Ausgabe 04/2021

Java aktuell

iJUG Verbund
www.ijug.eu

Open Source
Rechtliche Risiken, Erfahrungsberichte, APM-Tools

Kubernetes
Automatisches Nachladen von Konfigurationen ohne Neustart

Datenwissenschaft & KI
für Java-Entwickler, Kernprinzipien und Ethik von KI

OPEN Source

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

* die Online-Konferenz für IT-Expert*innen,
Studierende und Unternehmen
am 01. Oktober 2021

JUG SAXONY DAY

Infos & Tickets: **www.jug-saxony-day.org**

Liebe Leser/innen der Java aktuell,

Java und Open Source gehen Hand in Hand. Zahlreiche Java-Fans engagieren sich in einer Vielzahl von Open-Source-Projekten, um einen Mehrwert für die Community zu schaffen. Daher befasst sich diese Ausgabe der Java aktuell mit dem Thema Open Source.

Den Anfang macht Anwalt und Mitglied des DOAG Legal Council Dr. Jan Bohnstedt. Er beleuchtet mögliche Lizenzrisiken bei der Verwendung von Open-Source-Software. Im Anschluss nehmen uns Sandra Parsick und Georg Berky mit auf eine Zeitreise zurück zu den Anfängen ihrer Open-Source-Beiträge bis hin zu ihrem heutigen Maintainer-Status. Dabei zeigen die beiden, dass der Einstieg doch einfacher ist, als viele vielleicht denken.

Ab Seite 25 präsentiert Dr. Andreas Brunnert einige der beliebtesten frei verfügbaren Application Performance Monitoring (APM) Tools und gibt einen Überblick über deren Vor- und Nachteile. Zum Abschluss vergleicht er diese mit kommerziellen Werkzeugen. Im darauffolgenden Artikel stellt Toni Epple sein eigenes GitHub-Projekt näher vor. Darin präsentiert er frgaal: einen Retrofit-Compiler für Java, der es ermöglicht, moderne und sogar experimentelle Sprachfeatures zu verwenden und trotzdem die Kompatibilität mit älteren Runtimes zu erhalten. Er stellt den Compiler vor und zeigt Use Cases.

Eine Möglichkeit, wie man Java-Anwendungskonfigurationen automatisch und ohne Neustart in Kubernetes nachladen kann, beschreibt Frank Rosner ab Seite 42. Dabei verwendet er ausschließlich Kubernetes-Bordmittel und Dateisystemoperationen, sodass keine Abhängigkeiten zu externen Konfigurationsspeichern notwendig sind.

Michal Harakal befasst sich in seinem Artikel mit den Grundlagen der Datenwissenschaft und der künstlichen Intelligenz. Er zeigt, wie die Datenwissenschaft von der Softwareentwicklung profitieren kann und umgekehrt. Weiterhin beschreibt er, inwiefern es Überschneidungen der Themenbereiche gibt. Passend dazu nimmt Corinna Wiedenmann im nachfolgenden Artikel die Kernprinzipien der künstlichen Intelligenz unter die Lupe. Sie vergleicht diese mit den Grundprinzipien der Bioethik und führt dabei konkrete Beispiele auf.

Im vergangenen Jahr hat der Großteil der Menschen pandemiebedingt von Zuhause aus gearbeitet, sofern es die Möglichkeit dazu gab. Auch in Zukunft könnte das Homeoffice eine andauernde Option für Mitarbeiter werden. Die freiberufliche Softwareentwicklerin Birgit Kratz teilt ihre Erfahrungen und gibt wertvolle Tipps für Hardware, Software und den Kontakt zu Kollegen/innen.

Barrierefreiheit wird im täglichen Leben schon in vielen Bereichen umgesetzt. Doch wie sieht es im Web aus? Einen Crashkurs für die barrierefreie Gestaltung von Websites gibt Anna Maier in ihrem Artikel. Dabei gibt sie Tipps und Tricks, was man dabei beachten sollte und wie auch Benutzer ohne Einschränkung von der Umsetzung profitieren können.

Wir wünschen euch viel Spaß beim Lesen!

Eure

Lisa Damerow
Redaktionsleitung Java aktuell

Inhalt

Juristische Risiken von Open-Source-Software

Beliebte Open Source APM Tools und Standards im Überblick

Wie Datenwissenschaft und KI von der Softwareentwicklung profitieren und umgekehrt

Künstliche Intelligenz: Prinzipien und Vergleiche zur Bioethik

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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

17. März 2021

AdoptOpenJDK 16

Das hat ins letzte Tagebuch nicht mehr reingepasst, aber der Vollständigkeit halber: Java 16 ist gestern freigegeben worden. Die Inhalte liste ich nicht noch mal auf, sie sind ja auch leicht zu finden *[1]*. Und im AdoptOpenJDK-Projekt, das sich gerade in der Transformation zu Eclipse Adoptium befindet, sind die verschiedenen Varianten (OpenJDK und OpenJ9-basiert) seit heute verfügbar.

18. März 2021

MicroProfile und Jakarta, Fortsetzung

Die aus rechtlichen Gründen erforderliche Umbenennung des „alten" javax-Namespace in jakarta im Jakarta-EE-Projekt setzt sich nun als Teil der Annäherung zwischen den Projekten bei Eclipse MicroProfile fort. In der MicroProfile Google Group wird mit einem Votum die Bereitschaft der MicroProfile-Committer abgefragt, diese Migration noch 2021 zu beginnen und Spezifikationsprojekten die direkte Umsetzung zu erlauben. Das Votum scheitert am Ende knapp (14 Mal Ja zu 14 Mal Nein), aber so oder so ist klar, dass es noch enormen Diskussionsbedarf gibt. Wobei – am Ende geht es weniger um Diskussionen: Die meisten der Nein-Stimmen kamen von Red Hat (allerdings auch ein Ja, es gibt keinen „Fraktionszwang"). Der Konzern investiert viel in MicroProfile, und während in der Diskussion von anderer Seite das Stichwort „vendor neutrality challenge" fällt, fehlt wohl mindestens aus Red-Hat-Sicht ein konkreteres Commitment weiterer Partner (da gibt es ja noch weitere große Unternehmen…), um sich dann auch auf genauere Daten festzulegen. Dass es prinzipiell – und bald – nötig ist, darin scheinen sich zumindest alle einig zu sein.

25. März 2021

Jetty auf Jakarta EE 9

Der altehrwürdige Jetty Servlet Container ist jetzt auch auf die Servlet-5.0-Spezifikation migriert (Teil von Jakarta EE 9 mit dem neuen Namespace jakarta.*). Laut Eclipse-Blog erforderte die „Big Bang"-Migration eine große gemeinsame Anstrengung der Jetty Community, aber mit den Updates für Jakarta mitzuschwimmen, war offensichtlich allen sehr wichtig.

28. März 2021

Jakarta EE 10 Contributor Guide

Apropos „Community-Anstrengungen": Die Jakarta EE Ambassadors um Reza Rahman haben den „Jakarta EE 10 Contributor Guide" veröffentlicht. Wer mit dem Gedanken spielt, sich an einem größeren Open-Source-Projekt zu beteiligen: Viel einfacher wird es nicht mehr. Der Leitfaden beschäftigt sich nur am Rande mit dem bisschen Papierkram, der für tatsächliche Code-Beiträge nötig wird (Stufe 2 der Mitarbeit). Im Wesentlichen werden Themen aufgelistet, die für EE 10 wichtig sind, um potenziellen Mitstreiterinnen und Mitstreitern direkt konkrete Ideen zu geben *[2]*.

6. April 2021

Google gegen Oracle – der finale Akt

Vor gerade mal einem guten Jahrzehnt hatte Oracle den Konkurrenten wegen kopierten Java-Quellcodes im Android-Betriebssystem auf Schadensersatz beziehungsweise Gewinnbeteiligung in Milliardenhöhe verklagt. Was lange währt, wird endlich … nichts! Im abschließenden Verfahren vor dem US Supreme Court urteilten die Richter nun mit 6:2 Stimmen für den Angeklagten. Der entscheidende Satz im Urteil: „Google's copying of the Java SE API, which included only those lines of code that were needed to allow programmers to put their accrued talents to work in a new and transformative program, was a fair use of that material as a matter of law." *[3]* Oracle geht also leer aus. Eindeutiger Sieger: Google – und eine ziemlich große Schar hochbezahlter Anwälte auf beiden Seiten.

Für die Software-Industrie insgesamt dürfte das Urteil eher Segen als Fluch sein. Letztlich ging es um kopierten API-Code in nicht allzu großem Umfang. Beim „fair use" nach amerikanischem Recht geht es um die Abwägung des Urheberrechts gegen die Interessen der Allgemeinheit. Wenn APIs immer wieder zwanghaft neu „erfunden" werden, um keine Klagen der Konkurrenz fürchten zu müssen, ist das wohl ein größeres Problem, als wenn öfter mal erfolgreiche Schnittstellen in Teilen kopiert werden. Im Idealfall wird gleich gemeinsam an APIs gearbeitet – genügend Beispiele gibt's ja inzwischen auch in der Java-Welt.

10. April 2021

Jakarta EE und MicroProfile – Umfrageergebnis

Die in der letzten Tagebuch-Ausgabe erwähnte Community-Umfrage zum „Alignment" von MicroProfile und Jakarta EE ist ausgewertet. Etwas mehr als 200 Rückmeldungen sind nicht besonders viel, aber da laut Reza Rahman, der die Umfrage organisiert hat, viele zusätzliche Kommentare angefügt wurden, scheinen es zumindest diejenigen gewesen zu sein, die sich intensiver mit dem Thema auseinandergesetzt haben. Hier noch mal die Auswahlmöglichkeiten:

- A1: ausgewählte MP-Spezifikationen werden ins Jakarta-Projekt überführt, ohne Namespace-Anpassung
- A2: mit Anpassung
- B: MP-Spezifikationen werden aus Jakarta heraus referenziert
- C: in Jakarta EE werden eigene Versionen von MP-Spezifikationen entwickelt

Mit knapp 58 Prozent ist eine sehr deutliche Mehrheit für A2, gefolgt von 20 Prozent für A1, ganz hinten landet die viel aufwendigere Variante C. Das Ergebnis hat natürlich keine direkten Konsequenzen, kann aber die weitere Diskussion beeinflussen. Vermutlich ist das Meinungsbild aber auch in den betroffenen Projekten ähnlich *[4]*.

14. April 2021

Adoptium organisiert sich

Das neue Eclipse-Adoptium-Projekt – vormals AdoptOpenJDK – hat jetzt seine eigene Working Group innerhalb der Eclipse Foundation und konkretisiert fleißig Ziele und Prozesse. Ziel sei es zum einen, eigene Binaries unter dem Namen Temurin zu veröffentlichen, aber ebenso Varianten der „Working Group Members" zu promoten, die die Java-SE-TCK-Tests bestehen und den vom Adoptium AQAvit-Projekt definierten Qualitätskriterien genügen. Releases sollen wie bereits beim AdoptOpenJDK vierteljährlich und abgestimmt auf das „upstream" OpenJDK-Projekt erstellt werden. JDK Releases auf Basis von Eclipse OpenJ9 werden aber wohl aus organisatorischen Gründen erst mal nicht direkt von Adoptium zur Verfügung gestellt, sondern von IBM, die den J9-Code bereits vor einigen Jahren an die Eclipse Foundation übergeben hatten. Eines der Gründungsmitglieder in der Working Group ist übrigens der iJUG e.V., neben Schwergewichten wie IBM, Microsoft, Red Hat und ein paar kleineren Unternehmen.

20. April 2021

GraalVM 21.1 mit JDK 16 Support und zweistufiger Kompilierung

GraalVM 21.1 bringt neben Java-8- und Java-11-Binaries auch experimentellen Support für Java 16 (alle Komponenten darin werden laut Release Notes als experimentell angesehen). Für MacOS fallen allerdings die Java 8 Builds weg – zumindest in der kostenlosen Community Edition. Das Release bietet einige Optimierungen, etwa besseres Handling von „inverted loops" (mit Abbruchbedingung am Ende) oder der Wegfall unnötiger Speicherbarrieren bei sequenziellen „volatile writes" (die zum Beispiel sehr stark in ConcurrentHashMap auftreten können). Beim Erstellen von nativen Images kann jetzt die Liste der zu unterstützenden Locales und die Default Locale angegeben werden. Außerdem ist die mit 20.3 noch als experimentell eingeführte zweistufige („multi-tier"-)Kompilierung jetzt der Default für alle Sprachen, die auf dem Truffle Interpreter laufen (wozu inzwischen auch Java gehört). Damit werden für schnelleres Warm-up zwei Läufe hintereinander durchgeführt – der erste ist auf Kompilierungsgeschwindigkeit ausgelegt, der zweite für umfangreiche Optimierungen zuständig.

25. Mai 2021

Microsoft baut eigenes (Open)JDK

Logische Konsequenz der Microsoft-Strategie der vergangenen Jahre, unter anderem mit dem Beitritt zum OpenJDK-Projekt und dem Kauf von jClarity (die sich nicht nur mit JVM-Performance befasst haben, sondern dessen Gründer das AdoptOpenJDK-Projekt maßgeblich vorangetrieben haben): Der Konzern aus Redmond baut jetzt sein eigenes JDK und stellt dieses nicht nur seinen Azure-Kunden, sondern allen unter der GPL 2.0 mit Classpath Exception zur freien Verfügung. Die Builds basieren auf dem OpenJDK-Quellcode sowie den Skripten und Test Suites von Eclipse Adoptium. Die aktuellen Versionen sind Java 11, das als „LTS"-Version bis mindestens 2024 Updates erhalten soll, sowie Java 16 (beide für macOS, Linux und Windows). Soweit nichts, was andere Hersteller im Rahmen von Adopt(ium) nicht schon länger machen. Aber erstens ist es ein weiterer großer Hersteller, der sich aktiv an Entwicklung und Bugfixing beteiligt; und auch wenn sie – wie alle – im Zweifel erst an die eigenen Kunden denken, werden wichtige Fixes früher oder später im „Upstream"-Projekt landen. Zweitens engagiert sich Microsoft (natürlich auch nicht komplett uneigennützig) stark für ARM-Hardware, sodass es inzwischen auch Java 16 Releases für ARM unter Windows und Linux gibt und ein MacOS-Port als Early Access zur Verfügung steht.

6. Mai 2021

Kotlin 1.5

Version 1.5 der Programmiersprache Kotlin ist fertig. Es gibt ein paar neue Features, aber im Wesentlichen ging es um Stabilisierung und Konsolidierung. Unter anderem hat das „JVM Backend" den Beta-Status verlassen. Die verschiedenen Backends sind sozusagen die Endstufe des Compilers und bedienen die jeweilige Zielplattform (JVM, JavaScript, native); sie teilen sich nun alle eine gemeinsame „internal representation" (IR), was die zukünftige Entwicklung deutlich effizienter macht. Als Default JVM Target wird nun 1.8 angenommen. Mit 1.8 oder höher werden nun Single Abstract Methods (SAMs) erst zur Laufzeit über invokedynamic eingebunden (so wie die Java-eigenen Lambdas). Für „Plain Kotlin Lambdas" ist dieses Feature aber noch experimentell.

10. Mai 2021

CargoTracker jetzt in der Cloud

Die Jakarta-EE-Demo-Applikation „Cargo Tracker" hat sicher jede(r) zumindest schon mal gesehen. Jetzt ist sie auch als öffentliche Demo in der Cloud verfügbar (auf Kubernetes und jede Nacht frisch aus dem Cargo-Tracker-GitHub-Projekt gebaut) *[5]*. Gut, da steht noch Java EE auf der Titelseite, aber das lässt sich ja schnell im Projekt fixen – ist ja schließlich alles Open Source.

11. Mai 2021

Jakarta EE und MicroProfile: Neues Jakarta „Core Profile"

Die Jakarta EE Ambassadors haben sich zum Verhältnis zwischen MicroProfile und Jakarta EE und zur Cloud Native for Java (CN4J) Alliance ausgelassen, die die Zusammenarbeit zwischen beiden verbessern soll und aus Vertretern beider Lager besteht (na gut, so getrennt sind die beiden ja insbesondere auf der Mitglieder-Ebene nicht,

ich habe bewusst übertrieben). Im Wesentlichen handelt es sich um eine Zusammenfassung des gemeinsamen Verständnisses, das sich über beide Projekte hinweg gebildet hat, etwa: „Jakarta EE will continue to be the stable core for a very broad ecosystem. MicroProfile will continue to strongly focus on microservices, velocity, and innovation". Dabei geht es auch um ein neues Jakarta „Core Profile", das diejenigen Spezifikationen enthält, von denen MicroProfile abhängt, plus ausgewählte MicroProfile-Spezifikationen wie MP Config, die zu Jakarta migriert werden sollen, plus CDI Lite. Das Core Profile ist schon vom Jakarta-Projekt anvisiert, als Release 10 mit Zieldatum 31. Dezember 2021. Nebenbei: Das WebProfile soll in Release 10 um drei weitere Spezifikationen erweitert werden (MVC 2.1, Concurrency 3.0 und Config 1.0) *[6]*.

12. Mai 2021

Apache wird Gastmitglied bei Eclipse

Nochmal Jakarta: Die Apache Software Foundation wird zunächst als Gastmitglied Teil der Jakarta EE Working Group.

14. Mai 2021

Scala 3

Langes Warten ist nicht immer vergebens: Scala 3 ist endlich fertig! Die sowohl funktionale als auch objektorientierte Sprache wurde für die neue Version seit 2018 stark umgekrempelt. Bessere Sprachkonsistenz und leichtere Erlernbarkeit waren dabei die Hauptziele, neben der Einbettung in das theoretische Konzept des DOT-Kalküls („dependent object type calculus").

25. Mai 2021

Jakarta EE 9.1 – für Java SE 11

Jakarta EE 9.1 ist heute freigegeben worden. Die neuen Features sind schnell aufgelistet: Support für Java 11. Auf jeden Fall schwingt sich das Projekt jetzt auf regelmäßige Releases ein (das nächste Major Release soll vor Ende des Jahres kommen) und: Es gibt bereits fünf als kompatibel zertifizierte Implementierungen, die sechste soll bald folgen. Mit dabei: Apache TomEE, das zum ersten Mal seit zehn Jahren wieder offiziell zertifiziert ist. Die damalige Zertifizierung war für Java EE 6! Manche werden sich dunkel erinnern, dass die Apache SF damals – im Streit um eine TCK-Lizenz für die Java-SE-Implementierung Apache Harmony – den JCP verlassen hat. Danach kamen zähe Verhandlungen hinter den Kulissen, dann der Quasi-Stopp von Java EE und der Übergang zu Jakarta EE – und schwupp waren zehn Jahre um. Aber am Ende ist alles gut geworden!

26. Mai 2021

IBM und Microsoft arbeiten zusammen

IBM und Microsoft arbeiten gemeinsam an der Nutzung von WebSphere-Produkten in der Azure Cloud. Das neueste Ergebnis ist eine Lösung für das Netzwerk-Deployment des WebSphere Application Server auf Azure Linux VMs, mit dem das Aufsetzen größerer Cluster mit allen benötigten Ressourcen vereinfacht wird.

27. Mai 2021

Quarkus 2.0.0.CR3

Red Hats Java Cloud Native Framework Quarkus geht in die Zielgerade für Release 2.0 mit einem dritten Release Candidate. Die neue Version basiert unter anderem auf Vert.x 4 und der MicroProfile-4-Spezifikation und setzt Java 11 voraus. Außerdem enthält sie eine Alpha-Version von „continuous testing", das im Entwicklermodus kontinuierlich im Hintergrund laufen kann, um sofortiges Feedback zu geben, wenn Tests verletzt werden. Dabei werden in einem ersten Durchlauf alle Tests ausgeführt und gleichzeitig analysiert, welche Tests von welchem Code abhängen, sodass die Ausführung gezielt anhand der Änderungen gesteuert werden kann.

30. Mai 2021

Liste aller JUGs weltweit

Ist deine Gruppe gelistet? Unter *[7]* soll eine weltweite Liste und Karte aller JUGs entstehen. Einträge erfolgen per Pull Request auf GitHub.

Referenzen

[1] *https://jdk.java.net/16/*
[2] *https://jakartaee-ambassadors.io/guide-to-contributing-to-jakarta-ee-10/*
[3] *https://www.supremecourt.gov/opinions/20pdf/18-956_d18f.pdf*
[4] *https://reza-rahman.me/2021/04/10/jakarta-ee-microprofile-alignment-survey-results/*
[5] *https://cargo-tracker.j.scaleforce.net*
[6] *https://jakartaee-ambassadors.io/2021/05/10*
[7] *https://world-wide-jugs.github.io/GlobalWWJugs/*

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

In der letzten Ausgabe berichtete ich an dieser Stelle vom Beitritt des iJUG e. V. in die Eclipse-Arbeitsgruppen Jakarta und Adoptium. Die erstere kümmert sich bekanntlich um den Nachfolger von Java EE unter dem inzwischen jedem bekannten Namen Jakarta EE, die letztere stellt unter freier Lizenz eine JDK-Distribution für eine Vielzahl technischer Plattformen bereit.

Nun kommt noch eine dritte hinzu: Wir, also der iJUG, sind nun auch der Eclipse-Arbeitsgruppe „MicroProfile" beigetreten und sorgen mit einem „halben" Stimmrecht dafür, dass die Interessen unserer Mitglieder dort entsprechend Gehör finden. Das halbe Stimmrecht ergibt sich aus der Tatsache, dass die Eclipse Foundation leider eher kapitalistisch als demokratisch oder meritokratisch ist: Wer mehr bezahlt, darf mehr entscheiden, und angesichts der horrenden Preise eines vollen Stimmrechts sind wir mit dem halben (tatsächlich ist es sogar nur ein Bruchteil) schon gut bedient – es sei denn, es fände sich eine unerwartete Mehrheit für eine erhebliche Beitragsanhebung (was vermutlich niemand will, da bislang von niemandem beantragt).

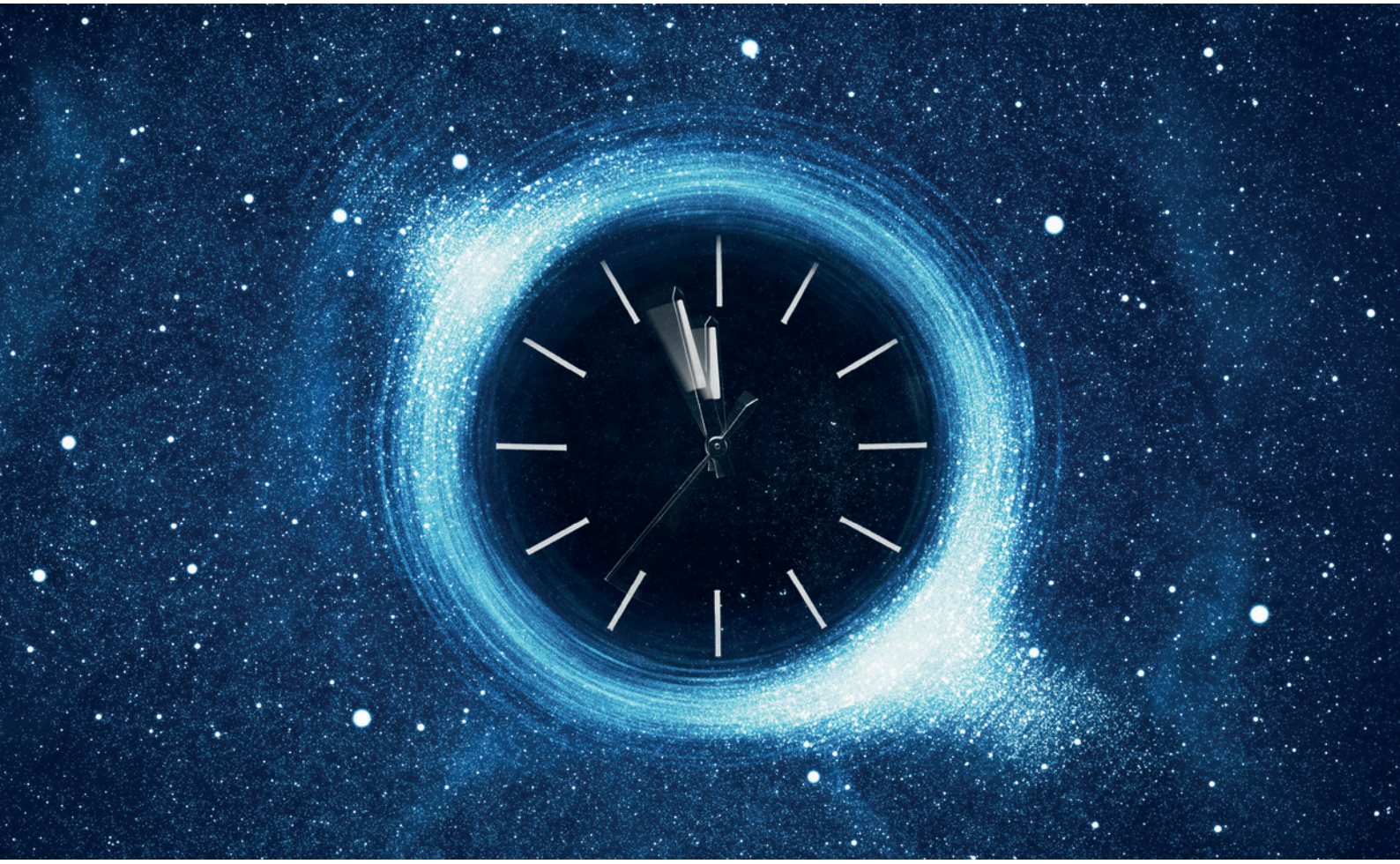
Damit sind wir, also der iJUG an sich, und über diesen auch wir Java-User, in einer starken Position. Wir bestimmen nun aktiv mit über Wohl und Wehe wichtiger Säulen des Java-Universums: Ex-Java EE und JDK (und damit übrigens indirekt über Lizenzart und Lizenzkosten von JDK und JRE; ich erinnere an das Drama damals um die Oracle-Lizenzpolitik). Damit sind wir einem wichtigen Ziel der Java-User-Vereinigung im deutschsprachigen Raum einen großen Schritt näher und gestalten nun in eurem Namen aktiv an der Zukunft der weltweit meistverbreiteten Programmierplattform. Das große Ziel, Java komplett in der Hand der User, ist zwar immer noch weit entfernt, aber eben nicht mehr ganz soooo weit.

Doch so ganz ohne euch geht das nicht. Einfluss ist das eine, doch da gibt es auch noch die Kehrseite der Medaille: Das, was wir da bestimmen, muss auch jemand umsetzen, also programmieren. Die Eclipse Foundation selbst, wie auch die Arbeitsgruppen und auch der iJUG selbst, verfügen weder über eigene Entwickler/innen noch bezahlen sie welche. Das Geld, das in die Arbeitsgruppen fließt, dient Marketing-, Rechtsberatungs- und Verwaltungszwecken. Direkt in das endgültige Produkt, also Jakarta oder JDK, fließt davon nichts! Die Arbeitsgruppen der Eclipse Foundation heißen ja aus gutem Grund „Arbeitsgruppen" – denn wer sich dort engagiert, soll Arbeit leisten. Und so waren sich Jan (Westerkamp) und ich (also eure Botschafter bei der Eclipse Foundation) recht schnell einig, dass wir nun einen weiteren Schritt gehen und euch ins Boot holen müssen: Wir brauchen Committer!

Da wir aber auch keine unbegrenzten Ausbildungsmöglichkeiten haben, sind wir im ersten Schritt darauf angewiesen, zunächst diejenigen von euch zu mobilisieren, die bereits Erfahrung mit (kleinen) eigenen Beiträgen zu Jakarta EE, MicroProfile und Adoptium haben. Wir würden euch gerne coachen und zu Committern ausbilden. Das bedeutet: Mentoren aus den genannten Projekten betreuen euch auf dem Weg vom gelegentlichen Contributor zum regelmäßigen Committer (ach ja: natürlich dann mit eigenem Stimmrecht!). Der iJUG und seine Partner legen weitere Vorteile obendrauf, beispielsweise eine Eintrittskarte zur JavaLand, wo ihr über eure Erfahrung in diesem „Stipendium" (so nennen wir das) berichten könnt. Wie wäre das? Klingt gut? Du programmierst gerne und möchtest regelmäßig am Zentralgestirn des Java-Universums hacken? Dann schreibt formlos an: **stipendium@ijug.eu**. Die Plätze sind begrenzt, also haltet euch ran! Wir freuen uns auf eure Zuschriften!

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.



Unbekannte Kostbarkeiten des SDK Heute: Ahead-of-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.

Unser heutiges Thema ist die AOT-Kompilierung, die mehrere Jahre Bestandteil des JDK war. Die AOT-Kompilierung ist ganz unzweifelhaft den unbekannten Kostbarkeiten zuzurechnen, da sie so selten verwendet wurde, dass sie nun wegen Nichtverwendung wieder aus dem JDK entfernt wird. Eventuell liegt es auch an wenig sinnvollen, dafür aber umso aufwendigeren Arten der Verwendung. Wir werden dies näher beleuchten.

Ein wenig Geschichte

Das JEP 295 – *Ahead-of-Time Compilation* *[1]* liest sich in der Zusammenfassung als „*compile Java classes to native code prior to launching the virtual machine*". Als explizites Ziel ist die Verringerung der Startzeit sowohl von kleinen als auch von großen Anwendungen ohne spürbare Performanzeinbußen genannt. Zusätzlich sollen Entwicklungsworkflows möglichst nicht geändert werden. Das JEP 295 datiert vom 15. September 2016. Die AOT-Kompilierung wurde im JDK 9 in der Linux-Variante realisiert, die im September 2017 veröffentlicht wurde. Die Windows-Variante folgte mit JDK 10.

Die AOT-Kompilierung basiert intern auf dem Graal-Compiler, der innerhalb des Graal-Projekts *[2]* entwickelt wurde. Graal ist ein JIT-Compiler, der allerdings so flexibel ist, dass er auch ahead of time verwendet werden kann. Durch das JEP 317 – *Experimental Java-based JIT Compiler* *[3]* gelangte Graal als JIT-Compiler in das JDK und durch das

JEP 243 – *Java-Level JVM Compiler Interface* *[4]* wurde eine Schnittstelle geschaffen, diesen und andere Compiler als Alternative zu den althergebrachten C1- und C2-Compilern des JDK verwenden zu können. Graal dürfte mittlerweile allen Java-Entwicklern namentlich bekannt sein, da er in der GraalVM sowohl als AOT- als auch als JIT-Compiler seinen Dienst verrichtet. Wir haben diese Zusammenhänge bereits beschrieben *[5]* *[6]* und auf mehreren Konferenzen, beispielsweise auf dem Java Forum Stuttgart 2018, vorgestellt.

Wie geht AOT-Kompilierung?

Doch wie kann man nun mit einem Standard-JDK der Version 9 (Windows-Version 10) oder höher Klassen oder Anwendungen ahead of time kompilieren? Das `bin`-Verzeichnis des JDK enthält den AOT-Compiler als `jaotc`. Dieser Befehl erwartet neben einigen Optionen eine Liste von Klassen- oder Modulnamen, JAR-Dateien oder Verzeichnisse mit Class-Dateien. Die ersten Gehversuche auf Hello-World-Niveau sind einfach und von uns auf einem Fedora-System durchgeführt worden. Zunächst wird die Klasse kompiliert, da `jaotc` auf Bytecode arbeitet. Dann wird der AOT-Compiler aufgerufen *(siehe Listing 1)*.

Die standardmäßig erzeugte Shared-Object(SO)-Datei im ELF-Format trägt den Namen `unnamed.so`, sodass sich die Vergabe eines sinnvolleren Namens, wie im Beispiel mit `--output` demonstriert, anbietet. Um dann mit der Java Virtual Machine diese SO-Datei zu verwenden, wird sie mit der Option `AOTLibrary` angegeben *(siehe Listing 2)*.

Wenn Sie die Beispiele selbst nachvollziehen wollen, raten wir beim Befehl `jaotc` die Option `--verbose` und beim Befehl `java` die Option `-XX:+PrintAOT` zusätzlich zu verwenden. Die erste Option gestattet einen Einblick in die verschiedenen Teilbereiche des Kompilierens, die zweite Option erlaubt zu verifizieren, dass tatsächlich die SO-Datei geladen und verwendet wird.

Versionen

Im Vorgriff auf das noch näher zu erläuternde JEP 410, das die Entfernung des AOT-Compilers zum Ziel hat, hier eine kleine Versionsanalyse, falls Sie die Beispiele nachvollziehen wollen. Das JEP 410 plant die Entfernung für Java 17. Auf dem Fedora-System des Autors funktionierten die beiden oben genannten Aufrufe von `jaotc` und `java` in den OpenJDK-Versionen 9 bis 13 wie beschrieben. In den Versionen 14 und 15 funktioniert das Kompilieren, allerdings wird die Verwendung mit der JVM mit einer Fehlermeldung quittiert, da die Verwendung der Option `AOTLibrary` als ein experimentelles Feature erklärt ist und der Aufruf daher mit der Option `-XX:+UnlockExperimentalVMOptions` zu erfolgen hat.

In der Version 16 des OpenJDK fehlt das Programm `jaotc` komplett. In der Fedora-Version von Java 16, gebaut von Red Hat, existiert das Programm. Die JVM muss mit der Option des experimentellen Features gestartet werden. Ebenfalls anzumerken ist der Umstand, dass der erzeugte Binärcode nicht rückwärtskompatibel ist, wie wir das von Java gewohnt sind. Die SO-Erzeugung mit `jaotc` in einer Version n kann nicht mit einer JVM-Version größer n verwendet werden. Es erfolgt in diesem Fall ein Rückgriff auf die Bytecode-Variante der Klasse beziehungsweise – falls diese nicht existiert – ein Abbruch des JVM-Aufrufs.

Verwendungen über dem Hello-World-Niveau

Verwendet die Klasse `HelloWorld` eine zweite Klasse, muss diese explizit auch kompiliert werden. Ein transitiver Verwendungsgraph, wie ihn etwa die Native-Image-Erzeugung der GraalVM nutzt, wird nicht erstellt und verwendet.

Die Kompilierung eines ganzen Moduls ist ebenfalls möglich und durch die Option `--module` realisiert. Um etwa das ganze Base-Modul zu kompilieren, führt man den in *Listing 3* gezeigten Befehl aus.

Die so erzeugte Datei hat eine Größe von 318 MB. Selbst die gestripte Version hat noch 194 MB. Bei der Verwendung müssen dann beide SO-Bibliotheken angegeben werden *(siehe Listing 4)*.

Die Verwendung der Option `-XX:+PrintAOT` zeigt hierbei recht eindrucksvoll, wie viele Methoden für ein einfaches Hello-World verwendet werden.

High-End-Features

Die Just-in-Time-Kompilierung der JVM erfolgt gestuft *[7]*. Die Levels 0 bis 5 decken die reine Interpretation des Bytecodes bis hin zu dem durch den C2-Compiler hochoptimierten Maschinen-Code ab. Damit einher gehen verschiedene Stufen des Profilings der Ausführung des Codes, auf dessen Ergebnisse die Optimierungen basieren. Die `jaotc`-Option `--compile-for-tiered` fügt dem Kompilat Code hinzu, der ein Profiling wie durch den C1-Compiler erzeugten Code des Tier 2 durchführt. Falls der Schwellenwert erreicht wird, wird zur Laufzeit erneut durch C1 mit Tier 3 und dadurch mit dem maximalen Profiling kompiliert, um die spätere Kompilierung durch C2 zu ermöglichen.

Der `jaotc`-Befehl versteht noch eine Reihe weiterer Optionen sowie die Möglichkeit, über eine Textdatei weitere Details der Kompilierung festzulegen. Wir gehen hierauf jedoch nicht weiter ein und

```
jaotc --output helloworld.so  HelloWorld
```

Listing 1

```
java -XX:AOTLibrary=./helloworld.so HelloWorld
```

Listing 2

```
jaotc --output libjavabase.so --module java.base
```

Listing 3

```
java -XX:AOTLibrary=./helloworld.so,./libjavabase.so HelloWorld
```

Listing 4

verweisen den Leser auf einen sehr interessanten Artikel von Ludovic Henry zum Verhältnis von AOT und JIT sowie Details über die Tiered Compilation *[8]*.

Rest in Peace AOT-Kompilierung

Das JEP 410 – *Remove the Experimental AOT and JIT Compiler [7]* hat die Entfernung des AOT- und JIT-Compilers basierend auf Graal zum Ziel. Das JEP formuliert explizit: *„This compiler has seen little use since its introduction and the effort required to maintain it is significant"*. Der Grund ist also schlicht und einfach die geringe Verwendungshäufigkeit und ein sehr hoher Wartungsaufwand. Im Titel dieses Artikels haben wir die JIT-Kompilierung unterschlagen, da diese ja weiterhin durch C1 und C2 vorhanden ist. Der Wegfall der AOT-Kompilierung ist schwerwiegender, sodass wir den Titel entsprechend formulierten.

