Ausgabe 02/2023

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 02

Java aktuell

iJUG Verbund
www.ijug.eu

Nachhaltigkeit
Ressourcen- und Energieeffizienz in der Softwareentwicklung

Energiesparmodus
Wie wir unsere eigene Energie produktiv nutzen

OAuth2
Authentifizierung für GraphQL


CloudLand
2023
DAS EVENT DER
DEUTSCHSPRACHIGEN
CLOUD NATIVE COMMUNITY
20. - 23. JUNI
im Phantasialand in Brühl
Weitere Informationen
www.cloudland.org
POP!
yay!
#CloudLand2023
Eventpartner: Heise Medien

Liebe Leserinnen und Leser,

das Motto dieser Ausgabe lautet Green Development. Nachhaltigkeit begleitet uns in nahezu jedem Aspekt unseres täglichen Lebens. Auch in der Softwareentwicklung nimmt der nachhaltige Umgang mit Ressourcen einen immer wichtiger werdenden Stellenwert ein. Unsere Autorinnen und Autoren nehmen diese Thematik unter die Lupe und geben Tipps für einen nachhaltigeren Umgang in der Informationstechnologie.

Den Auftakt in die Thematik bildet der Leitartikel von Frank Pientka. Dieser beleuchtet vorrangig die Vermeidung von unnötigem Code und Daten und gibt Anregungen, wie in der eigenen Software geringere Energie nachhaltiger genutzt werden kann. Im darauffolgenden Artikel widmet sich Michael Krämer der effizienteren Ressourcennutzung in der Softewareentwicklung. Er präsentiert Möglichkeiten zur nachhaltigeren Gestaltung einer Anwendung bei der Implementierung.

Das Autorenquartett Richard Vobl, Denis Angeletta, Nadja Hagen und Uwe Eisele zeigt ab Seite 20 Möglichkeiten auf, wie im Cloud-Native-Umfeld der Verbrauch von Ressourcen bei der Entwicklung und dem Betrieb von Softwaresystemen reduziert werden können. Franz Stefan stellt in seinem Artikel das AWS Well-Architected Framework vor, das Vor- und Nachteile bei der Erstellung von Workloads in AWS aufzeigt und so dabei unterstützt, diese zu optimieren und nachhaltiger zu gestalten.

Bei einer nachhaltigen Lebensweise denkt jeder sofort an den Umgang mit der Umwelt und knappen Ressourcen; der Fokus ist also nach außen gerichtet. Doch auch unsere eigene Energie ist eine knappe Ressource, von der unsere Produktivität abhängt. In ihrem Artikel ab Seite 37 zeigt Sabine Wojcieszak Möglichkeiten und Wege, diese dauerhaft effizient, nachhaltig und produktiv einzusetzen.

Zum Abschluss dieser Ausgabe widmen sich Heinz-Werner Hass und Francois Fernandes einem wichtigen, jedoch bei Entwickelnden oft nicht sehr beliebten Thema: der Authentifizierung von Benutzenden. Die beiden Autoren zeigen eine einfachere Möglichkeit der Authentifizierung für GraphQL mithilfe der OAuth2-Unterstützung von Spring auf.

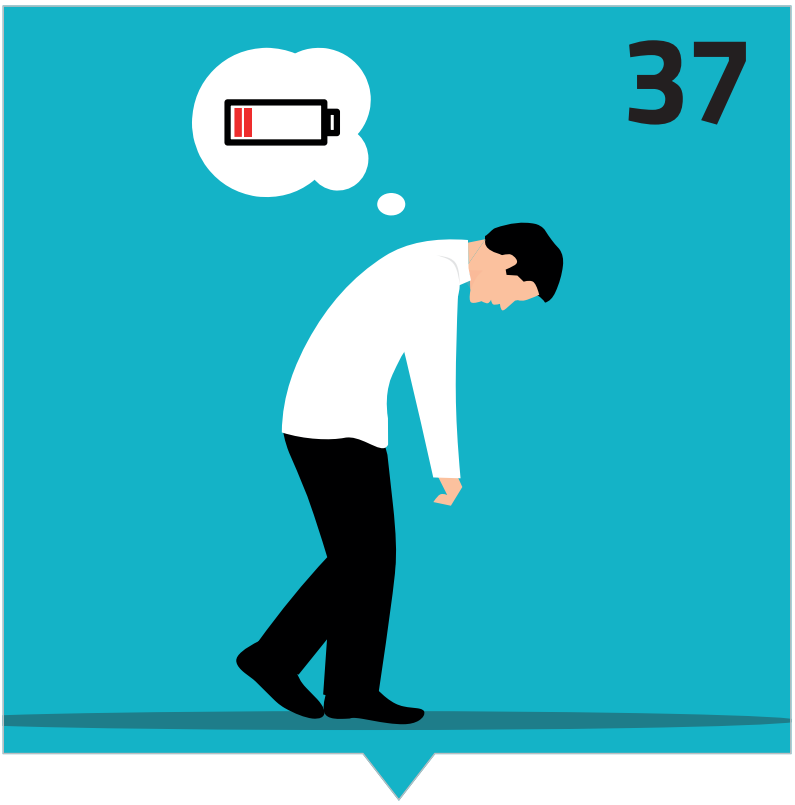
Wir wünschen euch viel Spaß beim Lesen!
Eure

Lisa Damerow
Redaktionsleitung Java aktuell

Inhalt

Reduzierung von Code und Daten für mehr Nachhaltigkeit

Ressourcenschonende Entwicklung und effizienter Betrieb von Cloud-Native-Software



Nachhaltiger Umgang mit der eigenen Energie

Einfachere Authentifizierung mit OAuth2 für GraphQL.

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

22.10.2022

JavaOne Nachlese

Die JavaOne ist nach fünf Jahren zurückgekehrt, nicht so groß wie früher, und jetzt in Las Vegas statt in San Francisco, aber viele der üblichen Verdächtigen haben sich offensichtlich wieder auf den Weg gemacht. Nur Duke war nicht zu sehen, wie Ivar Grimstad berichtet hat – hoffentlich nichts Ernstes, und er ist nächstes Jahr wieder dabei. Java ohne Duke, das geht gar nicht.

Doch es gab auch inhaltlich ein paar berichtenswerte Dinge: Oracle hat angekündigt, die GraalVM beziehungsweise genauer den für Java relevanten Code der GraalVM-Community-Edition (inklusive Native-Image-Technologie) zur weiteren Entwicklung an das OpenJDK-Projekt zu geben. Die „polyglotten" Bestandteile verbleiben jedoch bis auf Weiteres bei Oracle. Die übergebenen Teile sollen hinsichtlich der Lizenz und der Releases an das OpenJDK Java angeglichen werden, inklusive eines „Long Term Support"-Releases alle zwei Jahre, so das offizielle Statement – und zwar direkt ab 2023.

In der Keynote zählt George Saab, VP Java Development bei Oracle, die fünf Kernthemen für die Konferenz und die Zukunft von Java auf: Steigerung der Performance mit jedem Release, Stabilität (in Bezug auf Investitionen, nicht auf die „Runtime"), Sicherheit (Security), Kompatibilität und Wartbarkeit. Nichts davon überraschend, außer vielleicht der Reihenfolge. Wenn man sich die aktuell laufenden größeren OpenJDK-Projekte anschaut, die George Saab ebenfalls anspricht, passen die auch zu Performance auf Platz eins: Amber – Wartbarkeit (weniger, prägnanterer Code); Leyden – Performance (Startup-/Warm-up-Zeiten); Loom – Performance (Nebenläufigkeit); Panama – Kompatibilität (Interoperabilität mit „native" Code); Valhalla – Performance (Speicheroptimierungen); ZGC – Performance (Low-Latency Garbage Collection für Heaps bis 16 TB – nicht zu verwechseln mit Epsilon, dem „No-Op GC"). Was hier komplett fehlt, sind Stabilität und Security. Beim Thema Stabilität ist das sicher ok, weil dies über den Prozess umgesetzt wird. Gilt das für Security gleichermaßen? Es bleiben Zweifel.

Für die Keynote hat sich George Saab vielleicht aus diesem Grund externe Hilfe geholt: Olivier Gaudin, den CEO von SonarSource, bekannt durch SonarQube. Gaudin spricht logischerweise über das Tooling rund um die Java-Plattform, mit dem das Ziel „Clean Code" erreicht werden soll, insbesondere natürlich die statische Code-Analyse. Dadurch erreiche man dann nicht nur (unter anderem Security) eine Reduzierung der Risiken und Wartungskosten sowie eine höhere Langlebigkeit der Software, sondern auch die „Developer Attrition" (also die Kündigungsrate) werde reduziert – da mehr Zeit für Innovation als für das Debuggen verwendet wird.

23.10.2022

GraalVM 22.3

Die Ankündigung der GraalVM-Übergabe an das OpenJDK-Projekt ist zusammen mit der Freigabe der Version 22.3 erfolgt. Die Inhalte hatte ich noch nicht erwähnt: Neben der Unterstützung von Java 19 (virtuelle Threads in der Preview!) sind ein paar weitere interessante Features dabei – hier eine Auswahl: Einzeiler-Downloads der VM, die unter anderem für Build Pipelines nützlich sein können, wenn mal eben eine ganz bestimmte Version geladen werden soll; eine Reihe von Verbesserungen hinsichtlich des Monitorings und Debuggings, unter anderem mit einem feingranularen „--enable-monitoring" Switch, Support für jvmstat und den Java Flight Recorder in Native Images sowie mehr Unterstützung hinsichtlich Memory-Verbrauch und -Leaks. Gut für die (Überprüfung der) Sicherheit: Auch einen Schalter zum Einbetten der SBOM in die Applikationen gibt es nun, um Dependency Checker zu unterstützen.

28.10.2022

EclipseCon 2022 wieder vor Ort in Ludwigsburg

Gestern ist die EclipseCon 2022 zu Ende gegangen, die alljährlich stattfindende Hausmesse der Eclipse Foundation. Diesmal statt online wieder, wie früher, vor Ort in Ludwigsburg bei Stuttgart. Den Start machte wie immer der EclipseCon Community Day, gefolgt von der eigentlichen dreitägigen Konferenz. Das Programm war gewohnt international. Einige der Speaker/innen dürften noch Jetlag von der JavaOne gehabt haben. Anscheinend wird jetzt einiges an Reisen nachgeholt, was durch Corona in den letzten zwei Jahren nicht möglich war. Die Palette der Vorträge ist erwartungsgemäß breit, sie ist online verfügbar *[1]*.

03.11.2022

Sequenced Collections und Asynchronous Stack Traces

Es gibt zwei neue JEPs („Enhancement Proposals") für das OpenJDK: „Sequenced Collections" und ein „Asynchronous Stack Trace VM API". Ersteres ist ein neues, vereinheitlichendes API für Collections mit definierter Reihenfolge der Elemente, auf die vorwärts und rückwärts zugegriffen werden kann. Letzteres soll Profiler dabei unterstützen, ohne (beziehungsweise mit nur geringem) Einfluss auf Laufzeit und Speicherbedarf Stack Traces einzusammeln und auch über Signal-Handler verfügbar zu sein. Als Vorbild dient das AsyncGetCallTrace API. Das neue AsyncGetStackTrace API ist explizit nicht für den Einsatz in Produktion vorgesehen, da es die VM „crashen" kann, es soll aber hart daran gearbeitet werden, dieses Szenario möglichst unwahrscheinlich zu machen. Für welches Release die neuen Features vorgesehen sind, steht noch nicht fest, für Java 20 wird es jedenfalls eng (der für neue Features wichtige Meilenstein „Rampdown Phase 1" ist bereits am 8. Dezember).

05.11.2022

JDK 20

Was sind denn die bislang geplanten Features für das Frühjahrsrelease? Java SE 20 ist ja der Vorläufer des nächsten Long-Term-Support-Releases, und in erster Linie sind es daher auch (fortgesetzte) Previews, die hoffentlich bis Java 21 stabil sind: Zunächst das altbe-

kannte „Pattern Matching for switch“ (schon in der 4. Preview-Version) und Record Patterns (2. Preview). Virtual Threads und das „Foreign Function & Memory API“ (je als zweite Preview) gehören wohl auch dazu, wobei sich das JEP-Dashboard im OpenJDK Jira und die offizielle OpenJDK-Seite *[2]* nicht ganz einig sind. Daneben wird das „Structured Concurrency“ API (mehrere Threads als eine „unit of work“ behandeln) wohl ein zweites Mal als Inkubator-Version dabei sein, mit nur einer minimalen Änderung gegenüber dem ersten Vorschlag in Release 19. Dass noch weitere JEPs dazukommen, ist nicht so wahrscheinlich, auch wenn es eine ganze Reihe von Kandidaten gibt.

08.11.2022

Jakarta EE – was kommt nach der 10?

Die Diskussionen für das nächste Jakarta-Release sind in vollem Gang. Die Pläne, Java 11 nicht weiter zu unterstützen, lassen sich in einem 10.1-Release eigentlich nicht umsetzen, da es ein „Breaking Change“ wäre. Andererseits müssen Implementierungen von EE 10 auch Java SE 17 unterstützen, und damit ist die Abkündigung des SecurityManager verbunden. Mit Java 18 ist er bereits per Default deaktiviert, was mindestens mal sehr lästig für Hersteller und Anwender ist. Also schnell ein Release 11, um das Problem loszuwerden? Oder vielleicht erst ein Milestone-Release, um ein produktionsreifes Release 11 mit neuen Features vorzubereiten? Vielleicht sogar Milestone-Releases als generelles Muster in der Zukunft? Dazu müssten die Einzelspezifikationen aber mitziehen, und das könnte bei größerer Release-Frequenz schwierig werden. Noch scheint nichts entschieden, doch die Tendenz geht wohl zumindest diesmal zu einem Milestone-Release.

An der Feature-Front ist bislang nur die Ideensammlung für „EE Next“ weitergeführt worden, die ich im letzten Tagebuch schon vorgestellt hatte *[3]*.

10.11.2022

WildFly 27 auf Jakarta EE 10

Red Hats WildFly Application Server ist in Version 27 freigegeben worden. Diese ist als kompatibel mit allen drei Jakarta-EE-20-Profilen zertifiziert worden (Web, Full und das neue Core Profile). Für alle drei Profile gibt es jeweils Versionen für Java SE 11 und 17. Darüber hinaus gibt es unter anderem eine Preview für Micrometer-Unterstützung, Verbesserungen im Tooling insbesondere für OpenShift (S2I Builder Images) und natürlich Upgrades vieler Komponenten wie Hibernate Search und RESTEasy.

JavaLand 2023 – Programm online

Das Programm der JavaLand-Konferenz 2023 ist online. Sie wird vom 20. bis 23. März wieder im Phantasialand stattfinden. Am Montag, dem 20. März, finden die JavaLand 4 Kids und eine Community-Unconference statt, am 23. wieder der Schulungstag, die eigentliche Konferenz ist wie gewohnt dienstags und mittwochs. Insgesamt sind es deutlich mehr als 160 Vorträge, Workshops, Community-Aktivitäten und viele mehr, die diesmal aus der Rekordzahl von 550 Einreichungen ausgewählt wurden! Hier wird also einiges geboten. Vielleicht kommt die JavaLand 2023 dann auch wieder an die Besucherzahlen vor 2020 heran oder kann sie sogar übertreffen.

11.11.2022

Quarkus 3.0 für Februar 2023 geplant

Für Februar 2023 ist Quarkus 3.0 geplant. Damit werden dann unter anderem die neuen Versionen der Jakarta-EE-10- und MicroProfile-6-APIs unterstützt; außerdem Hibernate ORM 6 und das vor Kurzem von der IETF freigegebene, auf UDP basierende HTTP/3 (aka QUIC). Minimum-Java-Version ist 11, empfohlen wird aber 17. Getestet wurde jedoch vermutlich auch mit Java 19, denn – hatte ich schon mal Virtual Threads erwähnt? – es gibt bereits seit Version 2.10 experimentelle Unterstützung für diese, die nun in Vorbereitung auf Java SE 21 sukzessive verbessert werden soll. Hinsichtlich der möglichen „Breaking Changes“ beim Upgrade auf 3.0 wird eine Werkzeug-Unterstützung versprochen, mit der sich dieses möglichst automatisch durchführen lässt. Das ist insbesondere deshalb wichtig, weil Quarkus das Jakarta EE 9 Release ausgespart hat, in dem die javax-Packages zu jakarta umbenannt wurden, und direkt auf EE 10 setzt.

14.11.2022

Bund fördert OSS-Projekte

Mal etwas aus der Politik, das aber für die vielen Java-Open-Source-Projekte interessant ist: Der Bund will Open-Source-Projekte mit dem kürzlich gegründeten Sovereign Tech Fund fördern. Das ist wohl eine Einsicht aus den verschiedenen Schreckensszenarien der jüngeren Vergangenheit, von denen log4shell eines der prominentesten war. Im Amtsdeutsch geht es darum, offene digitale Basistechnologien (ODB) zu fördern. Details zum Programm gibt es hier *[4]*.

24.11.2022

Spring 6 freigegeben

Auch Spring bringt neue Major Releases raus: vor ein paar Tagen das Spring Framework 6.0, nun das darauf aufbauende Spring Boot 3.0, mit Java SE 17 und den Jakarta-EE-9-APIs als Minimalversion. Ein paar Dinge dazu hatte ich ja schon im letzten Tagebuch geschrieben. Unterstützung für GraalVM Native Images ist eines der prominentesten neuen Features, vorausgesetzt wird hier auch die neueste GraalVM-Version 22.3. Im Bereich „Observability“ – Micrometer, Prometheus – hat sich auch einiges getan. Ein direkter Blick auf die Release Notes lohnt sich, es ist doch einiges neu *[5]*.

28.11.2022

Adoptium und Temurin arbeiten an sichere(re)n Builds

Das Thema Java und Security hatten wir ja auch schon in diesem Tagebuch. Wer sich dafür interessiert, wie Adoptium und das Teilpro-

jekt Temurin daran arbeiten, eine sichere Infrastruktur und sichere Prozesse für seine Java-Releases zu schaffen, dem seien die unten gelisteten Blog-Artikel empfohlen. Die Stichworte sind: SLSA (das Community-Projekt „Supply chain Levels for Software Artifacts", ausgesprochen „Salsa" *[6]*) und SSDF (das „Secure Software Development Framework" der amerikanischen Standards-Behörde NIST *[7]*).

AWS Lambda SnapStart

Für diejenigen, die sich mit AWS-Lambdas beschäftigen und versuchen, etwa mit Native Images die Startup-Zeiten für Java-Prozesse zu optimieren, könnte das neue SnapStart als Alternative interessant sein. Das Prinzip ist, die Init-Phase einer Lambda-Funktion vorab aufzurufen, die für ein Framework wie Spring Boot, Quarkus oder Micronaut schon mal einige Sekunden dauern kann, und dann als „vorgewärmten" Snapshot (Speicher und Festplatte) in einem Cache zu halten, um diesen bei Bedarf schnell in die Ausführungsumgebung zu kopieren. Damit lassen sich Startup-Zeiten wohl von einigen Sekunden auf wenige hundert Millisekunden senken, zumindest für einige Beispiele, die ich bislang gesehen habe (von AWS selber *[8]*, oder beispielsweise in Adam Biens Blog).

07.12.2022

MicroProfile 6.0 Release

Eigentlich wollte ich zum Abschluss dieser Tagebuch-Ausgabe noch schreiben, dass MicroProfile 6.0 nun wie angekündigt freigegeben wurde. Aber ganz so einfach ist es dann doch wieder nicht. Im Release-Review hat es einige Gegenstimmen gegeben, und aktuell ist nicht mal klar, ob die nötige Mehrheit erreicht wurde – daher noch keine Freigabe. Nein, es werden keine Wahlmails mehr weit nach dem Stichtag erwartet. Die genau zehn Stimmen sind auch ausgezählt, bei sechsmal Ja, dreimal Nein und einer Enthaltung. Aber da die Abstimmungen normalerweise sehr deutlich ausfallen, ist hier noch ein genauer Blick in die Statuten der Working Group nötig. Naja, das wird sich sicher in den nächsten Tagen klären. Die Diskussionen wohl noch nicht – es ging im Wesentlichen um zwei Fragen: Erstens, ob für eine MP-Version unterschiedliche Jakarta-Versionen als Grundlage dienen können. Ganz konkret, ob MP-6.0-kompatible Implementierungen auch auf Basis der älteren Jakarta-EE-9.1-APIs erlaubt sind (was ein bisschen „tricky" ist, da MP 6.0 eine Abhängigkeit zum Jakarta EE Core Profile hat, das es erst in EE 10 gibt). Zweitens ging es um das Verhältnis zwischen MP Metrics und dem sich zum De-facto-Standard entwickelnden OpenTelemetry (und konkret der Frage, ob MP Metrics 5.0 nur noch optionaler Bestandteil sein soll). Der Plan ist aber, dass diese Fragen mit den beiden Folge-Releases 6.1 (Februar 2023) und 7.0 (Juni 2023) geklärt werden.

Referenzen:

[1] *https://www.youtube.com/playlist?list=PLy7t4z5SYNaRoQ4o40i6zfD0ZuoenX7ph*
[2] *http://www.openjdk.org/projects/jdk/20/*
[3] *https://docs.google.com/document/d/1m-dkvbL0iFFzitO4vt1S-Vq6GGSJyFdCDM2NU_FzGS10/edit#*
[4] *https://sovereigntechfund.de/SovereignTechFund_Machbarkeitsstudie.pdf*
[5] *https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-3.0-Release-Notes*
[6] *https://adoptium.net/de/blog/2022/11/slsa2-temurin/*
[7] *https://adoptium.net/de/blog/2022/11/secure-software-development/*
[8] *https://aws.amazon.com/de/blogs/aws/new-accelerate-your-lambda-functions-with-lambda-snapstart/*

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

Andreas Badelt ist stellvertretender Leiter der DOAG Cloud Native 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 Java Community, die ihren Bereich erweitert hat und nun Cloud Native Community heißt. 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 habe ich versprochen, erst dann einen Artikel zum Thema Jakarta EE 10 zu schreiben, wenn ihr das Ganze auch auf einem Application Server testen könnt, der für den Produktivbetrieb freigegeben und mit professionellem Kundendienst abgedeckt ist. Letztendlich seid ihr ja alle Profis und benutzt Jakarta EE nicht nur als Hobby. Stabilität und kurzfristig verfügbare Hilfestellung bei Problemen ist daher unabdingbar. Lange Zeit sah es ja entsprechend mau aus auf dem Markt. Zwar wollen alle großen Hersteller mitreden, wenn es um die Steuerung der Jakarta-EE-Arbeitsgruppe bei der Eclipse Foundation geht, und nominell ist auch die Beteiligung der Hersteller bei der Formulierung der APIs und Spezifikationen zumindest deklarativ vorhanden. Mit der Lieferung von kompatiblen Produkten sind diese dann jedoch eher zögerlich. Völlig unverständlich, da sie ja seit Jahren „an der Quelle" sitzen und weder Inhalte von Specs und APIs noch deren Veröffentlichungstermin überraschend waren. Ein wirklich interessiert agierender Hersteller hätte daher bereits während der Diskussions- und Abstimmungsphase von Jakarta EE 10 sein Produkt an den vereinbarten Stand der Technik angepasst und könnte somit postwendend am Tag der Veröffentlichung ausliefern.

Ich überlasse es jedem Hersteller, der das nicht so getan hat, sich seinen Kunden gegenüber mit mehr oder minder fadenscheinigen Ausreden zu rechtfertigen. Mein Tipp an dieser Stelle: Jakarta EE hat das Ziel, eine Anwendung portabel zu machen, das heißt, die Bindung an ein bestimmtes Produkt beziehungsweise einen bestimmen Hersteller weitestgehend aufzuheben, somit also den Wechsel zu einem anderen Produkt oder Hersteller zu erleichtern. Ist zum Zeitpunkt des Erscheinens dieses Artikels für „euren" Application Server immer noch kein kommerziell unterstütztes, Jakarta-EE-10-zertifiziertes Update verfügbar, dann stimmt einfach mit den Füßen ab. Um euch die Entscheidung zu erleichtern, „wohin" die Füße euch tragen sollten („woher" ihr kommt, wisst ihr ja selbst), habe ich mir heute mal auf der offiziellen Jakarta-EE-10-Downloadseite *[1]* angeschaut, was ihr (Stand heute) denn als Ziele haben könntet.

- Apache TomEE 9 ist noch nicht freigegeben und hat zudem nur EE 9.1 als Ziel, nicht EE 10.
- Eclipse GlassFish 7 ist noch nicht freigegeben, hat aber zumindest das Ziel EE 10.
- IBM WebSphere Liberty 22 ist lediglich kompatibel zu EE 9.1.
- Open Liberty 22 implementiert entsprecht auch nur ausgewählte Features von Jakarta EE 10.
- Möglicherweise bin ich ja einfach blind, aber welche Version von Jakarta EE Red Hats JBoss unterstützt, konnte ich auf dessen Produktseite nicht finden.
- Oracle WebLogic 14 implementiert laut Datenblatt lediglich Jakarta EE 8.
- Payara 6 erschien vor Kurzem und ist EE-10-zertifiziert. Leider war die Zeit bis zum Redaktionsschluss zu knapp für einen sinnvollen Test.
- Weniger bekannte Produkte darf gerne jemand anderes beurteilen, der die Zeit und Muße dazu hat. Zuschriften und Artikelanfragen herzlich willkommen!

Abgesehen davon übrigens ein weiterer Tipp: Wer an Neuentwicklungen denkt, sollte erst gar nicht mehr zu einem vollwertigen Application Server greifen. Application Server sind ein Anachronismus.

