen eines Plans:** Agilität bringt es fast zwangsläufig mit sich, dass im Projektverlauf Änderungen auftreten. Erfreulicherweise stellt bereits das klassische Requirements Engineering Methoden und Techniken bereit, um auf solche zu reagieren. Diese können den agilen Prozess sehr gut unterstützen – sofern sie auch in einem agilen Mindset angewendet werden. Um Änderungen schnell und effizient durchführen zu können, ist die Verpflichtung zur Einfachheit in allen Belangen ein wichtiges Gebot.

Diese Betrachtungen mögen auf den ersten Blick sehr allgemein erscheinen – doch zunächst geht es genau darum. Die Anwendungen von Tools und Methoden sollten sich nämlich stets nach dem (agilen) Mindset und seiner Philosophie richten – und nicht umgekehrt. Dies gilt sowohl für das agile Projektmanagement als auch für das agile Anforderungsmanagement. Wer diesen Weg geht, wird schnell herausfinden, dass viele der Prozesse und Werkzeuge, die aus der klassischen Vorgehensweise bekannt sind, auch weiterhin sinnvoll sind. Sie müssen „lediglich" auf agile Weise Anwendung finden!

Fallstricke des agilen Requirements Engineering und wie sie sich lösen lassen

Von der agilen Transformation zu sprechen und sie umzusetzen, sind zweierlei Dinge. Denn in der Praxis lauern einige Herausforderungen, die im reinen Modell nicht abgebildet werden beziehungsweise überhaupt abgebildet werden können. Das klassische Requirements Engineering muss sich dementsprechend adaptieren. Denn schließlich war es in der Vergangenheit des Wasserfall-Modells noch möglich, modellhafte Anforderungen zu Beginn präzise vorzuschreiben, die durch das ganze Projekt hinweg strikt durchexerziert werden. Ob die getroffenen Annahmen tatsächlich zutreffen, wurde in einer sehr späten Phase oder gar erst am Ende des Projekts deutlich. Im agilen Modus hingegen gehen Modell und Empirie stets Hand in Hand.

Ein sehr wichtiger Praxisaspekt ist dabei unter anderem die Implementierung der definierten Anforderungen im laufenden Projekt. Werden nämlich Anforderungen vor Projektstart zwar entwickelt und festgeschrieben, dann jedoch überhaupt nicht beachtet, ist die Sinnhaftigkeit des gesamten Prozesses infrage zu stellen. Die Problemstellung erschwert sich meist dadurch, dass im Verlauf des Projekts dessen ursprüngliches Ziel aus den Augen gerät, da sich die Richtung im Laufe weniger Sprints stark verändern kann. Die Herausforderung besteht also darin, einerseits flexibel auf Änderungen zu reagieren, aber dennoch die definierte Zielsetzung des Projekts kontinuierlich weiterzuverfolgen.

Ein weiterer Aspekt, den es in der Praxis zu berücksichtigen gilt, ist der organisatorische Kontext, in dem das agile Projekt stattfindet. Überlastung der Stakeholder mit Informationen, übermäßige Komplexität sowie kulturelle Hürden, etwa zwischen verschiedenen Abteilungen, sind dabei zu beachten und in das agile Anforderungsmanagement mit einzubeziehen. Ein häufiges Beispiel dafür ist die Zuordnung von Themen an eine Rolle, die in klassischen Organisationsstrukturen in der Regel an eine Hierarchie gekoppelt ist. Im agilen Denken spielt Hierarchie allerdings eine untergeordnete Rolle, wodurch ein Anpassungsbedarf entsteht.

Die genannten Hürden lassen sich nur durch einen Lern- und Reflexionsprozess überwinden, der alle Beteiligten miteinschließt. Das Gelingen dieses Prozesses ist glücklicherweise nicht dem Zufall überlassen, sondern lässt sich mithilfe verschiedener Werkzeuge unterstützen. Drei der wichtigsten werden hier kurz vorgestellt:

- **Design Thinking:**
  Hierbei handelt es sich um eine Methode, die durch einen iterativen Prozess zu innovativen Produkten und Lösungen führen soll. Das Design Thinking ist vor allem für die Klärung von Fragestellungen und die Definierung von Zielen aus Sicht des Anwenders geeignet. Es sollte vor allem in einer sehr frühen Phase eines Projekts zur Anwendung kommen, in der die übergeordnete Zielsetzung noch einer Klärung bedarf.

- **Lean Management:**
  Dieser Oberbegriff versammelt ein Set an wichtigen Prinzipien und Methoden zum Aufbau effizienter beziehungsweise „schlanker" Organisationsformen. Ein wichtiger Aspekt ist auch hier die Kunden- und Anwenderorientierung. So sind eine Organisation oder auch deren Prozesse stets dahingehend zu befragen, ob sie aus Sicht des Kunden tatsächlich notwendig sind. Für das Thema Requirements Engineering ist dies vor allem von Bedeutung, wenn es um die richtige Zuweisung von Befugnissen und Ressourcen innerhalb einer bestehenden Organisation geht.

- **Minimum Viable Product (MVP):**
  Dieses Vorgehen leitet sich aus den Prinzipien des Lean Management ab. Ein MVP ist ein Erzeugnis, das in einer Minimalkonfiguration zumindest eine Anforderung des Users abdeckt. Entwicklungsteams können mithilfe von MVPs Produkte und Lösungen bereits nach kurzer Entwicklungszeit einführen und testen, wodurch sich mögliche Fehlentwicklungen früh korrigieren lassen. Das MVP lässt sich im Rahmen des Anforderungsmanagements zum Beispiel als Meilenstein heranziehen.

Wichtig zu betonen ist, dass die genannten Hilfsmittel nicht nur einmalig zum Einsatz kommen müssen. Sie lassen sich auch zu jedem beliebigen Zeitpunkt des Projektablaufs wiederholen, um die initial aufgestellten Annahmen zu überprüfen und gegebenenfalls zu korrigieren. In der Praxis geht es am Ende vor allem darum, eng entlang der Nutzeranforderungen zu arbeiten. Diese findet man am besten durch kontinuierliches „Testen und Lernen" heraus.

Requirements Engineering lässt sich in Scrum einbetten

Im Vergleich zum Requirements Engineering sind die Werkzeuge und konkreten Abläufe in Scrum weniger formalisiert und standardisiert. Einen wichtigen Ausgangspunkt für Requirements Engineering in Scrum ist das Backlog, über das üblicherweise der Product Owner wacht. Hier sind auch die Anforderungen aus Nutzersicht

hinterlegt. Sie lassen sich anhand der Begriffe „Epic“ und „User Story“ definieren: Während die User Story einen einzelnen Schritt beschreibt, den der Anwender vollbringen muss, um das nächste (Teil-)Ziel zu erreichen, fassen Epics mehrere User Stories in einem Prozess zusammen, der zu einem übergeordneten Ziel führt.

Aus der Verknüpfung der Epics und User Stories entsteht schließlich die sogenannte „Story Map“, in der alle Elemente visuell verknüpft werden. Die Story Map eignet sich besonders gut, um einen schnellen Überblick über alle relevanten Anforderungen zu vermitteln und damit den Wissensaustausch unter den Stakeholdern zu vereinfachen. Der Product Owner kann alle Anforderungen bündeln, indem er Änderungen in der Story Map mit einem neuen Eintrag im Backlog dokumentiert. Damit lassen sich der aktuelle Stand der Anforderungen wie auch alle bisherigen Schritte jederzeit nachvollziehen.

Für die Beschreibung der einzelnen User Stories im Backlog können dann wiederum die bekannten Standards aus dem klassischen Requirements Engineering zum Einsatz kommen, beispielsweise vorformulierte Sätze oder auch Dateivorlagen. Was aus dem klassischen Handwerk sinnvoll ist oder auch nicht, ist je nach Projektkontext zu entscheiden. Wichtig ist, dass jede User Story den sogenannten „INVEST“-Kriterien entspricht: Independent (unabhängig), Negotiable (verhandelbar), Valueable (wertorientiert), Estimatable (abschätzbar), Small (klein), Testable (testbar).

Wie bereits erwähnt, kommt agiles Requirements Engineering über den gesamten Lebenszyklus eines Projekts hinweg zum Einsatz. Eine besonders wichtige Rolle spielt es jedoch in der Planungsphase vor dem eigentlichen Projektbeginn. Hier entscheiden sich nämlich grundlegende Eckpfeiler wie Projektvision, Transparenz und Teamaufbau. Dabei ist ein besonderes Augenmerk auf das Thema Anforderungen und die entsprechenden User Stories zu legen, um eine adäquate Bemessung der benötigten Ressourcen zu erreichen. Dies erfüllt die Vorgaben des Lean Management und hilft nicht zuletzt dabei, auch die Business-orientierten Stakeholder zu überzeugen. Hervorzuheben ist, dass es für die Einbettung des Requirements Engineering in Scrum kein feststehendes „Rezept“ gibt. Vielmehr hängt die Vorgehensweise stark vom Umfeld ab, wobei das Stakeholder Management eine wichtige Rolle einnimmt.

Lars Möller
lars.moeller@pentasys.de

Lars Möller ist Business Architect bei der PENTASYS AG. Er ist der Ansicht, dass es für die Einbettung des Requirements Engineering in Scrum kein feststehendes „Rezept“ gibt. Vielmehr hängt die Vorgehensweise stark vom Umfeld ab, wobei das Stakeholder Management eine wichtige Rolle einnimmt.

JavaLand4Kids 2019 – Roboter, I/O-Boards und Graphen

Lisa Rosenberg, msg David

Über 2.000 Entwickler aus aller Welt waren dieses Jahr auf der JavaLand-Konferenz zu Gast. Unter ihnen waren auch in diesem Jahr wieder zahlreiche an der Informatik interessierte Schülerinnen und Schüler. Sie lernten auf der JavaLand4Kids in den insgesamt fünf angebotenen Workshops viel Neues zu den Themen Robotik, Sensorik, I/O-Boards, Programmierung und Graphendatenbanken kennen.

Bereits in der fünften Iteration findet nun schon das erfolgreiche JavaLand4Kids am Vortag der Entwicklerkonferenz JavaLand im Phantasialand statt. Wie schon die letzten Male hat sich das Event auch dieses Mal vergrößert: Waren es 2018 noch etwa 40 Kinder und Jugendliche im Alter zwischen 9 und 18 Jahren, so nahmen dieses Jahr ganze 50 Teilnehmer teil. Ingesamt waren damit 20 Grundschüler und 30 Schüler aus der gymnasialen Oberschule regionaler Schulen auf der JavaLand4Kids 2019. Das Format spricht sich in den umliegenden Schulen herum und wird von Jahr zu Jahr populärer. Einige Schüler nahmen sogar schon zum wiederholten Male teil.

Einblick in den Tagesablauf

Gegen 9:00 Uhr trafen die Schüler im STOCK's beim Phantasialand mit dem Reisebus ein. Eine halbe Stunde später folgte die Begrüßung durch die beiden Haupt-Organisatoren der JavaLand4Kids Uwe Sauerbrei (Gründer der Kids4IT e.V. in Hamburg sowie Organisator der Mentoren) und Marcel Trebes (Sales- & Event-Manager der DOAG Dienstleistungen GmbH). Den Schülern sah man während der Begrüßung bereits an, dass sie schon sehr gespannt auf den Tag waren.

Bereits im Vorfeld haben sich die Schüler für einen Workshop entschieden, den sie gerne besuchen möchten. Nach der Begrüßung verteilten sich die Teilnehmer daher gemäß ihrer getroffenen Wahl auf die fünf unterschiedlichen Workshops. Nach einer kurzen Kennenlernrunde und Themenvorstellung in den kleineren Gruppen begann auch schon der Hauptteil der Veranstaltung: die Kinder aus der Grundschule lernten abwechselnd den "Calliope mini" und die kleinen Lego-WeDo-Roboter kennen, wohingegen sich die älteren Schüler entweder mit Lego Mindstorms EV3, der Graphendatenbank Neo4j oder VEX IQ beschäftigten.

Um 12:00 Uhr gab es eine Mittagspause, in der sich die Schüler bei leckerem Essen stärken konnten, bevor es in die zweite Tageshälfte ging. In dieser wechselten die jüngeren Schüler den Workshop und lernten damit ein zweites Thema kennen. Die älteren arbeiteten hingegen auf Hochtouren an ihren Aufgaben weiter. Gegen Ende wurden die Ergebnisse in den einzelnen Workshops untereinander vorgestellt – beispielsweise in Form eines kleinen Wettbewerbs.