Nachdem mittlerweile alle bekannten REST-Frameworks (Quarkus, Micronaut, Helidon, Spring) und andere Systeme die Erzeugung nativer Images mithilfe der Graal-VM unterstützen, erscheint es zunächst schwer verständlich, warum die Möglichkeiten der Erzeugung nativen Codes direkt mit dem OpenJDK zu Grabe getragen wird. Ein zweiter Blick macht aber neben dem im JEP 410 genannten hohen Wartungsaufwand das Handling als Grund aus. Wie wir im Beispiel des JDK-Basis-Moduls gesehen haben, übersetzt `jaotc` sämtliche Methoden des Moduls. Bei der GraalVM wird ein transitiver Verwendungs- beziehungsweise Abhängigkeitsgraph erzeugt und nur die tatsächlich verwendeten Klassen, sogar bis auf Methodenebene hinunter, kompiliert. Das erzeugte Kompilat ist damit deutlich kleiner. Außerdem muss nicht der Anwender die zu kompilierenden Klassen angeben, sondern die Frameworks beziehungsweise deren Build-Werkzeuge übernehmen dies. Eine deutliche Vereinfachung und ein ganz offensichtlicher Grund für die geringe Verwendungshäufigkeit der `jaotc`-Kompilierung.

JEP-Prozess

Das JDK Enhancement Proposal (kurz JEP) *[9]* legt einen Prozess zur Weiterentwicklung von Java fest. Ohne das Proposal im Detail ganz gelesen zu haben, scheint uns beim JEP 410 *[7]* nicht alles so gelaufen zu sein, wie man sich das als Außenstehender vorstellt. Das JEP 410 definiert die Entfernung der beiden Compiler für Java 17 als Ziel. Unter der Überschrift *Motivation* liest man dann im JEP 410: *„We have seen little use of these experimental features since they were introduced, and the effort required to maintain and enhance them is significant. These features were not included in the JDK 16 builds published by Oracle, and no one complained"*. Das hört sich unserer Meinung nach etwas willkürlich an nach dem Motto „Wir lassen es mal weg und schauen, ob es jemand merkt" – nicht ganz die feine englische Art. Die Compiler sind allerdings noch im OpenJDK-Repo enthalten und wer sich berufen fühlt, kann sie weiterhin in seinen JDK-Build einbauen, wie es etwa Red Hat praktiziert.

Zusammenfassung

Mit dem JEP 295 wurde mit Java 9 ein Ahead-of-Time-Compiler verfügbar. Mit dem JEP 317 wurde mit Java 10 ein in Java geschriebener Just-in-Time-Compiler verfügbar. Beide Compiler basieren auf Graal, dem Compiler, der auch in der GraalVM verwendet wird und für sie namensgebend war. Während der AOT-Compiler durch ein Programm im `bin`-Verzeichnis des JDK realisiert wird, wird der JIT-Compiler durch eine Option beim Aufruf der JVM aktiviert. Beide Compiler gehören zu den unbekannten Kostbarkeiten und wurden in der Praxis von uns, den Entwicklern, so selten verwendet, dass sie mit dem JEP 410 mit Java 17 wieder aus dem JDK entfernt werden. Da Java 11 als LTS sicher noch einige Jahre Verwendung finden wird, können der AOT- und der JIT-Compiler noch weiter genutzt werden. Das JEP 243, die JVM-Schnittstelle zur Verwendung eines externen JIT-Compilers, wird im JEP 410 explizit von der Löschung ausgenommen, sodass es sein könnte, dass irgendwann in der Zukunft der Graal-Compiler wieder über diese Schnittstelle der JVM zur Verfügung steht.

Referenzen

[1] JEP 295: Ahead-of-Time Compilation, *https://openjdk.java.net/jeps/295.*

[2] Graal Project, *https://openjdk.java.net/projects/graal*

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

[4] JEP 243: Java-Level JVM Compiler Interface, *https://openjdk.java.net/jeps/243*

[5] Bernd Müller, Native Images mit GraalVM, Java aktuell 02/2020

[6] Bernd Müller, Unbekannte Kostbarkeiten des SDK – Heute: Just-in-Time Compilation, Java aktuell 04/2019

[7] JEP 410: Remove the Experimental AOT and JIT Compiler, *https://openjdk.java.net/jeps/410*

[8] Ludovic Henry. AOT Compilation in HotSpot: Introduction, *https://devblogs.microsoft.com/java/aot-compilation-in-hotspot-introduction/*

[9] JEP draft: JEP 2.0, draft 2, *https://cr.openjdk.java.net/~mr/jep/jep-2.0-02.html*

Bernd Müller
Ostfalia
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.

Open-Source-Software – die juristische Sicht

Dr. Jan Bohnstedt, Rechtsanwalt

Der Gedanke von Open Source ist von Freiheit, Ideenaustausch und dem größten Nutzen durch gemeinsame Arbeit geprägt. Dazu passt kein Vorgehen, das an individueller Gewinnmaximierung orientiert ist. Andererseits will die Open-Source-Community auch nicht von gewinnorientierten Unternehmen ausgenutzt werden. Wer also wissentlich oder unwissentlich Open-Source-Software einsetzt, befindet sich mitten in einem dynamischen, von starken Interessen getriebenen Umfeld. Wie immer lohnt es sich, bei aller Offenheit für Neues, auch einen Blick auf die Risiken zu werfen; die ergeben sich auch aus juristischen Aspekten.

Open Source ist nicht nur eine Alternative zu proprietärer Software, es sind spezifische Grundsätze, Werte und Prinzipien, die die Open-Source-Bewegung antreiben. Der Open-Source-Gedanke passt zu einer Unternehmenskultur, die von offener und transparenter Zusammenarbeit und Kommunikation getragen wird, als Chance für Innovationen und Verbesserungen. Schließlich kann Open-Source-Software („OSS") auch eine aktive Rolle bei der Digitalisierung der deutschen Wirtschaft bedeuten. In vielen Softwaresystemen stecken heute zumindest Teile, die unter Open Source fallen, und der Einsatz von solchen Software-Komponenten nimmt weiter zu. Schätzungen gehen davon aus, dass bis zu 69 % aller Unternehmen OSS-Komponenten verwenden *[1]* und sich in nahezu 90 % aller Software OSS-Komponenten befinden *[2]*.

Tabubruch in der OSS-Community?

Der Einsatz von OSS war jahrelang wenig von juristischen Querelen belastet. Die klassischen Softwarehersteller fühlten sich dagegen belästigt, Microsoft bekämpfte das Open-Source-Betriebssystem Linux regelrecht, Gates Nachfolger Steve Ballmer bezeichnete Linux gar als „Krebs" der Branche *[3]*. Die OSS-Entwickler hielten dagegen und bemühten sich, Kunden zu finden, um den Branchengrößen im Wettbewerb Paroli zu bieten. Ein Vorgehen gegen die Nutzer von OSS wäre in dieser Situation kontraproduktiv gewesen.

Diese Zeiten haben sich geändert. Mit dem Erfolg von OSS gibt es nun auch Anteile an diesem Erfolg, die verteilt werden können, sei es in monetärer Hinsicht oder in immateriellen Werten. Entwickler von OSS begannen ernst zu nehmen, was bisher geflissentlich übersehen wurde. Nämlich, dass OSS nur unter bestimmten Lizenzbedingungen genutzt werden darf und dies auch durchgesetzt werden kann. Dies wird mitunter als Tabubruch gesehen, sofern unterstellt wird, es ginge den Entwicklern um eigene finanzielle Interessen. Zum anderen wird aber auch betont, dass der OSS-Gedanke nur weitergetragen werden kann, wenn der dazu notwendige rechtliche Rahmen auch durchgesetzt wird.

Um die Einhaltung von Lizenzbestimmungen besser überwachen und den Nachweis führen zu können, wem ein Urheberrecht an welchen Codeteilen zusteht, wird mehr Wert auf den Nachweis individueller Beiträge gelegt und Copyright-Vermerke nach eingehender Prüfung in den Code eingefügt. Die *gpl-violations.org* hat begonnen, gegen Verletzungen der Open-Source-Lizenzbestimmungen vorzugehen. Ein Core-Team-Mitglied für bestimmte Linux-Komponenten hat dies seit 2013 auf ein neues Niveau gehoben und soll durch Abmahnungen Beträge in siebenstelliger Höhe generiert haben. Deshalb soll er nicht weiter an der Entwicklung von Linux-Komponenten beteiligt worden sein. Mittlerweile werden deutsche Gerichte regelmäßig mit Klagen von OSS-Entwicklern beschäftigt, die gegen Nutzer ihrer OSS-Komponenten mit zum Teil erheblichen finanziellen Forderungen vorgehen.

Dass dies in der Praxis auch oft erfolgreich ist und damit erheblichen Aufwand erfordert, um die bisher verwendete Software weiter nutzen zu dürfen, ist nicht verwunderlich. OSS findet nicht im rechtsfreien Raum statt, sondern ist ebenso wie proprietäre Software vom Urheberrecht geschützt. Das funktioniert auch international sehr gut, denn wenn ein Urheberrecht in einem Staat entstanden ist, wird es in den meisten anderen Staaten wie ein einheimisches Urheberrecht geschützt. Daher ist es völlig unerheblich, wo auf der Welt eine OSS-Komponente entstanden ist – sie wird in Deutschland wie ein deutsches Urheberrecht geschützt. Damit ist die Nutzung von OSS-Komponenten grundsätzlich verboten und muss vom Urheber erlaubt werden. OSS-Komponenten dürfen nach standardisierten Regeln verwendet werden, die in der OSS-Szene entwickelt wurden und vom Urheber frei gewählt werden können. Dazu gehören recht enge Regeln, wie die GPL V2 oder eher freie Regelwerke wie Apache. Aktuelle Regelwerke sind Eclipse Public Licence 2.0, zlib-Lizenz oder die Unlicense. Bei all diesen Lizenzmodellen ist zunächst eine Grobeinteilung vorzunehmen, um festzustellen, ob eine Nutzung der Software überhaupt möglich ist.

Zum rechtlichen Hintergrund

Unter den Begriff Open Source im weitesten Sinne werden verschiedene rechtliche Gestaltungen verstanden. Entscheidend ist immer, für welche Gestaltung der Urheber sich hinsichtlich der Verbreitung seiner OSS entscheidet:

- **Public Domain** („Free Software"): Alles ist erlaubt, Verzicht auf das Urheberrecht, Quellcode wird nicht zwingend offengelegt
- **Freeware:** Proprietäre Software, Nutzung und Weitergabe sind frei, Änderungen sind nicht erlaubt
- **Shareware:** Proprietäre Software, Nutzung und Weitergabe sind frei, Änderungen nicht erlaubt, Nutzung kann eingeschränkt werden, beispielsweise nach Testphase oder gegen Entgelt
- **Open Source:** „Open-Source-Software" bezeichnet jede Software, die einer Lizenz unterliegt, die den Lizenznehmer auffordert, den Quellcode oder den eines Derivats neu zu verteilen, den Quellcode zum Kopieren, Nutzen und Ändern zur Verfügung zu stellen, eine Verbindung mit proprietärem Code untersagt, eine Lizenzgebühr oder keine Lizenzgebühr zulassen oder verlangen kann und/oder die zur Open-Source-Definition (wie von der Open-Source-Initiative veröffentlicht) oder zur Definition für freie Software (wie von der Free Software Foundation veröffentlicht) passt

Unproblematisch ist also nur Software, die als Freeware vertrieben wird. Bei allen anderen können Urheberrechte der Entwickler bestehen. Entscheidend bei der Auseinandersetzung mit einem Anspruchsteller wird sein, ob dieser nachweisen kann, Urheberrechte an der genutzten OSS-Komponente zu haben. Dies hängt stark davon ab, in welcher Weise die Software erstellt wurde. Da die Erstellung von Software in Abkehr vom früher sogenannten Kathedralen-Modell hin zu einem Basar-Modell erfolgt, ist es oft schwierig, die Urhebereigenschaft nachzuweisen, weil bei letzterem die Einzelbeiträge weniger scharf voneinander getrennt werden können und die Einzelleistung weniger stark ausgeprägt ist.

Entscheidend für die Abwehr von Ansprüchen der OSS-Entwickler ist daher, ob ein Miturheberrecht oder Bearbeitungsurheberrecht an konkret bezeichneten Teilen der Software nachgewiesen werden kann.

Gefährlich ist die Miturheberschaft nur, da jeder Miturheber, unabhängig von den anderen, Dritten die Nutzung untersagen kann und das Recht hat, eine strafbewehrte Unterlassungserklärung zu fordern. Wird gegen diese verstoßen, ist die Vertragsstrafe an den individuellen Miturheber zu zahlen. Ohne Vereinbarung einer Vertragsstrafe wird es schwierig, Zahlungen zu fordern, weil nur an alle Miturheber zu zahlen wäre.

Bei einer Bearbeiter-Urheberschaft entsteht ein eigenes Recht des jeweiligen Bearbeiters der OSS-Komponente. Dieser kann die Nutzung seiner Bearbeitung (nicht aber der Originalsoftware) untersagen und auch Schadenersatz fordern.

Es muss ein konkreter Beitrag in Form der Codezeilen, der eigenständig ohne Mitwirkung Dritter erbracht wurde, nachgewiesen werden. Jede Bearbeitung muss eine „eigene geistige Schöpfung" darstellen, um urheberrechtlich geschützt zu sein. Die Bezeichnung als „Maintainer" stellt keine Tätigkeit als Urheber, sondern eher als „Redakteur" dar, der Programmteile anderer auswählt. Nur als „Head of … Development" bezeichnet zu werden, reicht nicht aus. Auch „Core Team Member" ist kein Nachweis eigenständiger geistiger Leistung. Metadaten aus einem Repository reichen nicht als Nachweis aus.

Untersagung der Nutzung der Software

Vor allem in der GPL-V2-Lizenz, aber auch in anderen, weniger oft verwendeten OSS-Lizenzen, wird eine Copyleft-Klausel eingebaut. Das bedeutet, dass mit jedem Verstoß gegen die Lizenzbestimmungen das Recht zur Nutzung der Software erlischt. Es bestehen Bedenken gegen die rechtliche Wirksamkeit dieser Bestimmungen unter deutschem Recht, geklärt ist die Frage jedoch noch nicht. OSS-Lizenzen ohne Copyleft sind also weit weniger riskant für den Auftraggeber von Softwareerstellung oder Käufer einer Software, weil die Nutzung erlaubt bleibt, auch wenn einzelne Bestimmungen der Lizenz nicht erfüllt sind. Wir betrachten im Folgenden nur OSS mit Copyleft-Klauseln.

Grundsätzlich macht es einen Unterschied, ob OSS nur im eigenen Unternehmen genutzt wird oder eigene Software mit solchen OSS-Komponenten auch an Dritte vertrieben wird. Dann drohen auch Ersatzansprüche der Kunden. Beim Einsatz im eigenen Unternehmen ist natürlich ein Vertriebsverbot irrelevant, allerdings kann die Untersagung der Nutzung zum erzwungenen Abschalten wichtiger Software führen.

Ein denkbares Szenario besteht darin, dass ein Softwarehersteller OSS-Komponenten in der eigenen Software einsetzt und beim Kunden installiert oder in die Software des Auftraggebers einfügt, ohne dass der Auftraggeber dies erkennen kann. Werden dabei die Lizenzbestimmungen für die fragliche OSS-Komponente nicht eingehalten, wird die Software vom Kunden unrechtmäßig verwendet. Für den Rechteinhaber an der OSS-Komponente ist es naheliegend, sich an den Softwarehersteller zu wenden, um ihm den weiteren Vertrieb zu untersagen.

Sehr viel effektiver ist es jedoch, die Kunden des Softwareherstellers in Anspruch zu nehmen. Dies erfolgt durch die Urheber von OSS-Komponenten durch eine Abmahnung, die in einer sogenannten „strafbewehrten Unterlassungserklärung" münden soll. Erfolgt dann ein weiterer Verstoß gegen die Lizenzbestimmungen, ist eine Vertragsstrafe zu zahlen.

Wird die strafbewehrte Unterlassungserklärung nicht abgegeben, kann der Urheber der OSS-Komponente eine einstweilige Verfügung bei Gericht beantragen. Die Gerichte können auf diesem Weg auch ohne vorherige Anhörung die Nutzung der Software vorläufig untersagen.

Sobald der Kunde die Software nicht mehr benutzen darf, wird dieser sich an den Softwarehersteller wenden und auf diesen Druck ausüben, die Nutzung der Software sofort wieder zu ermöglichen. Rechtlich wird die Situation genauso behandelt, als ob der Softwarehersteller eine mangelhafte, funktionsuntüchtige Software ausgeliefert habe. Der Kunde kann also nicht nur rekurrierende Zahlungen für die Lizenz oder Wartung einstellen, sondern auch Schadenersatz fordern. Diese gesetzlichen Rechtsbehelfe setzen leider voraus, dass der Softwarenutzer gegenüber dem Softwarehersteller beweisen kann, dass seine Nutzung die Urheberrechte des OSS-Erstellers tatsächlich verletzt. Oder er muss dem OSS-Ersteller beweisen, dass er keine Urheberrechte verletzt. Dazu ist er am wenigsten von allen Beteiligten in der Lage, obwohl er die Hauptlast

des Streits, nämlich die Untersagung der Nutzung, zu tragen hat. Der Beweis der rechtmäßigen Nutzung kann Jahre in Anspruch nehmen und ist deshalb kein praktikabler Weg. Es kommt also vor allem darauf an, Probleme mit OSS-Komponenten zu vermeiden und wenn dies gescheitert ist, das Risiko unter Kontrolle zu behalten.

„Vergiftung" des proprietären Codes

Grundidee der OSS ist die freie Bearbeitung und Weiterverbreitung von Software. Daher darf der bearbeitete OSS-Code nicht unter strengeren Anforderungen als den ursprünglichen Lizenzbestimmungen vertrieben werden. Zu den Pflichten in der GPL V2 gehört zum Beispiel die Pflicht, den Quellcode jedem interessierten Dritten herauszugeben (gegen Erstattung der Kosten für Porto und den Datenträger). Hat der Softwareersteller OSS-Komponenten genutzt und ist dieser Open-Source-Code enger mit eigenem, proprietärem Code verbunden als über bloße Schreib-/Lesezugriffe, kann es sein, dass der proprietäre Code als Bearbeitung des Open-Source-Codes gilt und ebenfalls herausgegeben werden muss. Das bedeutet, dass damit wertvolles Know-how und durch eigenes Urheberrecht geschützte Software eventuell im Quellcode an Wettbewerber abgegeben werden muss.

Keine Haftung für Mängel

Die meisten OSS-Lizenzbestimmungen sehen vor, dass die Haftung des oder der Urheber für die Software ausgeschlossen wird. Ob dieser Haftungsausschluss nach deutschem Recht möglich ist, bleibt unklar, in AGB ist er unmöglich. Ist er aber wirksam, kann aus vertraglicher Haftung kein Schadenersatzanspruch bei Versagen der OSS-Komponente oder für über sie eingeschleppte Malware geltend gemacht werden. Im Rahmen der Herstellerhaftung wird es problematisch, wenn keine Urheber gefunden werden können. In OSS-Projekten bestehen oft sehr komplexe Erstellungsstrukturen. Es können Miturheber-Bearbeitungsrechte entstehen, in denen kaum noch zu identifizieren ist, welcher Mitarbeiter den Code bearbeitet hat und für sein Versagen verantwortlich ist. Das bedeutet, im Vergleich mit klassischer proprietärer Software muss auch das Risiko von Mängeln bewertet werden.

Was ist zu tun?

Bevor eine strafbewehrte Unterlassungserklärung abgegeben wird, sollte juristischer Rat eingeholt werden. Die dafür gesetzte Frist ist meist in Tagen, nicht in Wochen, bemessen.

Für den geschilderten Fall einer einstweiligen Verfügung durch den Urheberrechtsinhaber wird das Gericht oft Vollstreckungsschutz, also die vorübergehende Erlaubnis der Weiternutzung, gewähren. Sollte der Anspruchsteller jedoch tatsächlich Urheber der OSS-Komponenten sein und die Lizenzbestimmungen nicht eingehalten worden sein, besteht ein Schadenersatzanspruch für die Weiternutzung.

Natürlich sind auch Rechtsmittel gegen die einstweilige Verfügung möglich, alle diese Maßnahmen nehmen aber Zeit in Anspruch. Daher ist es sinnvoll, die gesetzlichen Rechtsbehelfe durch vertragliche Regelungen zu verbessern, mit dem Ziel, sofort eine Weiternutzung der Software zu ermöglichen, ohne finanzielles Risiko. Dies wird mit den sogenannten „Rechte-Dritter"-Klauseln erreicht, die den Softwarehersteller verpflichten, mit verschiedenen Maßnahmen die Weiternutzung der Software zu ermöglichen, wenn Dritte (OSS-Urheber) eine Nutzungsuntersagung geltend machen.

Entscheidend für die Effektivität dieser Klauseln ist, dass sie auch dann greifen, wenn eine Rechtsverletzung nur behauptet wird, also noch nicht vom Gericht rechtskräftig festgestellt wurde. Es geht ja gerade um diesen Zeitraum, in dem die Rechtslage noch offen ist.

Vorbeugung ist der beste Schutz

Lizenznehmer sollten auf einen offenen Umgang mit Open Source dringen. Für Software, die bereits in Nutzung ist, kann ein Code-Audit helfen. Die Wahrscheinlichkeit, fündig zu werden, sollte bei über 90 % liegen. Werden alle für diese Komponenten einschlägigen Lizenzbestimmungen eingehalten, besteht kein Problem. Ist dies aber nicht möglich, weil zum Beispiel der proprietäre Code mit dem OSS-Code eng verbunden („vergiftet") ist, sollte versucht werden, mit dem Lizenzgeber eine wirtschaftlich sinnvolle Lösung zu vereinbaren. Ob das gelingt und wie diese Lösung aussieht, hängt davon ab, ob bereits im Lizenzvertrag Vorsorge getroffen wurde. Das sollte immer durch Open-Source-Klauseln erfolgen. In diesen verpflichtet sich der Lizenzgeber offenzulegen, ob er OSS-Komponenten verwendet und wenn ja, welche Bedingungen für deren Nutzung zu erfüllen sind.

Quellen

[1] Bitcom Open-Source-Monitor 2019, *https://www.bitkom.org/opensourcemonitor2019*

[2] Arun Batchu, Thomas Klinect, Anne Thomas, Hype Cycle for Open-Source Software, Gartner, 2020.

[3] Handelsblatt 26.02.2021: „Open Source: Wer Code teilt, wird Zukunft ernten"

Dr. Jan Bohnstedt
Rechtsanwalt
mail@rajb.de

Dr. Jan Bohnstedt ist seit über 20 Jahren mit Fragestellungen an der Schnittstelle von Recht und Technik befasst. Ein Schwerpunkt ist das IT- und Datenschutzrecht. Nach Tätigkeit in großen Rechtsanwaltskanzleien ist er in eigener Kanzlei und als Jurist eines Start-ups im IT-Bereich tätig. Seit 2019 hat ihn das Handelsblatt in die Liste der „Best Lawyers" für IT-Recht in Deutschland aufgenommen. Er engagiert sich ehrenamtlich im DOAG Legal Council.

Open Source, ein Erfahrungsbericht: In kleinen Schritten von den ersten Contributions zum Maintainer

Georg Berky und Sandra Parsick

Die Open-Source-Welt erscheint oft riesig und ohne Erfahrungen gesammelt zu haben, ist es für Neulinge nicht offensichtlich, wie sie mit Open-Source-Beiträgen anfangen können. Mit Git als Werkzeug und GitHub als Plattform ist es für Einsteiger um einiges einfacher geworden. Obwohl es für viele immer noch schwierig erscheinen mag, war unsere Erfahrung, dass es leichter ist, als es am Anfang erscheint. Es gibt inzwischen viele Projekte, bei denen auch Neulinge aushelfen und wertvolle Beiträge leisten können.

Georg: Als ich meinen ersten Job in Vollzeit in der IT anfing, war das Erste, das mir auffiel, die schiere Menge an Bibliotheken, die es im Vergleich zu den Projekten gab, die ich aus Studium, Nebenjobs und vom Lehrstuhl kannte. Die meisten davon waren Open-Source-Bibliotheken und als Neuling kam es mir so vor, als säßen dort Leute mit jahrzehntelanger Erfahrung, die Software für alle Softwareentwickler da draußen schrieben – und diese Software war so gut, dass eine Menge Projekte sie nutzen. Ich wusste so gut wie nichts über Open Source, die Ökosysteme, die Community und wie man selbst dazu beitragen kann. Es war damals zugegebenermaßen auch noch etwas schwerer.

Sandra: In meiner ersten Firma, bei der ich nach dem Studium 2008 anfing, wurde zu 100 Prozent auf Open Source gesetzt. Zwar existierte schon GitHub, aber es war nicht so verbreitet wie heute, sodass die meisten Projekte entweder eigene Infrastruktur betrieben oder auf Dienste wie Sourceforge *[1]* setzten. Das bedeutet für jemanden, der etwas beisteuern wollte, Patches zu erstellen und sie über Tickets einzureichen.

Wir wollen euch in diesem Artikel zeigen, wie es bei uns gelaufen ist, was uns überrascht, aber auch motiviert hat, damit weiterzumachen, und wie wir inzwischen Maintainer unserer eigenen kleinen Projekte geworden sind. Begleitet uns auf einer kleinen Zeitreise durch unsere Open Source Contributions!

2011 – 2017

Sandra: Bewusst mit Open-Source-Projekten arbeite ich seit 2008, aber ich habe erst 2011 mit den ersten Versuchen, auch was zu den Projekten beizutragen, angefangen. Ich meldete damals vor allem Bugs, die mir in der täglichen Arbeit auffielen, oder stellte Fragen auf Mailing-Listen, wenn etwas unklar war.

Bei dem einen oder anderen Bug konnte ich auch einen Patch für einen Bugfix beisteuern oder die Dokumentation verbessern. Das war zum Teil recht mühselig, da nicht alle Projekte schon auf GitHub waren und deshalb mit Patch-Dateien gearbeitet werden musste *[2]*. Wer das noch nie gemacht hat: Ein Patch ist eine Datei, die das enthält, was sonst in einem Git-Commit steht, also die Änderungen, die auf den Dateien im Repository ausgeführt werden sollen. Die Patches hat man damals per E-Mail verschickt oder sie an den Bug-Report im Projekt angehängt, damit der Maintainer die Änderungen von Hand in den Code übernehmen konnte.

Wenn Projekte schon auf GitHub waren, dann kamen je nach Projekt noch Formalitäten in Form von Contributor License Agreements *[3]* hinzu. Contributor License Agreements sind für Organisationen wie Apache Foundation, Eclipse Foundation oder für Projekte wie Spring Framework, bei denen Firmen dahinterstehen, wichtig, um Lizenz-Streitigkeiten zu vermeiden.

Später habe ich dann meinen eigenen Blog *[4]* angefangen, um Lösungen zu meinen alltäglichen Problemen zu dokumentieren. Im Blog habe ich so immer mehr Material gesammelt und fing irgendwann an, Vorträge über Open-Source-Projekte zu halten sowie darüber, wie diese meinen Entwickler-Alltag vereinfachten.

2018

Im Jahr 2018 waren wir beide im selben Projekt und uns fiel beim Bearbeiten der aktuellen Story auf, dass uns ein Feature in AssertJ *[5]* fehlte. Wir mussten sicherstellen, dass ein String eine bestimmte Länge hatte, und AssertJ hatte ein solches Feature noch nicht *[6]*.

Wie der Zufall es wollte, hatten wir als Co-Organisatoren der Softwerkskammer Ruhr *[7]* ohnehin einen Hackergarten *[8]* geplant. Ein Hackergarten ist ein eintägiges Event, bei dem man sich in ungezwungener Runde trifft, um gemeinsam in kleinen Gruppen Contributions zu Open-Source-Projekten zu machen. Häufig sind auch Maintainer von Projekten anwesend, die vorstellen, was die Projekte tun und wobei sie dringend Hilfe gebrauchen könnten. Auch bei unserem Hackergarten in Dortmund waren einige Maintainer dabei, so auch Karl Heinz Marbaise für das bekannte Build-Tool Maven von Apache, der weltweit größten Open Source Foundation. Während des Hackergartens gab es auch einige Contributions der Teilnehmer zu diesem Projekt.

Wir erstellten beim Hackergarten also ein Issue im GitHub-Projekt von AssertJ, das das fehlende Feature beschrieb, und machten uns im Laufe des Tages an die Arbeit. Der Pull Request (PR) dazu war recht schnell erstellt, aber das war erst der Anfang.

Joel Costigliola, der Maintainer von AssertJ und inzwischen Java-Champion, hatte noch einiges an Feedback für uns. Die Dokumentation passte an einer Stelle noch nicht. Wir hatten den Javadoc einer anderen Methode übernommen und nicht angepasst. An einer anderen Stelle schlug Joel ein besseres Beispiel für die Benutzung des Codes vor. Die Fehlermeldung einer fehlgeschlagenen Assertion war noch nicht klar genug. Joel lobte aber auch gute Beispiele und sprechende Parameternamen, die wir im Code verwendet hatten. Nach mehreren Feedbackrunden und Verbesserungen akzeptierte er unseren Pull Request und wir waren glücklich, unseren ersten PR in nur einem Tag Arbeit fertig gestellt zu haben. Als Bonus obendrauf konnten wir den Code am Montag darauf auch schon im Projekt nutzen.

Die Betreuung von Joel war super. Wir wussten immer, in welche Richtung es mit unserem Feature gehen sollte, und er konnte gut vermitteln, was wir noch verbessern mussten. Sein Feedback zu Stil und Dokumentation war ebenfalls sehr gut. Das motivierte uns, zwei weitere Feature Requests zu erstellen. Einer wurde aus guten Gründen abgelehnt. Für den zweiten erstellten wir einen weiteren Pull Request, der akzeptiert wurde. Seitdem kann man in AssertJ prüfen, dass die Fehlermeldung einer Exception einem regulären Ausdruck entspricht oder einen bestimmten String nicht enthält.

Etwas später stießen wir auf ein offenes Problem im Projekt Commons-Lang von Apache. Das Feature konnten wir auch gut im Projekt gebrauchen, also dachten wir uns: Was bei AssertJ gut lief, könnte auch bei Commons-Lang funktionieren. Wir erstellten wieder einen Pull Request, aber – und auch damit muss man rechnen – der Pull Request wurde nicht gemergt, weil jemand anderes das Ticket gleichzeitig bearbeitet hatte.

Kurz nach dem Hackergarten fand die Aktion „Hacktoberfest" *[9]* im Oktober statt. Sandra machte mit und „contributete" vermehrt Dokumentationen zu verschiedenen Open-Source-Projekten.

Hacktoberfest ist eine Initiative von Digital Ocean, die seit 2013 Open-Source-Projekten helfen möchte, die Beteiligung an ebendiesen zu erhöhen. Dabei wird im Monat Oktober dazu aufgerufen,

mindestens vier Pull Request für ein beliebiges Projekt auf GitHub zu erstellen. Leider führte das Event in den letzten Jahren vermehrt zu Spam-Pull-Request, da die Teilnehmer dadurch ein T-Shirt ergattern konnten. Digital Ocean versucht dagegenzusteuern, sodass sich Open-Source-Projekte seit 2020 aktiv beim Event anmelden müssen, damit die Pull Requests für das T-Shirt gewertet werden.

August 2018 bis Januar 2019

Auch Vorträge und Workshops über Open-Source-Tools und -Bibliotheken sind Open-Source-Arbeit. Sie bringen neue Ideen in die Community, zeigen, welche Ansätze schon zum Erfolg oder Misserfolg geführt haben, und fördern den Austausch unter den Projekten. Ohne die Vorträge und Verbindungen zu anderen Mitgliedern unserer Communities hätten wir das Rad schon in mehreren Fällen zweimal erfunden, weil wir die passende Bibliothek für unser Problem noch nicht kannten.

Georg: In der Zwischenzeit hatte ich Groovy als Skriptingsprache für mich entdeckt. Es fing mit einigen Lightning Talks, die ich bei Treffen der Softwerkskammer hielt, an und entwickelte sich weiter zu Talks auf nationalen und internationalen Konferenzen.

Groovy *[10]* ist eine dynamische Programmiersprache auf der JVM, kann aber auch direkt als Interpreter für Shellskripte benutzt werden, als Alternative für *#!/bin/bash* am Skriptanfang. Verbunden mit Grape als Dependency-Manager steht dann das gesamte Java- und Groovy-Ökosystem an Bibliotheken für die eigenen Skripte zur Verfügung. Details dazu findet ihr in der Java aktuell 05/2019 *[11]* oder auf meinem Blog *[12]*.

Den ersten zehnminütigen Talk dazu gab ich beim Jahresabschlusstreffen der Softwerkskammer im Ruhrgebiet während einer Lean-Coffee-Session, bei der alle Teilnehmer ihr Lieblingstool kurz vorstellen konnte. Groovy fand guten Anklang, weil es voll syntaxkompatibel zu Java ist und viele Java-Entwickler sich freuten, nicht mehr in der für sie manchmal umständlichen Schreibweise von Bash entwickeln zu müssen. Groovy unterstützte auch schon vor Java 8 funktionale Ansätze, sodass es für viele eine Gelegenheit war, diese Konzepte auszuprobieren, als viele Projekte noch auf Java 7 liefen.

2020 und Corona: Dependency-Update-Plug-in während Corona

Wir wollten wieder mal zusammen hacken und konnten uns in den User Groups nicht treffen, weil sich gerade alle im Lockdown befanden. Sandra entdeckte bei der Recherche für einen Workshop das Dependency-Update-Plug-in, das von Oliver Weiler *[13]* ins Leben gerufen wurde, und wollte es vorantreiben, da sie die Idee super fand. Sie schlug vor, daran etwas zu machen. Georg hätte ein solches Plug-in auch schon öfter im Projekt gebrauchen können und war begeistert. Ein netter Nebeneffekt für beide: Wir konnten uns mit Kotlin auseinandersetzen, worin das Projekt geschrieben ist.

Das Plug-in ist die „Enterprise"-Version von Dependabot *[14]*, das man in geschlossenen Projekten leider oft aufgrund von Regularien nicht einsetzen kann. Es erstellt pro Maven Dependency einen Branch, in dem nur die Version der Dependency aktualisiert wird, und pusht die Branches automatisch, damit die CI durch Builds prüfen kann, ob ein Dependency Update den Build bricht.

Als wir anfingen, verwendete das Projekt JGit, eine Git-Implementierung in Java, für alle Operationen auf dem Repository. Den ersten Beitrag wollten wir dadurch leisten, dass wir nicht nur JGit, sondern auch das auf der Maschine des Benutzers installierte Git unterstützen, wo Zugangsdaten, SSL-Schlüssel, Author und Ähnliches meistens schon vorkonfiguriert sind und dadurch nicht nochmal in die POM des Projekts geschrieben oder beim Plug-in-Aufruf übergeben werden müssen. Das Plug-in sollte also einen JGit Provider und einen Native Git Provider bekommen, den man über die POM konfigurieren kann.

Die Arbeit an diesem Feature zog sich über das ganze Jahr 2020 hin und brachte Abwechslung und Freude über den Fortschritt in den von Homeoffice und Pandemie geprägten Alltag. Wir waren beide froh, dass wir uns wenigstens virtuell treffen konnten und im Laufe des Jahres nahm der Provider immer mehr Gestalt an, bis wir an den Punkt kamen, an dem wir auch den Plug-in-Aufruf mit Maven und einer echten POM testen mussten, um die verschiedenen Konfigurationsszenarien durchzuspielen.

Wie schreibt man einen Integrationstest für ein Maven-Plug-in? Sollen wir Maven über einen Process Builder aufrufen und den Output parsen? Instabil, weil alles kaputt geht, sobald sich eine Formulierung der Logmeldung ändert. Rückgabecodes? Dazu fanden wir keine Doku – und sind die auf Windows, Linux und Mac die Gleichen? Wir schienen in eine Sackgasse gelaufen zu sein.

Karl Heinz to the Rescue: Die Maven IT Extension

Zum Glück erinnerten wir uns zurück an den Hackergarten, bei dem Karl Heinz Marbaise *[15]* das Maven-Projekt vorgestellt hatte. Maven selbst bietet ein riesiges Ökosystem an Plug-ins und ein Großteil der Arbeit des Build Tools wird von Plug-ins erledigt. Es war also kein Wunder, dass dort bereits jemand ein Tool für Plug-in-Tests entwickelt hatte, aber unser Glück war, dass es Karl Heinz selbst war. Seine Maven IT Extension *[16]* war wie gemacht für unseren Use Case.

Wir hatten sogar doppeltes Glück, denn Karl Heinz hatte sofort Lust, uns beim Dependency-Update-Plug-in zu helfen. Jetzt waren wir zu dritt und machten uns daran, das Plug-in mithilfe von Karl Heinz zu testen. Dadurch fanden wir weitere Use Cases für Karl Heinz' Plug-in und beschlossen, einen Abstecher in dessen Projekt zu machen, um später mit weniger Aufwand und höherer Lesbarkeit unsere Tests schreiben zu können.