Sie entstanden vor über 20 Jahren aufgrund eines damals allgemein verbreiteten Paradigmas. In der heutigen Zeit denken Architekten aber ganz anders und Application Server sind eher ungeeignet (zu umständlich, zu fett, zu behäbig, zu schlecht zu skalieren etc.), um mit den heutzutage vorherrschenden Denkmustern wie Microservices schrittzuhalten. Stattdessen hat sich die Erkenntnis herauskristallisiert, Jakarta EE zunehmend als Sammlung von alleinstehenden APIs zu sehen und nur jene Bibliotheken zu verwenden, die man im jeweiligen Microservice auch wirklich benötigt. Beispielsweise lediglich JAX-RS. Diesem Trend folgend, ist etwa JAX-RS direkt auf Java SE ausführbar als Teil der eigenen Anwendung. Für viele Services genügt das bereits und ein Vendor-Lock-in umgeht man damit geschickt, denn die Bibliotheken, die man wählt, müssen – Jakarta EE sei Dank! – nicht einmal vom gleichen Hersteller stammen. Man kann problemlos beispielsweise Jersey (JAX-RS, Oracle) mit Hibernate (JPA, Red Hat) kombinieren. Wer mehr Bibliotheken braucht, greift zu einem Framework wie Quarkus oder Helidon. Leider begibt man sich bei Frameworks oft in ziemliche Abhängigkeit von diesem beziehungsweise dessen Hersteller. Wer das nicht will, ist eventuell mit einem Mittelweg gut beraten: Jakarta EE Core Profile. Über dieses werden wir im Rahmen der Berichterstattung über Jakarta EE noch ausführlicher berichten.

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.

Energiesparen beim Entwickeln von Software: Wie kann meine Software grüner werden?

Frank Pientka

Wäre die IT-Industrie ein Land, so würde sie mit einem groben jährlichen Energieverbrauch von 2.000 TWh auf Rang drei der Länderliste stehen. Durch die fortschreitende Digitalisierung und Cloudnutzung wird zwar die Effizienz gesteigert, jedoch auch der damit einhergehende Energieverbrauch. Der Anteil der IT-Industrie am globalen Energieverbrauch wird deshalb bis 2030 auf 20 % steigen.
Keine Software oder Daten sparen so viel Energie ein wie die, die keine benutzen. Hier geht es vor allem darum, unnötigen Code und Daten zu vermeiden. In diesem Artikel sollen ein paar Anregungen gegeben werden, wie Software grüner werden kann.

Wie grün ist dein Code?

Wie effizient eine Software ist, hängt einerseits von ihrer algorithmischen Komplexität, aber auch von deren Ressourcennutzung ab. Hier sollte man wissen, wie hoch der Ressourcenbedarf an Speicher, Netzwerk, CPU und RAM ist, um dafür zu sorgen, dass dieser ausreichend, aber auch nicht zu viel vorhanden ist *(siehe Abbildung 1)*.

Der effizienteste Code ist der, der nicht existiert. Oft wird Code produziert, der später nie verwendet wird. Gleiches gilt für extern eingebundene Bibliotheken. Das ist nicht nur ein Sicherheitsproblem, sondern auch Verschwendung von Ressourcen. Deswegen sollte man immer prüfen, ob unnötige oder veraltete Softwareteile existieren, und diese regelmäßig entfernen, wenn sie keinen Nutzen mehr erzeugen.

Bei der Erstellung von Algorithmen und Datenstrukturen sollte man die am besten passende Lösung wählen und nicht die aufwendigste Lösung. So ist beim Zero-Trust-Konzept in einer Private Cloud energietechnisch fraglich, ob wirklich jeder Ost-West-Netzwerkverkehr mit mTLS verschlüsselt werden muss. Besonders bei der Verwendung von kryptografischen Operationen sollte man überprüfen, ob hier die hardwarenahen Operationen (etwa für AES-Verschlüsselung, ChaCha20-Chiffren oder Zufallszahlenerzeugung) verwendet werden können und diese in neueren Java-Laufzeitumgebungen oder Kryptographie-Providern *[14]* aktiviert sind. Um nicht zu viel Aufwand für die Ver- und Entschlüsselung zu haben, sollte man überlegen, ob alternative Verfahren mit homomorpher Verschlüsselung oder ein signiertes Hashing nicht ausreichend sicher sind oder man wirklich Post-Quanten-Algorithmen oder Blockchains braucht. Gleiches gilt für Komprimierung. Oft ist gar nicht klar, welche Rechenaufwände durch einzelne Operationen ausgelöst werden. Hier helfen Flammengraphen, etwas Licht in das Dunkel zu bringen, um einen Ansatz zu finden, um den dahinterstehenden Energieverbrauch zu optimieren.

Wie grün sind deine Daten?

Neben dem Code sollte auf den korrekten Umgang mit Daten ein Auge geworfen werden. Bei sinkenden Speicherkosten ist die Versuchung groß, nie Daten zu löschen. Doch selbst bei ungenutzten Daten fallen Verwaltungs- und damit Energiekosten an, wie etwa bei Sicherungen, Indizierung und Zugriff.

Bei größeren und sich ändernden Daten ist eine Datenklassifizierung nach Zugriffshäufigkeit und Wichtigkeit hilfreich. Bei hierarchischen Speichersystemen können so die Daten automatisch auf günstigere und weniger energieintensive Speichermedien, wie Bänder oder DVDs ausgelagert werden. Für einen Objektspeicher wie AWS S3 ist so ein automatisches Lebenszykluskonzept bereits eingebaut und muss nur noch an die eigenen Anforderungen konfiguriert werden. Gerade in der Cloud ist ein Wechsel auf verschiedene Speichersysteme mit passenden Leistungsparametern oft einfacher möglich als bei On-Premises-Systemen, bei denen die Auswahl an unterschiedlichen Speichersystemen eher begrenzt ist.

Die zur Speicherung passenden Datenzugriffsverfahren spielen neben dem Cachen von häufig verwendeten Daten, wie bei den Schlüsseltabellen, eine große Rolle. Wenn man schnell startende, serverlose Datenbanken verwenden kann, muss beispielsweise nicht die ganze Zeit ein komplettes Datenbankcluster laufen, sondern kann auf Bedarf hochgefahren und erweitert werden. Eine Umspeicherung operationaler Daten in für die Analyse geeignete Datenmodelle und -Systeme kann nicht nur das operationale System entlasten, sondern erleichtert auch den effizienten Zugriff für Auswertungen und Berichte. Statt kleinteiliger Realtime- bietet die Batch-Verarbeitung bei großen Datenmengen oft mehr Optimierungsmöglichkeiten und ist energieeffizienter. Eine energieeffiziente Speicherung von Daten setzt mehrere Schritte und Verfahren voraus. Diese dürfen jedoch nicht zulasten anderer Qualitätsziele wie Verfügbarkeit, Skalierbarkeit, Sicherheit und Performance gehen *(siehe Abbildung 2)*.

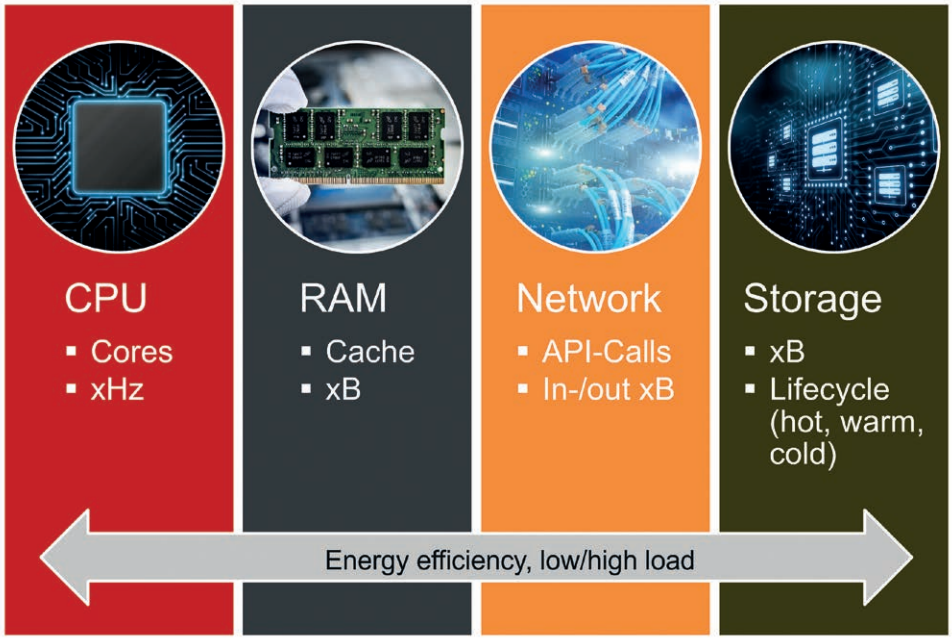


Abbildung 1: Optimierungsbereiche (CPU, RAM, Netzwerk, Speicher)

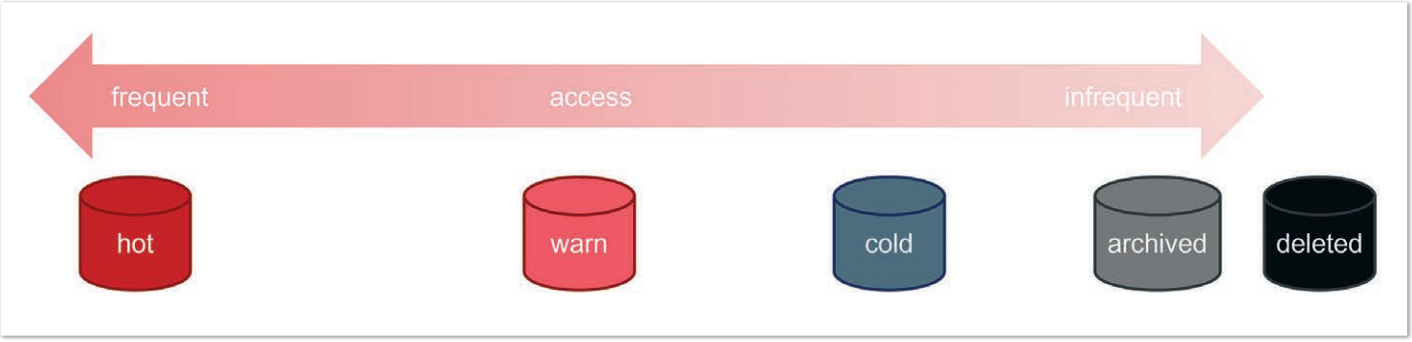


Abbildung 2: Daten-Lebenszyklen

Wie grün ist deine Website?

Am einfachsten, da öffentlich verfügbar, ist der Energieverbrauch einer Website zu messen, wenn man einmal von den auch beteiligten Backenenddiensten absieht. Der jährlich erscheinende Web-Almanach hat 2022 *[1]* erstmals zu nachhaltiger Websitegestaltung einige Zahlen und Empfehlungen zusammengestellt. Diese beginnen damit, ungenutzten Code oder Bilder nicht zu übertragen und kaputte Links zu entfernen. Die Verwendung von effizienten Übertragungsprotokollen *(http/2 und 3)* oder Kompressionsalgorithmen (zum Beispiel Brotli, ZSTD) für größere Medien führt zu einer schnelleren und effizienteren Kommunikation.

Generell ist es hilfreich, häufig verwendete Inhalte möglichst nah am Nutzer zu cachen (CDN, PWA offline, Cache Control Header, Favicon, Fonts) oder auf Bedarf nachzuladen (progressive Enhancement, lazy Loading). Eine Orientierung an den Google Chrome Core Web Vitals *[2]* und den in diesem Umfeld vorhandenen Werkzeugen helfen gleichzeitig mit der Geschwindigkeit einer Website, diese auch energieeffizient zu gestalten.

Trotz moderner Übertragungsprotokolle lohnt sich die Minifizierung von CSS/JavaScript weiterhin, da dabei für die Verarbeitung unnötige Zeichen entfernt werden. Die Verwendung von effizienten Medienformaten (WebP/AVIF) in einer ausreichenden Auflösung trägt zu einer effizienten Verarbeitung bei. Gerade die Einbettung von hochauflösenden und automatisch startenden Videos auf der Startseite ist nicht nur ein Ärgernis für den Nutzer, sondern eine Verschwendung von wertvollen Ressourcen. Hier sollte man das werbefinanzierte Monetarisierungsmodell vieler Apps und Seiten über aufdringliche Werbung klimatechnisch kritisch sehen.

Ausgewachsene CMS-Systeme mit dynamischen Webseiten sind zwar bequem, jedoch für einen effizienten Betrieb ungeeignet. Hier werden statische Codeseitengeneratoren (JAM-Stack) immer beliebter. Sie haben den Vorteil, dass wirklich nur die benötigten Inhalte konsistent und effizient ausgeführt werden, sodass zu deren Hosting oft ein CDN oder einfacher S3-Speicher reicht und kein permanent laufender Webserver benötigt wird. Deswegen verwendet die mit einem Solar-Balkon-Kraftwerk betriebene Website *[3]* auch einen statischen Seitengenerator (Pelican). Diese ist nur dann auch verfügbar, wenn genügend Strom durch die Sonne zur Verfügung gestellt wird. Das ist sicher ein extremes Beispiel, wie grüne Websites betrieben werden können, da die Hyperscaler viele Websites effizienter betrieben können als ein alter Webserver auf dem Balkon. Andererseits ist der Energieverbrauch hier direkt messbar und optimierbar.

Auf der Seite der Green Software Foundation gibt es nicht nur die JavaScript-Bibliothek CO2.js *[4]* (*[5]*, die das Sustainable-Web-Design-Modell und das OneByte-Modell des Shift Project verwendet), um den eigenen CO2-Verbrauch seiner Website abschätzen zu kön-

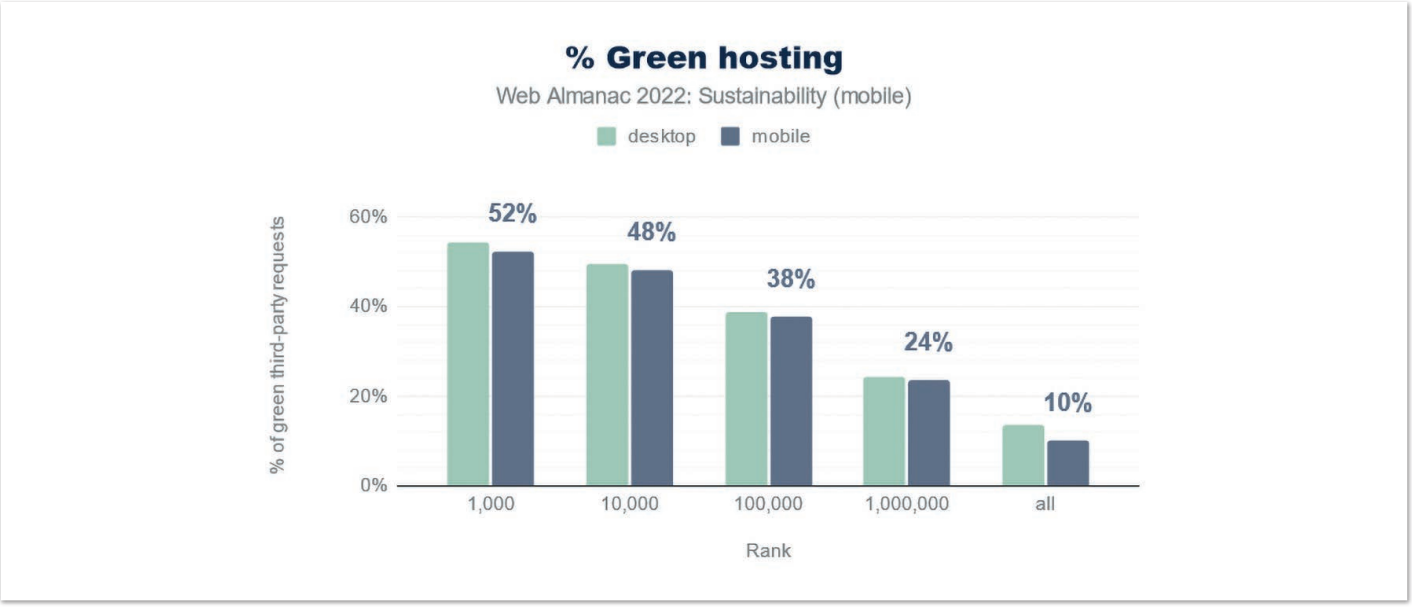


Abbildung 3: grünes Hosten (Web Almanac 2022)

nen, sondern seit 2006 die Möglichkeit *[6]*, seine Website, die mit grüner Energie läuft, zu registrieren. Die Anzahl der Seiten in deren Datenbank nimmt zu, leider noch auf niedrigem Niveau und eher bei kleinen Seiten. So haben nur zehn Prozent aller Webseiten sich für „grünes Hosten" entschieden *(siehe Abbildung 3)*.

Wie grün ist deine Cloud?

Laut einer Bitkom-Studie *[7]* lag der Energiebedarf von Rechenzentren in Deutschland 2020 bei 16 Milliarden Kilowattstunden, das sind rund drei Prozent des gesamten Stromverbrauchs in Deutschland.

Die meisten Rechenzentrumskapazitäten in Deutschland befinden sich in Frankfurt, sodass deren Energieverbrauch inzwischen schon zirka 20 Prozent des Stromverbrauches der Stadt Frankfurt ausmacht. Bei steigender Tendenz ist man bestrebt, die Abwärme neuer Rechenzentren dort zum Heizen einzelner Stadtteile zu verwenden. Noch besser ist es, noch weniger Abwärme und damit Energie für Kühlung zu benötigen.

Der Bereich, in dem am meisten eingespart werden kann, ist beim „grünen Hosten" in der Cloud. Die Regierung hat zwar im Koalitionsvertrag festgelegt, für IT-Beschaffungen des Bundes Zertifizierungen wie den Blaue-Engel-Standard *[8]* für die Nutzung erneuerbarer Energien zu verwenden, doch nur 55 von 190 Rechenzentren des Bundes verwenden den Blauen Engel. Außerdem werden die Rechenzentren des Bundes im Vergleich zu Cloud-Anbietern wie Google, Microsoft oder Amazon weniger umweltfreundlich betrieben. Deren Power-Usage-Effectiveness-Faktor (PUE=1 + Nicht-IT-Energie/IT Energie) liegt, je nach Baujahr, zwischen 1,30 und 1,60. Zum Vergleich erreicht Google für alle seine Rechenzentren 1,09. Im Durchschnitt liegt der typische PUE-Wert für Bare Metal über 1,8, für virtuelle Maschinen bei 1,5 und für Container bei 1,1.

Scale2Zero – weniger ist mehr

Das zeigt, dass ein Wechsel zu Containern oder serverlosen Anwendungen die Möglichkeit schafft, diese effizienter zu betreiben, wenn man es schafft, diese dynamisch zu skalieren und schnell zu starten und zu beenden. Das grundsätzliche Potenzial, das in der Virtualisierung zu einer besseren Auslastung der Ressourcen führen kann, muss jedoch mit einer starken Governance verbunden werden, da man sonst mit Jo-Jo-Effekten und einer steigenden Zahl von Zombie-VMs kämpfen muss, die die potenziellen Einsparungseffekte schnell zunichtemachen.

Deswegen sollten in der Cloud die vorhandenen Kostenoptimierungswerkzeuge verwendet werden, um für die Workloads angemessene Ressourcen zu finden. Hier kann der Wechsel auf kleinere und modernere Instanzen oder die Verwendung von ARM-Prozessoren erstaunliche Einsparungen bringen. Neben dem initialen Rightsizing spielen dynamische Autoscaler eine immer größere Rolle. Weniger ist auch hier oft mehr! Cloud-native Frameworks wie Quarkus verbrauchen nicht nur weniger Ressourcen für kurzlaufende Prozesse, sondern sie helfen auch mit kurzen Startzeiten, diese auf Bedarf schnell hoch- und wieder herunterzufahren. Anregungen, wie man mit Java und Quarkus grünere Software schreiben kann, hat Red Hat selbst auch veröffentlicht *[9]*. Wer noch tiefer in die Analyse seines Java-Source-Codes einsteigen möchte, dem sind Diagnoseansichten mit Flame Graphs *[10]* oder der Java-Agent JoularJX *[11]* weiterempfohlen.

Fazit

Es bleibt ein Paradoxon der Digitalisierung, dass Geräte immer smarter werden und damit auch die Optionen, um Energie effizienter zu verwenden. Andererseits steigt durch die fortschreitende Digitalisierung und damit Cloud-Nutzung der Energieverbrauch weiterhin. Wie in diesem Artikel gezeigt, gibt es viele Stellschrauben [1, 12, 13], um weniger Energie nachhaltiger zu nutzen. Die Politik und die IT-Industrie sind gefordert, mit gutem Beispiel voranzugehen, um mehr Augenmerk auf den effizienten Energiebetrieb von Systemen zu legen.

Referenzen:

[1] Web Almanac 2022 Chapter 20 Sustainability *https://almanac.httparchive.org/en/2022/sustainability*

[2] Core Web Vitals Sustainability *https://ecoping.earth/blog/core-web-vitals-and-sustainability*

[3] *https://solar.lowtechmagazine.com*

[4] CO2.js Bibliothek *https://developers.thegreenwebfoundation.org/co2js/explainer/methodologies-for-calculating-website-carbon/*

[5] *https://developers.thegreenwebfoundation.org/co2js*

[6] Green Domains Datasets *https://www.thegreenwebfoundation.org/green-web-datasets*

[7] Positionspapier nachhaltige Rechenzentren in Deutschland *https://www.bitkom.org/sites/main/files/2022-10/2210-Positionspapier-Nachhaltige-Rechenzentren.pdf*

[8] Blauer Engel *https://www.blauer-engel.de/de/produktwelt/rechenzentren*

[9] Writing greener Java applications *https://www.redhat.com/rhdc/managed-files/mi-greener-java-applications-detail-f32147pr-202211-en.pdf*

[10] Einführung in Flame Graphs *https://github.com/brendangregg/FlameGraph*

[11] JoularJX a Java-based agent for power monitoring at the source code level *https://www.noureddine.org/research/joular/joularjx*

[12] Green Software Foundation *https://greensoftware.foundation*

[13] Principles of Green Software Engineering *https://principles.green https://learn.greensoftware.foundation/practitioner/carbon-awareness*

[14] Amazon Corretto Crypto Provider *https://github.com/corretto/amazon-corretto-crypto-provider*

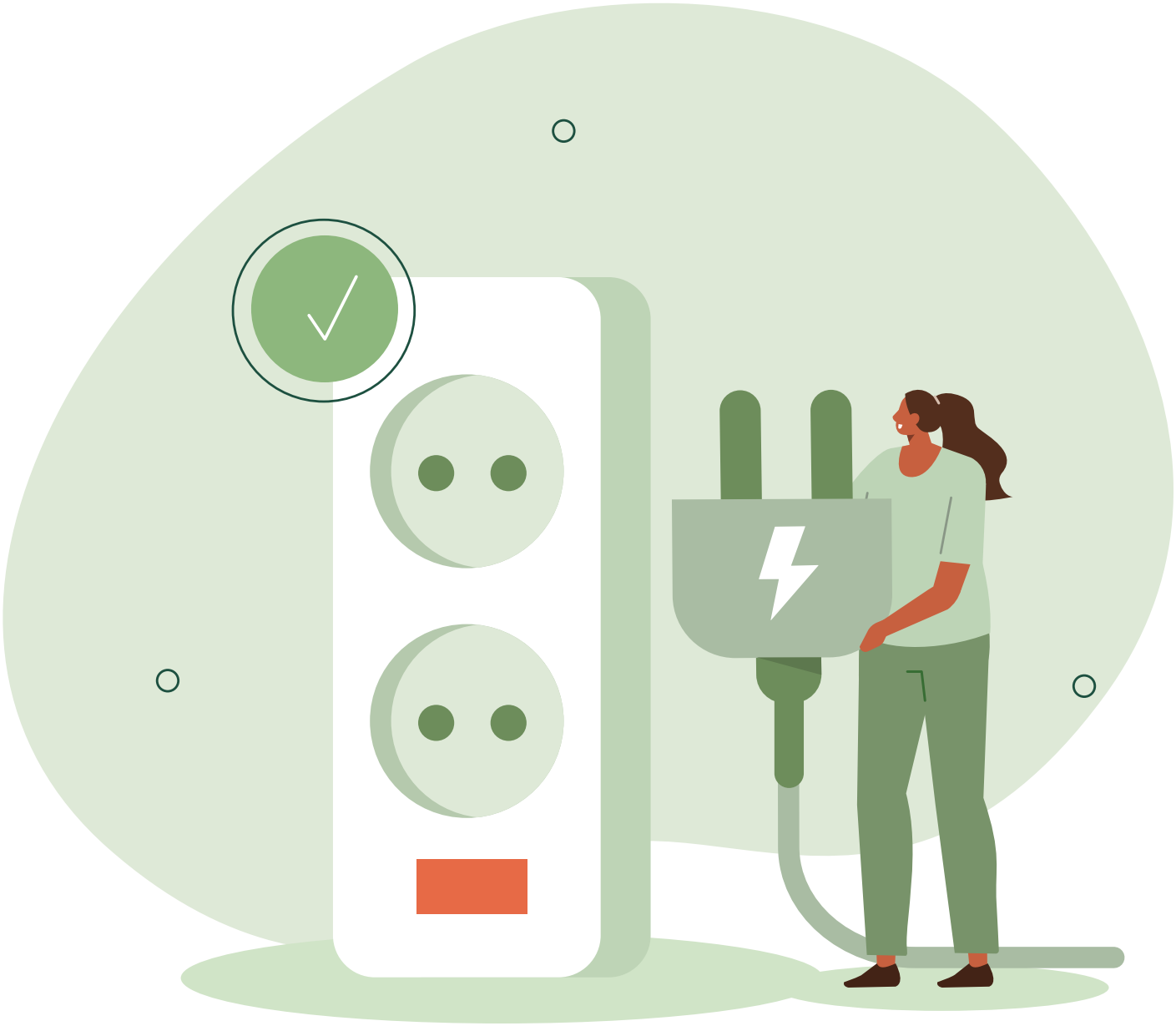
Frank Pientka

Frank.Pientka@gmx.de

Frank Pientka (@fpientka) ist Gründungsmitglied der iSAQB und arbeitet als Cloud Architect. Dabei begleitet er seine Kunden bei ihrer Reise in die Cloud. Er besitzt jahrzehntelange Erfahrung in der Modernisierung von Java-Anwendungen, und hat mehrere AWS-Zertifizierungen.