Um 14:30 Uhr endete die Arbeit in den Workshops. Sich von den Aufgaben zu trennen, war für viele Schüler sichtlich schwer, da sie voller Ideen und Motivation waren. Zwei Wiedergutmachungen folgten aber glücklicherweise prompt: In diesem Jahr bekamen die Schülerinnen und Schüler erstmals eine Teilnahmeurkunde, die sie beispielsweise einer Bewerbung für ein Praktikum, eine Ausbildung oder auch für ein duales Studium beilegen können. Zum Abschluss der JavaLand4Kids konnten die Schüler und Mentoren dann das Phantasialand erkunden und eine ganze Stunde lang mit einigen Attraktionen des Parks fahren. Damit wurde ein sehr schöner Abschluss geschaffen, der jedem sichtlich Spaß gemacht hat.

Das haben die Grundschüler in diesem Jahr im Detail gemacht

Für die Kinder wurden zwei Workshop-Themen gewählt, die möglichst praktisch orientiert sind und keine Vorkenntnisse erfordern. Denn viele der Grundschüler auf der diesjährigen JavaLand4Kids kamen zum ersten Mal in Kontakt mit dem Thema Programmierung. Die Informatik lässt sich Kindern am besten mit praktischen Übungen und viel Raum für die Umsetzung eigener Ideen als interessantes und vielseitiges Feld vermitteln.

Beim "Calliope mini" handelt es sich um ein deutsches I/O-Board mit vielen eingebauten Sensoren und zahlreichen Erweiterungsmöglichkeiten. Durch Elemente wie beispielsweise LED-Anzeigefeld, Mikrofon, Gyroskop, Lichtsensor, Pins sowie Steckplätze für weitere Sensoren und Aktoren ist der "Calliope mini" vielseitig einsetzbar. Zudem ist alles an dem I/O-Board programmierbar – sowohl über vereinfachte Programmierbausteine als auch direkt in der Auszeichnungssprache JavaScript. Das macht den kleinen "Calliope mini" für viele Workshops sehr interessant. Während des Workshops lernten die Kinder, wie man den "Calliope mini" beispielsweise (unüberhörbar) als Dezibel-Messer programmieren kann und die ermittelten Werte weiterverwendet.

Begrüßung der Teilnehmer (© Lisa Rosenberg)

Karussellfahren für alle Teilnehmer (© Stefan Hildebrandt)

Im Lego-WeDo-Workshop haben die Kinder einen kleinen Roboter aus Lego-Bausteinen gebaut und anschließend mit Scratch programmiert. Da der Roboter vielseitig eingesetzt werden kann, haben sich die Schüler vor dem Zusammenbauen für ein bestimmtes Ziel entschieden. Entweder sollte er als schnelles Rennfahrzeug fungieren und ein Rennen mit anderen Teilnehmern fahren, als kräftiger Lastenzug möglichst viele Gegenstände ziehen und schieben können oder als Sammelroboter verschiedene Frachtgüter so schnell wie möglich einsammeln. Der Lego WeDo kombiniert Robotik mit Programmierung und ermöglicht so einen anschaulichen Einstieg in die Informatik.

Diesjährige Themen für die älteren Jugendlichen unter der Lupe

Bei den Workshops für die älteren Schüler war es wichtig, durch herausfordernde Aufgaben ihre vorhandenen Kenntnisse in der Programmierung zu vertiefen und neue Felder der Informatik kennenzulernen.

Programmierung des LEGO-WeDo-Roboters (© Stefan Hildebrandt)

Analyse mit Neo4j (© Lisa Rosenberg)

Beim Graphen-Workshop wurde mit der Graphendatenbank Neo4j, der Abfragesprache Cypher und den im Vorfeld aus dem OpenStreetMap API in Neo4j importierten Kartendaten gearbeitet. Nach einer Einführung in die Themen Graphen und Neo4j hatten die Schüler die Aufgabe, sich mithilfe von Neo4j in der Stadt Brühl zurechtzufinden, Lokalitäten zu suchen und diese miteinander als Graphen zu verknüpfen. Hierzu mussten Abfragen in Cypher geschrieben und optimiert werden, sodass ein passender Graph ausgegeben wird, der beispielsweise einen möglichst optimalen Pfad zwischen zwei Knoten zeigt. Da viele der Schüler bereits im Schulunterricht mit MySQL gearbeitet haben, fanden sie sich schnell mit der Syntax und den Möglichkeiten von Cypher zurecht. Die Schüler haben die Datenbank dann noch um Attribute wie beispielsweise "Freunde" und "Bewertungen" erweitert sowie Überlegungen angestellt, welche sinnvollen Analysen mit den neuen Daten möglich sind.

Bei VEX IQ handelt es sich um einen modular aufbaubaren Roboter von VEX Robotics, der vielseitige Aufgaben bewältigen kann. Von der einfachen Fernsteuerung mit dem Controller bis zum steuerbaren Lastenheber, autonomen Segway-Roboter oder komplexen Rubiks-Würfel-Solver ist mit ihm alles möglich. Erste Aufgabe der Schüler im Workshop war es, den VEX IQ aufzubauen und mit Sensoren und Aktoren zu versehen. Der Roboter wurde im Anschluss durch einen vorgegebenen Parcours mit dem Controller (vorerst noch) manuell gesteuert. Mithilfe des auf der Programmiersprache C basierenden Dialektes RobotC, haben die Schüler den Roboter im Anschluss pro-

VEX-IQ-Roboter absolviert autonom den Parcours (© Stefan Hildebrandt)

grammiert. Ziel war es, dass der VEX IQ den vorgegebenen Parcours völlig autonom absolvieren konnte. Die Schüler nutzten hierzu die eingebauten Sensoren wie den Distanzmesser und den Farbsensor, um den Roboter bei den Hürden aus farbigen Würfeln korrekte Entscheidungen treffen zu lassen.

Ein ähnliches Thema verfolgten auch die Teilnehmer beim Workshop mit dem Lego Mindstorms EV3. Die Schüler bauten zu Beginn ein Kettenfahrzeug mit Lego und versuchten danach, mit dem Roboter eine auf dem Boden mit Klebestreifen gesteckte, vorgegebene Strecke abzufahren. Der Ansatz hierbei war herauszufinden, wie viele Motorumdrehungen für die Strecke benötigt werden, oder alternativ gleich einen Weg zu finden, mit Sensoren das Ende der Strecke zu detektieren. Nachdem diese Versuche erfolgreich waren, dauerte es nicht lange, bis die ersten selbsterdachten Strecken auf dem Boden zu sehen waren. Die Schüler programmierten den EV3-Roboter, sodass dieser Strecken mit beliebigen Mustern folgen kann. Nach dem Mittagessen gab es auch einen Wettbewerb: Innerhalb von 75 Minuten musste eine bestimmte Aufgabe gelöst werden, bei der die Schüler gegen die Mentoren antraten. Der Roboter sollte eine komplexe Strecke bis zum Ziel selbstständig meistern.

Organisation der Workshops

Um ein Event wie die JavaLand4Kids für die Schüler ermöglichen zu können, bedarf es viel Arbeit und Vorbereitung der beteiligten Mentoren und Organisatoren. Initiiert hat das JavaLand4Kids Uwe Sauerbrei, der auch Gründer der Kids4IT in Hamburg ist. Er wählt die Mentoren für das JavaLand4Kids aus und organisiert die Workshop-Themen. Marcel Trebes von der DOAG Dienstleistungen GmbH fungiert als Event-Manager auf der Seite der JavaLand und organisiert beispielsweise die Räumlichkeiten, die technische Ausstattung sowie das Essen für die Kinder und Mentoren.

Die Workshop-Themen werden mehrere Monate vor der Veranstaltung ausgewählt und ausgearbeitet. Hierzu nutzen die Mentoren das Chat-Programm Slack, um die Workshops zu planen und miteinander in Kontakt zu bleiben. Aus historischen Gründen leben zwar viele der Mentoren in Hamburg und Umgebung und könnten sich auch persönlich für die Vorbereitung treffen, doch kommen auch viele der Mentoren aus anderen Städten. Aus diesem Grund ist dieser Kommunikationskanal für die Mentoren sehr wichtig. Die Themen selbst wurden zum Teil auch schon bei anderen Workshop-Formaten wie Kids4IT oder Devoxx4Kids verwendet. Diese erprobten Themen wurden um neue Aufgaben oder andere Ideen für das JavaLand4Kids erweitert. Für andere Themen wie VEX IQ und Lego Mindstorms EV3 wirken auch Mitarbeiter der jeweiligen Firmen mit, planen den Workshop zusammen mit den anderen und bringen die benötigte Ausstattung zum Veranstaltungstag mit. Die Zusammenarbeit klappt hervorragend und bietet die Möglichkeit, den Kindern immer wieder neue Themen anzubieten.

Resümee

Die JavaLand4Kids hat sich für alle Beteiligten rundum gelohnt. Die Kids konnten viel neues Wissen und Spaß mitnehmen und die Mentoren haben neben dem guten Gefühl ihres erbrachten Engagements sicherlich auch das eine oder andere Neue dazugelernt.

Für mich war es die erste JavaLand4Kids, die ich als Mentorin erlebt habe. Das Event hat mir sehr viel Spaß bereitet und meine Erwartungen mehr als übertroffen. Ich habe eine sehr erfolgreiche Zusammenarbeit mit meinen Mit-Mentoren und Organisatoren sowohl in der Vorbereitungsphase als auch am Veranstaltungstag erlebt und viele talentierte und wissbegierige Schüler kennengelernt. Besonders die Vorkenntnisse der älteren Schüler haben mich positiv überrascht: Viele von ihnen hatten bereits Themen wie Java, JavaScript, Visual Basic, SQL und MySQL in ihrem Schulunterricht behandelt und arbeiteten an den Programmieraufgaben daher mit hoher Eigenmotivation und thematischem Verständnis. Es gibt sogar Schüler, die sich vorstellen können, selbst als Mentor aktiv zu werden. Eindrücke wie diese geben mir und vermutlich auch den anderen Mentoren das sichere Gefühl, dass das erbrachte Engagement auf fruchtbaren Boden fällt. Bei so einigen Schülern hatte ich das Gefühl, dass sie auch beruflich Fuß in der Informatik fassen und die IT-Branche mit ihrem Talent schon bald aktiv bereichern werden.

LEGO Mindstorms EV3 in Aktion (© Stefan Hildebrandt)

Auch nächstes Jahr findet wieder eine JavaLand4Kids statt. Wir Mentoren freuen uns schon jetzt auf das Event und werden uns auch für das nächste Jahr wieder tolle Themen und anspruchsvolle Aufgaben für die teilnehmenden Schüler ausdenken. Wenn ihr interessiert seid, als Mentor/in mitzuwirken, dann meldet euch gerne per E-Mail an **office@javaland.eu**.

Lisa Rosenberg
lisa.rosenberg@outlook.de

Lisa Rosenberg arbeitet seit 2017 als Entwicklerin bei der in Braunschweig ansässigen msg DAVID GmbH und studiert parallel an der TU Braunschweig Informatik (M.Sc.). In ihrer freien Zeit ist sie als Speakerin aktiv und engagiert sich als Mentorin bei unterschiedlichen IT-Workshops für Kinder und Jugendliche. Sie ist auf Twitter unter @rosenbeli aktiv.

Warum sagt mir ein Tool, dass mein Quelltext schlecht ist?

Anika Zohren

Wenn es um statische Quelltext-Analyse geht, schaut man gern schon einmal, ob jemand hinter einem steht, der einem gut gemeinte Ratschläge gibt. Oft scheitern Einführungen von Tools zur statischen Quelltext-Analyse – behauptet jedenfalls Anika Zohren und plaudert ein wenig aus ihrem mittlerweile 17 Jahre alten Programmiererinnen-Nähkästchen. Beginnen wir mit einer Geschichte.

So oder so ähnlich

Mein Vortrag hat gerade begonnen. Ich stehe an meinem Rednerpult, vor mir mein aufgeklapptes Notebook. Die Folien werden auf der Leinwand hinter mir angezeigt. Plötzlich poppt ein Video-Anruf auf. Einer der Projektleiter ruft mich an. Ich drücke ihn weg, entschuldige mich beim Publikum und setze meinen Vortrag fort. Er ist der Leiter des Projekts, für das ich ab morgen die Urlaubsvertretung eines Kollegen machen soll. Üblicherweise ist dabei nicht viel zu tun. Nur, wenn es einen Notfall gibt, muss ich einspringen. Aber warum sollte er mich heute schon anrufen? Da klingelt es ein zweites Mal.