Die Maven IT Extension ist eine Extension für JUnit 5 und stellt dem Benutzer einen Rahmen zur Verfügung, Maven-Projekte mit POM und weiteren Dateien als Ressourcen ins Projekt zu legen, Maven Goals auf dem Projekt auszuführen und die Ergebnisse von Setup und Plug-in-Lauf programmatisch mit Java zu untersuchen. Für jede Testmethode wird das Projekt in ein eigenes Zielverzeichnis kopiert, bevor Maven darauf aufgerufen wird. So bleiben die Tests isoliert und auch parallel ausführbar.

Wir erweiterten zusammen die Extension um die Möglichkeit, einfacher auf das Zielverzeichnis zuzugreifen, das für das Dependency-Update-Plug-in zum Git Repo werden musste. Wir wollten prüfen, ob die richtigen Branches angelegt wurden.

Durch die Zusammenarbeit mit Karl Heinz lernten wir noch einiges über die Interna von Maven, konnten die Tests für unser Plug-in verbessern und außerdem als echte Benutzer Feedback für die Maven IT Extension geben. Beide Projekte profitierten von der Zusammenarbeit und außerdem hatten wir viel Spaß beim wöchentlichen Entwickeln im Lockdown.

2021

Insgesamt ein Jahr später waren wir dann mit dem Native Git Provider und allen Tests fertig und bereit für den finalen Pull Request. Was dann kam, überraschte uns jedoch: Oliver bat uns, das Projekt komplett zu übernehmen, weil er weniger Zeit dafür hatte als wir. Wir mussten erst mal über das Angebot nachdenken, sagten dann aber zu und zogen das Projekt in Georgs GitHub-Account *[17]* um.

Wir waren also plötzlich Maintainer eines Open-Source-Projekts und dadurch kamen neue Aufgaben auf uns zu. Wir mussten einen automatisierten Build einrichten, Code Coverage messen, veraltete Dependencies aufspüren und die Lieferung nach Maven Central organisieren. Maven Central, unser Projekt landet in Maven Central… Das fühlte sich alles surreal an, aber zum Glück hatten wir wieder Hilfe von Karl Heinz, der das als Open-Source-Veteran schon öfter gemacht hatte.

Georg brauchte als Inhaber des Projekts in GitHub einen Account bei Sonatype. Für diesen wird eine Maven GroupId *io.github.georgberky* freigeschaltet, indem man seinen GitHub-Account verifiziert. Ab dann kann man GnuPG-signierte Artefakte mit dieser GroupId und den Sonatype Credentials ins Staging Repository pushen. Dort laufen noch ein paar Tests, etwa, ob unsichere Dependencies vorhanden sind. Wenn alle Tests bestanden sind, wird das Artefakt nach Maven Central befördert.

Als Co-Maintainer bekamen Karl Heinz und Sandra noch Push-Rechte auf die Unter-GroupId für das Maven-Plug-in *io.github.georgberky.maven.plugins.depsupdate*. Damit können sie das Plug-in pushen, aber nicht alles unter Georgs GroupId veröffentlichen.

Ende Mai schlossen wir die Arbeiten am Native Git Provider im Rahmen des „hack-commit-push"-Events *[18]* ab, was uns ermöglichte, die Bekanntheit des Plug-ins und der Extension zu vergrößern. Gegen Ende des eintägigen Open-Source-Events hatten wir unser erstes Release fertig und veröffentlichten es auf Maven Central *(siehe Abbildung 1)*.

Oliver hatte parallel ein Relocation-Release *[19]* gemacht, um den Benutzern zu zeigen, dass die nächsten Artefakte unter einer neuen GroupId zu finden sind.

Sandra: Am Anfang des Jahres half ich bei einer Java-8- auf Java-11-Migration in einem Eclipse-RCP-Projekt. Unter anderem wird das P2-Maven-Plug-in *[20]* eingesetzt, um Maven-Artefakte in ein P2-Repository bereitzustellen. P2 ist ein Repository-Format aus der Eclipse-RCP-Welt, um Dependencies in RCP-Anwendungen bereitzustellen. Bei der Java-11-Umstellung stellte sich heraus, dass das P2-Maven-Plug-in mit einigen JAR-Features nicht zurechtkam, die nach Java 8 dazu kamen, wie etwa Multi-Release JARs. Dazu kam, dass das P2-Maven-Plug-in-Projekt den Eindruck machte, dass es nicht mehr aktiv weiterentwickelt wurde. Da die Stellen, die ein Update benötigten, schnell im Code zu identifizieren waren, schrieb ich den Maintainer an und fragte nach, wie es um die Zukunft des Plug-ins aussah und wie man am besten im Projekt unterstützen könnte, da ich einfach einen Fork zu machen als letzte Möglichkeit ansah.

Ein Fork hätte mehrere Nachteile nach sich gezogen. Einmal hätte es das Update bei den Benutzern erschwert. Zum einen hätten sie über andere Kanäle mitbekommen müssen, dass ein neuer aktiver Fork existiert, zum anderen müsste der Fork unter neuen Maven-Koordinaten veröffentlicht werden.

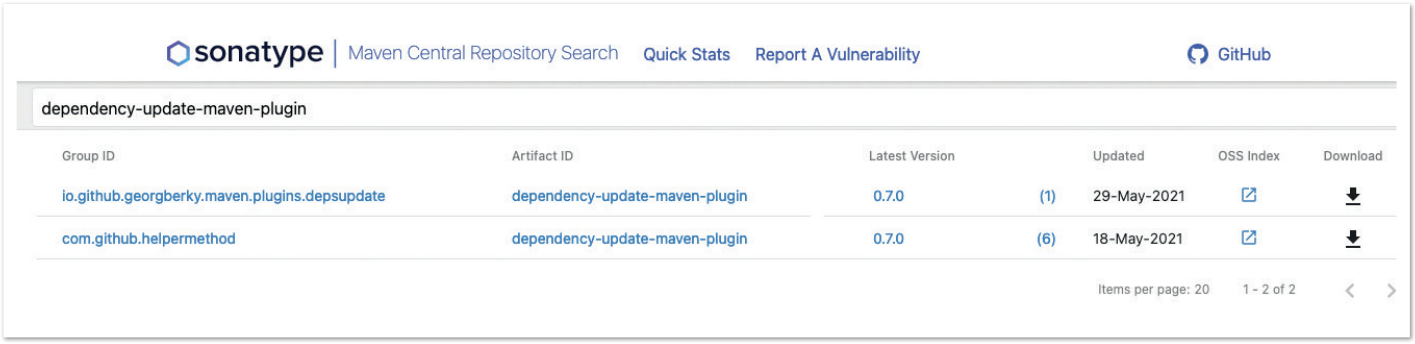


Abbildung 1: Dependency-Update-Maven-Plug-in – erstes Release in Maven Central

Der nächste Nachteil wäre die Trennung der Organisation der beiden Projekte gewesen. Beispielsweise wäre es schwer zu entscheiden, wo zukünftig Benutzer Bugreports melden sollen. Tomek Bujok *[21]* war erfreut, dass das Plug-in noch Verwendung fand, hat aber selbst wenig Zeit, sich weiter darum zu kümmern. Er bot mir daher an, gleich als Maintainer mitzumachen, und gab mir dafür die nötigen Rechte. Schnell stellte sich heraus, dass im Projekt erst mal Wartungsthemen (CI aufbauen, Abhängigkeiten aktualisieren, QA Checks einbauen etc.) bearbeitet werden mussten. Zum Glück hatte Benjamin Marwell *[22]*, ebenfalls ein Maven Committer, das zufällig mitbekommen und half mir mit den Wartungsthemen, sodass ich recht schnell auch die benötigten Fixes einbauen und releasen konnte. Dass im Repository wieder mehr Bewegung reinkam, bewegte auch wieder andere Contributor dazu, etwas beizutragen, sodass in kurzer Zeit ein weiteres Release entstand.

Unser Fazit

Open-Source-Arbeit hat viele Facetten. Es geht dabei nicht nur um die klassischen Code-Beiträge, sondern man kann auch durch Vorträge, Dokumentation, Hilfe in Foren und bei Treffen von User Groups der großen Community helfen. Gleichzeitig sind wir auch immer auf Leute getroffen, die uns geholfen haben. Von Karl Heinz und Oliver angefangen, über unsere Co-Organisatoren bei den User Groups bis hin zu den Organisatoren der Open-Source-Events, in deren Rahmen wir unsere Projekte weiterentwickelt haben. Ohne die Community im Rücken wäre es so viel schwerer gewesen. Darin liegt für uns auch der größte Reiz neben dem Schreiben von Code: Man ist mit einer großen Community verbunden, die gerne ihr Wissen teilt und hilft, wenn es Probleme gibt.

Bei den Hackergarten-Events haben wir gelernt, dass die Einstiegshürde in viele Projekte niedriger ist, als man denkt. Wir hatten ohne große Vorkenntnisse innerhalb weniger Stunden den ersten Beitrag zusammen. Danach mussten wir noch nacharbeiten, aber auch das ging schneller, als wir erwartet hatten, und das Gelernte half uns beim nächsten Beitrag. Viele Projekte freuen sich über Beiträge, die die Dokumentation verbessern oder Fehler melden.

Müssten wir den besten Zeitpunkt für einen Einstieg in Open-Source-Beiträge wählen, wäre es wohl das Hackergarten-Event kombiniert mit Pair-Programming. Dort bekommt man das Projekt vorgestellt und kann direkt mit einem der Maintainer sprechen, wenn es Fragen gibt. Außerdem ist man unter Gleichgesinnten und bekommt vielleicht sogar Inspiration und Hilfe aus anderen Projekten.

Referenzen

[1] *https://sourceforge.net/*
[2] Open Source Contribution via Patch-Dateien: *https://sourceforge.net/p/hsqldb/bugs/1186/*
[3] *https://de.wikipedia.org/wiki/Contributor_License_Agreement*
[4] *https://blog.sandra-parsick.de*
[5] *https://github.com/assertj*
[6] *https://github.com/assertj/assertj-core/issues/1319*
[7] *http://softwerkskammer.ruhr*
[8] *https://www.hackergarten.net/*
[9] *https://hacktoberfest.digitalocean.com*
[10] *http://www.groovy-lang.org/*
[11] *https://georg.berky.dev/publikationen/making-shell-scripts-groovy/*
[12] *https://georg.berky.dev*
[13] *https://github.com/helpermethod*
[14] *https://dependabot.com/*
[15] *https://www.soebes.de/*
[16] *https://github.com/khmarbaise/maven-it-extension*
[17] *https://github.com/georgberky/dependency-update-maven-plugin*
[18] *https://hack-commit-pu.sh/*
[19] *https://maven.apache.org/guides/mini/guide-relocation.html*
[20] *https://github.com/reficio/p2-maven-plugin*
[21] *https://twitter.com/tombujok*
[22] *https://twitter.com/bmarwell*

Georg Berky
hallo@berky.dev

Georgs Handwerk und Leidenschaft ist die Programmierung, meistens in JVM-Sprachen wie Java, Groovy, Kotlin oder Clojure. Dazu gehören für ihn auch Themen wie die Pflege von Legacy Code, Automatisierung von Builds und Deployments oder Agilität im Team. Seit einigen Jahren ist er Co-Organisator der Software-Craftsmanship-Communities im Ruhrgebiet und in Düsseldorf. Wenn er mal nicht programmiert, spielt er Trompete, pflegt seine Bonsai oder praktiziert Aikido.

Sandra Parsick
mail@sandra-parsick.de

Sandra Parsick ist freiberufliche Softwareentwicklerin im Java-Umfeld. Seit 2008 beschäftigt sie sich mit agiler Softwareentwicklung in verschiedenen Rollen. Ihre Schwerpunkte liegen im Bereich der Java-Enterprise-Anwendungen, agilen Methoden, Software Craftsmanship und in der Automatisierung von Softwareentwicklungsprozessen. Darüber schreibt sie gerne Artikel und spricht auf Konferenzen. In ihrer Freizeit engagiert sie sich in der Softwerkskammer Ruhrgebiet, einer Regionalgruppe der Software Craftmanship Community im deutschsprachigen Raum. Außerdem ist sie Java-Champion und Mitglied im Oracle-Groundbreaker-Ambassador-Programm.

Open Source Application Performance Monitoring (APM) Tools für Java-basierte Unternehmensanwendungen

Dr. Andreas Brunnert, RETIT GmbH

Die Anzahl der frei verfügbaren Application Performance Monitoring (APM) Tools für Java-basierte Unternehmensanwendungen (UA) hat in den letzten Jahren stark zugenommen. Das Tracing von Transaktionen in verteilten UA wird daher immer mehr zu einer Funktion, für die man nicht mehr zahlen muss. Selbst große kommerzielle APM-Hersteller unterstützen mittlerweile Standards wie OpenTracing, OpenCensus oder OpenTelemetry, um die Integration von eigenen Messpunkten zu vereinfachen. Dieser Artikel stellt die wichtigsten Open Source APM Tools (unter anderem Zipkin, Jaeger, PinPoint, Apache Skywalking) und Standards (OpenTracing, OpenCensus, OpenTelemetry) für UA vor und gibt einen Überblick über die jeweiligen Stärken und Schwächen. Abschließend wird erläutert, wie sich die kommerziellen APM Tools von diesen frei verfügbaren Tools abgrenzen.

Das Monitoring von Anwendungen ist ein Thema, das in den letzten Jahren durch die Integration von Betrieb und Entwicklung in DevOps-Teams immer mehr an Bedeutung gewonnen hat, um den Mitarbeitern schnell Informationen über den Anwendungszustand zur Verfügung zu stellen und schnell handeln zu können. In vielen Quellen wird dieses Thema auch als Observability bezeichnet, was meist durch die Erhebung von Traces, Metriken und Logs realisiert wird *[1]*.

Während Observability versucht, alle Aspekte des Anwendungszustands zu beschreiben (funktionale und nicht-funktionale Korrektheit), bezieht sich der Begriff Application Performance Monitoring (APM) primär auf das korrekte, nicht-funktionale Verhalten einer Anwendung in Bezug auf die Performance (im Sinne von Antwortzeit, Durchsatz und Ressourcenauslastung). Daher wird im APM-Bereich der Fokus vor allem auf die Erhebung von Traces und Metriken gesetzt, da Logs in erster Linie den Zweck haben, fachliche Probleme besser analysieren zu können. In diesem Artikel schränken wir zusätzlich ein, dass wir nur APM-Werkzeuge betrachten, die es ermöglichen, Traces zu erheben, da dies eine Fähigkeit ist, die es erst seit einigen Jahren in der Open-Source-Welt gibt. Metriken können schon seit vielen Jahren mit diversen Tools oder mit in der

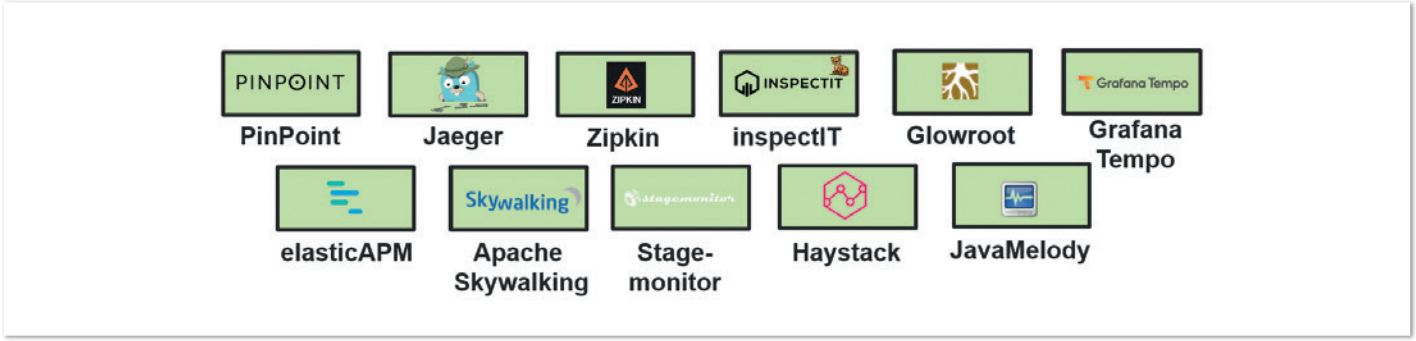


Abbildung 1: Übersicht über die Open-Source-APM-Werkzeuge (Quelle: eigene Abbildung, Icons von den jeweiligen Webseiten der Werkzeuge: [4] [7] [9] [11] [8] [3] [13] [6] [12] [2] [15] © RETIT GmbH)

JVM integrierten Bordmitteln (wie JMX) erhoben werden. Einige der vorgestellten Werkzeuge sind in der Lage, sowohl Traces als auch Metriken zu erheben, aber nicht alle.

Der Artikel gliedert sich in fünf wesentliche Abschnitte. Zunächst wird im ersten Teil motiviert, wie es zum Wachstum bei der Anzahl von Open Source APM Tools kam, und ein Überblick gegeben, welche Werkzeuge es gibt. Im zweiten Abschnitt wird thematisiert, in welchem Kontext die vorgestellten Werkzeuge bewertet werden. Darauf folgt eine Auflistung der wesentlichen Standards im Open-Source-APM-Bereich. Im Anschluss daran behandelt der vierte Abschnitt die relevantesten Open-Source-APM-Lösungen, bevor wir zum Abschluss einige Gründe durchgehen, wieso es trotz verfügbarer Open-Source-Lösungen Sinn ergeben kann, eine kommerzielle Lösung einzusetzen.

Motivation

Abbildung 1 gibt eine Übersicht über die wesentlichen Open-Source-APM-Werkzeuge, die in diesem Artikel betrachtet werden und in den letzten Jahren veröffentlicht wurden. Die bekanntesten Vertreter sind wahrscheinlich Zipkin *[2]*, Jaeger *[3]* und Apache Skywalking *[4]*. Zipkin ist ein APM-Werkzeug von Twitter, das die Konzepte umsetzt, die in dem Tracing-Werkzeug „Dapper" bei Google implementiert und 2010 auch in einem Paper veröffentlicht wurden *[5]*. Jaeger wiederum ist ein Fork von Zipkin, der von Uber Engineering als Referenzimplementierung eines OpenTracing-Backends entwickelt wurde, eines Standards, auf den wir später im Artikel genauer eingehen werden. Jaeger und Zipkin sind Tools, die sich nicht ausschließlich auf Java fokussieren, sondern es unabhängig von der eingesetzten Technologie ermöglichen, Traces zu speichern und zu visualisieren. Apache Skywalking wiederum unterstützt sowohl die Erhebung und Analyse von Traces als auch von Metriken. Ähnlich positioniert sich PinPoint *[6]*, das sich jedoch im Gegensatz zu den bisher genannten Werkzeugen primär auf Java fokussiert (teilweise wird PHP unterstützt). Auch elasticAPM *[7]* unterstützt Metriken und Traces und stellt eine Ergänzung des Elastic-Stacks dar, die durch den Aufkauf eines Startups erworben wurde.

InspectIT *[8]* war ursprünglich ein vollständiges APM-Werkzeug für Java-Umgebungen, das von einem Team bei Novatec in Deutschland entwickelt wurde. Durch die Verfügbarkeit unterschiedlicher Tracer entschied sich das Team jedoch im Jahr 2019, sich vor allem auf ihren eigenen Java-Agenten mit dem Namen InspectIT Ocelot zu fokussieren, der auf Basis des OpenCensus-Standards Daten erhebt. Ähnlich wie InspectIT positioniert sich Glowroot *[9]* als vollständiges APM-Werkzeug, das neben Traces auch automatisiert bestimmte JVM-Metriken erhebt, wie Heap-Auslastung oder Garbage-Collection-Statistiken. Jedoch ist bei Glowroot zu beachten, dass einige der Kernentwickler sich mittlerweile hauptsächlich beim OpenTelemetry-Projekt *[10]* engagieren, was die Aktivität des Projektes im letzten Jahr (2020), zumindest laut GitHub, etwas reduziert hat. Haystack *[11]* ist ein APM-Werkzeug des Expedia-Engineering-Teams, das nicht nur Java-basierte Anwendungen unterstützt, sondern auch viele andere Technologien. Stagemonitor *[12]* ist primär für den Einsatz während der Entwicklung gedacht und unterstützt Entwickler unter anderem dabei, frühzeitig zu erkennen, wo eventuelle Bottlenecks in der Requestverarbeitung auftreten können. Ähnlich wie bei Glowroot hat die GitHub-Aktivität des Projekts in der letzten Zeit nachgelassen, da einer der Hauptentwickler mittlerweile Teil des Elastic-APM-Teams ist. JavaMelody *[13]* ist der älteste Vertreter der aufgeführten Werkzeuge und wurde in erster Linie als Metrik-Monitoringwerkzeug entwickelt, unterstützt aber auch rudimentäre Tracing-Fähigkeiten, weshalb es der Vollständigkeit halber mit aufgenommen wurde.

Wie bereits eingangs erwähnt, liegt der Fokus dieses Artikels auf dem Tracing von Anwendungstransaktionen, daher wurden reine Metrikwerkzeuge wie Prometheus *[14]* nicht aufgeführt. Jedoch ist in der Liste Grafana aufgenommen worden, das oft zur Visualisierung von Metriken aus Prometheus verwendet wird und jetzt auch Traces unterstützt. Hierzu hat das Grafana-Team ein neues Tracing-Backend namens Grafana Tempo *[15]* entwickelt, das es Grafana erlaubt, neben Metriken auch Traces zu verwalten.

Die Gründe für die Entwicklung dieser großen Menge an Werkzeugen ist vielfältig, an dieser Stelle werden daher nur drei wesentliche Treiber näher erläutert.

Microservices: Die Teilung von Softwaresystemen in Microservice-Komponenten führt dazu, dass es erforderlich ist, die Interaktionen zwischen diesen zu überwachen, was durch Distributed Tracing möglich wird. Dies ist insbesondere notwendig, da Microservices auf eine Art und Weise miteinander interagieren können, die so nie geplant war. Weiterhin werden hierdurch indirekte Service-Abhängigkeiten ersichtlich.

Digitalisierung: Es gibt kaum eine Branche, für die Software nicht bereits ein wesentlicher Teil ihrer Wertschöpfungskette ist, daher haben Downtimes oder langsame Antwortzeiten schnell Auswirkungen auf das Betriebsergebnis. Um hier schnell reagieren zu können, ist die Überwachung durch APM-Werkzeuge sehr wichtig.

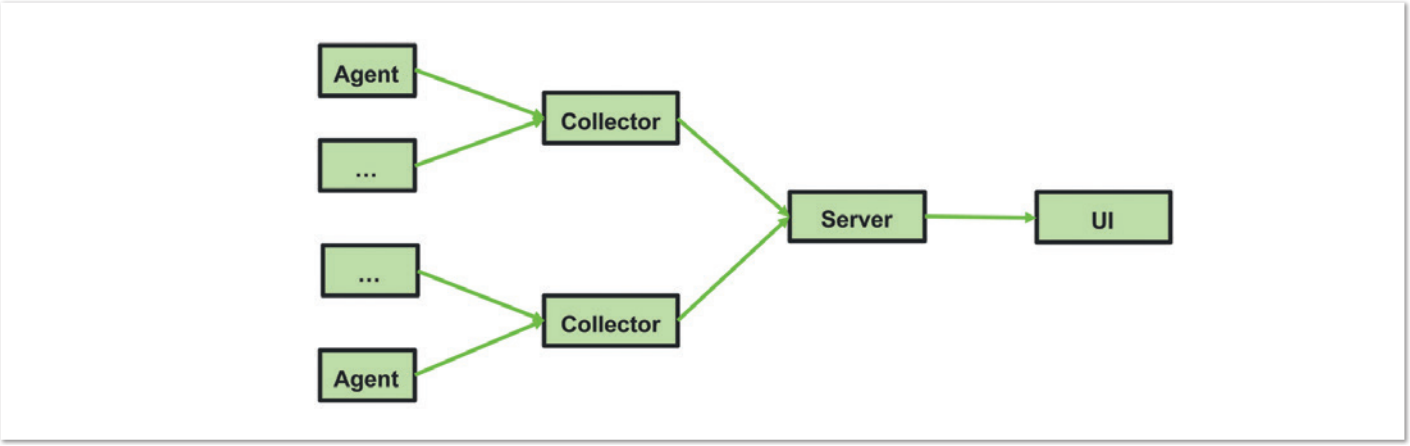


Abbildung 2: Architektur von APM-Lösungen (© RETIT GmbH)

Tracing-Standards: Die Entwicklung von Tracing-Lösungen ist aufwendig, insbesondere die Unterstützung verschiedener Java-Versionen (beispielsweise JDK 8 bis 17) oder unterschiedlicher Frameworks (zum Beispiel Hibernate, Spring, Java EE, Quarkus). Stets die aktuellen Versionen von Java oder unterschiedlicher Frameworks zu unterstützen, ist für ein einzelnes Open-Source-Projekt kaum zu bewerkstelligen. Daher wurden Tracing-Standards entwickelt, die es ermöglichen, diese Daten nur einmal zu erfassen und in verschiedenen Werkzeugen zu verarbeiten. Dies reduziert den Aufwand für die APM-Werkzeuge.

Kontext

Um die unterschiedlichen APM-Werkzeuge bewerten zu können, wird in diesem Kapitel zunächst die allgemeine Architektur aller Lösungen vorgestellt, um darauf aufbauend später aufzuzeigen, welche Teile durch welche Werkzeuge realisiert werden.

Die Architektur aller APM-Werkzeuge ist grundlegend ähnlich und wird in *Abbildung 2* dargestellt. Jedes Werkzeug basiert zunächst auf Agenten, die Daten (Traces und Metriken) über die Anwendung erheben. Diese Daten werden dann an sogenannte Collector-Komponenten weitergeleitet, die die Daten von einer Menge von Agenten entgegennehmen und eventuell bereits aggregieren. Dies soll dazu dienen, die Last vom zentralen Server zu reduzieren, an den die Collector-Komponenten die Daten weiterleiten. Der Server empfängt im Anschluss die Daten und stellt sie für die Nutzeroberfläche oder andere Clients über verschiedene Schnittstellen zur Verfügung. Für den Nutzer sichtbar ist am Ende das User Interface, das die Daten vom Server visualisiert.

Wenn man die Architektur einer solchen APM-Lösung vom Aufwand her betrachtet, ist es wichtig, sich vor Augen zu führen, dass dieser sehr ungleichmäßig verteilt ist. Während die Dinge, mit denen man als Nutzer am meisten zu tun hat (UI, Server, Collector) zwar sehr präsent sind, liegt der Aufwand für den Anbieter eines APM-Werkzeugs primär in der Entwicklung und der Pflege der Agenten. Der Hintergrund ist, dass man für einen Agenten eine Vielzahl von Java-Versionen (zum Beispiel JDK 8 bis 17) und Frameworks unterstützten muss und diese sich auch ständig ändern, was eine kontinuierliche Anpassung erforderlich macht.

Diese ungleiche Aufwandsverteilung ist der Grund, wieso die meisten Open Source APM Tools sich vor allem auf die Bereitstellung von Collector-, Server- und UI-Komponenten beschränken. Für die Erhebung der Daten wird in diesen Fällen eine Schnittstelle (API) bereitgestellt, die vom Nutzer dann selbst genutzt werden kann, um die notwendigen Daten zu erheben. Diese Beschränkung ist zwar nicht bei allen APM-Werkzeugen gegeben, aber dies ist der kleinste gemeinsame Nenner an Komponenten, die man in allen genannten Lösungen findet, wie in *Abbildung 3* dargestellt.

Da sich über die Zeit gezeigt hat, dass das Anbieten unterschiedlicher Schnittstellen für die Datenerhebung die Nutzung unterschiedlicher Werkzeuge erschwert, haben sich Standards entwickelt, die die Datenerhebung vereinheitlichen sollen.

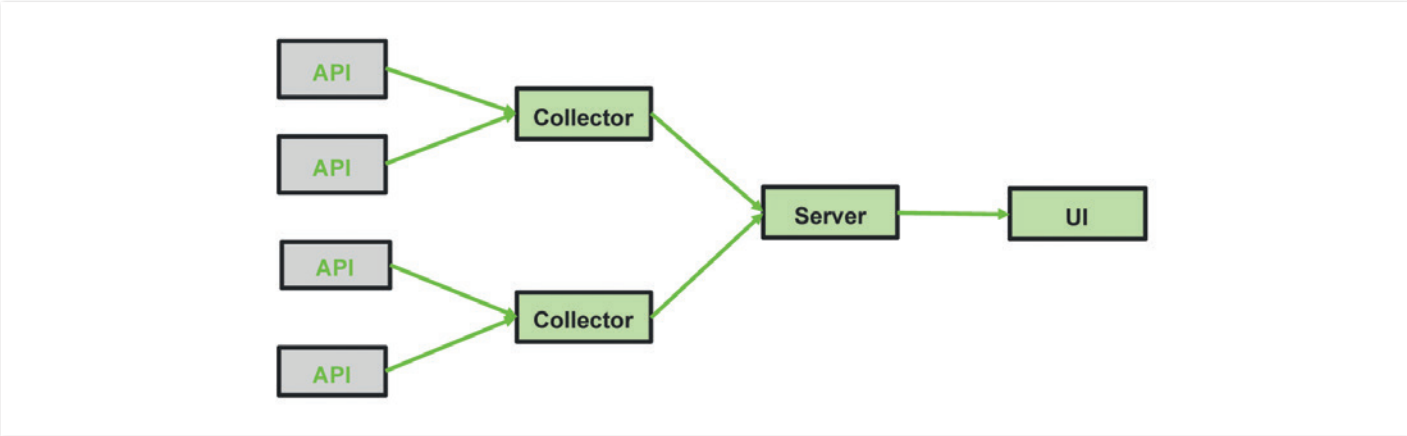


Abbildung 3: Funktionsumfang der meisten Open-Source-APM-Lösungen (© RETIT GmbH)

Open-Source-APM-Standards

Wie im vorigen Abschnitt erläutert, liegt der wesentliche Aufwand bei der Entwicklung eines APM-Werkzeugs in der Bereitstellung der Daten. Viele Open-Source-Werkzeuge haben hierzu initial individuelle Schnittstellen angeboten, die den Nutzern helfen sollten, Traces und Metriken zu erheben. Da sich Nutzer dadurch direkt an ein Werkzeug binden, haben viele den Aufwand gescheut. Aus diesem Grund haben sich sogenannte Open-Source-APM-Standards entwickelt. Diese sollen es ermöglichen, auf Basis von einheitlichen Schnittstellen die Trace- (und Metrik-) Daten zu erfassen.

Die wesentlichen beiden Standards waren zunächst OpenTracing *[16]* und OpenCensus *[17]*. Während OpenTracing primär die Erfassung von Traces standardisiert, hat OpenCensus sowohl die Erfassung von Traces als auch von Metriken vereinheitlicht. Da sich die Standards im Bereich der Traces nur unwesentlich unterscheiden, haben sich die Konsortien hinter beiden Standards 2019 entschieden, die Energien zu bündeln, die Standardisierung zu vereinheitlichen und sie unter dem Namen OpenTelemetry *[10]* fortzuführen *(siehe Abbildung 4)*.

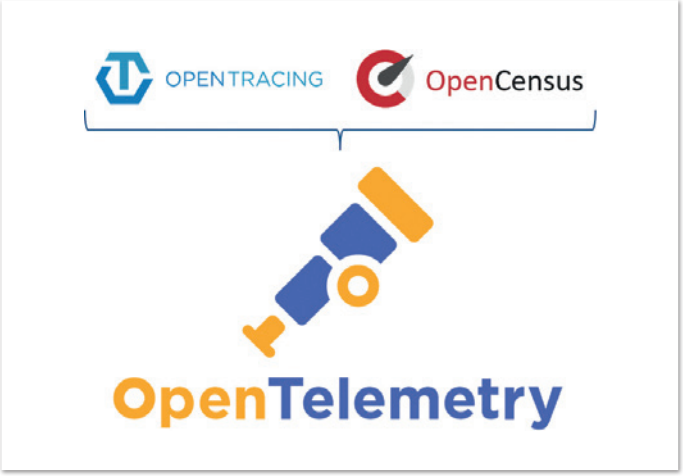


Abbildung 4: OpenTracing und OpenCensus sind in OpenTelemetry aufgegangen (Quelle: eigene Abbildung, Logo- und Iconbilder von den jeweiligen Webseiten der Standards: [10] [16] [17] © RETIT GmbH)

OpenTelemetry standardisiert sowohl die Erfassung von Traces als auch die von Metriken. Darüber hinaus bietet OpenTelemetry eine Kompatibilitätsschicht, die existierende OpenTracing- und OpenCensus-Instrumentierungen weiterhin unterstützt.

Neben dem Aspekt, dass OpenTracing nur die Trace-Erfassung vereinheitlicht und OpenCensus/OpenTelemetry Traces als auch Metriken vereinheitlicht, unterscheiden sich die Standards auch darin, wie viele Aspekte der Datenerhebung durch sie vereinheitlicht werden. In *Abbildung 5* werden die Standards im Kontext der bereits vorgestellten APM-Architektur visualisiert.

Wie aus dem Bild ersichtlich wird, standardisiert OpenTracing nur die Schnittstelle zur Datenerfassung (API). Dies macht es erforderlich, zusätzlich zum OpenTracing-API eine Agentenimplementierung dieses API (beispielsweise die Jaeger-Agentenimplementierung) mit in der Anwendung auszuliefern, die die Daten entgegennimmt und an einen technologiespezifischen Collector weiterleitet.

Im Vergleich dazu geht die Vereinheitlichung bei OpenTelemetry und OpenCensus weit darüber hinaus. Neben der Schnittstelle (API), werden sowohl der Agent als auch der Collector mit standardisiert. Dadurch wird die Datenverarbeitung bis hin zur Speicherung vereinheitlicht, was die Implementierung von APM-Werkzeugen, die auf diesen Standards aufsetzen, vereinfacht.

Der initiale Gedanke bei den ersten beiden Open-Source-APM-Standards war, dass sich durch die Vereinheitlichung Projekte entwickeln konnten, die die Datenerhebung einmal entwickelten und somit für viele nutzbar machen. Weiterhin hätten auf diesem Wege Framework-Anbieter (zum Beispiel Spring *[18]* und Hibernate *[19]*) einfach ihren eigenen Code mit einem entsprechenden Standard versehen können, was eine eigene Instrumentierung durch den Nutzer überflüssig machen würde. Diese Idee wurde von einigen Projekten (etwa OpenTracing Microprofile *[20]*) aufgegriffen und zusätzlich wurden Auto-Instrumentierungen für bestimmte Frameworks entwickelt.

Jedoch waren diese Initiativen nicht so umfangreich, als dass die Nutzer vollständig auf eigene Instrumentierungen verzichten könnten. Aus diesem Grund hat das OpenTelemetry-Projekt ein neues Projekt mit dem Namen „OpenTelemetry Instrumentation for Java" *[21]* gestartet, das den Aufwand für den Nutzer auf ein absolutes Minimum

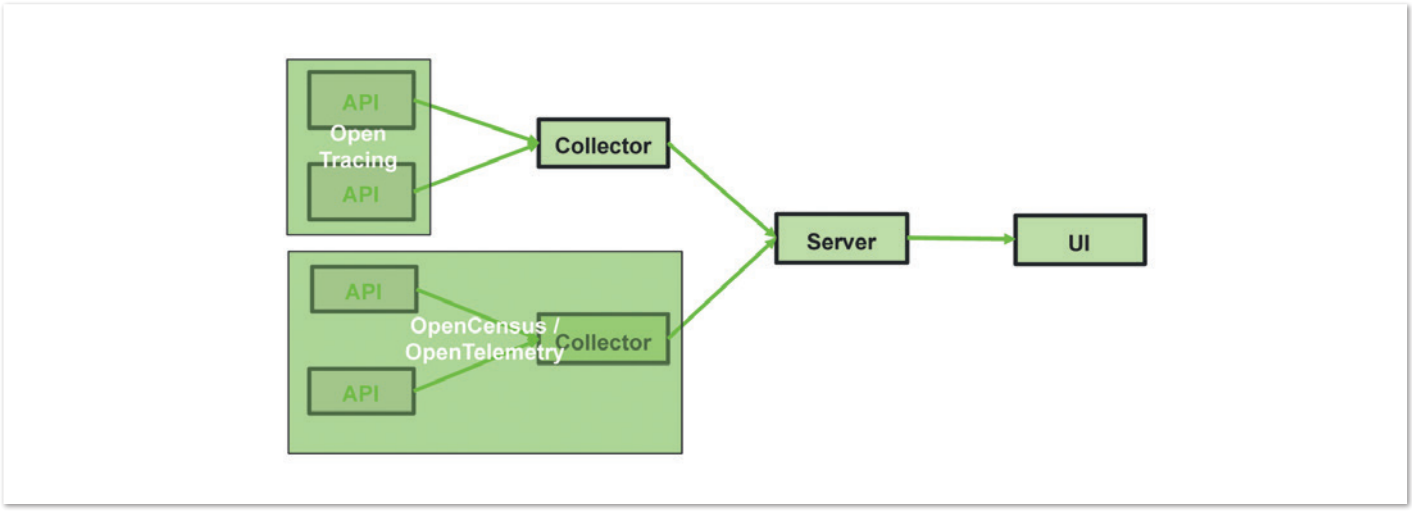


Abbildung 5: Open-Source-Standards im Kontext der APM-Tool-Architektur (© RETIT GmbH)

```
java -javaagent:path/to/opentelemetry-javaagent-all.jar -jar myapp.jar
```

Listing 1: Instrumentierung mit dem OpenTelemetry-Java-Autoinstrumentierungsagenten

reduzieren soll. Dieses Projekt stellt einen Java-Agenten bereit, der beim Start einer Java-Anwendung in den Java-Prozess eingebunden wird, wie in *Listing 1* gezeigt. Durch diesen werden sehr viele, typischerweise in Java-Unternehmensanwendungen genutzte Frameworks automatisch instrumentiert *[21]*, was den Aufwand für den Nutzer auf das Hinzufügen des Agenten beschränkt und einen großen Schritt für die Weiterentwicklung der Standards darstellt.