Green Software – Ressourcen effizient nutzen

Michael Krämer, INNOQ Schweiz



Um den Fußabdruck einer Softwarelösung zu reduzieren beziehungsweise direkt möglichst gering zu gestalten, gibt es drei Ebenen, auf denen man ansetzen kann. Die Infrastrukturebene, um die Laufzeitumgebung so effizient wie möglich zu gestalten. Die konkrete Implementierung der Software, um das Programm selbst effizient zu gestalten. Zuletzt noch die Fachlichkeit, um zum Beispiel Abläufe so zu gestalten, dass sie effizient umgesetzt werden können, oder bei einzelnen Funktionen Kosten und Nutzen zu hinterfragen. Man sollte sich nicht ausschließlich auf eine dieser Ebenen verlassen, aber wir wollen uns nun auf die Implementierung der Software konzentrieren, um konkret zu untersuchen, welchen Möglichkeiten wir als Entwickler/innen und Architekt/innen nachgehen können (siehe Abbildung 1).

Software zu implementieren, die zur Laufzeit effizient ausgeführt wird, ist der Beitrag zu grünerer Software, den wir als Softwareentwickelnde leisten können. Es ist unstrittig, dass es mehr und weniger effiziente Lösungsansätze gibt. Doch leider scheinen wir den Blick darauf etwas aus den Augen verloren zu haben. Im Bereich der serverbasierten Software steht seit mehr als zehn Jahren ausreichend Leistung für die allermeisten Anwendungen zur Verfügung. Darüber hinaus hat durch Virtualisierung und die Nutzung von Cloudumgebungen eine sehr starke Entkopplung von der Hardware stattgefunden, Skalierungsmöglichkeiten erfordern keine Hardwarelieferung und sind in kurzer Zeit umsetzbar. Ich sehe es als Konsequenz unserer gewohnten Umgebung, dass wir uns angewöhnt haben, Qualitätsmerkmale wie Ressourceneffizienz weniger stark zu gewichten und dafür Wartbarkeit und Modularität zu bevorzugen.

Effiziente Software zu schreiben ist kein neuer Job

In anderen Bereichen wie Embedded Computing war und ist Effizienz seit jeher weiter oben auf der Prioritätenliste. In der App-Entwicklung für Mobilgeräte ist ein konkreteres Thema die Batterielaufzeit und daraus resultierend der Energieverbrauch während der Nutzung. Unter der Annahme, dass Ressourcenverbrauch künftig teurer wird oder die Verbraucher vermehrt effiziente Produkte nachfragen werden, wird auch serverbasierte Software vermehrt auf Effizienz achten müssen. Wichtig ist an dieser Stelle, zwischen schneller und effizienter Software zu unterscheiden. In einigen Fällen ergänzen sich diese Merkmale, aber Software kann auch durch übermäßigen Ressourceneinsatz schnell werden. Heute ist es allerdings immer noch so, dass sich die Investition in Optimierungen rein wirtschaftlich nur bei hoher Auslastung eines Systems lohnt, denn Entwicklung ist im Vergleich zu Hardware immer noch teuer. Ich habe mir jedenfalls vorgenommen, in allen Projekten die Ressourceneffizienz als Qualitätsziel in die Architektur einzubringen, sodass zumindest aktiv über ihre Priorisierung entschieden wird. Dies halte ich für einen wichtigen Aspekt, um Aufmerksamkeit auf das Thema zu lenken.

Den Blindflug vermeiden

Aber was genau sollte man nun optimieren? Eine Entwicklerweisheit lautet, vorschnelle Optimierungen zu vermeiden. Wenn wir unsere Software auf geringen Ressourcenverbrauch trimmen wollen, müssen wir zuerst einen Weg finden, die Stellen zu identifizieren, an denen es sich lohnt zu optimieren. Auf dem gleichen Weg sollte es später möglich sein, die Wirkung der Optimierungen zu überprüfen.

Eine Problematik dabei ist allerdings, dass selbst die Begriffe im Bereich nachhaltiger Softwareentwicklung nicht klar definiert sind. Noch schwieriger ist es, eine konkrete Metrik zu finden und diese zu berechnen. Globale und absolute Metriken, die eine Aussage darüber treffen, wie viele Ressourcen eine Software verbraucht, sind derzeit noch sehr schwierig, weil beispielsweise nicht klar abgegrenzt ist, welche Effekte in die Messung einbezogen werden sollten und welche nicht. Beispiele für solche unklaren Abgrenzungen

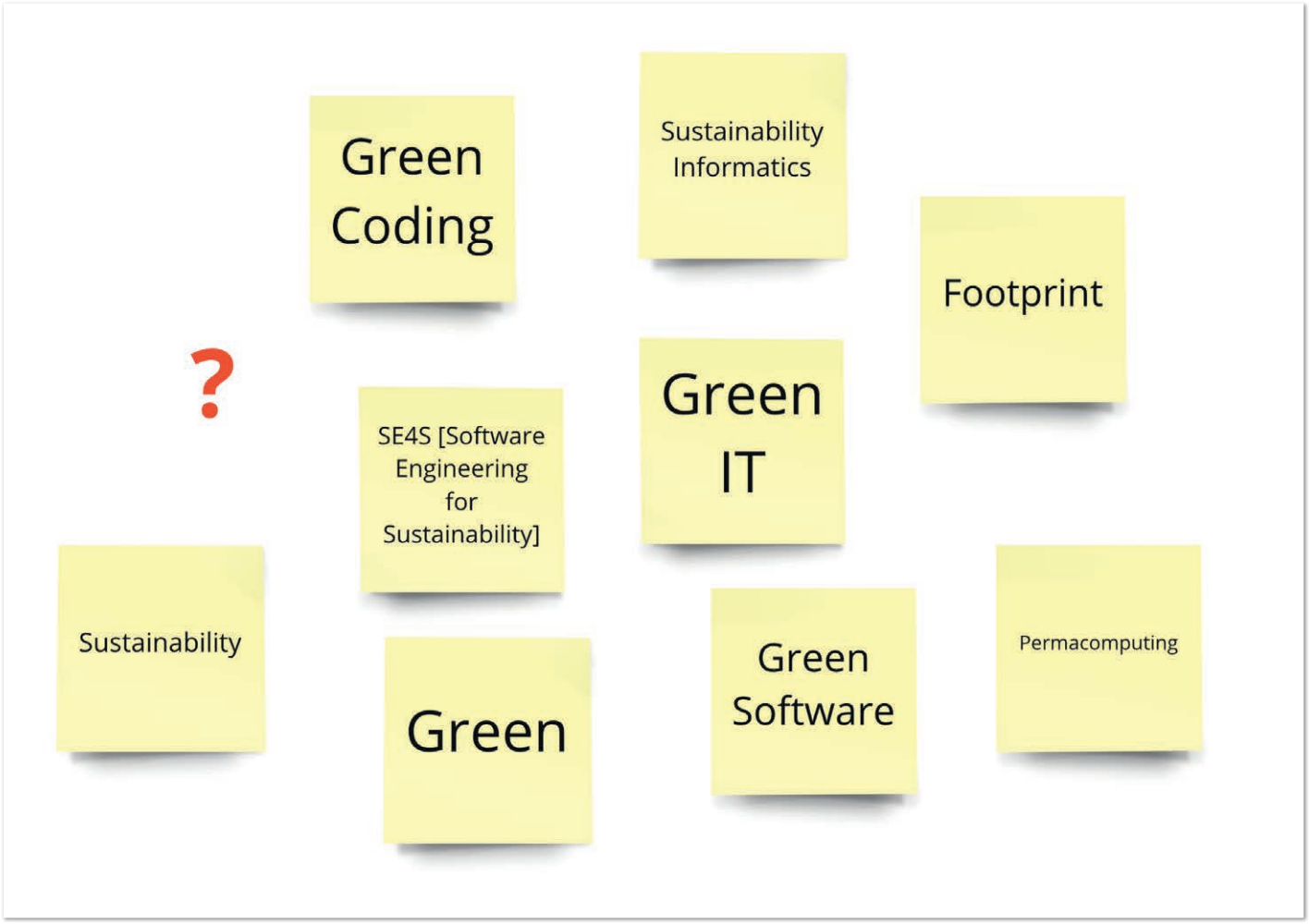


Abbildung 1

sind die Produktion und Lieferkette der Hardware oder die Frage, ob der Datenverkehr zum Endgerät mit einbezogen wird. Vergleichende Messungen sind – ähnlich wie bei anderen Metriken zur quantitativen Beurteilung von Software – deutlich einfacher zu handhaben und reichen aus, um Optimierungen zu beurteilen.

Dabei sollte man versuchen, die einfachste sinnvolle Einheit zu finden. Diese muss kein CO2-Äquivalent sein. Innerhalb eines Systems und einer Produktgruppe kann man beispielsweise genutzte Knoten zählen und vergleichen *[1]*. Auch Kosten lassen sich in der Cloud innerhalb einer solchen Kategorie vergleichen. Komplizierter wird es, wenn verschiedene Produktkategorien zum Einsatz kommen und Managed Services wie Datenbanken genutzt werden. Die großen Cloudanbieter verfügen zwar inzwischen alle über Dashboards, die den CO2-Verbrauch ausweisen, aber diese haben ihre Tücken. Sie erfordern Berechtigungen zum Zugriff auf den Billing-Account, was die Benutzer aus den technischen Abteilungen häufig nicht zur Verfügung haben. Manche Produkte scheinen in den Dashboards gar nicht aufzutauchen, bei einem größeren Projekt sehen wir beispielsweise die Nutzung des Elastic Container Service nicht im AWS Carbon Footprint Tool. Außerdem sind die Statistiken erst nach bis zu drei Monaten Verzögerung abrufbar, deutlich zu lang für eine Feedbackschleife. Zumal nicht zu erwarten ist, dass die genutzte Software über einen solchen Zeitraum frei von Änderungen bleibt.

Eine mögliche Alternative bietet das Cloud-Jewels-Modell von Etsy *[2]*, das es durch Heuristiken unter anderem auch ermöglicht, verschiedene Arten von Cloud-Diensten sowie unterschiedliche Anbieter – also sozusagen Äpfel mit Birnen – zu vergleichen. Dieses Modell bildet auch die Grundlage für die Software Cloud Carbon Footprint *[3]*. Generell kann man sagen, dass solche globalen Messungen, die den Verbrauch einer gesamten Komponente oder Plattform abschätzen, in Cloud-Umgebungen umsetzbar sind. Durch die Betrachtungszeiträume ist allerdings gegebenenfalls nur der Langzeittrend aussagekräftig. In privaten Umgebungen abseits der öffentlichen Cloud-Anbieter kann man solche Messungen tatsächlich mit einem Energiemessgerät durchführen und dort den Betrachtungszeitraum selbst wählen. Effekte wie schwankende Lasten oder über die Zeit steigende Kundenzahlen muss man aus diesen Daten in jedem Fall herausrechnen. Der Artikel zu den Etsy Cloud Jewels gibt auch dazu Beispiele.

Ab in die Whitebox

Möchte man spezifischer messen, gibt es auch dafür mehrere Möglichkeiten. Powertop *[4]* liefert den geschätzten Energieverbrauch pro Prozess und ist auch in Cloud-Umgebungen nutzbar. Da es eigentlich entworfen wurde, um den Batterieverbrauch in Laptops zu messen, ist die Genauigkeit der Werte im Netzbetrieb scheinbar nicht ganz gesichert. Sofern man mit Java arbeitet, liefert Powertop nur Aussagen pro JVM-Instanz. Im Java-Umfeld gibt es das Tool JoularJX *[5]*, einen Java-Agent, der ausgibt, welche Methoden einer Software wie viel Energie konsumieren. Dies ist ein Alleinstellungsmerkmal und macht zuerst große Freude beim Erkunden des Verhaltens der eigenen Software zur Laufzeit. JoularJx funktioniert über RAPL *[7]* auf Linux, eine Schnittstelle, um den Energieverbrauch über die Hardware möglichst genau auszulesen. Es gibt auch eine Version für Windows, aber keine für Mac. Leider ist diese Schnittstelle nur in Bare-Metal-Umgebungen nutzbar. Virtualisierte oder Containerumgebungen sind damit nicht kompatibel. Eine weitere Schwierigkeit in

der Praxis ist, dass JoularJX im Sekundentakt Dateien schreibt und dadurch schnell viele Dateien entstehen. Unter der Annahme, dass nur sehr wenige Produktionsumgebungen direkt auf Bare-Metal laufen, muss man zur Untersuchung seiner Software ein separates System aufstellen. Dabei ist zu beachten, dass es hinsichtlich des Verhaltens mit dem produktiven System vergleichbar sein muss.

Auf der Suche nach anderen, praxistauglicheren Ansätzen erschien die Nutzung eines JVM-Profilers plausibel. Tatsächlich lässt sich damit sehr gut herausfinden, welche Komponenten und Methoden einer Software die meiste CPU-Zeit benötigen. Dies sind die Kandidaten für Optimierungen, wie es auch schon abseits von Green Software war. Eine Alternative zu Profilern sind Metrik-Tools wie Micrometer *[7]* oder Dropwizard Metrics *[8]*, die das Gleiche leisten können und sich hervorragend dauerhaft in Systemen betreiben lassen. In jedem Fall sollte man sicherstellen, die relevanten Teile eines Systems zu finden, bei denen Optimierungen am Ende einen sichtbaren Effekt erzielen können.

Hat man eine Komponente oder eine Methode zur Optimierung identifiziert, gibt es übrigens sehr viele Papers und Untersuchungen zum Energieverbrauch von Codeabschnitten oder Klassen, etwa für die Java Collections *[9]*. In der Praxis wird dies aber vermutlich eher von untergeordneter Bedeutung sein.

Rezepte

Konkrete Potenziale, die in vielen aktuellen Systemen zu finden sind, liegen beispielsweise darin, die Autoscalefunktion eines Clustermanagers auch zu verwenden, um herunterzuskalieren. Aus energetischer Sicht optimal wäre es, Systeme, die nicht genutzt werden, komplett abzuschalten und erst bei Bedarf wieder hochzufahren. Cloud Functions beziehungsweise Lambdas bieten diese Möglichkeiten, haben aber bisher in vielen Projekten keine große Bedeutung. Dies könnte an den Einschränkungen liegen, die sie initial hatten. Die Möglichkeiten haben sich in den letzten Jahren deutlich erweitert. Vor allem KNative bietet die Möglichkeit, eine solche Technologie mit mehr Kontrolle und Freiheiten in einem eigenen Cluster einzusetzen. Sie sollten diese Option in jedem Fall untersuchen, wenn ihre Lösung Komponenten enthält, deren Last nicht gleichmäßig anfällt.

Ein weiterer Aspekt, der beleuchtet werden kann, ist Continuous Integration. Lange laufende Tests sind nicht nur ein potenzielles Problem für Green Software, sie sind auch oft ein Hindernis für Entwicklerproduktivität und -zufriedenheit. Die Teststrategie zu überarbeiten, die Software einfach testbar zu gestalten und dann entsprechend simple, aber aussagekräftige Tests unter Berücksichtigung der Testpyramide zu verwenden, ist das Ziel. Dies ist ein Beispiel dafür, wie aus dem Bestreben nach Green Software Synergien entstehen können, die weitere positive Auswirkungen an anderen Stellen haben. Eine weitere Möglichkeit ist, nicht mehr alle Tests bei jedem Commit laufen zu lassen. Ein Indikator, um Continuous Integration zu beleuchten, ist, wenn CI-Server und Entwicklungswerkzeuge viele Ressourcen benötigen.

Kritisches Hinterfragen von Datenflüssen ist ein weiteres solches Rezept. Insbesondere generische oder generierte Datenstrukturen tendieren dazu, große Mengen an Daten zu enthalten, die dann in mehreren Komponenten jeweils nur zu einem Bruchteil genutzt werden. Auch die Sortierung oder Filterung von Daten sollte nur einmalig stattfinden und dann entsprechend weitergegeben werden. Selbstverständlich sollte man die Funktionen der Datenbank ausnutzen, wo dies möglich ist. Dies ist besonders zu beachten, wenn ORM-Frameworks genutzt werden und Tabellen auf Objektstrukturen 1:1 abgebildet werden.

Zwei Anmerkungen seien noch gestattet. Um konkret Veränderungen beurteilen zu können, ist es wichtig, einige der möglichen Variablen zu fixieren. Daher wurde vereinfachend angenommen, dass durch die Nutzung von weniger Computerressourcen auch weniger Belastung entsteht. So lassen sich zwar keine Emissionen ableiten, aber Vergleiche durchführen. Zweitens, die Ebene der Fachlichkeit bietet in manchen Situationen einen sehr großen Hebel. Insbesondere im Rahmen eines agilen Entwicklungsprozesses ist es oft möglich, die Erkenntnisse aus der Implementierung einfließen zu lassen, um weitere Features optimal gestalten zu können.

Viel Erfolg bei der Umsetzung, jeder Beitrag zählt.

Quellen

[1] AWS re:Invent 2017: Deep Dive on Amazon EC2 Instances, Featuring Performance Optimiz *https://www.youtube.com/watch?v=mZy6E2I5Rek&t=815s*

[2] Etsy Cloud Jewels *https://www.etsy.com/codeascraft/cloud-jewels-estimating-kwh-in-the-cloud*

[3] Cloud Carbon Footprint *https://www.cloudcarbonfootprint.org/docs/getting-started*

[4] Using Powertop to Lower System Power Usage *https://cloud-cow.com/content/using-powertop-to-lower-system-power-usage/*

[5] JoularJx *https://www.noureddine.org/research/joular/joularjx*

[6] Energy measurements in Linux *https://blog.chih.me/read-cpu-power-with-RAPL.html*

[7] Micrometer Application Monitoring *https://micrometer.io/*

[8] Metrics - Dropwizard *https://metrics.dropwizard.io/4.2.0/*

[9] Energy Profiles of Java Collections *https://dl.acm.org/doi/10.1145/2884781.2884869*

Michael Krämer
INNOQ Schweiz
michael.kraemer@innoq.com

Michael Krämer entwickelt seit über 15 Jahren Software und arbeitet als Softwarearchitekt bei INNOQ. Er setzt sich in seinen Projekten sehr dafür ein, Komponenten mit klaren Verantwortlichkeiten zu erarbeiten und technisch angemessene Lösungen für fachliche Anforderungen zu finden. Außerdem beschäftigt er sich mit Machine Learning sowie der Integration von ML-Modellen in produktionstaugliche Softwareumgebungen und gibt Trainings für Softwarearchitektur.

JavaLand

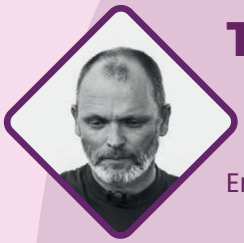
21. – 23. MÄRZ 2023

im Phantasialand bei Köln

Die Konferenz der Java-Community

Mehr Infos zum Event findet ihr hier:

www.javaland.eu



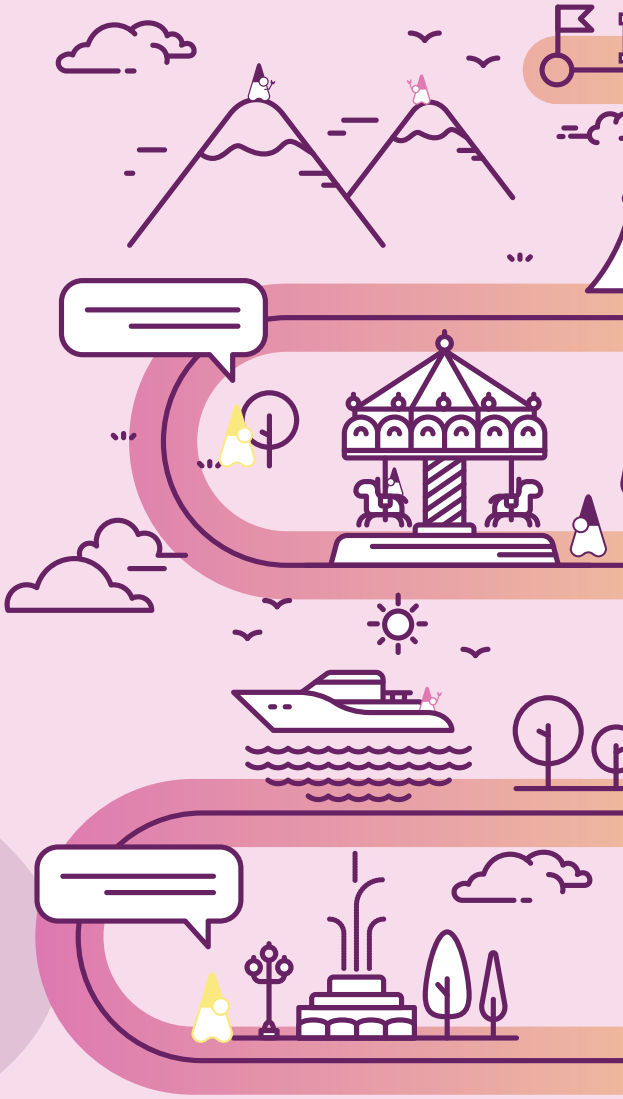
The Zen of Programming

In seiner Keynote schildert Sander Hoogendoorn seine persönliche Reise durch Plattformen, Sprachen, Prinzipien, Zweifel und Kämpfe, die vielen Entwickler:innen im Laufe ihrer Karriere begegnen.

Community-Keynote

Sexismus, Mobbing und Bossing, unkontrollierter Narzissmus und fehlende Eskalationswege im Job haben in den letzten Jahren auch vor der Java-Community keinen Halt gemacht. Wir lüften den dunklen "Tabu-Vorhang".

Teilt eure Story mit uns: kummerkasten@ijug.eu



- 2.000+ JAVA-FANS
- 160+ SESSIONS
- 50+ AUSSTELLER
- 100 % AUSTAUSCH
- 54 JAVA USERGROUPS
- 100 % SPASS

Präsentiert von: iJUG Verbund, Heise Medien, DOAG

Veranstalter: JavaLand

Ressourceneffizienz in der Cloud-Native-Softwareentwicklung für Energieeffizienz im Betrieb

Richard Vobl und Denis Angeletta, RETIT GmbH, Nadja Hagen und Uwe Eisele, envite consulting GmbH

Der Energiebedarf von Cloud-Rechenzentren wächst kontinuierlich und es ist durch die zunehmende Digitalisierung davon auszugehen, dass er weiter steigen wird. Je höher der Energiebedarf ist, desto schwieriger ist es, diesen aus regenerativen Energien zu decken. Als Endanwender hat man jedoch auf die Gestaltung von Cloud-Rechenzentren keinen direkten Einfluss, sondern nur indirekt über den Ressourcenverbrauch und die Betriebskonzepte der eigenen Software. Was kann man also tun? Dieser Artikel gibt einen Überblick über mögliche Maßnahmen zur Effizienzsteigerung in Entwicklung und Betrieb von Cloud-Native-Softwaresystemen.

Warum ist das wichtig?

Die fortschreitende Digitalisierung bedeutet, dass der Ressourcenbedarf für die IT permanent steigt. Zum einen betrifft das die Ressourcen für die Hardware selbst und zum anderen geht es um den Strombedarf für den Betrieb der Rechenzentren. Aktuell werden 16 TWh und somit 3 % des Gesamtstrombedarfs in Deutschland allein durch Rechenzentren verbraucht *[1, 2]*. Der Bitkom schätzt, dass diese Zahl bis 2030 auf 28 TWh steigen kann, was einen signifikanten Zuwachs von 75 % in acht Jahren bedeutet *(siehe Abbildung 1) [1]*.

(Energy Efficiency Directive, EED) vorgelegt *[3]*. Darin enthalten ist das „Energy Efficiency First"-Prinzip, das vorsieht, großen Wert auf Energiesparmaßnahmen zu legen. Durch die gestiegene Bedeutung der IT werden auch hier Einsparmaßnahmen entstehen müssen.