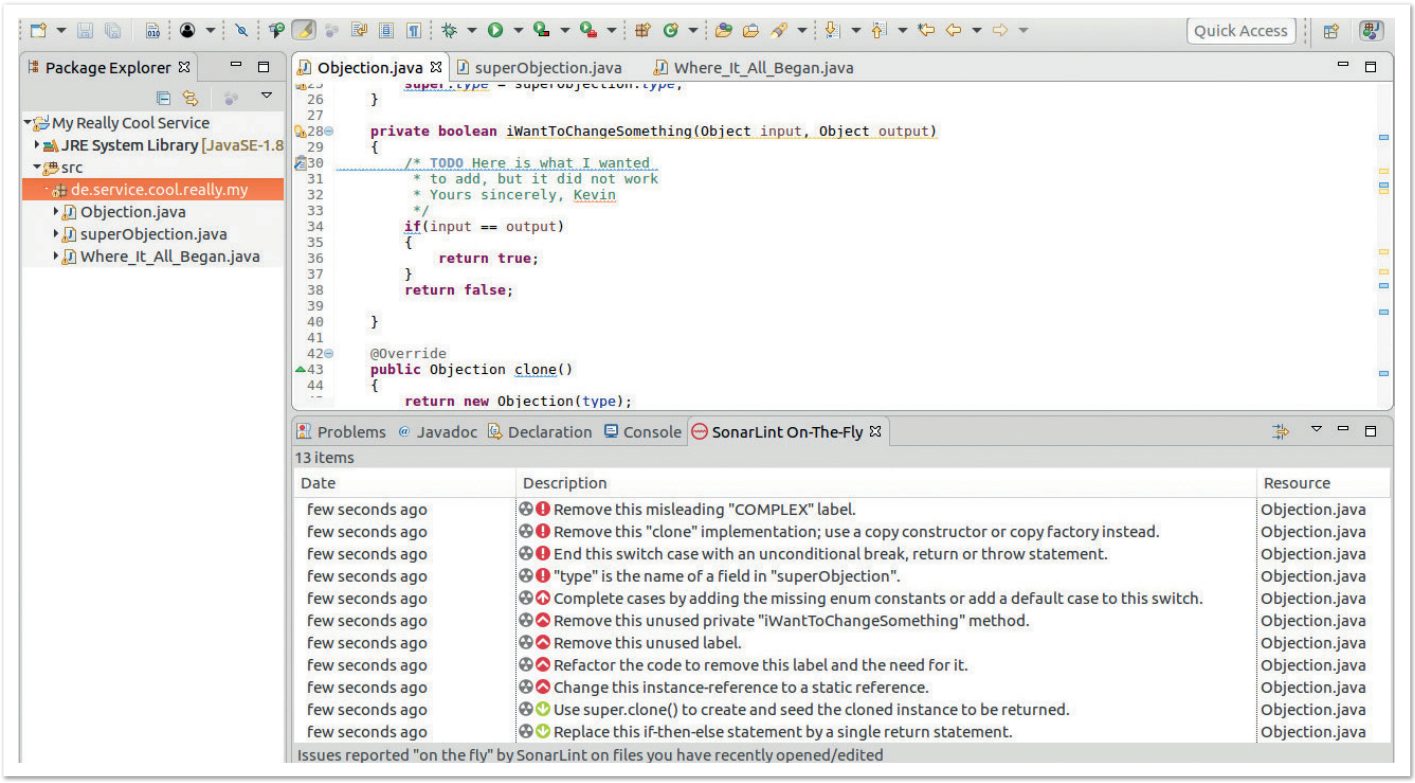


Abbildung 1: Beispiel für schlecht geschriebenen Quelltext (Quelle: Anika Zohren)

Dieses Mal gehe ich ran. Der Projektleiter klingt etwas panisch. Irgendetwas von GoLive in einer Stunde und es fehle ein Teil. Eben der Kollege, für den ich die Urlaubsvertretung mache, hat seinen Teil nicht abgeliefert – und ist nicht mehr erreichbar. Er ringt mir das Versprechen hab, dass ich schaue, was ich tun kann. Ich wechsele in unseren Firmen-Chat und schreibe in die Entwicklergruppe, was gerade los ist und ob jemand helfen kann. Unserer Lead Developer meldet sich zum Glück recht zügig.

Er berichtet, dass er den Kollegen gerade noch gesehen hätte. Er wollte anscheinend die letzten Änderungen an seinem Service in das Versionssystem einchecken. Jedoch klappte da wohl etwas nicht, da er zu fluchen und sich aufzuregen begann, dass er nun alles umbauen müsse und dann sei er einfach gegangen. Da ich den Quelltext ja schon ausgecheckt hätte, sollte es für mich ein Leichtes sein, zumal die Änderung nur eine Zeile beträfe. Ich solle das einfach „quick and dirty" erledigen.

Ich öffne also meine IDE, um die Änderung schnell durchzuführen. Beim Öffnen der Projektstruktur bietet sich mir ein Bild des Grauens: Mein Kollege hält sich anscheinend an keine der gängigen Coding Styles in der Java-Programmierung *(siehe Abbildung 1)*. Das Publikum ist erstaunlich entspannt und eher amüsiert.

Ich finde die Stelle zügig, obwohl mich sehr viel an diesem Quelltext irritiert. Zum Beispiel blau unterkringelte Passagen. Das ist neu. Vielleicht macht das eines der neuen Plug-ins, die installiert wurden. Wir als Entwickler bekommen regelmäßig aktualisierte IDEs mit den benötigten Plug-ins und Styling-Sheets zur Verfügung gestellt. Aber nicht jeder Entwickler benutzt sie.

Die Änderung ist schnell gemacht. Beim Einchecken erscheint jedoch in fröhlichem Rot „checkin rejected". Als Erklärung heißt es „Broken Quality Gate, too many code quality issues" – was soll das denn jetzt wieder?

Vielleicht sollte ich mir die blau markierten Stellen doch einmal anschauen. Ich suche und finde in meinen Sichten etwas, das „SonarLint On-Thy-Fly" heißt, und öffne es. Eine sehr lange Liste mit sehr vielen Warnungen erscheint. Ich klicke einige an. Es gibt sogar eine Ansicht mit Beispielen und Erklärungen für jede Regel, gegen die verstoßen wurde. Das ist nun keine Mal-eben-Aufgabe mehr, denke ich mir. Das war es wohl, was den Kollegen zum Fluchen brachte. Jetzt soll ich alles umbauen? Auf keinen Fall.

Ich bitte meinen Lead Developer um Hilfe. Er ist heute eigentlich nicht so gut auf mich zu sprechen, weil wir heute Morgen eine ausführliche Diskussion zum Thema Klammerstil *[2]* hatten – mal wieder. Er findet es sehr viel sinnvoller und lesbarer, wenn die geschweiften Klammern unter den Aufruf gesetzt werden und nicht daneben. Er ist allerdings der einzige Java-Entwickler bei uns, der das findet. Ich habe ihm vorgeworfen, dass er nicht immer an den Dingen festhalten kann, die für ihn vor 30 Jahren, als er noch C-Entwickler war, galten.

Ich schildere ihm die Situation und gebe zu bedenken, dass ich ja immer noch mitten in einem Vortrag bin, das Projekt in weniger als einer Stunde live gehen will und ich es nicht schaffe, das bis dahin zu bereinigen. Da fällt ihm doch glatt in diesem Moment ein, dass die Geschäftsführung eine E-Mail an alle Lead Developer geschickt hatte und über das neu eingesetzte Tool zur Verbesserung der Code-Qualität informierte. Dieses Tool lief wohl eine Zeitlang über den Nightly Build und ermittelte ziemlich viele Fehler. Es stand angeblich nicht in der E-Mail, dass sie das direkt scharf stellen und den Check-in blocken würden, aber das hatte sich bestimmt die DevOps*[3]*-Abteilung ausgedacht, meinte er. Einfach nur, weil es

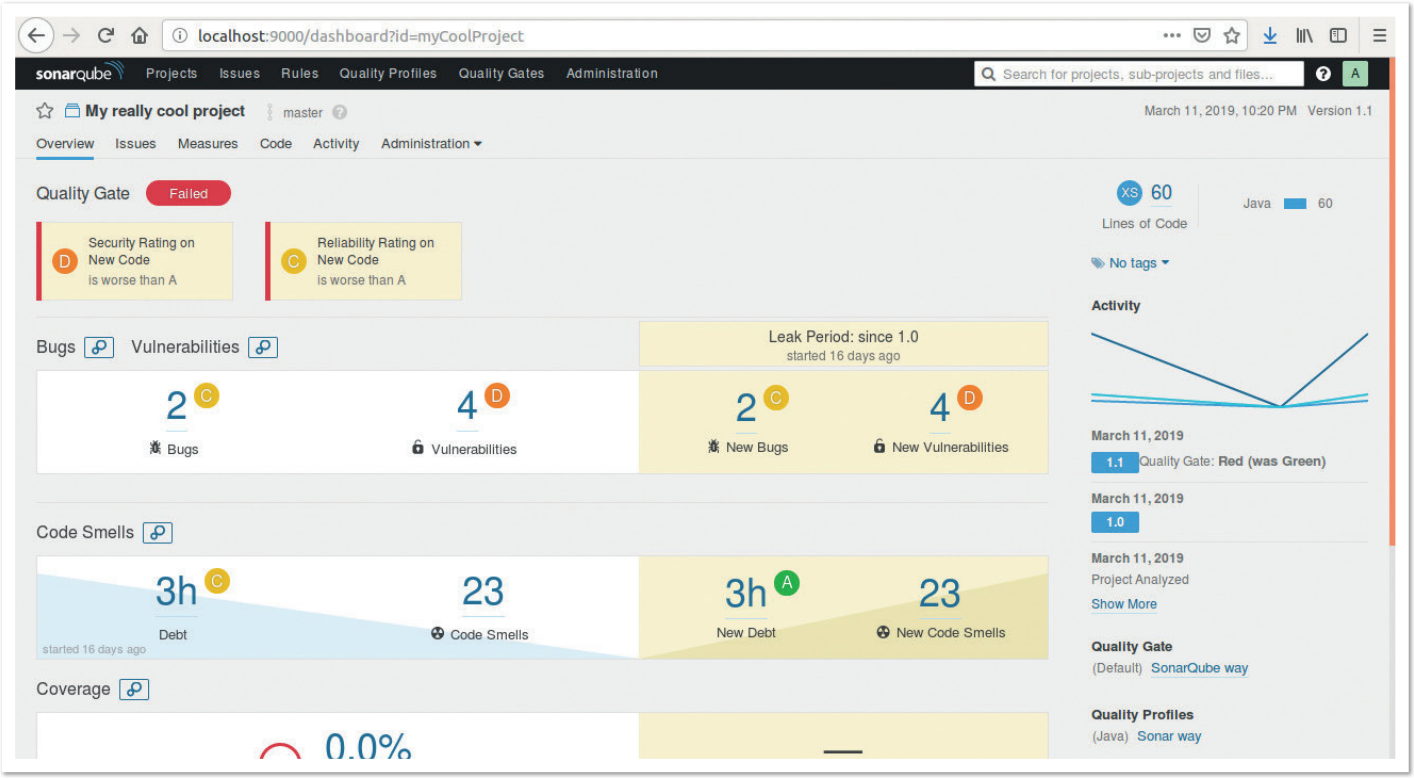


Abbildung 2: Projektansicht von SonarQube (Quelle: Anika Zohren)

geht. Die wollen immer zeigen, was sie Großartiges in fünf Minuten einrichten können.

Dann sehe ich, dass mein Lead Developer einen Lösungsvorschlag für mein Problem geschickt hat: SuppressWarnings – dass ich selbst nicht darauf gekommen bin! Ich probiere es sofort aus und siehe da, es funktioniert. Alle Fehler verschwinden. Jedenfalls in der Ansicht der IDE. Ich wechsele gleich zu meiner Versionsverwaltung und checke meine Änderung ein. Es funktioniert tatsächlich.