Beispiele der relevantesten Open-Source-Lösungen

Um die relevantesten Open-Source-APM-Lösungen zu identifizieren, gibt es viele Möglichkeiten. Eine Möglichkeit, deren Popularität zu messen, ist, die Anzahl der GitHub-Sterne zu betrachten *(siehe Abbildung 6)*. Apache Skywalking, Zipkin, Jaeger und Pinpoint sind mit Abstand die populärsten Vertreter der genannten APM-Werkzeuge auf Basis dieser Werte. Auch andere Werte, wie die Anzahl der GitHub-Contributer oder Google-Suchanfragen, ergeben ein vergleichbares Bild. Daher werden in den folgenden Abschnitten Beispiele für die Nutzung dieser vier populärsten Werkzeuge gegeben.

Die vier genannten Lösungen können in verschiedenen Konfigurationen verwendet werden, diese werden in *Abbildung 7* dargestellt. Da ein Einstieg mit OpenTracing oder OpenCensus derzeit nicht zu empfehlen ist, ist die erste Option, die Datenerhebung mit OpenTelemetry zu realisieren. Aktuell ist dies die wahrscheinlich zukunftssicherste Variante, da sich derzeit der größte Aufwand der Community auf dieses Setup fokussiert. In diesem Szenario

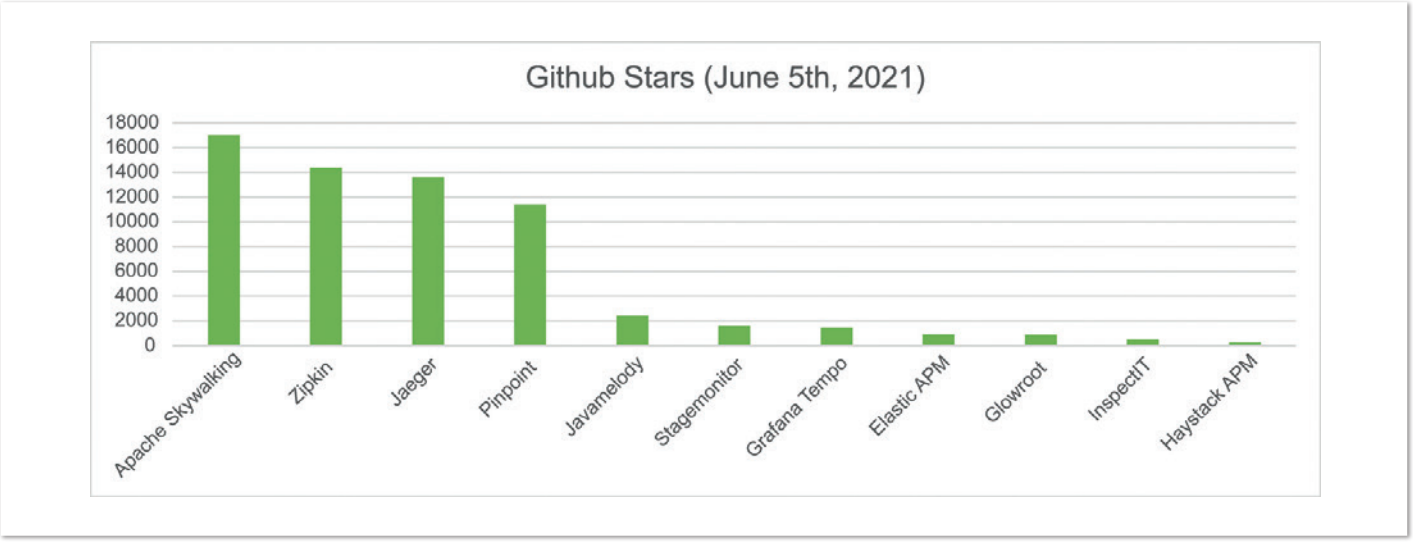


Abbildung 6: Popularität der Open-Source-APM-Werkzeuge anhand der GitHub-Sterne (© RETIT GmbH)

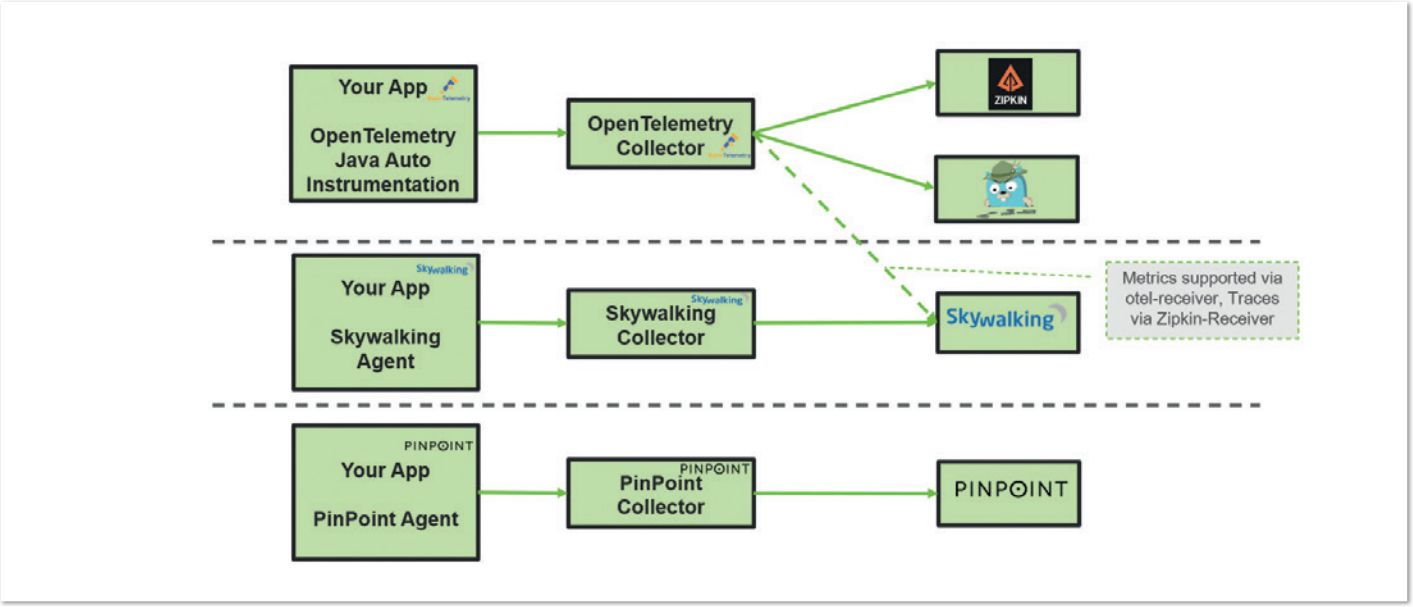


Abbildung 7: Mögliche Szenarien für die Nutzung der populärsten Open-Source-APM-Werkzeuge (Quelle: eigene Abbildung, Icons von den jeweiligen Webseiten: [4] [6] [3] [2] [10] © RETIT GmbH)

wird der OpenTelemetry-Java-Auto-Instrumentierungsagent verwendet, um die Datenerhebung zu automatisieren. Dieser Agent sendet die Daten an einen OpenTelemetry-Collector, der die Daten an ein entsprechendes APM-Werkzeug weiterleitet. In *Abbildung 7* wird dies entsprechend dem Fokus auf Tracing-Lösungen nur für Jaeger und Zipkin aufgezeigt, aber man könnte zusätzlich Metriken auf diesem Wege erheben und automatisiert an einen Prometheus-Server weiterleiten, womit man ein vollständiges APM-Setup aufbauen könnte. Neben diesen Setups ist es wichtig zu erwähnen, dass die Nutzung von OpenTelemetry es auch einfach ermöglicht, die APM-Werkzeuge der drei größten Cloud-Anbieter, nämlich Amazon Web Services (X-Ray und Cloudwatch *[22]*), Google Cloud Platform (Google Cloud Monitoring *[23]*) oder Microsoft Azure (Azure Monitor *[24]*) einzubinden, weshalb dies mit Abstand das flexibelste Setup ist.

Sollte man Apache Skywalking verwenden, wäre es möglich, das Setup „aus einer Hand" zu realisieren, da Skywalking einen eigenen Java-Agenten zur Datenerhebung anbietet. Dieser Agent leitet die Daten an einen von Apache Skywalking angebotenen Collector, der wiederum die Daten an den Skywalking-Server weiterleitet. Es wäre, wie durch den gestrichelten Pfeil in *Abbildung 7* angedeutet, bereits heute möglich, Trace- und Metrikdaten vom OpenTelemetry-Collector an den Skywalking-Server weiterzuleiten *[25]*. Jedoch sind die beiden Komponenten für diese Integration (OpenTelemetry-Receiver für Metriken und Zipkin-Receiver für Traces) auf der Seite von Skywalking derzeit noch in einem frühen Stadium der Entwicklung, weshalb die Nutzung der eigenen Komponenten von Skywalking derzeit die stabilere Lösung darstellt. Insbesondere der Zipkin-Receiver wird derzeit auf der Website noch als „experimental" bezeichnet *[2]*.

Bei Pinpoint sieht das Szenario ähnlich aus wie bei Skywalking, da Agent, Collector und Server alle vom selben Projekt angeboten werden. Eine Integration mit OpenTelemetry ist dem Autor derzeit nicht bekannt.

Im Folgenden werden die wesentlichen Funktionen der vier genannten Werkzeuge vorgestellt. Hierbei werden jeweils zwei bis drei wesentliche Ansichten der jeweiligen Werkzeuge erläutert:

Trace-Details-Sicht: In dieser Ansicht sieht man den Verlauf einer einzelnen Transaktion durch eine Anwendung. Es wird visualisiert, welche Operationen in welcher Reihenfolge aufgerufen wurden und wie lange ein einzelner Operationsaufruf gedauert hat. Die Ansichten sind in den Tools sehr vergleichbar und finden sich in *Abbildung 8* für Zipkin, *Abbildung 10* für Jaeger, *Abbildung 12* für Pinpoint und in *Abbildung 15* für Apache Skywalking.

Service-Dependency-Graph: In dieser Ansicht sieht man die Abhängigkeiten einzelner Services. Durch diese Information kann man unter anderem analysieren, welchen Effekt ein Ausfall oder langsame Antwortzeiten eines Service für das Gesamtsystem haben. Die Werkzeuge nutzen zur Visualisierung leicht unterschiedliche Ansätze, die jedoch grundlegend dieselben Informationen anzeigen. Für Zipkin, Jaeger und Apache Skywalking werden diese Dependency-Graphen in *Abbildung 9*, *Abbildung 11* und *Abbildung 16* dargestellt. Man erkennt, dass diese Darstellungen wenig technologiespezifische Informationen enthalten, wenn man diese mit dem Dependency-Graph von Pinpoint in *Abbildung 13* vergleicht. Der Hintergrund dieses Unterschieds liegt darin, dass Pinpoint ein auf Java spezialisiertes Werkzeug ist, während Zipkin, Jaeger und Apache Skywalking keinen speziellen Fokus auf eine Technologie legen.

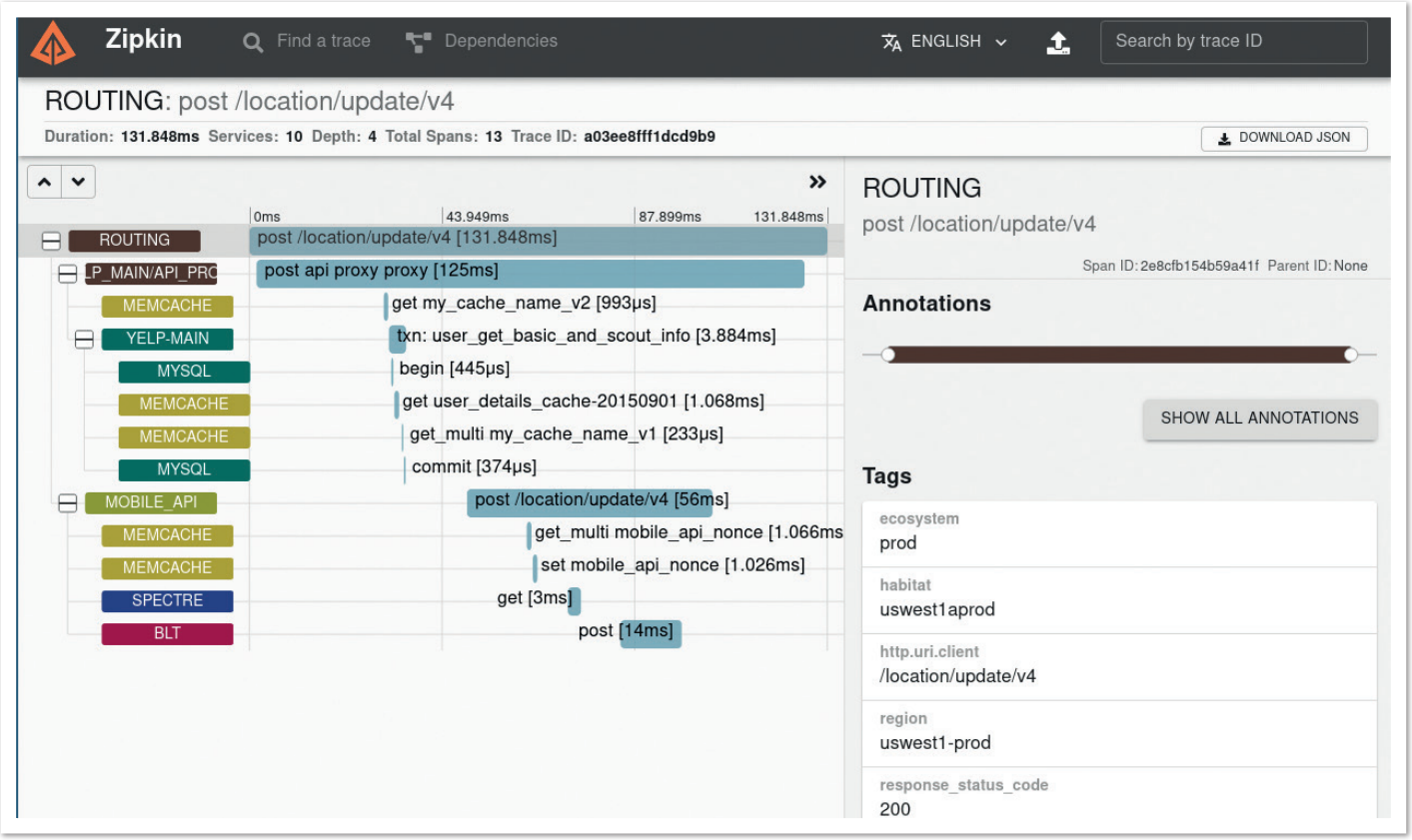


Abbildung 8: Zipkin Trace-Details-Ansicht (Quelle: [2])

Metrik-Dashboards: Eine Übersicht über Metriken bieten nur Pinpoint *(siehe Abbildung 14)* und Apache Skywalking *(siehe Abbildung 17)*, da diese sich wie eingangs erwähnt nicht nur auf Traces fokussieren. Weiterhin zeigt sich hier ein Vorteil der Entwicklung eines eigenen Agenten, da diese Open-Source-Werkzeuge alle nötigen Informationen selbst erheben können. Bei der *Abbildung 17* für Apache Skywalking handelt es sich darüber hinaus nicht nur um eine reine Metrikansicht, sondern um ein Dashboard. Dies ist eine Fähigkeit, die die anderen drei Werkzeuge nicht bieten. Hierdurch kann man sich für die Produktionsüberwachung die wesentlichen Informationen auf einen Blick zusammenstellen und von dort aus bei Problemen detaillierter in die anderen Sichten zur Analyse einsteigen. Dies ist einer der Gründe für die große Popularität von Skywalking im Vergleich zu den anderen Werkzeugen.

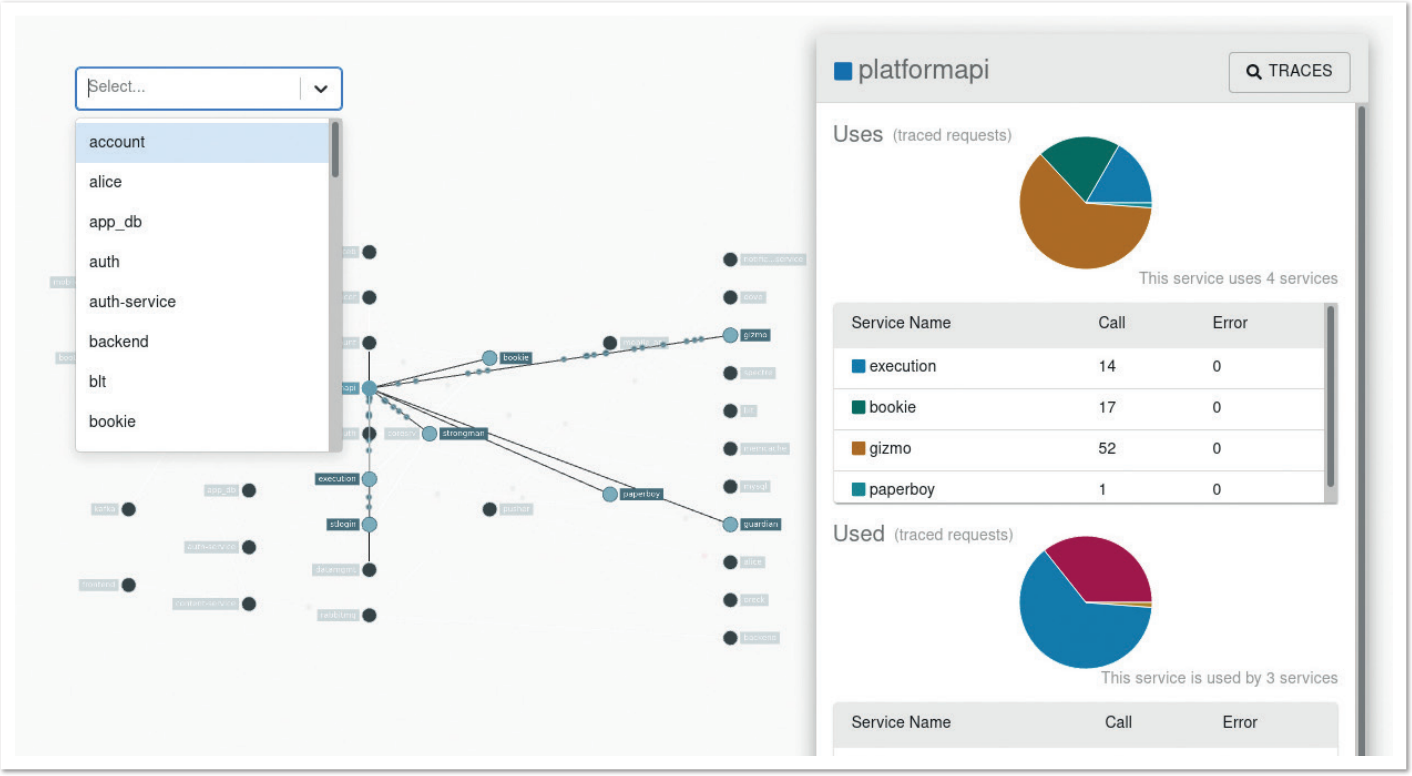


Abbildung 9: Zipkin Service-Dependency-Graph (Quelle: [2])

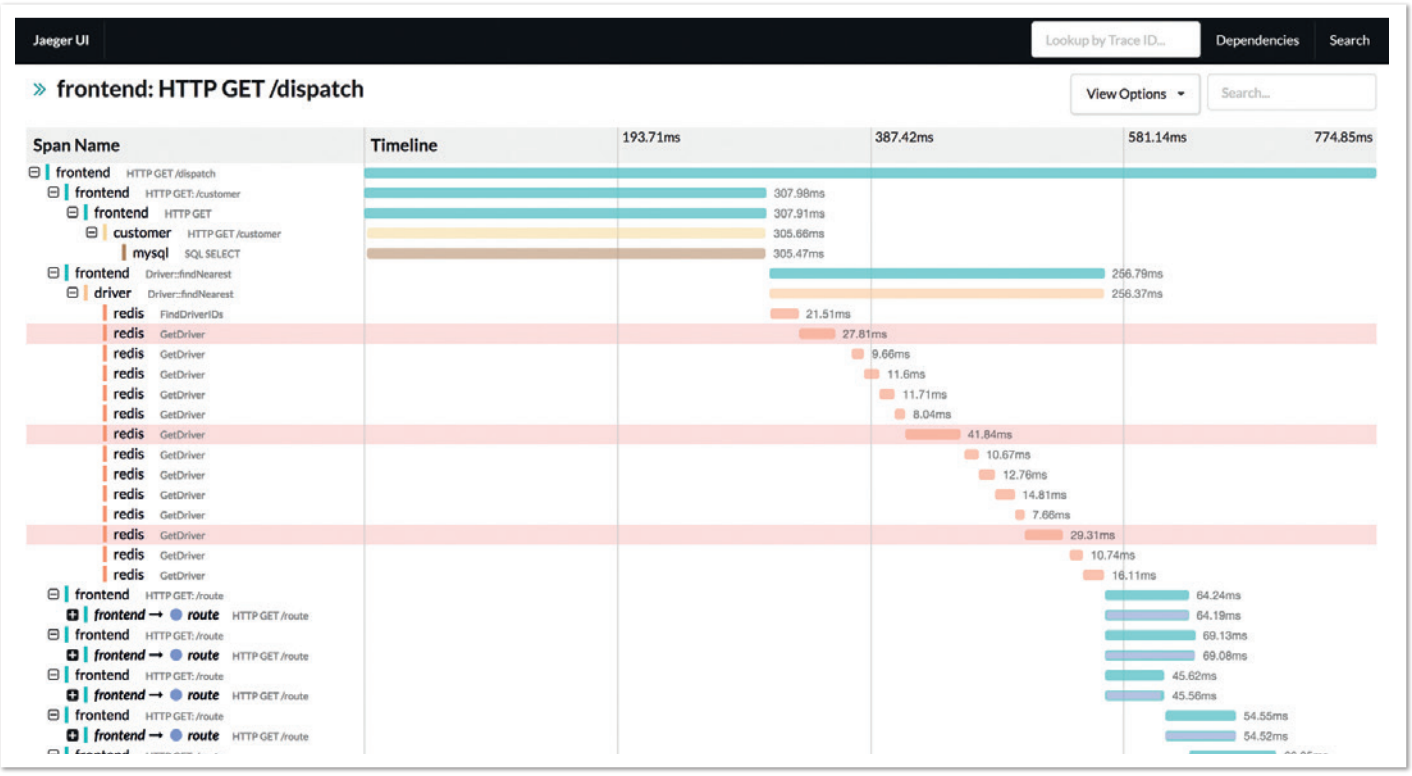


Abbildung 10: Jaeger Trace-Details-Ansicht (Quelle: [26])

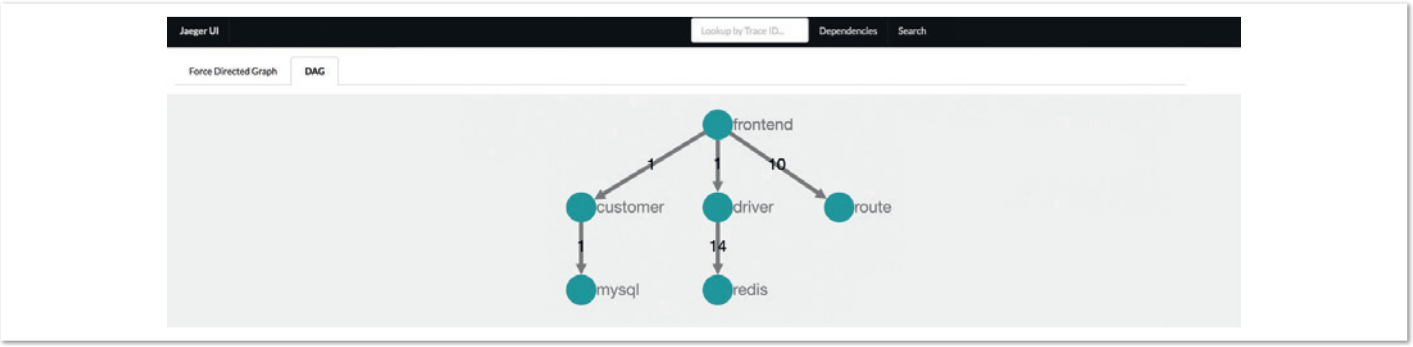


Abbildung 11: Jaeger Service-Dependency-Graph (Quelle: [26])

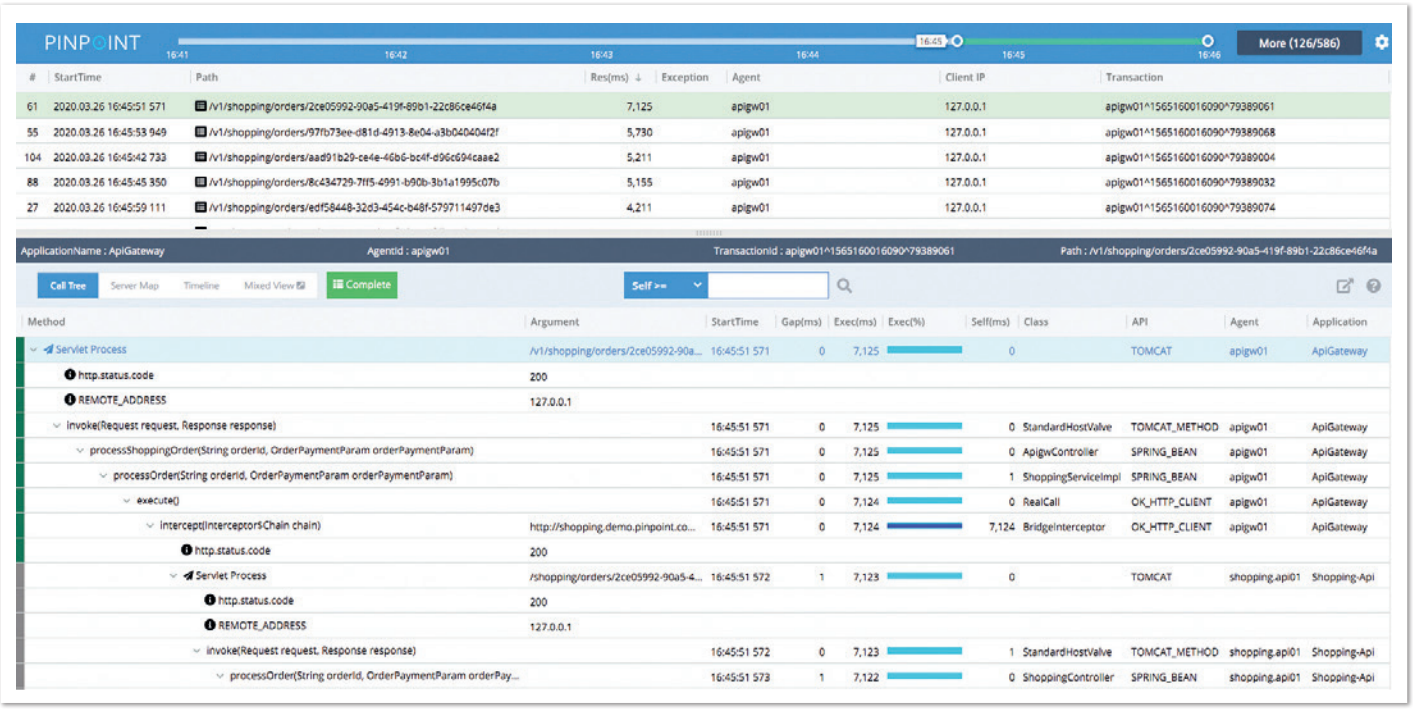


Abbildung 12: Pinpoint-Trace-Details-Ansicht (Quelle: [6])

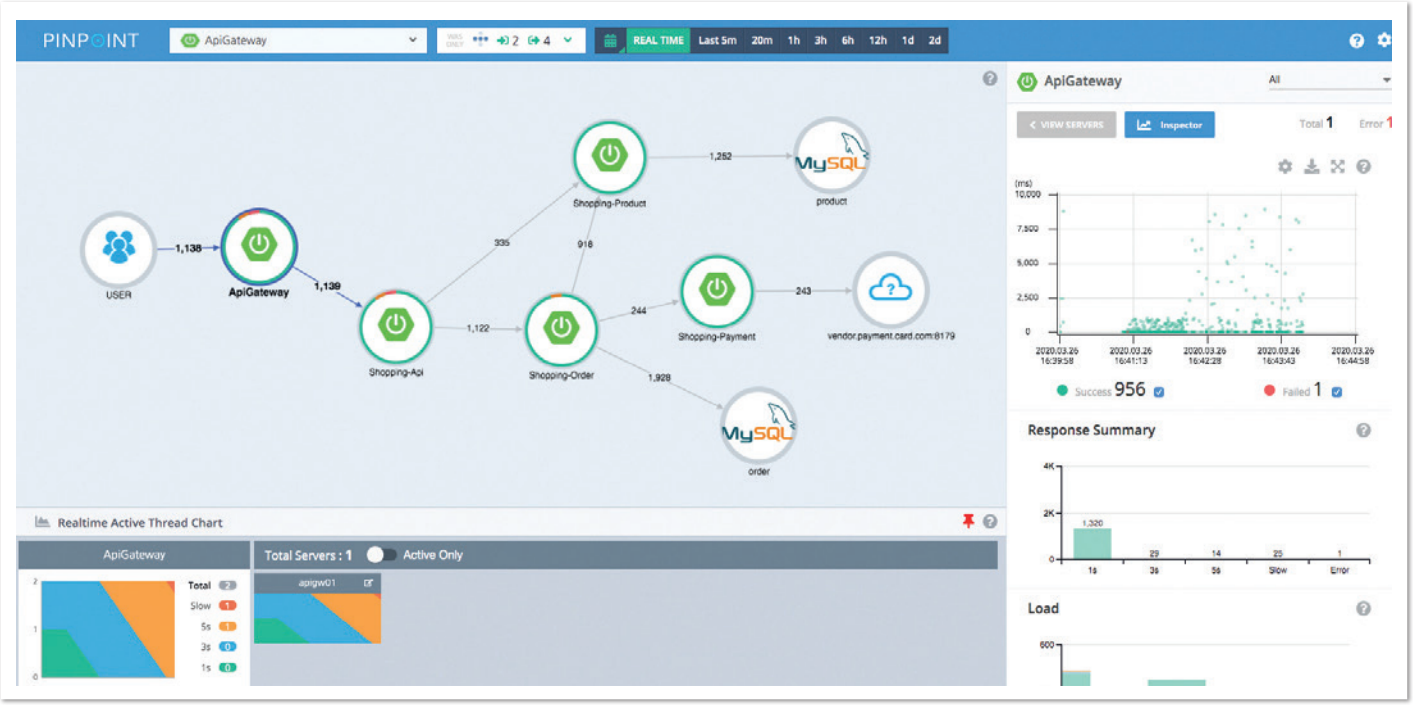


Abbildung 13: Pinpoint Service-Dependency-Graph (Quelle: [6])

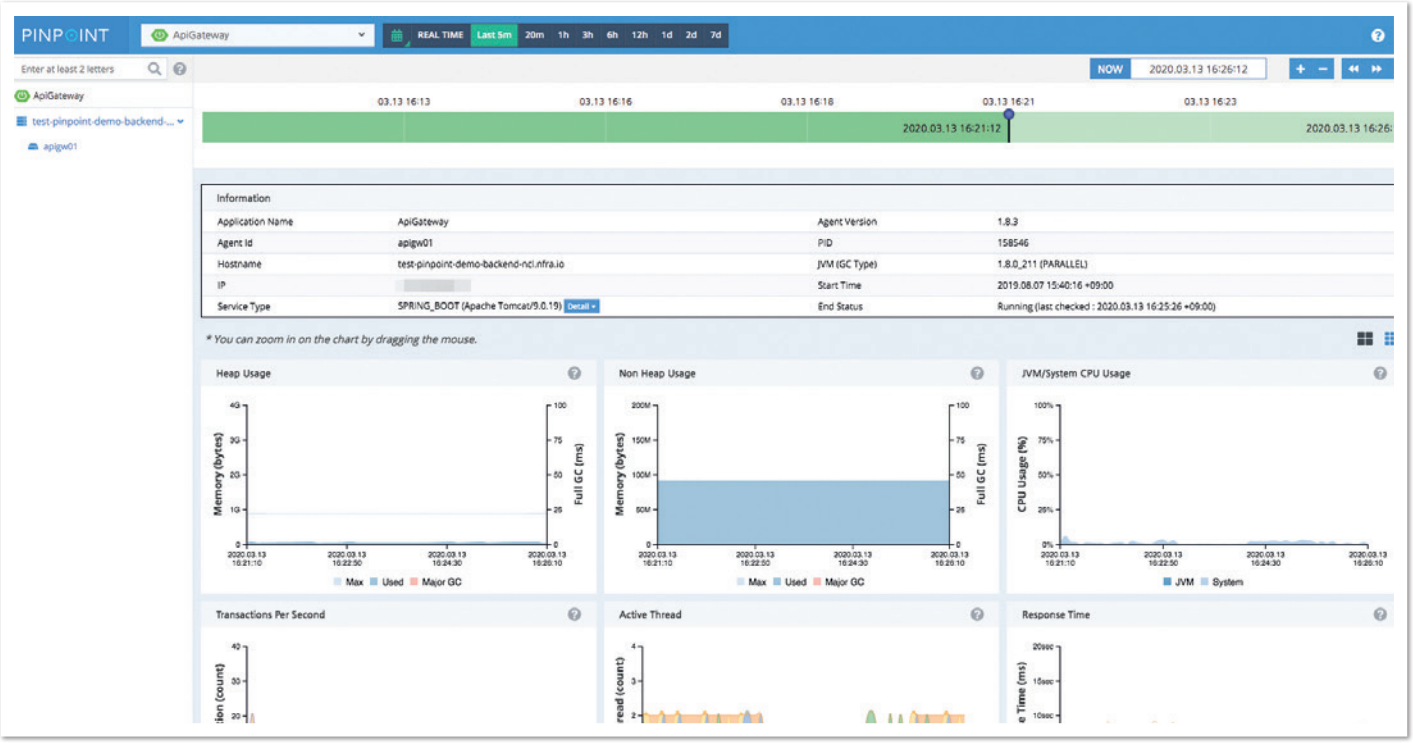


Abbildung 14: Pinpoint Metrik-Übersicht (Quelle: [6])

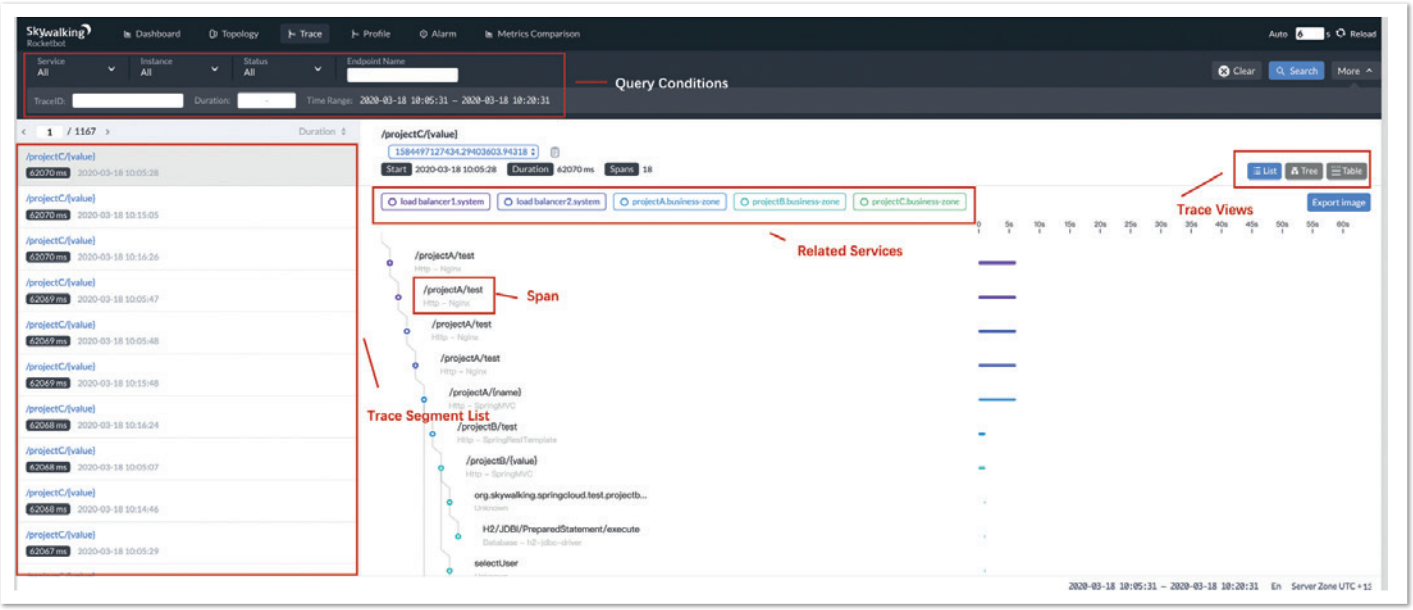


Abbildung 15: Apache Skywalking Trace-Details-Ansicht (Quelle: [4])

Gründe für kommerzielle Lösungen

Wie durch diesen Artikel gezeigt wurde, gibt es bereits sehr gute Open-Source-APM-Lösungen für Java auf dem Markt, die es nicht erfordern, Lizenzgebühren an einen Hersteller zu zahlen. Jedoch sollte man sich bei diesen Lösungen einige Dinge vor Augen führen, die eventuell doch für eine kommerzielle Lösung sprechen. Im Folgenden werden hierzu einige Gründe aufgelistet.