In der Vergangenheit waren Energieeffizienzmaßnahmen in der IT häufig auf die Hardware beschränkt, beispielsweise konnten Unternehmen im eigenen Rechenzentrum ineffiziente Geräte austauschen. Im Zeitalter des Cloud-Computing sind solche Maßnahmen

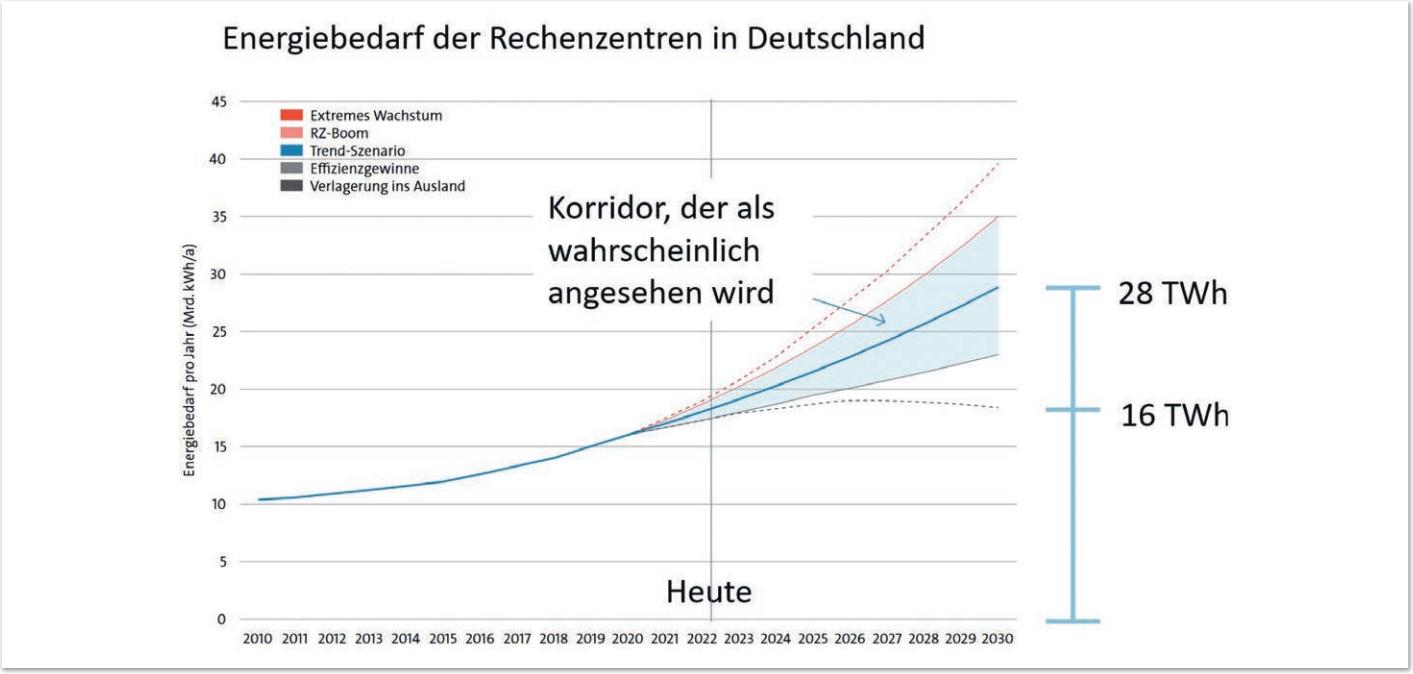


Abbildung 1: Rechenzentren in Deutschland 2022 (Quelle: [1])

Die Klimakrise und die akute Energiekrise zwingen die Politik zum Handeln. Mit dem klimapolitischen Großprojekt „Fit for 55" (Lit-Ref) vom Europäischen Rat wurde eine neue Energieeffizienzrichtlinie

nicht mehr möglich. Hier müssen wir uns auf die Software konzentrieren, da sie den eigentlichen Ressourcenverbrauch auf der Hardware erzeugt und der Betrieb von Software der Grund für die

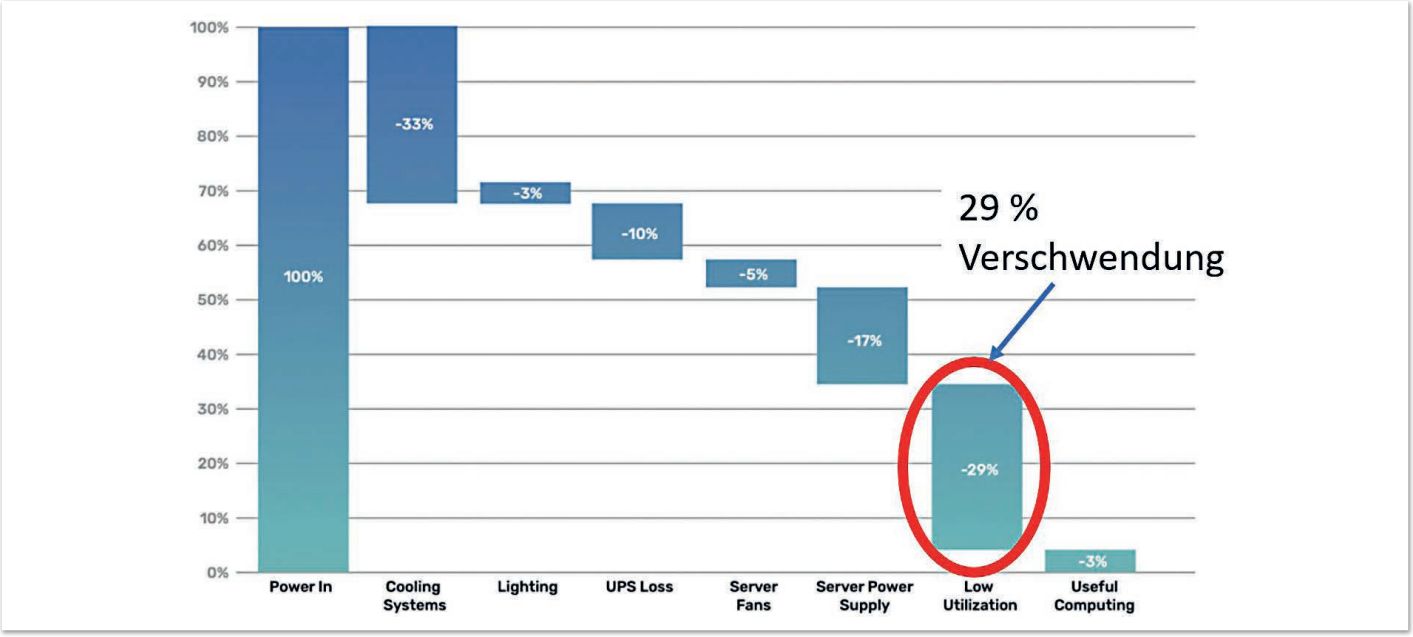


Abbildung 2: Verbrauch und Aufteilung des Energiebedarfs in einem Rechenzentrum (Quelle: [4])

Provisionierung von Rechenzentren und somit der Ursprung des Energieverbrauchs ist. Somit sind die primären Einflussmöglichkeiten auf den Ressourcenbedarf im Cloud-Native-Umfeld die Effizienz der eigenen Software und ein Betrieb, der diese Ressourceneffizienz bestmöglich ausnutzt.

Über das Einsparpotenzial durch Optimierung der entwickelten Software lässt sich nur schwer eine pauschale Aussage treffen, da dies sehr vom aktuellen Zustand der Software abhängt. Nach Erfahrungswerten durch unsere Projektarbeit ist dabei eine Reduzierung von 20 % bis 80 % der Ressourcenverbräuche möglich.

Welche Signifikanz die Steigerung der Auslastung hat, zeigt *Abbildung 2*. Während nur 3 % der eingesetzten Energie wirkliche Rechenleistung erbringen und für andere Aspekte wie Kühlung notgedrungen der hohe Beitrag von 30 % der Energie aufgewendet werden muss, ließen sich fast 30 % des Energieverbrauchs durch eine höhere Auslastung vermeiden.

Unser Ziel sollte daher die Entwicklung einer ressourceneffizienten Software sein, die die bereitgestellte Hardware und die Dienste des Cloud-Providers bestmöglich nutzt.

Was kann man während der Entwicklung tun?

Um die Ressourceneffizienz von Software beurteilen und verbessern zu können, werden Metriken benötigt, die diese messbar und quantifizierbar machen. Hierfür eigenen sich Ressourcenverbrauchsmetriken wie CPU-Rechenzeit, die Datenmenge, die zwischen Festplatte/Netzwerk transferiert wird, sowie der Speicherverbrauch *[5, 6]*. Wichtig ist hierbei, dass diese Metriken nicht mehr wie heutzutage meist als Auslastungswerte relativ für einen konkreten Server (etwa 20 % des verfügbaren Speichers wurden verbraucht) angegeben werden, sondern konkret für einzelne Operationen und Transaktionen einer Software. Ein Beispiel könnte sein, dass eine Operation 2 ms CPU-Zeit verbraucht, 20 MB Hauptspeicher allokiert und 50 Mbyte auf die Festplatte und 2 Mbyte ins Netzwerk schreibt. Dies ist insbesondere wichtig, um eine Vergleichbarkeit der Daten zu gewährleisten.

Zusätzlich zu den Ressourcenverbräuchen können weitere Kriterien zur Bestimmung der Effizienz verwendet werden. Dazu zählt insbesondere die Anzahl der Aufrufe anderer Services beziehungsweise Datenbankabfragen. In einer Microservice-Architektur werden Transaktionen typischerweise durch ein Zusammenspiel verschiedener Services verarbeitet. Wenn ein Service beispielsweise hinsichtlich des eigenen Speicherverbrauches optimiert ist, dafür zur weiteren Verarbeitung aber mehrere Aufrufe eines anderen Service benötigt, ist der Gesamtverbrauch unter Umständen höher als der Aufruf eines weniger optimierten Service. *Abbildung 3* zeigt diese Problematik anhand von zwei Varianten (A und B) der Realisierung einer Business-Transaktion. Es sieht zwar auf den ersten Blick so aus, als wenn Methode A ineffizienter ist als Methode B, doch insgesamt verbraucht die gesamte Transaktion in Variante A 50 Mbyte Speicher (30 Mbyte für Methode A + 20 Mbyte für Methode X) und somit 10 Mbyte weniger als die Variante B, die mit 60 Mbyte Speicher (20 Mbyte für Methode B + 20 Mbyte jeweils für die Methoden X und Y) zu Buche schlägt.

Mit den Metriken zur Beurteilung der Ressourceneffizienz können nun Anforderungen an ein Softwaresystem formuliert werden. Dadurch wird die Ressourceneffizienz Teil der Qualitätsmerkmale des Systems. Konkret wird also formuliert, wie viel Verbrauch die Verarbeitung einer bestimmten Transaktion des Systems verursachen darf. Ein Beispiel könnte also sein, dass keine Transaktion mehr als 50 MB Speicher verbrauchen darf und nicht mehr als 20 Datenbankabfragen verursachen soll. Auch die Betrachtung der Abfolge einzelner Transaktionen, die einen bestimmten Anwendungsfall realisieren, ist denkbar. Die Anforderung würde dann auf die Summe der verursachten Verbräuche zur Abarbeitung aller Transaktionen im Rahmen des Anwendungsfalles abzielen, etwa Anwendungsfall X darf in Summe nicht mehr als 1 GB Speicher verbrauchen. Dabei ist anzumerken, dass es teilweise schwierig ist, für alle Metriken sinnvolle Werte zu bestimmen. Unsere Erfahrung hat gezeigt, dass sich vor allem der Speicherverbrauch und die Anzahl externer Aufrufe gegen andere Services oder die Datenbank eignen, um mit in die Anforderungen einzufließen. Für den CPU-Verbrauch und die Verbräuche von Festplatte und Netzwerk ist dies vor allem durch die Abhängigkeit von der verwendeten Hardware nicht immer möglich.

Wenn die Anforderungen bezüglich der Ressourceneffizienz an das Softwaresystem feststehen, müssen diese in den Entwicklungsprozess mit einfließen. Damit soll sichergestellt werden, dass am Ende eines Entwicklungsprozesses das Softwaresystem weiterhin alle festgelegten Qualitätsmerkmale einschließlich der Ressourceneffizienz erfüllt. Dazu dienen sogenannte Quality Gates.

Eine kontinuierliche Überprüfung der Einhaltung der Quality Gates wird häufig mittels einer Continuous-Integation(CI)- oder Conti-

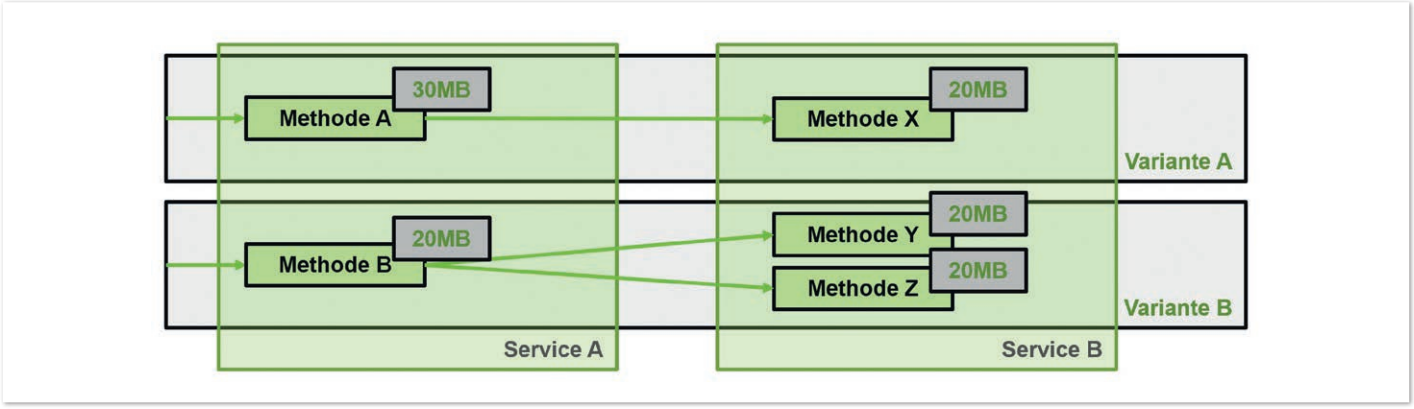


Abbildung 3: Speicherverbräuche in einer Microservice Architektur (© RETIT GmbH)

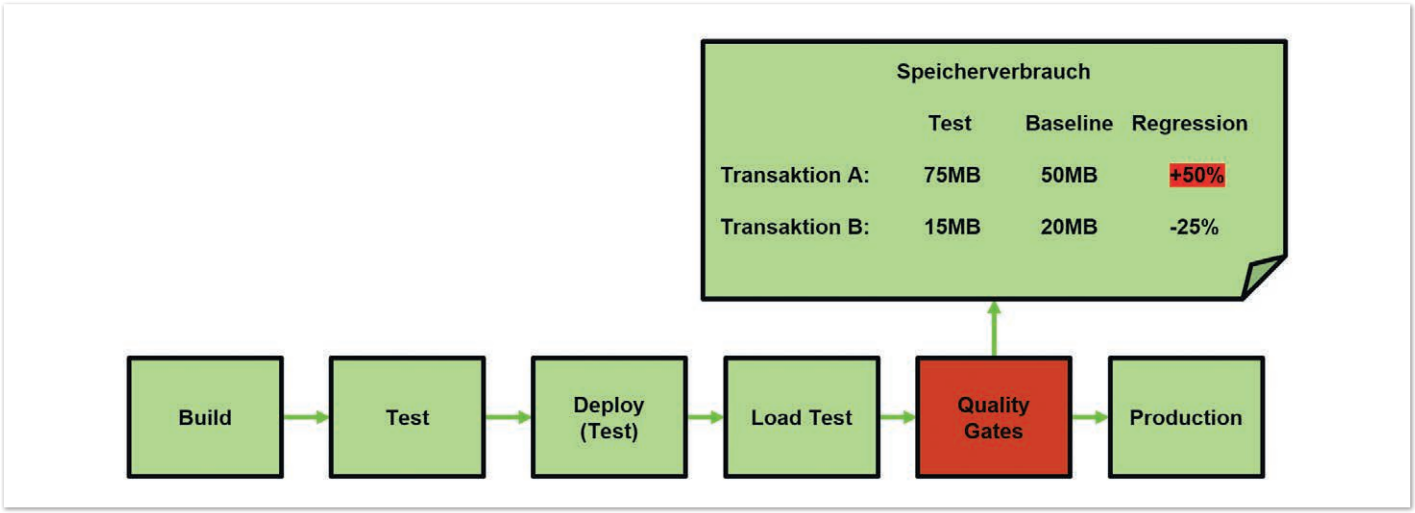


Abbildung 4: CI/CD-Pipeline zur Einhaltung der Quality Gates (© RETIT GmbH)

nuous-Delivery(CD)-Pipeline realisiert. *Abbildung 4* zeigt die vereinfachte Darstellung einer solchen Pipeline. Innerhalb des Quality Gate sind Anforderungen an den Ressourcenverbrauch definiert, die mit jeder Neuerung (Feature, Bug Fix, ...) der Software überprüft werden, in *Abbildung 4* sind dies Speicherverbrauchsanforderungen pro Transaktion. Um dies zu ermöglichen, wird jede Softwareversion, die auf Produktion deployt werden soll, einem Lasttest unterzogen. Während des Lasttests werden die Ressourcenverbräuche der einzelnen Transaktionen gemessen und anschließend mit den hinterlegten Anforderungen verglichen. Falls die Anforderungen nicht erfüllt werden, wird die Pipeline als fehlgeschlagen markiert. Die entsprechende Softwareversion kann somit nicht auf die Produktionsumgebung ausgerollt werden. Auch mit Einführung solcher Quality Gates sollte die Pipeline weiterhin vollautomatisiert ausgeführt werden. Hilfreich ist dabei zum Beispiel das RETIT Continuous Delivery Jenkins Plug-in *[7]*, das in der Lage ist, den Lasttest zu steuern und die währenddessen erhobenen Ressourcenverbrauchsmetriken gegen hinterlegte Grenzwerte zu testen.

Es gibt bereits eine Vielzahl von Messverfahren, um Ressourcenverbrauchsmetriken zu erheben. Profiling Tools oder Heap-Dumps sind gängige Mittel, um dies zu bewerkstelligen. Die Erhebung findet in beiden Fällen jedoch auf Ebene von Prozessen und Threads statt und beide Ansätze bieten keinen Kontext bezüglich der aufgerufenen Transaktionen, die sich potenziell über mehrere Prozesse oder Threads erstrecken können. Damit Entwickler Effizienzprobleme einer Anwendung analysieren und anschließend den hohen Ressourcenverbrauch einer Software effektiv bekämpfen können, ist allerdings die Erhebung der Metriken auf Ebene von Transaktionen notwendig. Transaktionen geben uns einen Überblick darüber, was in einer Anwendung während einer Anfrage passiert.

Zur Erhebung von Transaktionsabläufen werden meist Observability- oder Application Performance Monitoring (APM) Tools verwendet. In den meisten dieser Werkzeuge liegt der Fokus jedoch auf den Antwortzeiten der Transaktionen und der Ressourcenauslastung des Gesamtsystems (zum Beispiel CPU Utilization). Die Erhebung der Ressourcenverbräuche auf Ebene der Transaktionen ist noch nicht flächendeckend möglich. Eine Möglichkeit, die Ressourcenverbrauchsmetriken für alle Operationen einer Transaktion zu erheben, bietet die RETIT OpenTelemetry-Agent Extension *[8]* in Kombination mit dem OpenTelemetry Agent für Java *[9]*. Diese

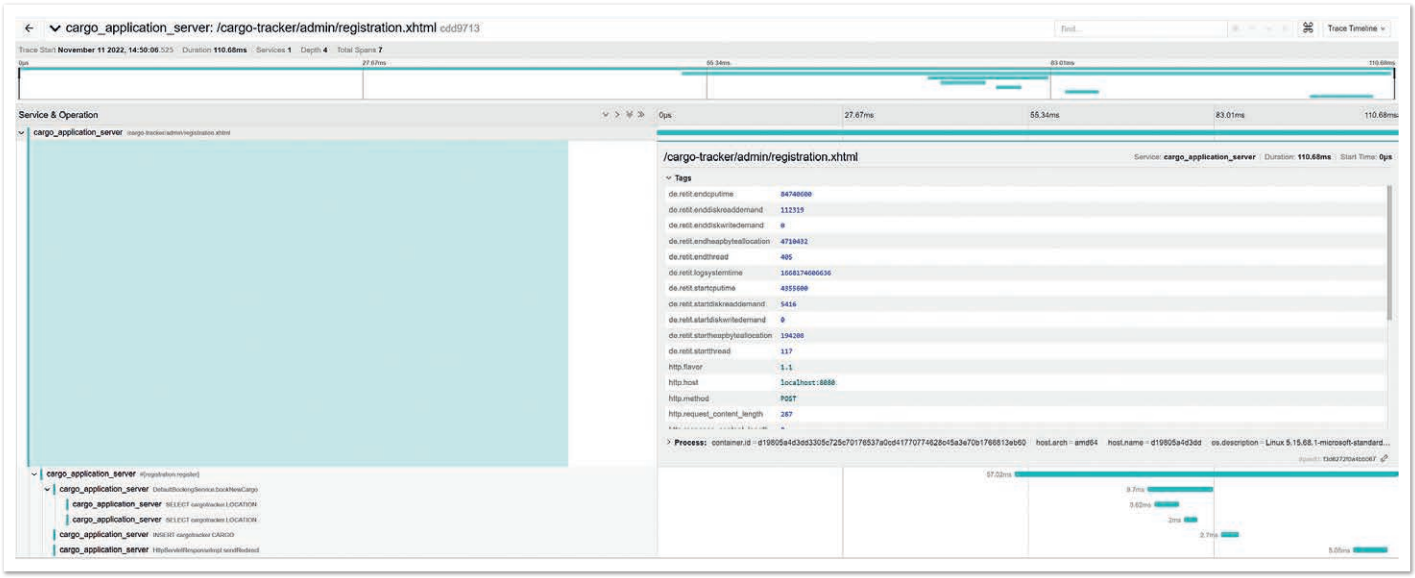


Abbildung 5: Transaktionsansicht in Jaeger [10] inklusive Ressourcenverbräuche (© RETIT GmbH)

Extension erhebt zusätzlich zu den normalen Trace-Informationen für alle Operationen einer Transaktion die bereits genannten Ressourcenverbrauchsmetriken und fügt diese als Kontextattribute zu den OperationTelemetry-Traces hinzu. In *Abbildung 5* wird dies für einen Trace einer Java-Anwendung in der Jaeger-UI *[10]* gezeigt. Man erkennt, dass für die unterschiedlichen Ressourcen (CPU – start-/endcputime, Speicher – start-/endheapbyteallocation und Festplatte – start-/ enddisk(read-/write)demand) jeweils die Verbräuche vor (start) und nach (end) der Ausführung gespeichert werden. Diese Daten bieten die Grundlage für die Quality Gates und ermöglichen es, pro Transaktion zu prüfen, ob die Anforderungen eingehalten werden.

Angenommen eine Pipeline ist aufgrund einer nicht erfüllten Anforderung durch ein Quality Gate fehlgeschlagen. Zwar haben wir durch die Metriken einen Hinweis darauf, welche der definierten Anforderungen nicht erreicht wurde, jedoch sagen uns die Metriken allein nicht, wo das Problem liegt. Hierzu ist es erforderlich, die Daten verschiedener Traces zu analysieren, um die Schwachstellen mit einem hohen Ressourcenverbrauch zu identifizieren. Eine Möglichkeit dazu bietet die RETIT Performance Analytics Suite (PAS) *[11] (siehe Abbildung 6)*, hier werden die Ressourcenverbrauchsdaten auf Ebene der Transaktionen und Operationen zusammenaggregiert und dies ermöglicht es, auf einen Blick zu erkennen, wo am meisten Ressourcen von welchem Typ verbraucht werden.

Durch die Definition von Anforderungen bezüglich der Ressourceneffizienz, ihrer kontinuierlichen Überwachung und die Möglichkeit der Analyse von Schwachstellen ist es möglich, die Ressourceneffizienz der eigenen Software kontinuierlich im Blick zu halten und diese zu optimieren.

Was kann man im Betrieb beachten?

Die im vorigen Abschnitt vorgestellten Maßnahmen ermöglichen uns die Entwicklung von ressourceneffizienter Software. Leider kann jedoch auch eine sehr effiziente Software sehr ineffizient betrieben werden. Daher wollen wir in diesem Abschnitt betrachten, was im Betrieb beachtet werden muss, um die Potenziale der effizient entwickelten Software nutzen zu können.

Ein Cloud-Anbieter übernimmt für uns einen Großteil des Infrastrukturmanagements. Grundsätzlich müssen wir daher die Annahme treffen, dass ein effizienter Betrieb in seinem eigenen wirtschaftlichen Interesse liegt und er diesen umzusetzen weiß. Müssen oder können wir also darauf vertrauen, dass dieser die richtigen Entscheidungen in Bezug auf einen nachhaltigen Betrieb trifft?

Noch bevor eine Softwareanwendung in Betrieb genommen wird, kann die spätere Ressourcen- und Energieeffizienz beeinflusst werden. Cloud-Anbieter besitzen verschiedene Service-Modelle, die sich je nach Managementgrad durch den Cloud-Anbieter unterscheiden. Je weniger durch den Cloud-Anbieter übernommen wird, desto höher ist das Risiko einer ineffizienten Auslastung *(siehe Abbildung 7)*. Das Infrastructure-as-a-Service-Modell (IaaS) bietet eine hohe Flexibilität, damit aber auch viele Gefahrenpunkte für eine suboptimale Auslastung der Ressourcen. Die Herausforderungen und die Optimierungspotenziale, die mit dem Betrieb in der Cloud einhergehen, sind also vor allem vom gewählten Service-Modell abhängig. Sofern möglich sollte daher stets ein Cloud-Service möglichst weit rechtsstehend in *Abbildung 7* gewählt werden.