Jetzt noch schnell den Merge Request erstellen und dann kann es weitergehen mit meinem Vortrag. Ich trage natürlich den Lead Developer als Reviewer ein, sonst stellt noch jemand Fragen, warum ich eine `SuppressWarnings("all")`-Annotation an die Klasse ge-

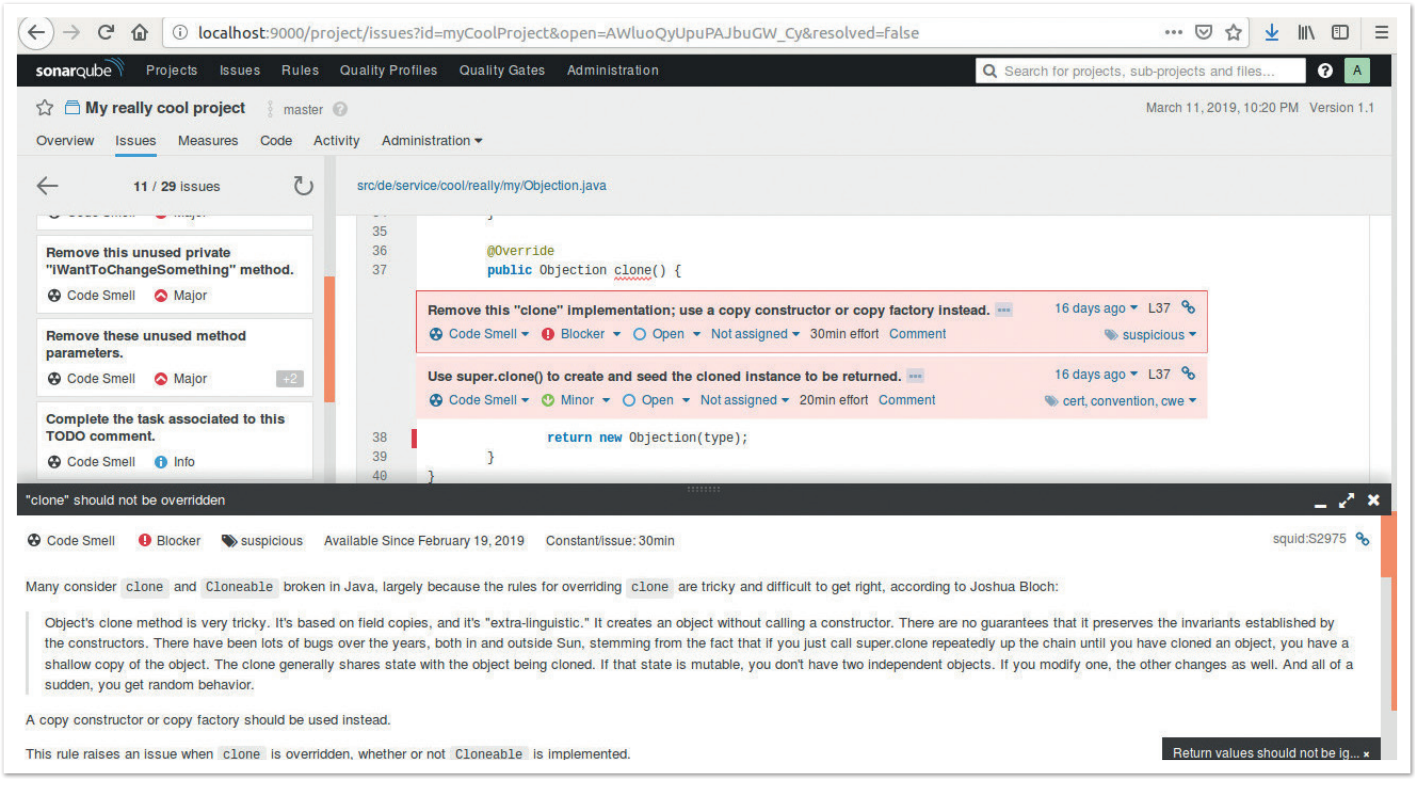


Abbildung 3: Anzeige eines Regelverstoßes inklusive Erklärung im Quelltext (Quelle: Anika Zohren)

heftet habe. Drei Sekunden später begrüßt mich ein unwillkommenes „Merge request rejected" in roter Schrift. „Short identifier are not necessary anymore, since we have the JIT compiler … don't live in the past". Das war ja klar, dass da noch eine Retourkutsche kommt. Ich wechsele in meine IDE, suche den kurzen Variablennamen und sehe ein `i1`. Gut, ich weiß nicht, was die Variable macht, aber ich soll ja nur einen längeren Namen vergeben – keinen sinnvolleren. Also nenne ich sie „integer1" und zur Sicherheit setze ich noch ein `SuppressWarnings("all")` an die Klasse. Ich checke die Änderung ein. Der Merge Request wird aktualisiert und drei Sekunden später habe ich auch das Approval vom Lead Developer. Das ist ja gerade nochmal gut gegangen.

Statische Quelltext-Analyse

Man könnte meinen, statische Quelltext-Analyse bedeutet: Jemand schaut dir über die Schulter und sagt dir, dass dein Quelltext schlecht ist.

In der Geschichte zu Beginn zeigen sich verschiedene Formen der statischen Quelltext-Analyse, die zu verschiedenen Zeitpunkten im Software Life Cycle ausgeführt werden. Der Lead Developer schaut über den Quelltext und gibt Anmerkungen. Ein Tool analysiert beim Einchecken den geänderten Quelltext. Das gleiche Tool prüfte allerdings auch schon nächtlich den gesamten Quelltext-Bestand. Die Urlaubsvertretung schaut sich den Quelltext vorher einmal an (oder auch nicht). All das ist statische Quelltext-Analyse.

Ob Mensch oder Maschine die Analyse durchführt – sie ist immer eine Betrachtung nach bestimmten Kriterien. Seien es hinterlegte Regelsätze oder die Erfahrung eines Leads oder Senior Developers. Das Ergebnis sind markierte Stellen im Quelltext. Bestenfalls mit ausführlichen Erklärungen, mündlich oder schriftlich. Jedoch wird der Quelltext nicht geändert. Als Entwickler muss man die Änderungen selbst vornehmen und das ist auch gut so. Denn sonst lernt man nichts und macht den gleichen Fehler beim nächsten Mal wieder.

SonarQube

In der Anfangsgeschichte wird ein Tool für die Analyse eingesetzt. Als ich das letzte Mal bei Wikipedia *[4]* nachschaute, wurden mir 42 Multi-Language-Tools für statische Quelltext-Analyse aufgelistet (z.B. SonarQube *[5]*) und für Java noch einmal 21 weitere (z.B. PMD *[6]*, FindBugs *[7]*, SpotBugs *[8]*, CheckStyle *[9]* usw.).

Ich möchte hier als Beispiel das Tool SonarQube vorstellen. Warum gerade SonarQube? Zum einen ist es in Java geschrieben. Außerdem ist eine stabile Community Edition *[10]* mit LTS verfügbar. Zum anderen besteht die Möglichkeit, es nicht nur durch Plug-ins zu erweitern, sondern auch selbst Plug-ins zu schreiben *(siehe Abbildung 2)*.

Die technische Hürde, sich SonarQube herunterzuladen und einfach einmal auszuprobieren, ist sehr gering. Es gibt vorgefertigte Docker-Images *[10]* und SonarQube selbst kommt mit einer vorkonfigurierten Installation um die Ecke („Sonar way"). Der "Sonar way" stellt ein vorkonfiguriertes Quality Gate und Quality Profile bereit.

Ein Quality Gate stellt die Grenze dar, bei deren Überschreitung Alarm geschlagen werden soll. Diese „Schmerzgrenze" lässt sich aus einer großen Auswahl von Bedingungen zusammenstellen. Zum Beispiel eine bestimmte Untergrenze an Code Coverage, eine bestimmte Obergrenze an neu auftretenden Fehlern, ein bestimmtes Security Rating bei neuem Quelltext usw.

Ein Quality Profile ist kurz gesagt eine Zusammenstellung von Regeln, die man auf sein Projekt angewandt sehen möchte. Das vorgeschlagene Quality Profile für Java enthält zum Beispiel etwa 300 Regeln *[12]*.

Mit dem IDE-Plug-in SonarLint*[13]* kann die IDE mit dem soeben heruntergeladenen SonarQube verbunden werden. Derzeit unterstützt werden eclipse, IntelliJ IDEA, Visual Studio, VS Code und Atom *(siehe Abbildung 3)*.

Meine persönliche Erfahrung

Meine erste Begegnung mit SonarQube kann man ungefähr so zusammenfassen: Etwas schaut mir über die Schulter und sagt mir, dass mein Quelltext schlecht ist.

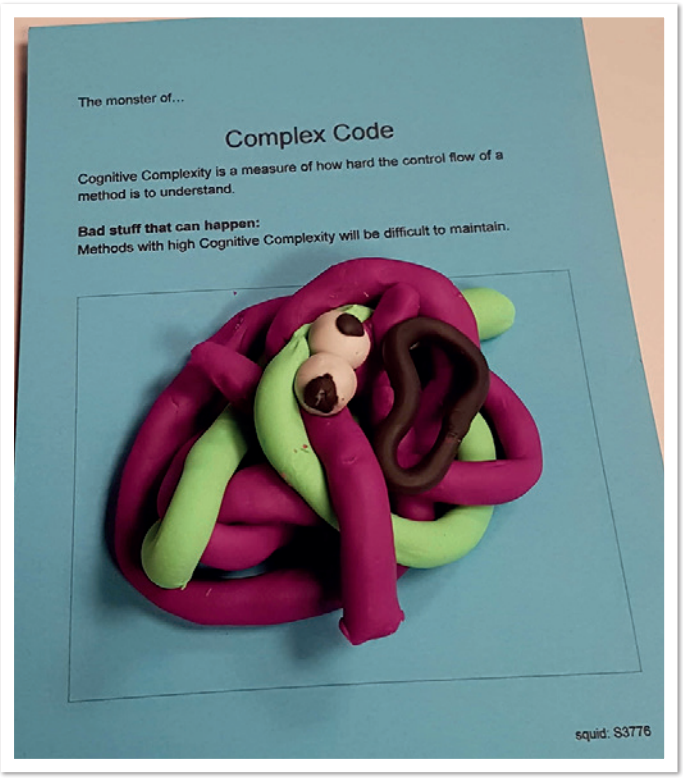
Ich habe damals als Java-Entwicklerin für ein Softwarehaus gearbeitet. Da ich jedoch Erfahrung mit Front-end-Technologien hatte, fand ich mich eines Tages in dem Team wieder, das eine neue Web-Plattform erstellen sollte. Drei Java-Entwickler begannen also, mit AngularJS (v1.x) *[14]* einen ersten Entwurf für ein Front-end zusammen-zu-schustern. Es war nicht schön, was wir da schufen. Und unser Lead Developer suchte nach einer Lösung, um die Quelltext-Qualität zu verbessern. Er fand sie in Gestalt von SonarQube und bat uns, das SonarLint-Plug-in in unsere IDE einzufügen und mit dem Server zu verbinden.

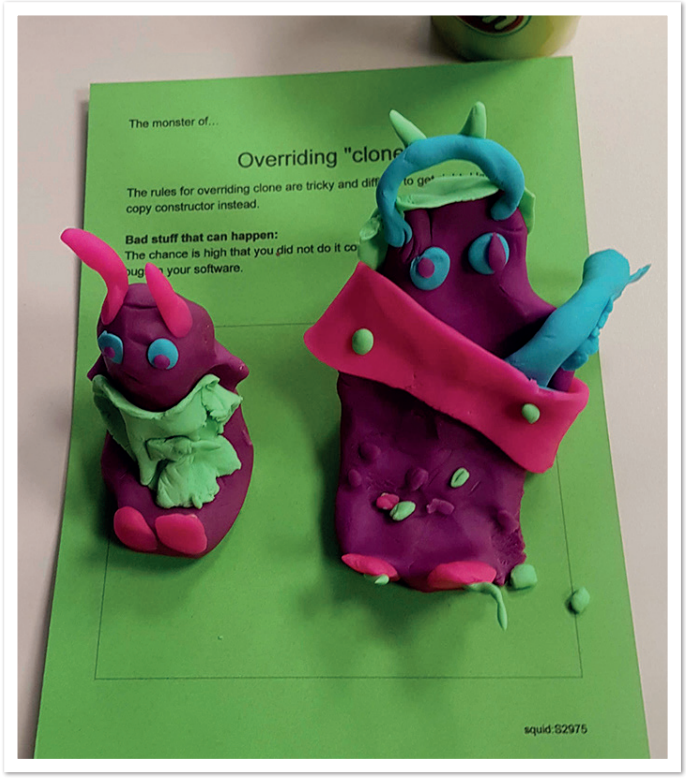
Ich gab dem Tool eine Chance, ärgerte mich jedoch schnell. Vor allem quälte mich die Regel „Functions should not have too many parameters" *[15]*. Ich begann also, Objekte zu bauen und Arrays zu übergeben. Dadurch wurden allerdings meine Funktionen größer und es gesellte sich „Functions should not have too many lines" *[16]* als neuer Regelverstoß hinzu. Also versuchte ich, Zeilen einzusparen. Das endete jedoch in erneuten Regelverstößen zum Thema „Cognitive Complexity of functions should not be too high".