Auch Open-Source-Lösungen kosten Geld: Obwohl für Open-Source-APM-Lösungen keine Lizenzkosten erhoben werden, kostet deren Einsatz Geld. Diese Kosten ergeben sich primär aus dem Aufwand für das Aufsetzen der Lösung, der Maintenance der Infrastruktur und die Schulung der Mitarbeiter. Insbesondere die Maintenance der Infrastruktur und die Schulungsaufwände sollten hierbei nicht unterschätzt werden, da nur durch eine entsprechende Stabilität der Lösung und ausreichendes Wissen der Mitarbeiter ein erfolgreicher Einsatz sichergestellt werden kann *[27]*. Manchmal zeigt sich dann, dass die Kosten für Pflege und Betrieb der Infrastruktur schnell ähnlich hoch sind wie die monatlichen Kosten einer Software-as-a-Service-Lösung eines kommerziellen APM-Anbieters.

Leichtere Problemlösung: Da die Erhebung der Daten über die Agenten sehr komplex ist, kann es hier schnell zu Problemen kommen. Typische Szenarien sind hierbei „blinde Flecken", wenn bestimmte Aspekte der Anwendungen nicht instrumentiert werden oder in der APM-Oberfläche nicht sichtbar sind. Beim Einsatz von

Abbildung 16: Apache Skywalking Service-Dependency-Graph (Quelle: [4])

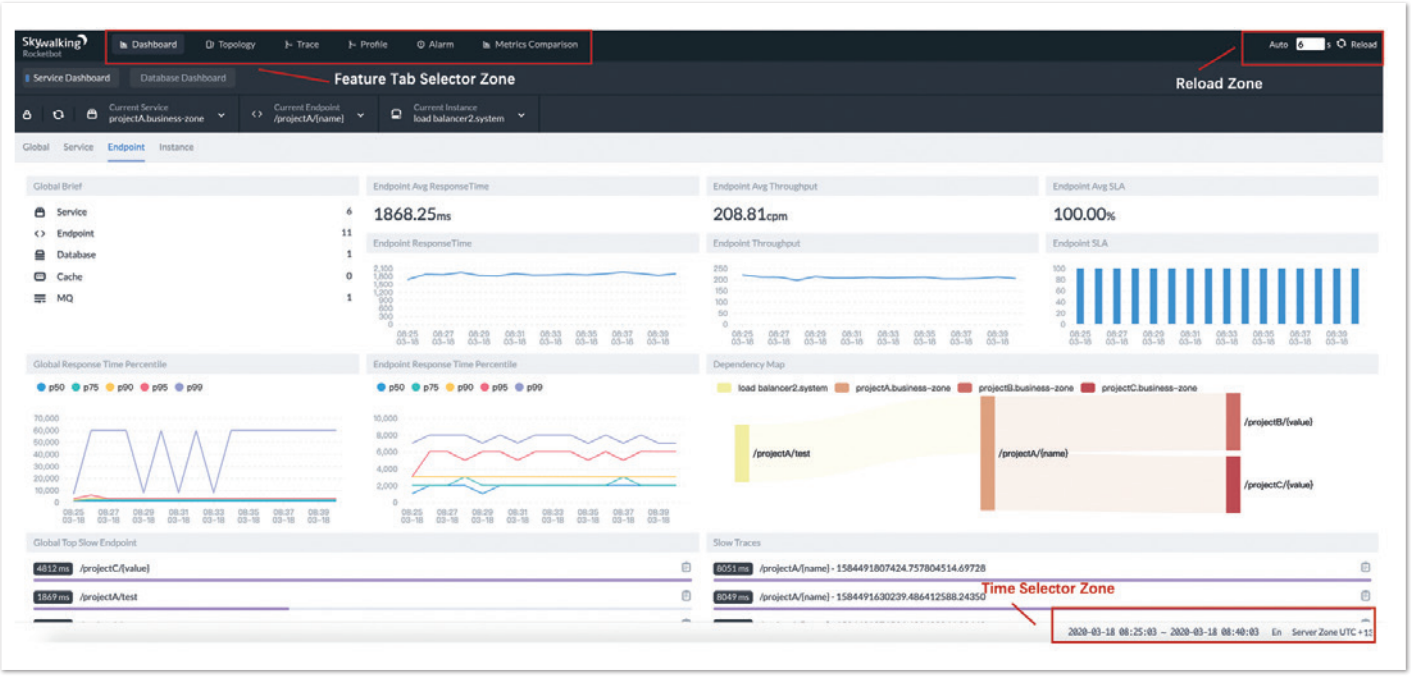


Abbildung 17: Apache Skywalking Metrik-Dashboard (Quelle: [4])

Open-Source-APM-Lösungen sind Sie bei der Analyse und Lösung eines solchen Problems meist auf sich allein gestellt. Natürlich können Sie in den Foren und Ticketsystemen der Open-Source-Projekte um Hilfe bitten, aber es gibt hier keine garantierten Service-Levels, auf die Sie sich verlassen können. Bei den kommerziellen Lösungen haben Sie hingegen die Möglichkeit, den Support zu kontaktieren, der Sie bei der Problemanalyse und -behebung unterstützt.

Breitere Technologieunterstützung: Wie bereits eingangs erwähnt, ist die Entwicklung von APM-Agenten sehr zeit- und damit kostenintensiv. Die Open-Source-Gemeinschaft kann nicht in jede einzelne Version einer Technologie die gleiche Menge an Arbeitskraft stecken (zum Beispiel Unterstützung von Tomcat 5 bis 10), wie dies die kommerziellen Anbieter mit ihren Mitarbeitern tun können. Hier ändert sich derzeit etwas durch die Konsolidierung auf den OpenTelemetry-Java-Autoinstrumentierungsagenten, aber generell existiert das Problem weiterhin und führt dazu, das kommerzielle Werkzeuge eine breitere Technologie-Unterstützung anbieten als Open-Source-Lösungen.

Sie können vorausplanen: Hersteller kommunizieren üblicherweise, bis zu welchem Zeitpunkt eine Softwareversion unterstützt wird, und unterstützen auch die Übergangsphase. Dies ist bei Open-Source-Software nicht immer der Fall und kann zu unerwünschten Überraschungen führen, wenn die eigene Technologie nicht mehr unterstützt wird.

Zusammenfassung

In diesem Artikel wurde eine Übersicht über die wesentlichen Open-Source-APM-Lösungen für Java-basierte Unternehmensanwendungen gegeben, mit dem Ziel, dem Leser eine Entscheidungsgrundlage für deren Einsatz zu bieten. Hierbei wurden zunächst die wesentlichen Lösungen vorgestellt und die Gründe für deren Entstehung. Anschließend wurde die generelle Architektur von APM-Lösungen aufgezeigt und erläutert, dass sich die meisten Open-Source-Lösungen auf die Speicherung und Visualisierung von APM-Daten fokussieren. Die Datenerfassung wird daher durch Open-Source-APM-Standards wie OpenTelemetry übernommen, die die Daten dann an die jeweiligen APM-Werkzeuge weiterleiten. Im darauffolgenden Abschnitt wurden mögliche Setups für die vier prominentesten APM-Werkzeuge (Apache Skywalking, Pinpoint, Zipkin, Jaeger) beleuchtet und ihre jeweiligen Fähigkeiten der Werkzeuge erläutert. Abschließend wurden noch Gründe für den Einsatz von kommerziellen Lösungen aufgezählt, die man bei der Entscheidung für eine APM-Lösung immer berücksichtigen sollte.

Referenzen

[1] F. Rössel, „Observability: Die Herausforderungen beim Monitoring beherrschen," Heise Online, 25.02.2020, *https://www.heise.de/hintergrund/Observability-Die-Herausforderungen-beim-Monitoring-beherrschen-4647333.html*

[2] „Zipkin," 2021, *https://zipkin.io/*

[3] „Jaeger: open source, end-to-end distributed tracing," The Linux Foundation, The Jaeger Authors, 2021, *https://www.jaegertracing.io/*

[4] „Apache SkyWalking," The Apache Software Foundation, 2021, *https://skywalking.apache.org/*

[5] B. H. Sigelman, . L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan und C. Shanbhag, „Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," Google, Inc., 2010

[6] „Pinpoint," 2021, *https://pinpoint-apm.github.io/pinpoint/*

[7] „Elastic APM," Elasticsearch, 2021, *https://www.elastic.co/de/apm*

[8] „InspectIT," Novatec, 2021, *https://www.inspectit.rocks/*

[9] „Glowroot," 2021, *https://glowroot.org/*

[10] „OpenTelemetry," 2021, *https://opentelemetry.io/*

[11] „Haystack," Expedia, 2021, *https://expediadotcom.github.io/haystack/*

[12] „Stagemonitor," iSYS Software GmbH, 2021, *https://www.stagemonitor.org/de/*

[13] „JavaMelody : monitoring of JavaEE applications," 2019, *https://github.com/javamelody/javamelody/wiki*

[14] „Prometheus," The Linux Foundation, Prometheus Authors, 2021, *https://prometheus.io/*

[15] „Grafana Tempo," Grafana, 2021, *https://github.com/grafana/tempo*

[16] „OpenTracing," 2021, *https://opentracing.io/*

[17] „OpenCensus," 2021, *https://opencensus.io/*

[18] „Spring," VMware, Inc., 2021, *https://spring.io/*

[19] „Hibernate," Red Hat, 2021, *https://hibernate.org/*

[20] „Eclipse MicroProfile OpenTracing," 2021, *https://github.com/eclipse/microprofile-opentracing/blob/master/spec/src/main/asciidoc/microprofile-opentracing.asciidoc*

[21] „OpenTelemetry Instrumentation for Java," 2021, *https://github.com/open-telemetry/opentelemetry-java-instrumentation*

[22] „AWS Distro for OpenTelemetry," Amazon Web Services, 2021, *https://aws-otel.github.io/*

[23] „What is OpenTelemetry?," Google, 2021, *https://cloud.google.com/learn/what-is-opentelemetry*

[24] „Azure Monitor OpenTelemetry Exporter client library for Java," Microsoft, 2021, *https://docs.microsoft.com/en-us/java/api/overview/azure/monitor-opentelemetry-exporter-readme*

[25] „Apache Skywalking Backend Receivers," 2021, *https://github.com/apache/skywalking/blob/master/docs/en/setup/backend/backend-receivers.md*

[26] Y. Shkuro, „Take OpenTracing for a HotROD ride," 06.05.2017, *https://medium.com/opentracing/take-opentracing-for-a-hotrod-ride-f6e3141f7941*

[27] T. Hafen, „Application Performance Monitoring - Im Gespräch mit Dr. Andreas Brunnert von RETIT," com! professional, 07.03.2019, *https://www.com-magazin.de/praxis/business-it/application-performance-monitoring-1686592.html?page=3_im-gespraech-mit-dr.-andreas-brunnert-von-retit*

Dr. Andreas Brunnert
RETIT GmbH
brunnert@retit.de

Dr. Andreas Brunnert ist Gründer der RETIT GmbH. Mit seinem Team arbeitet er kontinuierlich daran, die Performance komplexer Unternehmensanwendungen zu optimieren. Hierzu nutzen sie eine Kombination von mess- und modellbasierten Performance-Evaluationsansätzen, um im gesamten Anwendungslebenszyklus optimale Lösungen zu finden.

frgaal: Ein Retrofit-Compiler für Java

Anton Epple, Smart Access Solutions UG (haftungsbeschränkt)/Eppleton It Consulting

frgaal [1] (Abbildung 1) ist ein Retrofit-Compiler für Java, der es ermöglicht, moderne und sogar experimentelle Sprachfeatures zu verwenden und trotzdem die Kompatibilität mit älteren Runtimes zu erhalten. Braucht die Java-Welt mit ihren planbaren Deployments so etwas? In diesem Artikel geht es um Use Cases, in denen frgaal sehr hilfreich sein kann, und auch darum, wo die Grenzen liegen.

In der JavaScript-Welt müssen sich Entwickler mit einer Vielzahl von JavaScript-Engines in verschiedenen Browser-Versionen herumschlagen, während sich die Sprache selbst schnell weiterentwickelt. Zur Zeit der Entwicklung ist meist unbekannt, in welchen Browsern der Code zum Einsatz kommt. Dies ist nur möglich, weil es clevere Werkzeuge wie den Retrofit Compiler Babel *[2]* gibt, der es Entwicklern ermöglicht, moderne JavaScript-Sprachfunktionen zu verwenden und trotzdem mit den meisten Browsern kompatibel zu bleiben. Der Retrofit-Compiler übernimmt dabei transparent für den User die Rückportierung der neuen Sprachfeatures.

In der Java-Welt ist die Situation viel besser, da Java-Desktop-Anwendungen in der Regel mit einer eingebetteten JRE ausgeliefert werden und für Server-Anwendungen die JRE-Version klar definiert ist. Aber während Entwickler in der Regel wissen, welche Version von Java sie unterstützen müssen, sind sie häufig gezwungen, für alte Java-Versionen zu entwickeln. Hier kann ein Retrofit-Compiler helfen.

Abbildung 1: Das frgaal-Projekt auf GitHub (© Eppleton IT Consulting)

Babel für Java

frgaal richtet sich an all jene Entwickler, die auf einer Runtime festsitzen, die keine modernen Sprachfeatures unterstützt. Einige Beispiele, bei denen dies nützlich sein könnte, sind serverlose Funktionen, Legacy-Anwendungen oder einfach der Wunsch, neue und experimentelle Funktionen auszuprobieren. frgaal ist ein relativ neues Open-Source-Projekt, das direkt auf dem javac-Compiler aufbaut. Dieser ist nämlich durch minimale Änderungen durchaus in der Lage, viele der neuen Sprachfeatures ganz einfach in Java-8-Bytecode zu kompilieren.

frgaal für Maven

Um frgaal in einem Maven-Projekt einzubinden, müssen Sie lediglich das Java-Compiler-Plug-in konfigurieren, wie in *Listing 1* gezeigt.

Um Ihnen das Tippen zu ersparen, habe ich ein Demo-Repository vorbereitet, das Sie nun klonen können: `$ git clone https://github.com/eppleton/frgaal-demo.git`

Sehen Sie sich den Demo-Code in *Listing 2* an; er verwendet einige neuere Features wie Textblöcke und erweiterte `instanceof`-Syntax.

Sie können dieses Projekt wie jedes andere reguläre Java-Projekt bauen und dann ausführen. Zum Testen sollten Sie natürlich am besten JDK 8 verwenden. In *Listing 3* wird gezeigt, wie das Projekt gebaut und gestartet wird, sowie die erwartete Ausgabe.

Das war auch schon alles, was nötig ist, um frgaal zum Laufen zu bringen. So können Sie nun problemlos neue und experimentelle Features ausprobieren und nutzen, auch wenn Sie an eine alte JDK gebunden sind. Die frgaal-Website enthält auch die nötigen Instruktionen, um frgaal mit Gradle zu verwenden *[1]*.

Wie funktioniert das überhaupt?

frgaal baut auf dem javac-Compiler von OpenJDK auf und Sie können damit gültigen Java-Code kompilieren. Trotzdem ist es technisch gesehen, strenggenommen, kein Java. Das erklärt einige der seltsamen Formulierungen auf der Projekt-Website, die von „einer Java-ähnlichen Sprache" sprechen, obwohl ausschließlich gültiges Java und die experimentellen Features unterstützt werden.

Das Projekt patcht den javac-15-Compiler, damit er Bytecode für JDK8 erzeugen kann, und kompiliert diesen Compiler dann für JDK 8. Als Ergebnis hat der frgaal-Compiler alle Features von JDK15, kann aber unter JDK 8 laufen und JDK 8 kompatiblen Bytecode ausgeben.

```
<build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <dependencies>
                    <dependency>
                        <groupId>org.frgaal</groupId>
                        <artifactId>compiler-maven-plugin</artifactId>
                        <version>14.0.2</version>
                    </dependency>
                </dependencies>
                <configuration>
                    <compilerId>frgaal</compilerId>
                    <source>14</source>
                    <target>1.8</target>
                    <compilerArgs>
                        <arg>-Xlint:deprecation</arg>
                        <arg>--enable-safe-preview</arg>
                    </compilerArgs>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Listing 1: Konfigurieren des Java-Compiler-Plug-ins

```
import java.util.List;
import java.util.Arrays;

public class Main {
    private static List<Integer> useVar() {
        var list = Arrays.asList(6, 1, 3, 5);
        useTextBlock(list);
        return list;
    }

    private static void useTextBlock(List<Integer> list) {
        String text = """
            initial list content
            is...""";
        System.err.println(text + list);
    }


    private static String useInstanceOf(List<?> list) {
        final Object element = list.get(1);
        if (element instanceof Integer number) {
            return useSwitchExpr(number);
        }
        return "not a number!";
    }

    private static String useSwitchExpr(int number) {
        return switch (number) {
            case 3 -> "ok";
            default -> "bad";
        };
    }

    public static void main(String... args) {
        List<Integer> list = useVar();
        list.sort(null);
        System.err.println("after sorting: " + list);
        String switchTest = useInstanceOf(list);
        System.err.println(switchTest);
    }
}
```

Listing 2: Demo-Code aus dem Beispielprojekt

```
$ mvn clean install
$ java -jar target/demo-simple-1.0-SNAPSHOT.jar
initial list content
is…[6, 1, 3, 5]
after sorting: [1, 3, 5, 6]
ok
```

Listing 3: So wird das Beispielprojekt gebaut und ausgeführt

Es ist auf den ersten Blick vielleicht überraschend, dass frgaal Bytecode aus Java-15-Syntax erzeugen kann, der mit JDK 8 kompatibel ist. Das ist allerdings gar nicht so schwer, da JDK 9 bis 15 außer Module und Records keine Features enthält, die Bytecode-Änderungen erfordern würden. Das Schlüsselwort `var` ist rein syntaktisch, der kompilierte Code bleibt also derselbe. Das Gleiche gilt auch für `"""textblocks"""` Switch-Ausdrücke und `instanceof`. Auch diese erfordern keinen speziellen Bytecode. Die meisten dieser Funktionen verändern nur den Abstract Syntax Tree, sodass es einfach ist, JDK-8-kompatiblen Bytecode für den Generator auszugeben.

Was geht nicht?

Gegenwärtig gibt es folgende Einschränkungen für die Verwendung von frgaal:

- `module-info.java` kann nicht kompiliert werden bei Angabe von `--target 8`
- Records können nur mit `--target 15` verwendet werden
- Sealed Classes, die in Java 15 eingeführt wurden, können nur mit `--target 15` verwendet werden
- Alle APIs, die nach JDK8 eingeführt wurden, fehlen und müssen durch entsprechende Workarounds umgangen werden

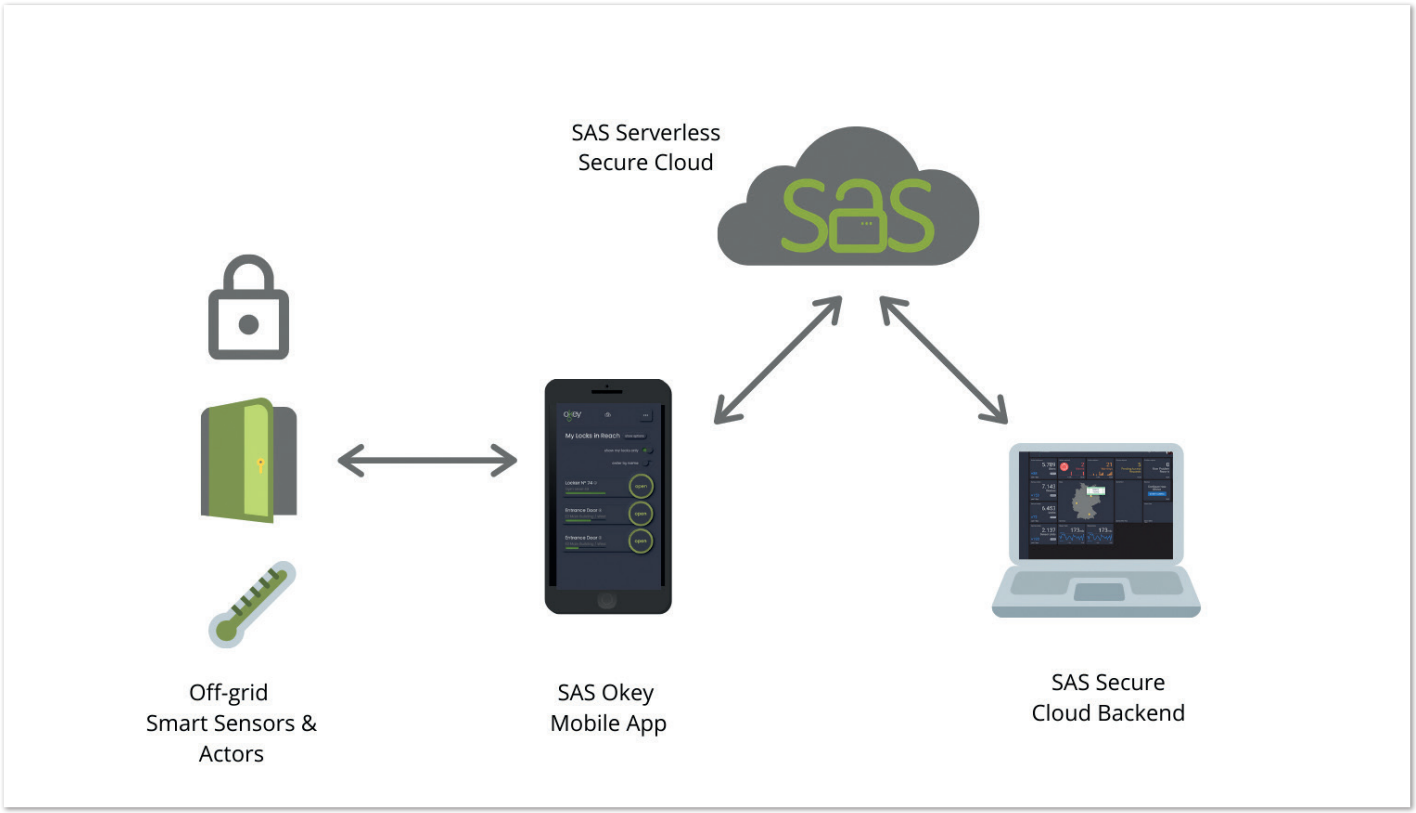


Abbildung 2: SAS setzt frgaal mit AWS Lambda ein, um Anwendungen im IoT-Bereich zu entwickeln (© Smart Access Solutions UG (haftungsbeschränkt))

```
$ cd aws-lambda-developer-guide/sample-apps/java-basic
$ git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

Listing 4: Klonen des Repository

```
java-basic$ ./1-create-bucket.sh
make_bucket: lambda-artifacts-a5e4xmplb5b22e0d
```

Listing 5: Erzeugen eines Bucket für Deployment-Artefakte

```
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
         <source>1.8</source>
         <target>1.8</target>
      </configuration>
    </plugin>
```

Listing 6: Eine Sektion aus der generierten pom.xml

frgaal für AWS Lambda

Schauen wir uns jetzt aber einen ganz konkreten Use Case an. In unserem Unternehmen Smart Access Solutions *[3]* bieten wir Lösungen für die zentrale Verwaltung von IoT-Sensoren und -Aktuatoren sowie Zugangsrechten für smarte Schlösser. Um unsere Anwendungen beliebig skalieren zu können, setzen wir dabei auf eine „serverlose" Architektur. Dabei ist es uns wichtig, Abhängigkeiten zu einem bestimmten Betreiber zu vermeiden. Darum halten wir unseren Business-Code durch Abstraktionen möglichst frei von anbieterspezifischen Bibliotheken. Leider waren wir jedoch anfangs gezwungen, bei der Java-Version auf den kleinsten gemeinsamen Nenner zwischen den Anbietern zu setzen.

AWS Lambda ist die beliebteste Umgebung für serverlose Funktionen. Aber derzeit bietet sie nur Laufzeiten für Java 8 und 11 *[4]*. Es ist etwas mehr als ein Jahr her, dass Amazon endlich seine Java-Lambda-Laufzeit auf Java 11 aktualisiert hat und seitdem ist nichts passiert. Offenbar werden nur noch LTS-Versionen unterstützt. Wenn der Trend so anhält, werden wir irgendwann im Jahr 2022 eine Java-15-Lambda-Laufzeit bekommen. Bis dahin müssen wir also entweder mit einem „uralten" SDK 11 arbeiten oder darauf warten, dass Amazon seine Runtime aktualisiert, bis wir all die schönen neuen Funktionen nutzen können, die in den Versionen 12, 13, 14 und 15 des SDK eingeführt wurden. Glücklicherweise gibt es jetzt frgaal als dritte Option.

Um neuere Funktionen zu nutzen, können Sie frgaal zu Ihrem Build hinzufügen und fröhlich drauflos programmieren. Lassen Sie uns dies mit den Beispielen aus dem AWS Developers Guide ausprobieren *[5]*. Es gibt ein GitHub-Repository mit Anweisungen zum Erstellen und Bereitstellen des Minimalbeispiels *[6]*. Das Beispiel verwendet nicht einmal die neueste Lambda-Laufzeit, sondern stattdessen die ältere Java-8-Runtime. Stellen Sie sicher, dass Sie die Anforderungen für die Ausführung des Beispiels erfüllen. Wir folgen der Anleitung, nehmen aber einige Änderungen vor, bevor wir die App bauen und bereitstellen. Folgen wir zunächst den Anweisungen für die Einrichtung. Zunächst klonen wir das Repository *(siehe Listing 4)*.

Anschließend wechseln wir in das Verzeichnis und erstellen per Skript einen Bucket für die Deployment-Artefakte *(siehe Listing 5)*.

Bevor wir bauen und deployen, werden wir die pom.xml ändern. Wenn Sie diese öffnen, finden Sie folgenden Abschnitt *(siehe Listing 6)*. Ersetzen Sie den Abschnitt aus *Listing 6*, um stattdessen frgaal zu verwenden, durch den Code in *Listing 7*.

Um zu testen, ob es funktioniert, werden wir nun auch den Code in Handler.java ändern und die Antwort durch einen mehrzeiligen Textblock ersetzen, der in Java 13 eingeführt wurden *(siehe Listing 8)*.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <dependencies>
        <dependency>
            <groupId>org.frgaal</groupId>
            <artifactId>compiler-maven-plugin</artifactId>
            <version>14.0.2</version>
        </dependency>
    </dependencies>
    <configuration>
        <compilerId>frgaal</compilerId>
        <source>14</source>
        <target>1.8</target>
        <compilerArgs>
            <arg>-Xlint:deprecation</arg>
            <arg>--enable-safe-preview</arg>
        </compilerArgs>
    </configuration>
</plugin>
```

Listing 7: Der modifizierte Abschnitt der pom.xml

Nun folgen wir wieder der originalen Anleitung, um unsere modifizierte Lambda-Funktion zu deployen *(siehe Listing 9)*. Danach können wir die neue Funktion aufrufen *(siehe Listing 10)*. Das Skript ruft die Funktion wiederholt auf. Sie können das mit der Tastenkombination CTRL+C abbrechen.

Das wars. Nur eine kleine Änderung in Ihrer Maven pom.xml und Sie können die neuesten Java-Funktionen in Ihrer Lambda-Funktion verwenden. Es besteht keine Notwendigkeit, Ihre eigene benutzerdefinierte Runtime zu entwickeln oder auf die nächste Version zu warten. Sie können Ihren Code auf dem neuesten Stand halten und müssen nicht alle neuen Entwicklungen nachholen, wenn eine neue Version herauskommt. Bei SAS verwenden wir genau diesen Ansatz und ich kann aus der Praxis bestätigen, dass frgaal hier problemlos und zuverlässig funktioniert *(siehe Abbildung 2)*.

Durch die Verwendung von frgaal binden Sie sich auch nicht dauerhaft. Sobald Ihre Ziel-Laufzeit verfügbar ist, können Sie frgaal einfach wieder entfernen oder Ihren Code mit noch neueren Funktionen weiter aktualisieren.

frgaal für Apache NetBeans

Zuletzt möchte ich noch einen weiteren interessanten, experimentellen Einsatzbereich für frgaal vorstellen. Seit langer Zeit verwendet die NetBeans-Entwicklungsumgebung den „nbjavac"-Compiler für das Parsing und Lexing, um Funktionen wie Syntaxfärbung, Codevervollständigung, Refactoring und dergleichen zur Verfügung zu stellen. Dieser Compiler ist ein Fork des normalen javac-Compiler und wird vom Oracle-JDK-Team entwickelt. Er hat einige Zusatzfeatures für den Einsatz in der IDE. Der Vorteil dieses Ansatzes ist, dass Fehlermeldungen und Warnungen während der Entwicklung exakt den javac-Compilermeldungen entsprechen und hier keine Überraschungen drohen.

Mit der Übertragung von NetBeans an die Apache Foundation ist es jedoch komplizierter geworden, nbjavac weiter zu nutzen und aktuell zu halten. Das JDK-Team hat kein Interesse, weiterhin einen separaten nbjavac-Fork zu pflegen. Um dieses Problem zu lösen, arbeiten sie derzeit daran, die fehlenden Features in den normalen javac-Compiler zu integrieren. Mit JDK 17 soll es dann so weit sein, dass nb-javac nicht mehr benötigt wird.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import java.util.Map;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String>{
  Gson gson = new GsonBuilder().setPrettyPrinting().create();
  @Override
  public String handleRequest(Map<String,String> event, Context context)
  {
    LambdaLogger logger = context.getLogger();
        String response = """
            200 OK
            Look a this multiline text block!
            ...""";
    // log execution details
    logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
    logger.log("CONTEXT: " + gson.toJson(context));
    // process event
    logger.log("EVENT: " + gson.toJson(event));
    logger.log("EVENT TYPE: " + event.getClass().toString());
    return response;
  }
}
```

Listing 8: Der modifizierte Handler mit Java-13-Textblock

```
java-basic$ ./2-deploy.sh mvn
[INFO] Scanning for projects...
[INFO] ------------------------< com.example:java-basic >------------------------
[INFO] Building java-basic-function 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
```

Listing 9: Deployment der Lambda-Funktion

```
java-basic$ ./3-invoke.sh
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
"200 OK\nLook at this multiline text\n..."
```

Listing 10: Deployment der Lambda-Funktion

Anders als andere Entwicklungsumgebungen wäre NetBeans dann aber nicht mehr in der Lage, auf einer älteren JRE zu laufen und trotzdem die neuesten Sprachfeatures zu bieten. Hier kann frgaal helfen. Ein Pull Request des NetBeans-Gründers Jaroslav Tulach zeigt, wie der JDK-17-javac-Compiler auf Java 8 zurückportiert werden kann *[7]*. Zunächst werden etwa 30 automatische Refactoring-Regeln mit dem JackPot Refactoring Tool *[8]* auf den Quellcode des javac-Compilers angewendet, um die Fälle zu behandeln, für die frgaal nicht ausreicht. Das sind insbesondere Aufrufe von APIs, die nach JDK 8 eingeführt wurden. Danach wird frgaal verwendet, um den modifizierten Quellcode zu kompilieren und die Verwendung neuerer Sprachfeatures rückzuportieren. Das Ergebnis ist ein JDK-17-javac-Compiler, der auf JDK 8 läuft und alle JDK-17-Sprachfeatures unterstützt. NetBeans hätte also einen – jetzt komplett automatisch generierten – neuen „nbjavac"-Compiler und könnte weiterhin auch auf alten JDKs die neuesten Sprachfeatures unterstützen.

Zusammenfassung

Das frgaal-Projekt ermöglicht es Ihnen, neue Java-Sprachfunktionen auf eine sehr einfache und bequeme Weise auszuprobieren oder zu verwenden. Es ist der schnellste Weg, um neue Java-Syntax und Sprachfunktionen auszuprobieren. Noch wichtiger ist, dass es eine Möglichkeit ist, diese Features in Ihren realen Projekten zu verwenden und Ihre Java-Kenntnisse aktuell zu halten, selbst wenn Sie auf eine ältere Runtime beschränkt sind, wie zum Beispiel in einer serverlosen Umgebung.

Das Projekt ist unter der GPL v2.0 + Classpath Exception verfügbar und kann kostenlos verwendet werden, um Ihre kommerziellen, privaten oder quelloffenen Projekte zu kompilieren. Auf der Projektseite finden Sie weitere aktuelle Informationen und detaillierte Anleitungen zum Konfigurieren des Compilers oder zur Verwendung auf der Kommandozeile.

Quellen

[1] *https://frgaal.org*
[2] *https://babeljs.io*
[3] *https://www.smart-access-solutions.com/de/*
[4] *https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html*
[5] *https://github.com/awsdocs/aws-lambda-developer-guide/tree/master/sample-apps/java-basic*
[6] *https://github.com/awsdocs/aws-lambda-developer-guide/tree/master/sample-apps/java-basic*
[7] *https://github.com/oracle/nb-javac/pull/12*
[8] *https://netbeans.apache.org/jackpot/HintsFileFormat.html*

Anton Epple
Eppleton IT Consulting/
Smart Access Solutions UG (haftungsbeschränkt)
toni.epple@eppleton.de

Anton "Toni" Epple ist Java-Champion, JavaONE Rockstar, Apache Committer für das NetBeans-Projekt sowie Autor mehrerer Java-Bücher und organisiert neben seinem Job als Java Consultant und CTO eines Technologie-Startups Trainings und Konferenzen rund um Java und Programmierthemen.

Automatisches Nachladen von Java-Anwendungskonfigurationen in Kubernetes

Frank Rosner, Datastax

Dieser Artikel demonstriert eine Möglichkeit, wie man Anwendungskonfigurationen von auf Kubernetes bereitgestellten Java-Programmen automatisch und ohne Neustart nachladen kann. Dabei werden ausschließlich Kubernetes-Bordmittel sowie Dateisystemoperationen verwendet, sodass keine Abhängigkeiten zu externen Konfigurationsspeichern notwendig sind.

Konfiguration von Zwölf-Faktor-Apps auf Kubernetes

Anwendungen, die der Zwölf-Faktor-Methodologie *[1]* folgen, sind üblicherweise unkompliziert zu betreiben. Der dritte Faktor umfasst das Thema Konfiguration. Zwölf-Faktor-Apps sollten eine strikte Trennung zwischen Konfiguration und Code vornehmen, sodass sie problemlos in verschiedene Umgebungen ausgeliefert werden können. Idealerweise werden alle Konfigurationsparameter in Form von Umgebungsvariablen gespeichert, da diese programmiersprachen- und betriebssystemunabhängig sind.

Im Kubernetes-Umfeld zieht eine Änderung der Umgebungsvariablen eines Deployment zwangsläufig einen rollenden Neustart aller Pods nach sich. Der Grund hierfür liegt in der Tatsache, dass Umgebungsvariablen in Linux beim Start eines Prozesses übergeben werden. Diese lassen sich nicht ohne Weiteres ändern, ohne den Prozess neu zu starten.

In manchen Fällen gibt es jedoch besondere Qualitätsanforderungen, die es erforderlich machen, Komponentenneustarts zu minimieren. Nehmen wir als Beispiel eine zustandsbehaftete, hochverfügbare Anwendung, etwa eine verteilte Datenbank. Will man solche Dienste neu starten, müssen bestehende Verbindungen gekappt, Caches wieder neu aufgebaut und gegebenenfalls sogar Zustände migriert werden.