Das Function-as-a-Service-Modell (FaaS) zeigt, wie maßgeschneiderte Software mit einem energieeffizienten Betrieb kombiniert

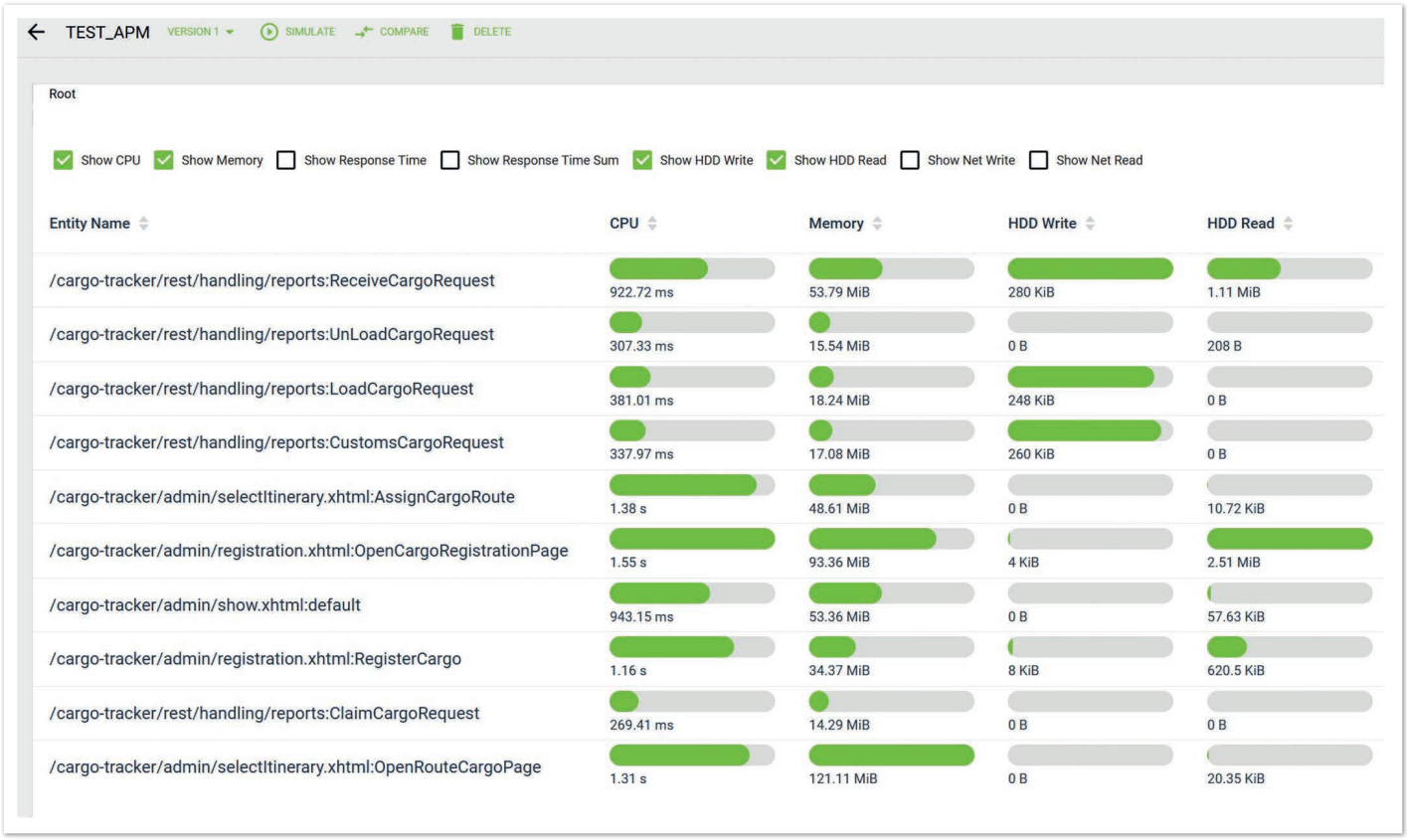


Abbildung 6: Darstellung der Ressourcenverbräuche auf Transaktionsebene RETIT PAS (© RETIT GmbH)

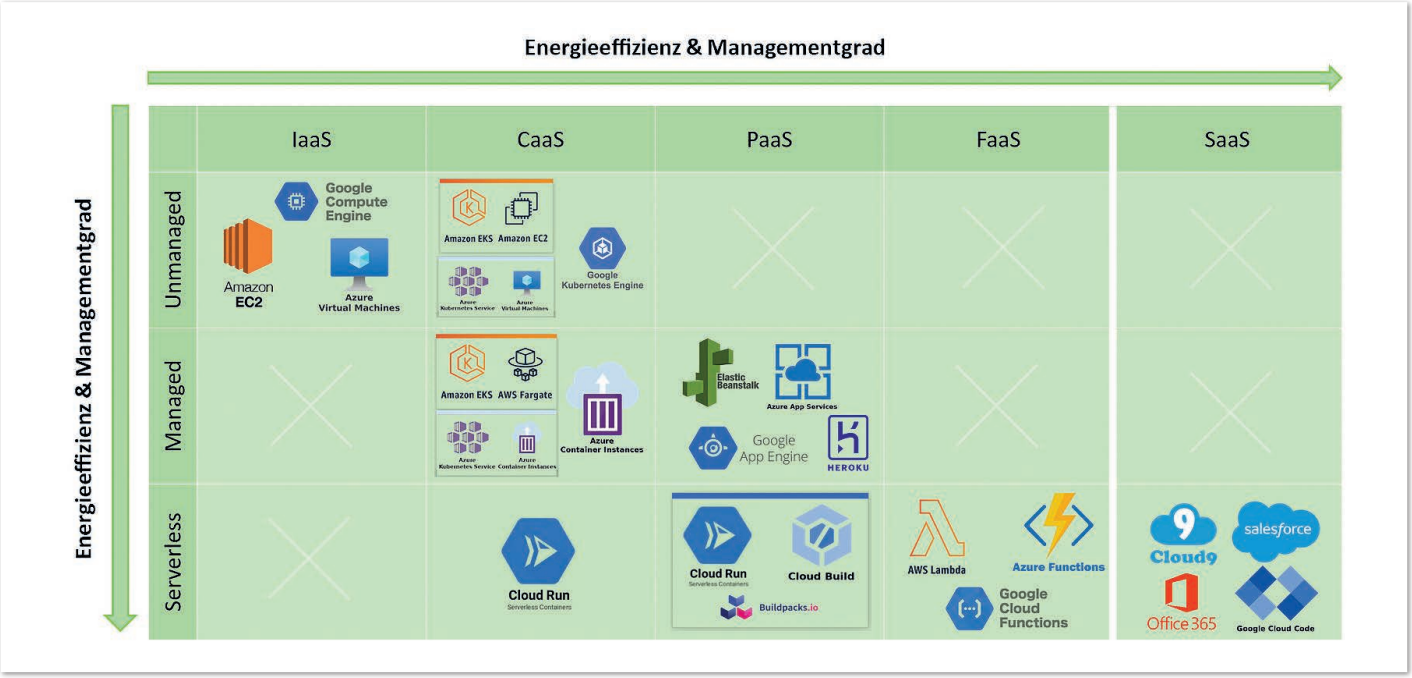


Abbildung 7: Beispiele für Cloud-Services und deren ungefähre Einordnung in Bezug auf Service-Modell, Energieeffizienz und Managementgrad; Service-Modelle: Infrastructure-as-a-Service (IaaS), Container-as-a-Service (CaaS), Plattform-as-a-Service (PaaS), Function-as-a-Service (FaaS), Software-as-a-Service (SaaS); (© envite consulting GmbH)

werden kann. Dabei werden die Anwendungen in Form von leichtgewichtigen Functions ausgeliefert, die üblicherweise nur eine Laufzeit von Sekunden bis Minuten besitzen. Der Cloud-Anbieter übernimmt hier das Management und die Skalierung, sodass Ressourcen passgenau bereitgestellt werden können *(siehe Abbildung 8)*. Sobald eine Function nicht mehr benötigt wird, wird sie auf null herunterskaliert. Die kurzen Laufzeiten und die dynamische Skalierung ermöglichen dem Cloud-Anbieter eine flexible Umverteilung und eine effizientere Ausnutzung von Ressourcen.

Im Gegensatz dazu stehen IaaS und dazwischenliegende Modelle, bei denen das Infrastrukturmanagement mit mehr Verantwortung einhergeht. Ziele, wie das schnelle und dynamische Bereitstellen von Anwendungen, stehen häufig im Konflikt mit einem ressourceneffizienten Betrieb. Beispielsweise können Auto-Scaling-Mechanismen von Instanzen bei langen Startzeiten der Anwendung nicht genutzt werden, da so keine dynamische Reaktion auf Lastspitzen möglich ist. Oft wird daher auf Überprovisionierung zurückgegriffen, die sich an Lastspitzen ausrichtet. In der Folge werden zu viele Ressourcen allokiert, die die meiste Zeit nicht benötigt werden.

Doch inwiefern wirken sich reservierte, jedoch nicht genutzte Ressourcen auf den Energieverbrauch des Cloud-Betriebs aus? Wenn mehr Ressourcen allokiert werden als eigentlich benötigt, ist das in erster Linie ein Kostentreiber. Zudem erschwert Überprovisionierung auch ein effizientes Management durch den Cloud-Anbieter. Reservierte Ressourcen können nicht anderweitig verplant werden, sodass der darunterliegende Server möglicherweise nicht vollständig ausgelastet werden kann. *Abbildung 9* zeigt, dass Server auch im unausgelasteten Zustand Energie verbrauchen *[13]*. Dadurch steigt die Energieeffizienz mit einer höheren Auslastung des Servers. Die Ressourcenauslastung zu optimieren, lohnt sich also nicht nur aus Kostensicht, sondern verringert in der Summe auch den Energieverbrauch. In welchem Bereich die optimale Auslastung eines Servers liegt, lässt sich allerdings nicht pauschal festlegen und hängt letztlich von den Anforderungen an die Latenz, von der Varianz der Last und von den Anwendungen selbst ab.

Eine häufige Ursache von Überprovisionierung ist eine falsche Anforderungskonfiguration. Für viele Workloads müssen die voraussichtlich benötigten Ressourcen im Vorfeld festgelegt werden. Zum

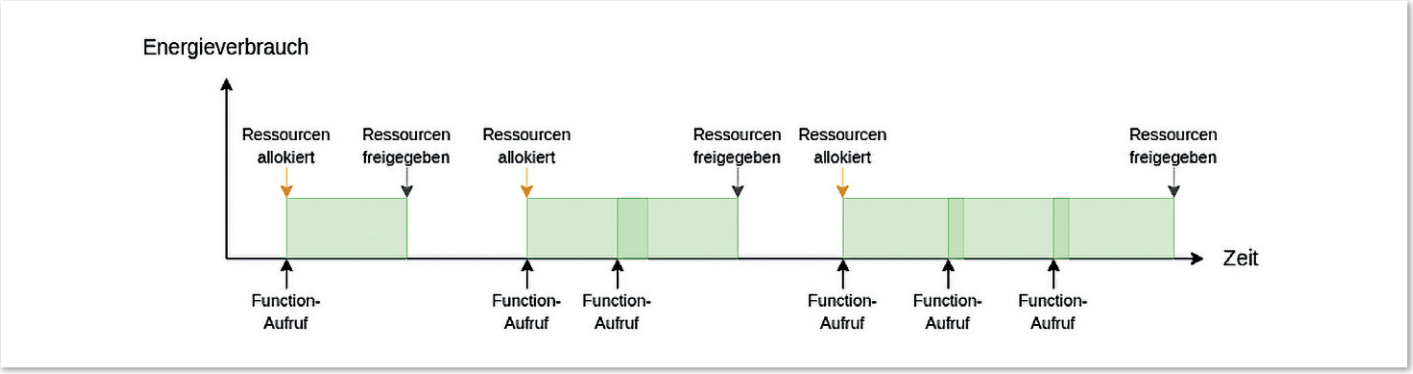


Abbildung 8: Energieverbrauch durch das Allokieren und Freigeben von Ressourcen beim Aufruf von Functions (Quelle: nach [12])

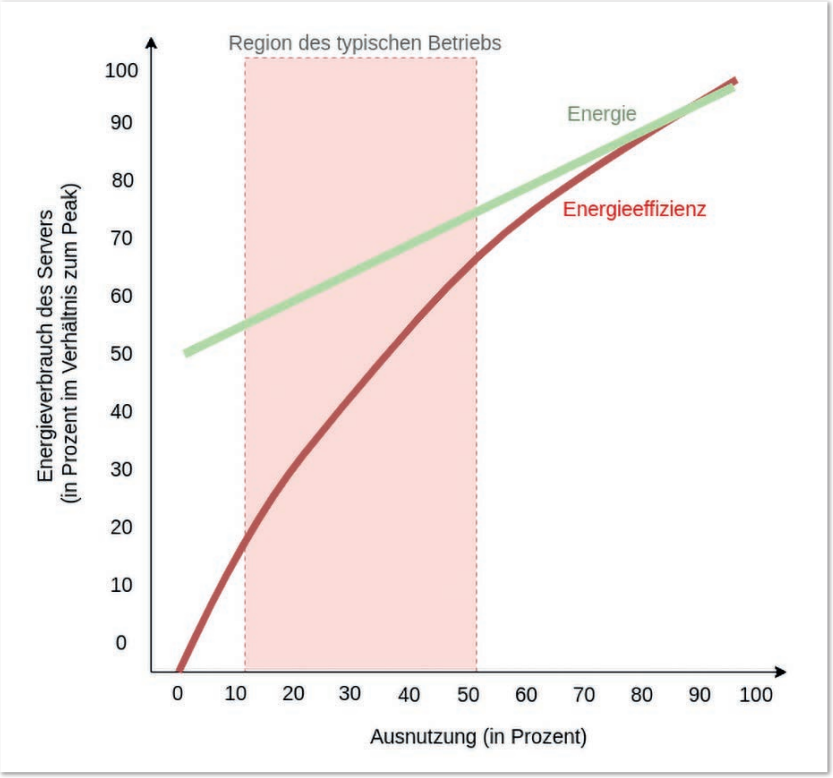


Abbildung 9: Verhältnis von Server-Auslastung und Energieverbrauch und die daraus resultierende Energieeffizienz (Quelle: nach [13])

Beispiel werden bei Kubernetes Pods diese Anforderungen mithilfe von Requests und Limits definiert. Geschätzte Konfigurationen entsprechen hier in den seltensten Fällen dem tatsächlichen Bedarf oder werden absichtlich zu hoch gesetzt, um beispielsweise einen schnellen Start der Pods zu gewährleisten oder Probleme im Betrieb zu vermeiden. Kubernetes-Metriken im Zusammenhang mit einem Grafana-Dashboard können helfen, um Differenzen zwischen Ressourcenanforderung und -nutzung zu überwachen und gegebenenfalls anzupassen. Zur Vereinfachung können Tools wie der Kubernetes Vertical Pod Autoscaler genutzt werden, die eine automatische Anpassung der Reservierung an den Bedarf vornehmen *[14]*.

Eine Anpassung der Ressourcenanforderungen verhindert Überprovisionierung aber noch nicht vollständig. Um darauf verzichten zu können, muss gewährleistet sein, dass Ressourcen bei Bedarf unmittelbar zur Verfügung stehen. Eine mögliche Strategie ist Skalierung „On-Prediction". Dieser Ansatz kann insbesondere dann genutzt werden, wenn Lastspitzen vorhersehbar sind oder zu regelmäßigen Zeitpunkten auftreten. Beispiele sind Unterschiede in der Auslastung je nach Tageszeit oder zyklisch auftretende Jobs wie Backups und Datenanalysen. Aus historischen Daten können diese Lastspitzen abgeleitet und Ressourcen rechtzeitig zur Verfügung gestellt werden. In AWS lässt sich diese Strategie beispielsweise mithilfe einer sogenannten predictive scaling policy umsetzen, die zu einer Auto Scaling Group hinzugefügt wird *[15]*. Wenn die Lastspitzen bekannt sind und regelmäßig zu klar definierten Zeiten auftreten, kann auch ein CronJob eingerichtet werden, der die Skalierung automatisch ausführt.

Eine alternative Strategie ist „On-Availability"-Skalierung, was auch als Demand Shaping bezeichnet wird. Dieser Ansatz bietet sich für Workloads ohne harte zeitliche Anforderungen und ohne Nutzerinteraktion an, wie beispielsweise Event-Streaming-Anwendungen. Dabei werden die genutzten Ressourcen limitiert und die Skalierung der Anwendungen orientiert sich an den freien Kapazitäten, wie beispielsweise der CPU-Leistung. Um diese Strategie sinnvoll umsetzen zu können, sollte jedoch eine Priorisierung von Workloads erfolgen. In Kubernetes ist dies mithilfe von Priority Classes umsetzbar. Niedrig priorisierte Workloads werden für höher priorisierte potenziell terminiert und daher möglicherweise langsamer ausgeführt. Dadurch werden Lastspitzen entzerrt und Überprovisionierung wird nicht mehr benötigt.

Nach Berücksichtigung der oben genannten Ansätze lässt sich dann eine potenziell höhere Zielauslastung der Server erreichen, da entweder ausreichend schnell skaliert wird oder die verfügbaren Ressourcen höher priorisierten Workloads eher zur Verfügung gestellt werden. Dennoch stellt sich die Frage, welcher Effekt damit auf den Energieverbrauch des Cloud-Betriebs erzielt werden kann. Die drei großen Cloud-Anbieter Azure, AWS und Google Cloud ermöglichen es derzeit leider noch nicht, den Energieverbrauch darzustellen. Allerdings bieten sie verschiedene Monitoring-Tools an, die die CO2e-Emissionen der eigenen Cloud-Nutzung verbildlichen [16, 17, 18]. Eine Open-Source-Alternative ist das Tool Cloud Carbon Footprint, das anbieterunabhängig genutzt werden kann *[19]*. Alle genannten Tools können noch keine detaillierten Analysen erstellen, eignen sich jedoch für das Nachvollziehen von ersten Optimierungsansätzen.

Zukünftige Entwicklung

In diesem Artikel wurde ein Überblick über die bestehenden Möglichkeiten gegeben, wie im Cloud-Native-Umfeld Ressourcenverbräuche im Rahmen der Entwicklung und des Betriebs von Softwaresystemen reduziert werden können. Derzeit stehen hier vor allem die Vorteile im Vordergrund, die allgemein mit einer Effizienzsteigerung des Softwaresystems erzielt werden können, etwa besseres Antwortzeitverhalten oder Reduzierung der Cloud-Provider-Kosten durch geringere Ressourcennutzung. Die aktuellen Entwicklungen werden die Anstrengungen in diesem Bereich weiter forcieren. Beispielsweise durch strengere Vorgaben der EU im Bereich des Reporting von CO2e-Emissionen für Unternehmen *[20]*. Ein weiteres Beispiel sind die Entwicklung von Umweltzeichen wie dem Blauen Engel für Softwareprodukte *[21]*. Es ist also ratsam, sich der aktuellen Entwicklungen bewusst zu werden und bereits jetzt die Weichen für ressourceneffiziente Software zu stellen.

Quellen

[1] R. Hintemann, S. Hinterholzer, M. Graß, & T. Grothey: Bitkom-Studie: Rechenzentren in Deutschland 2021 – Aktuelle Marktentwicklungen, 2021. Borderstep Institut, Berlin.

[2] Bundesnetzagentur: Pressemitteilung – Bundesnetzagentur veröffentlicht Daten zum Strommarkt 2021, 2022. Bundesnetzagentur, Mainz.

[3] Europäischer Rat: Fit für 55, 2022. [Online]. Available: *https://www.consilium.europa.eu/de/policies/green-deal/fit-for-55-the-eu-plan-for-a-green-transition/* [Zugriff am 11.11.2022]

[4] Ann Steffora Mutschler: Improving Energy And Power Efficiency In The Data Center, 2021. [Online]. Available: *https://*

semiengineering.com/improving-energy-and-power-efficiency-in-the-data-center/ [Zugriff am 11.11.2022]

[5] A. Brunnert, H. Krcmar. 2017. Continuous performance evaluation and capacity planning using resource profiles for enterprise applications. Journal of Systems and Software. *http://dx.doi.org/10.1016/j.jss.2015.08.030*

[6] A. Brunnert, K.Wischer, H.Krcmar. 2014. Using architecture-level performance models as resource profiles for enterprise applications. In Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures (QoSA '14). Association for Computing Machinery, New York, NY, USA, 53–62. *https://doi.org/10.1145/2602576.2602587*

[7] RETIT Continuous Delivery (RCD), 2022. [Online]. Available: *https://www.retit.de/home/#tab-12a2c74f9b708d0382f* [Zugriff am 15.11.2022]

[8] RETIT OTEL Java Agent Extension: Capturing even more with the RETIT Java Agent Extension, 2022. [Online]. Available: *https://www.retit.io/documentation/opentelemetry.html#part5* [Zugriff am 11.11.2022]

[9] OpenTelemetry Instrumentation for Java, 2022. [Online]. Available: *https://github.com/open-telemetry/opentelemetry-java-instrumentation* [Zugriff am 14.11.2022]

[10] Jaeger: open source, end-to-end distributed tracing, 2022. [Online]. Available: *https://www.jaegertracing.io/* [Zugriff am 11.11.2022]

[11] RETIT Performance Analytics Suite (PAS): Import, Analyse and Compare Performance Data, 2022. [Online]. Available: *https://www.retit.io/* [Zugriff am 11.11.2022]

[12] Lukasz Mastalerz: Quantifying greenness of FaaS, 2021. [Online]. Available: *https://www.linkedin.com/pulse/quantifying-greenness-faas-lukasz-mastalerz/* [Zugriff am 11.11.2022]

[13] Luiz André Barroso, Urs Hölzle: The Case for Energy-Proportional Computing, 2007. IEEE Computer Society.

[14] Amazon Web Service, Inc.: Vertical Pod Autoscaler, 2022. [Online]. Available:] *https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html* [Zugriff am 11.11.2022]

[15] Amazon Web Services, Inc.: Predictive scaling for Amazon EC2 Auto Scaling., 2022. [Online]. Available: *https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html* [Zugriff am 11.11.2022]

[16] Microsoft Corporation: Emissions Impact Dashboard for Azure, 2022. [Online]. Available: *https://appsource.microsoft.com/en-us/product/power-bi/coi-sustainability.emissions_impact_dashboard* [Zugriff am 11.11.2022]

[17] Amazon Web Services, Inc.: Customer Carbon Footprint Tool, 2022. [Online]. Available: *https://aws.amazon.com/aws-cost-management/aws-customer-carbon-footprint-tool/* [Zugriff am 11.11.2022]

[18] Google Cloud: Carbon Footprint, 2022. [Online]. Available: *https://cloud.google.com/carbon-footprint?hl=de* [Zugriff am 11.11.2022]

[19] Cloud Carbon Footprint: Cloud Carbon Footprint – Free and Open Source, 2022. [Online]. Available: *https://www.cloudcarbonfootprint.org/* [Zugriff am 11.11.2022]

[20] Corporate sustainability reporting, 2022. [Online]. Available: *https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en* [Zugriff am 11.11.2022]

[21] Ressourcen- und energieeffiziente Softwareprodukte (DE-UZ 215), 2020. [Online]. Available: *https://www.blauer-engel.de/de/produktwelt/ressourcen-und-energieeffiziente-software-produkte* [Zugriff am 11.11.2022]

Richard Vobl
RETIT GmbH
info@retit.de

Denis Angeletta
RETIT GmbH
info@retit.de

Richard Vobl und Denis Angeletta sind Software Performance Consultants bei der RETIT GmbH und lieben es, Performance-Probleme zu lösen.

Nadja Hagen
envite consulting GmbH
office@envite.de

Uwe Eisele
envite consulting GmbH
office@envite.de

Nadja Hagen und Uwe Eisele sind Consultants bei der envite consulting GmbH und arbeiten jeden Tag daran, die IT ein wenig „grüner" zu machen.

Mehr Nachhaltigkeit mithilfe der Cloud

Franz Stefan, Amazon Web Services

Der Klimawandel ist eine der größten Herausforderungen, die der Menschheit in den kommenden Jahrzehnten bevorsteht. Jede Zeile Code hinterlässt einen CO2-Fußabdruck. Somit liegt die Last auf uns allen, um sicherzustellen, dass dieser Fußabdruck so klein wie möglich ist. Es besteht ein erhebliches Einsparpotenzial an Energie. Dies gilt nicht nur für konventionelle Anwendungen wie Betriebssysteme, Bürotechnik oder Serveranwendungen. Hochskaliert auf Hunderte, Tausende oder sogar Millionen von Geräten (Desktops, Smartphones, Tablets...), kann jedes einzelne Codeartefakt einen wichtigen Beitrag zur Reduzierung des eigenen Energiekonsums oder des Emissionsausstoßes machen.

Nachhaltigkeitsbereiche im Fokus

Nachhaltigkeit umfasst viele Bereiche. Es gibt mehrere Möglichkeiten, sich dem Thema anzunähern.

Ein Bereich betrifft die **Dekarbonisierung**, in dem es vorwiegend darum geht, Kohlendioxidemissionen zu reduzieren beziehungsweise durch kohlenstoffärmere Energiequellen zu ersetzen. Weiterhin geht es darum, **Wasser** so effizient wie möglich einzusetzen und, wenn ein Verzicht nicht möglich ist, auch dafür Sorge zu tragen, Wasser wiederzuverwenden.

Nachhaltigkeit umfasst aber auch Bereiche, die nicht unbedingt mit ökologischen Gesichtspunkten verbunden sind. Es gibt auch die **soziale Nachhaltigkeit**, bei der es um Themen wie gleiche Bezahlung oder Menschenrechte geht.

Zu guter Letzt geht es natürlich auch um das Thema **Kreislaufwirtschaft**. Also wie vermeidet man, dass produzierte Güter auf einer Deponie landen und stattdessen idealerweise wieder in einem Kreislauf landen.

Auf den ersten Blick mögen diese Bereiche etwas abstrakt erscheinen, wenn man an nachhaltige Transformationen denkt. Doch wenn man es näher beleuchtet, spielen alle diese Bereiche auch eine Rolle aus Entwicklersicht. Unser Code rennt in den meisten Fällen auf Applikationsservern. Der darunterliegende Host ist entweder ein Hypervisor, Bare-Metal-Rechner oder die Anwendung rennt in einer serverlosen Umgebung. Die Infrastruktur selbst läuft vorwiegend in einem Rechenzentrum, was die Cloud einschließt.

Ein Rechenzentrum zu betreiben, kann sehr energieintensiv sein. Viele produzieren auch direkte Emissionen. Darunter fallen jene Bereiche, in denen man selbstständig daran beteiligt ist, Emissionen zu verbrennen. Beispiele dafür sind Dieselgeneratoren, die im Einsatz sind, wenn der Strom ausfällt. Aber auch in der Wertschöpfungskette werden direkte Emissionen produziert, wenn man kraftstoffbetriebene Fahrzeuge benutzt. Hier wäre ein Verbesserungsansatz, in die Elektrifizierung zu gehen.

Indirekte Emissionen entstehen etwa beim Einkauf von Strom. Hier besteht oft ein Energiemix, der nicht immer zu 100 % aus erneuerbarer Energie besteht.

Zu guter Letzt gibt es in der ganzen Wertschöpfungskette auch Lieferanten. Wenn ein Rechenzentrenanbieter zum Beispiel Hardware bezieht, werden auch Emissionen freigesetzt; sei es bei der Herstellung oder auch bei der Distribution (Flugzeug, LKW).

Woher kommt der Trend zu einem nachhaltigen Miteinander? Einerseits verlangen Konsumenten immer nachhaltigere Güter. Was aber immer öfter zu beobachten ist, ist, dass auch die Gesetzgebung mehr Nachhaltigkeit einfordert, sei es durch Regulatorien, um Emissionen einzusparen oder durch Berichtspflichten. Darüber hinaus verlangen Mitarbeiter/innen ihren Arbeitgebern eine erhöhte Nachhaltigkeitspflicht ab. Dass es auch Gründe zur Wettbewerbspositionierung oder mehr Regeln seitens Kapitalgeber gibt, sei noch am Rande erwähnt. Letzten Endes ist es ein Thema, das uns alle angeht und ambitionierte Ziele verfolgt.

Um den Bogen zum eigentlichen Thema wieder zu schließen, ist es wichtig, in der Applikationsentwicklung effizient zu sein. Dies spart Rechen- und Speicherressourcen und im Umkehrschluss wird die Anwendung beziehungsweise der Betrieb auch günstiger.
Weiterhin laufen viele Anwendungen immer öfter in der Cloud. Eine Migration zu einem Cloudanbieter führt in den meisten Fällen zu einer Emissionsreduktion (im Vergleich zum On-Premises-Rechenzentrum). Die Gründe liegen einerseits in der Verteilung der Workloads, um eine bessere Auslastung zu gewährleisten, neueste und effizientere Prozessoren, Investitionen in Bezug auf aktuellere und nachhaltigere Hardware, Innovationen in Bezug auf neue Kühlungsmöglichkeiten, Investitionen in erneuerbare Energien und noch weiteren Maßnahmen. Hier gibt es jedoch noch sehr viel Verbesserungspotenzial, wenn es um die Anwendung an sich geht und der Einflussbereich mehr beim Entwickler oder Architekten liegt – Stichwort geteilte Verantwortung.

AWS Well-Architected – Nachhaltigkeit

Das AWS Well-Architected Framework *[1]* unterstützt Sie dabei, die Vor- und Nachteile der Entscheidungen nachzuvollziehen, die Sie beim Erstellen von Workloads in AWS treffen. Das Framework hilft Ihnen, bewährte Architekturmethoden für den Entwurf und Betrieb zuverlässiger, sicherer, effizienter, kosteneffektiver und nachhaltiger

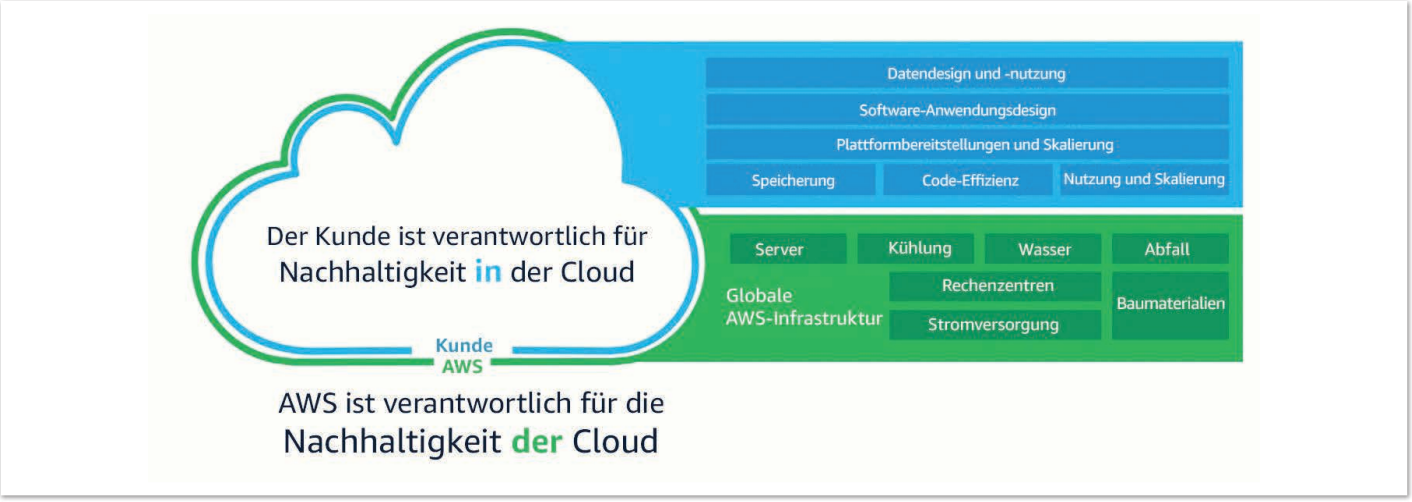


Abbildung 1: Geteilte Verantwortlichkeit – Nachhaltigkeit ([Amazon Web Services])

Workloads in der AWS-Cloud zu ermitteln. Es bietet Ihnen die Möglichkeit, Ihre Architekturen konsistent auf die Einhaltung bewährter Methoden zu prüfen und Verbesserungspotenzial zu identifizieren. Das Framework besteht aus den Säulen Operational Excellence, Sicherheit, Zuverlässigkeit, Leistungseffizienz, Kostenoptimierung und wurde vor Kurzem durch die sechste Säule Nachhaltigkeit ergänzt.

Die ökologische Nachhaltigkeit ist eine gemeinsame Verantwortung von Kunden und AWS *(siehe Abbildung 1)*.

- AWS ist verantwortlich für die Optimierung der Nachhaltigkeit des Cloud-Faktors – die Bereitstellung effizienter, gemeinsam genutzter Infrastrukturen, der verantwortungsvolle Umgang mit Wasser und die Beschaffung erneuerbarer Energie.
- Kunden sind verantwortlich für Nachhaltigkeit in der Cloud – Optimierung von Workloads und Ressourcennutzung sowie Minimierung der Gesamtressourcen, die für Ihre Workloads bereitgestellt werden müssen.

Es gibt hierzu viele Ressourcen von AWS, die Ihnen mit konkreten Beispielen helfen, Maßnahmen zu treffen, nachhaltiger zu sein. Einige Beispiele möchte ich aber aufführen:

- Löschen von redundanten oder nicht mehr benötigten Daten
- Verwenden von Lifecycle-Policies, um automatisiert nicht mehr benötigte Daten zu löschen oder in andere Speicherkategorien zu bewegen
- Verwenden Sie Technologien, die Ihnen bei Datenzugriffs- und Speichermuster helfen
- Vermeiden Sie eine Überprovisionierung von Blockspeichermedien
- Evaluieren Sie alternative Datenformate wie Parquet oder Avro anstatt CSV
- Verwenden Sie nur die nötige Anzahl an Hardware, um ihre Ziele zu erreichen
- Verwenden Sie Instance-Typen wie Graviton3, deren Prozessoren effizienter sind als ähnliche Instance-Typen
- Berücksichtigen Sie die Verwendung von gemanagten Services
- Berücksichtigen Sie die Verwendung von CDN-Diensten
- Berücksichtigen Sie die Verwendung von Serverless-Diensten

Aus Unternehmenssicht gibt es historisch geshen oft noch ein geringes Bewusstsein, den Energieverbrauch im Softwarebereich zu verringern. Dies ist auch dem Umstand geschuldet, dass Entwicklungs- und Betriebsbudgets meist getrennt sind. Der Weg zu mehr Nachhaltigkeit ist aus Unternehmenssicht sicher kein leichtes Unterfangen und erfordert eine End-to-End-Sichtweise.

Letztlich bedeutet es, im Designprozess zum Thema Green Coding einen neuen Aspekt zu berücksichtigen. Teams müssen sich fragen, ob es einen besseren Weg gibt, den gewünschten Nutzen mit dem geringstmöglichen Energieaufwand zu erzielen. Die Antwort auf diese Frage kann einen großen Einfluss auf das Design haben.

Zero Code Waste

In mehr als 90 % modernerer Softwareprojekte wird man Open-Source-Artefakte von Dritten wiederfinden. Was per se nicht schlecht ist, da es zu einer Anforderung in einer Bibliothek eine fertige Lösung gibt, doch andererseits steigt die Wahrscheinlichkeit redundanter Codezeilen, je häufiger man diese Bibliotheken in einem Projekt verwendet.

Im ersten Schritt sollte man sich auf jenen Code konzentrieren, der niemals ausgeführt wird. Insbesondere auf Bibliotheken, die als Abhängigkeit definiert werden. Dies manuell durchzuführen, kann sehr zeitaufwendig sein und ist auch mit Risiken verbunden. Ein guter Weg, um „toten Code" zu identifizieren, ist zum Beispiel der Einsatz von Bibliotheken wie ArchUnit *[2]*.

Einige mögen schon andere Tools zum Auffinden von Abhängigkeiten zwischen Packages und Klassen kennen, wie AspectJ, Checkstyle oder Findbugs. Jedes Tool hat seine Stärken und Besonderheiten und darüber hinaus noch weitere Einsatzfelder. Ein Vorteil von ArchUnit ist die Möglichkeit, benutzerdefinierte Regeln zu schreiben, bei denen auf importierte Klassen zugegriffen werden kann, ähnlich wie bei der Verwendung des Reflection-API. Es bietet einen natürlichen Ansatz, die volle Leistungsfähigkeit des Reflection-API für Ihre Tests zu nutzen. Aber es erlaubt auch, Tests zu schreiben, die Feldzugriffe, Methoden- oder Konstruktoraufrufe und Unterklassen untersuchen. Darüber hinaus benötigt es weder eine spezielle Infrastruktur noch eine neue Sprache, es ist pures Java und Regeln können mit jedem Unit-Testing-Tool wie JUnit evaluiert werden.

Mit mehr effizientem Code und Hilfe von Amazon CodeGuru Profiler zu mehr Nachhaltigkeit

Mithilfe des Amazon CodeGuru Profiler *[3]* finden Sie die ressourcenaufwendigsten (teuersten) Codezeilen und es hilft Ihnen dabei, das Laufzeitverhalten der Anwendungen besser zu verstehen. Ineffiziente Codezeilen lassen sich identifizieren und zur Optimierung der Leistung und auch zur Senkung der Kosten anpassen. Der Profiler analysiert das Laufzeitprofil der Anwendungen und ist somit in der Lage, intelligente Empfehlungen zur Codeoptimierung und somit auch zur nachhaltigeren Ressourcennutzung zu geben. Neben Java wird auch noch Python unterstützt.

Der Profiler funktioniert mithilfe eines Agenten, der zusammen mit der Anwendung als Thread gestartet wird und als Teil der Anwen-

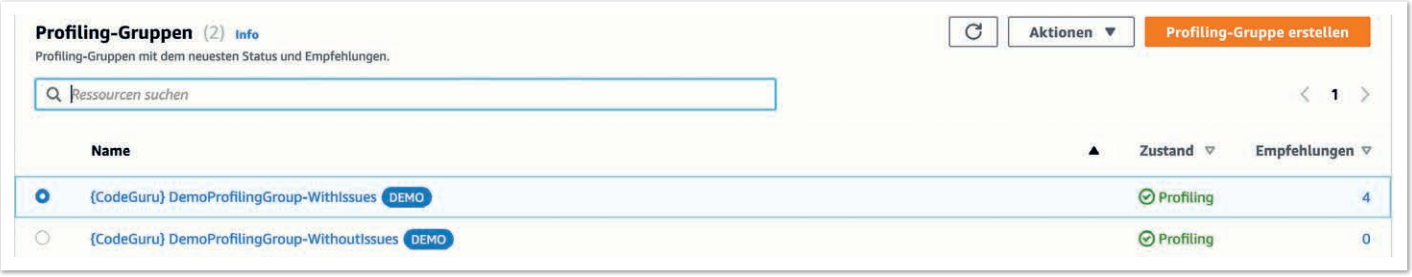


Abbildung 2: Übersicht Profiling-Gruppen ([Amazon Web Services])

dung läuft. Weiterhin gibt es noch einen Profiler-Service, der die gelieferten Daten aggregiert, die anschließend zur Visualisierung der Leistung einer Anwendung nutzbar sind. Interaktive Diagramme zeigen die Ergebnisse der CPU- und Latenzanalysen. Der CodeGuru Profiler analysiert die Anwendungslaufzeiten kontinuierlich, findet Anomalien, generiert Warnungen und erstellt intelligente, ML-gestützte Empfehlungen.

Die Demo beziehungsweise der dazugehörige Code ist hier verfügbar *[4]*. Das Beispieldashboard finden Sie im dazugehörigen Service „AWS CodeGuru" in der AWS-Managementkonsole.

Die Demo erstellt zwei Beispiel-Profiling-Gruppen für eine Java-Anwendung. Eine Profiling-Gruppe umfasst Anwendungen, die zusammen als Einheit profiliert werden. Anwendungsdaten werden vom Profiler-Agenten von Amazon CodeGuru Profiler an eine einzelne Profiling-Gruppe gesendet. Daten aus allen Anwendungen in einer Profiling-Gruppe werden aggregiert und gemeinsam analysiert. Die zwei Beispiel-Profiling-Gruppen veranschaulichen, wie CodeGuru Profiler mit Anwendungsgruppen umgeht, die Beeinträchtigungen haben (**{CodeGuru} DemoProfilingGroup-WithIssues**) oder keine (**{CodeGuru} DemoProfilingGroup-WithoutIssues**) *(siehe Abbildung 2)*.

Der **Profiling-Gruppenstatus** zeigt relevante Metriken an, die während der Profilerstellung erfasst wurden, wie die durchschnittliche Anzahl von deployten Agenten, den Zustand der Profilierung, Anzahl an Anomalien und auch die Anzahl der Empfehlungen, die im Zuge der Profilierung gefunden wurden *(siehe Abbildung 3)*.

- **CPU-Zusammenfassung** liefert Aufschluss darüber, wie viele CPU-Ressourcen von den profilierten Anwendungen verwendet werden.
- **CPU-Auslastung** gibt einen Hinweis darauf, wie viel der verfügbaren CPU-Ressourcen von der profilierten Anwendung verbraucht wird. Der Metrikwert ist ein Mittelwert aller Instances, die Daten an diese Profiling-Gruppe melden. Ein niedriger Wert (weniger als 10 %) deutet an, dass Ihre Instance überdimensioniert sein könnte. Ein hoher Wert (mehr als 90 %) deutet wiederum an, dass Ihre provisionierte Infrastruktur unterdimensioniert sein könnte.
- **CPU-Nutzung** des Agenten liefert eine Schätzung, wie viel der System-CPU-Ressourcen vom CodeGuru-Profiler-Agenten durchschnittlich über profilierte Instances verbraucht werden. In den meisten Fällen ist zu erwarten, dass dieser Wert niedrig ist (weniger als 1 %).

Abbildung 3: CodeGuru Profiler Dashboard ([Amazon Web Services])

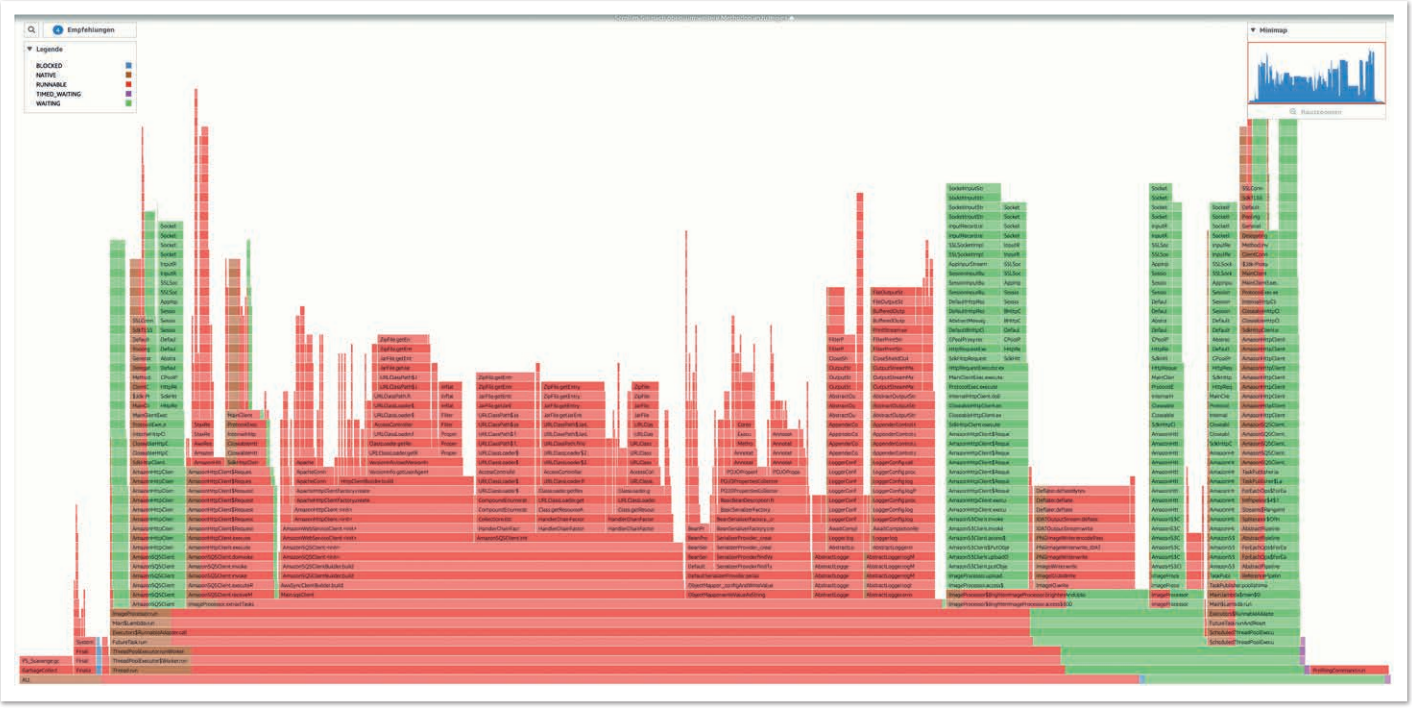


Abbildung 4: CodeGuru Profiler Latenzvisualisierung ([Amazon Web Services])

- **Für die Ausführung von Code aufgewendete Zeit** – diese Metrik zeigt an, wie häufig die Threads Ihrer Anwendung im „Ausführbar"-Thread-Status waren. Ein hoher Prozentsatz (mehr als 90 %) gibt an, dass die meiste Zeit Ihrer Anwendung für die Ausführung von Operationen auf der CPU aufgewendet wurde. Ein sehr niedriger Prozentsatz (weniger als 1 %) gibt an,

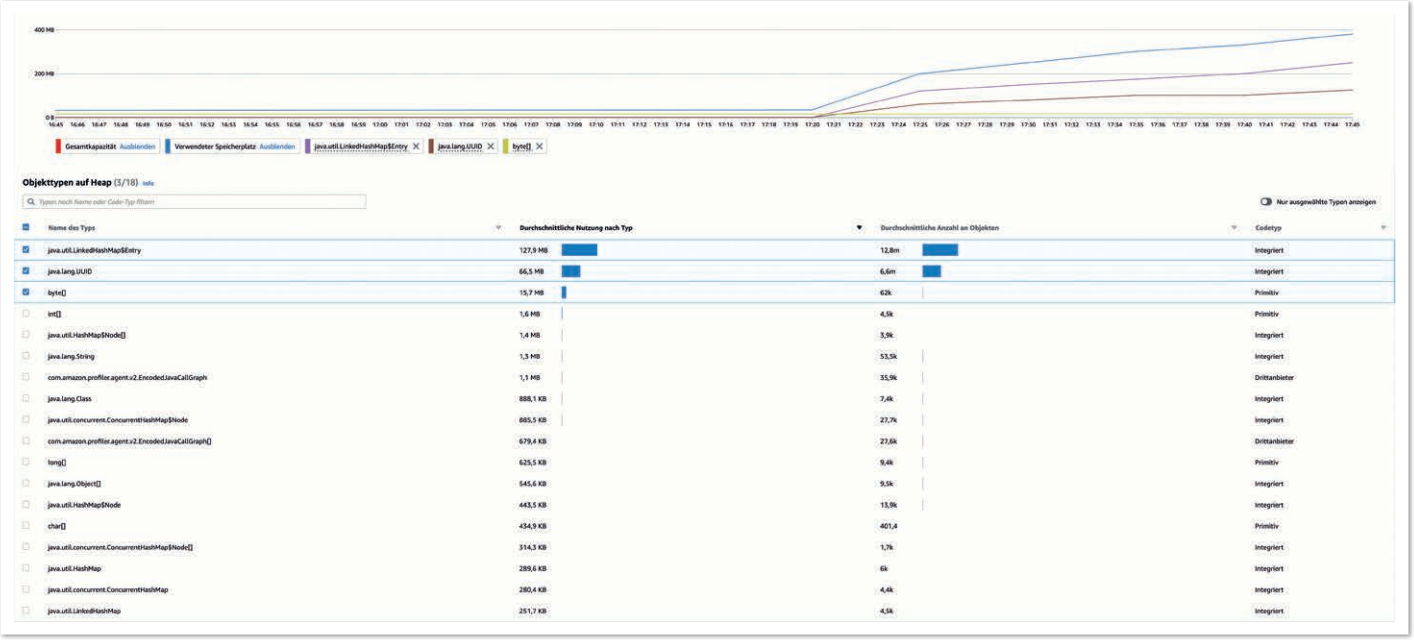


Abbildung 5: CodeGuru Profiler Heapvisualisierung ([Amazon Web Services])

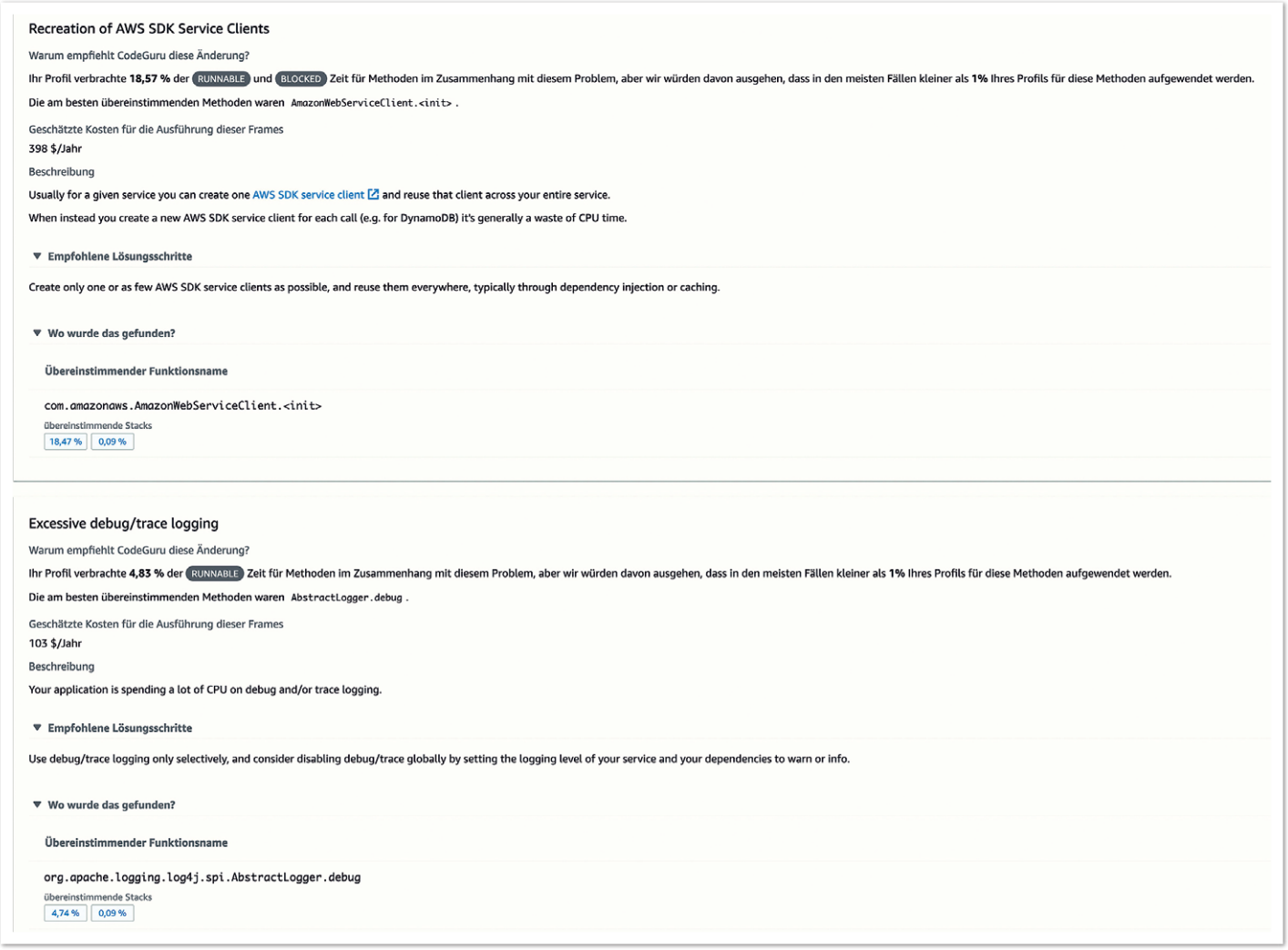


Abbildung 6: CodeGuru Profiler Empfehlungen Ausschnitt ([Amazon Web Services])

dass der Großteil Ihrer Anwendung in anderen Thread-Status verbracht wurde (etwa „Blockiert" oder „Warten"). Dies kann sich auf die Latenzwerte auswirken.

Die **Latenzübersicht** liefert Informationen darüber, wie oft sich Ihre Anwendungs-Threads im Status „Blockiert" oder „Leerlauf" befindet *(siehe Abbildung 4)*.

- **Blockierte Zeit** zeigt an, wie oft sich Ihre Anwendungs-Threads im Status „Blockiert" befinden, sobald wir alle Leerlauf-Threads ausschließen. Dies kann der Fall sein, wenn Ihre Anwendung häufig synchronisierte Blöcke oder Überwachungssperren verwendet. Die Latenz-Visualisierung kann Ihnen helfen zu verstehen, welche Codeabschnitte die Threads blockieren.
- **Wartezeit** zeigt an, wie viel Zeit die Threads Ihrer Anwendung im Thread-Status „Warten" und „Zeitgesteuertes Warten" verbracht haben, als Prozentsatz aller Thread-Status außer „Im Leerlauf". Dies wird häufig durch I/O-Operationen wie Netzwerkaufrufe oder Speicheroperationen verursacht. Die Latenzvisualisierung kann Ihnen helfen zu verstehen, welche Codeabschnitte Threads warten lassen.

Heap-Nutzung liefert Information über die Speicherzuweisung Ihres Anwendungsstacks *(siehe Abbildung 5)*.

- **Durchschnittliche Heap-Nutzung** zeigt an, wie viel der maximalen Heap-Kapazität Ihrer Anwendung durchschnittlich von Ihrer Anwendung für alle profilierten Instances verbraucht wird. Der angezeigte Prozentsatz ist der durchschnittliche verwendete Heap-Speicherplatz im Vergleich zur maximalen Heap-Kapazität der JVM.
- **Heap-Spitzennutzung** zeigt den höchsten Prozentsatz des von Ihrer Anwendung verbrauchten Speichers an. Dies basiert auf demselben Datensatz wie in der Heap-Zusammenfassung-Visualisierung.

Anomalien: CodeGuru Profiler erkennt Anomalien, indem Trends in Ihren Profiling-Daten analysiert und Abweichungen in diesen Daten erkannt werden. Alle gefundenen Anomalien werden hier zusammen mit Details dazu angezeigt, welcher Zeitraum anormal ist. Weitere Details finden Sie im verknüpften Bericht.

Empfehlungen: CodeGuru Profiler gibt Empfehlungen, mit denen Sie Ihre Anwendungen optimieren können. Hier finden Sie alle verfügbaren Empfehlungen sowie Details zu den geschätzten Auswirkungen auf Ihr gesamtes Anwendungsprofil. Weitere Details finden Sie im verknüpften Bericht *(siehe Abbildung 6)*.

Quellen

[1] Well Architect Framework- Nachhaltigkeit *https://docs.aws.amazon.com/de_de/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html* (2022)

[2] ArchUnit: *https://www.archunit.org/* (2022)

[3] Amazon CodeGuru Profiler *https://aws.amazon.com/de/codeguru/* (2022)

[4] Amazon CodeGuru Profiler demo application *https://github.com/aws-samples/aws-codeguru-profiler-demo-application/* (2022)

Franz Stefan
Amazon Web Services
stefanfz@amazon.at

Franz Stefan ist als Solutions Architect bei Amazon Web Services tätig. Die Jahre zuvor war er als Cloud Solutions Architect beziehungsweise als Java-Entwickler im Enterprise-Umfeld tätig. Die Meinungen des Autors sind seine eigenen und können von denen seines Arbeitgebers abweichen.

Mehr Produktivität durch den nachhaltigen Umgang mit uns selbst – oder: Warum auch uns der Energiesparmodus guttut!

Sabine Wojcieszak, getNext IT

Nachhaltigkeit wird heute zu Recht viel diskutiert. Dazu heißt es immer wieder: Nachhaltigkeit fängt bei dir an! Für die meisten bedeutet das, dass wir anfangen müssen, nachhaltig mit knappen Ressourcen umzugehen, Müll zu vermeiden oder keine Energie zu verschwenden. Dabei ist der Fokus immer nach außen gerichtet. Aber reicht das? Schon in den Prinzipien des Agilen Manifestes wird von Nachhaltigkeit gesprochen. In diesem Artikel wird diskutiert, wie Nachhaltigkeit auch nach innen gerichtet betrachtet werden kann und muss, wenn wir über unsere Produktivität nachdenken wollen.

Sicher hast auch du dir schon darüber Gedanken gemacht, wie du nachhaltiger leben und Nachhaltigkeit auch in deine tägliche Arbeit einbinden kannst. Doch hast du auch schon einmal darüber nachgedacht, was Nachhaltigkeit und Themen wie der Umgang mit knappen Ressourcen, Energiesparen oder Müllvermeiden mit dir und deinem Körper zu tun haben? Gerade in Bezug auf unsere Produktivität ist diese nach innen gerichtete Betrachtung wichtig, denn auch wir verfügen über wichtige, aber limitierte Ressourcen. Doch was sind eigentlich Ressourcen?

Je nach Disziplin hat der Begriff *Ressource* unterschiedliche Bedeutungen:

System Organisation	System Mensch/Team	
BWL, VWL, Organisation	**Psychologie**	**Soziologie**
Betriebsmittel, Geldmittel, Rohstoffe, Energie, Personen, (Arbeits-) Zeit	Fähigkeiten, persönliche Eigenschaften, geistige Haltung, „Stärken",	Bildung, Gesundheit (etwa körperliche und geistige Energie), Prestige, soziale Vernetzung

Im System Mensch können wir alle Ressourcen herunterbrechen auf die Ressource Energie:

- Haben wir zu wenig Fähigkeiten für eine Aufgabe, kostet es uns viel Energie, sie zu erledigen.
- Müssen wir immer nur an unseren Schwächen arbeiten, koste es viel Energie.
- Müssen wir ständig gegen unsere Überzeugung anarbeiten, verschwenden wir Energie.
- Ist unsere Gesundheit beeinträchtigt, kostet es uns viel Energie, trotzdem zu arbeiten, und vieles mehr.

Haben wir keine Energie mehr, nützen uns auch unsere Fähigkeiten, Stärken oder unser soziales Prestige nichts mehr. Unsere Produktivität geht dann automatisch gegen null.

Daher ist es wichtig, dass wir uns Klarheit über unsere Ressource Energie verschaffen.

- Wie viel Energie habe ich?
- Was sind Energiefresser?

Ein nachhaltiger Umgang mit der Ressource Energie ist absolut notwendig, wenn wir unsere Produktivität erhalten wollen. Wollen wir mit unserer Energie nachhaltig umgehen, benötigen wir auch hierfür eine Definition, um ein gemeinsames Verständnis sicherzustellen.

Nachhaltigkeit – eine Definition

Da der Begriff Nachhaltigkeit inzwischen in so vielen verschiedenen Kontexten auftaucht, folgt hier nun eine allgemeingültige Definition:

*„Nachhaltigkeit ist ein **Handlungsprinzip** zur **Ressourcen-Nutzung**, bei dem eine **dauerhafte Bedürfnisbefriedigung** durch die Bewahrung **der natürlichen Regenerationsfähigkeit** der **beteiligten Systeme** (vor allem von Lebewesen und Ökosystemen) gewährleistet werden soll. […]" [1]*

Übertragen wir einmal die verschiedenen in der Definition genannten Aspekte auf unsere nach innen gerichtete Betrachtung:

- **Handlungsprinzipien** ➔ unsere Verhaltensweisen
- **Ressourcen-Nutzung** ➔ wie gehen wir mit unseren Ressourcen um
- **Natürliche Regenerationsfähigkeit** ➔ was tun wir, um unsere Ressourcen wieder zu regenerieren
- **Beteiligte Systeme** ➔ das System Mensch oder ein zusammengesetztes System aus Menschen (Team)

Es geht also immer um Verhalten! Daher ist die Aussage „Nachhaltigkeit fängt bei uns an" absolut richtig, denn jedes Verhalten fängt bei uns an. Wenn wir also nachhaltiger mit uns selbst umgehen wollen, müssen wir unser Verhalten überdenken und managen. *Selbstmanagement* ist der Begriff, der an dieser Stelle genannt werden muss. Die gute Nachricht dabei ist:

An unserem Verhalten können wir immer etwas ändern, ohne dabei auf andere angewiesen zu sein!

Selbstverständlich muss in einem anderen Schritt dabei auch das System Organisation betrachtet werden. Doch wenn sich das System Organisation anders verhält, als wir es uns wünschen, können wir durch unser Verhalten einen nachhaltigen Umgang mit unseren eigenen Ressourcen praktizieren. Bisweilen ist schnell der Begriff *Zeitmanagement* genannt, wenn es um das Thema knappe Ressourcen geht.

Doch zunächst einmal erfordern alle bekannten Zeitmanagement-Methoden eine Verhaltensänderung. Diese wiederum ist nur möglich, wenn wir uns selbst managen können und wollen. Außerdem impliziert der Begriff, dass wir die Zeit managen können. Doch gerade im Arbeitskontext wird die uns zur Verfügung stehende Bearbeitungszeit von dem System Organisation gewährt. Oft ist unser Einfluss darauf, wie viel Zeit wir genau bekommen, begrenzt.

Die Zeit wiederum macht, was Zeit so macht:
Sie läuft gleichmäßig voran.

Es interessiert die Zeit überhaupt nicht, welche Zeitmanagement-methode wir einsetzen und ausprobieren.

Das mag etwas frustrierend klingen, ist es aber gar nicht. Denn was wir managen können, ist, wie und wofür wir die uns zur Verfügung stehende Energie in die begrenzt zur Verfügung stehende Ressource Zeit einbringen beziehungsweise wie wir die Zeit nutzen. Hinzu kommt beim Selbstmanagement der Bereich der Selbstfürsorge. Hier geht es unter anderem darum, sich darüber klar zu werden was

- mir gut tut,
- mir nicht gut tut und
- was ich brauche, um meine Energie wieder aufzuladen.

Schlaf, Ernährung und der Flüssigkeitshaushalt sind dabei nur ein paar wichtige Faktoren. Hinzu kommen Aspekte wie Bewegung, Beschäftigung, Ausgleich oder die Menschen, mit denen wir uns umgeben.

Mit dieser Betrachtung können wir unsere persönlichen Energiefresser und -booster erkennen und entsprechende Entscheidungen treffen.

Doch verbrennen wir mit unserer Arbeit mehr Energie, als wir in der Freizeit wieder aufbauen können, so betreiben wir Raubbau mit langfristigen Konsequenzen, nicht nur für unsere Produktivität, sondern auch für unsere Gesundheit. Gerade im Arbeitskontext sind es häufig wir selbst, die

in dem Glauben, etwas Gutes zu tun,

sinnlos unsere eigene Energie verschwenden. Multitasking ist wohl das prominenteste Beispiel hierfür.

Multitasking – die Illusion eines Super-Skills

Multitasking ist das Bearbeiten von zwei oder mehreren Aufgaben gleichzeitig. Unterschieden wird dabei zwischen drei Arten von Multitasking:

- gleichzeitiges Abarbeiten von zwei oder mehr Aufgaben
  - *telefonieren und dabei eine E-Mail schreiben*
  - *Teilnahme an zwei Online-Meetings gleichzeitig*
  - *Teammitglied erzählt was, während ich noch am Rechner arbeite*
- Task-Switching
  - *hin- und herspringen zwischen zwei Aufgaben*
  - *„Kannst du mal schnell"*
- von einer Aufgabe zur nächsten ohne Pause
  - *Online-Meetings, die nahtlos ineinander übergehen*

Bei einfachen Aufgaben (zum Beispiel duschen und dabei singen) ist Multitasking kein Problem.

Werden Aufgaben jedoch komplexer, wird die

👉 Bearbeitungszeit für die einzelnen Aufgaben nicht nur länger;
👉 auch die Qualität kann schlechter und
👉 die Fehlerzahl größer werden.

Obwohl wir der Überzeugung waren, etwas Gutes für unsere Produktivität gemacht zu haben, haben wir genau das Gegenteil erreicht: Wir haben viel Zeit verschwendet. Hinzu kommt, dass

👉 Multitasking ein absoluter Energiefresser für uns ist.

Multitasking erzeugt nicht nur negativen Stress und führt zur Erschöpfung; auch unser Gedächtnis wird geschwächt *[2]*. Entsteht durch Fehler oder schlechte Qualität noch die Notwendigkeit zur Nachbearbeitung, so kostet uns das Multitasking noch mehr Zeit. Laut einer Studie der American Psychological Association

kostet uns das Task-Switching bis zu 40 % unserer Produktivität. [3]

Wenn wir dann noch bedenken, dass nur ca. 2 % der Weltbevölkerung *[4]* überhaupt in der Lage sind, wirkliches Multitasking zu betreiben, wird klar, wie viel Zeit und Energie wir durch diesen Mythos eines Super-Skills verschwenden.

Aus diesem Wissen und Verständnis heraus können wir für uns ein hilfreiches Verhalten ableiten, das uns einerseits in den Energiesparmodus versetzt und gleichzeitig unsere Produktivität erhöht:

Fokussiert an einer Aufgabe gleichzeitig arbeiten!

Folgende Tipps können dabei helfen, das zu erreichen:

👉 **Fokuszeit einplanen und mögliche Störungen minimieren**
- 💡 zum Beispiel Pomodoro-Methode *[5]* anwenden
- 💡 Benachrichtigungen ausschalten
- 💡 Zeitblocker im Kalender eintragen

👉 **angemessen kommunizieren**
- 💡 im Team ankündigen, dass du jetzt gerne fokussiert arbeiten möchtest und Probleme/Fragen entweder jetzt oder später geklärt werden können
- 💡 ansprechen, wenn zu viele Aufgaben bei dir auf dem Tisch liegen

👉 Regeln im Team vereinbaren, wie mit Fokus-Zeiten umzugehen ist

👉 **Priorisierung von Aufgaben**
- 💡 entweder durch dich, das Team oder die Führungskraft

👉 **Teammitglieder nicht stören, wenn sie gerade konzentriert arbeiten**
- 💡 private Gespräche nicht am Schreibtisch führen
- 💡 Fragen später stellen; etwa im Teamchat die andere Person bitten, sich zu melden, sobald sie Zeit dafür hat

👉 **Pausen zwischen Meetings einplanen – auch online**

Sinnvoll oder nicht sinnvoll?

Wenn wir unsere Produktivität betrachten wollen, müssen wir stets auf die zur Verfügung stehende Zeit und unsere bereitstehende Energie schauen. In Bezug auf die Zeit haben wir nur einen Ansatzpunkt: Wie kann ich die zur Verfügung stehende Zeit möglichst sinnvoll einsetzen. Was „möglichst sinnvoll" genau bedeutet, hängt von dem Ziel ab, das erreicht werden soll.

Für unser eigenes, nachhaltiges Energiemanagement ist es absolut erforderlich zu verstehen, was mit unserer Arbeit erreicht und bewirkt werden soll: sinnvoll, zielführend und wichtig. Wer losarbeitet, ohne verstanden zu haben, wohin die Reise gehen soll, läuft Gefahr, viel Energie unnötig zu verschwenden. Hinzu kommt der Motivationsverlust, wenn sich am Ende rausstellt, dass wir das Falsche gemacht haben.

Doch da wir für unser eigenes Energiemanagement verantwortlich sind, ist es an uns,

💡 so lange zu fragen, bis das Ziel klar ist.

Nur wenn wir das Ziel kennen, können wir die knappe Ressource – unsere Energie – sinnvoll in die begrenzte Ressource Zeit einsetzen.

Der Erfolg, den wir so erreichen können, ist dann auch wieder ein Energiebooster!

Probleme und Störungen im Team – Energiefresser oder -booster?

Probleme und Störungen im Team wird es immer geben. Die Frage ist, wie wir damit umgehen.

Klar ist, dass Probleme und Störungen im Team immer erst einmal Energie kosten. Adressieren wir sie nicht zeitnah, werden sie zu ausgewachsenen Energiefressern. Eine Aussage wie: „Ich sage nichts, weil es nichts bringt und mich zu viel Energie kostet!", ist ein Zeichen dafür, dass hier auch in nächster Zeit viel Energie verschwendet wird. Ein echter Energiekiller ist es, sich dauerhaft über Dinge zu ärgern! Das zehrt an unseren Nerven, an unserer Motivation und an der Energie! Wenn wir uns dann auch noch immer bei anderen beschweren und über die Situation meckern, dann raubt es auch den Mitarbeitenden Energie. Wer sich in einer Situation befindet, sich ständig über etwas zu ärgern, bei einem gleichzeitigen Gefühl, nichts ändern zu können, sollte eine Entscheidung treffen! Entweder bleiben und die Umstände wirklich einfach akzeptieren – und sich dann nicht mehr ärgern – oder gehen!

Manchmal müssen wir uns trennen, um Energie zu sparen!

Werden Probleme und Störungen jedoch zeitnah und wertschätzend adressiert, so beinhalten sie das Potenzial, echte Energiebooster zu werden. Allein die Tatsache, dass etwas verändert wird, ist schon positiv. Aber auch das Wissen, sich in einem Umfeld zu befinden, in dem Empfindungen, Anregungen, Wünsche und Ideen ernst genommen werden, gibt uns Energie zurück.

Einfach mal drüber sprechen!

Ein wichtiger Aspekt, um Energiefresser im Team zu erkennen, zu eliminieren, Booster einzubauen und zu nutzen, ist es, einfach mal im Team darüber zu sprechen.

- 👉 Tauscht euch aus, wie es um eure persönliche Energie steht! Beispielsweise als Thema in einer Retrospektive
- 👉 Schafft euch eine „Energielevel-Anzeige" für das Team, etwa für den Tag, in Meetings, die Woche *(siehe Abbildung 1)*

Abbildung 1: Energielevel-Anzeige für das Team (© Sabine Wojcieszak)

Die Macht der Führungskraft

Jede Führungskraft sollte sich darüber bewusst sein, wie viel Macht sie hat – Macht über das Wohlbefinden von Menschen, über Produktivität und über Veränderungen.

Ja, das Führungsverhalten beeinflusst die Produktivität – im Guten wie im Schlechten. Auch eine Führungskraft sollte sich fragen, inwieweit sich ihr Verhalten auf die Energie von Menschen auswirkt, und das eigene Führungsverhalten entsprechend anpassen.

Ein Beispiel: Da Multitasking kontraproduktiv ist, sollte eine Führungskraft die Mitarbeitenden gar nicht erst in diese Situation bringen. Sie sollte vielmehr sogar darauf achten, dass Mitarbeitende nicht in dieses Verhalten fallen und als Coach fungieren.

Fazit: Energiesparmodus einschalten

Unsere Produktivität ist essenziell davon abhängig, wie viel Energie wir haben. Daher ist es kein „moderner Wohlfühlanspruch", wenn wir uns darüber Gedanken machen, was uns Energie raubt oder bringt. Es ist vielmehr eine wirtschaftliche Notwendigkeit, sich darüber Gedanken zu machen – sowohl auf Individualebene als auch auf Unternehmensebene! Rahmenbedingungen zu schaffen, die es ermöglichen, das eigene Verhalten so zu gestalten, dass es uns nachhaltig mit unserer Energie wirtschaften lässt, ist absolut notwendig. Doch noch wichtiger ist es, dass jede einzelne Person sich darüber im Klaren ist, dass unser Verhalten der Schlüssel für einen guten Energiehaushalt ist. Nur, wenn wir uns entsprechend reflektieren und unser Verhalten anpassen und den Energiesparmodus einschalten, werden wir langfristig produktiv bleiben. Genau das liegt in unserer eigenen Verantwortung! Also fang an, deine Energie zu sparen!

Quellen

[1] *https://de.wikipedia.org/wiki/Ressource*
[2] *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7644608/*
[3] *https://www.wrike.com/de/blog/der-hohe-preis-des-multitaskings-40-produktivitaetsverlust-durch-staendigen-aufgabenwechsel/*
[4] *https://www.newyorker.com/science/maria-konnikova/multitask-masters*
[5] *https://www.impulse.de/management/selbstmanagement-erfolg/pomodoro-technik/7292581.html*

Sabine Wojcieszak
getNext IT
sabine@getnext-it.com

Sabine Wojcieszak ist Trainerin und Coach bei getNext IT in Kiel. Sie arbeitet mit Unternehmen, Führungskräften und Teams an einer erfolgreicheren Zusammenarbeit mit mehr Spaß. Für sie ist es wichtig, das „große Schweigen" in Unternehmen zu beenden, um offen, wertschätzend und konstruktiv über die wichtigen und notwendigen Dinge zu sprechen. Sie ist überzeugt, dass Erfolg nur mit und durch die Menschen im System erreicht werden kann.

Sabine ist regelmäßig als Sprecherin auf internationalen Konferenzen anzutreffen. Als Speaker Coach unterstützt sie Menschen dabei, mehr Sichtbarkeit zu erlangen.

OAuth2 Authentication für GraphQL mit Google Firebase, Spring und React

Heinz-Werner Haas & François Fernandes, Digital Frontiers GmbH & Co. KG

Kaum eine Anwendung kommt ohne Authentifizierung aus. Dennoch gehört die Authentifizierung von Benutzern nicht zu den Lieblingsaufgaben der meisten Entwickler. Libraries sind nicht selten kompliziert, der Aufwand für eine entsprechende Infrastruktur hoch. Es geht jedoch auch deutlich einfacher. Mit der OAuth2-Unterstützung von Spring ist der Einstieg bei der Implementation einfach. Kombiniert mit Google Firebase erhält man zudem einen leicht zugänglichen Authentifizierungsservice, der die Benutzerverwaltung übernimmt. Schauen wir uns daher in diesem Artikel die Funktionsweise von OAuth2 an und wie diese in eine GraphQL-Kommunikation integriert werden kann.

In Ausgabe 05/2022 der Java aktuell erschien der Artikel mit dem Titel „GraphQL Subscriptions mit Spring Boot 2.7, Project Reactor und React", dessen Beispielapplikation auch für diesen Artikel als Grundlage dient. Bei dieser handelt es sich um einen exemplarischen Kurznachrichtendienst mit dem Namen Quacker. Das Besondere an diesem Dienst ist, dass Posts nahezu in Echtzeit für alle Anwender sichtbar sind. Gelöst ist das mittels GraphQL Subscriptions. Das Projekt selbst ist auf GitHub *[1]* zu finden.

Tooling mit Feuer: Google Firebase

Wie bereits einleitend erwähnt, soll Google Firebase für die Authentifizierung von Benutzern verwendet werden. Firebase ist aber nicht nur ein Authentifizierungs-Service, sondern eine Komplettlösung für eine Serverless-Infrastruktur, die aus vielen verschiedenen Bestandteilen besteht, wie beispielsweise NoSQL-Datenbank, Realtime-Datenbank, Hosting, Functions as a Service, Machine Learning, Analytics und mehr. Ein Werkzeugkasten, dessen Bestandteile je nach Bedarf eingesetzt werden können und die auch nur nach tatsächlicher Nutzung bezahlt werden müssen.

Zu diesem Werkzeugkasten gehört auch Authentication, worauf wir uns im Folgenden konzentrieren werden. Spannend ist dabei das Preismodell von Firebase, da es sehr großzügige kostenlose Kontingente hat. Für die Authentifizierung beispielsweise liegt das Limit bei 50.000 aktiven Nutzern pro Monat.

Firebase Authentication

Bei Firebase Authentication handelt es sich nicht nur um einen Authentication Service, sondern um eine vollständige Benutzerverwaltung. Damit ist es möglich, unterschiedlichste Methoden für eine Anmeldung anzubieten, wie beispielsweise Social Logins (Google, Facebook, Apple und mehr), einem klassischen Login mit E-Mail-Adresse und Passwort oder gar einer Authentifizierung per Telefon. Meldet sich ein Benutzer an (unabhängig von der Methode der Anmeldung), wird dieser auch automatisch in die Benutzerdatenbank aufgenommen. Damit können Informationen zu dem jeweiligen Benutzer mithilfe des Firebase Admin SDK serverseitig in der Applikation angefragt werden – sollte man dies benötigen. In unserem Beispiel ist das nicht notwendig, da der gültige Login für die Anwendung völlig ausreicht.