Ich wollte nicht aufgeben, also las ich mir alle Regeln noch einmal genau durch und begann, meine Funktionen auseinanderzunehmen und meine ganze Logik neu aufzubauen. Die Liste der Regelverstöße wurde daraufhin immer kürzer und es kamen keine neuen hinzu.

Als ich dann wieder Anforderungen im Hauptprodukt umsetzen sollte, war ich sehr glücklich. Endlich wieder Java. Und kein Tool, dass mir sagt, dass mein Quelltext schlecht ist. Doch schon nach einigen Wochen kamen auch wieder neue Anforderungen für die Web-Plattform. Ich war sehr unglücklich, doch das änderte sich für mich mit dem folgenden Schlüsselerlebnis.

Mit Unmut rief ich also wieder den Quelltext der Web-Plattform auf. Doch zu meiner Überraschung konnte ich meinen Quelltext ungewöhnlich schnell und ohne lange überlegen zu müssen lesen und die neuen Anforderungen umsetzen. Es beschlich mich der Gedanke, dass vielleicht doch nicht so falsch war, was das Tool gesagt hatte. Ich hatte es mithilfe des Tools geschafft, qualitativ hochwertigen, einfach lesbaren Quelltext zu schreiben. Und das mit einem Framework, dass ich nicht besonders gut kannte, in einer Skript-Sprache, die ich jahrelang nicht mehr angewandt hatte.





Abbildungen 4 und 5: Monster aus Knete für Complex Code und Overriding „clone" (Quelle: Anika Zohren)

Das war der Punkt, an dem ich begann, bei jedem Projekt lokal ein SonarQube zu installieren und mir beim Programmieren erklären zu lassen, was ich besser machen konnte. Ich nahm es plötzlich wahr als: Jemand schaut mir über die Schulter und unterstützt mich darin, besser zu programmieren.

Neulich habe ich von einem der ehemaligen Kollegen aus dem Web-Plattform-Team gehört, dass sich die Verwendung von SonarQube bis heute nicht für die Web-Plattform etabliert hat. Und es ist nicht das einzige Mal, dass ich gesehen habe, dass sich Firmen schwer damit tun, automatisierte statische Quelltext-Analyse einzuführen.

Da war zum Beispiel das Projekt, in dem ich – mittlerweile begeistert – von den Vorteilen der automatisierten Quelltext-Analyse erzählte. Ich bekam ein müdes Lächeln zurück mit dem Kommentar, das hätten sie schon alles installiert. Jedoch würde es sich nicht durchsetzen. Sie hätten es schon drei Mal ausgerollt. Aber es würde immer im Sande verlaufen, weil es niemand benutzt.

Was läuft falsch?

Die Geschichte zu Beginn zeigt ziemlich deutlich, was alles falsch laufen kann bei der Einführung eines Tools zur statischen Quelltext-Analyse und Verbesserung der Quelltext-Qualität.

Da haben wir zum Beispiel die fiktive Geschäftsführung, die mit der DevOps-Abteilung ein Tool installiert, das den Quelltext analysiert. Da es Tausende von möglichen Regeln gibt und jede Entwicklungsabteilung in jeder Firma andere Rahmenbedingungen hat, unter denen sie entwickelt, können die gewählten Regeln ohne Rücksprache mit den Entwicklern gar nicht passen. Auch eine Blockade beim Einchecken von Änderungen sehe ich skeptisch. Natürlich ist das technisch möglich. Aber nur, weil etwas technisch möglich ist, muss es noch lange nicht sinnvoll sein.

Das geflügelte Wort „quick & dirty" taucht in der Geschichte natürlich auch auf. Wer nicht erkannt hat, dass mit „quick & dirty" langfristig eine Quelltext-Basis entsteht, die weder wartbar noch erweiterbar ist, der beseitigt dieses Problem auch nicht damit, „mal eben zwischendurch" ein paar Tools zu installieren, die die Quelltext-Qualität verbessern sollen. Wenn kein Budget für Refactoring bereitgestellt wird, nützt das beste Code Quality Tool nichts. Wenn keine Zeit für das Erarbeiten eines passenden Regelsatzes und das Verständnis für die Regeln bei den Entwicklern investiert wird, bringt auch das Refactoring nichts, da immer wieder neue Regelverstöße auftreten.

Natürlich gibt es noch einige weitere Probleme, die das Einführen eines Tools zur statischen Quelltextanalyse und Verbesserung der Quelltext-Qualität scheitern lassen. Doch die grundsätzliche Einstellung, ob eine Firma bereit ist, Zeit und Geld in das Thema Code Quality zu investieren, und die langfristigen Vorteile erkennt, scheint mir ein entscheidender Faktor zu sein.

Lösungsansätze

Wie bereits erwähnt, ist es sehr wichtig, genug Zeit bereitzustellen, sich mit den Regeln und Regelsätzen auseinander-zu-setzen. Mit genug Zeit meine ich drei bis sechs Monate Vorlaufzeit. Die Auswahl der Regeln sollte von einer nicht zu großen Gruppe zusammengestellt werden. Sagen wir fünf Personen. Und dabei sollte auch gelten: Weniger ist mehr. Wenn sich diese Gruppe auf nur zehn Regeln verständigen kann, dann ist es so. Besser zehn Regeln anwenden, als keine.

Noch bevor irgendetwas installiert wird, sollten alle Entwickler alle ausgewählten Regeln kennen und verstehen. Viele Entwicklerteams veranstalten wöchentliche Entwickler-Meetings oder Knowledge Transfers, bei denen die Regeln zum Beispiel vorgestellt werden könnten. Jedoch besteht generell das Problem, dass die Aufmerk-

samkeit bei diesen Meetings nicht immer gegeben ist. Ich erinnere mich an eines dieser Meetings, an dem ich remote teilnahm. Dort wurde mir 45 Minuten lang erklärt, dass ich nun sieben verschiedene Code-Quality-Plug-ins in meine IDE installieren soll, ohne auch nur mit einem Wort auf die Regelsätze einzugehen. Ich war sehr produktiv in diesem Meeting. Am Ende hatte ich alle meine E-Mails gelesen, meinen Desktop aufgeräumt und die Zeiterfassung der letzten Woche fertiggestellt.

Um dies zu vermeiden, ist ein Schlagwort dazu sicherlich „Gamification". Mein liebstes Beispiel ist das der Klavier-Treppenstufen in einer U-Bahn Station in Stockholm *[17]*. Ziel war es, die Menschen dazu zu bewegen, nicht die Rolltreppe zu nehmen, sondern die Treppe. Also wurden die Treppenstufen nicht nur mit Folie beklebt, damit sie wie Klaviertasten aussahen. Sie wurden auch mit Sensoren und Lautsprechern ausgestattet, die beim Auftreten den entsprechenden Ton ausgaben. Laut eigener Aussage nahmen durch diese Änderung 66% Personen die Treppe statt der Rolltreppe.

Doch wie könnte dies bei der Erarbeitung von Regeln für die Verbesserung der Quelltext-Qualität aussehen? Das ist wieder sehr von der Firma und den Entwicklern abhängig und worauf sie sich einlassen. Da meine Mutter Kindergärtnerin ist, habe ich mein Wissen auch über die Kindergartenzeit hinaus ersungen, ertanzt, erknetet, ermalt und erbastelt und bin bis heute immer noch großer Fan dieser Methoden. Wenn mir technische Zusammenhänge nicht klar sind, fange ich an, sie aufzumalen. Mit vielen bunten Farben. Daher habe ich bei einem Meetup, bei dem es um Code Quality ging, die Regeln als „Monster" formuliert und die Teilnehmer wurden gebeten, diese mit Knete zu visualisieren. Am Ende stellte jeder „sein" Monster vor *(siehe Abbildung 4 und 5)*.

Hat man es jedoch mit wenig experimentierfreudigen Entwicklern zu tun, gibt es auch Lösungen wie zum Beispiel SonarQuest *[18]*. Dies macht aus dem Beheben von Regelbrüchen ein digitales Helden-Adventure.

Ist der Regelsatz erstellt und sind die Regeln von allen Entwicklern verstanden, sollten bei einer schon vorhandenen Quelltext-Basis jedoch erst neue Fehler begrenzt werden, bevor das Refactoring beginnt. Ich erinnere mich an ein Projekt, in dem wir es andersherum gemacht haben. Das führte dazu, dass während die alten Fehler behoben wurden, permanent neue hinzu kamen. Einer der Entwickler schaffte es sogar, kontinuierlich durch die Behebung von Regelverstößen neue Regelverstöße zu verursachen.

Doch sehr wichtig ist mir zu betonen, dass Regelverstöße, die von einem Tool zur statischen Quelltext-Analyse gemeldet werden, unter keinen Umständen zur Bestrafung oder Demütigung verwendet werden sollten. Möchte man die Statistiken aus dem Tool verwenden, so würde ich eine Bestenliste empfehlen mit den Top-5-Entwicklern, -Projekten oder -Services, die die wenigsten Regelverstöße verursachen. Oder eine besondere Aufmerksamkeit für Entwickler, die beim Refactoring einer vorhandenen Quelltext-Basis fleißig aufgeräumt haben. Oder man hängt sich ein Schild in den Eingang zur Entwicklungsabteilung, „Kein Quality-Gate-Bruch seit x Tagen".

Doch all das funktioniert nur, wenn auch alle mitmachen und die Quelltext-Analyse nicht empfinden als „Jemand schaut mir über die Schulter und sagt mir, dass mein Quelltext schlecht ist". Sondern als „Jemand schaut mir über die Schulter und hilft mir, qualitativ hochwertigen Quelltext zu schreiben". Es ist zum einen die Einstellung des Entwicklers, die wichtig ist für diese Einsicht. Zum anderen aber auch die Art und Weise, wie der Umgang der Kollegen und Vorgesetzten mit diesem Thema aussieht.

Quellen

[1] *https://de.wikipedia.org/wiki/Gespenster_Geschichten*
[2] *https://de.wikipedia.org/wiki/Einr%C3%BCckungsstil*
[3] *https://de.wikipedia.org/wiki/DevOps*
[4] *https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis*
[5] *https://www.sonarqube.org/*
[6] *https://pmd.github.io/*
[7] *http://findbugs.sourceforge.net/*
[8] *https://spotbugs.github.io/*
[9] *http://checkstyle.sourceforge.net/*
[10] *https://www.sonarsource.com/plans-and-pricing/community/*
[11] *https://hub.docker.com/_/sonarqube*
[12] *https://rules.sonarsource.com/java*
[13] *https://www.sonarlint.org/*
[14] *https://angularjs.org/*
[15] *https://rules.sonarsource.com/javascript/RSPEC-107*
[16] *https://rules.sonarsource.com/javascript/RSPEC-138*
[17] *https://www.youtube.com/watch?v=ivg56TX9kWI*
[18] *https://www.viadee.de/sonarquest*

Anika Zohren
anika.zohren@gmail.com

Hinter den Kulissen von SCS-Architekturen

Mirko Sertic, Thalia Bücher

Dieser Artikel schaut hinter die Kulissen einer großen SCS-Systemlandschaft. Am Beispiel eines bestehenden eCommerce-Systems wird gezeigt, wie die monolithische Struktur in eine auf dem Self-Contained-Systems-Ansatz basierende Systemlandschaft überführt wurde. Besonderes Augenmerk wird dabei auf die fiesen kleinen Details gelegt. Details, die im SCS-Manifest nicht erläutert werden, sich jedoch als Konsequenz dieser Architektur manifestieren. Die fiesen Details eben, die ein Migrationsprojekt durchaus erfolgreich verhindern können.