Glücklicherweise gibt es andere Wege, Änderungen an Anwendungskonfigurationen vorzunehmen, die keinen Neustart erfordern. In Kubernetes kann man beispielsweise die Konfiguration in einer `ConfigMap` speichern und diese stattdessen als Umgebungsvariable in Form einer Konfigurationsdatei ins Dateisystem des Containers einbinden. Aktualisiert man die `ConfigMap`, wird Kubernetes letztendlich die Datei aktualisieren und die Anwendung kann die neue Konfiguration laden.

Dieser Artikel soll aufzeigen, wie man einen solchen Mechanismus innerhalb einer Java-Anwendung umsetzen kann, die dann mithilfe von Kubernetes bereitgestellt wird. Im nächsten Abschnitt werden wir eine Beispielanwendung implementieren, die in der Lage ist, Änderungen in Konfigurationsdateien automatisch nachzuladen. Anschließend beleuchten wir, wie diese Anwendung in Kubernetes gestartet werden kann. Der Artikel schließt mit einem Diskussionsteil und einer Zusammenfassung. Der gesamte Quelltext kann auf GitHub eingesehen werden *[2]*.

Implementierung der Java-Anwendung

Wenn es um Konfigurationsverwaltung im JVM-Ökosystem geht, gibt es viele verschiedene Bibliotheken und Frameworks. Einer der „alten Hasen" im Geschäft ist Apache Commons Configuration *[3]*. Diese Bibliothek bietet generische Schnittstellen zur Verwaltung von Java-Anwendungskonfigurationen aus unterschiedlichen Quellen an.

Apache Commons Configuration unterstützt auch das automatische Nachladen von Konfigurationsquellen. Wir werden uns diese Funktionalität zunutze machen, um die Änderungen der in den Container eingebunden Konfigurationsdateien innerhalb unserer Anwendung verfügbar zu machen.

Zunächst definieren wir eine `ConfigReloader`-Klasse, die das periodische Nachladen der Konfiguration kapselt. Diese stellt eine Methode bereit, um die aktuelle Konfiguration abzurufen. Hierfür benötigen wir zwei Zutaten aus Apache Commons Configuration: `ReloadingFileBasedConfigurationBuilder` und `PeriodicReloadingTrigger`.

Der `ReloadingFileBasedConfigurationBuilder` ist dafür verantwortlich, die Konfigurationsdatei zu laden. In unserem Fall werden wir eine Java-Properties-Datei verwenden. Der `PeriodicReloadingTrigger` ruft den Builder in regelmäßigen Abständen auf, sodass dieser prüft, ob sich die Konfigurationsdatei seit dem letzten Aufruf verändert hat. Im Falle einer Änderung wird die Datei erneut gelesen und die geladene Konfiguration aktualisiert. *Listing 1* zeigt den Quelltext der `ConfigReloader`-Klasse.

Es gilt zu beachten, dass die von `builder.getConfiguration()` zurückgegebene Konfiguration unveränderlich ist. Somit wird garantiert, dass eine einmal abgerufene Konfiguration sich während der Verwendung nicht unvermittelt ändern kann. Dies bedeutet jedoch auch, dass wir bei jeder Konfigurationsabfrage den Builder verwenden müssen, da wir sonst keine Aktualisierungen erhalten. Der Builder hält intern eine Referenz zur aktuellen Konfiguration, die nur aktualisiert wird, wenn eine neue Datei gelesen wurde.

Als Nächstes entwickeln wir eine `main`-Methode, die in regelmäßigen Abständen eine Ausgabe tätigt. Dabei wollen wir die Abstände

```
import java.io.File;
import java.util.concurrent.TimeUnit;
import org.apache.commons.configuration2.Configuration;
import org.apache.commons.configuration2.FileBasedConfiguration;
import org.apache.commons.configuration2.PropertiesConfiguration;
import org.apache.commons.configuration2.builder.ReloadingFileBasedConfigurationBuilder;
import org.apache.commons.configuration2.builder.fluent.Parameters;
import org.apache.commons.configuration2.ex.ConfigurationException;
import org.apache.commons.configuration2.reloading.PeriodicReloadingTrigger;

public class ConfigReloader implements AutoCloseable {

  private final PeriodicReloadingTrigger trigger;
  private final ReloadingFileBasedConfigurationBuilder<FileBasedConfiguration> builder;

  public ConfigReloader(String configFilePath) {
    Parameters params = new Parameters();
    File propertiesFile = new File(configFilePath);
    builder = new ReloadingFileBasedConfigurationBuilder<FileBasedConfiguration>(
        PropertiesConfiguration.class)
        .configure(params.fileBased().setFile(propertiesFile));
    trigger = new PeriodicReloadingTrigger(
        builder.getReloadingController(),
        null, 1, TimeUnit.SECONDS);
    trigger.start();
  }

  public Configuration getConfig() {
    try {
      return builder.getConfiguration();
    } catch (ConfigurationException cex) {
      throw new RuntimeException(cex);
    }
  }

  @Override
  public void close() {
    trigger.stop();
  }
}
```

Lising 1

```
import java.util.Date;
import org.apache.commons.configuration2.Configuration;

public class App {
  public static void main(String[] args) throws InterruptedException {
    try (ConfigReloader configReloader = new ConfigReloader("/config/config.properties")) {
      while (true) {
        Configuration config = configReloader.getConfig();
        String name = config.getString("name");
        int sleepInterval = config.getInt("sleepIntervalMillis");
        System.out.println(String.format("Hello %s, it is %s", name, new Date()));
        Thread.sleep(sleepInterval);
      }
    }
  }
}
```

Lising 2

und die Nachricht konfigurierbar machen. Zunächst müssen wir einen `ConfigReloader` initialisieren und dabei den Pfad zur Konfigurationsdatei übergeben. Diese wird später in den Container eingebunden. Anschließend geben wir in einer Endlosschleife immer wieder eine Grußnachricht auf den Standardausgabestrom aus. *Listing 2* zeigt die Klasse, die die `main`-Methode enthält.

Bereitstellung in Kubernetes

Um unsere Anwendung in Kubernetes bereitzustellen, müssen wir sie zunächst in ein Docker-Image einbetten. Hierfür können wir das Jib-Maven-Plug-in verwenden. *Listing 3* zeigt, wie dieses in die `pom.xml` eingebunden wird.

Um unsere Anwendung auszuprobieren, werden wir uns Minikube *[4]* zunutze machen. Minikube ist eine lokale, aus einem Knoten bestehende, VM-basierte Kubernetes-Installation. Der Befehl `minikube start` startet den Minikube. Mit `eval $(minikube docker-env)` setzen wir die Umgebungsvariable `DOCKER_HOST`, sodass diese auf den Docker-Daemon in unserem Cluster zeigt. Anschließend erzeugen wir mit `mvn compile jib:dockerBuild` das Docker-Image.

Als Nächstes erstellen wir die entsprechenden Kubernetes-Objekt-Manifeste. Wir benötigen eine `ConfigMap`, die unsere Properties-Datei enthält *(siehe Listing 4)*, sowie einen `Pod` für unsere Anwendung *(siehe Listing 5)*.

Das `Pod`-Manifest beinhaltet neben der Containerdefinition auch ein `ConfigMap`-Volume, das über ein Mount eingebunden wird. Es sei an dieser Stelle darauf hingewiesen, dass man Pods üblicherweise nicht direkt anlegt, sondern stattdessen zum Beispiel von einem `Deployment` verwalten lässt. In diesem Artikel wird darauf jedoch verzichtet, um die Komplexität gering zu halten.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8s-java-config-reload-configmap
data:
  config.properties: |-
    name=Frank
    sleepIntervalMillis=1000
```

Lising 4

```
<plugin>
  <groupId>com.google.cloud.tools</groupId>
  <artifactId>jib-maven-plugin</artifactId>
  <version>3.0.0</version>
  <configuration>
    <to>
      <image>k8s-java-config-reload</image>
    </to>
  </configuration>
</plugin>
```

Lising 3

Da nun alle benötigten Objekte definiert sind, müssen wir diese nur noch in Kubernetes bereitstellen. *Listing 6* zeigt in der linken Spalte die entsprechenden Kommandozeilenbefehle, um die `ConfigMap` und den `Pod` bereitzustellen. Anschließend ändern wir die Konfiguration, indem wir den Namen auf „James" setzen und das Ausgabeintervall halbieren. Die rechte Spalte zeigt die Ausgabe der Anwendung.

Diskussion

Wie man in der rechten Spalte von *Listing 6* gut sehen kann, dauert es einige Zeit, bis die Änderungen der Konfiguration vollständig propagiert werden. Der Grund dafür liegt darin, dass der Kubernetes-Agent `kubelet` eingebundene `ConfigMap`-Volumes standardmäßig minütlich synchronisiert (siehe `--sync-frequency` *[5]*).

```
apiVersion: v1
kind: Pod
metadata:
  name: k8s-java-config-reload-pod
spec:
  containers:
    - image: k8s-java-config-reload
      name: k8s-java-config-reload
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: config-volume
          mountPath: /config
  volumes:
    - name: config-volume
      configMap:
        name: k8s-java-config-reload-configmap
  restartPolicy: Always
```

Lising 5

Kommandozeile	Container Logs (kubectl logs k8s-java-config-reload-pod -f)
kubectl apply -f k8s-java-config-reload-pod.yaml kubectl apply -f k8s/k8s-java-config-reload-pod.yaml kubectl edit configmap k8s-java-config-reload-con- figmap # Name und Sleep Interval ändern # Kubernetes aktualisiert ConfigMap Volume # Anwendung lädt neue Konfiguration	Hello Frank, it is Sun May 02 11:53:03 GMT 2021 Hello Frank, it is Sun May 02 11:53:04 GMT 2021 Hello Frank, it is Sun May 02 11:53:05 GMT 2021 Hello Frank, it is Sun May 02 11:53:06 GMT 2021 Hello Frank, it is Sun May 02 11:53:07 GMT 2021 Hello Frank, it is Sun May 02 11:53:08 GMT 2021 Hello Frank, it is Sun May 02 11:53:09 GMT 2021 ... Hello Frank, it is Sun May 02 11:54:19 GMT 2021 Hello Frank, it is Sun May 02 11:54:20 GMT 2021 Hello Frank, it is Sun May 02 11:54:21 GMT 2021 Hello James, it is Sun May 02 11:54:22 GMT 2021 Hello James, it is Sun May 02 11:54:22 GMT 2021 Hello James, it is Sun May 02 11:54:23 GMT 2021 Hello James, it is Sun May 02 11:54:23 GMT 2021

Lising 6

Hinzu kommt, dass `ConfigMap`-Werte in einem Cache gespeichert werden, der erst invalidiert werden muss, bevor die Änderungen im Container sichtbar werden. Natürlich trägt auch noch das periodische Nachladen in der Java-Anwendung zur Verzögerung bei. Man kann den Prozess beschleunigen, indem man die `Pod`-Annotationen aktualisiert. Dadurch startet `kubelet` eine sofortige Synchronisierung. Verwendet man eine Templating-Lösung zum Definieren der Manifeste (zum Beispiel Helm), so könnte man einen Hash des `ConfigMap`-Inhaltes als `Pod`-Annotation speichern, der automatisch aktualisiert wird, sobald sich die `ConfigMap` verändert.

Müssen Konfigurationsänderungen noch unmittelbarer propagiert werden, kann auf andere Techniken zurückgegriffen werden. Entwickler können, anstatt auf eine gemountete Properties-Datei zurückzugreifen, einen Key-Value-Store wie Consul, etcd oder AWS Systems Manager Parameter Store verwenden. Diese Lösungen geben zwar eine direktere Kontrolle über Konfigurationsänderungen, bringen jedoch auch neue Herausforderungen mit sich.

Zum einen benötigen sie zusätzliche Infrastrukturkomponenten. Zum anderen kann es aufwendiger sein, solche Konfigurationen in einem Versionskontrollsystem unterzubringen, da gegebenenfalls spezielle Werkzeuge wie dedizierte Terraform-Provider notwendig sind. Hinzu kommt, dass die Anwendung nun auch spezielle Protokolle verwenden muss, um die Konfiguration laden zu können.

Ein weiterer Anwendungsfall, bei dem das Einbinden von `ConfigMap`-Volumes nicht die beste Lösung ist, besteht beim Laden von Geheimnissen, wie beispielsweise Zugangsdaten. Der `ConfigMap`-Mechanismus lässt sich praktischerweise jedoch eins zu eins auf `Secret`-Objekte übertragen. Dennoch kann es aus verschiedenen Gründen notwendig sein, ein externes Werkzeug zur Geheimnisverwaltung zu verwenden, das die gleichen Vor- und Nachteile wie eine externe Konfigurationsverwaltung mit sich bringt.

Zusammenfassung

Zusammenfassend lässt sich festhalten, dass die Zwölf-Faktor-Methodologie zu bevorzugen ist, insofern es keine besonderen Qualitätsanforderungen gibt, die ein Abweichen rechtfertigen. Umgebungsvariablen sind eine einfache, programmiersprachen- und betriebssystemunabhängige Möglichkeit, Anwendungen zu konfigurieren.

Müssen Konfigurationsanpassungen ohne Anwendungsneustart propagiert werden, bieten `ConfigMap`-Volumes in Kombination mit anwendungsseitigem Nachladen eine einfache Umsetzungsmöglichkeit. Der Vorteil dieser Lösung liegt darin, dass sie ausschließlich auf Kubernetes-internen Mechanismen und Dateisystemoperationen basiert. Bestehen speziellere Anforderungen, so kann auf externe Konfigurations- und Geheimnisspeicher zurückgegriffen werden.

Quellen

[1] *https://12factor.net/de/*
[2] *https://github.com/FRosner/k8s-java-config-reload*
[3] *https://commons.apache.org/proper/commons-configuration/index.html*
[4] *https://minikube.sigs.k8s.io/docs/*
[5] *https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/*

Frank Rosner
Datastax
frank.rosner@datastax.com

Franks Interessen liegen im Bereich von Big Data, Machine Learning, Cloud-Native-Anwendungen und Softwareentwicklung. Er liest gerne Paper und Quelltexte, um zu verstehen, wie die Dinge funktionieren, die er verwendet. Auf der JVM entwickelt er in Java, Kotlin und Scala.

Grundlagen der Datenwissenschaft und künstliche Intelligenz für JVM-Entwickler

Michal Harakal, Deutsche Telekom AG

Die Softwareentwicklung ist schon seit Längerem keine Angelegenheit nur einer Programmiersprache oder eines Frameworks. Mehrere unterschiedliche Fähigkeiten und die vollgepackte Werkzeugkiste eines Softwareentwicklers bieten viele Vorteile in der Welt der Datenwissenschaften. Auch umgekehrt kann die Softwareentwicklung von den neuesten Erkenntnissen und Fortschritten in den Bereichen der künstlichen Intelligenz und Datenwissenschaft profitieren. In diesem Artikel schauen wir uns aus der Perspektive eines Softwareentwicklers mit Erfahrungen mit Java und JVM-Sprachen an, was sich hinter den Begriffen versteckt und wie man davon profitieren kann.

Datenwissenschaft und künstliche Intelligenz (KI) sind nicht mehr nur Begriffe aus der Sci-Fi. Sie haben einen immer größeren Einfluss auf unser tägliches Leben. Ob es sich um eine Risikobewertung bei einer Kreditanfrage, eine Produkt-Empfehlung in einem beliebten Online-Handel oder einen Chatbot im Web oder am Telefon handelt – es sind Systeme, die bereits Techniken der Datenwissenschaft und der künstlichen Intelligenz nutzen. Auch Technologie-Konzerne aus Kalifornien zeigen, dass KI-basierte Funktionen in deren Produkten eine immer wichtigere Rolle spielen, dass sie bereits zur Grundausstattung gehören oder schon sogar Teil eines Betriebssystems sind. Eine Handykamera mit KI-gestützten Funktionen, wie zum Beispiel Personenerkennung in Echtzeit, ist nicht mehr wegzudenken.

Die globale Vernetzung, unzählige Sensoren, Kameras, von Menschen produzierte Texte und Videos im Internet sind massive Datenquellen. Die Daten und der Anstieg der Rechenleistung gehen Hand in Hand mit dem erheblichen Fortschritt, sodass man die Erkenntnisse auch in der Praxis anwenden kann. Sie ermöglichen nicht nur neue und innovative Funktionen, Anwendungen und Dienstleistungen, sie dienen auch als Alleinstellungsmerkmal. Sie können sogar zu einem generellen Paradigmenwechsel in der Softwareentwicklung führen (manchmal auch als Software 2.0 bezeichnet). Bei der traditionellen Programmierung werden Regeln in Form von Quelltext definiert und aus gegebenen Daten werden neue Resultate berechnet. Dieser Ansatz hat eigentlich sehr lange sehr gut funktioniert. Das maschinelle Lernen dreht diesen Ansatz ein bisschen um. Aus Daten und bekannten Antworten versucht man die Regeln abzuleiten und diese auf neue Daten anzuwenden. Der Unterschied ist, dass es nicht der Entwickler ist, der die Regeln herausfindet, sondern der Rechner. Dies kann sehr hilfreich sein, zum Beispiel bei Anwendungen, bei denen es für einen Menschen schwer ist, die Regeln abzuleiten, oder sich Regeln mit der Zeit verändern (etwa ein Spamfilter, der auf immer kreativere Versuche reagieren soll). Damit es funktioniert, benötigt man eine große Menge an Daten und Rechenleistung. Ein Vorteil ist, dass man die Modelle trainieren kann und dabei tatsächlich ein Lernprozess stattfindet. Die beiden Ansätze kann man vereinfacht wie in *Abbildung 1* darstellen.

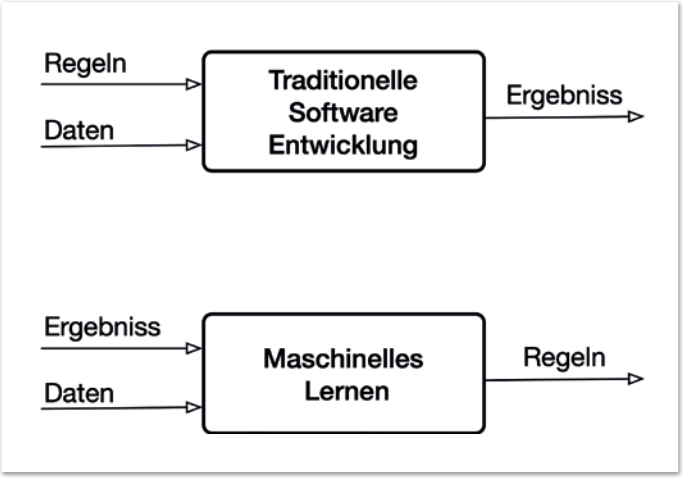


Abbildung 1: Traditionelle Softwareentwicklung versus maschinelles Lernen (© Michal Harakal, 2021)

Softwareentwickler und Datenwissenschaft

Wenn wir versuchen, die Datenwissenschaft (Data Science) mit einer Sprache der Mengenlehre zu beschreiben, könnte man sie vereinfacht als eine Vereinigungsmenge von Techniken und Theorien aus den Fächern Mathematik, Statistik und Informationstechnologie beschreiben. Die Informationstechnologie und die Softwareentwicklung spielen gerade in der Praxis eine sehr wichtige Rolle. Angefangen von den Praktiken, die in der Softwareentwicklung gang und gäbe sind, wie das Schreiben von sauberem und gut lesbarem Quelltext und Test-getriebener Entwicklung, über Versionsverwaltung bis hin zu DevOps, CI/CD oder agilen Methodologien.

Wie wichtig die Fähigkeit ist, Daten mit einer Datenbank zu schreiben und zu lesen, um zum Beispiel Testresultate vom Training zu protokollieren, braucht man nicht lange auszudiskutieren. Somit überrascht es auch nicht, dass das Aufsetzen einer quelltextbasierten Pipeline in einer Cloud-Umgebung seiner Wahl auch zu den Aufgaben eines Datenwissenschaftlers gehört. Ein weiteres interessantes Gebiet, bei dem man als Entwickler Vorteile hat, ist die Integration in bestehende oder neue Produkte. Ob man ein neuronales Netzwerk zur Inferenz über REST-API Microservices einsetzt, ein auf KI basierendes Modell zur Suche in einen ELK-Stack einbaut oder die Erkennung eines Weckworts in einen Mikrocontroller implementiert – ein Softwareentwickler ist auch hier im Vorteil.

Jeder Entwickler betreibt bereits ein bisschen Datenwissenschaft

Eine datenbasierte Entscheidung, etwa, welche neuen Features demnächst im Projekt entwickelt werden sollen, eine Log-Auswertung oder eine Visualisierung des Speicherverbrauchs – all das sind auch Methoden der Datenwissenschaft. Techniken, um Datenpunkte bereitzustellen, zu serialisieren, zu speichern oder zu verarbeiten, gehören bereits in die Werkzeugkiste eines Entwicklers. Umso interessanter und wichtiger ist es, weitere Methoden und Werkzeuge zu lernen und auszuprobieren, die in der Datenwissenschaft und bei der künstlichen Intelligenz bekannt und im Einsatz sind.

Der Open-Source-Gedanke, der in der Softwareentwicklung bekannt ist, lebt auch bei der Datenwissenschaft weiter. Nicht nur der Quelltext, der zum Berechnen in wissenschaftlichen Publikationen benutzt wird, wird häufig unter Open-Source-freundlicher Lizenz veröffentlicht *[1]*, auch Datensätze oder trainierte neuronale Netze werden oft zur Verfügung gestellt. Dafür sind sogar Online-Plattformen wie HuggingFace *[2]* oder Tensorflow Hub *[3]* entstanden, die, ähnlich wie GitHub, einen Austausch ermöglichen. Die wichtigsten Frameworks, die beim maschinellen Lernen zum Einsatz kommen, werden zwar von Technologiekonzernen maßgeblich vorangetrieben, aber unter freier Lizenz entwickelt und verteilt.

Keine Angst vor Mathematik

Die Mathematik spielt eine bedeutende Rolle in der Datenwissenschaft. In der Praxis muss man aber selten komplizierte Formeln lösen oder eine Implementierung zur Lösung entwickeln können. Es ist jedoch von Vorteil, wenn der Entwickler ein tieferes Verständnis hat, wie eine Bibliothek funktioniert, die man zum Lösen einer bestimmten Aufgabe verwendet. Man sollte wissen, welche Stellschrauben man mit dem Parameter X justieren kann, was die Zahlen bei Metriken in der Validierung der Ergebnisse bedeuten und ob es besser ist, wenn eine Zahl größer oder kleiner wird. Gerade im praktischen Einsatz könnte der eine oder andere doch die Schönheit der sonst grauen Theorie entdecken – was zu Begeisterung und neuem

Interesse für weitere Erkenntnisse, Methoden und Formeln führen kann. Man lernt nie aus – das gilt hier noch mehr.

Lernen, um zu lernen

Die Datenwissenschaft ist ein sehr breites Gebiet, das mehrere Fächer zusammenbringt. Die schiere Menge an Neuem könnte einen schon abschrecken. Nicht immer hat man die Freiheit, sich komplett auf das Neue zu fokussieren, die Berufstätigkeit und soziale Verpflichtungen machen das nicht einfacher.

Obwohl viele ihren Lerntyp bereits kennen, empfiehlt es sich, die neuen Erkenntnisse der Neurowissenschaften anzuschauen und eventuell auch etwas Neues auszuprobieren – ob man sich das Smartphone-gestützte Kärtchen-System Anki anschaut oder Techniken zum gezielten fokussierten Lernen probiert *[4]*. Ebenso können Meditationsübungen, eine gesündere Ernährung, die kognitive Fähigkeiten des Gehirns fördert, Methoden zur Reduzierung der Prokrastination und Methoden zur Motivation erheblich helfen.

Es ist sehr erfreulich festzustellen, dass Lernmaterial in unterschiedlichen Formaten in sehr guter Qualität frei zur Verfügung steht. Zum Beispiel YouTube-Videos oder Aufnahmen von Vorlesungen unterschiedlicher Universitäten, die in dem jeweiligen Themenbereich als Vorreiter bei der Forschung und des Publizierens gelten.

Maschinelles Lernen

Inzwischen überrascht es niemanden mehr, wenn ein neuer Rechner mit einem Prozessor angepriesen wird, der dedizierte „Neural Cores" enthält oder Smartphone-Kameras schon automatisch Korrekturen im Bild vornehmen, die durch ein oder mehrere neuronale Netzwerke mit dedizierter Hardware berechnet worden sind. Chatbots im Web oder eine intelligente Textvervollständigung in einem Smartphone sind weitere Beispiele, bei denen Techniken der künstlichen Intelligenz zum Einsatz kommen. Computer Vision und das maschinelle Verarbeiten von durch den Menschen produzierten Texten, Audioaufnahmen und Videos haben in den letzten Jahren eine wahre Renaissance erlebt. Auch, weil es immer besser funktioniert. Es gibt Anwendungen, bei denen Maschinen schon den Menschen ersetzen können oder bessere Ergebnisse liefern.

So ein System zu bauen, erfordert viele Schritte und bei jedem ist ein Entwickler gefragt – angefangen bei der Datenvorbereitung über das Trainieren von Modellen bis hin zum Betrieb in einer Cloud oder als App.

Das Streben, Bilder maschinell zu bearbeiten, die Platzierung von Objekten zu erkennen oder die Klassifizierung von Objekten ist nicht neu. Auch die Grundprinzipien eines neuronalen Netzes sind nicht neu. Das Perzeptron, ein vereinfachtes künstliches neuronales Netz, wurde schon 1957 und das tiefe vorwärts gerichtete Konvolutionsnetz schon in der 80ern vorgestellt. Nach einiger Zeit der „KI-Eiszeit" wurden neuronale Netze wiederbelebt und der Boom brachte neue Typologien der Verbindungsnetzwerke, wie zum Beispiel ein rekurrentes neuronales Netz und große beschriftete Datensammlungen zum Trainieren, mit sich. Das alles steht in Verbindung mit der Rechenleistung.

Ein einfaches neuronales Netz kann man relativ einfach in der Programmiersprache seiner Wahl umsetzen. Zum Erlernen ist dies jedenfalls ein empfehlenswerter Schritt. Ein neuronales Netz besteht aus Neuronen (ähnlich einem menschlichen Neuron), die miteinander verbunden sind. Die Art der Verbindung ergibt sich aus der gewünschter Topologie-Architektur. Üblich ist, dass die Neuronen in Schichten organisiert sind. Eine schematische Darstellung eines Neurons ist in *Abbildung 2* zu finden. Das Trainieren selbst findet in mehreren Schritten statt, die sich wiederholen.

Im ersten Schritt werden die Eingangsdaten durch einen Faktor (Gewicht) auf das jeweilige Neuron skaliert, um einen Verstärkungsfaktor korrigiert und dann das Ergebnis mit einer sogenannten Aktivierungsfunktion auf nachfolgende Neuronen weitergeleitet. Diesen Schritt nennt man „Forward Propagation". Das Ergebnis wird mit einem gewünschten Wert verglichen und durch eine Verlustfunktion ausgewertet. Der dabei entstandene Fehler wird mithilfe von

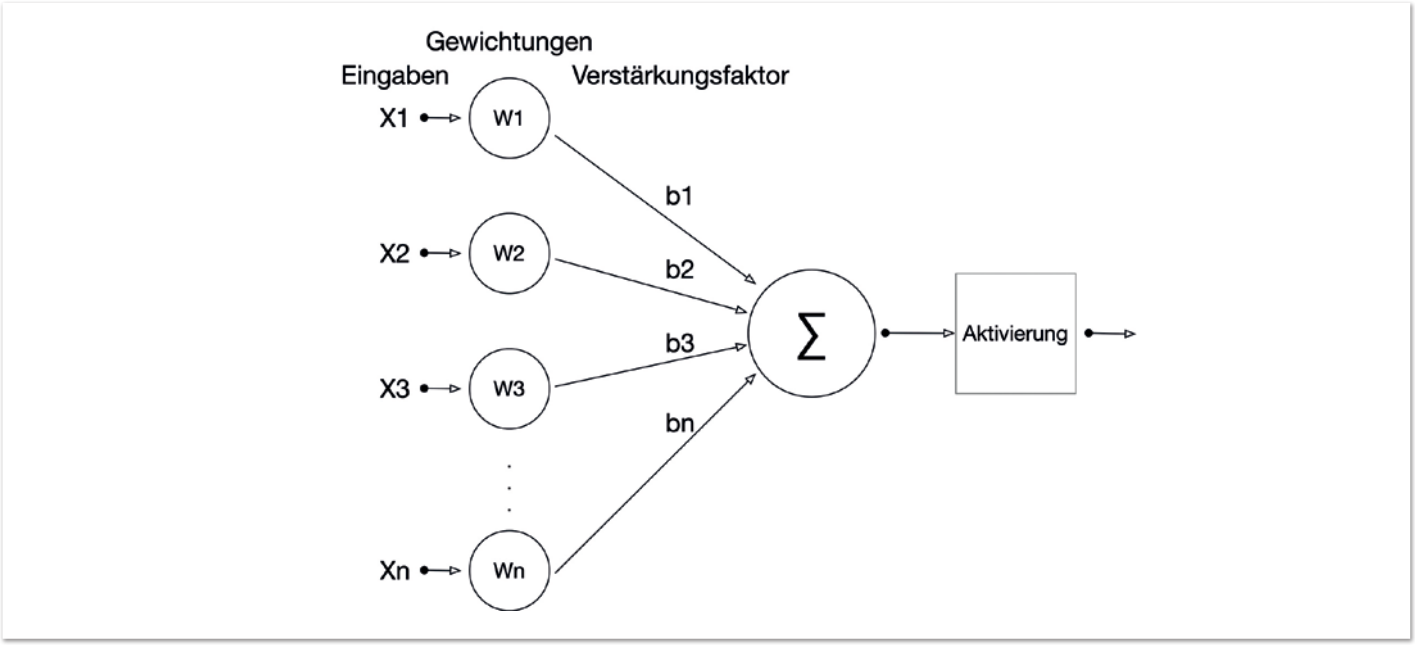


Abbildung 2: Neuron, eine schematische Darstellung (© Michal Harakal, 2021)

mathematischen Regeln zurückgerechnet und die Gewichtungen sowie Verstärkungsfaktoren werden angepasst, um das Ergebnis der Verlustfunktion zu minimieren. Dieser Schritt nennt sich „Backward Propagation" und kann wiederholt werden, bis ein gewünschtes Ergebnis erzielt wird oder bis durch Ablauf der Verlustfunktion oder andere Metriken klar ist, dass keine weitere Verbesserung möglich ist. Mit diesem Prozess der Selbstanpassung der Parameter des Netzes wird der Lerneffekt erreicht, dass das neuronale Netz selbstständig die Regeln lernt.

Die gleiche Berechnung, die man im ersten Schritt bei der „Forward Propagation" anstellt, führt man bei der Inferenz durch, wenn wir für neue Eingabewerte das Ergebnis durch das neuronale Netz berechnen lassen. In *Listing 1* kann man eine einfache Implementierung der `predict`-Funktion in Kotlin sehen. Eine Implementierung der `Matrix`-Klasse samt Manipulationsmethoden dürfte sehr schnell von einem Entwickler machbar sein. Die vollständige Implementierung befindet sich auf GitHub *[5]*.

Das Trainieren eines eigenen neuronalen Netzwerks erfordert eine spezialisierte Hardware. Allzweck-CPUs eignen sich dafür nur begrenzt, trotz enormem Leistungszuwachs können sie, zum Beispiel mit den Grafikkarten, nicht mithalten. Des Weiteren werden zum Berechnen spezialisierte Recheneinheiten entwickelt, die nicht nur sehr schnell sind und weniger Strom verbrauchen, sondern auch eine massive Parallelisierung ermöglichen. Das alles ist nur in einem Zusammenspiel der Hardware und Software möglich, mit der neue Programmiersprachen und komplett neue Frameworks entwickelt werden. Als Beispiel dient die Implementierung des beliebten Frameworks Tensorflow in Swift, das eher als die Programmiersprache zum Entwickeln von Apps für iOS bekannt ist, oder eine neue Generation von Prozessoren mit RISC-V-Architektur. Mit dem Ansatz „Ich habe es immer so gemacht!" kann man nicht lange mithalten.

Python und die JVM-Sprachen

Python ist und bleibt vermutlich noch für einige Zeit die unumstrittene Nummer eins als Programmiersprache der Datenwissenschaften. Gerade bei Java könnte der Umstieg von einer statischen, kompilierten und stark typisierten Sprache zu einer dynamischen Skriptsprache zu einer Vielzahl angenehmer oder unangenehmer Überraschungen führen. Auch hier werden gerne leidenschaftliche Diskussionen der Art „Emacs versus Vi" geführt und auch hier zählen nur die Ergebnisse und die Erfolge, nicht der Quelltext-Editor oder das benutzte Framework. Jedenfalls gibt es genug Methoden und Ansätze mit Python, die man zurück in die JVM-Welt führen kann. Es ist auch viel effektiver und schneller, eine interaktive, webbasierte Umgebung mit vielen Visualisierungsmöglichkeiten mit dem Namen Jupyter Notebook zu starten, als ein neues Maven-Projekt anzulegen, wenn es darum geht, einmalig und schnell eine CSV-Datei einzulesen und die Zeilen nach einem Spaltenwert zu sortieren. Und nicht zuletzt schadet es nicht, eine der populärsten Programmiersprache zu kennen.

Trotz der bedeutenden Rolle von Python muss man als Datenwissenschaftler Java und Co. nicht an den Nagel hängen – im Gegenteil: Python eignet sich zwar für viele Aufgaben in dem Bereich, da das Ökosystem an Bibliotheken und Blogbeiträgen sehr groß ist, dennoch gibt es Bereiche und Themen, in denen die Schwächen überwiegen. Ein Beispiel sind bestehende Anwendungen, die auf dem JVM-Stack aufgebaut sind und um eine neue, KI-basierte Funktion