Der eigentliche Login-Vorgang ist für die Anwendung weitestgehend transparent. Der Firebase SDK bietet für die jeweilige Login-Methode passende APIs, die den Login-Vorgang initiieren und abschließen. In der Anwendung erhält man dann lediglich einen authentifizierten Benutzer oder eine Fehlermeldung, wenn der Login nicht erfolgreich war. Firebase verwendet im Hintergrund OAuth2 für die Authentifizierung und stellt ein entsprechendes OAuth-Token für die Kommunikation mit dem Server bereit. Damit stellt sich aber auch die folgende Frage:

Was ist eigentlich OAuth2?

OAuth2 ist ein offener Standard, der mittlerweile eine breite Anwendung in unterschiedlichsten Systemen findet. Es definiert eine Reihe von Komponenten und deren Interaktion untereinander, um eine zuverlässige Authentifizierung zu realisieren.

Die Spezifikation von OAuth2 sieht für den Ablauf der Authentifizierung verschiedene Akteure mit definierten Rollen vor, die in *Listing 1* zu sehen sind.

Akteur	OAuth2-Rolle
Benutzer mit Quaker Client im Browser	Resource Owner
Browser	Client
Quaker	Resource Server
Authentifizierungsservice	Authorization Server

Listing 1: Tabelle der notwendigen Akteure des OAuth-Flows

Die beschriebenen Akteure sind dabei ein wichtiger Bestandteil des Authentifizierungs-Flows, der in *Abbildung 1* schematisch dargestellt ist.

Das Ziel ist es, ein Token zu erhalten, das einen Benutzer eindeutig identifiziert und damit authentifiziert. Hierbei findet ein JSON Web Token (JWT) Anwendung. Wer gerne mehr über JWT erfahren möchte, wird unter *[6]* fündig werden.

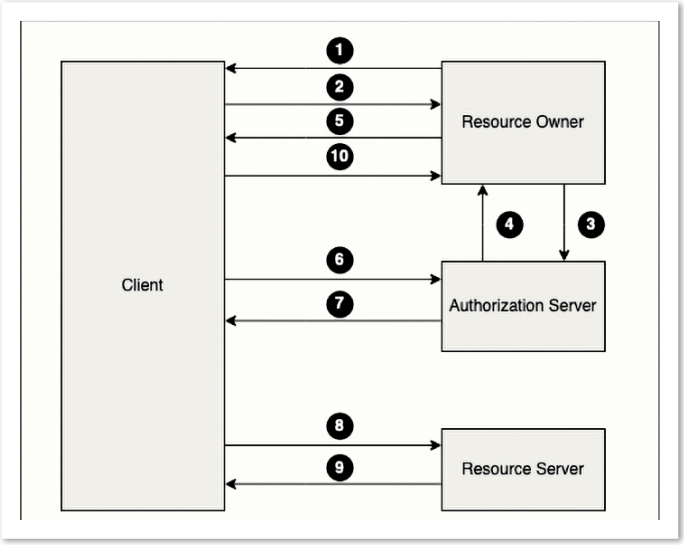


Abbildung 1: Beispielhafter Ablauf des OAuth2-Flows

Eine Authentifizierung beginnt immer mit dem eigentlichen Aufruf der Anwendung selbst *(Pfeil 1)*. Hierbei stellt der Client fest, dass dieser noch nicht authentifiziert ist und leitet den Anwender dann zur Authentifizierung weiter *(Pfeil 2)*.

Nicht authentifiziert bedeutet hierbei, dass der Browser (bedient vom Resource Owner) beim Client nicht eingeloggt ist und dadurch noch nicht dazu berechtigt ist, auf die gewünschten Nachrichten zuzugreifen. Im Sprachgebrauch von OAuth bedeutet dies, dass ein OAuth Grant durchgeführt werden muss.

Dafür leitet der Client den Browser an den Authorization Server weiter *(Pfeil 3)*.

Anschließend wird ein Login beim Authentifizierungsserver durchgeführt, um die Identität des Nutzers festzustellen. Wie die Authentifizierung durchgeführt wird, liegt dabei in der Verantwortung des Anbieters (beispielsweise mit Benutzername und Passwort). Ist diese erfolgreich, wird der OAuth Grant fortgesetzt und der User-Agent mit einem Code zurück zum Client geleitet *(Pfeil 4 und 5)*.

Bei diesem Code handelt es sich um einen sogenannten Authorization Code, der ein einmalig verwendbares Token ist. Dieses Token ist an die Client ID gebunden und dient als Vorautorisierung.

Nach Erhalt dieses Codes fordert der Client im Tausch gegen diesen ein Access Token vom Authorization Server an *(Pfeil 6)*.

Dabei werden im Body dieser Anfrage der Authorization Code, die zuvor genutzte Redirect URI und der gewählte Grant mitgeschickt. Sind all diese Credentials des Clients korrekt, wird dieser gegen das dauerhaftere Access Token eingetauscht.

Mit einem durch den Authorization Server ausgestellten Access Token kann der Client auf Ressourcen im Namen des Anwenders zugreifen *(Pfeil 7)*. Nun fehlt noch die Validierung der Anfrage selbst, was die Aufgabe des Resource Server ist. Dieser prüft beispielsweise die Signatur des Tokens auf Gültigkeit und antwortet bei erfolgreicher Validierung mit den gewünschten Informationen *(Pfeil 9 und 10)*.

Authentifizierung für GraphQL

Wie der vorhergehende Abschnitt zu OAuth2 bereits gezeigt hat, wird die Authentifizierung durch den Client getrieben. Der Backend Server hat letztlich nur die Aufgabe, das mittels HTTP Header übertragene Token auf Gültigkeit hin zu überprüfen.

Hierbei ist zu bedenken, dass die ausgestellten Tokens eine kurze Gültigkeitsdauer haben. Im Falle von Firebase sind Tokens eine Stunde gültig und müssen dann erneuert werden. Praktischerweise wird dies im Fall von Firebase transparent durch das API erledigt.

Zusätzlich gilt es zu beachten, dass GraphQL-Anwendungen mit Subscriptions typischerweise sowohl HTTP Request und Response als auch WebSockets für die Kommunikation verwenden. HTTP Requests kommen dabei für Query und Mutation zum Einsatz, Subscriptions werden über WebSockets abgehandelt.

Bei HTTP Requests wird hierbei das Token mithilfe des Authorization-Header im Request übertragen. Für WebSockets ist es leider jedoch nicht ganz so einfach, da das clientseitige API für den Aufbau einer WebSocket-Verbindung nicht erlaubt, HTTP Header zu setzen. In diesem Fall bedient man sich typischerweise des Protokolls für graphql-ws über WebSockets, das einen initialen Handshake mit dem Server durchführt und dabei eine connection_init-Message verschickt. Der Inhalt dieser Message kann frei vergeben werden, was wir im Folgenden für die Übertragung des Tokens nutzen werden.

React-Client-Implementation

Wie bereits im Artikel aus Ausgabe 05/2022 der Java aktuell beschrieben, verwenden wir React für die Implementation des Clients, mit Apollo als GraphQL Client *[7]*.

Um Firebase für die Authentifizierung zu verwenden, muss ein passendes Projekt über die Firebase-Konsole eingerichtet werden. Dies ist ausführlich unter *[5]* dokumentiert.

Mit der erfolgreichen Einrichtung des Projekts fehlt dann lediglich die Aktivierung von Firebase Authentication und der Konfiguration von Authentication-Providern. In unserer Beispielapplikation haben wir uns für den Social Login mit Google und eine Authentifizierung mittels E-Mail und Passwort entschieden.

Praktischerweise gibt es für die Durchführung des Logins fertige React Hooks im Package *react-firebase-hooks [3]*, die wie in *Listing 2* zu sehen angewendet werden.

Diese Hooks führen den Authentifizierungs-Flow transparent durch und liefern letztlich bei erfolgreicher Authentifizierung ein User-Objekt, das das Token enthält.

Wie bereits erwähnt, muss das Token bei HTTP Requests mit übertragen werden. Hierfür wird der Apollo GraphQL Client über setContext angepasst, sodass bei jedem Request das jeweils aktuelle Token übertragen wird *(Listing 3)*.

```
const [signIngWithGoogle] = useSignInWithGoogle(firebaseAuth);
  const [signInWithEmailAndPassword, user, loading, error] =
     useSignInWithEmailAndPassword(firebaseAuth);
```

Listing 2: React Hooks für die Firebase-Authentifizierung

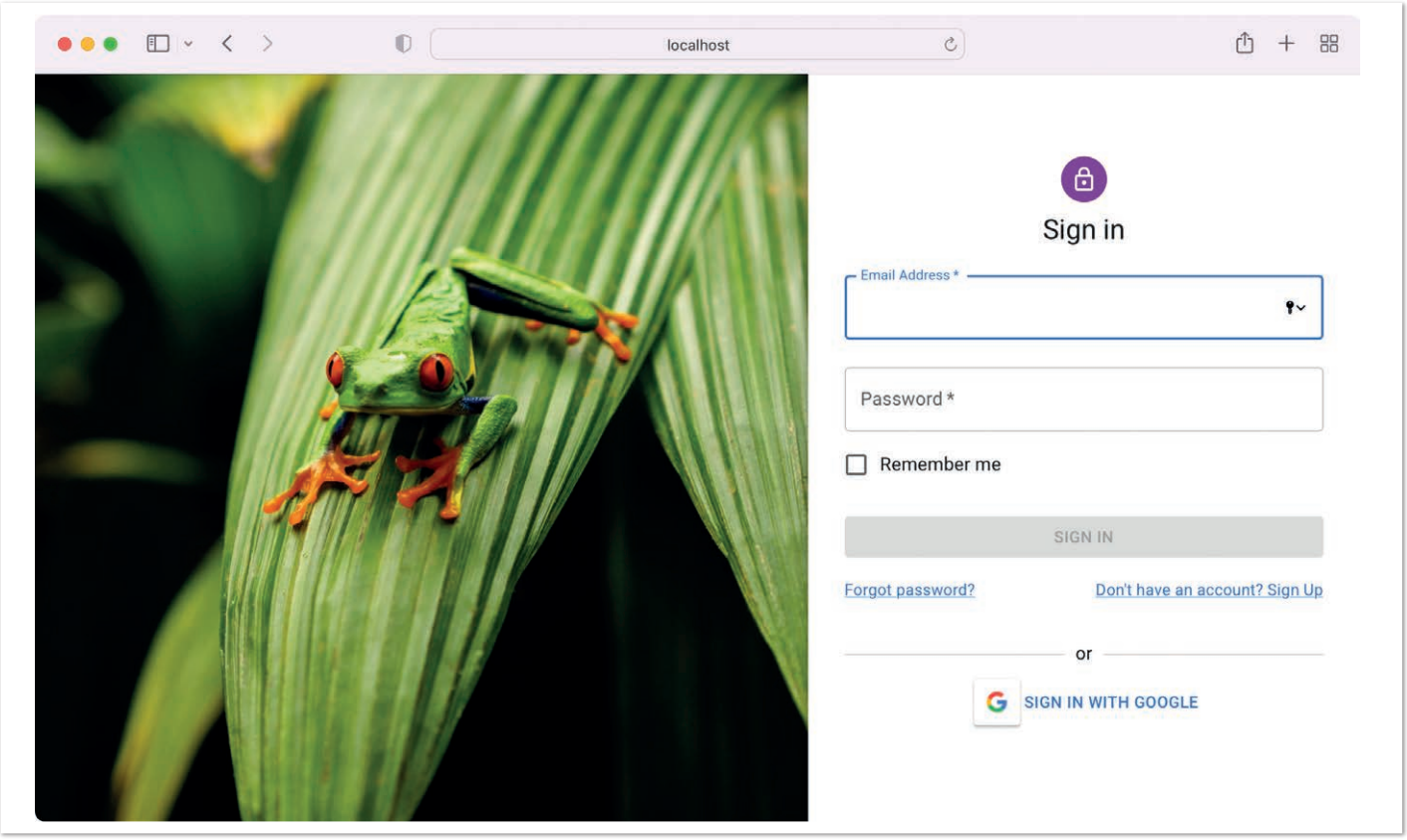


Abbildung 2: Screenshot der Login-Seite von Quacker

Wie schon in Kapitel *Authentifizierung für GraphQL* erwähnt, wird nicht nur HTTP für die Kommunikation verwendet, sondern auch WebSockets für Subscriptions.

Hierbei muss das Token in der connection_init-Message enthalten sein, was sich bei der Erzeugung des graphql-ws-Clients problemlos angeben lässt. Allerdings wird eine WebSocket-Verbindung unter Umständen über die Gültigkeitsdauer des Tokens hinaus aufrechterhalten, was eine erneute Übertragung des Tokens notwendig macht.

Listing 4 zeigt eine Methode, wie das Token transparent für die Anwendung aktualisiert werden kann. Der Trick hierbei ist, dass beim Öffnen der Verbindung die verbleibende Gültigkeitsdauer des Tokens errechnet wird. Mit einem Timeout, der zwei Minuten vor dem Ablauf des Tokens zuschlägt, wird die WebSocket-Verbindung geschlossen, was den Client zu einem erneuten Verbindungsaufbau zwingt.

Der User bleibt somit weiterhin authentifiziert und muss sich nicht nach jeder Stunde aktiv neu einloggen.

Dadurch sind sowohl Anfragen mittels HTTP als auch über WebSockets mit einem geeigneten Authentifizierungstoken versehen und werden an den Server übertragen. Die Fragmente aus *Listing 3* und 4 sind hierbei nur ein Ausschnitt der Client-Konfiguration, die vollständige Implementierung ist unter *[8]* zu finden.

Abschließend fehlt nur noch die Validierung des Tokens durch den Server, damit alle Funktionalitäten der Anwendung genutzt werden können.

```
const authLink = setContext(async (_, {headers}) => {
  const token = await user.getIdToken()
  return {
     headers: {
        ...headers,
        authorization: token ? `Bearer ${token}` : "",
     }
  }
});
```

Listing 3: Implementierung für das Mitschicken des Tokens an den Server

Backend-Implementation

Bei der Umsetzung des Servers unterstützt Spring Security mit einer Standard-Integration für OAuth2, die das Token ausliest und validiert. Dies funktioniert für Anfragen per HTTP mit den richtigen Abhängigkeiten und ein wenig Konfiguration problemlos, lediglich bei Subscriptions wird es etwas aufwendiger – dazu jedoch später mehr.

Beginnen wir mit dem Hinzufügen der notwendigen Abhängigkeiten: `spring-boot-starter-security` und `spring-boot-starter-oauth2-resource-server`.

Durch die Kombination von Spring Security mit der OAuth2 Resource Server Integration ist es ein Leichtes, die Authentifizierung umzusetzen. Hierbei muss lediglich eine Zeile der application.properties (alternativ application.yml) hinzugefügt werden, wie in *Listing 5* zu sehen.

Der Platzhalter `<PROJECT-ID>` muss dabei durch die jeweilige ID des Firebase-Projekts ersetzt werden, die sich zum Beispiel in der Übersicht der Projekte befindet.

```
const graphQLWsClient = createClient({
  url: "ws://localhost:8080/graphql",
  connectionParams: async () => {
     const result = await user.getIdTokenResult()
     expirationTime = new Date(result.expirationTime).getTime()
     return {token: result.token};
  },
  on: {
     connected: (socket: any) => {
        const refreshIn = expirationTime - 12000 - Date.now()

        activeTimeout = setTimeout(
              () => {
                 if (socket.readyState === WebSocket.OPEN)
                    socket.close(CloseCode.Forbidden, 'Forbidden');
              },
              refreshIn
        )
     },
     closed: () => {
        if (activeTimeout) {
           clearTimeout(activeTimeout)
           activeTimeout = undefined
        }
     }
  },
})
```

Listing 4: Mechanismus zum Wiederaufbau des WebSocket

```
spring.security.oauth2.resourceserver.jwt.issuer-uri=https://securetoken.google.com/<PROJECT-ID>
```

Listing 5: Konfiguration der Issuer URI in der application.properties

```
@EnableWebFluxSecurity
class SecurityConfiguration {
  @Bean
  fun springSecurityFilterChain(http: ServerHttpSecurity) =
     http {
        authorizeExchange {
           authorize("/graphql", permitAll)
           authorize(anyExchange, authenticated)
        }
        oauth2ResourceServer {
           jwt { }
        }
        httpBasic { disable() }
        formLogin { disable() }
     }
}
```

Listing 6: Spring-Security-Konfiguration

Basierend auf dieser URI konfiguriert Spring Security die Prüfung der ausgestellten Tokens. Über die Issuer URI, in Kombination mit „Well Known URIs", erhält Spring Security alle für eine Validierung notwendigen Informationen.

Damit fehlt nur noch eine Spring-Security-Konfiguration wie in *Listing 6* zu sehen.

Dem Spring-Security-erfahrenen Leser wird hierbei sicher auffallen, dass der `/graphql`-Endpunkt alle Anfragen auch ohne Authentifizierung erlaubt. Es ist nicht unüblich, dass Teile eines GraphQL-API auch ohne Authentifizierung erreichbar sein sollen, um beispielsweise eine öffentliche Startseite anbieten zu können. Ob eine Authentifizierung notwendig ist, kann dann mit `@PreAuthorize` feingranularer pro Controller-Methode entschieden werden, wie in *Listing 7* anhand einer Mutation exemplarisch zu sehen ist.

```
@MutationMapping
@PreAuthorize("isAuthenticated()")
fun createPost(@Argument message: String, principal:
Principal): Post =
  postService.createPost(
     userService.getUser(principal),
     message
  )
```

Listing 7: Spring-Security-Konfiguration

GraphQL Subscriptions

So einfach die Absicherung eines Controllers für Query und Mutations mittels Spring Security ist, so kompliziert wird es leider für Controller, die eine Subscription über WebSockets implementieren. Das Token wird als connection_init-Parameter während der Handshake-Phase übergeben.

Zum Zeitpunkt der Drucklegung dieses Artikels gibt es noch keine native Integration zwischen GraphQL Subscriptions und Spring Security, die diese Art der Authentifizierung ermöglicht. (Details hierzu finden sich im Ticket des Spring GraphQL Projekts unter *[4]*.)

Hier spielt uns die durchweg modulare Struktur von Spring und Spring Boot in die Hände und ermöglicht es, die Logik für die Authentifizierung selbst umzusetzen. Die zentrale Komponente bei der Authentifizierung mit Spring Security ist dabei das Interface ReactiveAuthenticationManager. Die für OAuth2 relevante Implementation ist der JwtReactiveAuthenticationManager.

Um das Token auszulesen, muss in die WebSocket-Message-Verarbeitung eingegriffen werden, was mit der Implementation eines WebSocketGraphQlInterceptor möglich ist.

Dieser hat zwei Methoden:

- handleConnectionInitialization: Diese Methode wird für jeden neuen Handshake ausgeführt.

```
override fun handleConnectionInitialization(
  sessionInfo: WebSocketSessionInfo,
  connectionInitPayload: MutableMap<String, Any>,
): Mono<Any> {

  val token = connectionInitPayload["token"] as? String
  if (token != null) {
     sessionInfo.setAuthentication(
        BearerTokenAuthenticationToken(token))
  }

  return Mono.empty()
}
```

Listing 8: Behandlung der connection_init-Message

- intercept: Jede neue Nachricht an den Server (etwa, um eine Subscription auszuführen) wird über diese Methode geleitet. Damit ist dies auch der perfekte Zeitpunkt, um eine Authentifizierung zu prüfen und, im Falle einer erfolgreichen Authentifizierung, diese Information in einem Context zu hinterlegen.

Jede Verbindung startet natürlich mit dem Handshake, der die erwähnte connection_init-Message enthält und in der auch das übertragene Token zu finden ist. Der Zugriff ist durch das Implementieren der Methode handleConnectionInitialization möglich, die den connectionInitPayload (und damit das Token) erhält und auch Zugriff auf die sessionInfo hat, die mit dieser WebSocket-Verbindung assoziiert ist.

Das Token wird aus dem connectionIniPayload ausgelesen und in der sessionInfo als BearerTokenAuthenticationToken abgelegt. Der Aufruf sessionInfo.setAuthentication() ist dabei eine Extension Function, die lediglich die Lesbarkeit erhöht. Dahinter verbirgt sich das setzen des Tokens als Wert in einer Map.

Mit dem extrahierten Token kann nun eine Authentifizierung durchgeführt werden, die bei jeder eingehenden Nachricht erneut geprüft werden muss, um sicherzustellen, dass das Token noch nicht abgelaufen ist. Das wird in der Implementation der Methode intercept realisiert, wie in *Listing 9* zu sehen.

```
override fun intercept(
  request: WebGraphQlRequest, chain: Chain
): Mono<WebGraphQlResponse> {

  if (request !is WebSocketGraphQlRequest) {
     return chain.next(request)
  }

  val securityContext = Mono.just(request)
     .ofType<WebSocketGraphQlRequest>()
     .mapNotNull { it.sessionInfo.getAuthentication() }
     .flatMap { authenticationManager.authenticate(it) }
     .map { SecurityContextImpl(it) }

  return chain.next(request)
     .contextWrite(ReactiveSecurityContextHolder
        .withSecurityContext(securityContext))
}
```

Listing 9: Durchführung der Authentifizierung basierend auf dem Token

Die eigentliche Authentifizierung besteht dabei aus zwei Schritten. Aus dem Auslesen des Tokens aus der sessionInfo des Request und dem Authentifizieren durch einen ReactiveAuthenticationManager. In unserem Fall wird dabei das bereits erwähnte BearerTokenAuthenticationToken vorausgesetzt. Als Resultat wird ein SecurityContext erzeugt. Dieser SecurityContext wird der reactive chain als Context (mittels contextWrite) gesetzt, womit alle weiteren Aktionen in der reactive chain einen entsprechenden SecurityContext erhalten und die Spring-Security-Mechanismen wie @PreAuthorize oder das Injizieren eines Principal funktionieren.

Nun bleibt nur noch das korrekte Setzen des ReactiveAuthenticationManager. Auch dies gestaltet sich sehr einfach, wie in *Listing 10* zu sehen.

```
@Bean
fun graphqlWsInterceptor(
  @Value("\${spring.security.oauth2.resourceserver.
jwt.issuer-uri}")
  issuerUri: String
) = WebSocketAuthenticationInterceptor(
  JwtReactiveAuthenticationManager(ReactiveJwtDecode
rs.fromIssuerLocation(issuerUri))
)
```

Listing 10: Erzeugung des ReactiveAuthenticationManager

Es wird ein JwtReactiveAuthenticationManager erzeugt, der als Konfiguration die aus *Listing 5* bekannte Issuer URI enthält. ReactiveJwtDecoders.fromIsserLocation() verwendet dabei wieder die bereits erwähnten „Well known URIs", um die passende Konfiguration zu erzeugen.

Fazit

Authentifizierung kann mit den richtigen Mitteln einfach, schnell und zuverlässig umgesetzt werden. Gerade die Kombination einer React-Applikation, kombiniert mit Firebase und einem Spring Boot Backend, bietet hier einen einfachen Einstieg und viel Flexibilität. Auch GraphQL Subscriptions lassen sich mit geringem Aufwand absichern.

Quellen

[1] *https://github.com/dxfrontiers/graphql-subscriptions*
[2] *https://console.firebase.google.com/*
[3] *https://github.com/csfrequency/react-firebase-hooks*
[4] *https://github.com/spring-projects/spring-graphql/issues/268*
[5] *https://firebase.google.com/docs/web/setup*
[6] *https://jwt.io/*
[7] *https://www.apollographql.com/apollo-client*
[8] *https://github.com/dxfrontiers/graphql-subscriptions/blob/master/frontend/src/graphql/client.ts*

Heinz-Werner Haas
Digital Frontiers GmbH & Co. KG
heinz.haas@digitalfrontiers.de

Heinz-Werner Haas beschäftigt sich als Consultant bei Digital Frontiers mit agiler Softwareentwicklung vorwiegend im TypeScript-Umfeld. Neben seinen Tätigkeiten als Entwickler berät er seine Kunden im Bereich IT-Security, analysiert ihre Software auf potenzielle Schwachstellen und hilft ihnen, das Bewusstsein für das Themenfeld Sicherheit zu erhöhen.

François Fernandes
Digital Frontiers GmbH & Co. KG
francois.fernandes@digitalfrontiers.de

François Fernandès ist Senior Solution Architect bei Digital Frontiers. Er verfügt über langjährige Erfahrung in den Bereichen Software-Architektur, Cloud-Architektur und dem Java-Ökosystem. Spring und Spring Boot sind für ihn seit Langem treue Begleiter.

Mitglieder des iJUG

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

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: Björn Bröhl. 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: Fried Saacke
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

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

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

Bildnachweis:
Titel: Bild © brgfx
https://freepik.com
S. 10 + 11: Bild © auntspray
https://123rf.com
S. 15: Bild © Irina Strelnikova
https://stock.adobe.com
S. 20 + 21: Bild © Man As Thep
https://stock.adobe.com
S. 30 + 31: Bild © eireenz
https://stock.adobe.com
S. 37: Bild © mohamed_hassan
https://pixabaycom
S. 38: Bild © rawpixel
https://123rf.com

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

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

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

Inserentenverzeichnis

JavaLand!

on demand

Javaland 2022 verpasst?

Jetzt On-demand-Ticket buchen und Vortragsaufzeichnungen anschauen!

Alle Angebote im On-demand-Ticket-Shop

Community-Partner: iJUG Verbund

Präsentiert von: DOAG Heise Medien