Worum geht es hier also genau? Es geht um ein großes Unternehmen aus Deutschland, das durch Omni-Channel-Buchhandel Bekanntheit erlangt hat. Dieses Unternehmen wird im Jahr 2019 100 Jahre alt. Nicht ganz so alt ist die Code-Basis des eCommerce-Systems. Dieses System hatte eine monolithische Struktur. Gerade diese Struktur machte es immer schwerer, mehr Features in kürzerer Zeit für die Kunden zu implementieren. Dieser Artikel beschäftigt sich mit der Migration dieser Systemlandschaft in eine neue Struktur, die auf dem Self-Contained-Systems-Ansatz basiert. In Form einzelner Lektionen möchte ich auf Details eingehen, die für einen erfolgreichen Einsatz des SCS-Ansatzes berücksichtigt werden müssen. Ich arbeite bei der Thalia Bücher GmbH. Bevor ich jedoch in der Verlegenheit komme, Interna auszuplaudern und meinen Job zu verlieren, habe ich deshalb einen Disclaimer. Bei dem hier gezeigten Migrationsprojekt handelt es sich natürlich nicht um die eCommerce-Landschaft der Thalia Bücher GmbH. Es geht hier um unseren schärfsten Konkurrenten, die „Welt, bleib wach!" Online Bücher GmbH *(siehe Abbildung 1)*!

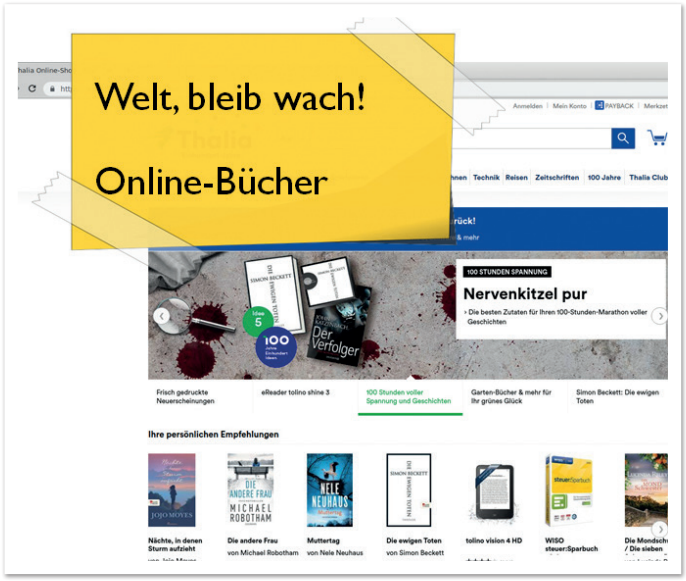


Abbildung 1: Internetauftritt „Welt, bleib wach!" (Quelle: Mirko Sertic)

Lektion #1: Die Organisation

Die Organisation ist das erste, fiese Detail. Der Self-Contained-Systems-Ansatz soll Organisationen skalieren. Diese Skalierung hat im Wesentlichen das Ziel, mehr Funktionalität in weniger Zeit umzusetzen und insgesamt (IT-)Projekte erfolgreicher und wirtschaftlicher zu machen. Im ersten Schritt muss also die Organisation angepasst werden. Ohne diesen Schritt wird der SCS-Ansatz nicht den gewünschten Effekt haben. Bei der Skalierung müssen zwei Fragen gestellt werden: „Does it scale up?" und „Does it scale down?" Hinter diesen Fragen verbirgt sich die Wahrheit, dass nicht jede Organisation die gleichen Anforderungen hat wie beispielsweise Netflix, Spotify oder auch Google. Die „Welt, bleib wach!" GmbH hat schon sehr große Lastanforderungen an den Online-Auftritt, ist aber eben doch deutlich kleiner als Netflix. Diese Organisation muss also nicht die gleichen technischen und fachlichen Probleme lösen wie Netflix. Die Kunst ist also, eine Organisation zu finden, die konkrete Probleme löst. Als Ansatz für die neue Organisation wurde ein Schnitt auf Basis von Produkten gewählt. Mit Produkt ist in diesem Sinne eine Sammlung von Funktionalität aus Sicht des Kunden gemeint. Für diese Produkte wurden Entwicklungsteams definiert, die vom Anfang bis zum Ende für den kompletten Produktlebenszyklus verantwortlich sind. Wichtig bei diesem Schnitt war, keine Überlappungen zwischen den Produkten zu haben. Es geht also darum, klare Schnittstellen zwischen den Produkten und somit auch zwischen den Entwicklungsteams zu schaffen. Beispielhaft für die „Welt, bleib wach!" GmbH sieht der Produktschnitt dann so aus, wie in *Abbildung 2* zu sehen.

Der Screenshot aus *Abbildung 2* zeigt die Produkte aus Sicht eines Besuchers. Wir sehen das Suchformular, das zu einem anderen Produkt gehört als zum Beispiel das Anmelde-Formular, die redaktionellen Inhalte oder auch der Empfehlungsbereich. Die klare Trennung zwischen den Produkten sorgt dafür, dass etwa neue Features

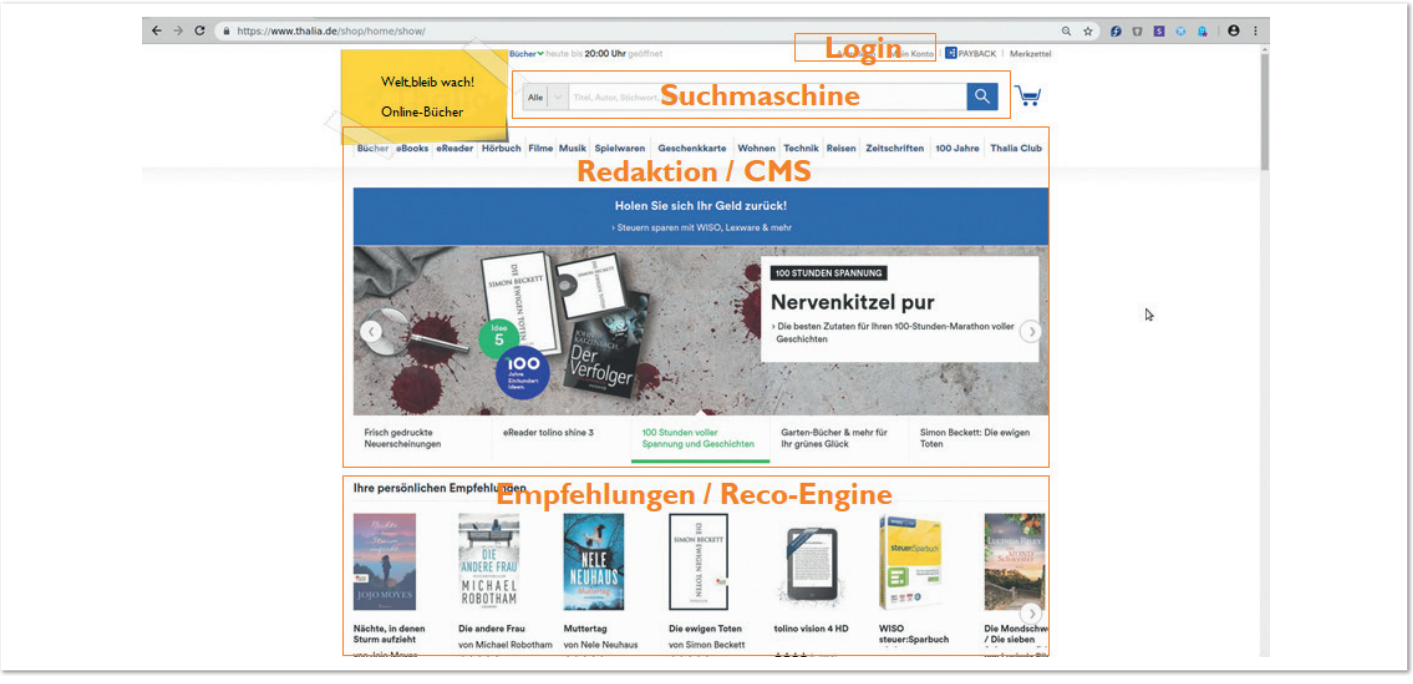


Abbildung 2: Produktschnitt (Quelle: Mirko Sertic)

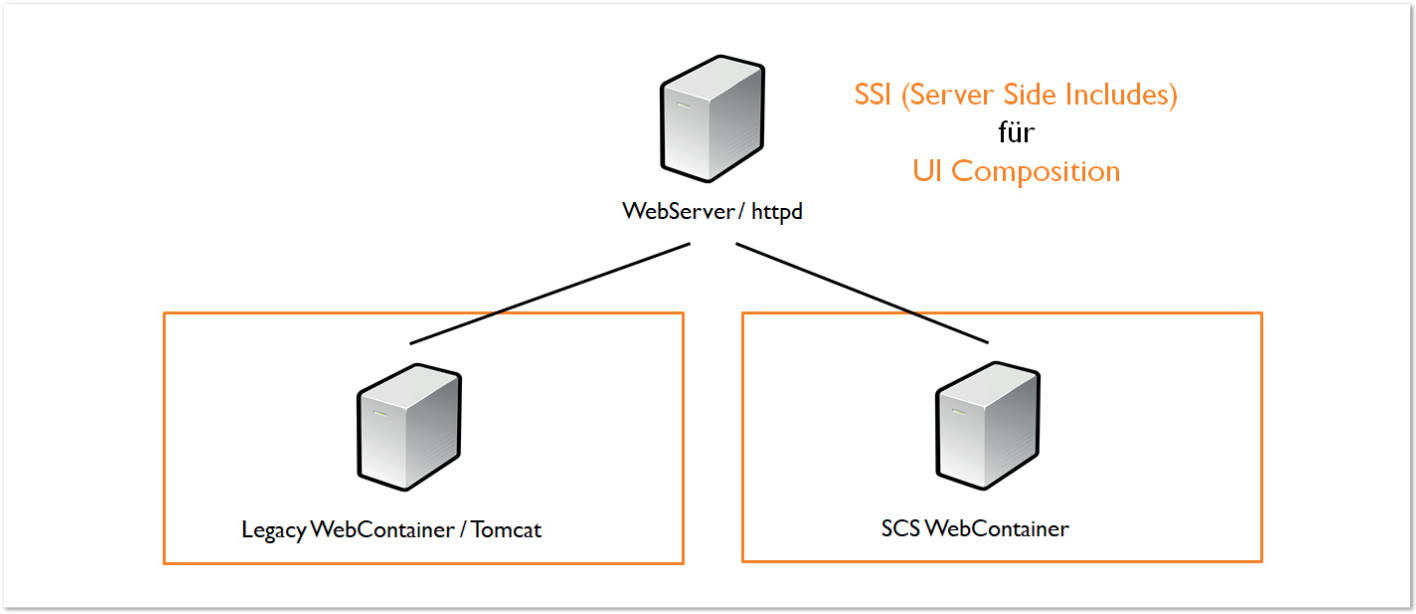


Abbildung 3: Webserver mit SSI (Quelle: Mirko Sertic)

für redaktionelle Inhalte nicht das Login beeinträchtigen können. Es gibt einen klaren Verantwortlichen für die Umsetzung dieses neuen Features pro Produkt. Produkte können unabhängig voneinander entwickelt werden. Dies ist der größte Vorteil dieser Organisationsform. Ohne diesen Schritt wird aus der Migration nur ein technisches Reengineering.

Lektion #2: Der Migrationsweg

Eine wichtige Lektion ist der Migrationsweg auf die neue Systemlandschaft. In der Theorie werden pro Produkt ein eigenes System gebaut und die Inhalte der Systeme beziehungsweise die Funktionalitäten in ein gemeinsames Frontend integriert. Aber wie soll das nun im Detail funktionieren? Ein genauer Blick auf die Inhalte kann helfen, das Problem zu lösen. Der Webshop der „Welt, bleib wach!" GmbH besteht bei genauerer Betrachtung aus zwei unterschiedlichen Inhaltstypen. Es gibt Inhalte wie das Suchfeld, die Anzeige des Anmeldestatus oder auch Slider mit Produktempfehlungen. Diese Inhalte sind in sich abgeschlossen, tauchen allerdings nie allein auf. Sie werden in einen größeren Kontext eingebunden. Dieser größere Kontext ist der zweite Inhaltstyp. Es sind eigenständige Seiten wie z.B. die Startseite, ein Suchergebnis oder auch der Inhalt des Warenkorbs beim Checkout. Für diese Seiten gibt es ein System, das führend für den Inhalt ist. Zusätzlich können jedoch auch noch andere Inhalte, wie eben der Anmeldestatus oder auch Produktempfehlungen aus anderen Systemen, eingebunden werden.