```
class DenseNet(private val layers: List<Layer>) {
    fun predict(input: Matrix): Matrix {
        var data = input
        layers.forEach { layer ->
            val weighted = layer.weights.
multiply(data)
            val biased = weighted.add(layer.bias)
            data = sigmoid(biased)
        }
        return data
    }
}
```

Listing 1: Vereinfachte Implementierung der `predict`-Funktion in Kotlin

erweitert werden sollen. Im Bereich der Datenwissenschaften ist Java bereits gut mit ernstzunehmenden Bibliotheken und Frameworks vertreten, ob es sich um reine Mathematik, Visualisierung oder maschinelles Lernen handelt. Dazu gibt es JVM-basierte Laufzeitumgebungen, die bereits vortrainierte Modelle ausführen können. Bei Apps für mobile Geräte möchte man auch auf Python verzichten und die beste User Experience mit nativen Mitteln erreichen. Gerade bei Android ist man mit Java und Kotlin gut bedient.

Dazu kommt, dass die wichtigsten Python-Frameworks für das maschinelles Lernen (Tensorflow *[6]* und PyTorch *[7]*) einen JVM Wrapper anbieten. Für mobile Geräte existiert eine Lite-Variante von Tensorflow.

Quellen

[1] *https://paperswithcode.com/*
[2] HuggingFace: *https://huggingface.co/*
[3] TensorflowHub: *https://www.tensorflow.org/hub/*
[4] *https://www.coursera.org/learn/learning-how-to-learn*
[5] Einfache Implementierung eines neuronalen Netzwerks in Kotlin: *https://github.com/michalharakal/sKaiNet*
[6] *https://www.tensorflow.org*
[7] *https://pytorch.org/*

Michal Harakal
Deutsche Telekom AG
michal.harakal@telekom.de

Michal Harakal arbeitet seit 2010 bei der Deutsche Telekom AG. Angefangen hat er als Android-Entwickler und arbeitet aktuell als Machine Learning Engineer. Sein Werkzeugkasten ist mit vielen Tools und Programmiersprachen gefüllt, in dem Java, Kotlin und Kotlin Multiplatform eine bedeutende Rolle spielen. Er schreibt gerne offene Software. Das Lernen und die Gemeinschaft spielen für ihn eine sehr wichtige Rolle, sei es die Entwicklung eines alternativen Planers für die JavaLand-Konferenz oder das Organisieren eines „Data Science Python Study Group"-Meetups.

Kernprinzipien für die künstliche Intelligenz – eine Übersicht im Dschungel der Prinzipien

Corinna Wiedenmann, Freshfields Bruckhaus Deringer

Anwendungen künstlicher Intelligenz finden immer häufiger Einsatz. Sie beeinflussen potenziell unsere Werte in der Gesellschaft, indem sie die individuellen Rechte und die Gleichheit verletzen und die Grenzen der Verantwortung verwischen können. Daher ist die Beschäftigung mit ethischen Richtlinien für Innovationen, die auf dieser Technologie basieren, von so großer Bedeutung. Es verwundert also nicht, dass bereits mehrere Prinzipienkataloge für eine ethische Nutzung von Institutionen nationaler und internationaler Ebene erarbeitet wurden. Um bei der Prinzipienvielfalt nicht den Überblick zu verlieren, haben Philosophen mithilfe einer Meta-Analyse fünf Prinzipien herausgearbeitet, die sich als zentraler Kern ethischer Überlegungen für künstliche Intelligenz wiederfinden. Diese Prinzipien werde ich im Folgenden erklären, zur Veranschaulichung einen Vergleich mit der Bioethik ziehen und konkrete Beispiele aus Anwendungsfällen anführen, um die Wichtigkeit dieser Prinzipien zu bekräftigen.

In den vergangenen Jahren hat die durch die rasanten technischen Fortschritte getriebenen Entwicklung der künstlichen Intelligenz (KI) das Leben privat, im Arbeitsumfeld und in der Gesellschaft verändert. Manche Neuerung integrierte sich subtil in unseren Alltag, ohne dass die Existenz von KI explizit genannt wurde, wie beispielsweise die Autokorrektur und Vorschlagsfunktion bei Suchmaschinen; manch andere Innovation war offensichtlicher, wie beispielsweise der Einzug von Sprachassistenten in viele Wohnzimmer.

„Im Kern ist KI eine Computerprogrammierung, die lernt und sich anpasst. Sie kann nicht jedes Problem lösen, aber ihr Potenzial, unser Leben zu verbessern, ist tiefgreifend... Wir sind uns bewusst, dass eine so mächtige Technologie ebenso mächtige Fragen über ihre Nutzung aufwirft. Die Art und Weise, wie KI entwickelt und eingesetzt wird, wird die Gesellschaft in den kommenden Jahren maßgeblich beeinflussen." *[1]*

Neben den Vorteilen von KI, die Anwendungen beispielsweise in der Gesundheitsversorgung, Landwirtschaft, Bildung, Energieversorgung, im öffentlichen Sektor oder der Sicherheit erzeugen können, bringen die Anwendungen auch ein Set an Risiken in Gebieten des Verbraucherschutzes, in der Sicherheit, für die Demokratie und für Menschenrechte mit sich.

Motivationen für ethische KI

Weil mit diesen Erfindungen sowohl Vorzüge als auch ethische Risiken einhergehen, haben sie in den vergangenen Jahren immer wieder Diskussionen rund um deren ethische Verankerung aufgeworfen. Die Relevanz der Beschäftigung mit diesem Thema lässt sich in zwei Motivationsgruppen unterteilen. Die Beweggründe werde ich im Folgenden als intrinsische und extrinsische Motivationsgründe darstellen.

Die intrinsischen Motivationsfaktoren umfassen Beweggründe, die aus inneren Anreizen, also aus sich selbst entstehenden Motivationen bei Menschen, resultieren. Einer dieser Motivationsgründe ist, dass Menschen die Moralvorstellungen, denen sie selbst folgen, auch in den von ihnen geschaffenen Systemen und Produkten repräsentiert sehen wollen. Dies kann dem Zugehörigkeitsgrundmotiv zugeordnet werden *[2]*. Denn wir versuchen, die Erstellung eines Systems mit widersprüchlicher Weltsicht zu vermeiden.

Eine weitere Motivation ist, dass Menschen ein Interesse daran haben, Missbrauch von KI zu verhindern und sich gegenseitig bei der Entwicklung solcher Systeme Regeln aufzuerlegen, um den eigenen Schutz sicherzustellen. Dieses Bedürfnis ist eine Ausprägung des Grundmotives der Macht *[2]*.

Mit extrinsischer Motivation bezeichne ich die Faktoren, die als von außen kommende Triebkräfte den Bedarf ethischer Richtlinien für KI begründen. Zu diesen zählen die zunehmende Bedeutung von ethischer Adäquatheit von KI bei Anwendungen für Kunden. Eine Studie von Capgemini zeigt auf, dass Kunden von KI neben den funktionalen Eigenschaften auch Transparenz bei der Erhebung der Daten, dem Einsatz und der Steuerung von Datenverarbeitungssystemen und den Entscheidungen, die basierend auf den zur Verfügung stehenden Daten getroffen werden, erwarten *[3]*.

Einen weiteren extrinsischen Faktor stellen die regulatorischen Anforderungen dar. Unternehmen sind gefordert, bei neuen Produktentwicklungen den regulatorischen Rahmen der Gesetzgeber und Regulatoren einzuhalten. Auf diesen Aspekt werde ich im letzten Absatz noch einmal eingehen, um einen Ausblick auf die Entwicklungen in Europa zu geben.

Zuletzt sind auch Wettbewerbsgründe als externer Faktor zu nennen. In der europäischen Diskussion wird die Qualität und moralische Integrität, die man in den Anwendungen erzielen möchte, als Unique Value Proposition gesehen, also als Alleinstellungsmerkmal im Vergleich mit technischen Anwendungen aus dem nicht-europäischen Ausland.

Vielfalt der Richtlinien

In den vergangenen Jahren wurden von verschiedenen Institutionen Richtlinien und Prinzipienkataloge veröffentlicht, die sich dem Thema Ethik für KI widmeten. Um die Übersicht dabei nicht zu verlieren, untersuchte eine Gruppe von Philosophen in einer Metastudie internationale Prinzipienkataloge daraufhin, ob es in diesen wiederkehrende Muster und Kategorien gibt, die man quasi als kleinsten gemeinsamen Nenner aus allen diesen Werken ziehen kann. Dafür wurden sechs Prinzipienkataloge verglichen. Diese enthielten 47 Prinzipien, die sich in fünf Kernprinzipien zusammenführen ließen *[4]*. Diese sind:

- Gemeinwohl,
- Verhinderung von Bösem,
- Autonomie,
- Gerechtigkeit und
- Erklärbarkeit.

Interessanterweise sind die ersten vier Prinzipien auch in der Bio- und Medizinethik wiederzufinden. Es sind somit die Prinzipien, die Ärzten als ethische Richtlinien für schwierige, manchmal auch Dilemma-Entscheidungssituationen gelten. Zur Veranschaulichung werde ich daher nach der Erklärung jedes Prinzips für KI einen Vergleich mit der Bioethik ziehen.

Prinzip des Gemeinwohls

Das erste Prinzip, das ich beschreiben möchte, ist eins, das in Debatten über die Ethik für KI stets mit aufgeführt wird. KI-Anwendungen sollen zum Guten eingesetzt werden. Sie sollen einen Mehrwert für den Nutzer schaffen, der im Einklang mit dem Wohl der Menschheit steht. Diesen moralischen Anspruch kann man auch so formulieren: KI soll Gutes wollen, also mit einer guten Intuition gebaut sein und handeln. Was jedoch als gut gilt, kann sehr breit gefasst werden.

Sowohl Produktivitätssteigerungen als auch das Erzeugen von Freude, wie beispielsweise die intelligenten Filterfunktionen für Fotos, können darunterfallen. In manchem Prinzipienkatalog zählt zu dieser Kategorie auch Nachhaltigkeit als ein Gedanke der Generationengerechtigkeit und damit als dem Gemeinwohl über die Gegenwart hinaus dienlich. Das kann eine KI gegebenenfalls sogar besser priorisieren als wir Menschen. Denn sie kann langfristig das Wohl der Menschen im Blick behalten und langfristige Prioritäten kurzfristigem Nutzenstiften überordnen.

Im Vergleich mit der Bioethik und der Tätigkeit von Ärzten lässt sich das Prinzip sogar noch etwas anschaulicher beschreiben: Ärzte haben die Pflicht, dem Patienten Gutes zu tun, also seinen Nutzen zu

steigern, indem sie positive Schritte ergreifen, um die Situation des Patienten zu verbessern. Die Heilung oder Genesung ist stets der Handlungsimperativ und gilt als Orientierung für alle Entscheidungen.

Prinzip der Verhinderung des Bösen

Auch wenn das Prinzip der Verhinderung des Bösen als das Nichts-Böses-Wollen dem Gemeinwohl synonym erscheint, stellt es unterschiedliche Prinzipien dar. Bekannt wurde dieses Prinzip insbesondere durch den Science-Fiction Autor Isaac Asimov und die nach ihm benannten Gesetze für Roboter. In seiner Kurzgeschichte Runaround schreibt er: Ein Roboter soll keinen Menschen verletzen oder ihm schaden, auch nicht durch Unterlassung *[5]*.

Dieses Prinzip findet sich ebenfalls in allen Debatten und den untersuchten Prinzipienkatalogen wieder. Es wird in der Regel mit dem Verbot der Verletzung von Menschenrechten verknüpft, schließt jedoch in einigen Regelwerken auch das Verbot der Verletzung von Datenschutzrechten mit ein.

In der Bioethik gilt dieses Prinzip ebenfalls: Ärzte dürfen dem Patienten nicht absichtlich schaden, weder durch Handlung noch durch Unterlassung. Auch Folgeschäden und Konsequenzen für den Patienten, die aus Fahrlässigkeit resultieren, müssen verhindert werden. Somit ist der Arzt angehalten, Patienten auch keinem unangemessenen Risiko auszusetzen. Das Prinzip bekräftigt die Notwendigkeit der medizinischen Kompetenz, solche Entscheidungen und eine Risikoabwägung auch treffen zu können.

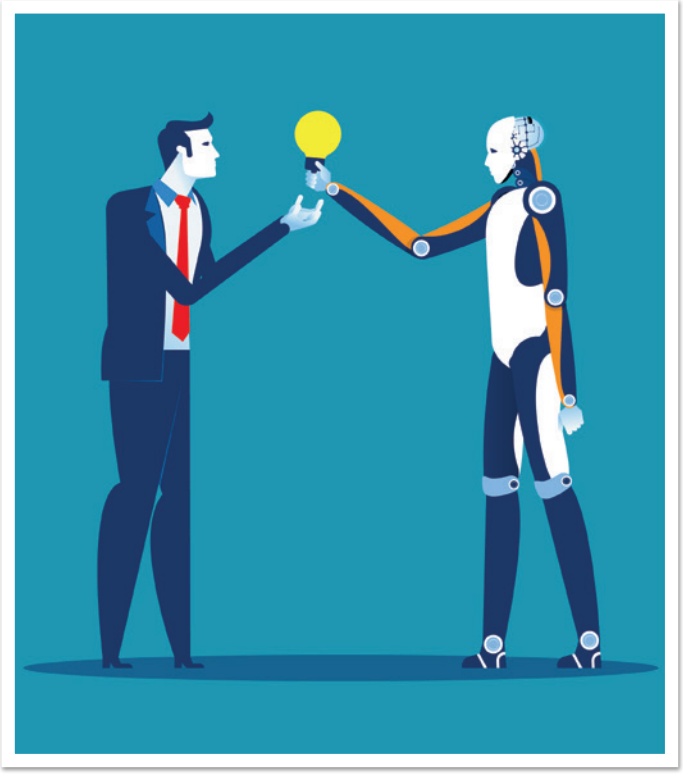
Dies wiederum wird in Zukunft eine spannende Fragestellung für KI-Anwendungen werden, denn was bedarf es, um zu beurteilen, ob eine solche Reife und Kompetenz auch der KI zugrunde liegt? Welche Reife muss sie haben, um Handlungsunterlassungen und Schadensbegrenzungen sicherzustellen? Und wird es irgendwann ein Unterlassungsrisiko sein, KI-Anwendungen nicht zu nutzen, obwohl sie mehr Sicherheit bieten als nicht-KI unterstützte Anwendungen?

Prinzip der Autonomie

Das dritte Kernprinzip ist das der Erhaltung menschlicher Autonomie. Auch wenn jeder Einsatz von KI bewusst das Delegieren von Aufgaben darstellt, wie beispielsweise eine Autokorrektur, die keine händische Überprüfung der Rechtschreibung mehr erfordert, oder eine Filterfunktion, die aufwendige graphische Nachbereitungen ersetzt, herrscht doch Einigkeit darüber, dass das finale Entscheiden über die Endergebnisse beim Menschen verbleiben soll. Diese Entscheidungshoheit, die Widerrufbarkeit von Entscheidungen und insbesondere auch die Entscheidung darüber, welche Entscheidungen delegiert werden, sollen bei jeder KI-Anwendung in menschlicher Hand bleiben.

Dadurch soll die Gefahr einer Automatisierung und möglichen Autonomie der KI verhindert werden. Dieses Prinzip wird auch als „entscheide zu delegieren" „Decide to Delegate" bezeichnet *[6]*. Menschen sollen die Möglichkeit der Wahlfreiheit behalten, die Möglichkeit zur Intervention – wenn nicht selbst, dann über Dritte – bekommen und Entscheidungen hinterfragen, anfechten und auch wieder rückgängig machen können.

In der Bioethik findet sich dieses Prinzip ebenfalls wieder. Im Beispiel des Arzt-Patientenverhältnisses bedeutet es, dass der Patient



die Möglichkeit haben muss, sein Einverständnis zu geben und ohne kontrollierende Einflüsse eine Entscheidung, gegebenenfalls auch gegen die Empfehlung des Arztes, zu treffen. Dieses Prinzip bildet die Grundlage für die sogenannte „informierte Zustimmung" im Arzt-Patientenverhältnis.

Prinzip der Gerechtigkeit

Das vierte Prinzip, das sich in fast allen Prinzipienkatalogen wiederfindet, ist das der Herstellung von Gerechtigkeit. Für die KI bringt das Prinzip die Anforderung von Gleichheit und Konsistenz mit sich. Es geht dabei darum, zu gewährleisten, dass Zugangsbarrieren abgebaut werden und Fairness bei der Nutzung besteht. In der Bioethik bedeutet dieses Prinzip, Aristoteles gemäß: „Jedem das geben, was ihm zusteht."

Das beinhaltet sowohl eine Zugangsgewährleistung, also, wo möglich, gerechte Verteilung von Zugängen zu Gütern in der Gesellschaft, aber auch die Gleichbehandlung von Patienten fällt darunter. Ärzte dürfen keine Unterschiede bei der Behandlung von Patienten aufgrund von Unterschieden machen, die außerhalb der für die Behandlung relevanten Kriterien liegen.

Prinzip der Erklärbarkeit

Das fünfte und letzte Kernprinzip, das sich in allen Prinzipienkatalogen wiederfindet, ist das der Erklärbarkeit. Erklärbarkeit meint sowohl Verständlichkeit, die ein bestimmtes Maß an Transparenz voraussetzt, als auch Verantwortlichkeit, also eine Antwort auf die Frage, wer die Verantwortung für die Funktionen trägt. Dieses Prinzip fordert, dass Menschen die Möglichkeit bekommen, Entscheidungsverhalten von KI und deren Resultate nachvollziehen zu können. Lange Zeit galt KI als komplette Black Box und damit als Gefahrenquelle, die in der gesellschaftlichen Debatte oft als Hauptgrund für die Beschränkung der Weiterentwicklung dieser Technologie aufgeführt wurde.

Seit einigen Jahren mehren sich daher Forschungsbestrebungen rund um „erklärbare KI". Die Idee dahinter ist, dass die KI in der jeweiligen Nutzungsanwendung die Möglichkeit gibt, die Entscheidungsfindung nachzuvollziehen und damit einhergehend auch, wenn erforderlich, einen Verantwortlichen zu benennen.

Dieses Prinzip wurde bisher in der Bioethik nicht mit aufgeführt, wahrscheinlich weil ein Arzt in der Regel für Rückfragen zur Verfügung steht und die Verantwortlichkeit ebenfalls offensichtlich ist.

Regulierung auf europäischer Ebene

Aufgrund der Geschwindigkeit des technologischen Wandels und des damit einhergehenden Einzugs von KI-Anwendungen in unser Leben gibt es auf internationaler Ebene Bestrebungen, der Weiterentwicklung ein Rahmenwerk zu geben.

In der KI-Strategie der Europäischen Kommission weist sie drei Ziele aus, die auf europäischer Ebene in den kommenden Jahren erreicht werden sollen:

- öffentliche und private Investitionen in KI ankurbeln
- die Fähigkeiten zur optimalen Nutzung von KI und Automatisierung am Arbeitsplatz fördern
- Gewährleistung eines angemessenen ethischen und rechtlichen Rahmens

Um den ethischen und rechtlichen Rahmen zu gewährleisten, publizierte die Europäische Kommission im April 2021 einen Vorschlag zur Festlegung harmonisierter Vorschriften über künstliche Intelligenz (Gesetz über künstliche Intelligenz) und zur Änderung bestimmter Rechtsakte *[8]*.

Der Gesetzvorschlag der Europäischen Kommission wird, sollte er so umgesetzt werden, einen großen Einfluss auf Unternehmen haben. Denn der Vorschlag eröffnet das breite Anwendungsspektrum, in dem KI bereits zum Einsatz kommt, und stellt neue Anforderungen für KI-Anwendungen mit hohem Risiko. Diese gelten insbesondere für Fälle, in denen die Sicherheit von Menschen direkt betroffen ist, aber auch dort, wo Datenanalysen und Profiling durchgeführt werden können *[9]*. Darüber hinaus wird von der Europäischen Kommission die Bedeutung von Kontrollen betont, um Transparenz zu verbessern und unbewusste Beeinflussung zu verhindern.

Ausblick

Es bleibt abzuwarten, wie schnell die Bestrebungen der Regulierung unter Berücksichtigung ethischer Prinzipien zu gesetzlichen Anforderungen werden. Insgesamt lässt sich diese Entwicklung jedoch sehr begrüßen, um den angesprochenen Risiken zu begegnen und mit der fortschreitenden Entwicklung dieser einflussreichen Technologie aus gesellschaftlicher und sozialer Sicht Schritt zu halten.

Der Philosoph Hans Jonas bringt in seinem Buch „Prinzip der Verantwortung" die Relevanz, dass Ethik bei wissenschaftlicher Weiterentwicklung berücksichtigt wird, folgendermaßen auf den Punkt: „Der endgültig entfesselte Prometheus, dem die Wissenschaft nie gekannte Kräfte und die Wirtschaft rastlosen Antrieb gibt, ruft nach einer Ethik, die durch freiwillige Zügel seine Macht davor zurückhält, dem Menschen zum Unheil zu werden." *[10]*

Quellen

[1] Übersetzung gemäß Sundar Pichai (2018): AI at Google: our principles. Google blog The Keyword, *https://www.blog.google/technology/ai/ai-principles/*

[2] David McClelland (1987): Human Motivation. Cambridge University Press, Cambridge

[3] Capgemini (2020): AI and the ethical Conundrum. How organizations can build ethically robust AI systems and gain trust. Capgemini Research Institute, *https://www.capgemini.com/wp-content/uploads/2020/10/AI-and-the-Ethical-Conundrum-Report-1.pdf*

[4] Luciano Floridi und Josh Cowls (2019): A Unified Framework of Five Principles for AI in Society. MIT Press, *https://hdsr.mitpress.mit.edu/pub/l0jsh9d1/release/7*

[5] Isaac Asimov (1950): Runaround. I, Robot, Doubleday, New York City

[6] Luciano Floridi, Josh Cowls et al. (2018): AI4People - An Ethical Framework for a Good AI Society: Opportunities, Risks, Principles, and Recommendations, Minds & Machines, Springer Link, *https://link.springer.com/article/10.1007/s11023-018-9482-5*

[7] Europäische Kommission (2021): Excellence and Trust in AI - Brochure. EC Library, *https://digital-strategy.ec.europa.eu/en/library/excellence-and-trust-ai-brochure*

[8] Europäische Kommission (2021): Proposal for a Regulation laying down harmonised rules on artificial intelligence, Brüssel COM(2021) 206 final, *https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence*

[9] Ben Cohen, Natalie Pettinger Kearney, Giles Pratt, Andrew Austin, Sascha Schubert und Christoph Werkmeister (2021): The EU seeks to balance AI risks and benefits. Freshfields Our Thinking, *https://www.freshfields.com/en-gb/our-thinking/campaigns/digital/artificial-intelligence/the-eus-proposed-ai-regulation/*

[10] Hans Jonas (1979): Das Prinzip Verantwortung: Versuch einer Ethik für die technologische Zivilisation. Suhrkamp, Frankfurt

Corinna Wiedenmann
Freshfields Bruckhaus Deringer
corinna.wiedenmann@gmail.com

Corinna Wiedenmann studierte Philosophie und Wirtschaft, arbeitete mehrere Jahre im Innovationsmanagement und beschäftigt sich daher stark mit Themen rund um Ethik und Digitalisierung. Heute arbeitet sie als Business Development Managerin für die Kanzlei Freshfields Bruckhaus Deringer und ist dort für den europäischen Financial-Institution-Sektor verantwortlich.

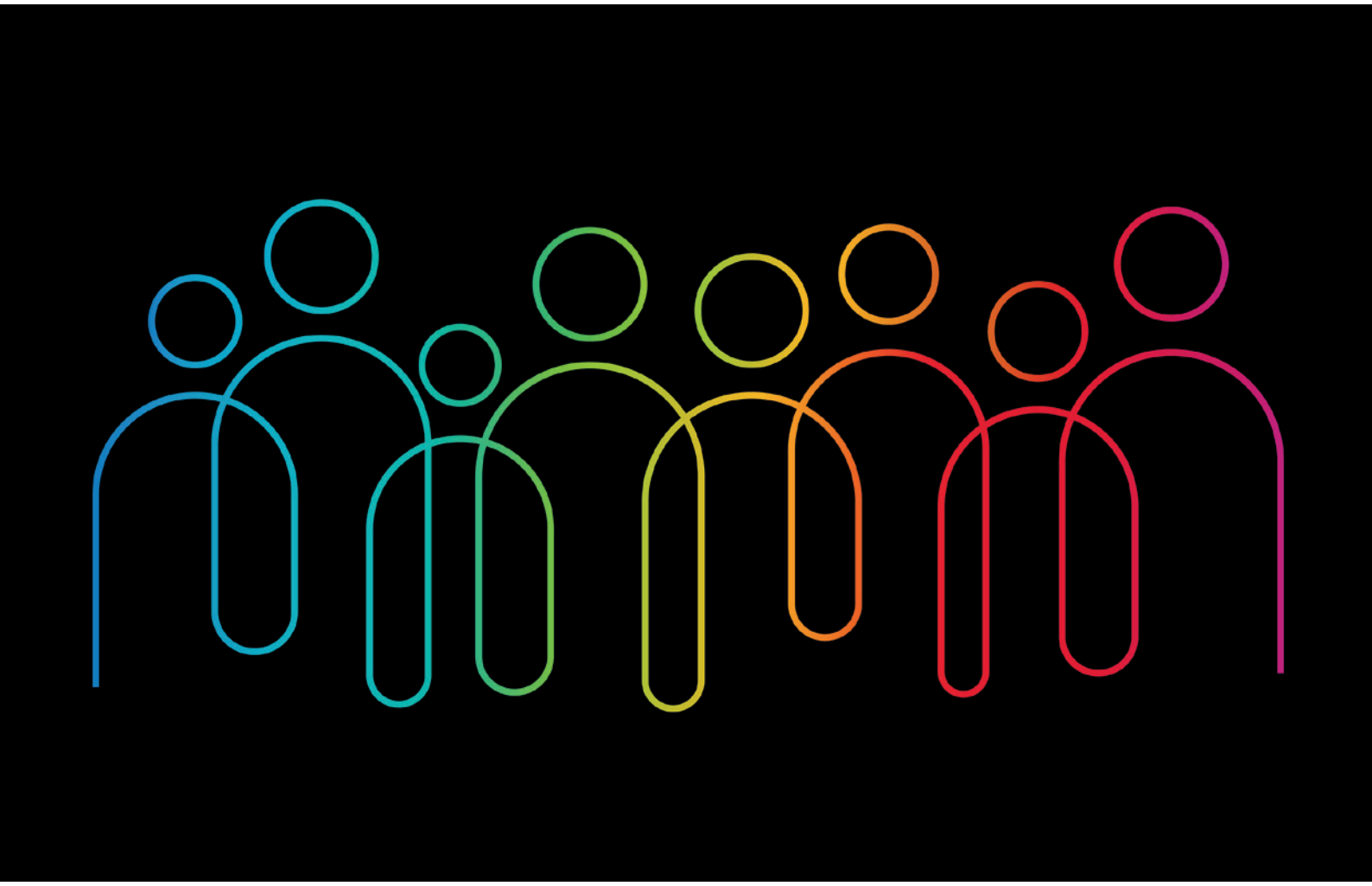
Mitmachen und Autor werden!

Sie kennen sich in einem bestimmten Gebiet aus dem Java-Themenbereich bestens aus und möchten als Autor Ihr Wissen mit der Community teilen?

Nehmen Sie Kontakt zu uns auf und senden Sie Ihren Artikelvorschlag zur Abstimmung an **redaktion@ijug.eu.**

Wir freuen uns, von Ihnen zu hören!

iJUG Verbund



Barrierefreiheit – ein kleiner Crashkurs

Anna Maier, TOPdesk

Barrierefreiheit oder Accessibility sind Schlagwörter, die immer häufiger auftauchen. Doch was versteht man eigentlich darunter und wie macht man eine Website barrierefrei? Dieser Artikel soll einen Einstieg in das komplexe Thema bieten.

In den letzten Jahren rückte die Barrierefreiheit von Websites immer stärker in den Fokus. Das liegt im Wesentlichen an der geänderten Gesetzeslage: Ab 23. Juni 2021 müssen Websites des öffentlichen Sektors barrierefrei sein; Websites der Privatwirtschaft folgen später *[1]*. Auch aktuelle Debatten zu Diversität und Inklusion haben ihren Teil dazu beigetragen.

Zum ersten Mal damit konfrontiert, schrecken viele Entwickler zurück. Zum einen fehlen Erfahrungswerte, da es unter Umständen wenig Berührungspunkte zu Nutzern mit Beeinträchtigung gibt. Zum anderen gibt es einen ganzen Stapel Richtlinien zu befolgen, was auf den ersten Blick sehr abschreckend wirkt.

Viele denken auch, dass Barrierefreiheit nur blinde Nutzer betrifft und sind wenig motiviert, für diese kleine Gruppe Anpassungen vorzunehmen. Dabei profitieren viel mehr Nutzer von einer barrierefreien Website, als man denkt. In der EU gibt es circa 80 Millionen Menschen mit einer Beeinträchtigung, die auf vielfältige Weise Probleme mit der Benutzung einer Website haben können. Darüber hinaus können aber auch „normale" Menschen von Barrierefreiheit profitieren.

Ein breites Nutzerspektrum

Für Nutzer mit einer Sehbeeinträchtigung ist es beispielsweise wichtig, dass die Elemente einer Website einen guten Kontrast haben und auch bei hohem Zoomfaktor benutzbar sind – Features, die auch bei Smartphone-Nutzern beliebt sind. Eine niederländische Studie hat gezeigt, dass 33 Prozent aller niederländischen iPhone-Nutzer die Textgröße ändern und 27 Prozent den Dark Mode aktivieren *[2]*.

Nutzer mit einer motorischen Beeinträchtigung der Arme sind oft nicht in der Lage, eine Maus zu bedienen und benutzen den PC mit Tastatur oder Sprachsteuerung. Oft treffen sie dabei auf Hindernisse, etwa, dass auf einer Website nicht alles mit dem Keyboard steuerbar oder die Tab-Reihenfolge komplett unvorhersehbar ist. Eine gute Bedienbarkeit mit der Tastatur hilft nicht nur diesen Nutzern, sondern auch Menschen, die vorübergehend in der Mausbedienung eingeschränkt sind, zum Beispiel Menschen mit Karpaltunnelsyndrom *[3]*.

Viele Anforderungen von Nutzern mit kognitiven Einschränkungen würde man auf den ersten Blick gar nicht unbedingt mit Barrierefreiheit in Verbindung bringen, sondern eher unter Usability verbuchen – etwa klare Strukturen und Navigation, um sich auf der Website zurechtzufinden. In Formularen gehören informative Labels, Fehlermeldungen und die Unterbindung von „Auto Submit" dazu. Die Möglichkeit, sich eine Seite vorlesen zu lassen, ist ein Feature für Menschen mit einer Lese- oder Sehschwäche – wird aber auch gern von Normalos genutzt.

Aus den Beispielen wird deutlich, dass Barrierefreiheit einer großen Gruppe von Menschen zugutekommt. Schon allein deswegen ist das Thema es wert, sich damit zu beschäftigen.

Richtlinien

Aus dem vorigen Abschnitt wird auch deutlich, dass es nicht einfach ist, die unterschiedlichen Bedürfnisse aller Nutzer zu berücksichtigen. Sehr hilfreich dabei sind die Web Content Accessibility Guidelines (WCAG) *[4]* vom W3C, die sich genau das zum Ziel gesetzt haben.

Die Richtlinien teilen sich in vier Prinzipien ein: wahrnehmbar, bedienbar, verständlich und robust. Nur wenn eine Website für alle Benutzer diese Prinzipien erfüllt, ist sie barrierefrei.

Zusätzlich gibt es drei Konformitätslevel: A, AA und AAA. Level AA ist die Grundlage für die europäischen Gesetze. Wenn eine Website sich an diese Richtlinien hält, ist sie für die meisten Nutzer zugänglich und gleichzeitig gesetzeskonform.

Die Guidelines sind mit 78 Punkten sehr umfangreich. Im weiteren Verlauf des Artikels werde ich die eine oder andere Richtlinie erwähnen, doch für eine komplette Übersicht fehlt hier der Platz.

Barrierefreiheit umsetzen

Selbst mit den WCAG als Vorgabe ist die Umsetzung einer barrierefreien Website nicht trivial. Es gibt zwar jede Menge Tools, die einen dabei unterstützen, aber es führt kein Weg daran vorbei, sich mit dem Thema eingehender zu beschäftigen.

Idealerweise wird Barrierefreiheit bereits in der Design-Phase der Website berücksichtigt. Das ist erfahrungsgemäß weit weniger Arbeit, als eine bestehende Website zu migrieren – und auch sehr viel erfolgversprechender.

Ein erster Schritt ist es, Nutzer als Menschen mit unterschiedlichen Fähigkeiten, Vorlieben und Wissensständen wahrzunehmen und darauf abzuzielen, die Website möglichst vielen dieser Menschen zugänglich zu machen. Folgerichtig sollte man auch beim User Research auf Diversität setzen. Moderne Designansätze wie Inclusive Design oder Universal Design zeigen, wie das funktionieren kann.

Auch bei konkreteren Aspekten wie dem Farbschema kann man Barrierefreiheit von Anfang an berücksichtigen. Laut WCAG 1.4.3 muss ein Text einen Kontrast zum Hintergrund von mindesten 4,5:1 aufweisen. Größerer Text kommt mit 3:1 davon. Wenn man seine Farbpalette entsprechend aufbaut, ist man hier schon auf der sicheren Seite.

Beim Design des Seitenaufbaus sollte man sich schon Gedanken über die Navigation der Seite mit dem Keyboard machen. Wie kann sich der Benutzer über die Seite bewegen? Wenn sich beispielsweise oben eine Navigationsleiste befindet, muss dem Nutzer die Möglichkeit gegeben werden, diese zu überspringen (WCAG 2.4.1).

Strukturierter Seitenaufbau

Die meisten Elemente in HTML haben eine eingebaute Semantik. Zum Beispiel bedeutet `<h1>` immer, dass es die Hauptüberschrift der Seite ist, unabhängig davon, wie diese Überschrift aussieht und wo sie positioniert ist. Allerdings gibt es auch nicht-semantische Elemente so wie `<div>`. Ein `<div>` kann optisch zu allem Möglichen werden, auch zur Hauptüberschrift. Leider geht dann die Bedeutung des Elements „Hauptüberschrift" verloren.

Das kann auf verschiedene Weise zu Problemen in der Darstellung führen. Blinde Nutzer lesen den Bildschirminhalt mit einem sogenannten Screen Reader aus. Sie verlassen sich sehr häufig auf Überschriften, um sich auf einer Website zu orientieren *[5]*. In *Listing 1* sieht man ein Beispiel, in dem Überschriften mit semantischen und nicht-semantischen Elementen realisiert wurden.

Abbildung 1 zeigt die Darstellung der Seite im Browser und die Struktur, die sich daraus für Screen-Reader-Nutzer ergibt. Visuell sieht man keinen Unterschied zwischen den semantischen Überschriften und den nicht-semantischen. Der Screen-Reader-Nutzer bekommt

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<div style="font-weight: bold; font-size: 24px">Wannabe Heading 2</div>
<div style="font-weight: bold; font-size: 18px">Wannabe Heading 3</div>
```

Listing 1: Semantische und nicht-semantische Überschriften

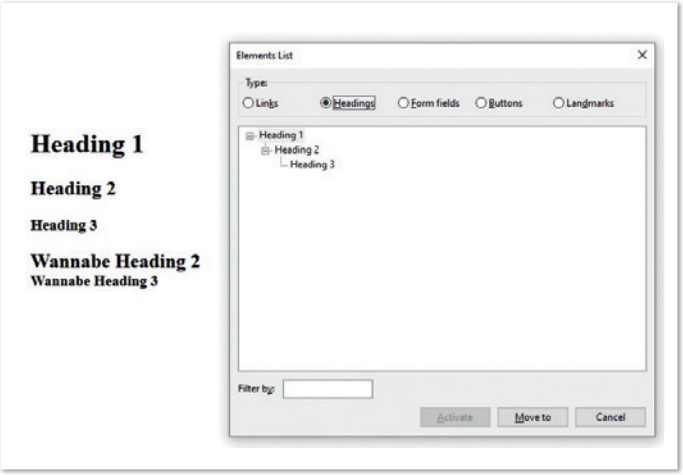


Abbildung 1: Darstellung des Überschriftenbeispiels links im Browser, rechts im Screen-Reader-Menü (© Anna Maier)

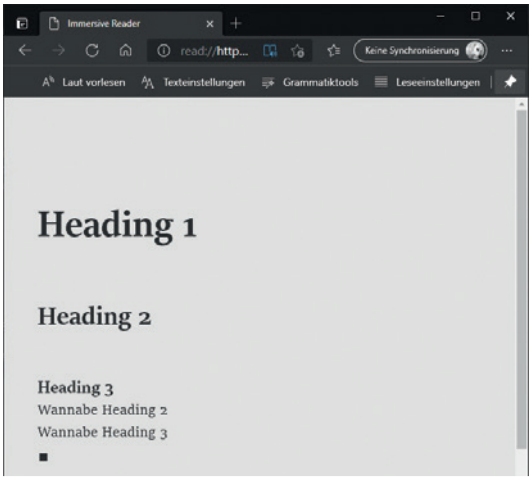


Abbildung 2: Darstellung des Überschriftenbeispiels im Edge Immersive Reader (© Anna Maier)

nur die Hälfte der Informationen: Die semantischen Tags werden erkannt und als übersichtlicher, navigierbarer Baum dargestellt. Die `<div>`s werden schlicht ignoriert.

Mehrere Browser bieten jetzt auch einen speziellen Lesemodus an. In diesem Modus wird die Website auf ihre Struktur reduziert und der Nutzer kann die Darstellung nach seinen Bedürfnissen anpassen. *Abbildung 2* zeigt, wie die Seite im Edge Immersive Reader aussieht. Auch hier werden die `<div>`s nicht als Überschriften erkannt.

Interaktive Elemente

Während fehlende Überschriften eine Seite nicht komplett unbenutzbar machen, kann fehlende Semantik in interaktiven Elementen zum Knock-out-Kriterium werden. Leider werden auf vielen Seiten anstatt der nativen HTML-Elemente `<button>`, `<select>`, `<radio>` und so weiter selbstgebaute Elemente verwendet.

Listing 2 zeigt einen selbstgebauten Button: ein `<div>`, das per CSS als Button dekoriert wurde. Die fehlende Semantik führt dazu, dass der Button vom Browser und von assistierender Software, wie Screen Reader oder Speech Recognition Software, nicht als solcher erkannt wird. Darüber hinaus kann er auch nicht mit der Tastatur angesteuert, geschweige denn gedrückt werden. Man schließt also eine ganze Reihe von Nutzern aus und verstößt auch gegen die WCAG-Richtlinie 4.1.2, nach der jedes Element entsprechende Semantik haben muss.

Bis zu einem gewissen Grad kann man die Semantik „reparieren". Die HTML-Erweiterung ARIA (Accessible Rich Internet Applications, *[6]*) bietet Attribute, mit denen man semantische Informationen an nicht-semantische Elemente anhängen kann. In unserem Beispiel kann man das `role`-Attribut verwenden, wie *Listing 3* zeigt.

Nun wird das Element vom Browser und assistierender Software auch als Button erkannt. Leider ist das aber auch schon alles. Die Tastatur-Interaktion mit dem Element muss noch extra programmiert werden. Bei komplexeren Elementen, wie einer Ausklappliste oder einem Datepicker, kann das durchaus größeren Aufwand bedeuten. Die WAI-ARIA Authoring Practices geben dazu wertvolle Hilfestellungen *[7]*.

```
<div class="button">Click me</div>
```

Listing 2: Ein selbstgebauter Button ohne Semantik

```
<div class="button" role="button">Click me</div>
```

Listing 3: Selbstgebauter Button mit ARIA-Attribut

```
<span>Name</span>
<input type="text" />
```

Listing 4: Text mit Input daneben; ohne programmatische Verknüpfung

Hinzu kommt, dass die Unterstützung für ARIA nicht durchgängig implementiert ist – weder von den Browsern noch von assistierender Software *[8]*. Man kann sich also nicht darauf verlassen, dass das Element richtig erkannt wird.