Mit welcher Technologie kann nun diese Frontend-Integration umgesetzt werden? Ein Blick auf die aktuelle Systemlandschaft zeigt, dass bereits flächendeckend HTTPD als Webserver eingesetzt wird. Dieser Webserver bringt eine interessante Technologie für die UI-Komposition mit: „Server Side Includes", kurz SSI *(siehe Abbildung 3)*.

Mit SSI-Direktiven kann der Webserver angewiesen werden, Inhalte aus anderen Systemen in ein Dokument zu integrieren. Im ersten Schritt können nun Inhalte vom ersten Typ, zum Beispiel die Slider, von einem neuen System/Produkt ausgeliefert werden. Der bisherige Monolith liefert die Slider nicht mehr aus, sondern nur noch die SSI-Direktiven, die dann vom Webserver aufgelöst werden. Durch diese Migration wird die Funktionalität in das neu zuständige System verschoben, der alte Monolith wird immer weiter ausgehöhlt und entschlackt. Bereits jetzt können die ersten Früchte der Migration geerntet werden; Slider können unabhängig als Produkt weiterentwickelt werden.

Im nächsten Schritt werden komplette Seiten in die neuen Systeme verschoben, die Inhalte vom Typ 2 werden migriert. Auf Webserver-Ebene kann dann via URL-Rewriting entschieden werden, ob eine einzelne Seite vom Monolithen oder von einem anderen SCS ausgeliefert wird. Grundvoraussetzung dafür ist natürlich, dass jede Seite eine eindeutige URL hat.

Im Normalfall bewegt sich ein Kunde in einer Session durch den Webshop. Mit dieser Session sind Zustandsinformationen verknüpft, wie etwa der Anmeldestatus oder der Inhalt des Warenkorbs. In der monolithischen Struktur gab es nur eine Session, die eventuell via Sticky-Session noch an eine bestimmte Tomcat-Instanz gebunden war. In der neuen Welt gibt es keine zentrale Session mehr, und es gibt auch keine zentrale Datenspeicherung für den Inhalt der Session. Der Anmeldestatus wird vom Login-SCS gespeichert, der Warenkorb vom Warenkorb-SCS usw. Jedes beteiligte System braucht nur noch die ID der Session zu kennen. Die Frage ist nun, wer diese ID vergibt. Die Session-ID wird als Cookie vom vorgelagerten Webserver vergeben. Die beteiligten Systeme inklusive Monolithen verwenden nur diese ID. Das dafür benötigte Federated Session Management kann sehr einfach durch einen Java-Servlet-Filter oder auch durch Frameworks wie Spring Session umgesetzt werden. Für die Migration haben wir nun eine grobe Methode – wie sehen nun die Details der Inhaltstypen aus?

Lektion #3: Frontend-Architektur

Frontends sind aus Sicht eines Backend-Entwicklers der absolute Horror. Es gibt laufend neue Frameworks, tolle, neue Build-Systeme und überhaupt immer neue Erfindungen, die alles versprechen, jedoch leider in vielen Fällen nichts halten. Hier sollte das Mantra „Resist the Hype!" gelten. Im Zweifel zählen die Produktivität eines Teams und die Time-to-Market eines Produktes mehr als die

Evaluation eines neuen Frameworks oder einer Methode. Dies soll natürlich nicht bedeuten, dass Stagnation in der technischen Basisstruktur Einzug hält. Aber gerade in einem Umfeld des stetigen Wandels ist es sinnvoll, Trends sehr genau zu beobachten, bevor auf einen fahrenden Zug aufgesprungen wird. Bei der Frontend-Architektur stellt sich zuerst die Frage, wie viele Frontends es denn überhaupt gibt.

Bei dem Migrationsprojekt der „Welt, bleib wach!" GmbH zeigte sich, dass es mehrere Frontends *(siehe Abbildung 4)* für ein SCS gibt. Redaktionelle Inhalte gibt es zum Beispiel im Online-Shop, auf den eReadern und auch in einer Android- und iOS-App. Aus historischen Gründen basiert der Online-Shop auf dem Foundation Framework, die Darstellung auf dem E-Reader basiert jedoch auf Bootstrap. Die Darstellung in den Apps ist nativ gelöst. Das SCS-Manifest schreibt vor, dass ein SCS eine autonome Webapplikation ist und die komplette Logik zur Darstellung der Inhalte mitbringt. Damit nun die beschriebenen Geräte unterstützt werden können, müssen unterschiedliche HTML-Templates für unterschiedliche Geräte definiert werden. Bei der Entwicklung von neuen Features müssen unterschiedliche HTML-Frontend-Frameworks berücksichtigt werden. Im ersten Schritt bedeutet dies natürlich einen Mehraufwand für die Produktteams. In einem Folgeschritt können die verwendeten Frameworks weiter vereinheitlicht werden. Die Motivation hier sollte natürlich die Produktivität und Time-to-Market für neue Features sein und nicht einfach die „Coolness" eines anderen Frameworks.

Am Beispiel der Apps zeigt sich ein problematischer Punkt im SCS-Ansatz: Er funktioniert nur mit Webapplikationen. Das steht natürlich auch so im Manifest, ist allerdings fern der Realität. Native Apps gibt es und sie haben ihre Rechtfertigung. Natürlich könnte die App auch durch eine WebView/Progressive Webapplication ersetzt werden, die Diskussion über Für und Wider könnte Seiten füllen. Zusammengefasst kann ich hier nur sagen, dass native Apps in Verbindung mit der SCS-Architektur problematisch sind.

Die organisatorische Trennung der Produkte soll Schnittstellen zwischen den Entwicklungsteams reduzieren und klarstellen. Damit jedoch die User-Experience nicht leidet, wenn mehrere Systeme auf der Ebene „Benutzerschnittstelle" aggregiert werden, brauchen wir ein Design-System. Dieses soll einen Leitfaden für die Implementierung der Benutzerschnittstelle bieten und als Grundlage für die Diskussion von Entwürfen dienen. Es ist also ein Kommunikationsmedium und eine Kommunikationshilfe.

Bei der technischen Implementierung von HTML-Frontends und der Aggregation sind einige Grundlagen zu beachten. Ein CSS-Namespacing auf der Ebene „Produkt" verhindert, dass CSS-Styles sich gegenseitig beeinflussen. Das Gleiche gilt für den Einsatz eines JavaScript-Modulsystems, um Seiteneffekte zu verhindern. Assets wie etwa CSS- und JS-Dateien müssen versioniert werden. Diese Versionierung ist die Grundlage für einen sinnvollen Einsatz von Content-Delivery-Networks und Caching auf den Ebenen „CDN" und „Browser". Das Caching soll natürlich die Ladezeit aus Sicht des Kunden optimieren. Eine hohe Cache-Hit-Rate im CDN kann jedoch auch helfen, Infrastrukturkosten zu sparen, da letztendlich weniger Server im Backend für die Bewältigung der gleichen Last benötigt werden.

Lektion #4: Frontend-Performance

Der Einsatz von SSI hat keinen guten Ruf. Je nach Verwendung von Apache HTTPD oder nginx werden die SSI-Direktiven sequenziell oder parallel abgearbeitet. Durch den exzessiven Einsatz von SSI-Direktiven kann eine Seite auch verlangsamt werden. Dabei ist SSI gar nicht das Problem, sondern die Tatsache, dass jede SSI-Direktive einen synchronen HTTP-Aufruf eines anderen Systems darstellt. Gerade dieses System muss den Inhalt erstmal aufbereiten, indem zum Beispiel Datenbanken befragt und Template-Engines aufgerufen werden, um HTML oder andere Inhalte zu generieren.

Da wir ja schon einen klaren Zuständigen für dieses System definiert haben, haben wir auch einen klaren Zuständigen, wenn es um das Thema Performance geht. Wenn die Seitenauslieferungszeit an den Kunden eine eindeutige Obergrenze hat, ergibt es Sinn, im Vorfeld schon Vorgaben in Form von Performance-Anforderungen an das zuständige Team zu stellen und diese Vorgaben auch durch ein entsprechendes technisches Monitoring zu überprüfen.

Die einfachste Form der Optimierung hier ist, Inhalte erst dann wirklich aufzubereiten, wenn sie tatsächlich sichtbar und für das Zielgerät relevant sind. Das Stichwort hier ist „Progressive Enhancement". Ein zuliefernder System kann beispielsweise, statt kompletten Inhalten, einen Platzhalter mit einem JavaScript ausliefern. Dieses JavaScript würde den eigentlichen Inhalt erst kurz, bevor er sichtbar wird, nachladen.

Abbildung 4: Frontends (Quelle: Mirko Sertic)

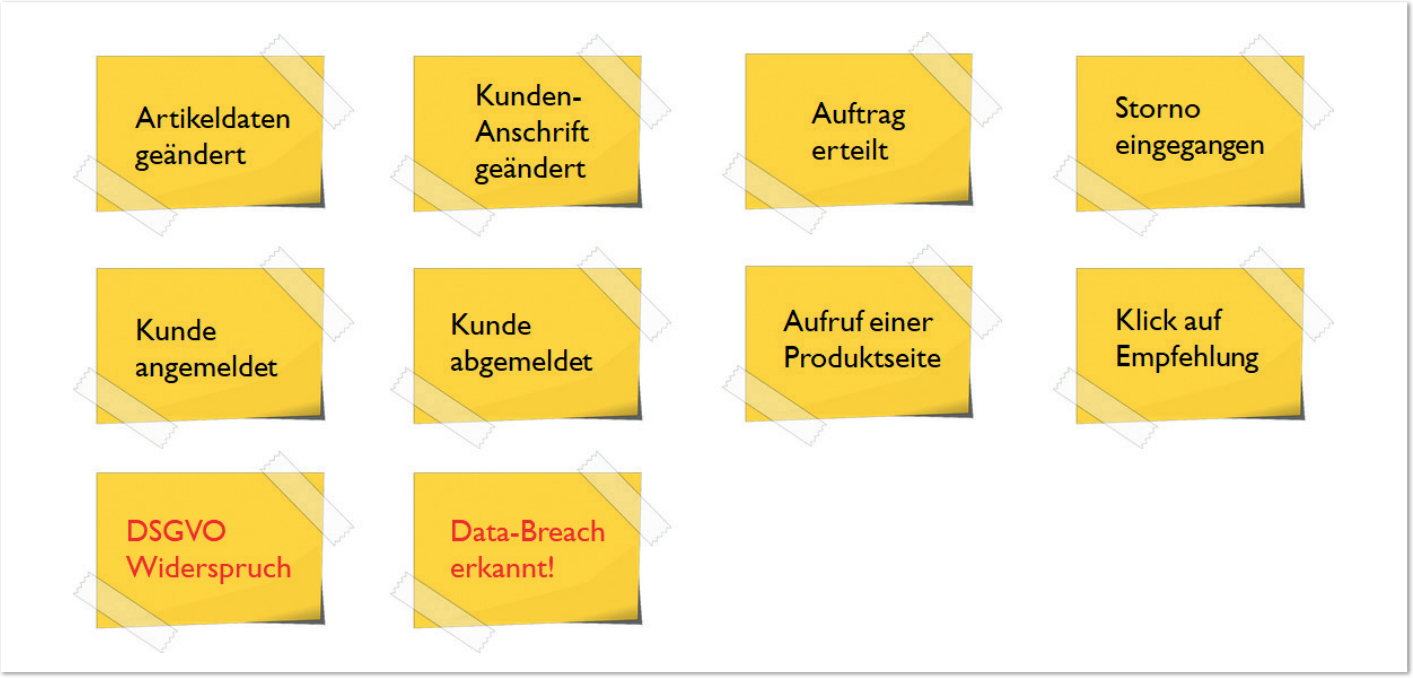


Abbildung 5: Event Storming (Quelle: Mirko Sertic)

Dynamische Inhalte haben natürlich auch ihre Schattenseite. Above-the-Fold-Inhalte nachzuladen, kann problematisch werden, wenn dadurch ein kompletter Content-Reflow der ganzen Seite gestartet wird. Auch ein Flash-of-Unstyled-Content kann in Verbindung mit Lazy-Loading das Seitenlayout vorübergehend zerstören. Auf diese Effekte muss daher unbedingt beim Einsatz von Progressive Enhancement geachtet werden. Auch das Zusammenspiel von Progressive Enhancement und Search Engine Optimization (SEO) hat seine ganz speziellen Nuancen. Allein über diesen Themenkomplex können ganze Artikel geschrieben werden. Zusammengefasst daher folgendes: Wenn SEO wichtig und geschäftskritisch ist, muss hier schon frühzeitig darauf geachtet werden.

Lektion #5: Systemintegration

Systemintegration kann in Verbindung mit dem SCS-Ansatz eine hart erlernte Lektion werden. Mit Systemintegration ist die Kommunikation einzelner SCS untereinander sowie die Kommunikation mit Drittanbietern gemeint. Fehler bei dieser Integration können die ganzen Vorteile der SCS-Architektur negieren. Das SCS-Manifest schreibt vor, dass Kommunikation soweit wie möglich asynchron erfolgen soll. Zusätzlich schreibt das Manifest vor, dass geteilte Infrastruktur minimiert werden soll. Was bedeutet das nun für die technische Umsetzung? Asynchron bedeutet, dass keine synchrone HTTP-Kommunikation zwischen Systemen und Komponenten erfolgen soll. Keine geteilte Infrastruktur bedeutet, dass es keine zentrale Datenbank mit allen Stammdaten für Kunden oder Produktinformationen gibt. Die Idee hinter diesen bewussten Redundanzen ist einfach: Zu viele synchrone Schnittstellen erhöhen das Risiko, letztendlich doch einen verteilten Monolithen zu bauen. Die Nicht-Verfügbarkeit eines Systems kann nicht die Verfügbarkeit anderer Systeme beeinträchtigen. Diese Grundsätze haben weitreichende Konsequenzen. Da es keinen zentralen Datentopf mit Stammdaten gibt, müssen diese in alle abhängigen Systeme repliziert werden. Diese Replikation kann beispielsweise via Event Sourcing erfolgen. Da in der Konsequenz auch sensible Daten wie Kundeninformationen oder Finanztransaktionen in unterschiedlichen Systemen verteilt werden, darf auf keinen Fall das Thema Datenschutz zu kurz kommen.

Als Grundlage für das Event Sourcing benötigen wir eine Spezifikation für die benötigten fachlichen Events. Diese Spezifikation kann über ein Event Storming gefunden werden, eine Teildisziplin des Domain-Driven Design. Diese Methode hilft ebenfalls dabei, Ursache-Wirkungs-Ketten zu erkennen und besser zu verstehen *(siehe Abbildung 5)*.

In der ersten Zeile finden wir recht offensichtliche Events wie „Artikeldaten geändert", „Kundenanschrift geändert" oder „Auftrag erteilt". Diese Events können Fachlogik starten und steuern. Sie können aber auch genutzt werden, um Daten zwischen Systemen zu synchronisieren. Ein „Artikeldaten geändert"-Event kann zum Beispiel genutzt werden, um die lokale Kopie von Artikeldaten in einem SCS zu aktualisieren.

Die zweite Zeile beinhaltet weniger offensichtliche Events, wie „Kunde angemeldet", „Kunde abgemeldet" oder auch „Klick auf eine Produktempfehlung". Diese Events werden beispielsweise für ein Recommender-System und die dahinter liegenden Machine-Learning-Algorithmen benötigt, können jedoch auch für den Aufbau eines Data-Warehouse oder für Predictive Analytics genutzt werden.

Die dritte Zeile hat es in sich. Wie schon erwähnt, gelten besondere Anforderungen an das Thema Datensicherheit. Die Ereignisse „DSGVO-Widerspruch" müssen etwa das Löschen oder die Anonymisierung von persönlichen Daten in allen Systemen starten. Im Umkehrschluss muss das „Data Breach erkannt!"-Event alle Kennwörter oder Tokens von betroffenen Benutzern in allen Systemen invalidieren oder auch ein Report über die betroffenen Daten an eine zentrale Instanz generieren.

Das SCS-Manifest schreibt vor, dass zentrale Infrastruktur minimiert werden soll. Als Konsequenz der asynchronen Architektur

benötigen wir dennoch eine Middleware für den Datenaustausch. Hier empfiehlt sich der Einsatz einer Pull-Architektur, da dies die Komplexität auf der Ebene „Message-Broker" reduziert und die Implementierung von Back-Pressure erleichtert. Apache Kafka kommt als Lösung infrage, da wir auf diesem Wege nicht nur ein Publish-Subscribe Messaging bekommen, sondern auch ein verteiltes Transaktionslog. Dieses Log zahlt auf eine wichtige Qualitätsmetrik der verteilten Architektur ein: Mean-Time-to-Repair/Recover. Über das Kafka-Transaktionslog ist eine Zeitreise in die Vergangenheit möglich, um so effizienter im Fehlerfall einen konsistenten Systemzustand herzustellen.

Finale Lektion: Lessons Learned

Ich möchte nun zur letzten Lektion kommen: der Zusammenfassung. Was hat die „Welt, bleib wach!" GmbH bei der Migration auf die SCS-Architektur gelernt, was ist wichtig, was funktioniert, was würde im nächsten Projekt anders gemacht werden?

Im Wesentlichen waren die wichtigsten Punkte aus dem Migrationsvorhaben die Ergebnisse aus Lektion #1 und Lektion #7. Ohne organisatorische Anpassungen kann die SCS-Architektur ihre größten Hebel nicht entfalten. Die organisatorischen Anpassungen bedingen auch eine Anpassung der technischen Schnittstellen. Durch Fehler bei der technischen Systemintegration können alle organisatorischen Anpassungen wieder zunichte gemacht werden. Zu viele synchrone Schnittstellen erhöhen das Risiko, letztendlich doch nur einen verteilten Monolithen zu bauen. Eine Landschaft aus SCSs ist eine verteilte Landschaft. Verteilte Systeme bedeuten auch potenziell verteilte Probleme. Die Fehleranalyse und Wartung können deutlich komplexer werden als bei einem Single-Point-of-Failure-Monolithen. Infrastrukturkosten müssen im Auge behalten werden. Der Monolith wird in Systeme zerlegt, die wiederum aus einzelnen Services bestehen können. Jeder einzelne Service braucht Rechenzeit, Speicher usw. In der Summe werden nach der Migration deutlich mehr Ressourcen benötigt als für den Monolithen. Das ist nicht verwunderlich, da beispielsweise für jeden Service das Spring Framework in eine Java VM geladen wird, und genau diese Redundanzen kosten Speicher und damit auch Geld. Die verteilte Infrastruktur bringt für das Thema Datensicherheit ganz neue Anforderungen und Probleme mit sich. Es lohnt sich, diese Aspekte schon in einer frühen Projektphase zu adressieren.

Domain-Driven Design ist ein Evergreen. Die fachliche Denkweise hinter DDD ist eine sehr gute Hilfe, um technische und fachliche Strukturen eng aneinander zu binden und so Kommunikations- und Verständnisprobleme zwischen Stakeholdern und Projektteams zu reduzieren und alle Beteiligten an einem Strang ziehen zu lassen. Event Storming ist ein gutes Hilfsmittel, um in einer komplexen Welt Ursache-Wirkungs-Ketten zu identifizieren und zu verstehen. Der daraus entstehende, reaktive und asynchrone Ansatz bringt eine ganze Fülle an Nachrichten und damit verbunden Nachrichtentypen hervor. Diese Nachrichten sind Teil des API und sollten mit entsprechender Sorgfalt spezifiziert werden. Bei der Implementierung der verteilten Welt muss viel Wert auf Resilience gelegt werden. Ein einzelnes, nicht verfügbares System darf nicht die komplette Systemlandschaft beeinträchtigen. Für die Nachrichtenverarbeitung empfiehlt sich Pull-Messaging in Verbindung mit Back-Pressure, wie es etwa mit Apache Kafka sehr einfach zu implementieren ist. Die Konsequenz der verteilten Architektur ist die systemübergreifende Eventual Consistency, eine Erfahrung, die für Anwender eines monolithischen Systems neu sein dürfte. Mit der wichtigste Qualitätsfaktor dieser verteilten Systemarchitektur ist die „Mean Time to Recover/Repair". Ein verteiltes System lässt sich nicht so einfach komplett neu starten wie ein Monolith. Für ein Desaster-Recovery kann es notwendig werden, Stammdaten komplett neu zwischen Systemen zu synchronisieren. Die Laufzeit ist abhängig vom Datenvolumen und kann durchaus Stunden bis Tage dauern.

Trotz aller Fallstricke kann auf jeden Fall gesagt werden, dass die SCS-Architektur ein guter Ansatz für die Produktentwicklung ist. Mit dem verteilten Ansatz hinter der SCS-Architektur können Organisationen gut skaliert werden. Die SCS-Architektur lenkt den Fokus auf das System/das Produkt und auf dessen Schnittstellen. Auch lässt sich sagen, dass die SCS-Architektur mit Brown-Field-Migrationsprojekten funktioniert, auch wenn an dieser Stelle das Risikomanagement noch intensiver erfolgen sollte als bei gewöhnlichen IT-Projekten.

Die Erfahrungen mit dem SCS-Ansatz sind durchaus positiv. Wichtig zu betonen ist der agile Grundsatz: Es ist besser, auf Veränderungen zu reagieren, als einen strikten Plan zu verfolgen. Ein Migrationsprojekt kann nicht von Anfang bis zum Ende komplett durchgeplant werden. Es werden immer Probleme auftauchen, und auf diese Probleme muss sachgemäß und zeitnah reagiert werden. Das SCS-Manifest ist in diesem Kontext als Leitplanke zu verstehen. Leitplanken bieten klare Abgrenzungen, sollten jedoch auch verschoben oder komplett abgebaut werden, etwa wenn eine breitere Straße benötigt wird oder gerade ein Schwertransporter den Weg passiert. Das soll natürlich keine Ausrede für Chaos sein, sondern mehr eine Aufforderung, allzu dogmatische Vorgehensweisen mit einem kritischen Auge zu betrachten.

In diesem Sinne möchte ich mich für die Aufmerksamkeit bedanken. Für Rückfragen, Anmerkungen und Kritik stehe ich natürlich gerne zur Verfügung. Vielen Dank!

Mirko Sertic
mirko@mirkosertic.de

Mirko ist Software Craftsman im Web/eCommerce-Umfeld. In Funktionen als Software-Entwickler, Architekt und Consultant in Projekten in Deutschland und der Schweiz sammelte er Erfahrungen mit einer Vielzahl von Frameworks, Technologien und Methoden. Heute arbeitet er als IT-Analyst bei der Thalia Bücher GmbH in Münster mit Schwerpunkt auf Java, eCommerce sowie Such- und Empfehlungs-Technologien. Seine Freizeit verbringt er mit seiner Familie und hin- und wieder mit Open-Source-Projekten.

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 Interessenverband 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
ViSdP: Mylène Diacquenod
Redaktionsleitung: Lisa Damerow
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:
Caroline Sengpiel,
DOAG Dienstleistungen GmbH

Fotonachweis:
Titel: Original © vanillamilk | *https://de.fotolia.com*
S. 9: Bild © freepik | *www.freepik.com*
S. 13: Bild © Le Moal Olivier | *https://de.123rf.com*
S. 15: Bild © sergign | *https://de.fotolia.com*
S. 18: Bild © maxicam | *https://de.123rf.com*
S. 22: Bild © Dusit Pamyakhom | *https://de.123rf.com*
S. 30: Bild © echiechi | *https://de.123rf.com*
S. 36: Bild © Marina Putilova | *https://de.123rf.com*
S. 40: Logo © MongoDB | *www.mongodb.com*
S. 46: Bild © Andrii Torianyk | *https://de.123rf.com*
S. 50: Bild © Stefan Hildebrandt
S. 54: Bild © Monsit Jangariyawong | *https://de.123rf.com*
S. 60: Bild © Diana Johanna Velasquez | *https://de.123rf.com*

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

Jetzt Anmelden!

Sei dabei

Go DevOps 2019

2.+3. Sept. 2019 in Berlin

DOAG

godevops.doag.org

esentri

IT'S IN ALL OF US *TO CREATE*

Jetzt bewerben unter
career@esentri.com

#inallofus
www.esentri.com