Die Beispiele zeigen, dass man bei der Implementierung in erster Linie auf die eingebauten HTML-Elemente setzen sollte, da sie die entsprechende Semantik gleich mit sich bringen. Wenn es nicht anders möglich ist, kann man auf ARIA zurückgreifen.

Kontext geben

Während die Semantik uns Informationen über die Struktur gibt, gibt uns der Kontext Informationen über den Inhalt.

Ein prominenter Anwendungsfall ist alternativer Text für Bilder. Jedes Bild auf der Seite sollte eine Beschreibung haben. So ist es auch in der allerersten WCAG-Richtlinie 1.1.1 festgelegt. Nur für dekorative Elemente kann man die Beschreibung leer lassen. Leider haben bis dato geschätzt ein Drittel der Bilder auf Websites noch keinen alternativen Text *[9]*.

Neben Bildern sind auch Labels sehr wichtig. Zum Beispiel in einem Formular, um zu verstehen, was man da gerade ausfüllt. Das ist in Richtlinie WCAG 3.3.2 festgelegt. *Listing 4* zeigt ein Negativbeispiel.

In diesem Beispiel wird das Label zwar neben dem Input angezeigt, aber es ist nicht programmatisch verknüpft. Screen-Reader-Nutzer müssen raten, wofür das Feld da ist. *Listing 5* zeigt ein anderes Negativbeispiel.

```
<input type="text" placeholder="Name" />
```

Listing 5: Input mit Placeholder-Text

Hier wird das Label im Textfeld selbst angezeigt. Allerdings verschwindet die Information, sobald man in das Feld schreibt. Das ist problematisch für Nutzer mit kognitiven Einschränkungen oder für Nutzer, die die automatische Formularausfüllung aktiviert haben: Man kann nicht erkennen, ob die Felder mit den richtigen Informationen ausgefüllt sind. Außerdem ist der Kontrast des Placeholder meistens zu gering, sodass der Text für manche Nutzer nur schwer lesbar ist. *Listing 6* zeigt, wie man das Formularfeld für alle Nutzer zugänglich machen kann.

```
<label for="nameField">Name</label>
<input type="text" id="nameField" />
```

Listing 6: Input mit Label, programmatisch verknüpft

Das Label ist sowohl visuell als auch programmatisch verknüpft, verschwindet nicht und kann mit ausreichend Kontrast dargestellt werden. Somit sollte der Kontext für alle Nutzer gegeben sein. Ein Bonus-Feature ist auch, dass man jetzt auf das Label klicken kann, um das Eingabefeld zu fokussieren, und es somit einfacher zu aktivieren ist.

Testing

Oberflächlich betrachtet ist das Testen auf Barrierefreiheit recht einfach. Es gibt genug Tools, die eine Website automatisch auf WCAG-Verstöße überprüfen können. Zum Beispiel „Lighthouse" in den Chrome Developer Tools oder das Browser Plug-in „Wave". Leider können die Tools selbst bei optimistischen Schätzungen nur ungefähr 30 Prozent der Probleme entdecken.

Beispielsweise können die Tools erkennen, ob ein Bild eine alternative Beschreibung hat – jedoch nicht, ob diese Beschreibung sinnvoll ist. Ein Button wie in *Listing 2*, der aus einem `<div>` besteht, wird gar nicht als Button erkannt, und somit auch dessen Probleme nicht. Das Testen ist also Handarbeit.

Besonders wenn man anfängt, auf Barrierefreiheit zu testen, fühlt man sich häufig von der Menge der Richtlinien überfordert. Im Netz kursieren diverse Checklisten, die an die Richtlinien angelehnt sind und als Grundlage genutzt werden können. Das Tool Accessibility Insights bietet eine interaktive Checkliste, die besonders für Einsteiger sehr hilfreich ist *[10]*.

Nutzerverständnis aufbauen

Für das Testen, aber auch die Entwicklung, ist es sehr hilfreich, ein gewisses Nutzerverständnis aufzubauen, sich also in beeinträchtigte Nutzer hineinzuversetzen. Der beste Weg ist natürlich, Nutzer mit Beeinträchtigungen direkt zu befragen. Meistens hat man diese aber nicht so ohne Weiteres an der Hand, deshalb muss man zu Alternativen greifen.

Verschiedene Browsertools, wie der Web Disability Simulator, bieten die Möglichkeit, Beeinträchtigungen per Overlay zu erfahren. Andere Tools erlauben es, den Kontrast zu verstellen, oder bieten erweiterte Zoomoptionen.

Darüber hinaus kann man das Setup von Nutzern mit Beeinträchtigungen simulieren. Also zum Beispiel die Maus ausstöpseln *[11]*, den High-Contrast-Mode einstellen oder einen Screen Reader installieren. Dadurch gewinnt man oft einen neuen Blickwinkel für Probleme mit Barrierefreiheit.

Fazit

Barrierefreiheit ist ein komplexes Thema, das immer wichtiger wird. Um es richtig anzugehen, ist es unumgänglich, sich mit Richtlinien und Nutzerbedürfnissen auseinanderzusetzen. Doch es lohnt sich! Egal, ob der Nutzer eine Einschränkung hat, gestresst ist, in der Sonne sitzt oder nur ein kleines Display hat: Eine barrierefreie Website funktioniert unter den unterschiedlichsten Bedingungen und ist einfach zu bedienen. Deswegen profitieren am Ende alle Nutzer von einer zugänglichen Website.

Quellen

[1] *https://www.bundesfachstelle-barrierefreiheit.de/DE/Themen/EU-Webseitenrichtlinie/BGG-und-BITV-2-0/Die-neue-BITV-2-0/die-neue-bitv-2-0_node.html*
[2] *https://accessibility.q42.nl/*
[3] *https://www.joshwcomeau.com/blog/hands-free-coding/*
[4] *https://www.w3.org/WAI/standards-guidelines/wcag/*
[5] *https://webaim.org/projects/screenreadersurvey8/#finding*
[6] *https://www.w3.org/TR/wai-aria/*
[7] *https://www.w3.org/TR/wai-aria-practices-1.1/*
[8] *https://caniuse.com/wai-ariassss*
[9] *https://webaim.org/projects/million/#alttext*
[10] *https://techblog.topdesk.com/accessibility/testing-accessibility-with-accessibility-insights/*
[11] *https://www.smashingmagazine.com/2018/07/web-with-just-a-keyboard*

Anna Maier
TOPdesk Deutschland GmbH
a.maier@topdesk.com

Anna Maier ist Softwareentwicklerin bei TOPdesk und Mitgründerin der JUG Kaiserslautern. Sie ist im Backend und Frontend unterwegs und setzt sich bei TOPdesk für Barrierefreiheit ein.

Homeoffice, gekommen um (zuhause) zu bleiben

Birgit Kratz

Homeoffice – Sehnsuchtsort oder Produktivitätskiller? Viele haben sich die Möglichkeit gewünscht, auch mal im Homeoffice zu arbeiten. Einige wollten es nie. Aber seit über einem Jahr macht es in der IT-Branche fast jeder. So auch ich. Welche Erfahrungen ich bei meiner täglichen Entwicklerinnenarbeit gesammelt habe, worauf es mir dabei technisch, aber auch zwischenmenschlich und emotional ankommt, will ich in diesem Beitrag schildern.

Als Freiberuflerin habe ich mir schon häufiger Kunden gewünscht, die Remote-Arbeit als Option sehen. Meist war allerdings nicht mehr als ein Homeoffice-Tag pro Woche möglich.

Aus rein technischer Sicht war die Anwesenheit beim Kunden auch schon vor Corona-Zeiten nicht notwendig. Der Kunde wollte aber das Gefühl von mehr Kontrolle über die Mitarbeiter/innen haben. Es galt so was wie die Formel: Anwesenheit im Büro = mehr Produktivität im Projekt. Mitnichten. Seit Corona sollten auch die Kunden/Arbeitgeber diesbezüglich sehr viel dazugelernt haben.

Da mein derzeitiges Projekt während des ersten Lockdowns gestartet wurde, arbeite ich seitdem zu 100 Prozent im Homeoffice. Es gibt Kollegen und Kolleginnen, die ich noch nie persönlich getroffen habe und mit denen ich trotzdem täglich produktiv zusammenarbeite. Während dieser Zeit haben wir in-time und in-budget unser Projekt vorangetrieben und zum Erfolg geführt.

Hardware

Als Freiberuflerin hatte ich schon länger ein eigenes Büro in meinem Zuhause. Bei der Einrichtung und Ausstattung ist jedoch einiges zu beachten.

Idealerweise hat man einen eigenen, ruhigen Raum zur Verfügung, wo man auch mal die Tür hinter sich schließen kann. Häufig muss allerdings auch das Schlafzimmer oder eine Ecke im Wohnzimmer herhalten. Da kann es schon mal eng werden, wenn noch mehr Haushaltsmitglieder im Homeoffice oder Homeschooling sind.

Tageslicht, ein ausreichend großer, eventuell sogar höhenverstellbarer Schreibtisch, eine ergonomische Sitzmöglichkeit, gute Lichtquellen im Raum und am Schreibtisch (diese sind vor allem in der dunklen Jahreszeit wichtig), mindestens ein guter Bildschirm (und ich meine nicht den Laptopbildschirm), eine (externe) Tastatur, Maus, Webcam, Headset und Mikrofon gehören dann zur idealen Grundausstattung. Diese Liste könnte ich noch beliebig fortsetzen.

Mir ist klar, dass nicht jede/r diese Möglichkeiten hat. Man sollte dennoch versuchen, dem Ideal so nahe wie möglich zu kommen, um effizient arbeiten zu können, ohne sich den Rücken kaputtzumachen oder nachts vor Verspannungen keinen Schlaf zu finden. Manche Arbeitgeber investieren nicht umsonst in ergonomisch eingerichtete Arbeitsplätze. Wie gut das ist, merkt man leider erst, wenn man schon dem etwas älteren Semester angehört. In jungen Jahren steckt man Fehlhaltungen noch leicht weg – das wird sich später jedoch unweigerlich rächen.

Software

Nachdem man also seinen neuen Heimarbeitsplatz so gut wie möglich eingerichtet hat, soll es jetzt um die nötige Software-Infrastruktur gehen. Schließlich will man ja mit seinen Kolleg/innen kommunizieren.

Ein erster Punkt auf der Liste ist sicherlich eine schnelle Internetverbindung. Leider kann einem keiner so genau sagen, welche Schnelligkeit ideal ist. Da gilt wohl der Spruch: „Schnelles Internet ist durch nichts zu ersetzen, nur durch noch schnelleres Internet."

Dankenswerterweise gibt es inzwischen zahllose Tools, um per Chat, Voicecall, Videocall und Screen-Sharing mit den Kolleg/innen in Verbindung zu bleiben und gemeinsam zu arbeiten. Die gängigsten sind sicherlich Teams, Slack, Zoom, WebEx, Google Meet, Jitsi und viele mehr. Die Hersteller dieser Tools waren in letzter Zeit natürlich auch nicht untätig und haben kräftig nachgerüstet. Aspekte wie Sicherheit, vielfältige Auswahlmöglichkeiten, was beim Screen-Sharing geteilt wird, Breakout-Rooms, um sich für eine gewisse Zeit in kleine Gruppen aufzuteilen, um danach wieder zusammenzukommen, sind nur einige Beispiele aus einer langen Liste.

Die eierlegende Wollmilchsau unter den Tools gibt es allerdings nicht. Jedes hat seine Stärken und Schwächen und ist für ein Einsatzszenario besser oder schlechter geeignet.

Allerdings haben die genannten Tools eines gemeinsam: Man sitzt allein zu Hause vor dem Rechner und muss Chatnachrichten an seine Kolleg/innen senden oder sie per Videocall anrufen und eventuell andere dazu holen.

Pragli

Pragli *[1]* geht da einen etwas anderen Weg. Hier wird versucht, die Anwesenheit in einem Büro reeller nachzuempfinden. Bei Pragli gibt es Räume, die man betreten kann. Per Default sind Kamera und Ton aktiv. Ganz so, als würde man wirklich ein Büro betreten. Man ist sichtbar und hörbar. Es ist leicht erkennbar, wer sich in welchem Raum aufhält (per Avatar). Wenn beispielsweise jemand in der Kaffeeküche ist, den ich schon lange einmal sprechen wollte, dann bewege ich mich dorthin und kann sofort anfangen, zu kommunizieren. Natürlich gibt es dort unter anderem auch Screen-Sharing, Chats und Privatnachrichten.

Jeder kann sehen, wer gerade in einem Meetingraum ist. Dieser kann allerdings von innen „abgeschlossen" werden und man muss „anklopfen", um eingelassen zu werden.

Pragli hat noch viele andere interessante Features und ist sicherlich eine Betrachtung wert.

Teams, die hauptsächlich per Pair-Programming oder Mob-Programming zusammenarbeiten, benötigen natürlich auch dafür geeignete Tools. Am einfachsten (aber nicht unbedingt am effizientesten) ist die Benutzung von Screen-Sharing. Jedoch hat man hierbei, neben einer gehörigen Latenz, auch den Nachteil, dass nur derjenige, der gerade seinen Bildschirm teilt, der Driver ist (also an der Tastatur sitzt). Um jetzt den Driver zu wechseln, müssen alle Codeänderungen gepusht werden, der neue Driver muss alles pullen, wiederum den Bildschirm teilen und erst dann kann es weiter gehen. Dabei können schon mal einige Minuten vergehen, ehe man sich wieder der eigentlichen Aufgabe widmen kann. Folglich wird der Driver nicht sehr oft gewechselt, was aber wiederum nicht der Idee des Pair-Programming entspricht.

Ich habe in letzter Zeit zwei Tools probiert, die genau diese Nachteile verhindern wollen.

Mob

„mob", entwickelt von Dr. Simon Harrer *[2] (siehe Abbildung 1)*, will die Übergabezeit zwischen Drivern vereinfachen und verkürzen. Als Versionsverwaltung wird Git vorausgesetzt.

Einmal installiert, benötigt man im Wesentlichen nur drei Befehle:

Abbildung 1: mob (© https://mob.sh)

- mob start
- mob next
- mob done

„mob start" startet eine Session *(siehe Abbildung 2)*. Die Ausgabe im Terminal zeigt genau an, was gemacht wird. Unter anderem wird ein Work-in-progress(WIP)-Branch „mob-session" angelegt. Man kann dem Befehl auch noch eine Zeitspanne mitgeben, nach der man informiert wird, dass es an der Zeit ist, die Tastatur abzugeben.

```
→  ~/Workspace/Private/aoc-2020 git:(master) mob start 10
   git fetch origin --prune
   git merge FETCH_HEAD --ff-only
> starting new mob session from origin/master
   git checkout -B mob-session origin/master
   git push --no-verify --set-upstream origin mob-session
> 10 minutes timer started (finishes at approx. 23:19)
> you are mob programming
> on wip branch mob-session (base branch master)
→  ~/Workspace/Private/aoc-2020 git:(mob-session)
```

Abbildung 2: Starten einer mob-Session

Für den Driver-Wechsel gibt der bisherige Driver „mob next" ein *(siehe Abbildung 3)*. Damit werden alle Änderungen in den „mob-session"-Branch gepusht und der nächste Driver kann per „mob start" eine neue Session starten.

```
→  ~/Workspace/Private/aoc-2020 git:(mob-session) ✗ mob next
   git add --all
   git commit --message mob next [ci-skip] --no-verify
   git push --no-verify origin mob-session
README.md | 2 ++
 1 file changed, 2 insertions(+)
→  ~/Workspace/Private/aoc-2020 git:(mob-session)
```

Abbildung 3: Zeit, den Driver zu wechseln

Ist die Aufgabe fertig, werden die Änderungen mit „mob done" in die Staging-Area des main-Branche gebracht und können dann committet werden *(siehe Abbildung 4)*.

Man braucht natürlich noch ein zusätzliches Tool für Screen-Sharing und Videocall. Die Übergabe zwischen Drivern ist aber sehr schnell erledigt.

Code With Me

JetBrains IntelliJ IDEA bietet mit dem „Code With Me"-Plug-in *[3]* eine in die IDE integrierte Lösung zum gemeinsamen Arbeiten am Code inklusive Voice- und Videocall. Allerdings geht hierbei, wenn man keine On-Premises-Lösung installiert hat, der Code über JetBrains-Server. Nach Angaben von JetBrains Ende-zu-Ende-verschlüsselt.

Wer sowieso mit der IDEA von JetBrains entwickelt, für den/die ist das Plug-in eine sehr interessante Lösung, bei der auch mehrere Entwickler/innen gleichzeitig an verschiedenen Stellen im Code arbeiten können. Zudem braucht auch nur der Einladende (Host) wirklich eine IntelliJ-IDEA-Installation. Als Gast erhält man über einen Link einen Client. Dieser Client ist dem Look and Feel der IDE sehr ähnlich und die Funktionalitäten werden ständig erweitert, um ein gutes Programmiererlebnis zu bieten.

Vielfältige Möglichkeiten der Zugriffssteuerung für die Gäste ermöglichen viele Varianten der Zusammenarbeit. PairProgramming, ClassRoom, Guiding, Mentoring – um nur einige zu nennen.

Wer mehr darüber wissen will, wie „Code With Me" entstanden ist und wie es funktioniert, dem empfehle ich das Video „Code With Me – Behind the Scenes" *[4]*.

Zeichnen, Skizzieren

Manchmal sagen Bilder oder Grafiken mehr als 1.000 Worte. Auch dafür gibt es mittlerweile endlos viele Tools, die geeignet sind, gemeinsam an Entwürfen, Ideensammlungen, Architekturbildern oder Ähnlichem zu arbeiten und die Ergebnisse zu teilen.

Miro *[5]* gehört sicherlich zu den bekannteren Vertretern. Nach einer einfachen und kurzen Registrierung kann man Miro entweder im Browser oder als App verwenden.

Miro liefert schier endlos viele Templates für Charts, Agile Boards, Sticky Notes und Mindmaps mit jeweils zugehörigen Erklärungen. Ebenso sind Kommentare, Chats, Videocalls und Screen-Sharing möglich.

```
→  ~/Workspace/Private/aoc-2020 git:(mob-session) mob done
   git fetch origin --prune
   git push --no-verify origin mob-session
   git checkout master
   git merge origin/master --ff-only
   git merge --squash --ff mob-session
   git branch -D mob-session
   git push --no-verify origin --delete mob-session
README.md | 2 ++
 1 file changed, 2 insertions(+)
 👉 To finish, use

   git commit

→  ~/Workspace/Private/aoc-2020 git:(master) ✗
```

Abbildung 4: Aufgabe erledigt

Wer allerdings nur mal eben beim Videocall etwas mit einer Skizze oder Zeichnung erklären möchte, ohne dabei ein schwergewichtiges Tool zu verwenden, für den/die könnte Excalidraw *[6]* interessant sein. Es ist ein reines Browsertool, das sehr aufgeräumt und leichtgewichtig daherkommt, aber trotzdem eine Fülle von Möglichkeiten bietet *(siehe Abbildung 5)*. Zusammenarbeit und Export seien hier nur stellvertretend genannt.

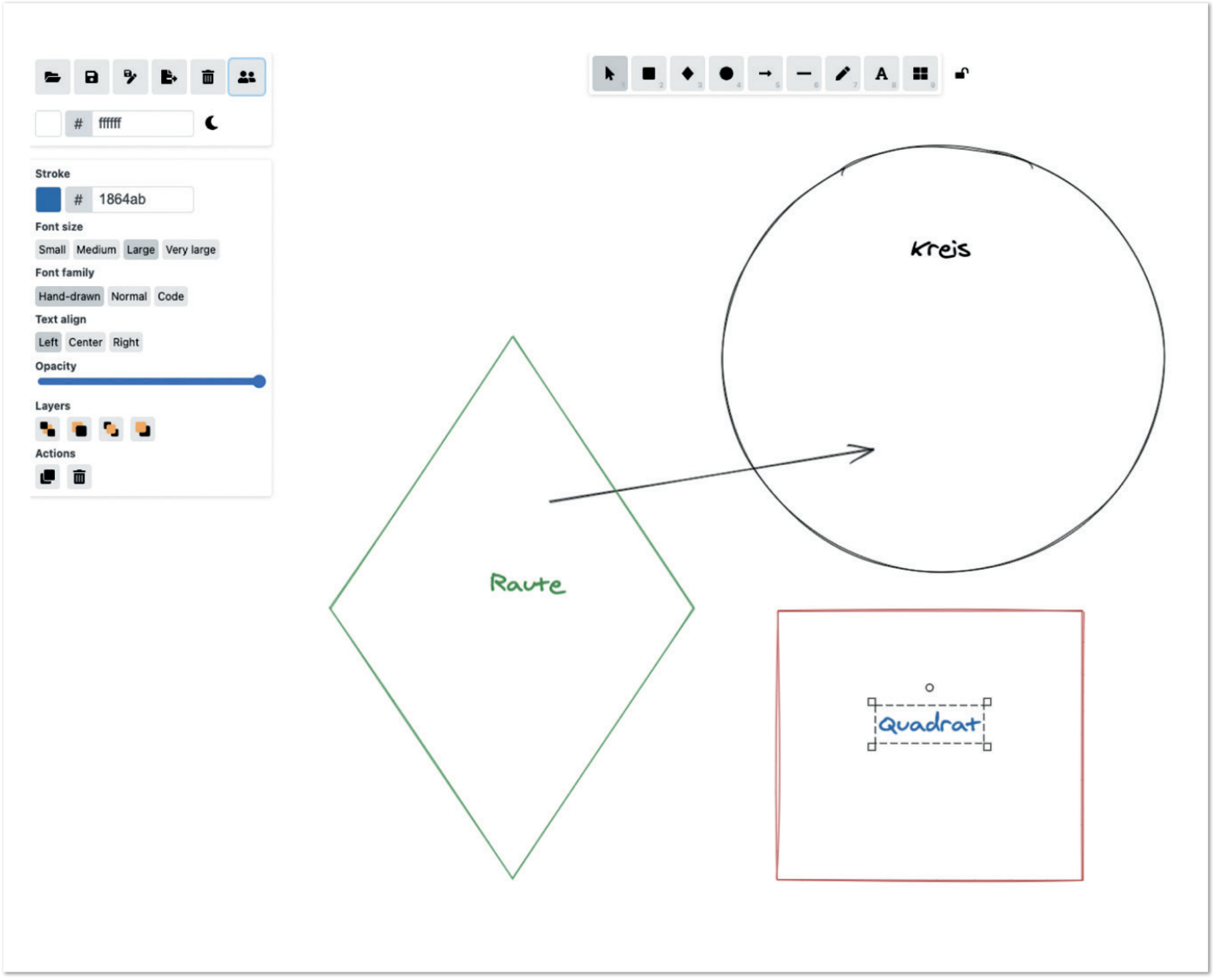


Abbildung 5: Excalidraw

Den Tag strukturieren

Vor Jahren dachte ich beim Thema Homeoffice immer, dass ich über kurz oder lang zum Nachtmenschen mutieren würde. Ganz einfach, weil tagsüber zu viele Ablenkungen lauern. Nachts ist es dann ruhig und man kann sich in seine Arbeitshöhle verziehen und vor sich hin werkeln.

Die Realität sieht allerdings anders aus. Die meisten von uns arbeiten in oder mit einem Team. Die Teammitglieder und Ansprechpartner werden einem aber nicht den Gefallen tun, ebenfalls nachts zu arbeiten.

Die Herausforderung ist, auch bei der Arbeit im Homeoffice eine gewisse Routine zu finden und den Ablenkungen zu widerstehen.

Der Weg zur Arbeit ist sehr kurz geworden. Somit auch die mentale Transferzeit. Der Weg ins Büro hatte zumindest einen guten Aspekt: Man hatte Zeit, sich aus seinem Privatleben zu lösen und sich auf den Arbeitstag einzustellen.

Genauso verhält es sich mit dem Feierabend. Auch hier fehlt die Zeit, auf dem Heimweg von der Arbeit abzuschalten und wieder im Privatleben anzukommen. Man ist irgendwie die ganze Zeit in beiden Leben gleichzeitig.

Also dachte ich mir, warum nicht einfach zur selben Zeit aufstehen wie zu Office-Zeiten, um die morgens „gewonnene Zeit", die ich jetzt nicht im Auto oder in der Bahn verbringe, sinnvoll zu nutzen. Manchmal fange ich meinen Tag mit Sport an. In jedem Fall frühstücke ich in aller Ruhe. Manchmal lese ich noch etwas oder höre mir einen Podcast an, während ich schon mal etwas Hausarbeit erledige. Und dann setze ich mich zur gewohnten Zeit an den Schreibtisch.

Zur Mittagspause ist man früher mit den Kollegen und Kolleginnen in die Kantine gegangen und danach eventuell noch eine Runde um den Block. Jetzt heißt es selbst kochen oder den Pizzadienst anrufen und dann allein eine Runde gehen. Wichtig ist nur, die Mittagspause auch wirklich zu nehmen, den Schreibtisch zu verlassen und den Status im Chat auf „Bitte nicht stören" zu stellen.

Das ständige Arbeiten in Videocalls ist anstrengender als vermutet. Meist jagt ein Videocall den anderen. Also öfter mal zehn Minuten Pause einlegen, die Beine vertreten, die Waschmaschine oder Spülmaschine bestücken oder ausräumen, mit der Katze oder dem Hund schmusen, sich einen Kaffee kochen.

Schwer ist es, zur gewohnten Feierabendzeit auch wirklich ein Ende zu finden. Ich erwische mich selbst auch viel zu oft dabei, nochmal eben etwas zu erledigen oder auszuprobieren. An diesem Thema arbeite ich noch.

Etikette

Die Zusammenarbeit per Videocall, Chat, remote PairProgramming oder MobProgramming hat ihre eigenen Gesetze. Was teilweise komplett fehlt, ist die non-verbale Kommunikation. Man sitzt halt nicht mehr gemeinsam in einem Büro und bekommt alle Schwingungen mit. Videocalls passieren nicht zufällig. Es gibt zwar Ansätze mit sogenannten Kaffeeklatsch-Calls, aber mal ehrlich: Oft genug verpasst man die Zeit oder es treffen sich dort sowieso immer dieselben Nasen.

Im letzten Jahr habe ich daher einige Sachen gelernt, die für mich bei der Arbeit im Homeoffice wichtig sind.

Morgens schicke ich immer als Erstes ein „Guten Morgen" an alle in unseren Team Slack Channel. Slack warnt mich zwar immer, dass ich damit „herumkrähe". Aber genau das ist meine Absicht. Meine Kolleg/innen sollen erfahren, dass ich ab jetzt anwesend und ansprechbar bin.

Wenn man Pause macht, in einem Meeting ist, sonst gerade nicht erreichbar ist oder nicht gestört werden will, sollte man das in den Chat-Tools kenntlich machen (vergesse ich auch ständig). Umgekehrt sollte man die Statusmeldungen der anderen berücksichtigen.

Auf Chatnachrichten sollte man keine sofortige Reaktion erwarten. Vielleicht sind die angesprochenen Personen mit anderen Dingen beschäftigt oder gerade im Programmier-Flow. Sollte aber nach einer gewissen Zeit keine Reaktion kommen, dann einfach die Nachricht durch einen Ping nochmal ins Gedächtnis rufen.

Wichtig finde ich auch, bei Videocalls die Webcam einzuschalten, da so zumindest etwas mehr non-verbale Kommunikation möglich ist. Wenn ich mir nur Avatare ansehen soll, kann ich auch gleich zum Telefonhörer greifen. Ausnahme ist natürlich, wenn es aus technischen Gründen nicht möglich sein sollte. Keine Ausrede ist, dass man seine häusliche Umgebung anderen nicht zeigen möchte. Dafür hat fast jedes Tool mittlerweile Hintergrundbilder parat.

Aufgrund der zum Teil doch nicht unerheblichen Latenzen ist es noch wichtiger, die Gesprächspartner ausreden zu lassen. Ins Wort fallen ist kritisch und führt zu Verwirrungen. Am besten gewöhnt man sich an zu sagen, wenn man mit dem, was man mitteilen wollte, fertig ist. Bei einer größeren Gruppe kann man auch per Handzeichen anzeigen, dass man etwas sagen will.

Manchmal wird es auch ruhig um Kolleg/innen. Man hat schon länger nichts von ihm oder ihr gehört und fragt sich, woran er/sie gerade arbeitet. In so einem Fall einfach mal Kontakt suchen. Das sollte allerdings nicht den Eindruck einer Kontrolle erwecken, sondern eher Interesse zeigen.

Es gibt immer mal wieder auch Unstimmigkeiten, Meinungsverschiedenheiten, Stress oder gar Konflikte, die sich auf die Beziehungen zu den Arbeitskolleg/innen auswirken. Dadurch, dass man eben nur noch per Kamera und Mikrofon (und meist auch noch unter Anwesenheit mehrerer Personen) Kontakt hat und ansonsten mit seinen Gedanken allein ist, können sich schon mal gewisse Spannungen ansammeln. Ich habe die Erfahrung gemacht, dass ein offenes, persönliches Gespräch super dabei helfen kann, diese Spannungen zu lösen. Wichtig ist nur, nicht zu lange damit zu warten.

Bei uns im Team haben wir die Regel, nach Möglichkeit über Gruppenkanäle zu kommunizieren, damit auch diejenigen, die einige Zeit nicht anwesend waren, nachlesen können, welche Themen diskutiert oder welche Entscheidungen getroffen wurden. Antworten auf Nachrichten sollten immer im Thread erfolgen, um den Zusammenhang zu wahren.

Bevor man eine Aktion durchführt, die eventuell andere in ihrer Arbeit betrifft, sollte man diese in dem entsprechenden Kanal ankündigen und nach eventuellen Einwänden fragen. Oft genug hat man nicht alles bedacht und wird dadurch vor groben Fehlern mit noch mehr Nacharbeit bewahrt.

Ich bin auch eher ein Kommunikationsmuffel. Aber im letzten Jahr habe ich gelernt, auch meine Kolleg/innen öfter nach ihrem Befinden zu fragen und auch mal abseits der Arbeit etwas von mir zu erzählen. Was mir teils ganz besonders fehlt, sind sich spontan ergebende Gespräche unter Kollegen, die teilweise ins Absurde abgleiten und oft für viel Gelächter sorgen.

Fazit

Homeoffice beziehungsweise mobiles Arbeiten ist meiner Meinung nach endgültig in unserem Arbeitsalltag angekommen und wird auch nach Corona bleiben. Jedenfalls ist das meine Hoffnung. Sowohl Arbeitnehmer/innen, Freiberufler/innen als auch Arbeitgeber/innen und Kund/innen können nur davon profitieren. Aber ich denke auch, die Mischung macht es. Manch eine/r kommt gut allein zuhause zurecht. Mach einer/einem fehlt die persönliche Interaktion. Manch eine/r will für seinen Traumjob nicht ihr/sein privates Leben aufgeben müssen und die Zeit in Hotels, im Flugzeug oder im Auto verbringen. Arbeitgeber/innen können die für sie besten Mitarbeiter/innen nun auch dann finden, wenn diese nicht in der Nähe des Firmenstandortes ihren Lebensmittelpunkt haben. Die Arbeitswelt wird diverser werden. Aber jeder einzelne muss dafür auch verantwortlicher sich selbst gegenüber werden und eben zu gegebener Zeit den Laptop zuklappen und das Handy (zumindest das dienstliche) ignorieren. Ab und zu trifft man seine Kollegen dann im Büro zu Meetings oder Veranstaltungen. Auch wenn das dann mehrheitlich ein Tag voller „Schwätzchen halten" wird, muss es nicht gleich ein unproduktiver Tag sein. Es wird ein Tag voller non-verbaler Kommunikation sein, vielleicht auch ein anstrengender, aber einer, bei dem man sich freut, seine Kollegen/Kolleginnen endlich wieder persönlich zu treffen.

Sicherlich gibt es noch viele andere Aspekte der Arbeit im Homeoffice, speziell für Arbeitnehmer/innen. Ich denke dabei beispielsweise an rechtliche Voraussetzungen, Haftungen, Sicherheit von persönlichen Daten, Arbeitsschutz, Einhaltung des Arbeitszeitgesetzes und viele mehr. Es gibt zwar keine Pflicht zum Homeoffice (außer eventuell zu Pandemiezeiten), aber meiner Meinung nach gibt es auch kein passendes Argument mehr dagegen. Jeder sollte in dem Umfeld arbeiten können, in dem sie oder er sich am wohlsten und produktivsten fühlt. Sicherlich muss man immer Kompromisse eingehen. Schließlich arbeitet man meistens in oder mit einem Team und nicht in einer idealen Welt – und das ist auch gut so!

Quellen

[1] *https://pragli.com*
[2] *https://mob.sh*
[3] *https://www.jetbrains.com/code-with-me/*
[4] *https://youtu.be/SrPqIxsfxF8*
[5] *https://miro.com/*
[6] *https://excalidraw.com/*

Birgit Kratz
mail@birgitkratz.de

Birgit Kratz ist freiberufliche Softwareentwicklerin und Consultant im Java-Umfeld. Ihre Schwerpunkte und auch ihre Leidenschaft liegen in der Anwendung agiler Entwicklungsmethoden und der Verbreitung des Software-Crafting-Gedankens. Seit vielen Jahren ist sie deshalb auch Co-Organisatorin der Softwerkskammern in Köln und Düsseldorf. In ihrer Freizeit ist sie leidenschaftliche Rennradfahrerin.

Mitglieder des iJUG

01 Android User Group Düsseldorf
02 BED-Con e.V.
03 Clojure User Group Düsseldorf
04 DOAG e.V.
05 EuregJUG Maas-Rhine
06 JUG Augsburg
07 JUG Berlin-Brandenburg
08 JUG Bremen
09 JUG Bielefeld
10 JUG Bonn
11 JUG Darmstadt
12 JUG Deutschland e.V.
13 JUG Dortmund
14 JUG Düsseldorf rheinjug
15 JUG Erlangen-Nürnberg
16 JUG Freiburg
17 JUG Goldstadt
18 JUG Görlitz
19 JUG Hannover
20 JUG Hessen
21 JUG HH
22 JUG Ingolstadt e.V.
23 JUG Kaiserslautern
24 JUG Karlsruhe
25 JUG Köln
26 Kotlin User Group Düsseldorf
27 JUG Mainz
28 JUG Mannheim
29 JUG München
30 JUG Münster
31 JUG Oberland
32 JUG Ostfalen
33 JUG Paderborn
34 JUG Passau e.V.
35 JUG Saxony
36 JUG Stuttgart e.V.
37 JUG Switzerland
38 JSUG
39 Lightweight JUG München
40 SOUG e.V.
41 SUG Deutschland e.V.
42 JUG Thüringen

iJUG Verbund

www.ijug.eu

Impressum

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

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

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

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Christian Luda
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

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

Titel, Gestaltung und Satz:
Caroline Sengpiel,
DOAG Dienstleistungen GmbH

Fotonachweis:
Titel: Bild © Bro Vector | *https://stock.adobe.com*
S. 10: Bild © robert | *https://stock.adobe.com*
S. 14+15: Bild © sergign | *https://stock.adobe.com*
S. 17: Bild © djvstock | *https://de.123rf.com*
S. 19: Bild © artinspiring | *https://stock.adobe.com*
S. 22+23: Bild © MICHAEL ZECH | *https://stock.adobe.com*
S. 25: Bild © Visual Generation | *https://stock.adobe.com*
S. 36: Bild © deepagopi2011 | *https://stock.adobe.com*
S. 42: Bild © Tierney | *https://stock.adobe.com*
S. 46: Bild © polygraphus | *https://stock.adobe.com*
S. 50+51: Bild © treety | *https://stock.adobe.com*
S. 53: Bild © zenzen | *https://stock.adobe.com*
S. 56: Bild © Anch | *https://stock.adobe.com*
S. 60: Bild © Visual Generation | *https://stock.adobe.com*

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org

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

Druck:
WIRmachenDRUCK GmbH,
www.wir-machen-druck.de

Inserentenverzeichnis

16. + 17. SEPTEMBER 2021

BERLINER EXPERTENSEMINAR

DOAG

SVEN BERNHARDT

GUIDO SCHMUTZ

Microservice/Cloud-native Apps Development

KURSÜBERBLICK

Cloud-native ist in aller Munde. Der Trend geht weg von einer monolithischen, schwerfälligen hin zu einer verteilten, leichtgewichtigen Applikationsarchitektur. In dieser neuen Welt werden Business Capabilities über einen oder mehrere sogenannter Microservices abgebildet. Im Berliner Expertenseminar vermitteln Ihnen die Referenten die Kernprinzipien der Microserviceentwicklung.

ZIELGRUPPE

Java-Entwickler und -Architekten

VORAUSSETZUNGEN

Grundlagenkenntnisse in

- Java-Entwicklung
- Containerisierung (Docker)
- Softwarearchitektur
- Domain Driven Design
- Cloud

www.doag.org/go/expsem_bernhardt-schmutz

JavaLand

2022

15. - 17. März

im Phantasialand
bei Köln

Early Bird bis
11.01.2022

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund