Ausgabe 06/2018

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 06

Java aktuell

iJUG Verbund
www.ijug.eu

Tipps von Experten
Der richtige Umgang mit Zeitzonen

Im Interview
Mike Milinkovich, Executive Director Eclipse Foundation

Praktische Erfahrungen
Security innerhalb der agilen Software-Entwicklung

Die Reise nach Jakarta

Data Analytics 2019

Die Datenexplosion meistern

26. & 27. März | Phantasialand bei Köln

analytics.doag.org

ORACLE Cloud

DOAG

Mitten in der Java-Community

Ort des Geschehens: In der Stuttgarter Liederhalle findet am 5. Juli 2018 das Java Forum Stuttgart statt. Die rund 2.000 Teilnehmer erhalten fast fünfzig Vorträge in sieben parallelen Streams; eine gute Möglichkeit, sich umfassend über Themen zu Java sowie im Java- und JVM-Umfeld zu informieren. Die Veranstaltung ist auch für mich eine gute Gelegenheit, in meine alte Heimat zu reisen und meine Eltern zu besuchen.

Gleich zu Beginn verteile ich zusammen mit dem iJUG-Vorstand Stefan Koospal und Jürgen Thierack von der Java User Group München die aktuelle Ausgabe der Java aktuell an die eintreffenden Besucher. Die ganze Palette mit den Zeitschriften ist im Nu verteilt. Ich freue mich jedes Mal darauf, meinen Leserinnen und Lesern in die Augen schauen zu können. Es macht mich glücklich zu sehen, für wen ich meine Arbeit mache und wie gerne die Zeitschrift angenommen wird. Das motiviert mich und gibt mir die Energie für die kommenden Ausgaben.

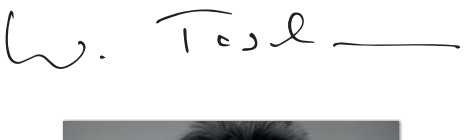
Später am iJUG-Stand treffe ich Markus Karg von der Java User Group Goldstadt, mit dem mich neben der Arbeit eine Freundschaft verbindet. Ich schätze seine offene Art und sein konstruktiv kritisches Auftreten sehr. Wir reden auch über die Kooperation des iJUG mit der Eclipse Foundation (danke an Jan Westerkamp von der Java User Group Darmstadt für sein Engagement) und sind beide gespannt, welchen Weg das ehemalige Java EE als Jakarta EE nehmen wird. Ich habe dazu auch Mike Milinkovich, Executive Director der Eclipse Foundation, für diese Ausgabe interviewt und Markus hat mir dazu auf Seite 9 eine offene und ehrliche Stellungnahme verfasst.

Die Veranstaltung ist auch eine willkommene Gelegenheit für mich, Autoren der Java aktuell zu treffen und neue zu gewinnen. Das Ergebnis finden Sie in dieser Ausgabe. Fast alle Themen wurden vor Ort als Vortrag gehalten.

Das Konzept des Java Forum Stuttgart erweist sich als erfolgreich. Das Treffen ist als Low-Budget-Veranstaltung für alle Beteiligten ausgelegt. Die Einnahmen durch Sponsoren, Aussteller und Sponsored Talks werden vor allem zur Erzielung einer attraktiven Besuchergebühr eingesetzt. Die Organisation erfolgt durch die Java User Group Stuttgart, unterstützt durch viele ehrenamtliche Helfer. Weitere solche Veranstaltungen erfolgen im Lauf des Jahres: der JUG Saxony Day, das Java Forum Nord oder die Berlin Expert Days, um nur einige zu nennen.

Ich wünsche Ihnen viele neue Erkenntnisse beim Besuch eines der von der Community veranstalteten Events sowie beim Lesen der Java aktuell.

Ihr

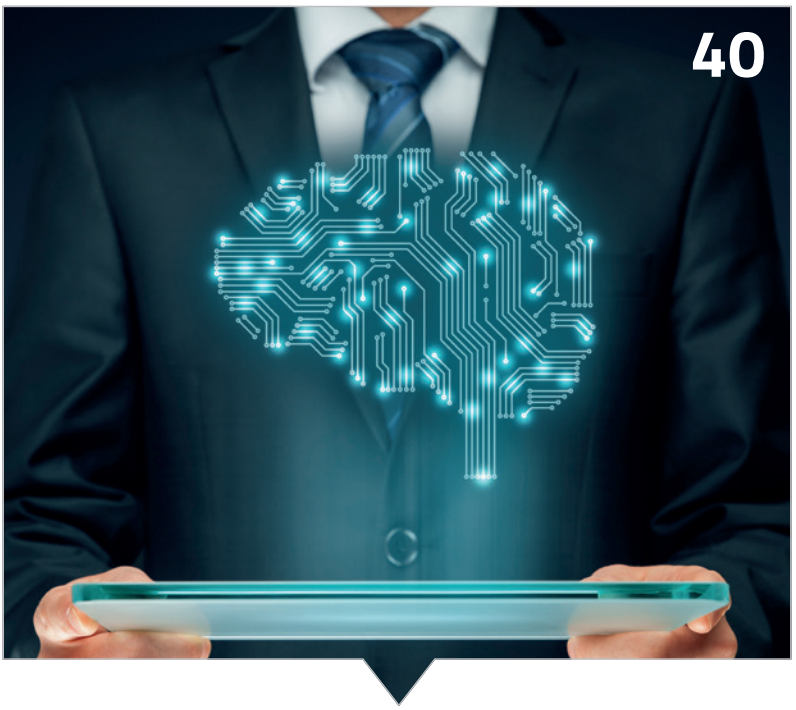


Wolfgang Taschner
Chefredakteur Java aktuell

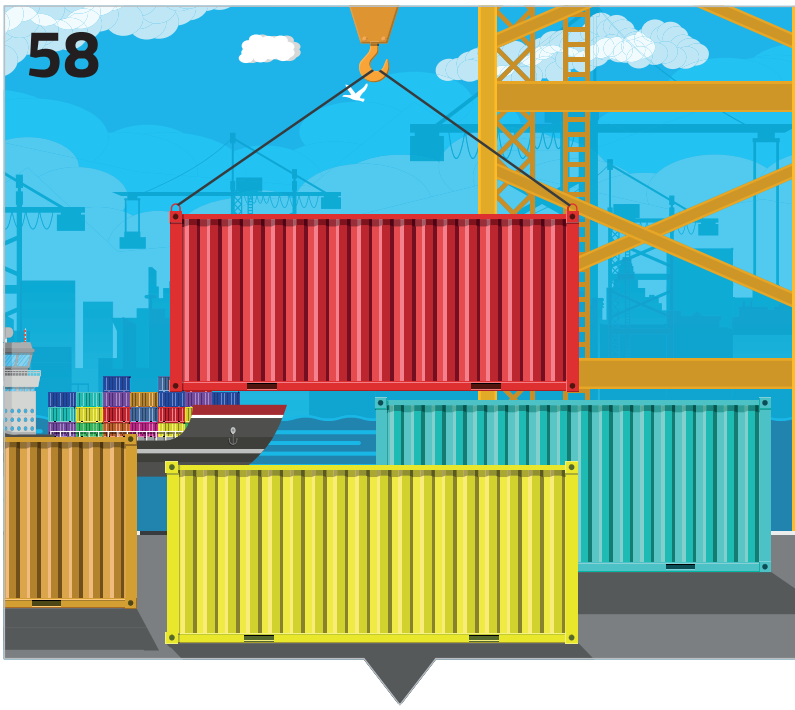
Inhalt

Aus der Vergangenheit lernen, von der Zukunft träumen

Zeitzonen in Computersystemen abbilden



Machine-Learning- und Analytics-Projekte in Unternehmen



Container erfreuen sich immer größerer Beliebtheit

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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

12. Juni 2018

Prinzipien und Entscheidungsfindung für Jakarta EE

Wer sich mal genauer über die Organisations- und Entscheidungsstrukturen von EE4J und Jakarta EE bei Eclipse informieren möchte und darüber, wie das Ganze beispielsweise mit MicroProfile zusammenhängt – denen sei ein (nicht zu langer) Artikel von Steve Millidge von Payara ans Herz gelegt. Dazu auch das Interview mit Mike Milinkovich auf Seite 8 in dieser Ausgabe.

tinyurl.com/y7wth679.

20. Juni 2018

MicroProfile 1.4 und 2.0

Jetzt ist es soweit: MicroProfile 1.4 und 2.0 sind freigegeben. 1.4 enthält Updates zu folgenden APIs: Config, Fault Tolerance, JWT Authentication, Open Tracing und REST Client. Release 2.0 basiert bei allen Java-EE-Dependencies (unter anderem CDI und JAX-RS) auf den EE-8-Versionen und ist ansonsten deckungsgleich mit 1.4. Nur auf die Implementierungen werden wir noch ein wenig warten müssen. Eine Referenz-Implementierung gibt es bei „MP" ja nicht, und keines der beteiligten Unternehmen hat bislang Unterstützung jenseits von MicroProfile 1.3 konkret angekündigt.

http://microprofile.io/blog/2018/06/eclipse-microprofile-1.4-2.0-available

20. Juni 2018

JDK 11 startet erste Rampdown-Phase

Das OpenJDK 11-Projekt hat wie geplant die „Rampdown"-Phase 1 begonnen, gleichbedeutend mit einem „Feature Freeze". Jetzt sind nur noch Bug Fixes möglich, in der in einem Monat beginnenden Phase 2 nur noch Fixes für kritische Bugs. Pünktlich im September können wir dann das neue OpenJDK-Release erwarten, unter anderem mit dem Flight Recorder (bislang nur im Oracle-JDK verfügbar) und dem No-Op Garbage Collector Epsilon, der Speicher nicht wieder freigibt (sobald dieser erschöpft ist, fährt die JVM herunter). Aber auch die Entfernung der in JDK 9 als „deprecated" markierten Java-EE- und Corba-Modulen gehört zu den „Features".

http://openjdk.java.net/projects/jdk/11

25. Juni 2018

Java-EE-Spezifikationen: Übergabe an Eclipse fraglich

Josh Juneau hat in der Jakarta-EE-Mailingliste eine Diskussion um die Spezifikations-Dokumente losgetreten. Diese sind im Gegensatz zum Source Code bislang nicht übergeben worden und es besteht die konkrete Möglichkeit, dass die Übergabe aufgrund von Intellectual-Property-Problemen zumindest teilweise scheitert. Ein Jakarta EE ohne Spezifikation wäre etwas dürftig – ein Neuschreiben im Sinne von Abschreiben könnte sich aus juristischen Gründen auch als schwierig erweisen. Das Problem ist bekannt, hat aber neben anderen, zeitkritischeren juristischen Diskussionen nicht die nötige Priorität. Auch die Zahl der Silicon-Valley-Anwälte scheint endlich zu sein...

https://accounts.eclipse.org/mailing-list/jakarta.ee-community

11. Juli 2018

Docker-Images mit Google Jib erstellen

Google hat ein neues Open-Source-Projekt veröffentlicht, das Java-Entwicklern das Erstellen von Docker-Images erleichtern soll: Jib ist ein Plug-in für die Build-Werkzeuge Maven und Gradle. Es übernimmt das Bauen von Images und Pushen in eine Docker-Registry direkt aus dem Java-Build heraus, ohne ein Dockerfile oder auch nur eine Docker-Installation zu erfordern.

https://cloudplatform.googleblog.com/2018/07/introducing-jib-build-java-docker-images-better.html und https://github.com/GoogleContainerTools/jib

11. Juli 2018

WebSphere Liberty 18.0.0.2 unterstützt EE 8

IBMs WebSphere Liberty ist in der Version 18.0.0.2 freigegeben worden – der erste Application-Server (abgesehen von der Referenz-Implementierung GlassFish), der die volle Java-EE-8-Zertifizierung hat und unter anderem auch MicroProfile 1.3 abdeckt.

https://developer.ibm.com/wasdev/blog/2018/06/29/java-ee-8-liberty-18-0-0-2

23. Juli 2018

Oracle: Graal statt Nashorn

Oracle hat einen Migrationspfad für Nutzer der in der JVM enthaltenen JavaScript-Engine Nashorn vorgestellt: Die neue, polyglotte GraalVM, die unter anderem innerhalb des OpenJDK laufen kann, soll als Ersatz dienen. Sie bietet laut Oracle auch eine weitaus vollständigere Unterstützung der neuesten JavaScript-Standards. Da die APIs für die Interaktionen zwischen Java und JavaScript-Code sich unterscheiden, haben die Hüter der GraalVM das Kommandozeilen-Flag „nashorn-compat" eingeführt. Damit ist in der GraalVM ein weitgehend Nashorn-kompatibles API verfügbar. Dazu gibt es eine Migrations-Anleitung, die auch die weiteren noch nötigen Schritte beschreibt. Die GraalVM steckt noch in den Kinderschuhen, also ist vor einer Migration erst einmal gesunde Skepsis angesagt. Aber eine Weile wird uns das alte Nashorn ja auch noch begleiten – mit JDK 11 ist es zunächst nur „deprecated".

https://github.com/graalvm/graaljs/tree/master/docs/user

30. Juli 2018

NetBeans 9

Nach drei Jahren ist das nächste große Release der NetBeans-IDE erschienen. Neben Kleinigkeiten wie dem neuen Modulsystem in Java 9 hat wohl die Übergabe der Software von Oracle an die Apache Software Foundation (ASF) für deutliche Verzögerungen gesorgt. NetBeans befindet sich bei der ASF immer noch im Inkubator. Aber jetzt ist Release 9 da, mit (Projekt-)Support für das Modulsystem, „jshell"-Integration und anderen Java-9-spezifischen Features sowie Unterstützung für „Local Variable Type Inference" aus Java 10. Bislang beschränkt sich das Release auf Java SE. Die Module für Java EE, JavaScript, PHP etc. lassen noch etwas auf sich warten, für C und C++ ist noch nicht einmal der Übergabeprozess an die ASF abgeschlossen.

http://wiki.netbeans.org/NetBeans_9

31. Juli 2018

Jakarta EE Committee Election Results

Die Wahlen zu den Jakarta EE Committees sind abgeschlossen – alle gewählten Repräsentanten sind bekannte Namen in der europäischen Java-Community: Alex Theedom und Werner Keil (Specification Committee), Simon Maple und Ivar Grimstad (Marketing und Brand) sowie Martijn Verburg und nochmals Ivar Grimstad (Steering).

https://www.eclipse.org/org/press-release/20180731-jakartaEE_committee_election.php

17. August 2018

EclipseCon Europe im Oktober mit Community Day

Die Eclipse Foundation plant am Montag, 22. Oktober 2018, einen Community Day vor der eigentlichen EclipseCon Europe Konferenz in Ludwigsburg. Zu verschiedenen Themen soll es dort von der Community selbst organisierte Treffen beziehungsweise Workshops geben, etwa zu Jakarta EE und MicroProfile.

https://www.eclipsecon.org/europe2018

25. August 2018

MicroProfile plant nächste Releases

Die nächsten MicroProfile-Releases sind anvisiert, das Modewort „Reactive" hält auch hier Einzug: Release 2.1 soll am 8. Oktober 2018 erscheinen, rechtzeitig vor den wichtigen Herbstkonferenzen, und unter anderem eine „Reactive Streams Operator"-Spezifikation enthalten, ähnlich wie beispielsweise RxJava. Diese ist allerdings nicht direkt für die Applikationsentwicklung gedacht, sondern als Integrations-API, um das künftige Zusammenspiel verschiedener MicroProfile-Libraries in Bezug auf Reactive Streams zu definieren. Ein tatsächlicher Nutzen in 2.1 ist also nicht gegeben, außer für MicroProfile selbst – nämlich Aufmerksamkeit für das Thema zu erzeugen. Aber parallel wird an „Reactive Messaging 1.0" gearbeitet, das dann mit MicroProfile 2.2 bereits Mitte Dezember erscheinen soll.

http://microprofile.io

28. August 2018

Mission Control

Oracle hat Java Mission Control als Open Source freigegeben und will es als separaten Download für OpenJDK und Oracle-JDK-Nutzer verfügbar machen. Damit kann es unabhängig von den einzelnen JDK-Versionen entwickelt, freigegeben und betrieben werden. Gemeinsam mit der Aufnahme des Java Flight Recorder in das OpenJDK 11 steht dann auch hier das mächtige Werkzeug zur Verfügung.

http://www.oracle.com/technetwork/java/javaseproducts/mission-control/java-mission-control-1998576.html

28. August 2018

Der Preis von Java

Eine detaillierte Antwort auf die Frage „Wird Java jetzt kostenpflichtig?" geben Hendrik Ebbers und Timo Brandstätter. Sie gehen auf die Support-Problematik ein, zeigen die anfallenden Kosten für kommerzielle Nutzer sowie Alternativen. Nur ein Punkt ist schon fast wieder obsolet: Unterschiede zwischen Oracle JDK und OpenJDK soll es mit Version 11 ja nicht mehr geben – mit dem Flight Recorder und Mission Control wird das letzte kommerzielle Feature aufgegeben (abgesehen vom Tooling drumherum). Zahlen werden die Kunden dann hauptsächlich für den Support – und den Zeitgewinn, um nicht immer den Releases hinterherhecheln zu müssen.

https://www.heise.de/developer/artikel/Wird-Java-jetzt-kostenpflichtig-4144533.html

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

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

Der Weg nach Jakarta

Im April 2018 hat die Eclipse Foundation die Verantwortung für Enterprise Java übernommen. Mike Milinkovich, Geschäftsführer der Eclipse Foundation, stellt sich den Fragen des iJUG.

Der iJUG ist seit Kurzem ein Mitglied der Eclipse Foundation. Was kann die deutschsprachige Java-Community für die Eclipse Foundation tun?
Sich beteiligen! Beteiligung und Mitwirkung sind das Herzblut jeder Open-Source-Community. Wir freuen uns, dass wir den iJUG als Mitglied haben, und sind gespannt auf die Zusammenarbeit mit euch und euren Mitgliedern. Die Java-Community in Deutschland, Österreich und der Schweiz ist sehr stark und wir werden ihre Hilfe benötigen, um Jakarta EE zu einem Erfolg zu verhelfen.

Im Gegenzug ist die Eclipse Foundation Fördermitglied des iJUG geworden. Was war die Motivation dafür?
Wir möchten unsere Unterstützung für die iJUG-Community zum Ausdruck bringen. Meines Wissens ist die Eclipse Foundation die einzige nordamerikanische Open-Source-Software-Foundation mit einer Niederlassung in Deutschland. Die Eclipse-Community war hier schon immer stark und wir möchten daher als Community weiterwachsen, um die Leute aufzunehmen, die Jakarta EE für ihre Systeme und Anwendungen nutzen.

Wie verläuft die Migration von Java EE zur Eclipse Foundation?
Bisher verläuft sie gut. Bei dieser Migration gibt es zwei wesentliche Teile. Der erste ist die Übernahme von GlassFish als Open-Source-Referenz-Implementierung für Java EE. Dieser Teil verläuft sehr gut, wie auch auf unserer EE4J-Statusseite zu erkennen ist. Wir denken, dass wir immer noch im Zeitplan zur Veröffentlichung vom Java-EE-8-zertifizierten Eclipse GlassFish für diesen Herbst sind. Der zweite Teil ist der Aufbau eines neuen Spezifizierungsprozesses und der Start zur Entwicklung der Spezifikationen zu Java EE. Dieser Teil verläuft bedauerlicherweise recht schleppend. Wir hoffen, dass das Ende der Sommerferien dafür sorgt, dass dieses Thema Fahrt aufnimmt.

Wie funktioniert die Jakarta-EE-Arbeitsgruppe?
Die Arbeitsgruppen der Eclipse Foundation führen die Mitglieder zusammen, die ein Interesse für einen bestimmten Bereich teilen. Bei Jakarta EE ist das Ziel die Gründung eines Konsortiums aus Herstellern, System-Integratoren und Unternehmen im Ökosystem von Jakarta EE (derzeit Java EE). Die Jakarta-EE-Arbeitsgruppe hat eine Satzung, in der das Führungsmodell mit seinen verschiedenen Ausschüssen zum Steuern ihrer Tätigkeiten konkretisiert ist. Die Mitarbeiter der Eclipse Foundation bieten fortlaufenden Support für ihre vielfältigen Aktivitäten.

Was ist der Jakarta-EE-Spezifizierungsprozess?
Der Jakarta-EE-Spezifizierungsprozess wird den Java Community Process (JCP) ersetzen, um die Spezifikationen für Java-Infrastrukturen in Unternehmen zu erstellen und weiterzuentwickeln. Als vollständiger Ersatz für JCP wird er sich hoffentlich dahin entwickeln, zukünftige Java-Lösungen für Unternehmen in den kommenden Jahrzehnten abzubilden.

Wie motiviert die Eclipse Foundation die Community zum Mitmachen bei Jakarta EE?
Ehrlich gesagt müssen wir da gar nicht viel machen. Die Entwickler, die die Jakarta-EE-Plattform verbessern und weiterentwickeln möchten, sind bereits motiviert. Wir stellen Infrastruktur und Prozesse für den Projekterfolg bereit, aber wir motivieren keine einzelnen Mitwirkenden.

Was sind die zentralen Ergebnisse der Jakarta-EE-Umfrage?
Die von uns im März durchgeführte Umfrage war wirklich sehr interessant, da sich einige Dinge, die wir angenommen haben, bei denen wir allerdings nicht ganz sicher waren, als wahr herausstellten. Die drei wichtigsten Erkenntnisse sind: Erstens möchten Entwickler, dass die Java-Unternehmensplattform bessere Unterstützung für Microservices bietet. Wir waren recht überrascht, dass bereits zwei Drittel der Befragten Microservices in Java erstellen oder das zumindest kurzfristig planen. Die zweite Erkenntnis ist die native Integration mit Kubernetes. Da es ein großes Interesse an der Nutzung von Kubernetes als Cloud-portable Orchestrierungsschicht gibt, hat uns das nicht überrascht. Der dritte Kernbereich ist die Steigerung des Innovationstempos innerhalb der Java-EE-Technologie-Plattform. Auch wenn Stabilität als wichtig erachtet wurde, möchten Entwickler schon, dass ihre Plattform auf die neuesten Technologie-Trends viel schneller reagiert, als wir das in den vergangenen Jahren gesehen haben.

Mike Milinkovich
Geschäftsführer Eclipse Foundation

Mike Milinkovich ist seit mehr als dreißig Jahren in der Software-Industrie tätig und hat bereits alle Rollen ausgefüllt, angefangen bei der Software-Entwicklung über Produkt-Management bis hin zu IP-Lizenzierung. Er ist seit dem Jahr 2004 Geschäftsführer der Eclipse Foundation.

„Es wäre zu schön, wenn die Eclipse Foundation es zumindest schafft, den JCP zu ersetzen …“

Ja, ja, ich weiß. Nun komme ich wieder und mache alles mies. Ich bin es ja auch leid, immer die Kassandra zu geben, aber auch bei der Eclipse Foundation ist nun mal nicht alles Gold, was glänzt. Ein Kommentar vom Markus Karg von der Java User Group Goldstadt.

Wie Mike Milinkovich im Interview zugibt, läuft der Übergabeprozess schleppend. Abgesehen vom erwähnten Spezifikationsprozess ist das TCK noch nicht an die Community übergeben, und wenn es kommt, ist es erst mal ein einziges, großes Code-Monster, da Oracle keine einzelnen TCKs pro API besitzt. Von den Spezifikations-Dokumenten wollen wir erst gar nicht reden: Es ist noch nicht einmal ausgehandelt, ob Oracle die überhaupt an die Eclipse Foundation übergeben wird. Und, apropos nicht ausgehandelt, es gibt noch keinen Vertrag, der es erlaubt, weitere Funktionen zu bestehenden APIs hinzuzufügen. Deswegen ist es bis auf Weiteres untersagt, beispielsweise Nightly Builds von solchen neuen Funktionen zu veröffentlichen. Die von der Community beispielweise in JAX-RS 2.2 investierte Arbeit von Monaten ist damit zunächst einmal wertlos. Einen Termin zur Lösung dieser Probleme wollen weder Oracle noch die EF benennen. Doch ohne die Lösung aller dieser Probleme wird es kein Jakarta EE geben. Der vor einem Jahr auf der ECE 2017 in Ludwigsburg genannte Termin (Jakarta EE 8.0 zur ECE 2018) ist nahezu unmöglich zu halten.

Die Vorstellung, Java EE sei an die Community übergeben worden, ist schlicht falsch. Hinter Jakarta EE steht eine Working Group der Eclipse Foundation (EF). Entsprechend deren Regeln bestehen solche Arbeitsgruppen jedoch meist aus Vertretern zahlungskräftiger Unternehmen – selbst wenn diese an einem Projekt gar nicht mitprogrammieren. Alle wesentlichen Entscheidungen werden somit nicht von denen getroffen, die die Arbeit machen (also den Code pflegen), sondern von Angestellten. Programmierer haben zwar eine gemeinsame Stimme, die wiegt allerdings die vielen Stimmen der zahlenden Unternehmen nicht annähernd auf. Die EF hat insofern eine ungewöhnliche Interpretation von „Community“ und „Demokratie“.

Faktisch hat Oracle weiterhin die Macht. Zwar sieht die EF vor, dass Committer Wahlen abhalten, um Projektleiter und neue Committer zu benennen. Die Erstbesetzung dieser Rollen hat die EF allerdings weitgehend mit Oracle-Mitarbeitern vorgenommen. Da diese weiterhin die Mehrheit haben, biegen sie gerne mal die EF-Regeln, diskutieren und entscheiden hinter verschlossenen Türen und machen teilweise einfach so weiter, als wäre nichts gewesen. Erst auf lautstarke Intervention einiger externer Committer hat sich die EF zu einem kurzen Ordnungsruf durchgerungen – der jedoch wirkungslos blieb.

Oracle lässt auch weiterhin neue Oracle-Mitarbeiter von bestehenden Oracle-Mitarbeitern durchwinken. Das Project Management Committee (PMC) erweist sich hier als zahnloser Tiger, da man selbst keine Rekrutierungsstrategie besitzt. Dass hier neben Ivar Grimstad kein weiterer Community-Vertreter sitzt, sondern ausschließlich Mitarbeiter von Application-Server-Herstellern, verwundert da schon gar nicht mehr. Ebenso wenig wie die Tatsache, dass Dmitry Kornilov von Oracle zwar ständig im Namen des PMC Regeln festlegt, der tatsächliche PMC-Lead Ivar Grimstad aber eher nie in Erscheinung tritt. Nach echter Führung durch die Community sieht das nicht aus.

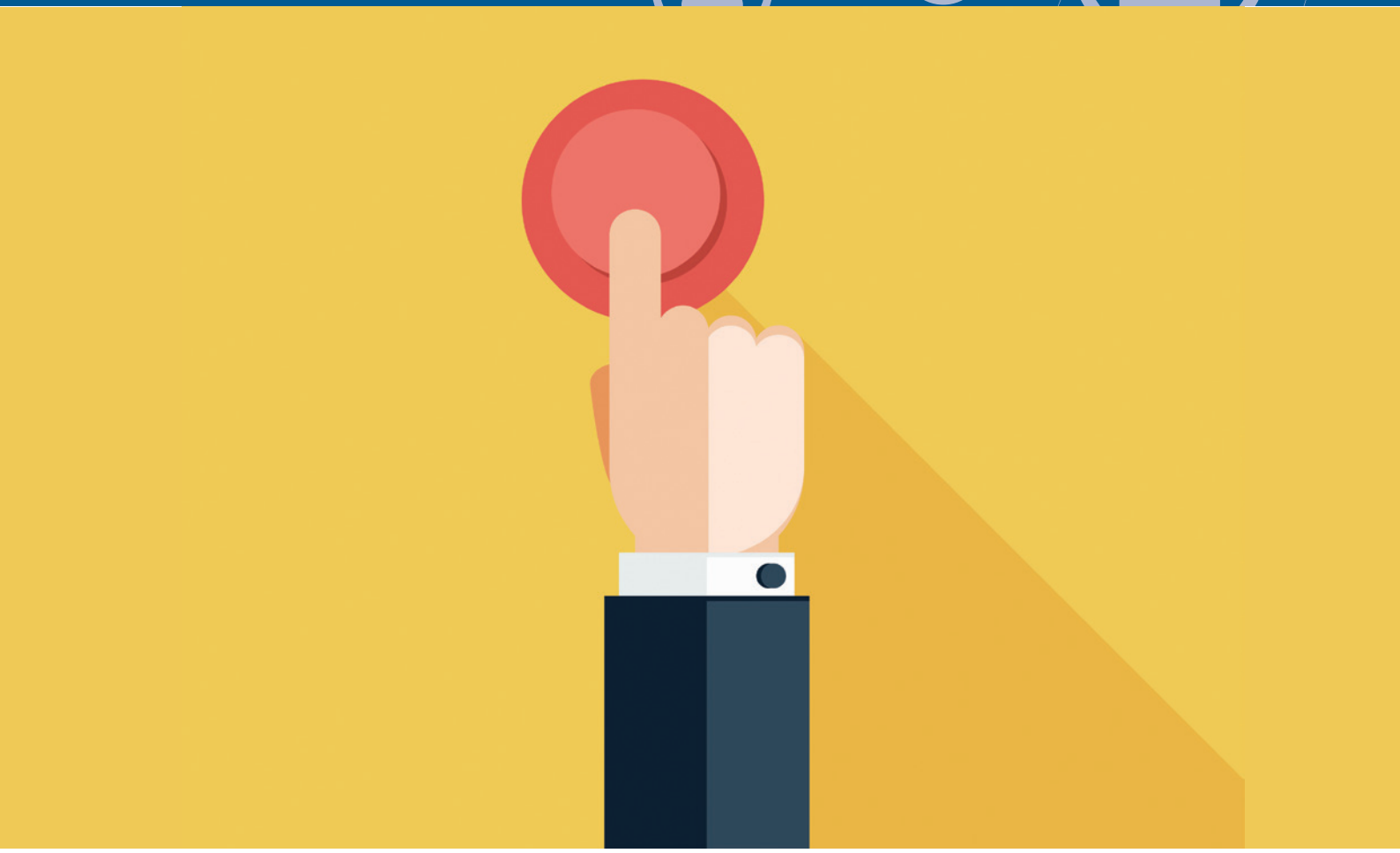
Zu schön wäre es, wenn die EF es zumindest schafft, den Java Community Process (JCP) zu ersetzen. Das wird sie jedoch leider nicht. Niemals. Und zwar aus zwei simplen Gründen: Erstens hat der Oracle-beherrschte JCP keinerlei Anweisung und Plan, sich selbst aufzulösen. Zweitens hat Oracle dem JCP das alleinige Recht zugesprochen, Standards für die Java-Welt festzulegen und dafür die Packages „java.*“ und „javax.*“ zu verwenden. Ein entsprechendes Abkommen hierzu besteht mit der EF bis heute nicht. Es macht insofern also gar nichts aus, dass die EF bis heute – abgesehen von einem teilweise heftig kommentierten ersten Entwurf eines Regelwerks – nichts in Richtung „Standardisierung“ vorweisen kann, sie dürfte das Ergebnis sowieso weder Java nennen, noch hätte sie die Macht, eine Normierung durchzusetzen. Die EF tut sich damit extrem schwer, denn sie ist nun mal keine Standardisierungsorganisation wie der JCP oder die ISO.

Die EF ist lediglich ein Hersteller-Club: Sie organisiert die gemeinsame Arbeit an Produkten durch bezahlende Mitglieder. Mehr nicht. Es mag sein, dass Mike Milinkovich nun glaubt, die EF sei der Hort der Innovation, wie er es auf der ECE 2017 verkündet hat. Viel kam dabei allerdings bislang nicht heraus.

Es wäre zu schön gewesen, nach einem Jahr seit der Ankündigung Java EE wirklich unter Kontrolle der Anwender zu wissen, mit Hunderten Freiwilligen, die per Pull Request in hoher Kadenz neue Versionen der wichtigen APIs hinausfeuern. Pustekuchen. Die Handvoll aktiv mitwirkender Enthusiasten, zu der auch ich mich zähle, genügt bei Weitem nicht, um ein Großprojekt wie Jakarta EE zu stemmen. Und dass Oracle wirklich die Macht abgibt, ha ha ha, das hat doch nicht wirklich jemand ernsthaft geglaubt, oder...?

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.



IoT-Hands-on – vom Prototyping zur Business-Idee

Stefan Rauch und Lars Orta, iteratec GmbH

Um etwas tun zu können, muss man es verstehen. Um etwas zu verstehen, muss man es erleben. Genau darum geht es in diesem Artikel – zu inspirieren, das Internet der Dinge nicht mehr als abstrakten Begriff in der Ferne zu sehen, sondern selbst anzufangen und zu erleben, wie einfach es ist, Sensoren und Aktoren an einen Mikrocontroller anzuschließen und diesen zum Beispiel mit der Arduino-IDE zu programmieren.

Der Artikel zeigt zudem, wie man unterschiedlichste Systeme und Schnittstellen über die Node-RED-Plattform integrieren kann und wie einfach es ist, auch das Smartphone einzubinden. Die Autoren geben auf unterhaltsame und spielerische Weise einen Einblick in die IoT-Welt und versuchen dazu zu begeistern, selber gleich loszulegen. Denn nur wer beginnt, viele eigene Ideen umzusetzen, wird auch die richtigen Ideen für sein Unternehmen finden.

Es bedarf vieler Ideen, die tatsächlich erprobt werden, um eine gute Lösung für ein Problem zu finden oder eine wirkliche Innovation zu gestalten. Das sollte Kern der Gedanken sein und man sollte nicht resignieren, wenn die eine bahnbrechende Idee nicht sofort kommt. Vielmehr sollte man sich zuerst das Handwerkszeug zulegen, um Ideen auf einfache Art und Weise prototypisch umzusetzen und dann loszulegen: Egal, ob man sich schon immer über etwas geärgert hat und das verbessern will oder eine coole, nette, lustige Idee hat, die man schon immer mal ausprobieren wollte – einfach loslegen …

Abbildung 1: Prototyp von Square

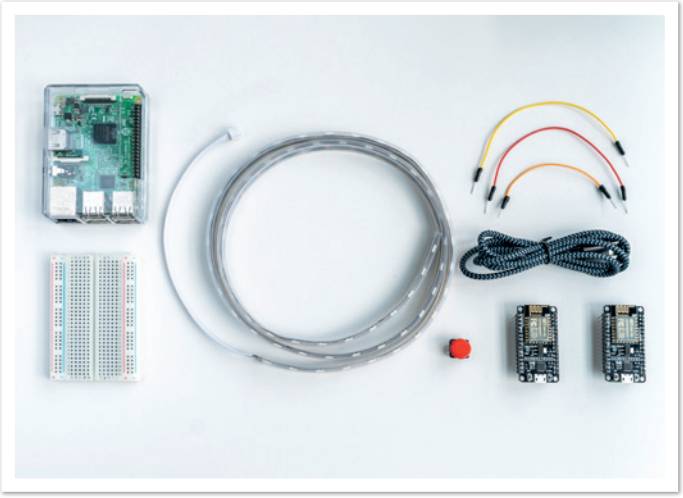


Abbildung 2: Das ist notwendig, um anzufangen

Eine motivierende Beispielgeschichte

Eines der Lieblingsbeispiele der Autoren ist die Geschichte von Jim McKelvey (kennt man vermutlich nicht) und Jack Dorsey (ist gegebenenfalls ein Begriff für Twitter-Nutzer). Es geht um die Gründungsgeschichte von Square. Der studierte Informatiker Jim McKelvey hat viele Talente. Eine Sache beherrscht er allerdings besonders gut: Glasblasen, also die Kunst, Glas in bestimmte Formen zu bringen. Dafür hat er im Mittleren Westen der USA in der Nähe von St. Louis ein eigenes Geschäft eröffnet.

Einige Kunden sind jedoch nicht in der Lage, die Kunstwerke im Preis von zweihundert bis viertausend US-Dollar in bar zu bezahlen, nur weil Jim McKelvey keine Kreditkarten akzeptiert. Dafür sind die Kreditkarten-Abrechnungssysteme im Jahr 2009 schlicht zu schlecht. Es würden zu hohe Kosten entstehen, würde er eines der etablierten Systeme nutzen, etwa für die Abrechnungen von American-Express-Kreditkarten. Als er deswegen wieder einmal ein Geschäft im Wert von zweitausend US-Dollar verliert, ruft er seinen Freund Jack Dorsey an und sie beschließen, selber eine Lösung zu basteln. Sie haben die Idee, einen möglichst kleinen Magnetkartenleser zu bauen, ihn mit einem iPhone zu verbinden und über dieses die Abrechnung zu tätigen.

Mithilfe seines Vaters Jack Dorsey, Dekan für Maschinenbau an der Washington University in St. Louis (WUSTL), einem weiteren Freund sowie Professor Bob Morley, ebenfalls von der WUSTL, gelingt es Jim McKelvey, in wenigen Monaten einen funktionierenden Prototyp zu bauen. Man zieht dabei die Kreditkarte durch das kleine Lesegerät, das in die Audiobuchse des iPhone gesteckt wird. Die Kreditkarte wird erkannt und über die iOS App kann die Abrechnung erfolgen *(siehe Abbildung 1)*.

Im Jahr 2010 geht der Service auch für andere an den Markt und entpuppt sich als absolutes Erfolgsmodell: Im November 2014 wird die 1.000.000.000ste Zahlung über Square abgewickelt. Beim Börsengang im Jahr 2015 wird Square mit rund vier Milliarden US Dollar bewertet. Das Geschäftsmodell von Square soll vor allem kleinen Unternehmen helfen und den teuren Mittelsmann (in diesem Fall die Kreditkarten-Abrechnungssysteme) eliminieren, typisch für disruptive, digitale Technologie-Startups. Das konsequente Nutzen der vorhandenen Daten, die auch allen Beteiligten am Zahlungsprozess mehrwertig zur Verfügung gestellt werden, hat das Unternehmen – unabhängig von Hardware – erfolgreich gemacht. Mittlerweile wird der Kreditkarten-Reader in vielen Fällen nicht mehr benötigt, sondern es wird direkt über das Handy bezahlt.

Square ist ein tolles Beispiel dafür, wie aus der Lösung eines individuellen Problems durch das Verbinden von Hardware (Magnetstreifenleser) mit dem Internet (hier über das Smartphone) ein Nutzen geschaffen wurde. Sicher haben weder Jim McKelvey noch Jack Dorsey zu Beginn gedacht, dass eine Bastelei sie zu Milliardären macht. Entscheidend ist aber, dass sie angefangen haben. Genau dazu möchten die Autoren inspirieren.

In der Embedded-Welt beginnen

Im IoT-Bereich haben wir den Vorteil, Dinge im wahrsten Sinne des Wortes anfassbar machen zu können. Wer nach „IoT" googelt und auf die Bildersuche klickt, kann etwas erhalten, das hier nicht gemeint ist. Wir möchten es einfach halten. Für einen einfachen IoT-Use-Case sind drei Dinge erforderlich *(siehe Abbildung 2)*:

- Ein Sensor, der Daten produziert
- Ein Nachrichtenbroker, der Daten verteilt
- Ein Aktor, der mit den Daten etwas macht

Jetzt ist nur noch festzulegen, wie Sensor und Aktor mit dem Internet verbunden werden und welches Nachrichten-Protokoll zum Einsatz kommt. Die Autoren empfehlen zum Start:

- MQTT als Protokoll nutzen und einen MQ-Broker als Nachrichten-Broker. Die Vorteile von MQTT liegen in der enormen Verbreitung (es gibt für viele Umgebungen sehr gute Open-Source-Libraries), in der leicht zu implementierenden Security, in der Strukturierung über „topics", in der Tatsache, dass sich neue Clients dynamisch

zur Laufzeit zum Senden und Empfangen anmelden können, sowie in der breiten Verfügbarkeit von freien Message-Brokern und Anleitungen im Netz *(etwa siehe „https://www.hivemq.com/blog/MQTT-essentials")*.

- Man kann zum Beispiel einen günstigen ESP-8266-Microcontroller für die Programmierung, Internet-Verbindung sowie Anbindung von Sensor und Aktor einsetzen. Dieser besitzt bereits ein WLAN-Modul und ist sehr einfach mit der Arduino-IDE programmierbar (C-basiert).
- Außerdem ein von den Autoren vorbereitetes Raspberry-PI-Image nutzen, um den Raspberry inklusive des vorinstallierten Mosquitto-MQ-Brokers als Server zu Hause einzusetzen *(siehe „https://www.iteratec.de/iot-hands-on-tutorial")*.

Zu Beginn sucht man sich einen möglichst simplen Use-Case aus, zum Beispiel den folgenden:

- Man nutzt einen Button als Sensor – jedes Mal, wenn er gedrückt wird, wird eine Nachricht gesendet. Die Nachricht wird im Beispiel auf das topic „iot/button" gelegt.
- Man nimmt einen LED-RGB-Stripe als Aktor. Dieser wird ebenfalls physisch mit einem ESP-8266-Microcontroller verbunden und lässt sich so unabhängig vom Button programmieren. Man macht einen Subscribe mit dem LED-RGB-Stripe auf das Topic „iot/button". Jedes Mal, wenn der Button gedrückt wird, wechselt man die Farbe oder schaltet den Stripe an und aus. Fertig.
- Im Idealfall hat das weniger als 60 Minuten Zeit gekostet und der erste IoT-Use-Case ist umgesetzt.

Mit exakt dem gleichen Setup, nur einem anderen Sensor, lassen sich sehr schnell sinnvolle Use-Cases zu Hause oder im Büro umsetzen:

- Man setzt eine Wägezelle in seinen Briefkasten. Wenn sich das Gewicht ändert, sendet man das aktuelle Gewicht beispielsweise auf „iot/briefgewicht" und verbindet diesen mit dem LED-RGB-Stripe. Abhängig vom gesendeten Wert signalisiert dieser, ob Post im Briefkasten ist.
- Man legt einen Temperatursensor mit ESP-8266 auf die Fensterbank, den Balkon oder in den Garten, sendet alle zehn Minuten den aktuellen Wert an den Topic „iot/outsidetemperature", verbindet diesen mit dem LED-RGB-Stripe und überlegt sich eine Farbskala zum Anzeigen der Außentemperatur.
- Man installiert einen Anwesenheitssensor in einem nicht buchbaren Meetingraum und bringt den LED-RGB-Stripe außen an der Tür an. Dann sendet man die Anwesenheitswerte im Zehn-Sekunden-Takt auf „iot/anwesenheit/raumbezeichner", verbindet diesen mit dem LED-RGB-Stripe und signalisiert die Raumbelegung mit einer entsprechenden Farbgebung.
- Zur weiteren Anbindung von Sensoren/Aktoren finden sich unter dem Stichwort „Arduino" viele Beispielprojekte im Internet. Sie lassen sich als Ausgangspunkt für eigene Erweiterungen/Ideen nutzen.

Der Untertitel des Artikels lautet „vom Prototyping zur Business-Idee"; zugegeben, die beschriebenen Use-Cases verheißen nicht sofort das große Geld, aber man betrachte dazu die Historie von Ring auf Forbes.com unter *„https://bit.ly/2vs31iw"*. Die allerersten Anfänge von Jamie Siminoff waren ähnlich. Es ist enorm wichtig, tatsächlich anzufangen. Es ist wahrscheinlich nicht die erste Idee beziehungsweise der erste Prototyp, der zur richtigen Business-Idee wird. Wichtig ist, die Sensoren und Aktoren aus dem Embedded-Bereich schnell und einfach auszuprobieren und auch mit weiteren Schnittstellen und Elementen aus dem Internet zu verbinden.

Embedded allein reicht nicht

Nach Einschätzung der Autoren ist es wichtig, im Embedded-Bereich eigene Hands-on-Erfahrungen gemacht zu haben, um die Prinzipien und Möglichkeiten des IoT zu verstehen. Wer seine Prototypen auf den Embedded-Bereich beschränkt, ist einerseits limitiert, was die Möglichkeiten der Anbindung weiterer Schnittstellen, der Visualisie-

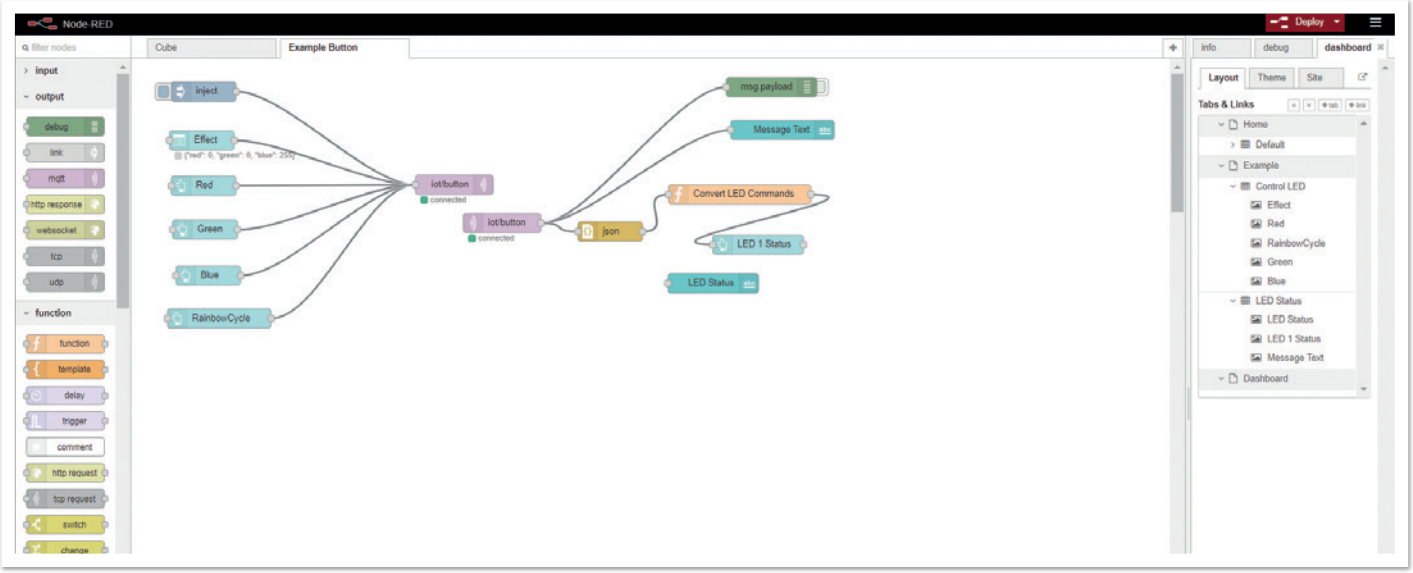


Abbildung 3: So könnte der Flow für das IoT/Button-Beispiel aussehen

rung und der Verbreitung der Daten und Anwendungsfälle angeht. Andererseits ist man dadurch auch oft nicht schnell genug, um eine Idee einfach mal auszuprobieren. Hier empfiehlt sich ausnahmsweise ein grafisches Entwicklungswerkzeug.

Für viele Software-Entwickler sind grafische Benutzeroberflächen mit WYSIWIG-Editor zu Recht eine unbefriedigende Lösung, aber für den speziellen Fall des Prototyping im IoT-Bereich ist Node RED ein sehr gut geeignetes Werkzeug. Ursprünglich unter dem Namen „IBM Node RED" hat IBM ein Open-Source-Werkzeug zur Verfügung gestellt, das sehr einfach und intuitiv zu bedienen ist. Zudem lassen sich damit sehr schnell Hardware-Geräte, APIs und Online-Dienste als Teil des Internets der Dinge miteinander verbinden.

Node RED bietet einen browserbasierten Flow-Editor, mit dem JavaScript-basierte Funktionen erstellt werden können. Die Funktionen werden dabei als Knoten (Nodes) in den Flow gezogen, gegebenenfalls konfiguriert und/oder programmiert (JavaScript) und per Maus miteinander verbunden. Es gibt eine Vielzahl vordefinierter Knoten, die es dem Benutzer erlauben, Funktionen tatsächlich per Drag and Drop zu seinem Use-Case hinzuzufügen. Die Laufzeitumgebung von Node RED ist auf NodeJS aufgebaut. Somit lassen sich verschiedenste Knoten quasi als Libraries über NodeJS in die Node-RED-Instanz einspielen. Jeder Flow lässt sich exportieren und in einer anderen Instanz über die Weboberfläche wieder einspielen. Unter *„https://nodered.org"* stehen viele Beispiele und Anleitungen.

Anhand der im Embedded-Bereich begonnenen Use-Case-Beispiele wird nun gezeigt, wie Node RED hilft, die Ideen zu erweitern und schnell auszuprobieren. Dafür wurde im Raspberry-PI-Image auch eine Instanz von Node RED installiert. Sie ist über jeden Webbrowser unter *„http://IP_des_Raspberry:1880/"* zu erreichen. Einige Beispiel-Flows stehen auf der Landingpage *„https://www.iteratec.de/iot-hands-on-tutorial"* zur Verfügung.

Man kann direkt mit dem ersten Beispiel anfangen. Es wurde bewusst ein Button gewählt, da es sehr einfach ist, diesen zu stecken und zu implementieren. Dennoch kann man exakt für das Beispiel „Button -> RGB–Stripe" den Button auch in Node RED simulieren. Dazu werden auf dem Flow zwei Dinge hinzugefügt:

- Ein Inject Node
- Ein MQTT-out Node

Den Inject Node verbindet man mit dem MQTT-out Node. Beim Inject Node werden per Doppelklick die Eigenschaften wie folgt geändert: Als „Payload" wird nicht „timestamp", sondern „string" versendet. Man kopiert eine Nachricht des Embedded-Buttons und fügt diese genau hier ein. Der Name ist „Button-Simulator". Mit einem Klick auf „done" ist alles fertig und der Inject Node sendet exakt diese Nachricht, und zwar jedes Mal, wenn man im Flow auf die kleine Schaltfläche vor dem „Button Simulator" klickt.

Der MQTT-out Node verbindet einen über die Eigenschaften mit dem „mosquitto Broker" vom Raspberry-Image. Da bereits einige Beispiel-Flows hinterlegt sind, ist der mosquitto Broker in der Eingabemaske hinter dem Feld-Server bereits voreingestellt. Man setzt nun das richtige Topic „iot/button" und klickt auf „done". Der Flow ist nun fertig und man hat den Embedded-Button in Node RED simuliert *(siehe Abbildung 3)*. Die Änderungen werden wirksam, wenn man per „deploy" (oben rechts) den Flow veröffentlicht. Bei jedem Klick auf den Button-Simulator wird das RGB-LED-Stripe nun die Farbe wechseln.

Node RED bringt von Haus aus eine Vielzahl von Oberflächen-Elementen mit, die man nutzen kann, um prototypisch eine Webseite zu gestalten. Node RED generiert dabei die Webseite dynamisch auf Basis der verwendeten UI-Knoten. Für alle genannten Beispiele bietet eine Webseite mit Informationen abhängig von den Daten des Sensors gegebenenfalls einen Mehrwert. Um dies auszuprobieren, ist Folgendes zu tun:

- Einen MQTT-in Node zum Flow hinzufügen und auf das Topic des Beispiels („iot/briefgewicht", „iot/outsidetemperature" oder „iot/anwesenheit/raumbezeichner") subscriben

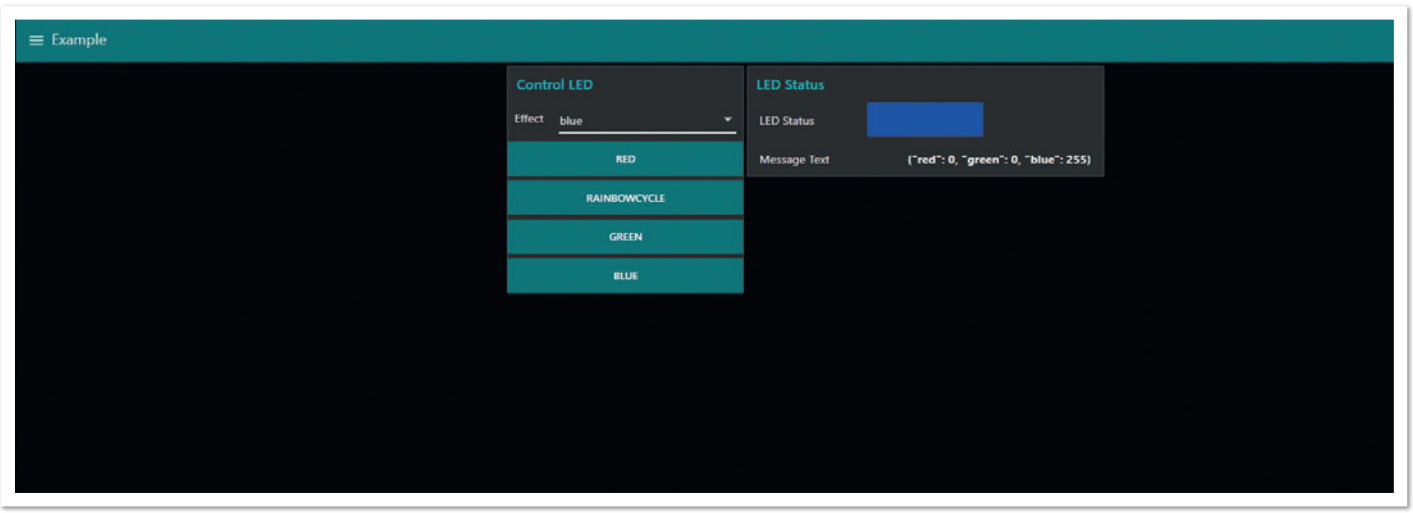


Abbildung 4: Website-Prototyp aus NodeRED zur Anzeige und Steuerung des LED-Stripes

- Einen text Node zum Flow hinzufügen und den MQTT-in Node mit dem text Node verbinden.
- Die Eigenschaften des text Node ändern: Group -> „default". Name -> „Name eigener Wahl".
- Die Änderungen per Deploy veröffentlichen und die Webseite über *„http://IP_des_Raspberry:1880/ui/"* öffnen

Es gibt nun eine Webseite, die in einem Textfeld den Payload der jeweiligen Nachricht des Sensors anzeigt *(siehe Abbildung 4)*. Das hat vermutlich weniger als fünf Minuten gedauert. Der Payload in einem Textfeld ist natürlich nur ein allererster Prototyp einer sinnvollen Webseite. Mithilfe weiterer Nodes sowie ein klein wenig JavaScript- und JSON-Know-how lässt sich jedoch sehr schnell eine sinnvolle Webseite erstellen:

- Eine Webseite, die die aktuelle Temperatur im Garten/auf dem Balkon anzeigt, den Verlauf über die letzten fünf Tage in einem Diagramm visualisiert sowie den Zustand des Briefkastens mit einem farblichen Element anzeigt
- Eine Webseite, die den Belegungszustand sämtlicher nicht buchbarer Räume im Büro zum aktuellen Zeitpunkt anzeigt

Auf der genannten Landingpage finden sich vorgefertigte Flows für diese Webseiten. Node RED kann allerdings nicht nur den Sensor simulieren und dabei helfen, schnell eine Webseite zu erstellen. Es ermöglich auch, Use-Cases zu erweitern, indem man ein beliebiges API anspricht und integriert. Bereits in der Standard-Installation von Node RED besteht so die Möglichkeit, mit wenigen Klicks die wichtigsten technischen Kommunikationsprotokolle zu nutzen: MQTT, http/https (und damit natürlich auch alles REST-Services), TCP, WebSockets sowie eine Vielzahl von Adaptern zu bestehenden Systemen (wie Twitter, E-Mail, etc.). Für das Beispiel mit der Außentemperatur haben die Autoren auf der Webseite beispielsweise Vergleichswerte von „accuweather" integriert, da es hier ein sehr einfaches API gibt.

Für die Meetingräume kann man natürlich auch den Exchange-Server ansprechen und so anzeigen, ob die Räume frei, besetzt, frei werdend etc. sind. Anhand dieser Informationen könnte man mit einem RGB-LED-Stripe zum Beispiel auch im Raum visualisieren, wenn das Meeting bald zu Ende sein sollte (etwa ein gelbes Blinken fünfzehn Minuten vor Ende).

Beim Ausprobieren von Node RED wird man schnell merken, mit welcher Geschwindigkeit sich erste Ideen umsetzen und Sensoren und Aktoren sinnvoll mit weiteren APIs integrieren lassen. Das leichtgewichtige Erstellen von einfachen Web-Oberflächen hilft enorm, die Prototypen auch für andere erlebbar zu machen. Gerade das ist wichtig: Man ist schnell begeistert von seiner eigenen Idee – mit Node RED lässt sich testen, ob auch andere die Idee gut finden und nutzen wollen. Das ist ein entscheidender Faktor auf dem Weg vom Prototyping zur Business-Idee: Den Prototyp so früh wie möglich mit echten Nutzern testen und ihn anhand des Feedbacks verbessern.

Jetzt noch das Smartphone nutzen

Eine Internet-of-Things-Anleitung kann heutzutage nicht ohne den Einsatz eines Smartphones geschrieben werden. Und natürlich sollte man darüber nachdenken, in welcher Form das Smartphone als Aktor oder auch als Sensor integriert werden kann. In den genannten Beispielen kann das Smartphone per Notification den Benutzer darauf hinweisen, dass neue Post im Briefkasten oder ein bestimmter Raum gerade frei geworden ist. Genauso kann anstelle mit einem Anwesenheitssensor in den nicht buchbaren Räumen mit einem Bluetooth-Scanner erkannt werden, ob ein Smartphone im Raum ist – und es somit indirekt als Sensor verwendet werden.

Das Smartphone hat einige Sensoren, die bereits heute in vielen echten Anwendungsfällen genutzt werden (GPS, Bewegungssensor etc.), und bietet – ähnlich wie Node RED – die sehr einfache Integration komplexer APIs. Man kann alle Funktionen des Smartphones nutzen, die man auch bei der bisherigen App-Entwicklung eingesetzt hat. Die Anbindung an die IoT-Welt erfolgt über das Senden und Empfangen von MQTT-Nachrichten mit einer der gängigen Libraries *(etwa „https://www.eclipse.org/paho/clients/android")*.

Die Autoren stellen auf ihrer genannten Landingpage eine kleine Demo-Applikation zur Verfügung, die aufzeigt, wie man ein Android-Smartphone an den mosquitto Broker des Raspberry Image anbindet und auf „iot/briefgewicht" per Notification reagiert, wenn der Inhalt der Nachricht suggeriert, dass neue Post im Briefkasten ist. Darüber hinaus ist ein Beispiel dafür hinterlegt, wie einfach es ist, den LED-RGB-Stripe per Sprachsteuerung auf eine bestimmte Farbe zu setzen.

Fazit

Die Autoren sind überzeugt, dass im Zeitalter der digitalen Transformation die Fähigkeit, mit disruptiven Technologien umzugehen, ein entscheidender Erfolgsfaktor ist. Die Technologien selber hands-on erlebt zu haben, verschafft ein ganz anderes Verständnis für mögliche Einsatzzwecke. Egal, ob man in der IT arbeitet oder in einer anderen Funktion tätig ist: Man sollte sich mit IoT, Machine Learning, Blockchain oder Augmented Reality befassen.

Die Autoren möchten mit diesem Artikel animieren und inspirieren. Das Setup ist bewusst darauf ausgelegt, sehr schnell erste Prototypen entwickeln zu können, um Ideen schnell testen zu können. Vom Prototyp zur Business-Idee gibt es nach ihrer Meinung nach keinen direkten Weg, auch wenn das einige so versprechen. Selbst eine noch so gute Business-Idee scheitert schnell, wenn man keine Möglichkeit findet, sie mit einem Prototyp zu validieren.

Das Verständnis der Technologien zusammen mit dem Verständnis seines Business und einem Vorgehen, das frühes Lernen selbstverständlich macht, erhöht die Chancen, irgendwann die richtige Business-Idee auch passend umzusetzen. Die Autoren hoffen, dazu ein bisschen beigetragen zu haben, und wünschen viel Spaß mit dem Internet der Dinge.

Stefan Rauch
Stefan.Rauch@iteratec.de

Stefan Rauch arbeitet seit mehr als zwanzig Jahren in der IT-Branche. Bis vor Kurzem war er der verantwortliche Projektmanager für alle iteratec-Aktivitäten im BMW-CarSharing-Kontext, dabei hat er insbesondere die Transformation von DriveNow vom Startup zum international etablierten CarSharing-Anbieter begleitet. Stefan Rauch ist seit jeher sehr an Technologien mit hohem Business-Impact und allen möglichen Innovationsprozessen interessiert. Daher ist er auch einer der treibenden Köpfe für den Innovation Frei-Day und weitere Digitalisierungsthemen bei iteratec.

Lars Orta
Lars.Orta@iteratec.de

Lars Orta arbeitet als Senior IT-Architekt bei iteratec. Nach seinem Informatikstudium an der TU München im Jahr 1999 fand er schnell den Weg zu iteratec. Nach sechs Jahren wechselte er zur Swiss Life. Seit dem Jahr 2014 gehört er wieder zur iteratec-Familie und kümmert sich seitdem schwerpunktmäßig um Versicherungen und Unternehmen aus dem Finanzsektor. Lars Orta verfügt über langjährige Erfahrung in der Architektur komplexer IT-Landschaften und -Systeme einschließlich Steuerung und Mitarbeit in deren Implementierung. Zudem führt er regelmäßige Architektur-Reviews durch.

Microservices at Scale mit Kubernetes und Istio

Florian Assmus, PRODYNA SE

So wie es aussieht, sind Microservices gekommen, um zu bleiben. Die erhofften Vorteile einer agileren Entwicklung sowie einer verbesserten Wartbarkeit und Skalierung der einzelnen Komponenten wurden jedoch nicht immer erreicht. Grund dafür ist eine in vielen Aspekten signifikant erhöhte Komplexität. Um diese Problematik zumindest auf technischer Ebene zu adressieren, verschieben Kubernetes und Istio große Teile der Komplexität in die Infrastruktur. Der Artikel erklärt die Grundkonzepte beider Komponenten auf einfache Weise und zeigt auf, wie elegant man teils hochkomplexe Problemstellungen lösen kann.

Getreu dem Motto „Divide and Conquer" wurden in der Software-Entwicklung über viele Jahre unterschiedlichste Ansätze für die Aufteilung immer komplexerer Software-Systeme entwickelt. Im Java-Umfeld sind hier vor allem Java EE oder OSGi zu nennen. Microservices führen diesen Ansatz konsequent weiter, indem einzelne Komponenten in dedizierte Prozesse ausgelagert werden und die Kommunikation zwischen den Komponenten ausschließlich über standardisierte Netzwerk-Protokolle und APIs erfolgt.

Was auf der einen Seite die bekannten Vorteile gegenüber einer monolithischen Architektur ausmacht, bringt auf der anderen Seite alle Herausforderungen eines verteilten Systems mit sich. Wir können uns also nicht mehr auf die umfangreichen Garantien unseres Application-Servers verlassen. Plötzlich muss man sich um viele Dinge selbst kümmern. Hier nur ein kleiner Ausschnitt:

- Registrieren und Auffinden von Services
- Robuste und sichere Kommunikation zwischen Services
- Dynamische Skalierung einzelner Services je nach Auslastung
- Verwaltung von Konfigurationen und Geheimnissen (wie Zertifikate und Passwörter)
- Operative Transparenz (Metriken, Logs, Tracing)

Schlüssel-Technologie Software-Container

Mit Docker-Containern wurde ein entscheidender Baustein für die praktische Umsetzung von Microservices einem breiten Nutzerkreis zugänglich gemacht. Software-Container entkoppeln die einzelnen Services vollständig vom zugrunde liegenden Betriebssystem und ermöglichen eine effiziente, in sich geschlossene und interoperable Paketierung und Auslieferung. Zum Verteilen dieser Images und zum Ausführen eines Containers ist letztendlich nur noch eine Container-Laufzeit-Umgebung wie „containerd" (Docker) oder „cri-o" notwendig.

Die für die Umsetzung der Services verwendeten Programmiersprachen und Frameworks spielen bei der Auslieferung und Ausführung letztendlich keine Rolle mehr. Im Rahmen der Open Container Initiative (OCI) entstehen nun offene Industrie-Standards rund um das Format sowie die Ausführung von Containern. Dies macht Software-Container zu einem entscheidenden Baustein auf dem Weg zu einer tragfähigen Microservices-Architektur.

Service-Orchestrierung mit Kubernetes

Mit der allgemeinen Verfügbarkeit von Software-Containern dank Docker und dem Ziel, das eigene Cloud-Geschäft zu stärken, entschied sich Google im Jahr 2014, Kernkonzepte seiner bis dahin streng geheimen Container-Infrastruktur namens „Borg" und „Omega" im Rahmen des Open-Source-Projekts Kubernetes öffentlich zu machen. Kubernetes ermöglicht es, Software-Container dynamisch auf einem Cluster von bis zu fünftausend Servern auszuführen. *Abbildung 1* zeigt eine Übersicht der einzelnen Kubernetes-Komponenten und deren Zusammenspiel.

Die Server in einem Kubernetes-Cluster lassen sich grundsätzlich in zwei Klassen unterteilen. „Master-Server" dienen dem Management des Cluster-Zustands, wohingegen „Worker" für das Ausführen der eigentlichen Services zuständig sind. Die Kontrollschicht von Kubernetes besteht im Wesentlichen aus vier Komponenten:

- **kube-apiserver**
  Der API-Server bildet die zentrale Schnittstelle zwischen allen involvierten Komponenten und dem Nutzer. Er nimmt Zustandsbeschreibungen sowie Status-Informationen entgegen und benachrichtigt andere Komponenten über entsprechende Änderungen.
- **etcd**
  Der konsistente und hochverfügbare Key-Value-Store wird vom API-Server für die Ablage des gesamten Cluster-Zustands genutzt. Der Zugriff erfolgt ausschließlich über den API-Server.
- **kube-scheduler**
  Der Scheduler überwacht den Cluster-Zustand auf neu eingestellte oder nicht zugewiesene Arbeitspakete und selektiert einen passenden Worker-Knoten. Bei der Auswahl werden eine Vielzahl unterschiedlicher Parameter berücksichtigt (wie Ressourcen-Anforderung, Hard- und Software-Einschränkungen, „affinity"- und „anti-affinity"-Spezifikationen etc.).
- **kube-controller-manager**
  Der Controller-Manager enthält eine Reihe unterschiedlicher Controller, die auf Zustandsänderungen im Cluster reagieren. Dies können etwa Änderungen des gewünschten Deployments durch den Nutzer (wie Skalierung, Versions-Update) oder der Wegfall einzelner Knoten sein.

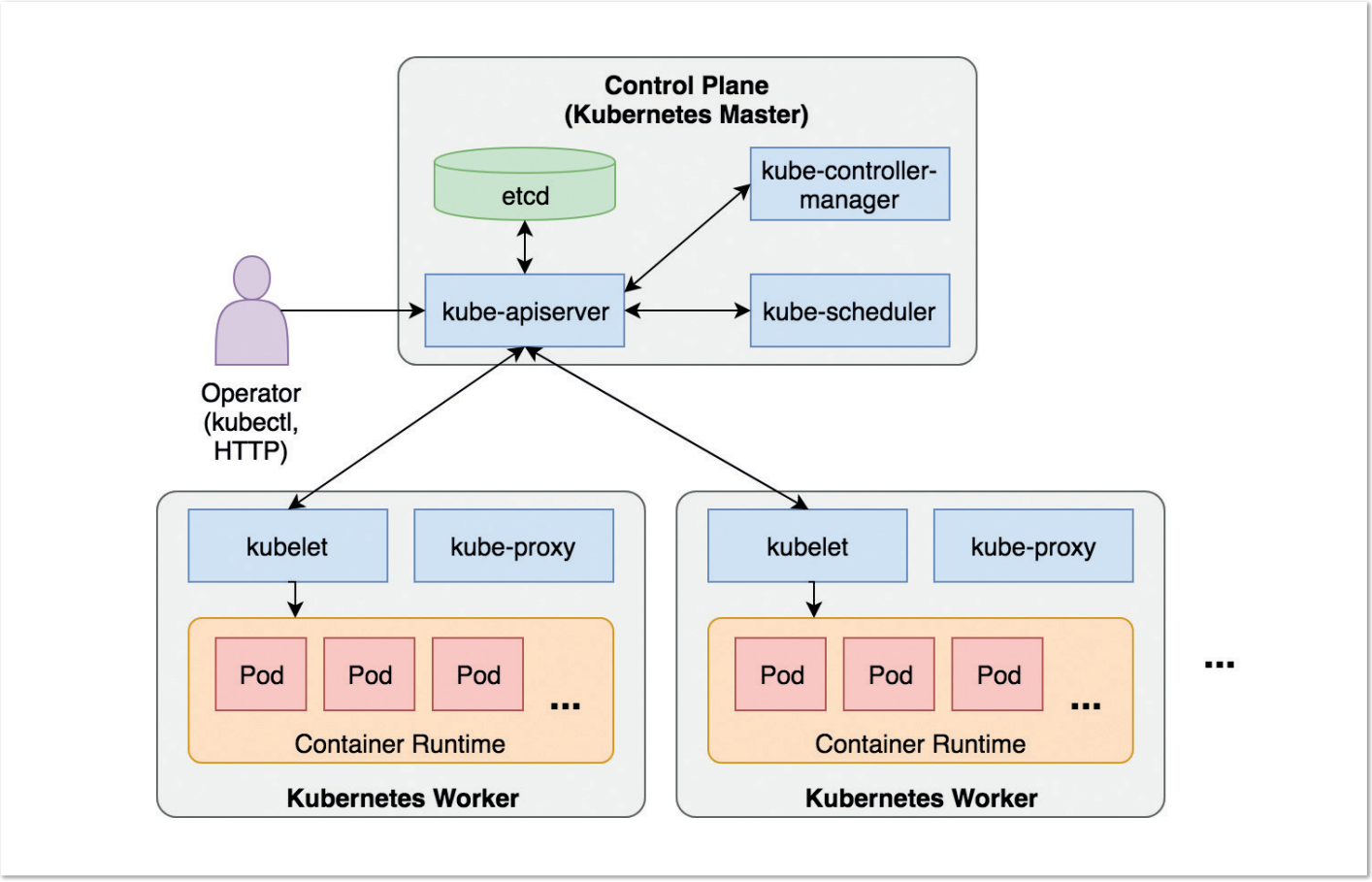


Abbildung 1: Kubernetes High-Level-Architektur

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.29.1
    command: ['sh', '-c', 'echo Hello K8s! && sleep 3600']
```

Listing 1

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Listing 2

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Listing 3

Die Worker-Knoten sind dagegen mit lediglich zwei Komponenten deutlich einfacher aufgebaut. „kubelet" dient als Agent, der die Befehle des API-Servers entgegennimmt und entsprechend Container startet, konfiguriert und wieder stoppt. Dafür interagiert dieser mit der eigentlichen Container-Runtime. Wo anfangs noch direkt Docker angebunden wurde, hat sich nun das Container-Runtime-Interface (CRI) etabliert. Es definiert eine offene und standardisierte Schnittstelle und wird unter anderem von Docker beziehungsweise „containerd", „rkt" und „cri-o" implementiert. Zusätzlich läuft auf jedem Knoten eine „kube-proxy"-Instanz. Sie ermöglicht die Umsetzung eines Service-Konzepts, indem das Netzwerk-Routen und Weiterleitungen zwischen einzelnen Containern im Cluster dynamisch konfiguriert sind.

Der Nutzer stellt seine Services als Container-Images bereit und beschreibt die gewünschte Service-Landschaft deklarativ unter Verwendung der durch Kubernetes definierten API-Objekte. Neben unterschiedlichen Objekten zum Ausführen von Containern bietet Kubernetes unter anderem Konzepte für die Verwaltung von Konfigurationen und Geheimnissen, Netzwerk-Konfiguration inklusive einfacher Lastverteilung sowie der Bereitstellung von persistentem Speicher.

Einmal per API an Kubernetes übergeben, versuchen unterschiedliche Kontrollprozesse kontinuierlich, den gewünschten Zustand herzustellen. Verfügbare Server werden auf ausreichende Ressourcen geprüft und bekommen entsprechend Arbeit zugewiesen. Die Arbeit wird dabei nicht, wie zu vermuten, als einzelner Container definiert, sondern als sogenannter „Pod". Im einfachsten Falle enthält ein Pod auch nur genau einen Container *(siehe Listing 1)*.

Ein Pod kann sich auch aus mehreren Containern zusammensetzen, die als eng gekoppelte Einheit zu verstehen sind. Container in einem Pod laufen immer gemeinsam auf einem Server, teilen sich ein Netzwerk-Interface und haben gleichermaßen Zugriff auf externe Mount-Punkte *(siehe Abbildung 2)*. Pods werden in Kubernetes grundsätzlich als flüchtig betrachtet. Sie bekommen beim Starten eine zufällige IP im Pod-Netzwerk zugewiesen und können im Falle eines Fehlers oder Serverausfalls einfach an anderer Stelle im Cluster neu gestartet werden.

In der Praxis werden jedoch selten Pods durch den Benutzer erstellt. Um eine „Selbstheilung" der Service-Landschaft zu erreichen, implementiert Kubernetes höherwertige Konzepte, die wiederum den Lebenszyklus eines Pod kontrollieren. Alle diese Konzepte folgen einem einheitlichen Muster. Der Nutzer definiert seinen gewünschten

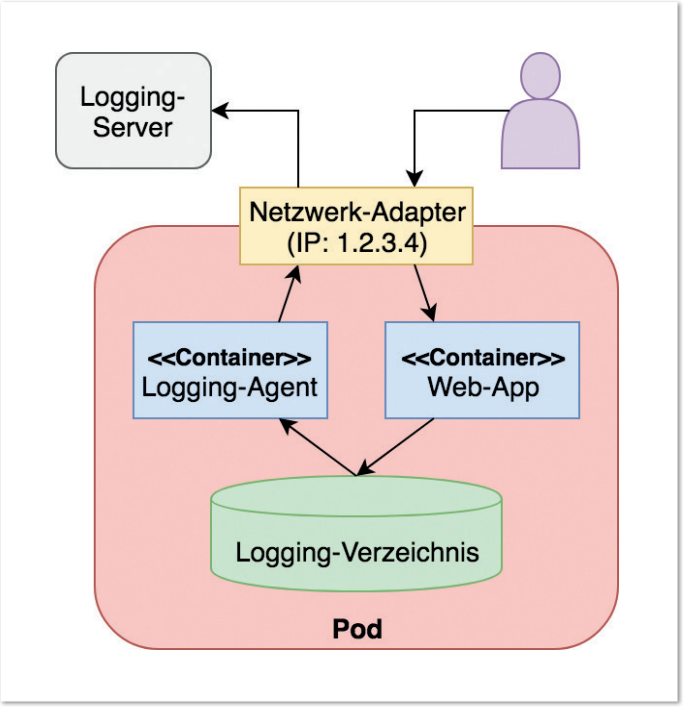


Abbildung 2: Kubernetes Pod

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: k8s.gcr.io/fluentd-elasticsearch:1.20
        volumeMounts:
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Listing 4

Zustand in einem passenden API-Objekt, ein zum Objekt passender Controller überwacht alle Anpassungen an seinen Instanzen und führt die notwendigen Aktionen aus, um den gewünschten Zielzustand zu erreichen. Im Standard bietet Kubernetes unter anderem die folgenden Controller an:

- **Deployment**
  Die genaue Spezifikation eines Pod (Pod-Template) sowie die gewünschte Anzahl der Instanzen lassen sich über ein Deployment-Objekt definieren. Der Deployment-Controller überwacht Änderungen an den Deployment-Objekten und stellt den gewünschten Zustand her. Dafür implementiert der Deployment-Controller unter anderem Funktionen zum kontrollierten Skalieren sowie Aktualisieren (Rolling-Update) der einzelnen Pods. Primär wird das Deployment für zustandslose Services genutzt; Name und IP-Adressen der Pods werden jeweils zufällig gewählt *(siehe Listing 2)*.
- **StatefulSet**
  Das StatefulSet ist speziell für das Deployment von Services mit eigenem Zustand konzipiert worden (wie Datenbanken). So gibt es im Vergleich zum Deployment zusätzliche Garantien hinsichtlich der Reihenfolge, in der Pods gestartet und gestoppt werden, und folgt bei deren Benennung einem stabilen Namensmuster. Zudem stellt das StatefulSet sicher, dass persistente Speicher immer wieder dem passenden Pod zugeordnet sind *(siehe Listing 3)*.
- **DaemonSet**
  DaemonSets stellen sicher, dass auf jedem Knoten im Cluster genau eine Instanz des beschriebenen Pod läuft. Kommt ein neuer Knoten hinzu, wird automatisch ein passender Pod gestartet *(siehe Listing 4)*.

Kubernetes-Services

Wie bereits erwähnt, sind Pods sehr dynamisch, da sie jederzeit auf einen anderen Knoten verschoben werden können und ihre IP-Adressen dynamisch vergeben sind. Damit stellt sich die Frage, wie wir auf unsere Services zugreifen können. Hier setzt das Service-Konzept von Kubernetes an. Mit dem „kube-proxy" und einem Service-API-Objekt lässt sich ein virtueller Service ähnlich einem klassischen Load-Balancer im Cluster definieren. Dieser bekommt einen stabilen DNS-Namen und leitet Anfragen an einen seiner verfügbaren Endpoints weiter. Die Endpoints entsprechen hierbei einem Set von Pods, die mithilfe eines Label-Selektors ermittelt werden. Jeder Pod mit den passenden Labels wird automatisch in die Liste der Endpoints des Service aufgenommen. Kommen Pods hinzu oder fallen weg, wird die Liste dynamisch aktualisiert *(siehe Listing 5)*.

Zusammenfassend bietet Kubernetes eine Plattform, um Services in Form von Containern deklarativ auf einem Cluster von Servern laufen zu lassen. Updates der Services sowie das Hoch- und Runterskalieren von Instanzen übernehmen die unterschiedlichen Controller automatisch. Das Service-Konzept sowie Funktionen zum Verteilen von Konfigurationen oder Geheimnissen implementieren weitere wichtige Bestandteile einer Microservices-Architektur. Darüber hinaus fehlen jedoch Konzepte für eine robuste und sichere Kommunikation und die operative Transparenz wird durch Kubernetes auch nur rudimentär adressiert.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Listing 5

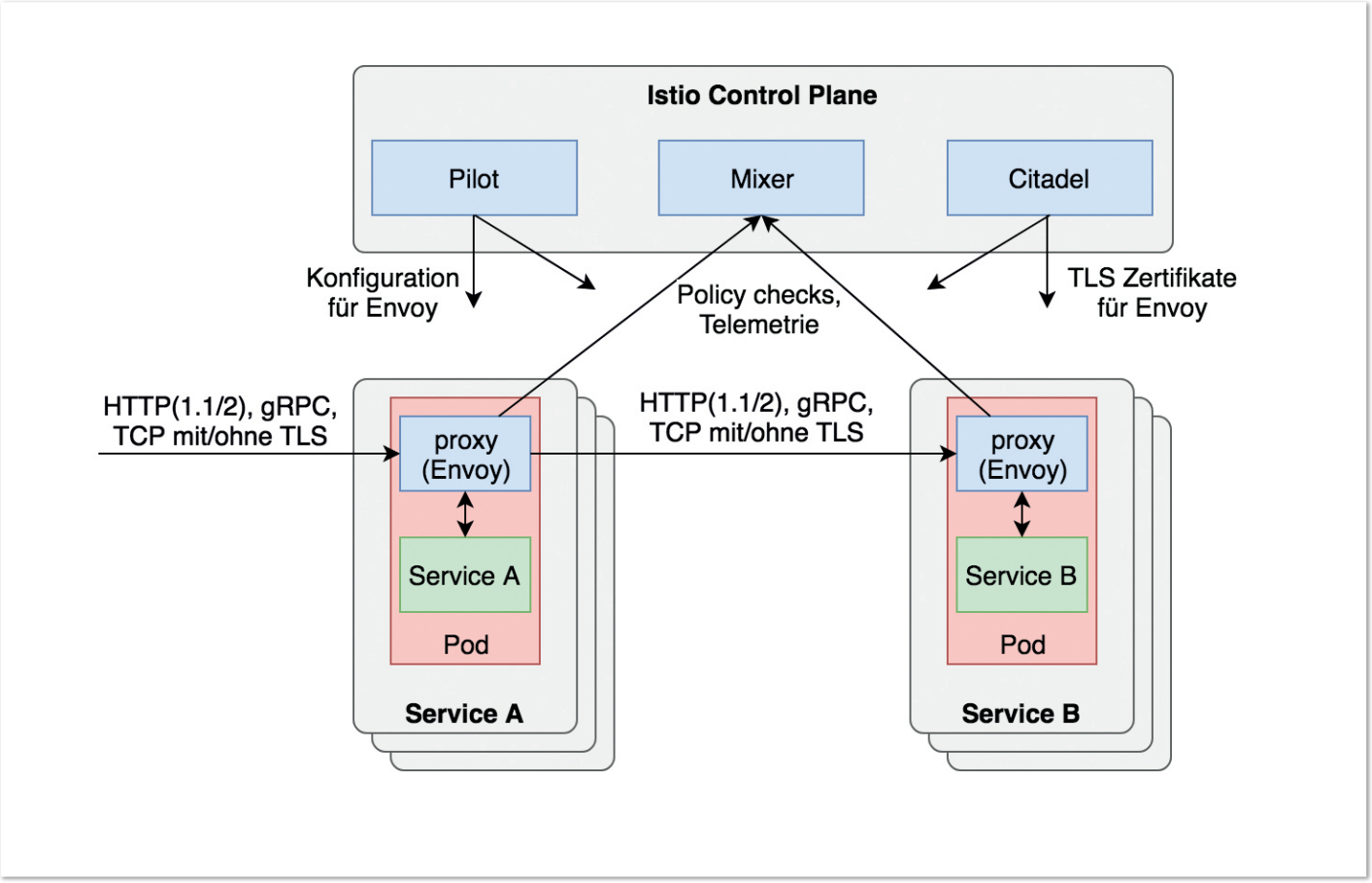


Abbildung 3: Istio-Architektur

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - match:
    - headers:
        cookie:
          regex: "^(.*?;)?(user=jason)(;.*)?$"
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

Listing 6

Istio Service-Mesh

Das, was in Kubernetes noch an Fähigkeiten hinsichtlich einer verteilten Microservices-Architektur fehlt, adressiert das Open-Source-Service-Mesh Istio. Wie die Bezeichnung „Service-Mesh" schon vermuten lässt, betrachtet Istio das gesamte Netz einer verteilten Service-Landschaft und stellt eine ganze Reihe querschnittlicher Funktionen bereit. Istio ermöglicht, die Kommunikation zwischen Services sehr feingranular und dynamisch zu steuern, abzusichern und zu überwachen, ohne dabei die Implementierung der einzelnen Services anzupassen. Möglich wird dies durch den Einsatz eines speziellen „Sidecar"-Proxy namens „Envoy". Dabei erhält jeder Service sozusagen als Beiwagen einen eigenen Proxy, der jegliche eingehende und ausgehende Kommunikation abfängt und anhand von zentral verwalteten Konfigurationen und Regeln weiterleitet *(siehe Abbildung 3)*.

Istio ist sehr stark in Kubernetes integriert, auch wenn andere Service-Deployments möglich sind (wie Consule oder Eureka auf VMs). Das bereits beschriebene Pod-Konzept von Kubernetes lässt sich beispielsweise exzellent für die Umsetzung des Sidecar-Proxy nutzen. Seit Kubernetes 1.9 muss der Proxy nicht einmal mehr explizit in die Pod-Spezifikation aufgenommen werden. Mithilfe eines sogenannten „Admission Controllers" wird der Pod beim Anlegen automatisch modifiziert und um den Proxy erweitert. Innerhalb des Pod wird per IP-Tables der gesamte Netzwerkverkehr auf den Proxy umgeleitet. Für den Entwickler eines Service ist damit das Service-Mesh vollkommen transparent.

Der in Istio eingesetzte Open-Source-Proxy Envoy wurde ursprünglich vom Fahrdienstleister Lyft speziell für ein solches Szenario entwickelt. Envoy ist in C++ geschrieben und weist, neben einer sehr hohen Performance, einen relativ geringen Speicherbedarf auf. Envoy verfügt über ein umfangreiches Konfigurations-API und beherrscht eine ganze Reihe erweiterter Lastverteilungsfunktionen:

- Automatisches Wiederholen von Aufrufen
- Circuit-Breaking
- Globale Bandbreitenbeschränkung
- Zonenbasierte Lastverteilung

Gesteuert werden die einzelnen Envoy-Instanzen, ähnlich wie bei Kubernetes, durch eine zentrale Kontrolleinheit. Diese setzt sich bei Istio aus den im Folgenden beschriebenen Komponenten „Pilot", „Mixer" und „Citadel" zusammen.

Pilot

Pilot ist für die Konfiguration der Envoy-Instanzen zuständig und integriert sich in die zugrunde liegende Service-Plattform *(siehe Abbildung 4)*. Er definiert ein einheitliches Datenmodell für die Beschreibung der Service-Konfigurationen und ein entsprechendes API für die Pflege.

Der jeweilige Plattform-Adapter realisiert die Persistenz der Konfigurationen und interagiert mit der Service-Discovery der Plattform, um Informationen über die aktuelle Service-Landschaft zu erlangen. Im Falle von Kubernetes werden alle Istio-Konfigurationen über den „kube-apiserver" im „etcd" abgelegt. Möglich wird dies durch ein weiteres Konzept von Kubernetes, die Custom-Resource-Definition (CRD). Zudem überwacht Pilot die Kubernetes-Service-Objekte, die als Basis für die erweiterten Istio-Service-Konfigurationen dienen und die Endpunkte der eigentlichen Service-Pods enthalten. Per Konfiguration können so eine Vielzahl von Funktionen unabhängig vom eigentlichen Applikations-Code realisiert werden:

- Anteilige Verteilung von Anfragen auf unterschiedliche Service-Versionen (wie Canary-Releases, schrittweiser Rollout etc.)
- Verteilung von Anfragen auf unterschiedliche Service-Versionen anhand von Inhalten (etwa A/B-Tests)

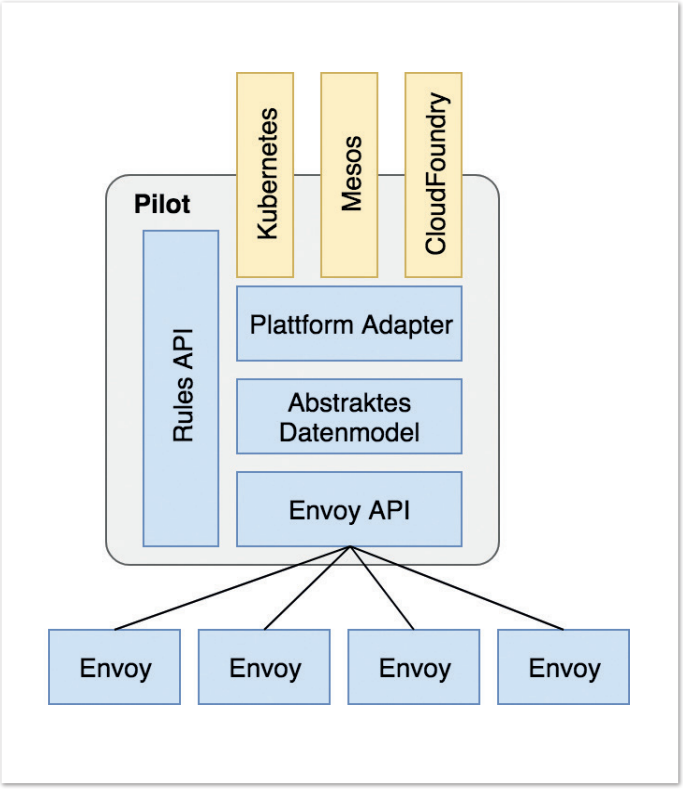


Abbildung 4: Istio-Pilot

- Fehlerbehandlung (wie Timeouts, Retries, Circuit-Breaking)
- Fehler-Injizierung zum Testen der korrekten Fehlerbehandlung

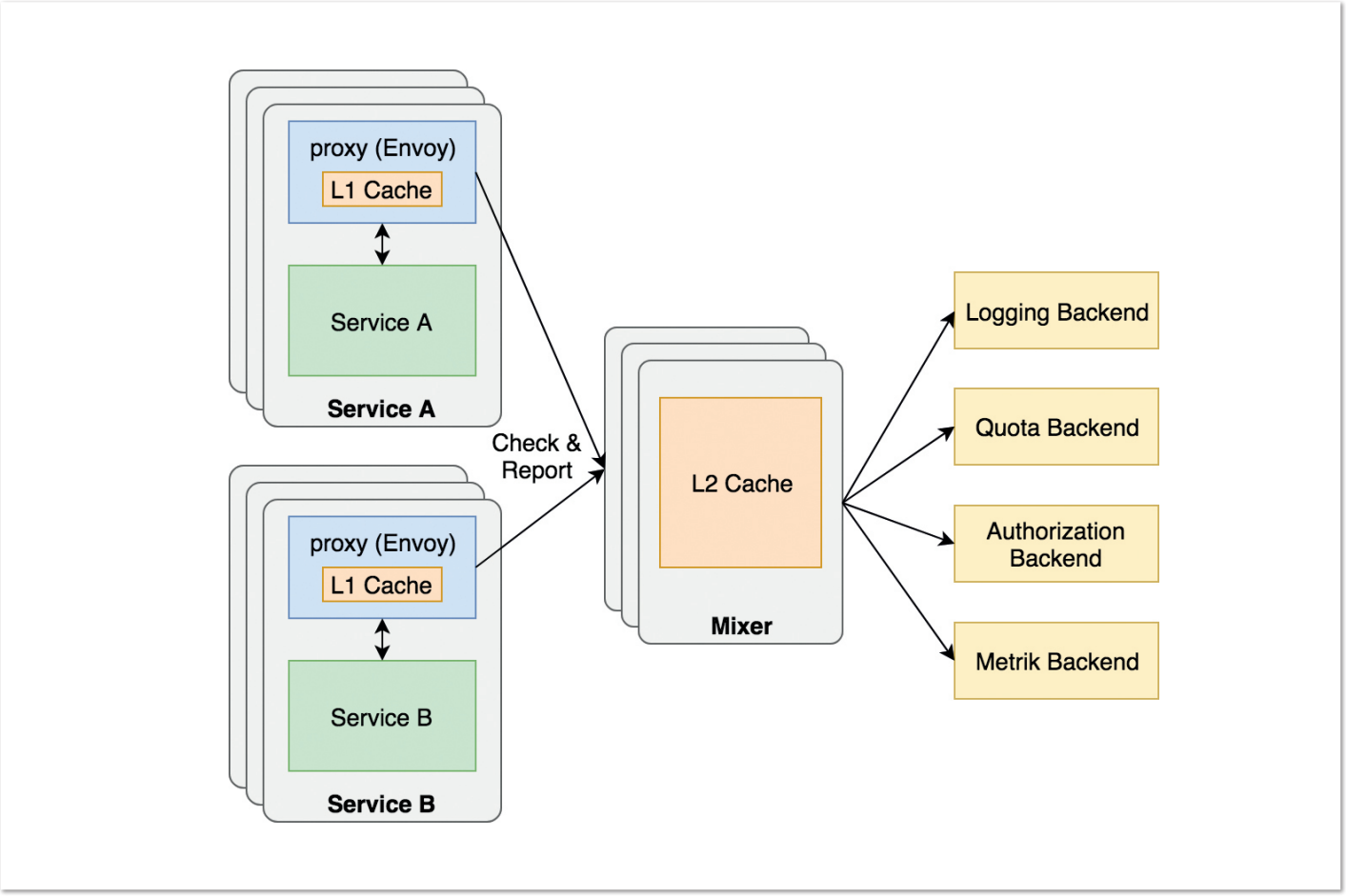


Abbildung 5: Istio-Mixer

Listing 6 zeigt, wie eine neue Service-Version dediziert für einen User aktiviert werden kann. Es handelt sich hierbei um ein sehr einfaches Beispiel, das jedoch die Mächtigkeit von Istio bereits vermuten lässt.

Mixer

Vor und nach jedem Request eines Service konsultieren die Envoy-Instanzen den Istio-Mixer. Vor dem Request prüft Mixer beispielsweise Zugriffsrechte, Quotas oder beliebige weitere Richtlinien. Nach dem Request nimmt Mixer diverse Telemetrie-Daten entgegen und gibt diese an externe Dienste weiter *(siehe Abbildung 5)*.

Mixer wurde für diesen Zweck hochgradig erweiterbar konzipiert. Viele in Istio enthaltene Funktionen, wie Role Based Access Control (RBAC) zwischen Services oder die Anbindung des Monitoring-Systems „Prometheus", sind als Mixer-Plug-ins realisiert. Falls notwendig, können aber auch eigene Systeme angebunden werden.

Da Mixer in dem Gesamtsystem eine sehr zentrale Rolle einnimmt, wurde bei dessen Design vor allem auf Zuverlässigkeit und Latenz geachtet. Neben einem zweistufigen Cache-Konzept ist Mixer „stateless" implementiert und kann somit einfach horizontal skaliert werden. Jede einzelne Mixer-Instanz ist zudem auf eine Verfügbarkeit von höher als 99,999 Prozent ausgelegt.

Citadel

Citadel ist zuständig für eine starke Authentifikation zwischen einzelnen Services sowie dem Benutzer. Istio nutzt beidseitiges TLS (mutual TLS) für die Absicherung der Service-zu-Service-Kommunikation. Dafür stellt Citadel unter anderem die notwendigen Schlüssel-Paare passend zu den Services zur Verfügung. Diese werden automatisch über Kubernetes-Mechanismen in die einzelnen Service-Pods transportiert und dort von Envoy verwendet. Es müssen also nicht mehr manuell Zertifikate verteilt und in die einzelnen Services eingebunden werden. Da die Verschlüsselung erst durch Envoy vorgenommen wird, hat Envoy zudem jederzeit Zugriff auf den Inhalt und kann entsprechende Routing-Regeln anwenden. Für die Benutzer-Authentifikation steht aktuell OAuth 2 mittels JWT zur Verfügung.

Fazit

Der Einsatz von Kubernetes und Istio ermöglicht somit, einen großen Teil der Komplexität einer Microservices-Architektur in die Infrastruktur zu verlagern und gleichzeitig das volle Potenzial einer solchen Architektur zu nutzen. Die für die Entwicklung von fachlichen Services zuständigen Mitarbeiter können sich so voll und ganz auf die Schaffung von Mehrwerten für das Unternehmen konzentrieren. Kombiniert mit organisatorischen Konzepten wie DevOps und einem hohen Grad an Automatisierung, lässt sich so eine deutliche Erhöhung der Entwicklungsgeschwindigkeit und der Flexibilität in der IT-Organisation erreichen.

Florian Assmus
florian.assmus@prodyna.com

Florian Assmus beschäftigt sich seit mehr als fünfzehn Jahren mit professioneller Software-Entwicklung im Konzernumfeld. Seit Abschluss seines Informatik-Studiums in Darmstadt arbeitet er als Berater für die PRODYNA AG und unterstützt diverse mittlere bis große Unternehmen bei der Umsetzung ihrer IT-Projekte als Software- und System-Architekt. Bei PRODYNA leitet er das Kompetenzfeld „Cloud Technologies", das sich primär mit der Transformation aktueller IT Landschaften hin zu einer Cloud-Native-Architektur beschäftigt.

iJUG und Eclipse Foundation beschließen gegenseitige Mitgliedschaft

Im Rahmen der Mitgliederversammlung des iJUG in Stuttgart sprachen sich die Vertreter der deutschsprachigen Java User Groups mit deutlicher Mehrheit dafür aus, dem Antrag der Eclipse Foundation Europe GmbH auf Fördermitgliedschaft im iJUG stattzugeben. Gleichzeitig wurde einstimmig beschlossen, Solutions Member in der Eclipse Foundation zu werden. Somit besitzt der iJUG nun Stimmrecht bezüglich der Zukunft von Jakarta EE, vormals bekannt als Java EE.

„Wir sehen in der Mitgliedschaft in der Eclipse Foundation die historische Gelegenheit, die sich nach der Übertragung von Oracle ergibt, die Weiterentwicklung der Jakarta-EE-Projekte – inklusive dem bereits vorhandenen MicroProfile in dessen neu geplanter Rolle als Inkubator – von Beginn an zu begleiten und so die Verantwortung der Community, in unserem Falle der deutschsprachigen Anwender, aktiv wahrzunehmen", so Jan Westerkamp, im iJUG für die Kooperation verantwortlich. „Wir werden unser Stimmrecht nutzen, um die Eclipse Foundation und Jakarta EE im Sinne unserer Mitglieder voranzubringen, und zwar auf Augenhöhe mit der Industrie", ergänzt sein Stellvertreter Markus Karg von der Java User Group Goldstadt.

„Wir freuen uns, den iJUG als Mitglied der Eclipse Foundation und der Jakarta EE Working Group begrüßen zu dürfen", sagt Mike Milinkovich, Executive Director der Eclipse Foundation. „In Zusammenarbeit mit der iJUG-Community in Deutschland, Österreich und der Schweiz möchten wir offene Innovationen im Cloud Native Java und Erneuerungen innerhalb des Java-Ökosystems vorantreiben."

JVM Con

Die Konferenz für Java-Entwickler

27.–28. November 2018 | Köln, Pullman Cologne Hotel

Als Java aktuell Leser 10% sparen mit Code: JVMCon18doag

Themenauswahl (u.a.):

- Java 10, 11, 12 und darüber hinaus – Große, kleine und noch nicht bestätigte Features
- Erste Schritte mit Neo4j – eine Einführung in OGM
- Wieviel Frontend steckt noch in Java?
- Blue/Green Deployments and Canary Releases on Kubernetes
- Warum Haskell auf der JVM eine gute Idee ist
- Wie aus Java-Anwendungen Container werden

Referenten (u.a.):

Peter Palaga
Red Hat

Stephan Müller
Openknowledge

Rudy De Buscher
C4J

Bernd Rücker
Camunda Services GmbH

Jan Weinschenker
Holisticon AG

Anmeldung und weitere Infos unter www.jvm-con.de #jvmcon18

Sponsoren: GEBIT Solutions, GFU Cyrus AG, Saxonia Systems So geht Software.

Veranstalter: Neue Mediengesellschaft Ulm mbH, developer media


TRIDENT

Von Nixdorf zur Cloud City

Oliver Böhm

Software-Entwicklung: Lerne aus der Vergangenheit, träume von der Zukunft und entwickle in der Gegenwart. Frei nach Katharina [1]

Dies ist die Geschichte einer Firma, die auszog, die Rezept-Abrechnung zu vereinfachen, um Ärzten, Apothekern und anderen Leistungserbringern im Gesundheitswesen (wie Physio-/Ergo-Therapeuten, Logopäden, Hebammen) Freiräume für wichtigere Dinge zu geben. Dazu hat das „Optica Abrechnungszentrum", ein Unternehmen der Dr. Güldener Firmengruppe, bereits Ende der 1970er-/Anfang der 1980er-Jahre früh auf Digitalisierung gesetzt. In einem großen Schreibbüro wurden dabei Abrechnungsdaten auf „Datensammelsystemen" – anfangs auf Lochkarten, später auf Magnetbändern (Nixdorf 8850) – gespeichert.

Damals hieß IT noch EDV. Die Firma Nixdorf (später Siemens-Nixdorf) war ein typischer Vertreter der „Mittleren Datentechnik". Lochkarten beziehungsweise Magnetbänder wurden in die EDV-Abteilung gegeben, um daraus die Abrechnungen zu erstellen und in den Druck zu geben. Netzwerke, wie wir sie heute kennen, gab es damals nicht.

Architektonisch waren die Anwendungen für die Nixdorf 8850 nach dem EVA-Prinzip gestrickt: Eingabe – Verarbeitung – Ausgabe, wobei der Fokus auf der Verarbeitung lag (dem „V" in „EDV"). Schöne Oberflächen waren damals kein Thema – Batchverarbeitung, auch als Dunkelverarbeitung bekannt (etwa im Versicherungsbereich), war Stand der Technik. Es galt, den Datenstrom für die Ein- und Ausgabeverarbeitung nicht abreißen zu lassen, um die teuren Maschinen auszulasten. Üblich waren dabei Bänder, von denen gelesen und auf die das Ergebnis geschrieben wurde, Abrechnungslauf für Abrechnungslauf *(siehe Abbildung 1)*.

Mitte/Ende der 1980er-Jahre etablierten sich dann relationale Datenbanken. Diese dienten nicht nur als zentrale Basis zur Ablage, sondern erlaubten auch die Modellierung von Beziehungen. Eine relationale Datenbank der ersten Stunde war Informix *[2]*, die dort auf einem Unix-Server (Sun Microsystems) zum Zuge kam und nach und nach die Nixdorf-Maschine ablösen sollte.

Wie aber bekommt man für die Übergangsphase die neuen Daten in die „alte Welt", die doch nur Datenströme kennt? Durch (digitale) Transformation – man übersetzt die Tabellen aus der relationalen Welt in Records beziehungsweise Dateien der Nixdorf-Welt. In die andere Richtung (Nixdorf nach „neue Welt") übersetzt man Datenströme in Tabellen. Auf diese Art und Weise konnte man die bestehenden Alt-Anwendungen weiterverwenden, gleichzeitig jedoch auch neue Anwendungen auf Basis einer Client-Server-Architektur mit damals zeitgemäßer Oberfläche entwickeln *(siehe Abbildung 2)*.

Mit zunehmender Verbreitung der Unix-Maschinen in den 1990er-Jahren wurde das Internet mit Erfindung des World Wide Web sichtbarer. Damit war die Welt außerhalb des eigenen Netzwerks immer interessanter, sodass man bereits im Jahr 1997 innerhalb der Dr. Güldener-Gruppe mit einer eigenen Homepage präsent war, über die elektronisch Rezepte abgegeben werden konnten.

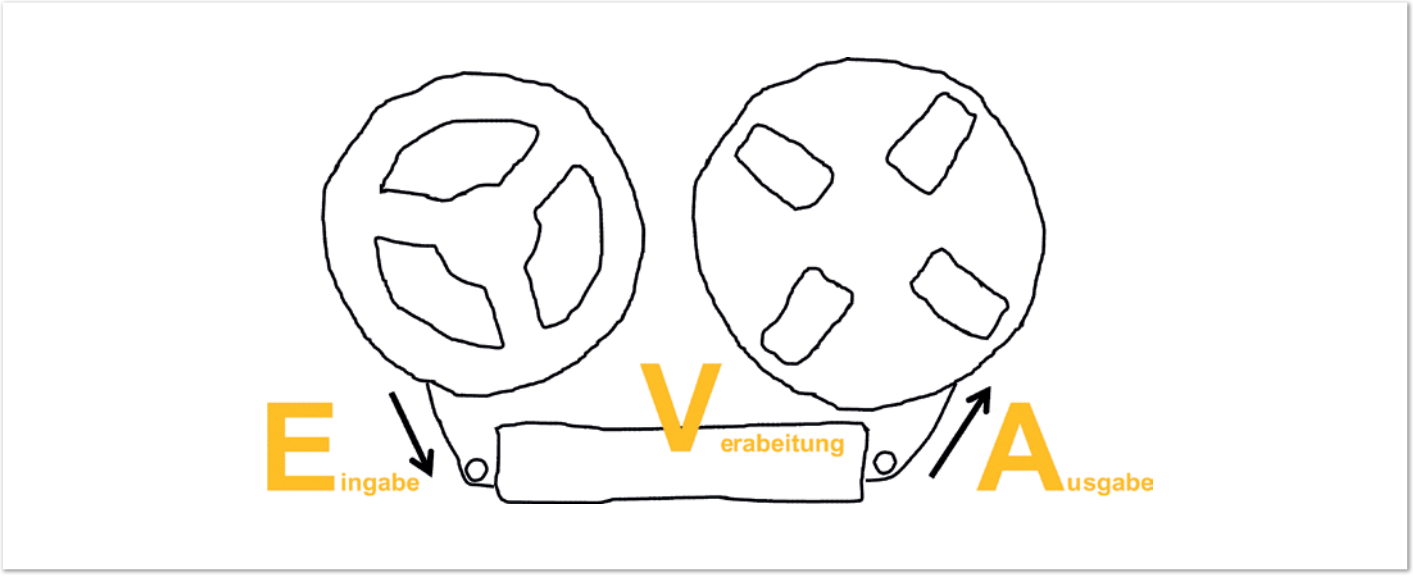


Abbildung 1: Die typische Architektur von damals

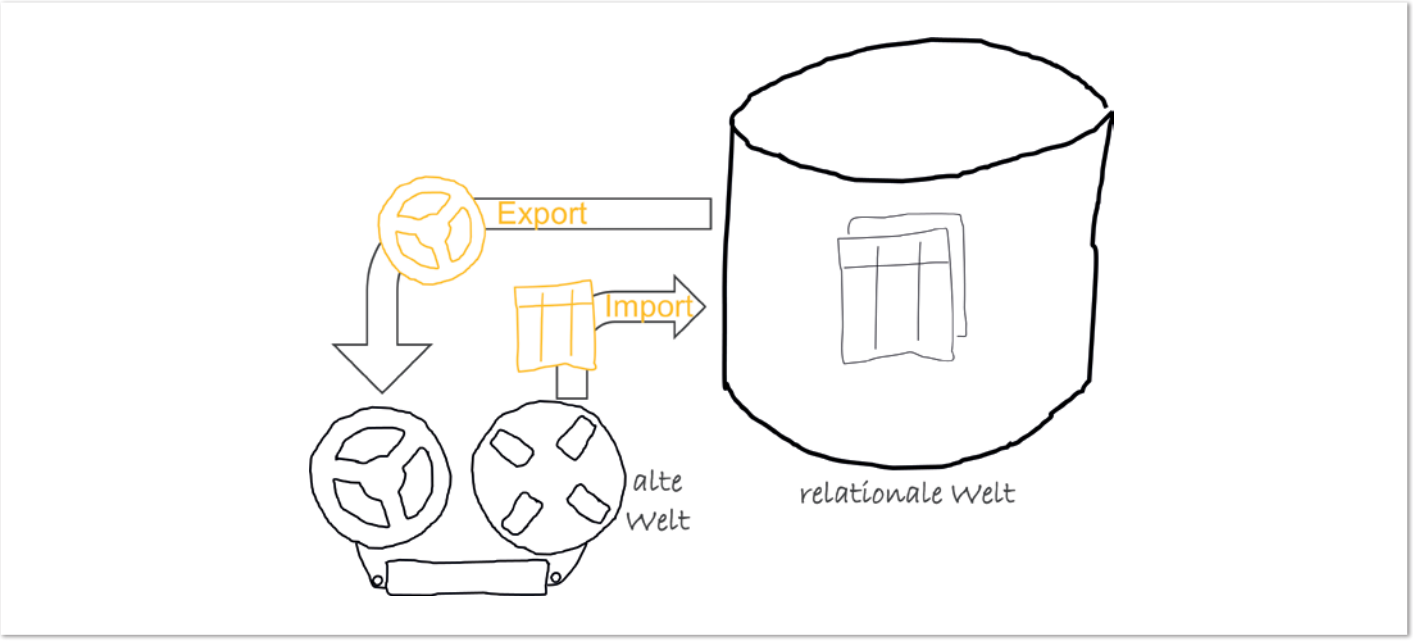


Abbildung 2: Die Verbindung zwischen „alter" und „neuer" Welt

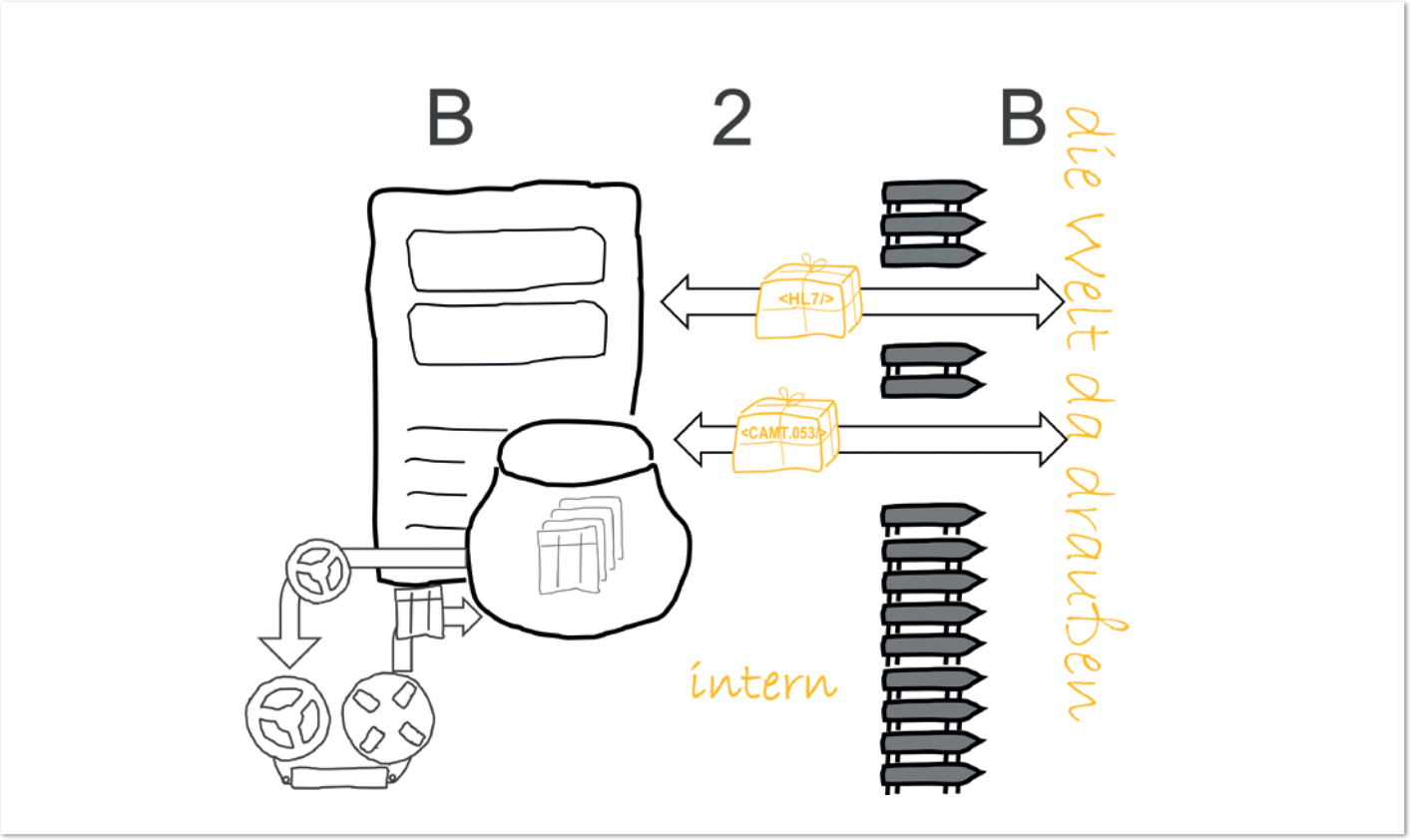


Abbildung 3: Service Oriented Architecture

Mit dem Internet-Hype und Aufstieg des neuen Markts wurden große Hoffnungen in B2C- und B2B-Schnittstellen gesetzt. Auch wenn die Dotcom-Blase Anfang der 2000er-Jahre platzte, hat sich diese Idee in Form von Service Oriented Architecture (SOA) verfestigt und neue Geschäftsfelder erschlossen. Die Datenbank rückte weiter in den Mittelpunkt vieler Anwendungen und wurde ständig erweitert und angepasst *(siehe Abbildung 3)*. Allmählich drohte sie allerdings, zum Flaschenhals zu werden, auch weil der DB-Optimierer an seine Grenzen stieß.

Datenbanken können geclustert werden, was aber nicht zwangsläufig einen Performance-Gewinn bedeutet. Performance-Steigerungen sind insbesondere davon abhängig, ob sich die Daten anhand der Zugriffe aufteilen lassen. Dies führte dann zu grundlegenden Überlegungen, wie eine zukünftige Architektur für die nächsten Jahrzehnte aussehen könnte.

Ein Resultat dieser Überlegungen war die Aufteilung der Anwendungslandschaft nach Domain Driven Design in verschiedene Berei-

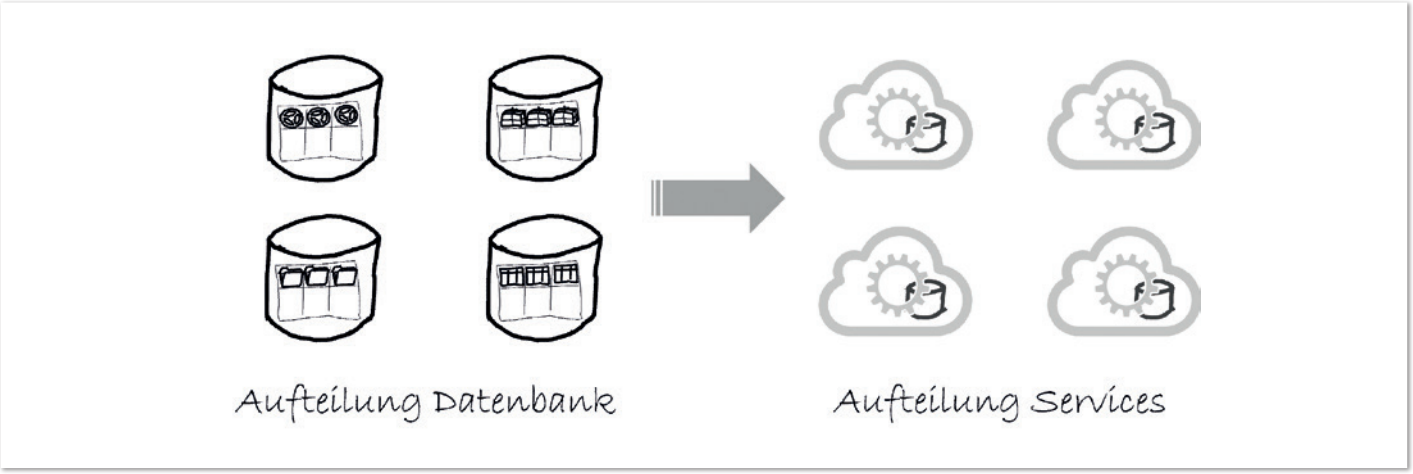


Abbildung 4: Domain Driven Design

che (Stammdaten, Rechnungen, Rezepte), in der jede Domain sein eigenes Datenbank-Schema in Postgres-Datenbanken bekommen sollte. Um damit für die Zukunft gewappnet zu sein, packt man noch einen REST-Service davor, über den alle Anfragen und Abfragen laufen. Damit wird man von der Datenbank unabhängig und kann sie später beispielsweise gegen eine NoSQL-Datenbank austauschen *(siehe Abbildung 4)*.

Eine REST-Schnittstelle hat jedoch noch andere Vorteile: Im Zeitalter der Virtualisierung und Cloud lässt sich die Schnittstelle auch mehrfach starten, wenn die Antwortzeiten zu langsam werden (falls die Datenbank nicht der Flaschenhals ist). Die Architektur ändert sich damit zu einer Microservice-Architektur, die sich einfacher skalieren lässt, allerdings auch höhere Anforderungen an den Betrieb stellt.

Aber nicht nur der Betrieb, auch die Entwicklung wird anspruchsvoller. Nicht umsonst hat Sun Microsystems, Erfinder von Java, aber auch des Slogans „The Network is the Computer", in den 1990er-Jahren vor den Mythen der Netzwerk-Programmierung gewarnt:

1. Netzwerk fällt nie aus
2. Latenz = 0
3. Bandbreite = unendlich
4. Netzwerk ist sicher

Diese Mythen beziehungsweise Irrtümer von Bill Joy, manchen auch als Vater der C-Shell bekannt, wurden im Jahr 1994 von L. Peter Deutsch ergänzt:

5. Topologie ändert sich nicht
6. Es gibt einen Administrator
7. Transport-Kosten = 0

Später hat James Gosling noch hinzugefügt:

8. Netzwerk ist homogen.

Natürlich kann das Netzwerk ausfallen oder zu nicht akzeptablen Wartezeiten führen. Glücklicherweise gibt es dazu bereits einige vorgefertigte Bausteine, die den Umgang mit diesen Herausforderungen vereinfachen. In unserem Fall haben wir uns neben Spring als Basis-Framework für folgende Komponenten entschieden:

- **Keycloak (SSO)**
  Bei Keycloak handelt es sich um eine Single-sign-on-Lösung von Red Hat, die OAuth2 unterstützt und die Anmeldung und Authentifizierung eines Benutzers abnimmt. Die Absicherung der Kommunikation erfolgt dabei über Tokens, die mit jedem Request weitergereicht und auf Gültigkeit geprüft werden.
- **Eureka (Registry)**
  Eureka von Netflix erlaubt die Registrierung von Microservices. Zusammen mit Zuul als API-Gateway (siehe unten) ist es die einzige Adresse, die ein Microservice kennen muss, um auf andere Microservices zugreifen zu können.
- **Zuul (API-Gateway)**
  Zuul dient als API-Gateway und leitet eingehende Anfragen auf den entsprechenden Microservice weiter. Er ist ein wichtiger Baustein für die Skalierung von Microservices, da er Server-seitig Load Balancing mithilfe von anpassbaren Filter- und Routing-Regeln unterstützt.
- **Ribbon + Feign**
  Ribbon ist ein Client-seitiger Load Balancer, der seine Informationen von Eureka bezieht. Feign wiederum erleichtert die Deklaration von REST-Schnittstellen, die dann mit Ribbon kombiniert werden können.
- **Hystrix**
  Hystrix ist die Netflix-Implementierung des Circuit Breaker Pattern [3]. Wenn ein Microservice ausfällt, wird der Aufruf nicht mehr weitergeleitet und auf einen Timeout gewartet, sondern gleich der Fehler (oder ein Fallback) zurückgemeldet.
- **Hystrix-Dashboard/Turbine**
  Das Hystrix-Dashboard zusammen mit Turbine visualisiert die Anzahl von Anfragen und Fehlerfällen – sozusagen das EKG der Microservices.
- **Sleuth + Zipkin**
  Während Sleuth verteiltes Tracing über verschiedene Log-Dateien unterstützt, kann Zipkin diese visualisieren. Dies ist ein wichtiges Hilfsmittel für die Fehlersuche.
- **Spring Boot Actuator**
  Das Actuator-Modul bietet Endpunkte an [4], die über den Gesundheitszustand eines Microservice Auskunft geben.
- **Spring Boot Admin**
  Spring Boot Admin bietet die passende Oberfläche, um sich auf einen Blick eine schnelle Übersicht über die einzelnen Services zu verschaffen.

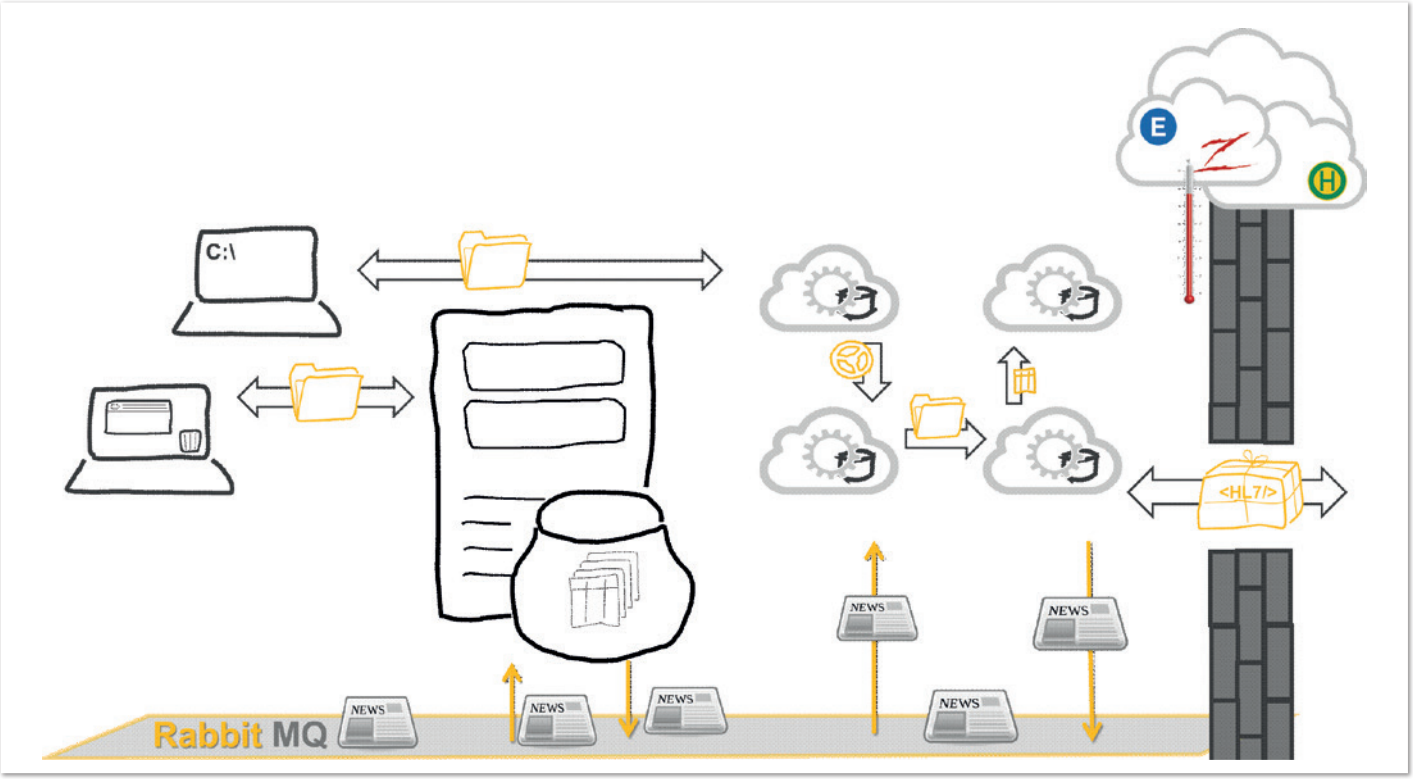


Abbildung 5: Ein Blick in die Zukunft

Dies sind nur einige Bausteine, auf deren Basis aktuell Microservices entwickelt werden, die dann nach und nach bestehende Anwendungen ergänzen oder ersetzen sollen. Zur Ausfallsicherheit werden wichtige zentrale Dienste wie Eureka oder Zuul geclustert sowie durch regelmäßige Last- und Stress-Tests die geforderten Antwortzeiten überprüft. Dies dient auch zur Abschätzung, welche Services mehrfach gestartet werden müssen und wie sich das System beim Ausfall von Services verhält.

Zurück zum Status quo: Während die alte (Nixdorf-)Welt bereits in der Vergangenheit abgelöst wurde (bis auf einige wenige antiquarische Sonderfälle), sind wir jetzt dabei, die mittelalte (Client/Server-)Welt mit ihren starren Strukturen und Zünften zu verlassen, überlieferte Prozesse zu überdenken und nach und nach einzelne Anwendungen auszusortieren oder in die Neuzeit zu überführen. Dies bedeutet in vielen Fällen eine Neuentwicklung von schlankeren Oberflächen auf Basis moderner Web-Technologien, über die die neuen Microservices angesprochen werden. Es bedeutet allerdings auch, dass die Entwickler sich weiterentwickeln müssen und dafür auch Schulungen eingeplant werden.

Spannend wird es während der Übergangsphase: Bis unsere Informix-Datenbank abgeschaltet werden kann, werden wir eine redundante Datenhaltung in beiden Welten haben. Dies ist die größte Herausforderung bei der Modernisierung der Anwendungslandschaft. Technisch wird das über Nachrichten auf Basis von RabbitMQ gelöst, die bei jeder Datenänderung versendet (per Trigger auf der Datenbank und von den Microservices selbst) und von der Gegenstelle (Microservice oder vorgeschalteter Sync-Service für die „alte" Datenbank) empfangen werden *(siehe Abbildung 5)*.

Wer mehr dazu wissen oder gar mitmachen will, darf mich gerne ansprechen. Nächste Gelegenheit dazu wird im September die BED-Con in Berlin sein oder nächstes Jahr wieder im Juli das Java Forum Stuttgart – vielleicht mit einem neuen Wetterbericht aus Cloud City: „Wolkig mit Aussicht auf Fleischbällchen" *[5]*.

Weitere Informationen

[1] *https://www.spruch-archiv.com/completelist/seite-2/action-vote/?query=Vergangenheit+Gegenwart+Zukunft&id=14783&vote=10&pos=15&key=13ljzUhECR&sid=4bac20aa1123325ec08780f21485aa1f#pos15*
[2] *https://de.wikipedia.org/wiki/Informix*
[3] *https://de.wikipedia.org/wiki/Sicherung_(Entwurfsmuster)*
[4] *https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html*
[5] *https://www.javaland.eu/formes/pubfiles/9974384/2018-nn-holly_cummins-cloudy_with_a_chance_of_meatballs__cloud_surprises_for_the_java_developer_-praesentation.pdf*

Oliver Böhm
ob@jugs.org

Oliver Böhm beschäftigt sich mit Java-Entwicklung unter Linux und Aspekt-Orientierter SW-Entwicklung. Neben seiner hauptberuflichen Tätigkeit als JEE-Architekt bei T-Systems ist er Buchautor, Projektleiter bei PatternTesting und Board-Mitglied der Java User Group Stuttgart.

ZURICH

Die Zukunft geschlossener Ecosystems ist ungewiss.

Aktuell dominieren Plattformen und Portale, da diese die Endkunden auf unterschiedlichste Kanäle erreichen.

Unternehmen können sich via APIs diesem Trend öffnen und wieder ihrer Kernkompetenz zuwenden.

Der Markt ändert sich

25,2%[1] der Kunden buchen ihre Übernachtung über ein Online Portal. Tendenz steigend! Amazons Marktanteil im E-Commerce liegt bei über 50%[2]. Tendenz steigend! 18,5%[3] der KFZ-Versicherungen werden online abgeschlossen. Tendenz steigend!

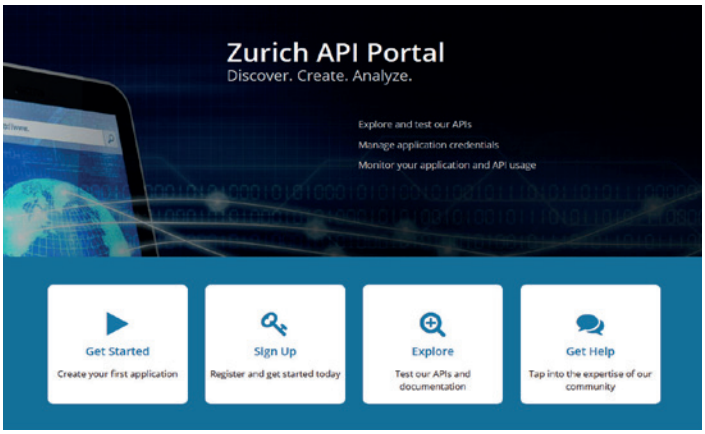
Was haben diese Beispiele gemeinsam? Sie zeigen den Siegeszug von Portalen: Der zentrale Anlaufpunkt für Kunden, die nach einem Produkt suchen. Sie sind in Form von Apps, Websites und Skills für alle möglichen Plattformen verfügbar. Im Hintergrund nutzen Portale sogenannte APIs der Anbieter (Hotels, Modelabel etc.). Sie aggregieren und visualisieren die Ergebnisse. Ermöglichen teilweise den Kauf, ohne dass der Kunde je die echte Anbieterseite gesehen hat.

Unternehmen können sich auf ihr Kerngeschäft fokussieren.

Hotels wissen, wie man Gäste beherbergt. Produkthersteller wissen wie man Trends erkennt und entsprechende Produkte entwickelt. Versicherungen sind Profis bei Risikokalkulation und Compliance. Aber sind sie auch Profis für mobile Apps? Für Alexa-Skills und gute IT-Usability? APIs bieten die Möglichkeit, sich auf die eigene Kernkompetenz zu fokussieren. Das gilt für Hersteller, wie auch Portalbetreiber.

Der Schlüssel zum Erfolg sind APIs

Unternehmen wie Trivago, Amazon und Check24 verstehen sich auf den digitalen Kunden. Sie wissen wie Apps und attraktive Websites gestaltet werden. Sie haben den direkten Kundenkontakt, ohne effektiv etwas zu produzieren. Auch ein Großteil der Fintechs arbeitet so: Sie haben immer wieder neue und coole Ideen. Sie nutzen moderne Technologien und Trends wie Big Data, AI und Machine Learning, um attraktive Zwischen-Produkte zu entwickeln. Aber sie alle sind weder ein Hotel, noch ein T-Shirt-Hersteller oder ein Versicherer.
APIs bieten genau hier die Möglichkeit, sowohl mit den etablierten Marktteilnehmern als auch den aufgehenden Sternen der Start-up Szene zusammenzuarbeiten und damit dem Kunden maßgeschneiderte Gesamtlösungen anzubieten.
Und die Kunden erwarten das zunehmend: Maximale User Experience auf IT-Seite bei gleichzeitiger Perfektionierung der Kernkompetenz der Hersteller: Gute Zimmer, gute T-Shirts und gute Versicherungen über tolle Plattformen. Die Zukunft liegt folglich im Aufbau einer API Economy, die allen beteiligten ermöglicht, effizient und effektiv Dienstleistungen anzubieten und Synergien zu nutzen.



Mission Identified: Aufbau eines API-Ecosystems in und für Versicherungen

Für Zurich ist es daher die zentrale Aufgabe der nächsten Jahre, eine komplexe Anwendungslandschaft zu APIfizieren und damit für die Zusammenarbeit mit Partnern, Insurtechs und Vergleichportalen zu öffnen. Und das auf Basis höchster Sicherheit und moderner Programmierung.

---

[1] https://www.ahgz.de/news/vertriebsstudie-buchungsportale-generieren-ein-viertel-aller-uebernachtungen,200012238751.html

[2] https://t3n.de/news/amazon-erzeugt-2017-rund-53-934948/

[3] https://www.gdv.de/de/themen/news/kunden-versichern-auto-haeufiger-online-11534

OpenAPM – make your own solution

Alicia Bondanza, NovaTec Consulting GmbH

Application Performance Management (APM) ist ein wesentlicher Bestandteil der heutigen IT-Landschaft. Bei der Beratungstätigkeit der NovaTec Consulting GmbH trifft man meist auf kommerzielle APM-Suites. Während der Markt für ganzheitliche, sofort einsatzbereite APM-Werkzeuge sicherlich von kommerziellen Anbietern dominiert wird, bietet die Open-Source-Community zunehmend eine Vielzahl von Technologien, die sich alle durch einen individuellen Aspekt von APM auszeichnen. Neben diesen Technologien hat die Community auch APM-bezogene Standards und Frameworks wie OpenTracing oder OpenCensus ins Leben gerufen.

Die Kombination der Aspekte und Stärken verfügbarer Open-Source-APM-Technologien zur Erstellung einer zugeschnittenen APM-Lösung ist je nach ihrem spezifischen APM-Anwendungsfall lohnenswert. Ein solches kombiniertes System kann eine sinnvolle Alternative gegenüber kommerziellen Lösungen darstellen oder auch kommerzielle Lösungen in einzelnen Aspekten sehr gut ergänzen. Zum Beispiel bietet Grafana hervorragende Dashboard-Funktionen, die weit über alles hinausgehen, was kommerzielle Lösungen bieten können. Wenn solche Schlüssel-Aspekte für den Anwendungsfall wichtig sind, kann eine benutzerdefinierte APM-Lösung die perfekte Antwort sein.

OpenAPM

Kombinieren und Tailoring erfordern jedoch fundiertes Wissen über die Funktionalität und Interoperabilität von Open-Source-Tools, um sie an die spezifischen Kontexte und Anwendungsfälle anzupassen. Da es viele verschiedene Kombinations- und Integrationsmöglichkeiten der Open-Source-Tools untereinander gibt, ist es eine Herausforderung, geeignete Lösungen zusammenzustellen. Dies führt zu einem hohen Aufwand für die Web-Recherche und zu der Notwendigkeit, verschiedene Prototypen auf eine Art und Weise auszutesten, die auf „Trial and Error" beruht. Daher hat die NovaTec GmbH die OpenAPM-Initiative ins Leben gerufen.

OpenAPM fördert die Verwendung offener und maßgeschneiderter APM-Lösungen. Sie konsolidiert und stellt Wissen über die Interoperabilität von relevanten Open-Source-Tools bereit. Darüber

hinaus strebt die OpenAPM-Initiative an, Interoperabilitätslücken zu identifizieren und zu schließen, indem man erforderliche Erweiterungen und Tool-Integrationen sammelt und bereitstellt. Die öffentlich verfügbare, interaktive Landschaft *(siehe „https://openapm.io“)* ermöglicht es, OpenAPM-Lösungen zu skizzieren, indem man die Möglichkeiten der Interoperabilität zwischen relevanten Open-Source-Tools visualisiert.

Um die Einführung einer auf OpenAPM basierenden, angepassten APM-Lösung zu bewerten, sollte man zuerst die verschiedenen Aspekte von APM betrachten und entscheiden, welche für einen gegenwärtigen APM-Anwendungsfall wichtig sind. Es folgt eine Beschreibung der wesentlichen Aspekte, die beim Erstellen von benutzerdefinierten APM-Lösungen berücksichtigt werden müssen.

Aspekte von OpenAPM

Die Idee einer ganzheitlichen APM-Lösung beinhaltet eine Reihe technischer Merkmale und Anforderungen. Auf einer hohen Ebene lässt sich APM als eine Daten-Pipeline betrachten, die in den Aspekten „Datenerfassung“, „Transformation“, „Speicherung“ und „Wertschöpfung“ (Nutzung) aus Performance-Messungen strukturiert ist. OpenAPM ist genau auf diese verschiedenen Aspekte ausgerichtet *(siehe Abbildung 1)*.

Datensammlung

APM verwendet Sonden zur Erfassung von Messungen. Die Instrumentierung passt den Code an, um Laufzeiten zu messen, Methoden-Aufrufe zu korrelieren und Kontextdaten zu extrahieren. Dies geschieht entweder mittels Agenten oder mit Instrumentierungsgerüsten und Bibliotheken:

- Agenten werden der Anwendung beim Start hinzugefügt und als Teil des Anwendungscodes ausgeführt, um die gewünschten Daten zu sammeln. In dynamischen Sprachen (etwa Java, .Net) können Agenten die Instrumentierung sogar zur Laufzeit anpassen.
- Instrumentierung, Frameworks und Bibliotheken sind ein jüngeres Phänomen. Sie werden als direkte Abhängigkeiten in den Anwendungscode integriert. Somit sind Messungen Teil des eigentlichen Quellcodes der Anwendung. Die Instrumentierung wird damit zur Kompilierungszeit angewendet.

Die extrahierten Messdaten werden dann zu Kollektoren gesendet, die oft als Hub für einen bestimmten Satz von Agenten dienen. Nach einer möglichen Stapelverarbeitung der Daten leiten die Kollektoren die Daten an die nachfolgenden APM-Komponenten weiter.

Transformation und Speicherung

Bevor man die Daten analysiert, können die erfassten Daten transformiert werden. Transformationstools empfangen Daten in einem Format, reichern diese an und konvertieren sie potenziell dann in eine andere Darstellung. Beispiele sind die Aggregation von Rohdaten oder die Generierung von Zusatz-Information. Komponenten, die Datentransformationen durchführen, empfangen normalerweise Datenverkehr aus mehreren Quellen und können diese auch an mehrere Zielsysteme senden.

Um die Leistung zu überwachen und sie im Laufe der Zeit zu analysieren, müssen die Leistungsdaten dann auf Festplatte oder Arbeitsspeicher gespeichert werden. In APM findet man normalerweise verschiedene Arten von Langzeitspeicher-Systemen:

- Zeitreihen-Datenbanken eignen sich gut zum Speichern großer Mengen metrischer Daten, beispielsweise aggregierter Antwortzeitmaße. Diese Zeitreihen-Daten lassen sich analysieren und transformieren, um beispielsweise Baselines zu berechnen oder Anomalien zu erkennen.
- Bestimmte NoSQL-Technologien eignen sich zum Speichern großer Mengen von unstrukturierten Performance-Daten wie Traces oder Benutzersitzungsdaten. Diese Art von Daten ist wertvoll für die detaillierte Problem-Analyse sowie für die Analyse des Benutzerverhaltens.

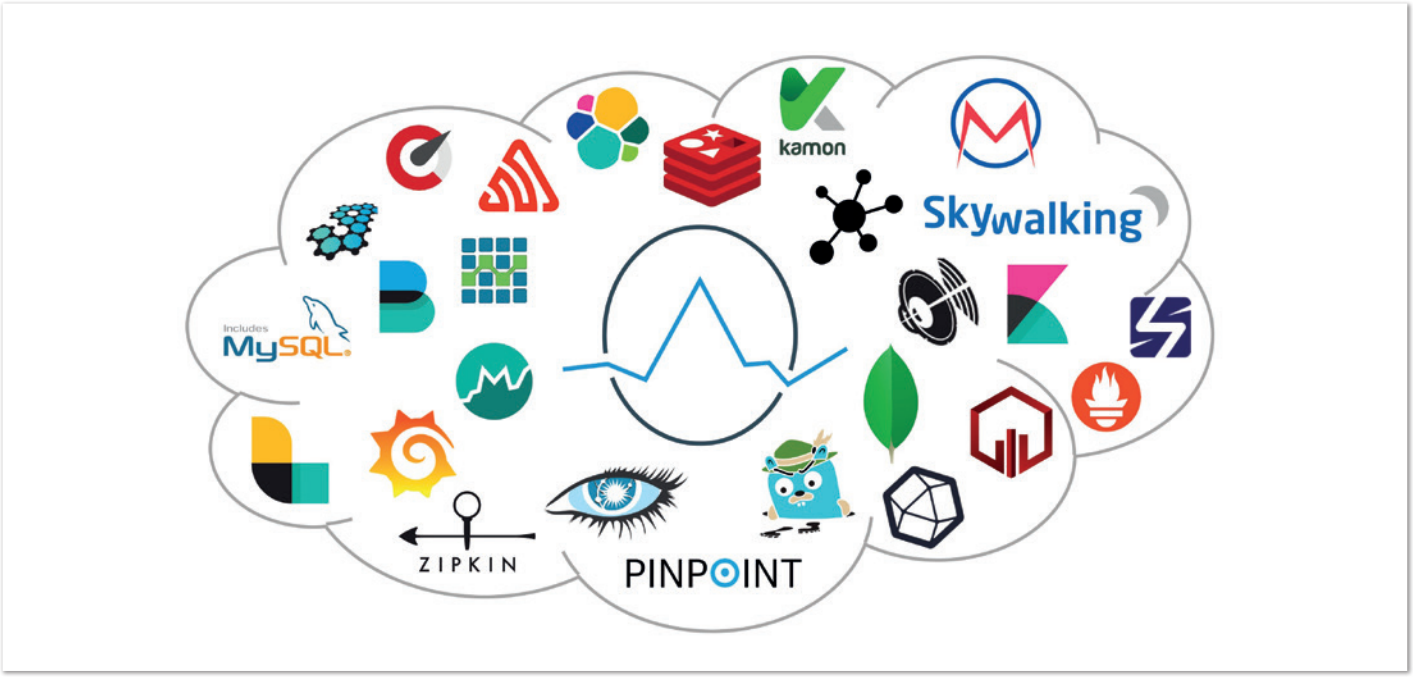


Abbildung 1: Integrierte Tools in OpenAPM

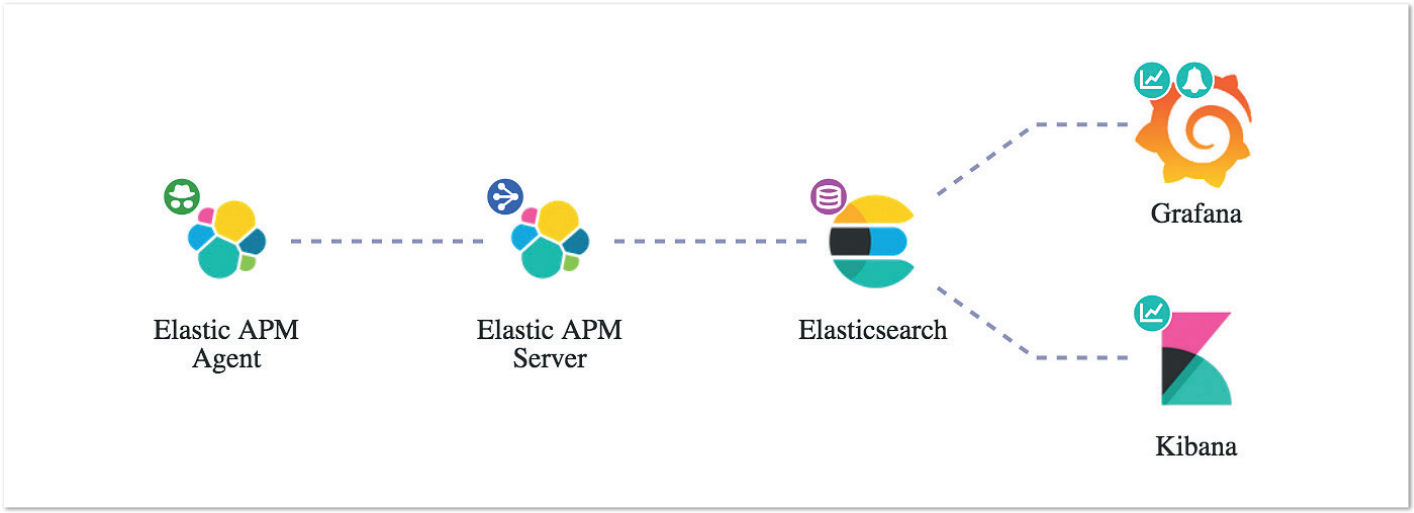


Abbildung 2: Beispiel einer OpenAPM-Lösung

- Relationale Speichersysteme werden auch in APM verwendet, hauptsächlich um Konfigurationsoptionen und Metadaten über die Anwendungslandschaft zu persistieren.

Nutzung und Wertschöpfung

Die Daten stehen jetzt zur Verfügung und zur Nutzung bereit. Um sie zugänglich zu machen, benötigt man zunächst Visualisierungs-Technologien, die Benutzer-Schnittstellen bieten, um mit Leistungsdaten, Konfigurationen, APM-Ergebnissen und Erkenntnissen zu interagieren. In APM decken Visualisierungen typischerweise verschiedene Aspekte ab, etwa die Anzeige von Traces, Metriken oder Anwendungs-Architekturen und Flüssen. Mit Dashboards ist es möglich, bestimmte wichtige Leistungs-Indikatoren, die für ein bestimmtes Thema relevant sind, in einer einzigen Sicht zusammenzufassen, etwa eine bestimmte Anwendung oder einen Geschäftsvorgang.

Ein weiterer typischer APM-Anwendungsfall ist die Alarmierung. Alerting ist der Prozess der kontinuierlichen Überprüfung definierter Bedingungen, basierend auf operativen Leistungsmetriken. Ist eine Bedingung nicht erfüllt, senden APM-Systeme entsprechende Benachrichtigungen über Kanäle wie E-Mail, SMS oder Anwendungen von Drittanbietern, um relevante Interessengruppen zu informieren.

Die Open-APM-Landschaft

Nachdem die typischen Aspekte einer APM-Architektur vorgestellt sind, folgt nun ein Beispiel dafür, wie man mit „openapm.io" benutzerdefinierte APM-Lösungen erstellen kann. Die OpenAPM-Landschaft wird anhand eines Beispiels konkretisiert.

Angenommen, ein Unternehmen verfügt bereits über einen Elastic-Stack. Die Frage stellt sich, welche Open-Source-Tools in den Elastic-Stack integriert werden können, um eine maßgeschneiderte APM-Lösung zu erstellen. Hier kommt Node.js als Programmiersprache für die Anwendungen zum Einsatz. In der OpenAPM-Landschaft besteht die pragmatische Möglichkeit, die Anwendungen zu überwachen, verteilte Traces abzurufen und die

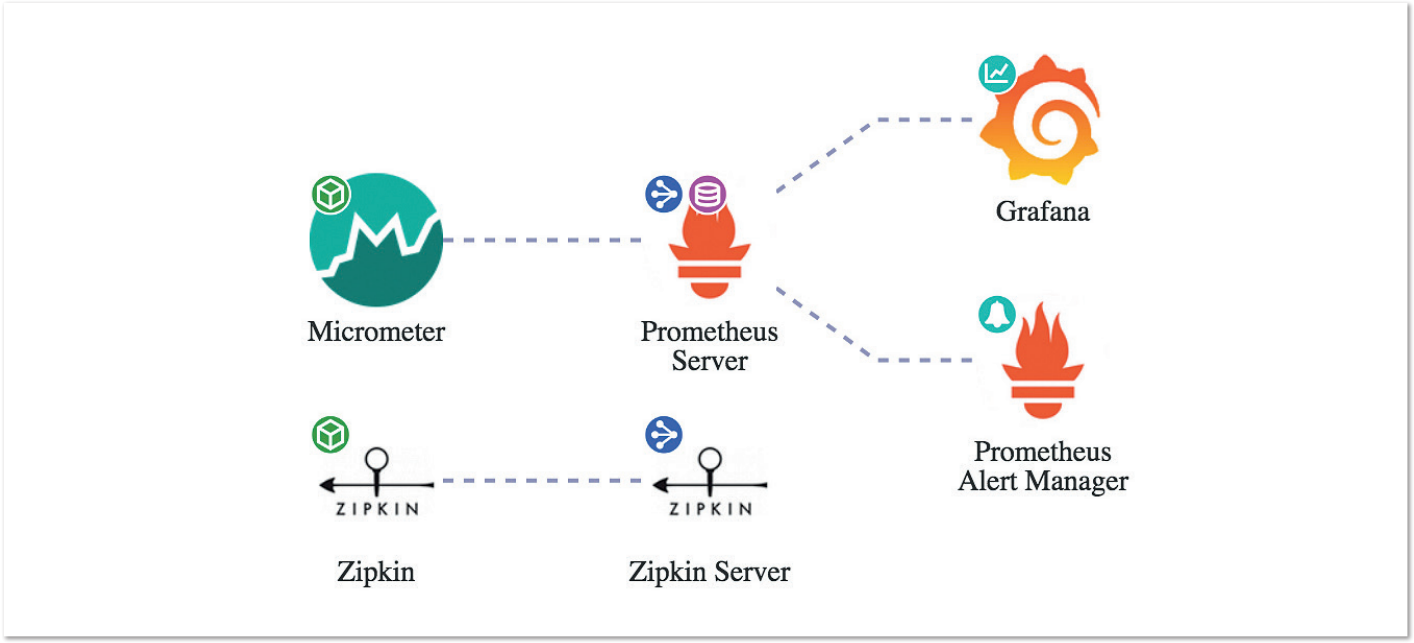


Abbildung 3: Auswahl an Werkzeugen für die APM-Lösung

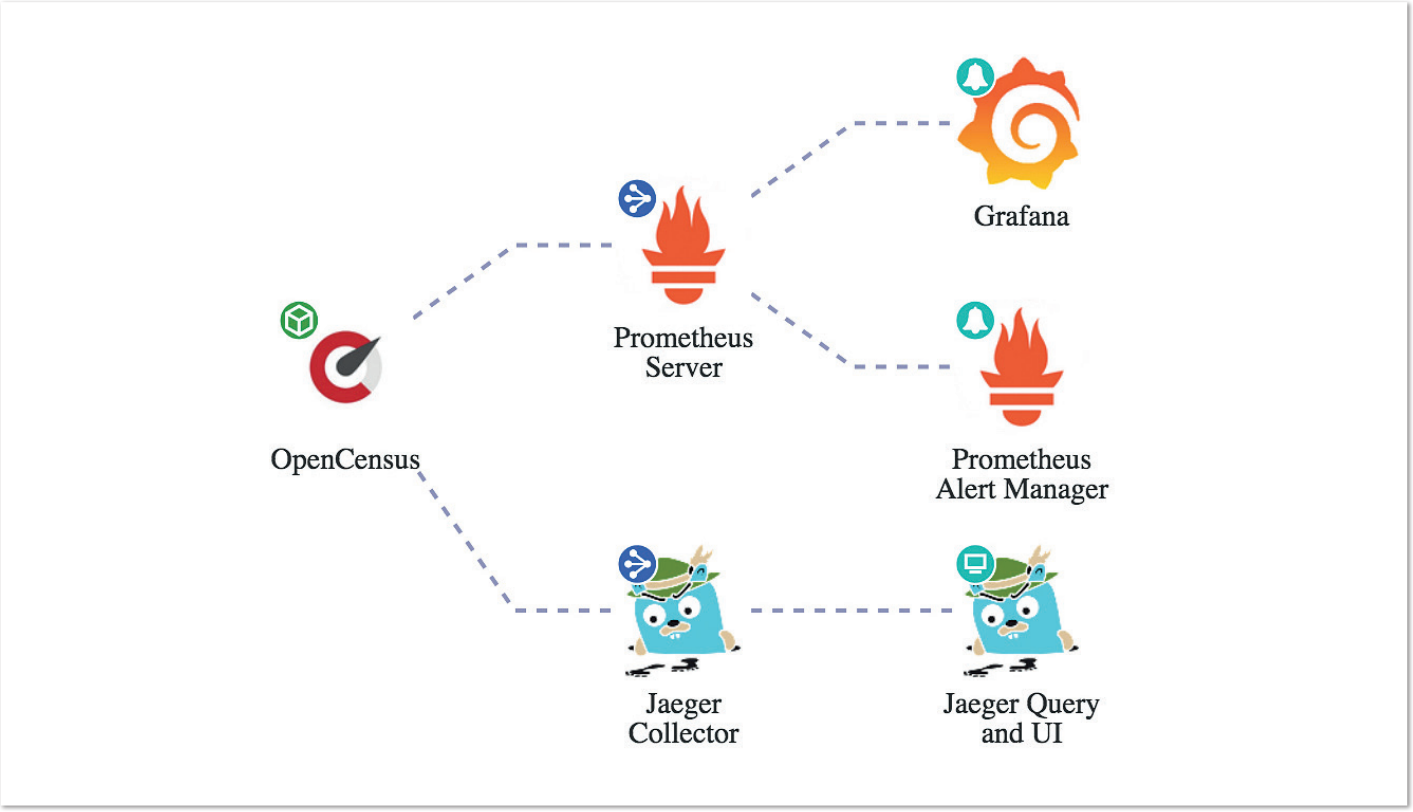


Abbildung 4: Überwachung von Anwendungen

Ergebnisse in Dashboards zu visualisieren. Dazu wählt man zuerst Elasticsearch für die Speicher-Dimension. Danach besteht die Möglichkeit, im Filterbereich nach Node.js und Tracing zu filtern, um nur Tools zu sehen, die in der Lage sind, Tracing-Information aus Node.js-Anwendungen zu erfassen. *Abbildung 2* zeigt eine OpenAPM-Lösung, die zu diesem Beispiel passt. Falls Elastic in Zukunft Java-Monitoring oder Anomaly Detection anbietet, wird die Open-Source-Community die Landschaft entsprechend aktualisieren.

Überwachen von Spring-Boot-Anwendungen

Ein anderes Beispiel ist die Überwachung von Spring-Boot-Anwendungen. Für diese möchte man auch Metriken und Traces überwachen sowie zusätzlich Warnfunktionen integrieren. Nachdem die Landschaft gefiltert wurde, um diese Anforderungen zu berücksichtigen, zeigt sie eine Auswahl von Werkzeugen, aus denen man eine geeignete APM-Lösung zusammenstellen kann *(siehe Abbildung 3)*.

Soll stattdessen die Leistung von in Go geschriebenen Anwendungen überwacht werden, führt das Durchsuchen der OpenAPM-Landschaft möglicherweise zu einer Lösung, wie in *Abbildung 4* dargestellt. Die OpenAPM-Landschaft umfasst auch das Open-Source-APM-Tool inspectIT. Möchte man Java-Anwendungen überwachen, könnte ein typisches Setup ebenfalls so aussehen.

Fazit

Mit diesem reichhaltigen OpenAPM-Toolset kann eine zugeschnittene APM-Lösung erstellt werden, die den individuellen Anforderungen entspricht. Dies erfordert natürlich einen gewissen Anpassungsaufwand, abhängig von den spezifischen APM-Anforderungen. Im Vergleich zu kommerziellen APM-Lösungen erhält man jedoch einzigartige Vorteile:

- Die Lösung besitzen und nicht mehr an die Entscheidungen eines Herstellers gebunden sein. Die Lösung kann flexibel und erweiterbar sein und sich im gewünschten Tempo weiterentwickeln.
- Nur für das bezahlen, was man auch wirklich nutzt. Bei kommerziellen APM-Suites stellt man häufig fest, dass Kunden hohe Lizenzgebühren zahlen und viel Zeit in eine Erstkonfiguration investieren. In der Praxis wird jedoch immer nur ein Bruchteil der verfügbaren Funktionen genutzt.
- Wiederverwendung und Integration von Technologien, die in dem Unternehmen bereits vorhanden sind. Man fügt dem vorhandenen Elastic-Stack eines Teams beispielsweise einen Elastic-Agenten hinzu, um mit dem benutzerdefinierten APM zu starten.

Weitere Informationen unter *„https://openapm.io"*.

Alicia Bondanza
alicia.bondanza@novatec-gmbh.de

Alicia Bondanza ist Digital Content Creator im Bereich Application Performance Management der NovaTec Consulting GmbH. Alicia studiert Sprach- und Textwissenschaften an der Universität Passau und durch ihre Beiträge, bereitet sie einem breiten Publikum das Fachwissen der IT-Berater der NovaTec Consulting GmbH auf.

Zeit ist eine Illusion – Mittagszeit erst recht [1]

Andreas Heigl, bitExpert AG

Das „pling, pling, pling", mit dem Leland Stanford den letzten, goldenen, Nagel in die hölzerne Schwelle treibt und damit den Bau der ersten transkontinentale Eisenbahn der Vereinigten Staaten beendet, ist weit über die Prärie zu hören. Er markiert damit auch den Beginn eines der am meisten gefürchteten Themen nicht nur bei Programmierern: Zeitzonen.

Was haben Eisenbahnen und Zeitzonen miteinander zu tun? Vieles, wie sich gleich herausstellen wird. Im Jahr 1869 sind Eisenbahnen en vogue und für deren Fahrpläne müssen genaue Zeit-Informationen angegeben werden. Aber auf Linien, die in Ost-West-Richtung verlaufen, weichen die Zeitangaben für Fahrpläne und die lokale Zeit voneinander ab; die Zeit am Abfahrtsort plus die Fahrtzeit führt schon nach dreißig Kilometern zu einer Abweichung von einer Minute zur lokalen Zeit – von Frankfurt nach Fulda sind das schon drei Minuten Unterschied. Der Einfachheit halber wurden deshalb die Fahrplanzeiten an der ganzen Strecke auf Basis des Abfahrtsorts angegeben. Das führte in manchen Städten zu Verwirrungen, da unterschiedliche Bahnhöfe in der Stadt nun unterschiedliche Uhrzeiten hatten; so stand in Genf ein Kirchturm, der die Pariser, die Genfer und die Berner Zeit anzeigte.

Jetzt gab es in Amerika eine Eisenbahnlinie, deren Verlauf nicht mehrere Minuten Zeitunterschied überbrücken musste, sondern ganze vier Stunden. Daher trafen sich im Jahr 1883 Vertreter amerikanischer Eisenbahnunternehmen in Chicago und verabschiedeten am 11. Oktober das amerikanische Standard-Zeit-System. Dies sah vor, Amerika in fünf Zonen einzuteilen. Jede Zone hat ihre eigene Zeit, die exakt um eine Stunde von der nächsten Zone abweicht.

So wurde am 18. November 1883 um zwölf Uhr mittags auf dem 90. Breitengrad ein Zeitsignal per Telegraph an alle Stationen übermittelt; dort wurden die Bahnhofsuhren dann auf die jeweilige Zeit ihrer Zone eingestellt. Die Zeitzone hatte das Licht der Welt erblickt *(siehe Abbildung 1)*. Allerdings war dieses System zu diesem Zeitpunkt einzig für die Eisenbahngesellschaften von Inte-

resse und galt nur auf den Bahnhöfen. Die Kirchturmuhren zeigten immer noch die lokale Zeit.

Standardisierung

Um dieses Chaos zu vereinheitlichen, wurden gegen Ende des 19. Jahrhunderts in vielen Staaten Gesetze zur Vereinheitlichung der Zeitbestimmung verabschiedet. Diese orientierten sich in der Masse an der Idee, die Erdoberfläche in 24 Bereiche mit jeweils gleicher Uhrzeit einzuteilen, in denen jeweils dieselbe Zeit gilt und die jeweils eine Stunde Abstand zu den Nachbarzonen hat. So wurde im Jahr 1893 für das Deutsche Reich bestimmt, dass die mittlere Sonnenzeit des 15. Längengrades östlich von Greenwich als gesetzliche Uhrzeit festgelegt wird.

So weit, so gut – eigentlich eine ganz einfache Sache: Deutschland hat seine eigene Zeit und die Nachbarstaaten ebenso. Im Grunde hat jeder Staat seine eigene Zeit. England nutzt die lokale Uhrzeit an der Sternwarte von Greenwich (also am Null-Meridian), Deutschland nutzt die lokale Zeit am 15. Längengrad (also exakt eine Stunde Abweichung vom Null-Meridian), Frankreich, Belgien, die Niederlande, Norwegen, Dänemark und viele andere Staaten nutzen die lokale Zeit der jeweiligen Hauptstadt.

Hier beginnen die Probleme, die viele mit Zeitzonen haben, denn eine Zeitzone ist der Bereich der Erdoberfläche, der durch eine staatliche Regelung die gleiche Zeit hat. Das bedeutet, dass es eine Zeitzone für den europäischen Teil von Frankreich gab, eine weitere für Belgien, wiederum eine eigene für die Niederlande und eine eigene für das Deutsche Reich. Zum damaligen Zeitpunkt betrug der Zeitunterschied zwischen diesen Zeitzonen zum Teil nur wenige Minuten. Das machte die Zeitbestimmung innerhalb der Länder sehr einfach, aber über die Landesgrenzen hinaus nicht.

Dies bringt uns zu einigen Definitionen, mit denen wir im weiteren Verlauf hantieren werden. Zum einen ist da die Zeitzone *(siehe Abbildung 2)*, zum anderen aber die Zonenzeit, die die Zeit innerhalb einer Zeitzone definiert. Außerdem gibt es den Offset, also die Differenz der Zonenzeit gegenüber einer Vergleichszeit. Und dann ist da noch die Zeitzonen-Abkürzung, auf die wir später noch stoßen. Doch mit welcher Zeit werden Zonenzeiten verglichen? Hier müssen wir einen kurzen Abstecher in die Definition der Zeit selbst machen.

Exkursion: Zeit

Im Jahr 1884 wurde in der internationalen Meridiankonferenz der Längengrad, der durch die Sternwarte von Greenwich verläuft, als internationaler Referenz-Längengrad festgelegt. In diesem Zuge wurde auch die lokale Sonnenzeit an dieser Sternwarte als internationale Referenz-Zeit definiert, die Greenwich Meantime (GMT). Diese wurde dann auch als Vergleichszeit für den Offset einer Zeitzone verwendet.

Im Jahr 1955 wurde die Atom-Uhr erfunden, die es erlaubt, Zeit präziser zu messen als über den Sonnenhöchststand. Diese universelle Zeit (UT) definierte eine Sekunde jetzt nicht mehr als den 86.400-sten Teil zwischen zwei Sonnen-Höchstständen an der Sternwarte von Greenwich, sondern als die Zeit, in der ein Cäsium-Atom 9.192.631.770 mal den Spannungszustand ändert. Damit war eine Zeiteinteilung geschaffen, die vollkommen unabhängig von der Sonne ist.

Abbildung 1: Der Start der transkontinentalen Eisenbahn in Amerika

Da die Rotation der Erde allerdings immer langsamer wird, hat dies zur Folge, dass die Atomzeit und die Sonnenzeit nicht mehr deckungsgleich sind. Bis heute weicht die UT daher von der GMT um mehr als dreißig Sekunden ab. Das scheint nicht viel, sollte aber regelmäßig ausgeglichen werden, um nicht im Laufe der Jahrzehnte zu größeren Abweichungen zu kommen. Es führte im Jahr 1960 zur Einführung der „Universellen Zeit – koordiniert mit GMT" (Universal Time, Coordinatet, UTC), die die Abweichung zwischen UTC und GMT auf weniger als eine Sekunde beschränkt. Dazu werden je nach Bedarf zu fixen Terminen Schaltsekunden eingefügt, die letzte der aktuell 37 Schaltsekunden am 31. Dezember 2016. Eine solche Schaltsekunde bedeutet, dass eine Minute tatsächlich statt 60 auch 61 Sekunden haben kann.

Mittlerweile basiert die Zeitmessung in den meisten Staaten auf UTC und der Offset beträgt ganze Stunden, aber es gibt Ausnahmen. So beträgt der Offset der Zeitzone von Indien fünfeinviertel Stunden, der von Nepal fünfdreiviertel Stunden. Nichts spricht gegen einen vollkommen ungeraden Offset, der sogar Sekunden oder Bruchteile von Sekunden enthält.

Die Sommerzeit

Und um dies alles noch interessanter zu gestalten, wurde dann am 30. April 1916 im Deutschen Reich zum ersten Mal die Sommerzeit eingeführt. Um die Sonnenstunden besser ausnutzen und dadurch Energie sparen zu können, wurde an diesem Tag die Uhr eine Stunde vorgestellt, sodass die Helligkeit während des Sommers länger in den Abend hinein dauert. Interessanterweise ändern sich durch diese Sommerzeit nicht die Zeitzone, sondern nur der Offset und dadurch die Zonenzeit. Auch hier muss die Änderung nicht exakt eine Stunde betragen und es spricht auch nichts dagegen, mehr als eine solche Verschiebung zu haben.

Im Jahr 1947 wurde in Deutschland eine Hochsommerzeit eingeführt, die einen Offset von drei Stunden hatte. Es spricht auch nichts dagegen, mehrmals im Jahr zwischen Normalzeit und Sommerzeit zu wechseln. In Marokko beispielsweise begann dieses Jahr am 25. März die Sommerzeit, die aber bereits am 13. Mai wieder endete, nur um am 17. Juni wieder zur Sommerzeit zu wech-

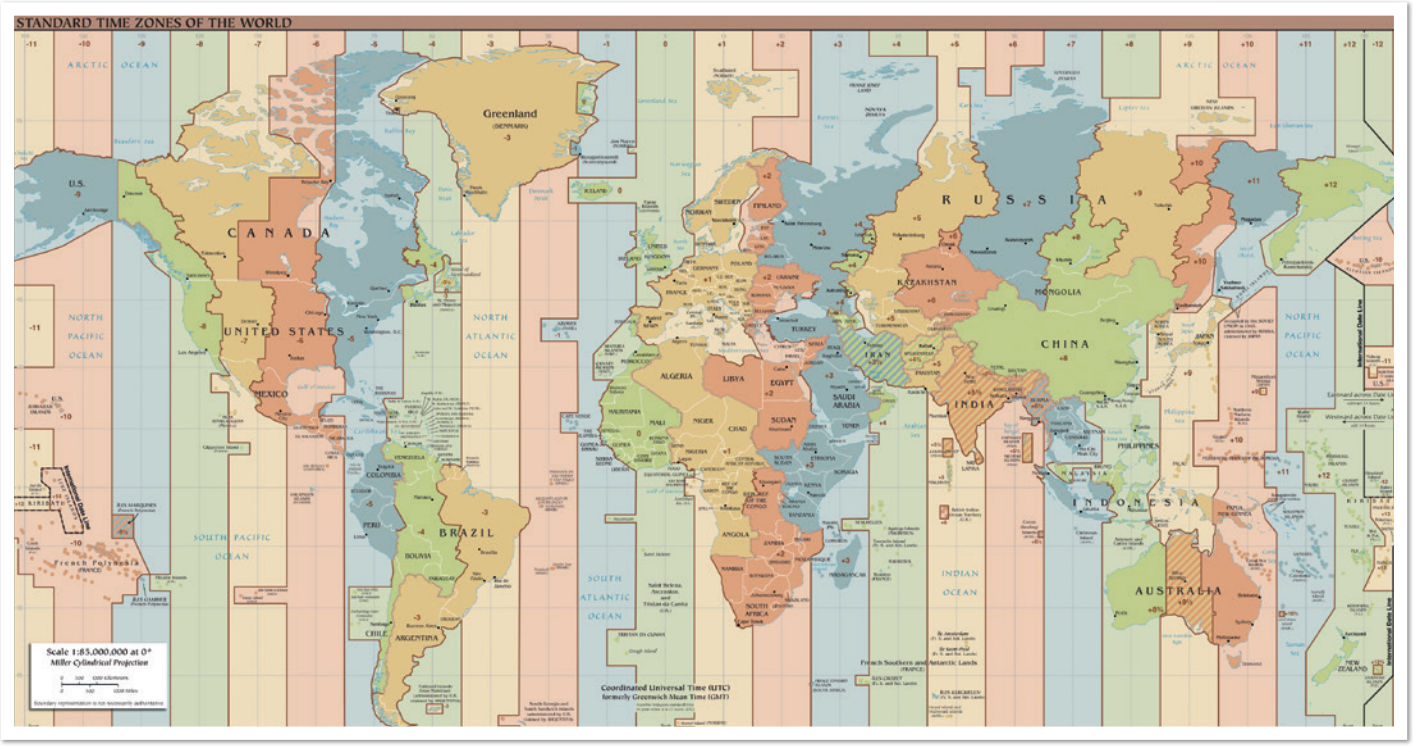


Abbildung 2: Die weltweiten Zeitzonen

seln, die dann bis zum 28. Oktober dauert. Grund dafür ist, dass die überwiegend muslimische Bevölkerung vom 16. Mai bis zum 14. Juni Ramadan feierte.

Beginn und Ende der Sommerzeit unterliegen also ebenfalls der staatlichen Kontrolle und Willkür. Die EU-weit einheitliche Regelung, dass die Sommerzeit am letzten Sonntag im März beginnt und am letzten Sonntag im Oktober endet, besteht erst seit dem Jahr 1996. Aktuell gibt es eine Umfrage der EU, ob diese einheitliche Regelung weiterhin Bestand haben soll oder nicht. In Nordamerika dagegen gelten ganz andere Regeln. Daher kann es sein, dass in den USA bereits Sommerzeit herrscht, in Europa jedoch nicht. Dass die Sommerzeit auf der Südhalbkugel der Erde dann beginnt, wenn sie auf der Nordhalbkugel endet, trägt zusätzlich zur Verwirrung bei.

Dann gibt es auch noch Ausnahmen: So wurde etwa in Deutschland die Sommerzeit im Jahr 1980 wieder eingeführt. Nur ein kleines Dorf unbeugsamer Rebellen hat sich dem verweigert. In der Exklave Büsingen am Hochrhein wurde erst ein Jahr später, zusammen mit der Schweiz, die Sommerzeit einführte. Um dem Rechnung zu tragen, hat Büsingen (ein Dorf mit nicht ganz 1.500 Einwohnern) eine eigene Zeitzone.

Zeitzonen und Computer

Um all diese Informationen und Willkürlichkeiten in Computersystemen abzubilden, gibt es mittlerweile eine Datenbank, die bei der IANA zum Download zur Verfügung steht *[2]*, die sogenannte „Olson-Datenbank", benannt nach ihrem Gründer Arthur David Olson, der das Projekt um das Jahr 1986 herum gründete. Hauptidee war es, eine einheitliche Datengrundlage zu erstellen, die es Programmierern ermöglicht, Zeit-Informationen basierend auf all den beschriebenen Absurditäten möglichst genau zu bestimmen. Dazu wurden die Zeitzonen nach Kontinenten gegliedert und innerhalb der Kontinente nach bekannten Städten benannt. Daher gibt es etwa die Zeitzone „Europe/Amsterdam", obwohl die Hauptstadt der Niederlande Den Haag ist. Die Idee dahinter war, dass Hauptstädte sich ändern können, die bekannten Städte jedoch weniger. Aus diesem Grund wird auch nicht der Name des Landes verwendet. Die Städte werden nach der bekanntesten englischen Schreibweise genutzt, was zum Beispiel dazu führt, dass die Zeitzone „Europe/Rome" heißt und nicht „Europe/Roma".

Für diese Zeitzone sind in der Datenbank auch die (bekannten) Informationen zu Sommer- und Winterzeit hinterlegt. Da diese Datenbank von Freiwilligen als Open-Source-Projekt geführt wird *[3]*, können nur die Informationen enthalten sein, die dem Projekt zu Ohren kommen. Oft genug werden Änderungen auch sehr kurzfristig mitgeteilt.

Dass es Änderungen an der Zeitzonen-Datenbank gibt, steht außer Frage. Zeitzonen ändern sich. Dabei gibt es Gegenden, in denen sich die Zeitzonen häufiger ändern, andere, in denen das seltener ist. Aber es vergeht kein Jahr, in dem nicht zehn oder zwölf Versionen der Zeitzonen-Datenbank erscheinen. Also ist die Frage nicht, „ob" sich eine Zeitzone ändert, sondern „wann".

Neben den Zeitzonen-Namen gibt es in dieser Datenbank auch Abkürzungen. Eine davon ist „CET", was für „Central European Time" steht, die zentraleuropäische (Standard-) Zeit. Daneben gibt es auch noch die Abkürzung „CEST" für die zentraleuropäische Sommerzeit. Dabei können diese Abkürzungen für mehrere Zeitzonen gelten, aber auch mehrere Abkürzungen für eine Zeitzone vorhanden sein. So kann beispielsweise für die Zeitzone „Europe/Berlin" die Abkürzung „CET" oder „CEST" genutzt werden – je nachdem, ob gerade Sommer- oder Standard-Zeit gilt. Allerdings gelten dieselben Abkürzungen auch für „Europe/Amsterdam" oder „Europe/Paris", um nur einige zu nennen. Es kann also nicht aus einer Abkürzung auf eine Zeitzone geschlossen werden. Daher sollten solche Versuche,

so verlockend sie erscheinen, in der Programmierung unterlassen werden. Die Empfehlung ist, diese Abkürzungen lediglich für Darstellungszwecke zu benutzen – wenn überhaupt.

Kleine Randbemerkung an dieser Stelle: Die in den USA so häufig verwendeten Bezeichnungen „Mountain Time" oder „Atlantic Time" sind ebenfalls solche Abkürzungen und enthalten beispielsweise die Zeitzonen „America/Denver" oder „America/New_York". Da aber der amerikanische Bundesstaat Nevada im Gegensatz zum Rest der Vereinigten Staaten keine Sommerzeit nutzt, wechselt die Zeitzone „America/Phoenix" zwischen den Abkürzungen „Mountain Standard Time" und „Pacific Daylight Time" hin und her.

Das Chaos überwinden

Wie können Entwickler dieses Chaos von ständigen Änderungen überwinden? Indem sie bestehende Lösungen nutzen und das Rad nicht neu erfinden! Nichts ist frustrierender, als selbst entwickelte Zeit-Mathematik, die über Sekunden-basierte Multiplikationen mit jeder Sommer- oder Winterzeit-Änderung wieder Fehler produziert. Die Zeitzonen-Datenbank existiert und es gibt wohl kaum eine Programmiersprache, in der es nicht eine entsprechende Umsetzung gibt. Darum sollte diese auch genutzt werden. Diese Bibliotheken werden von Menschen gewartet, die sich sehr intensiv mit der Materie beschäftigt haben, die vielfältigen Ausnahmen kennen und mit ihnen umzugehen wissen. In Java ist dies seit Version 8 das „java.time.*"-Package *[4]*. In vorherigen Versionen sollte die JodaTime-Bibliothek *[5]* verwendet werden.

Aber auch diese Bibliotheken basieren immer wieder im Grunde auf der Zeitzonen-Datenbank. Wenn die letzte Version der Zeitzonen-Datenbank auf dem Computer fünf Jahre alt ist, können auch nur die Umrechnungen vorgenommen werden, die in dieser fünf Jahre alten Datenbank verzeichnet sind. Die Umstellung der aktuellen Zeit von Nord-Korea vom Mai 2018 wird dort ebenso wenig verzeichnet sein wie die Änderung der Zeitzone auf der Krim von „Europe/Kiew" zu „Europe/Simferopol".

Daher ist einer der wichtigsten Grundsätze beim Hantieren mit Zeitzonen, immer aktuell zu bleiben. Die jeweiligen Aktualisierungen der JRE bringen die entsprechenden Zeitzonen-Datenbanken mit sich *[6]*, aber diese Aktualisierungen müssen natürlich auch vorgenommen werden! Alternativ lassen sich auch nur die Zeitzonen-Informationen aktualisieren *[7]*.

Handhabung von Daten

Wie aber können jetzt Daten ohne große Probleme behandelt werden? Oftmals wird der Ratschlag gegeben, alle Daten in UTC umzurechnen, dann kann es keine Probleme geben. Der Autor kann nur eindringlich davor warnen, dies pauschal umzusetzen. Es kommt immer darauf an, um welche Datums-Informationen es sich handelt, denn aktuelle Informationen wie die Datums-Angabe in einer Log-Datei oder das Erstellungsdatum in einem Event-basierten System können anders behandelt werden als Termin-Absprachen in zwei Jahren.

Wie bereits erwähnt, ändert sich die Zeitzonen-Datenbank kontinuierlich und teilweise auch recht unmittelbar. Es gab schon Fälle, in denen den Machern mitgeteilt wurde, dass sich der Offset einer Zeitzone vor 14 Tagen geändert hatte. Daher kann eine Umrechnung von einer Zeitzonen-basierten Datumsangabe nach UTC nur innerhalb von zwei bis drei Wochen um den aktuellen Zeitraum herum vorgenommen werden. Für weiter voraus- (oder auch zurück-)liegende Zeiträume erhöht sich die Wahrscheinlichkeit, dass es Änderungen an der Zeitzone gibt, die dann bedeuten, dass mit den falschen Informationen nach UTC umgerechnet wird *[8]*.

Logging-Informationen können daher meist problemlos nach UTC umgerechnet werden, da es aktuelle, unmittelbare Zeitpunkte betrifft. Hier gilt also der Ratschlag, nach UTC zu konvertieren. Kalender-Informationen hingegen, die weiter in der Zukunft (oder der Vergangenheit) liegen, sollten in der lokalen Zeit zusammen mit der Zeitzonen-Information genutzt werden; dann kann es weniger zu Problemen kommen. 15:00 Uhr in Istanbul wird immer 15:00 Uhr in Istanbul sein, egal, ob die türkische Regierung beschließt, wieder Winterzeit zu nutzen, die Sommerzeit beizubehalten oder den Offset von drei Stunden auf drei Stunden und eine bestimmte Anzahl Minuten zu ändern (das steht nicht zur Debatte, sondern soll lediglich die grundlegende Idee verdeutlichen).

Aber um diese 15:00 Uhr mit anderen Uhrzeiten zu vergleichen, kann nun durch die Zeitzone und die aktuelle Zeitzonen-Datenbank nach den aktuell gültigen Regeln verglichen werden. Übrigens gilt diese Regel auch für das Konvertieren in einen Timestamp. Denn per Definition ist der Timestamp die Anzahl der Sekunden seit 00:00:00 Uhr am 1. Januar 1970 UTC. Daher ist auch die Konvertierung in einen Timestamp eine Konvertierung nach UTC.

Datenbanken

Irgendwann müssen die Informationen auch dauerhaft gespeichert werden, meistens in Datenbanken. Um sowohl Zeitzone als auch Datum zu speichern, empfiehlt es sich, zwei Felder zu nutzen: eines mit lokaler Zeit für das Datum und ein String-Feld für die Zeitzone. In MySQL lässt sich das wie in *Listing 1* anlegen. Dann kann ein Datum mit der Anweisung in *Listing 2* in die Datenbank geschrieben werden. Aus der Datenbank lässt es sich wie in *Listing 3* wieder auslesen.

Für andere Datenbanken kann dieses Beispiel analog übernommen werden. Wichtig ist jedoch, dass das „DATETIME"-Feld nur die Datums- und Uhrzeit-Informationen enthält, nicht aber einen Offset. Auch ein „TIMESTAMP" sollte nicht verwendet werden, da dies den lokalen „DateTime"-Wert in einen Timestamp und damit nach UTC konvertiert. Vorsicht vor der oft irreführenden Bezeichnung „Datetime with timezone" (etwa in PostgreSQL). Dieses Datumsfeld enthält keine Zeitzone, wie der Name suggeriert, sondern speichert den Offset mit ab, was wiederum zu Ungenauigkeiten führen kann.

„DateTime"-Werte in Datenbanken vergleichen

Wenn ein Timestamp oder ein nach UTC konvertierter Wert in der

```
CREATE TABLE `datetime` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `zeit` datetime DEFAULT NULL,
  `zone` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

Listing 1

```
ZonedDateTime date = ZonedDateTime.now();
// z.B. 2018-07-15T07:11:03+02:00[Europe/Berlin]

// the mysql insert statement
String query = "INSERT INTO datetime (zeit, zone) VALUES (?, ?)";

// create the mysql insert preparedstatement
PreparedStatement preparedStmt = conn.prepareStatement(query);
preparedStmt.setString(1, date.format(
    DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")
));
preparedStmt.setString(2, date.getZone().toString());
preparedStmt.exexute();
```

Listing 2

```
String select = "SELECT zeit, zone FROM datetime";
PreparedStatement preparedSelect = conn.prepareStatement(query);
ResultSet result = preparedSelect.executeQuery(select);

DateTimeFormatter pattern = DateTimeFormatter.ofPattern(
  "yyyy-MM-dd HH:mm:ss"
);

while (result.next()) {
  LocalDateTime datetime = LocalDateTime.parse(
     result.getString("zeit"),
     formatter
  );
  ZoneId zone = ZoneId.of(rs.getString("zone")

  ZonedDateTime zdatetime = ZonedDateTime.of(datetime, zone);

  System.out.println(
     zdatetime.format(DateTimeFormatter.ISO_DATE_TIME) +
     " - " +
     zdatetime.withZoneSameInstant(ZoneId.of("UTC"))
              .format(DateTimeFormatter.ISO_DATE_TIME)
  );
  // 2018-07-15T07:11:03+02:00[Europe/Berlin] - 2018-07-15T05:11:03Z[UTC]
}
```

Listing 3

Datenbank gespeichert wird, sind Vergleiche über dieses Feld einfach möglich. Wie aber soll das funktionieren, wenn Datum/Uhrzeit und Zeitzone in verschiedenen Feldern gespeichert sind? Hier bringen die meisten Datenbanken dankenswerterweise eigene Zeitzonen-Operationen mit, die diese Vergleiche vereinfachen. Damit ist es möglich, die Werte mit den jeweils aktuellen Zeitzonen-Informationen umzurechnen und zu vergleichen. Werte aus der gerade gezeigten Tabelle können beispielsweise in MySQL wie in *Listing 4* mit einem gegebenen Wert verglichen werden *[9]*. Dieselbe Abfrage in PostgreSQL sieht dagegen wie in *Listing 5* aus *[10]*.

Vorsicht mit solchen Zeitzonen-Operationen in Datenbanken, da die Funktionen je nach Datenbank unterschiedlich oder eventuell gar nicht vorhanden sind. Auch muss etwa MySQL erst für diese Operationen mit „mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql“ vorbereitet (und nach jeder Aktualisierung der Zeitzonen-Informationen aktualisiert) werden.

Zeitzone ermitteln

Wie aber lässt sich die Zeitzone eines Benutzers ermitteln, die man dann in seiner Anwendung speichert? Gibt es da eine einfache Methode, um dem Benutzer das lästige Eingeben abzunehmen? Antwort: Das kommt auf die Anwendung an. In einer Web-Anwendung ist dies nicht zweifelsfrei möglich. Die „moment.js“-Bibliothek *[11]* bietet zum Beispiel die Möglichkeit, die aktuelle Zeitzone zu schätzen, aber dies ist nicht immer zuverlässig, da der Browser weder

```
SELECT * FROM `datetime` WHERE
    CONVERT_TZ(zeit, zone, 'UTC') >= "2018-07-15 12:00:00";
```

Listing 4

```
SELECT * FROM datetime WHERE
    zeit AT TIME ZONE zone AT TIME ZONE 'UTC' >= '2018-07-15 12:00:00'
```

Listing 5

```
curl http://api.timezonedb.com/?lat=50&lng=8&format=json&key=xxxx | jq
{
  "gmtOffset" : "7200",
  "countryCode" : "DE",
  "status" : "OK",
  "zoneName" : "Europe/Berlin",
  "timestamp" : 1528458213,
  "dst" : "1",
  "message" : "",
  "abbreviation" : "CEST"
}
```

Listing 6

ungefragt auf die Geolocation noch direkt auf die Zeitzone des Systems zugreifen kann, was bei nativen Anwendungen dagegen möglich ist. Dort kann also beispielsweise die aktuell eingestellte Zeitzone des Systems mit „java.time.ZoneId.systemDefault().toString();" ermittelt werden.

Doch auch hier wird lediglich die systemweit eingestellte Information abgerufen, die aber nicht mit der lokalen Gegebenheit übereinstimmen muss. So wird ein Laptop auf jedem Flughafen dieser Welt die Zeitzone „Europe/Berlin" zurückgeben. Um also die echte lokale Zeitzone zu ermitteln, kann es hilfreich sein, eine externe Schnittstelle zu nutzen, die anhand der Geo-Koordinaten die gerade aktuelle Zeitzone ermittelt. Ein Beispiel ist das „timezonedb"-API *[12]*, das wie in *Listing 6* genutzt werden kann.

Allerdings ist die Frage zu stellen, ob dies tatsächlich die Zeitzone ist, mit der der Benutzer normalerweise agiert. Daher sollte es immer die Möglichkeit geben, eine Zeitzone für einen Eintrag nachträglich zu verändern. Besonders hilfreich wird dies etwa beim Hinterlegen von Reise-Informationen – besonders für Flüge, da hier oft Abflug- und Ankunftszeit in unterschiedlichen Zeitzonen liegen. Im iCalendar-Standard *[13]* ist deshalb explizit vorgesehen, dass Start- und End-Zeitpunkt für einen Termin jeweils eigene Zeitzonen haben.

Zum Abschluss nochmal als kurze Zusammenfassung einige Vorschläge, die das Hantieren mit Zeitzonen vereinfachen sollen.

Bitte nicht

- Versuchen, von einem Offset oder einer Zeitzonen-Abkürzung auf eine Zeitzone zu schließen.
- Interessante Sekunden-Arithmetik im Code oder in der Datenbank versuchen. Eine Minute kann mehr als 60 Sekunden haben, ein Tag mehr oder weniger als 24 Stunden und nicht jeder Monat hat 30 Tage.
- Den Offset oder die Zeitzonenabkürzung in der Datenbank anstelle der Zeitzone speichern. Dadurch gehen Informationen verloren.
- Je nach Anwendungszweck kann es schwierig sein, mit Zeitzonen-Funktionen der Datenbank zu arbeiten, besonders wenn auch Datenbanken unterstützt werden müssen, die keine Zeitzonen-Informationen (oder ausschließlich Offset-Informationen) speichern.
- Lokale Uhrzeiten nach UTC konvertieren. Dabei gehen wertvolle Informationen verloren. Dies gilt nicht für Logging-Informationen, da diese zum aktuellen Zeitpunkt anfallen und mit den aktuell gültigen Zeitzonen-Informationen problemlos umgerechnet werden können.

Aber gerne

- UTC für Logging verwenden. Dies gilt auch für alle anderen Arten von Daten, die zur aktuellen Zeit anfallen und entsprechend umgerechnet werden können, wie Erstellungs- oder Änderungszeitpunkte.
- Die lokale Zeitinformation ohne Offset und den Zeitzonen-Namen gemeinsam abspeichern (für Logging-Informationen kann auf den Zeitzonen-Namen auch verzichtet werden).
- Bei Web-Anwendungen zum Rendern auf entsprechende Bibliotheken wie momentjs *[14]* zurückgreifen.

Um die Zusammenfassung zusammenzufassen: Datums- und Uhrzeit-Informationen ohne Zeitzone sind wie Geldbeträge ohne Währung.

Weitere Informationen

[1] Douglas Adams, Per Anhalter durch die Galaxis
[2] *https://www.iana.org/time-zones*
[3] *https://github.com/eggert/tz*
[4] *https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html*
[5] *http://www.joda.org/joda-time*
[6] *http://www.oracle.com/technetwork/java/javase/tzdata-versions-138805.html*
[7] *http://www.oracle.com/technetwork/java/javase/tzupdater-readme-136440.html*
[8] *https://andreas.heigl.org/2016/12/22/why-not-to-convert-a-datetime-to-timestamp*
[9] *https://andreas.heigl.org/2016/04/18/timezones-and-mysql*
[10] *https://andreas.heigl.org/2016/05/29/timezones-and-postgresql*
[11] *https://momentjs.com/timezone*
[12] *https://timezonedb.com*
[13] *https://tools.ietf.org/html/rfc5545*
[14] *https://momentjs.com*

Andreas Heigl
andreas@heigl.org

Andreas Heigl arbeitet als Software-Entwickler bei der bitExpert AG in Mannheim. Er co-organisiert die Frankfurter PHP-Usergroup und hält Vorträge auf nationalen und internationalen Konferenzen sowie Usergroup-Treffen.

Machine Learning mit H2O und Java

Stephan Schiffner und Dr. Jonathan Boidol, Steadforce

Schätzungsweise 60 Prozent aller Machine-Learning- und Analytics-Projekte in Unternehmen schaffen es nicht über Experiment- oder Pilotphasen hinaus [1]. Eine der Herausforderungen ist die Distanz zwischen den vertrauten Entwicklungsumgebungen von Statistikern, Analytics-Experten und Data Scientists, hinzu kommt der üblichen Technologie-Stack von Software-Entwicklern, die für Umsetzung und Einsatz in Produktivsystemen verantwortlich sind. Eine mögliche Lösung dafür ist H2O, eine Machine-Learning-Plattform, die es ermöglicht, beide Welten einfach miteinander zu verbinden. Dieser Artikel stellt beispielhaft vor, wie mit H2O ein Machine-Learning-Modell in der Statistiker-Sprache R entwickelt und in Java auf Streaming-Daten live angewendet wird. Zum Einsatz kommen H2O in R, Spark Streaming in Java und Livedaten aus einem Twitter-API.

Der prototypische Workflow eines Data Scientist sieht etwa folgendermaßen aus: Er bekommt einen Datensatz, der manuell annotiert, geprüft, gelabelt, kuratiert oder auf andere Weise angereichert ist, füttert diese Daten in einen geeigneten Algorithmus und erhält daraus ein fertig trainiertes Modell. Dieses Modell kann jetzt auf Daten von der gleichen Art angewandt werden, die allerdings noch nicht manuell annotiert wurden. Ziel ist es, die Arbeit des Menschen zu replizieren, mit weniger Aufwand, auf größeren Datenmengen und im besten Fall sogar mit größerer Genauigkeit und Zuverlässigkeit.

Ein konkretes Beispiel wäre das Unterscheiden von Objekten auf Bildern: Wir haben einen Datensatz von ein paar Hundert Bilder, auf denen entweder eine Rose oder eine Tulpe zu sehen ist, jedes Bild mit einem passenden Label versehen. Das Modell kann nach dem Training auf neuen Bildern ohne Label entscheiden, ob diese eine Rose oder eine Tulpe zeigen. Solche Klassifikationsaufgaben sind typische Anwendungsgebiete und ein wichtiger Teilbereich von Machine Learning *(siehe Abbildung 1)*.

Wer solche Modelle entwickelt, hat oft einen Hintergrund als Mathematiker, Statistiker oder Physiker und arbeitet mit Werkzeugen,

Label	Tweet-Inhalt
Negativ	"Uqhhh it's sooooo hot outside ! #Texasweather"
Neutral/teilt Informationen	"Weather Alert: Flood Warning issued May 22 at 6:32PM MDT expiring May 23 at 9:32AM MDT by NWS Glasgow {link}... {link}"
Positiv	"Friday evening. Weather? Beautiful. Last man standing at the office. Bitter? Nah. It only makes payday that much sweeter. Stay hungry."
Nicht wetterbezogen	"Community Blood Center Media Blood drive Tues Noon-6 at Westridge Mall by Penneys lower level - Blood to help Joplin st ..."

Tabelle 1: Beispiele für Tweets mit Label

die für solche Modellierungsaufgaben und Datenexploration ausgerichtet sind. Häufig zum Einsatz kommen etwa Python oder das unter Statistikern weitverbreitete R *[2]*.

Im Gegensatz dazu arbeiten Software-Entwickler in Sprachen, Technologien und Umgebungen, die auf ihre spezifischen Anforderungen zugeschnitten sind. Wie ist es möglich, die Analyse-Ergebnisse oder Machine-Learning-Modelle, entstanden etwa in R, sinnvoll in ein Produktivsystem, realisiert etwa als Java-Backend, zu übernehmen? H2O erlaubt es uns, in R entwickelte Modelle in die Java-Welt zu bringen, ohne sie mühsam und fehleranfällig nachzubauen. Das wird an einem Beispiel zur Analyse von Textnachrichten gezeigt.

Demo Use Case: Tweet Klassifikation

Eine der schwierigeren, aber auch interessanteren Aufgaben aus dem Bereich „Machine Learning und Data Mining" ist es, menschliche Stimmungen und Gefühle zu erkennen – man spricht dabei auch von Sentiment-Analyse. Mit den entsprechenden Trainingsdaten und Werkzeugen werden wir einen Versuch in diese Richtung machen. Wir sehen uns Mitteilungen an, die Nutzer bei Twitter veröffentlichen, um zu erkennen, welche Einstellung jemand über das aktuelle Wetter hat – ist es ihr oder ihm zu heiß, zu kalt oder zu nass oder freuen sich die Twitterer im Gegenteil gerade über den herrlichen Sonnenschein. Dazu verwenden wir einen freien Satz von Trainingsdaten, in dem tausend Tweets per Hand in eine von vier Klassen eingeteilt wurden, nämlich positive, negative, neutrale Einstellungen zu Wetter sowie zur Abgrenzung auch einige Beispiele von Tweets, die nichts mit dem Wetter zu tun haben. Der Datensatz ist unter *[3]* verfügbar.

Tabelle 1 zeigt ein paar Beispiele solcher gelabelter Tweets. Ziel ist es, neue, noch nicht gelabelte Tweets einer dieser vier Klassen zuzuordnen. Dazu verwenden wir die Machine-Learning-Werkzeuge, die uns H2O zur Verfügung stellt. Um diese Sentiment-Vorhersage live durchzuführen, nutzen wir Spark Streaming, um in dieser Umgebung laufend die neuesten Tweets zu klassifizieren.

H2O ist eine vergleichsweise neue Plattform für Machine Learning, die bereits einige sehr fortschrittliche Fähigkeiten bietet und kürzlich auch zum Leader im Gartner-Magic-Quadrant für Data-Science- und Machine-Learning-Plattformen aufgestiegen ist. Die Open-Source-Lösung wird von einigen bekannten Stanford-Professoren beraten, Freunde des statistischen Lernens kennen sicher Namen wie „Tibshirani" und „Hastie".

H2O bietet nicht nur die üblichen Standard-Algorithmen und einige „State-of-the-art"-Methoden an, bei der Implementierung wurde auch sehr auf Effizienz und Performance geachtet, etwa durch interne Komprimierung von Daten. H2O arbeitet In-Memory und skaliert als ML-Plattform auf verteilte Rechencluster.

Eines der interessanten Features sind die Schnittstellen: H2O lässt sich in verschiedenen Programmiersprachen einsetzen, vorneweg Java, Python, R und Scala, genauso lässt es sich in weitere Tools wie Tableau und Spotfire einbinden. Das funktioniert, weil im Hintergrund ein (in Java implementierter) H2O-Cluster läuft. Dieser kann lokal aufgebaut werden oder auf mehrere Nodes verteilt sein. Die Funktionsaufrufe, mit denen ein Programm H2O verwendet, werden in REST-Calls an den Cluster übersetzt, die jeweiligen Program-

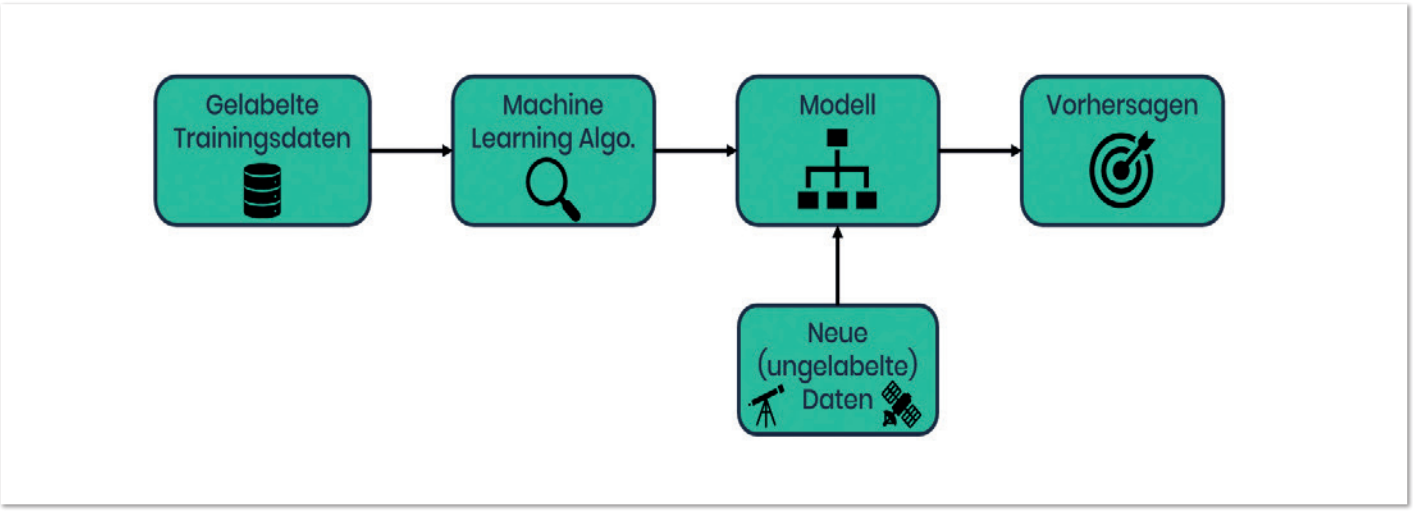


Abbildung 1: Schema für das Lernen und Anwenden eines Klassifikators

```
library(h2o)
h2o_context <- h2o.init(ip="localhost", port=4321)
```

Listing 1

miersprachen dienen als Interfaces und Operationen werden etwas versteckt innerhalb des Clusters ausgeführt. Das ermöglicht einen hohen Grad von Abstraktion und große Interoperabilität verschiedener Umgebungen über H2O *(siehe Abbildung 2)*.

Modell-Entwicklung in R

Wir starten unsere Pipeline zunächst weit weg von Java und verwenden R, um die Daten vorzubereiten und unser Modell zu trainieren. Als Entwicklungsumgebung bietet sich hier das weitverbreitete R-Studio an, es gibt allerdings auch Plug-ins für Eclipse, die R-Unterstützung ermöglichen.

Zunächst binden wir die H2O-Library ein und verbinden uns mit dem „init()"-Befehl zu einem H2O-Cluster. Läuft an der angegebenen Adresse noch kein Cluster, wird direkt eine lokale Instanz erzeugt *(siehe Listing 1)*.

Der erste Arbeitsschritt besteht darin, die Daten vom Filesystem in den Cluster zu übertragen. Das geschieht für uns sehr einfach mit „importFile()", und zwar unabhängig von der Datenquelle, auch wenn die Daten etwa aus einem Hadoop-System kommen. Im Hintergrund wird in beiden Fällen nur ein REST-Aufruf an den H2O-Cluster getätigt, die Details wurden für uns wegabstrahiert. Im Ergebnis wird ein H2O-Dataframe im Speicher des Clusters abgelegt. Dabei handelt es sich um ein spezielles, Tabellen-ähnliches Datenformat von H2O, das stark dem Dataframe ähnelt, der in R die Standard-Struktur für Datensätze aller Art ist *(siehe Listing 2)*.

Anstatt direkt auf den Tweets in Textform zu arbeiten, führen wir sie in ein Word-Embedding über. Jeder Text lässt sich mit einem solchen Embedding in einem Vektorraum darstellen, in dem ähnliche Texte auf nahe zusammenliegende Wörter abgebildet werden. Der Vorteil dabei ist einerseits, dass wir Texte mit einem Vokabular von vielleicht 100.000 oder mehr Wörtern kompakt darstellen können, hier im Beispiel in einem 100-dimensionalen Vektor. Andererseits erleichtert diese Transformation auch die Klassifikation der Tweets, da eben ähnliche Texte in ähnliche Bereiche des Vektorraums abgebildet werden, also gewissermaßen schon vorsortiert sind. Auch um Synonyme, Verneinungen und Wortkombinationen müssen wir uns nicht separat kümmern.

Ganz praktisch müssen wir die Wörter der Tweets zunächst etwas bereinigen, trennen sie also erst an Leerzeichen, entfernen zu kurze Wörter und Zahlen und vereinheitlichen zu Kleinbuchstaben. Die Hilfsfunktion dazu definieren wir in R mit den Tools von H2O. Das Training des Word-Embedding erfolgt wieder mit den High-Level-Funktionen von H2O, in unserem Fall haben wir als Modell Word2Vec mit der passend benannten Funktion „word2vec()" gewählt, ein Embedding auf Basis eines neuronalen Netzes *(siehe Listing 3)*.

```
# aus Filesystem:
twitter_data_df <- h2o.importFile("weather-agg-DFE.csv", header = T)
# aus Hadoop:
twitter_data_hdfs <- "hdfs://node-1:/user/data/twitter/weather-agg-DFE.csv"
twitter_data_df <- h2o.importFile(twitter_data_hdfs, header = T)
```

Listing 2

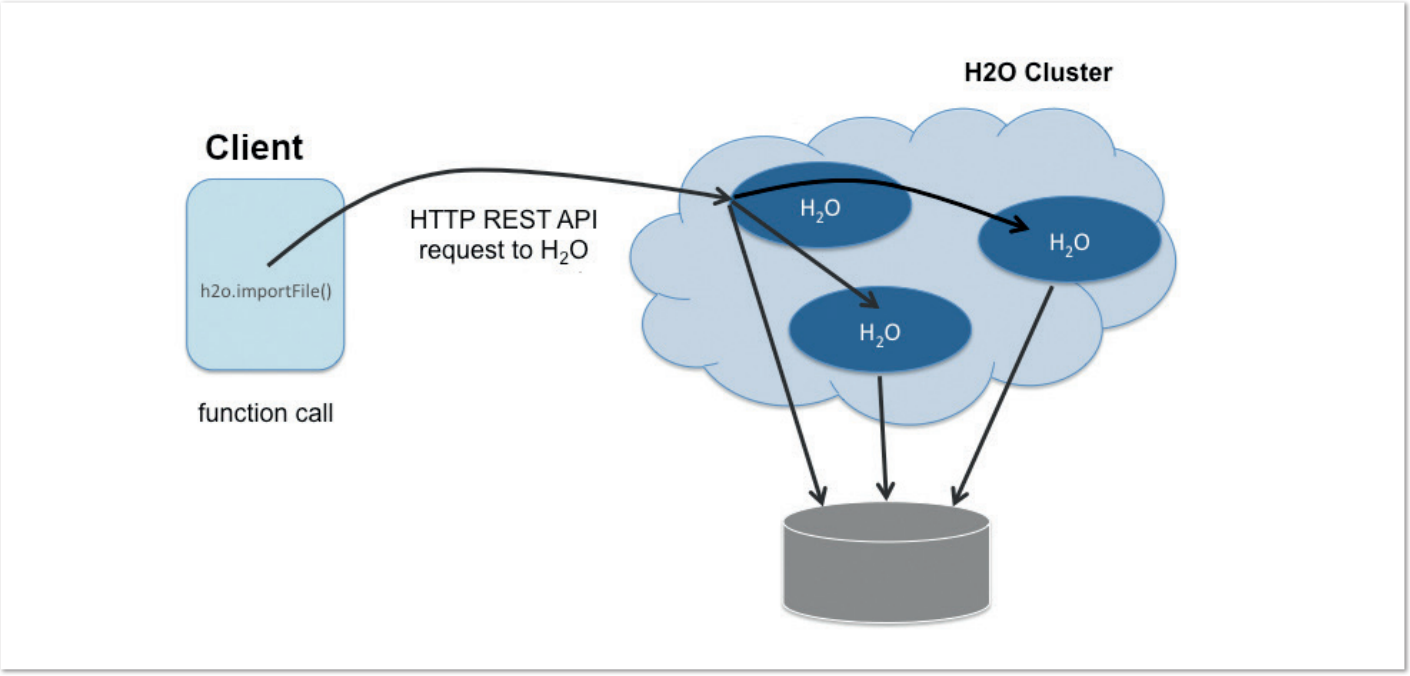


Abbildung 2: H2O-Cluster-Kommunikation [4]

```
tokenize <- function(sentences) {
  tokenized <- h2o.tokenize(sentences, "\\\\W+")
  tokenized.lower <- h2o.tolower(tokenized)
  tokenized.lengths <- h2o.nchar(tokenized.lower)
  tokenized.filtered <- tokenized.lower[is.na(tokenized.lengths) || tokenized.lengths >= 2,]
  tokenized.words <- tokenized.filtered[h2o.grep("[0-9]", tokenized.filtered, invert = TRUE, output.logical =
TRUE),]
  tokenized.words[is.na(tokenized.words),]
}
tweets_tokenized <- tokenize(twitter_data_df$tweet_text)

w2v_model <- h2o.word2vec(tweets_tokenized, vec_size = 100)
tweets_vecs <- h2o.transform(w2v_model, tweets_tokenized)
twitter_data_embedded <- h2o.cbind(twitter_data_df$emotion, tweets_vecs)
```

Listing 3

Wir haben nun einen fertig aufbereiteten Datensatz bestehend aus tausend Tweets, dargestellt als numerische Vektoren, und dem jeweils zugehörigen Label, das uns sagt, welche Emotion durch den Tweet ausgedrückt wurde. Diesen Datensatz können wir zum Training eines Klassifikationsmodells einsetzen. Wir verwenden ein Modell auf Basis von Entscheidungsbäumen. Etwas vereinfacht bauen wir dabei eine hierarchische Baumstruktur, in der wir für jeden neuen Tweet einem Pfad folgen können, der uns zu einem Label führt, das wir als Vorhersage benutzen wollen *(siehe Abbildung 3)*.

Ein kleiner Baum auf Basis von Stichwörtern, die im Tweet entweder vorkommen oder fehlen, könnte wie in *Abbildung 2* aussehen: Erscheint das Stichwort „sun" im Tweet, folgen wir der rechten Abzweigung. Jetzt könnte dem Autor aber auch zu heiß sein, erkennbar am Stichwort „too". Abhängig davon entscheiden wir uns für ein positives oder ein negatives Label. In unserem Beispiel verwenden wir einen Algorithmus, der viele solcher Bäume parallel erzeugt und ihre Vorhersagen kombiniert. Um dieses, „Random Forest" genannte, Modell zu trainieren, geben wir die vorherzusagende Variable an sowie den Datensatz, von dem gelernt werden kann. Das sind eben die Emotion des Tweets und die Tweet-Vektoren. Diese Angaben sind für alle Arten von Modellen nötig. Speziell für den Random Forest spezifizieren wir noch einige Parameter über die Zahl und Art der zu trainierenden Bäume *(siehe Listing 4)*.

An dieser Stelle könnten wir noch weitere Schritte zur Modell-Entwicklung durchführen. Wir können versuchen, die Parameter des Modells optimal einzustellen oder andere Modelle ausprobieren. Ein sehr wichtiger Punkt ist auch die Validierung des Modells. Dabei wird geprüft, ob das trainierte Modell auch auf neue Daten sinnvoll anwendbar ist, also nicht nur die Trainingsdaten auswendig gelernt hat, sondern gut generalisiert. Für unser Beispiel soll uns das jedoch nicht weiter interessieren. Die wichtigere Frage lautet, wie wir das fertige Modell weiterverwenden können.

Unser Ziel ist es, das Modell auf Live-Daten in Spark anzuwenden. Dazu könnten wir die Spark-Integration von H2O verwenden, die als „Sparkling Water" verfügbar ist. Aber eigentlich wollen wir die Welt von R verlassen und auch der H2O-Cluster ist nicht zwingend nötig, um die Modelle weiterzuverwenden. Stattdessen exportieren wir das fertige Modell in Java-kompatiblem Format. Dazu bietet H2O grundsätzlich zwei Möglichkeiten an: POJOs und MOJOs. *Tabelle 2*

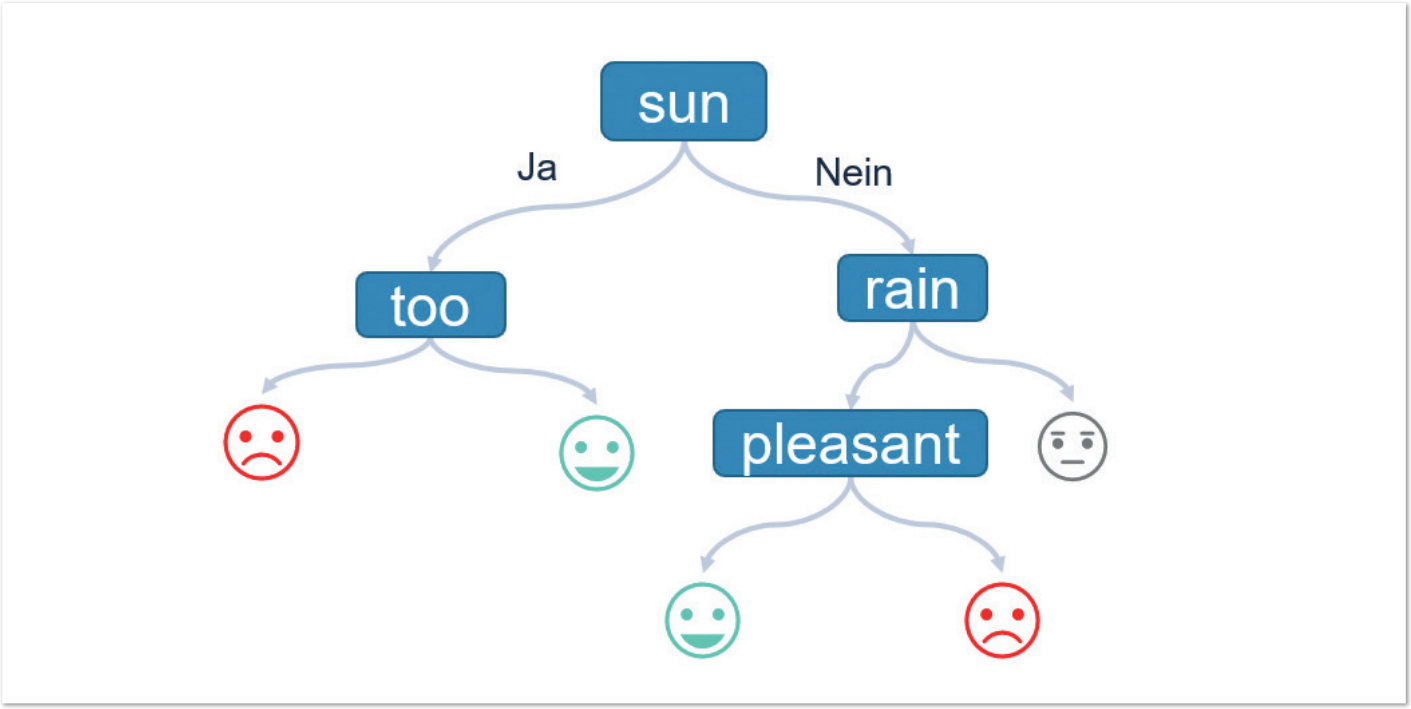


Abbildung 3: Einfacher Entscheidungsbaum

```
target <- "emotion"
emotion_model <- h2o. randomForest(y = target,
                        training_frame = twitter_data_embedded,
                ntrees = 100,
                        max_depth = 15)
```

Listing 4

Format	Typ	Eigenschaften
POJO	- Plain Old Java Object - kompilierbare Java-Klassen, die das Modell abbilden	- Java-Versions-unabhängig - Quellcode inspizierbar - Hängt von jar-file mit H2O-Klassen ab
MOJO	- Model Object Optimized - Binär-Format	- Platzsparend - Erst für wenige Modelle verfügbar - Hängt von jar-file mit H2O-Klassen ab

Tabelle 2: Modell-Exportformate von H2O

gibt einen kurzen Überblick über diese beiden Formate. Beide lassen sich direkt in Java als native Objekte verwenden, benötigen allerdings zusätzlich einen Satz von H2O-Java-Dependencies. Diese lassen sich mit unserem letzten Call zum H2O-Cluster zusammen mit dem MOJO oder POJO erzeugen.

Unser Modell wird dann als MOJO-File abgelegt, das „dependencies.jar" enthält die nötigen Klassen, um es in Java anzuwenden. Genauso exportieren wir auch das Word2Vec-Modell, das die Tweets in Vektoren transformiert *(siehe Listing 5)*.

Live-Klassifikation mit Spark Streaming

Wir wollen unser Modell zur Klassifikation von neuen Tweets einsetzen. Diese werden kontinuierlich erzeugt und stehen über ein kostenloses API von Twitter als Datenstrom zur Verfügung. Diesen Datenstrom bewältigen wir mit Spark Streaming, für unser Beispiel wieder möglichst einfach gehalten. Auf ein paar Konfigurationsschritte und Implementierungsdetails mussten wir aus Platzgründen verzichten, der Quellcode im Folgenden zeigt jedoch alle wichtigen Schritte.

Apache Spark ist eine der populärsten Plattformen für Cluster-Computing, das zusätzliche Streaming-Modul erlaubt die effiziente Verarbeitung von eben nicht nur Batch-, sondern Streaming-Daten. Die zugrunde liegende Spark-Architektur erlaubt es, die anfallende Last auf Server-Nodes zu verteilen. Einer der Vorteile von Spark und damit auch von Spark-Streaming ist die Verarbeitung In-Memory, was hohen Datendurchsatz ermöglicht. Die grundlegende Datenstruktur in Spark Streaming sind DStreams, eine verteilte Datensammlung, die man sich wie eine Collection von Records (streng typisiert) vorstellen kann.

Das Setup und die Programmierung von Spark Streaming erfolgen wieder vollständig in Java: Wir erzeugen einen Spark-Streaming-Context, der als Einstiegspunkt für die gesamte Spark Engine dient. Diesem geben wir eine Batchdauer mit, im Beispiel zwei Sekunden. Das bedeutet, dass Spark Streaming für zwei Sekunden Daten sammelt und diese als Mini-Batches weiterverarbeitet, was wieder die Performance gegenüber anderen Streaming Engines deutlich erhöht.

Um die echten Twitter-Daten in Spark Streaming einzubinden, verwenden wir die TwitterUtils-Bibliothek *[5]*. Um nicht von einem Tweet-Sturm überwältigt zu werden, filtern wir an dieser Stelle schon Tweets heraus, die ein paar Wetter-bezogene Keywords enthalten. „createStream()" erzeugt dann einen DStream, der laufend die aktuellen Tweets des Mini-Batch enthält *(siehe Listing 6)*.

Als nächsten Schritt binden wir die Modelle ein, die wir aus dem H2O-Cluster exportiert haben. Da es sich letztlich um Java-Objekte handelt, ist das wieder ganz einfach und geschieht mit den Hilfsfunktionen, die wir in den H2O-Dependencies finden *(siehe Listing 7)*.

```
modelfile <- h2o.download_mojo(emotion_model,
                               path = "/model/dir",
                               get_genmodel_jar = T,
                               genmodel_n = "dependencies.jar")
```

Listing 5

```
SparkConf sparkConf = new SparkConf().setMaster("local[4]").setAppName("StreamClassificationDemo");
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));
String[] filters = { "weather", "rain", "sunshine" };
JavaReceiverInputDStream<Status> tweetStream = TwitterUtils.createStream(ssc, filters);
```

Listing 6

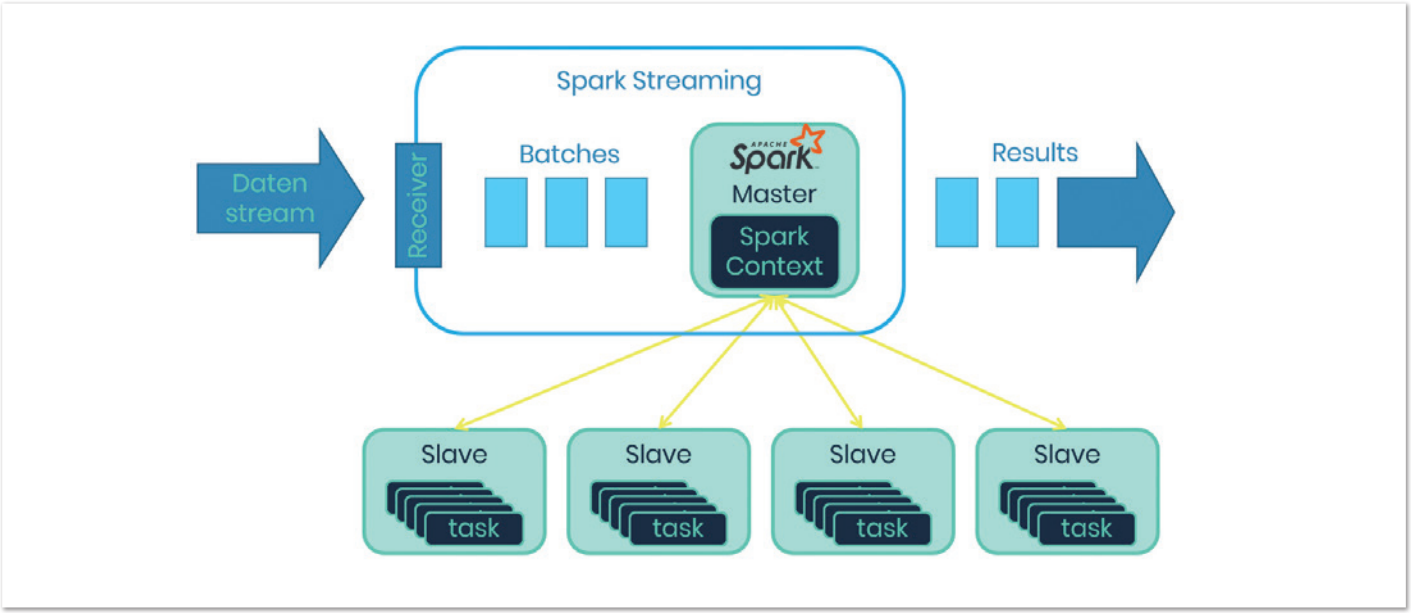


Abbildung 4: Spark-Streaming-Architektur

```
EasyPredictModelWrapper predictionModel = new EasyPredictModelWrapper(
  MojoModel.load("static/emotion_model.zip"));
WordEmbeddingModel w2vModel = Word2VecMojoModel.load("static/Word2Vec_model.zip");
```

Listing 7

Für das eigentliche Scoring, also die Klassifikation, der Tweets schreiben wir wieder eine Java-Hilfsfunktion, um die Tweets, schon in einzelne Wörter tokenisiert als Liste von Strings, zunächst wieder mithilfe des Word2Vec-Modells in einen numerischen Vektor zu übersetzen und diesen Vektor dann an das trainierte Klassifikations-Modell weiterzureichen *(siehe Listing 8)*.

Auf den Spark-Streaming-DStream angewandt, wird die Klassifikation dann für jeden Tweet, der über das API hereinkommt, ausgeführt. Das Modell muss auf jedem der Nodes vorhanden sein *(siehe Listing 9)*.

Wir haben Spark Streaming konfiguriert, eine Datenquelle angebunden, die Prädiktionsmodelle eingebunden und wenden diese auf die Streaming-Daten an – damit ist unsere Pipeline fertig definiert. Zu guter Letzt können wir die gesamte Pipeline endlich anwerfen und die Resultate beobachten. Dazu starten wir den Spark-Streaming-Context und lassen so lange Batches der Twitterdaten verarbeiten,

```
private static void classifyTweet(ArrayList<String> tweet, EasyPredictModelWrapper predictionModel,
        WordEmbeddingModel w2vModel) throws PredictException {
     final int vecSize = w2vModel.getVecSize();
     float[] tweetEmbedding = new float[vecSize];
     tweet.stream().map(word -> w2vModel.transform0(word, new float[vecSize])).reduce(tweetEmbedding,
           Utils::sumFloatarrays);

     RowData row = new RowData();
     for (int i = 0; i < vecSize; i++) {
        row.put("C" + i+1, Float.toString(tweetEmbedding[i]));
     }
     MultinomialModelPrediction prediction = predictionModel.predictMultinomial(row);
     System.out.println(prediction.label + ":");
}
```

Listing 8

```
tweetStreamTokenized.foreachRDD(rdd -> {
        rdd.foreachPartition(tweet -> {
           while (tweet.hasNext()) {
              classifyTweet(tweet.next(), predictionModel, w2vModel);
           }
        });
     });
```

Listing 9

Vorhersage	Tweet-Inhalt
Negativ	▪ "daijah gay ass pulled me out of work to kiss under the fucking rain" ▪ "england winning yday isnt enough to justify the weather"
Neutral/teilt Informationen	▪ "enjoying this glorious weather in the uk fond memories of this time last week" ▪ "Wind 0 mph gusting to 0 mph. Temperature 71.5 F. Humidity 84%. Rain Today: 0.00in"
Positiv	▪ „happy birthday america wishing you all fair weather safe shenanigans and quality time with friends and family" ▪ "I can't complain with 6 weeks off and lots of sunshine. Hope you're enjoying the summer break."
Nicht wetterbezogen	▪ "allen is going to hell for the last one" ▪ "is your caption about the weather or yourself?"

Tabelle 3: Einige Beispiele für klassifizierte Tweets

```
ssc.start();
ssc.awaitTermination();
```

Listing 10

bis wir das Programm abbrechen *(siehe Listing 10)*. Die *Tabelle 3* zeigt ein paar unserer Vorhersagen – nicht alle sind perfekt, einige der Tweets auch mehrdeutig, aber insgesamt ist das Ergebnis recht zufriedenstellend.

Natürlich bietet H2O noch viel mehr Funktionalitäten für Machine Learning, als hier gezeigt sind. Wer sich für neuronale Netze interessiert, findet hier etwa die Möglichkeit, auch solche Modelle zu entwickeln. Die angesprochene Parameter-Optimierung von Modellen ist mit mehreren Strategien wie Grid-Search oder Random-Discrete-Search möglich. Wer sich nicht für ein Modell entscheiden möchte, kann in sogenannten „Ensembles" einfach mehrere Modelle optimal miteinander kombinieren. Dazu kommen natürlich die üblichen Toolkits zu Modell-Validierung, Daten-Vorverarbeitung und -Cleaning. Wem das nicht genügt, der kann immer auf die nativen Möglichkeiten der gewählten Schnittstellensprache zurückgreifen.

Take aways

In der Praxis laufen Modellentwicklung und Produktivanwendung in unterschiedlichen Geschwindigkeiten, Arbeitsumgebungen und mit anderen Methoden und Zielen. Letztlich existieren also zwei parallele, getrennte Pipelines, wie wir auch in unserem Klassifikationsbeispiel gesehen haben. Den Output der ersten Pipeline, nämlich das Ergebnis von Analysen und fertige Modelle, in die zweite Pipeline zu bringen, ist keine triviale Aufgabe. Einzelne Elemente sind nicht übertragbar und müssen im schlimmsten Fall neu- oder nachimplementiert werden. Die H2O-Plattform bietet hier eine einfache Lösung, indem sie den Export und Transfer in die Java-Welt unterstützt oder selbst als Analytics-Backend einsetzbar ist. Dank der Unterstützung von Cloud Services wie AWS und Cluster-Computing wie Spark lassen sich auch Projekte mit großen Datenmengen mit H2O umsetzen.

Weitere Informationen

[1] *https://www.gartner.com/newsroom/id/3130017*
[2] *https://www.r-project.org*
[3] *https://data.world/crowdflower/weather-sentiment*
[4] *https://www.h2o.ai/h2o*
[5] *http://bahir.apache.org/docs/spark/current/spark-streaming-twitter*

Stephan Schiffner
stephan.schiffner@steadforce.com

Stephan Schiffner ist Director Advanced Analytics bei Steadforce und gesamtverantwortlich für Kunden-Projekte sowie die strategische Weiterentwicklung des Bereichs. Neben seinen beruflichen Aufgaben ist er seit dem Jahr 2011 als Lehrbeauftragter an der Hochschule München in den Bereichen „Software-Entwicklung" und „Software Engineering" aktiv.

Dr. Jonathan Boidol
jonathan.boidol@steadforce.com

Jonathan Boidol hat einen Master of Science in Bioinformatik erworben und an der LMU München zur Analyse von Streaming-Daten promoviert. Er ist Autor internationaler Fachartikel über Data Mining und Machine Learning. Jonathan Boidol ist Data Scientist bei Steadforce und arbeitet in Projekten von der Use-Case-Evaluierung bis zum Design und zur Implementierung von Advanced-Analytics-Modellen.

Agil – aber sicher!

Andreas Falk, NovaTec Consulting GmbH

Es vergeht kaum ein Tag ohne Meldung über eine gehackte Software-Anwendung, einhergehend mit dem Diebstahl sensibler Daten wie Kreditkarten-Daten oder Passwörter. In der Software-Entwicklung sind inzwischen agile Vorgehensweisen wie Scrum oder Kanban weitverbreitet. Sicherheit ist heute jedoch häufig immer noch lediglich eine Rand-Erscheinung innerhalb des agilen Software-Entwicklungsprozesses, die am Ende dazu implementiert oder hineingetestet werden soll. Dieser Artikel zeigt Ansätze und Wege, um die Security von Anfang an in den gesamten agilen Entwicklungsprozess zu integrieren.

Software ist in unserer Welt inzwischen so gut wie überall mit integriert. Mit dem Internet of Things (IoT) hält die Software auch zunehmend Einzug in unsere physische Welt und kann damit potenziell auch direkt Leben gefährden. Speziell in diesem Bereich wurden bereits Angriffsversuche bekannt wie etwa ferngesteuerte Jeeps oder fremde Stimmen aus internetfähigen Baby-Phones. Software wird in immer kürzeren Zyklen entwickelt und ausgerollt. Viele Unternehmen schieben neue Releases im Minutentakt in Produktion. Doch wie sicher kann eine solche Software in einem sich immer schneller drehenden Rad noch sein?

In der im Wasserfall-Modell praktizierten, klassischen Vorgehensweise wird die Software auf Basis einer vorab erstellten, genauen Spezifikation über die Entwicklung und einer Testphase bis zur Produktionsreife entwickelt und am Ende vor der Inbetriebnahme durch die eigene IT-Sicherheitsabteilung oder externe Penetrationstester anhand von Security-Gates auf ihre Sicherheit geprüft. Dieser län-

gere Prozess war bisher bei Release-Zyklen von mehreren Monaten beziehungsweise Jahren auch kein Problem.

Im Zeitalter der agilen Entwicklung von Microservices in der Cloud mit Praktiken wie Continuous Delivery bzw. Continuous Deployment verkürzen sich diese Zyklen aufgrund der stetig zunehmenden „Time to Market"-Anforderungen drastisch, teilweise bis auf wenige Minuten. Das bedeutet, die Software wird in kurzen Zeitabständen in agilen Vorgehensmodellen erstellt und praktisch jeder Commit landet dann direkt in der Produktion. Allerdings läuft die Sicherheitsprüfung häufig immer noch parallel in einem Wasserfall-ähnlichen Modell ab. Hin und wieder wird punktuell ein Deployment durch Sicherheitsexperten unter die Lupe genommen und nachfolgende Releases sind dann als potenziell sicher eingestuft. Gerne wird die Sicherheitsprüfung neben der Dokumentation und sonstigen Tests schnell auch komplett wegoptimiert, wenn die Zeit im Projekt mal wieder knapp wird.

Abbildung 1 zeigt eindrucksvoll die Gefahren eines derartigen Vorgehens. Hacker machen keinen Urlaub und kennen auch keine Feiertage oder Wochenenden. Während die Angreifer also praktisch rund um die Uhr versuchen, in das System einzudringen und bei lohnenswerten Zielen dabei immer schwerere Geschütze auffahren, wird die Anwendung selbst nur vergleichsweise selten punktuell auf Sicherheitslücken getestet.

Diese Tatsache hat auch die im Jahr 2017 neu aufgelegte Version der OWASP-Top-10-Liste in dem neu aufgenommenen Punkt A10 („Insufficient Logging & Monitoring") berücksichtigt *[1]*. Damit soll jedem Entwickler bewusst gemacht werden, dass die eigene Web-Anwendung ständig automatisierten Angriffen ausgesetzt ist und sie daher entsprechende Verteidigungsstrategien als Reaktion benötigt.

Aus diesen Gründen muss in den aktuell immer kürzer werdenden Entwicklungszyklen auch die Sicherheit der Anwendung unbedingt mit der Deployment-Frequenz Schritt halten. Dies kann nur gewährleistet werden, wenn der Security-Aspekt auch in alle agilen Aktivitäten innerhalb des Continuous-Delivery-Prozesses mit integriert ist.

Eine etwas andere Sicht auf die Sicherheitsproblematik in agilen Softwareprojekten macht *Abbildung 2* deutlich. In den dargestellten Sprints werden die vom Product Owner priorisierten User Stories vom Entwicklungsteam als Businesswerte in Software umgesetzt. Diesen User Stories fehlen überwiegend jegliche Anteile bezüglich Security in deren Akzeptanz-Kriterien.

Wichtige funktionelle Sicherheits-Features (welche ernstzunehmende Anwendung benötigt tatsächlich keinerlei Authentifizierung beziehungsweise Autorisierung?) werden häufig ans Ende geschoben: „Das kann man ja noch hinterher machen, wir brauchen erstmal Businesswerte". Sicherheit in einem der letzten Sprints hinterher einzubauen, ist sehr aufwendig und damit teuer.

Auf der anderen Seite wird diese aufgrund der knappen Zeit als Konsequenz oft nur lückenhaft umgesetzt und aufgrund der späten Umsetzung auch kaum laufend mit getestet. Die letztendliche Prüfung der Sicherheit wird dann aus meiner Erfahrung in der überwiegenden Anzahl von Projekten ganz am Schluss, oftmals sogar parallel zur bereits erfolgten Livesetzung durchgeführt.

Ein agiles Security-Manifest

Der bis hierher geschilderte Status quo bezüglich Sicherheit in agilen Projekten führt glücklicherweise in Bewegungen wie SecDevOps oder Rugged Software auch zunehmend zu einem kulturellen Umdenken: Sicherheit muss Teil der Arbeit jedes Projektbeteiligten werden.

Aus verschiedenen Erhebungen weiß man inzwischen, dass auf rund hundert Entwickler nur ein Sicherheitsexperte kommt *[2]*. Allein aus diesem Grund muss klar sein, dass die Security auf mehrere

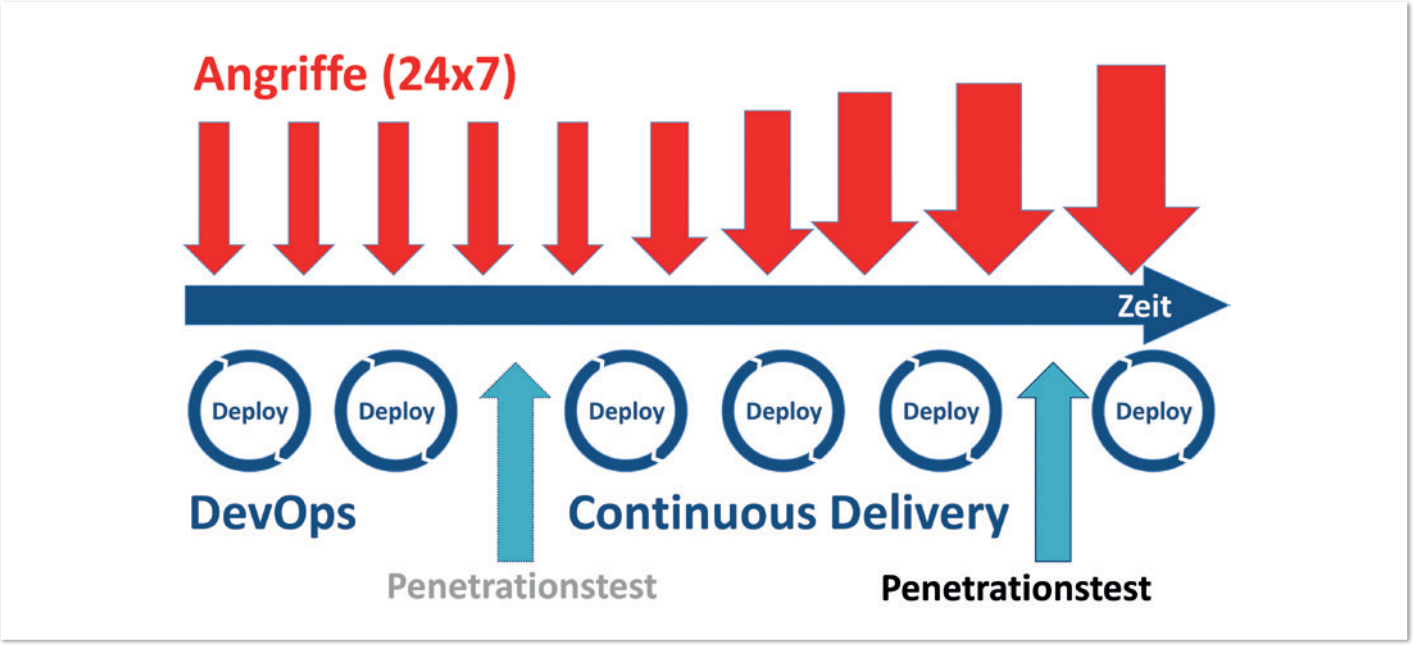


Abbildung 1: Angriffe (24x7) gegenüber punktuellen Verteidigungsmaßnahmen

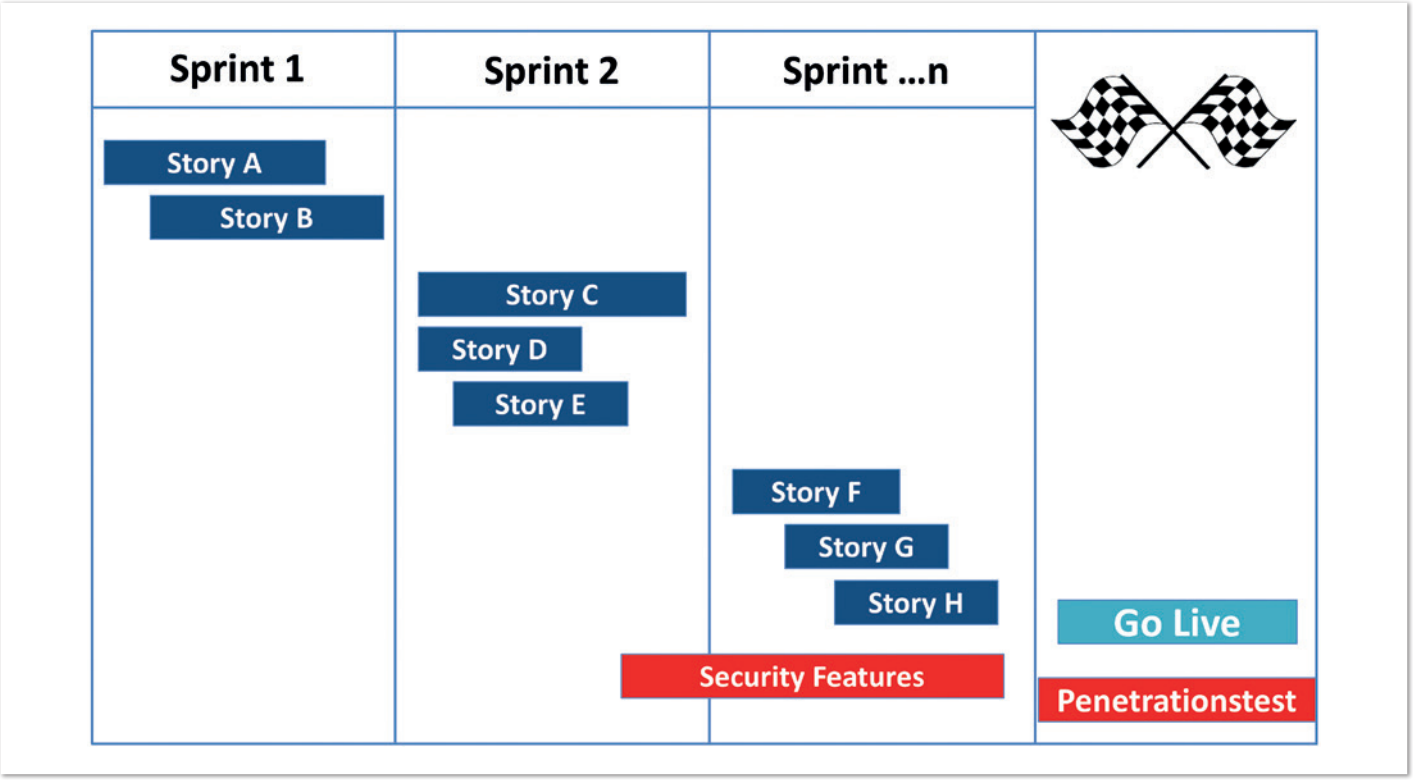


Abbildung 2: Security in der Sprintplanung vieler aktueller Projekte

Schultern zu verteilen ist. Analog zum allseits bekannten Manifest für agile Softwareentwicklung *[3]* kann man dies auch in einem Manifest für die sichere agile Softwareentwicklung in Worte fassen:

- Security ist der Job des gesamten Entwicklungsteams
- Security-Risiken vorab identifizieren statt nur Sicherheitslücken nachzujagen
- Konkrete Security-Anforderungen statt impliziter Annahmen
- Sichere Entwicklung statt Sicherheit hinterher in der Software zu testen
- Einsatz etablierter Security-Bibliotheken und Frameworks
- Pro-aktives Security-Monitoring statt reaktiver Maßnahmen nach einem Angriff

In den folgenden Abschnitten wird anhand des in der Entwicklung überwiegend praktizierten Scrum-Prozess-Frameworks erläutert, wie darin das agile Security-Manifest vom Scrum-Team zum Leben erweckt wird *(siehe Abbildung 3)*. Das Scrum-Team besteht dabei aus dem Product Owner, dem Entwicklungsteam und dem Scrum Master. Darüber hinausgehende Rollen sind in Scrum nicht vorgesehen.

Das Herzstück von Scrum ist der Sprint, in dem in einer Iteration von einer bis vier Wochen ein potenziell auslieferbares Produkt-Inkrement erstellt wird. Im Zuge der sicheren Entwicklung sollte dies als erste Aktion gleich ergänzt werden zu einem potenziell auslieferbaren, sicheren Produkt-Inkrement.

Vor einem Sprint finden Events wie das Story-Refinement und die Sprint-Planung statt. Nach einem Sprint erfolgen ein Review des Produkt-Inkrements durch die Stakeholder und eine Retrospektive des vergangenen Sprints durch das Scrum-Team. Security-Aktivitäten müssen folglich nicht nur in die eigentliche Sprint-Iteration, sondern auch in die Phasen vor und nach einer Iteration integriert werden.

Dazu hat es sich bewährt, einen Security-Officer (synonym auch als „Security-Champion" oder „Security-Master" benannt) als Teil des Scrum-Teams einzuführen. Dieser sorgt dafür, dass sich die erforderlichen Security-Aktivitäten im gesamten Scrum-Team etablieren. Der Security-Officer rekrutiert sich meist aus einer Person aus dem Entwicklerkreis, die sich für das Thema „Security" interessiert und mit Trainings entsprechend fortgebildet ist. Manchmal wird aber auch einer der rar gesäten Security-Spezialisten mit der Security-Officer-Funktion als Teil des Scrum-Teams rekrutiert.

Scrum stellt frei, dass einzelne Teammitglieder wie der Security-Officer spezielle Fähigkeiten mitbringen und besondere Bereiche abdecken. Wichtig hier ist aber vor allem, dass der Security-Officer ein vollwertiges Mitglied des Scrum-Teams ist. Ein weiterer Baustein ist dann der Zusammenschluss mehrerer Security-Officer in sogenannten „Communities of Practice", um sich über die Projekte hinweg auszutauschen. Dann sind auch wechselseitige Sicherheitsprüfungen der einzelnen Projekte durch die Security-Officer möglich.

Sicherheit entsteht bereits weit vor dem Sprint

Die prä-iterativen Sicherheitsaktivitäten beginnen bereits bei der Produktvision, den ersten Anforderungen und Design-Prototypen. Schon hier sollten Business-Analysten zusammen mit den IT-Architekten und dem Security-Officer frühzeitig erste Überlegungen hinsichtlich schützenswerter Businesswerte (Assets), der Sicherheitsrisiken und der damit verbundenen Angriffs-Szenarien vornehmen. Dabei hat sich das Threat Modelling als hilfreich erwiesen. Hier wird auf Basis der vorhandenen Anwendungsarchitektur und zu schützender Assets ein Angriffsmodell abgeleitet, das unter anderem durch Festlegung von Trust Boundaries zwischen den Komponenten hilft, potenzielle Angriffspunkte und -typen zu identifizieren. Eine sehr gute Einführung in das Threat Modelling findet sich in *[4]*.

Aus den Diskussionen rund um das Threat-Modell ergeben sich Anforderungen an die Sicherheit. Eine Kategorisierung von User Stories sieht dann so aus:

- User Stories für Security Features (wie Authentifizierung und Autorisierung)
- AbUser Stories (Details weiter unten)

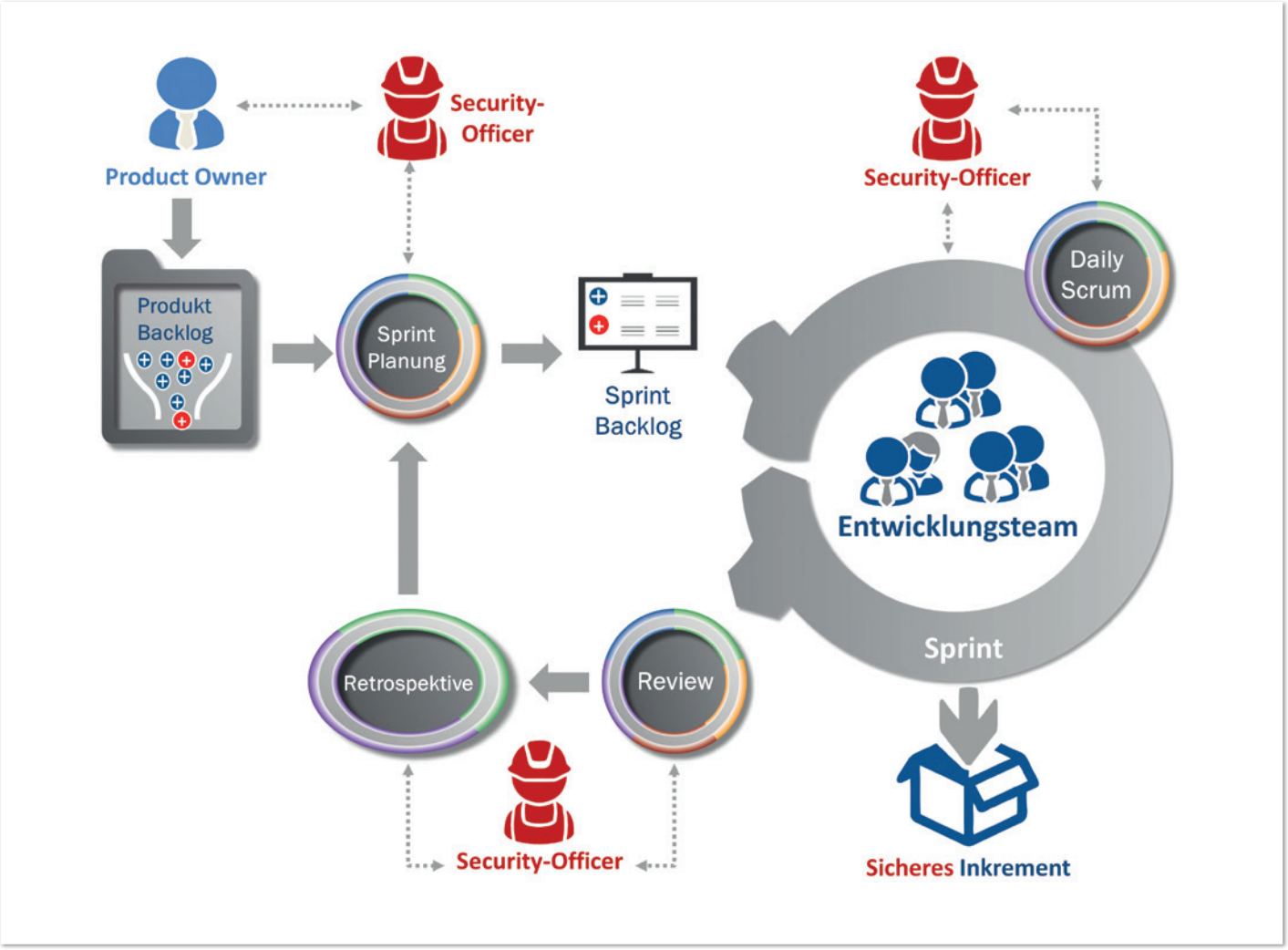


Abbildung 3: Integration von Security-Aktivitäten in Scrum mit neuer Rolle als Security-Officer

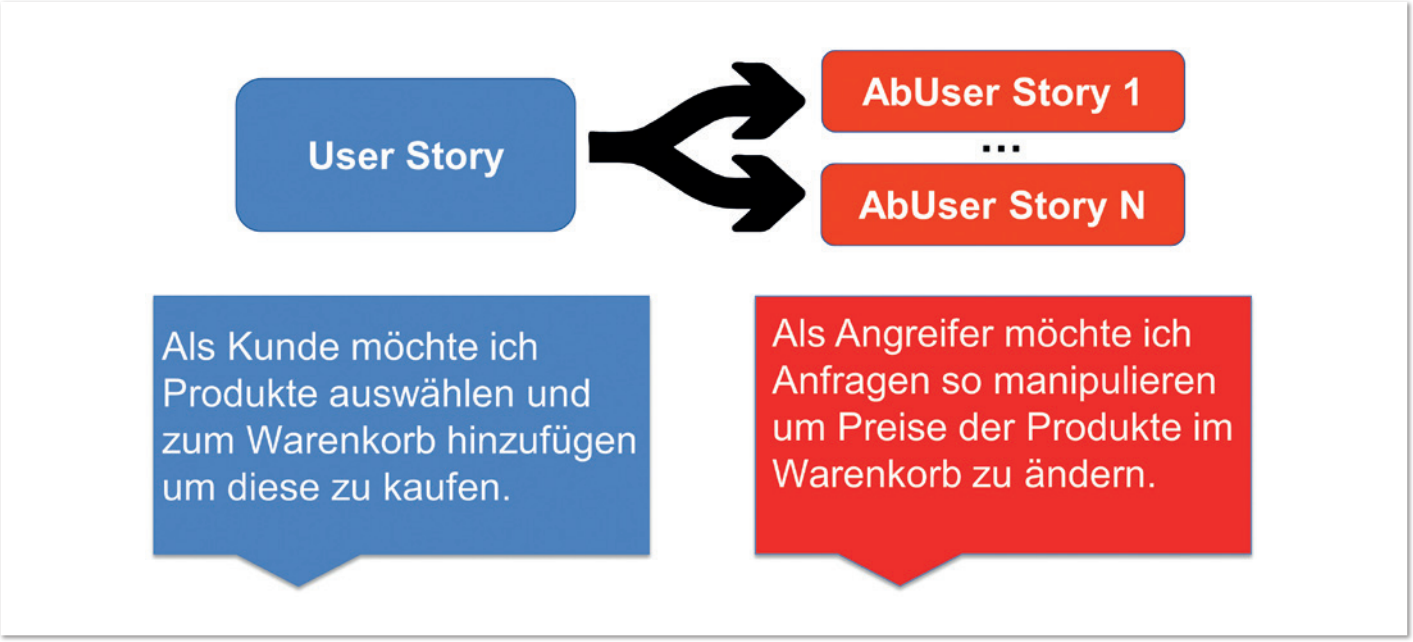


Abbildung 4: Aus User Stories abgeleitete AbUser Stories

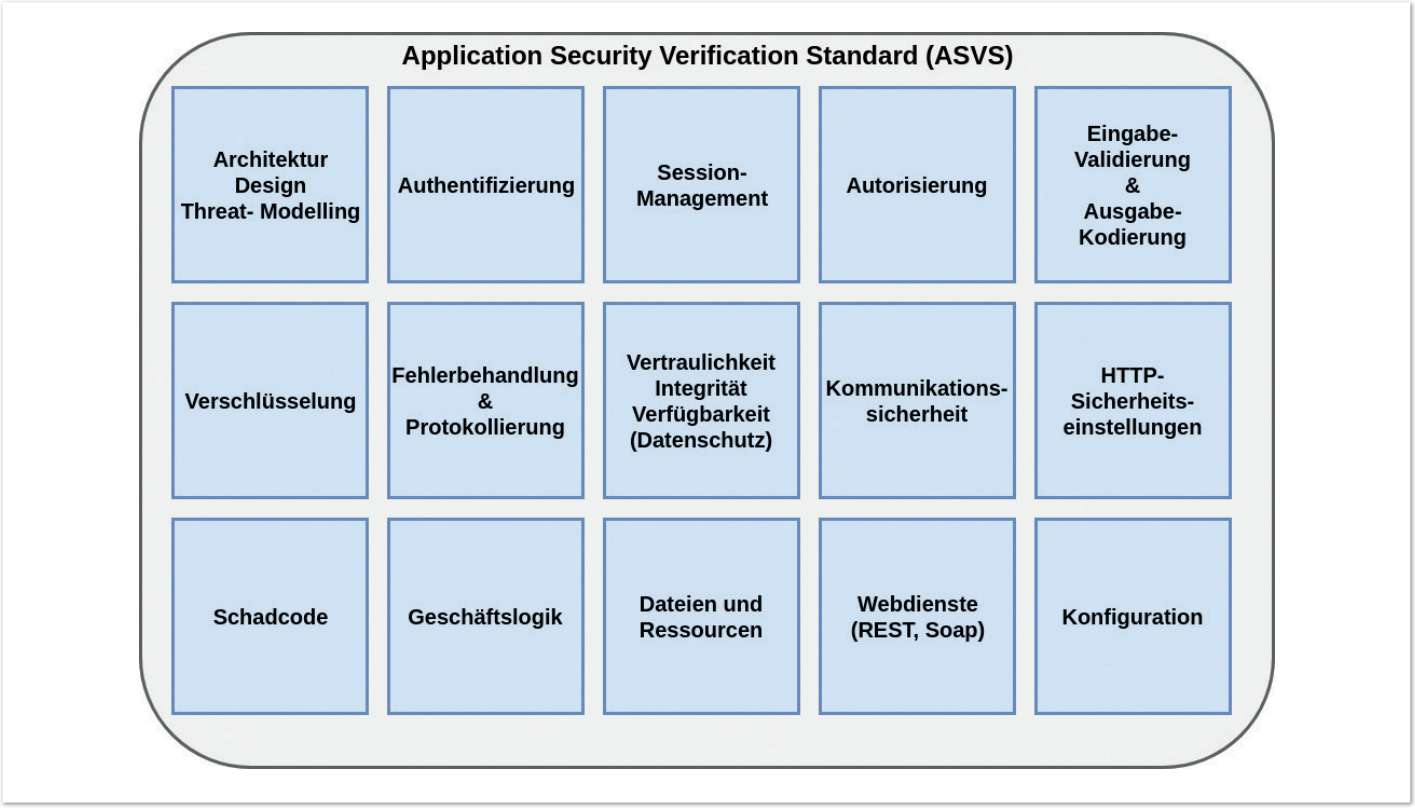


Abbildung 5: Prüfbereiche im Application-Security-Verification-Standard

- User Stories mit sicherheitsrelevanten Akzeptanzkriterien
- User Stories ohne jeglichen Security-Bezug

AbUser Stories (auch bekannt als „Evil" User Stories) werden aus den fachlichen User Stories abgeleitet und beschreiben die damit verbundene, gewünschte Funktionalität aus Sicht des Angreifers. Dabei sollten analog wie bei fachlichen User Stories auch Personas verwendet werden; statt dem allgemeinen „Hacker" könnten also ein Skript Kiddie, ein Firmen-Insider oder ein Wirtschaftsspion vorkommen. *Abbildung 4* zeigt ein Beispiel für eine fachliche (Business) User Story mit einer abgeleiteten AbUser Story.

Eine AbUser Story beinhaltet ebenso wie die normalen User Stories auch Akzeptanzkriterien. Diese sind hier jedoch umgekehrt formuliert; das Entwicklungsteam muss also hier durch entsprechende Tests nachweisen, dass diese Akzeptanzkriterien nicht eintreten. Als hilfreicher Input für derartige Akzeptanzkriterien hat sich der Application Security Verification Standard (ASVS) von der OWASP erwiesen *[5]*. ASVS definiert drei verschiedene Prüfungsstufen für Sicherheitsanforderungen. Für jede Stufe sind Prüfungskriterien aus unterschiedlichen Kategorien wie Authentifizierung, Session-Verwaltung und Fehlerbehandlung festgelegt *(siehe Abbildung 5)*.

Die unterste Stufe bietet eine minimale Grundabsicherung der Anwendung gegen die häufigsten Angriffe der OWASP Top 10. Die zweite, als Standard empfohlene Stufe sichert die Anwendung gegen die meisten heute bekannten Sicherheitsrisiken ab. Die dritte, höchste Stufe ist für besonders sicherheitssensible Anwendungen bestimmt. Dies umfasst militärische oder andere kritische Bereiche mit Safety-Anteilen wie Flughäfen, Kraftwerke, Medizin oder sonstige streng regulierte Einsatzgebiete. *Abbildung 6* zeigt einen Ausschnitt der Anforderungen aus dem Bereich „Authentifizierung" (jeweils mit dem betreffenden Sicherheitslevel).

Durch AbUser Stories und deren Akzeptanzkriterien lassen sich im weiteren Verlauf für die Entwicklung im Rahmen der Sprint-Planung entsprechende Aufgaben erstellen. Auch hier steht der Security-Officer dem Product Owner und dem Entwicklungsteam bei der Erstellung und Pflege der User Stories – auch im Rahmen von Refinement-Meetings – beratend zur Seite.

Neben der „Definition of Done" für das Entwicklungsteam hat sich inzwischen auch eine „Definition of Ready" als Reifegradmesser für die User Stories etabliert. Beide Definitionen sind um sicherheitsspezifische Aspekte zu ergänzen. Schließlich ist hier auch der richtige Zeitpunkt, um sich mit Aspekten wie Richtlinien zur sicheren Programmierung oder einzusetzenden Standardbibliotheken für Sicherheitsfeatures zu beschäftigen.

Sicherheit entsteht vor allem in der Entwicklung im Sprint

Eine neue Sprint-Iteration wird durch die Sprint-Planung eingeleitet. Hier stellt der Product Owner die potenziell als nächstes umzusetzenden User Stories dem Entwicklungsteam vor. Auch da sind sicherheitsspezifische Rückfragen seitens des Teams an den Product Owner beziehungsweise an den Security-Officer explizit erwünscht. Hier bietet sich die Möglichkeit, den User Stories gegebenenfalls noch fehlende sicherheitsrelevante Akzeptanzkriterien und Testszenarien hinzuzufügen. Darüber hinaus kann hier eine zusätzliche AbUser Story entstehen, die dann für einen der nachfolgenden Sprints eingeplant wird. Die Security kann damit auch durchaus einen Einfluss sowohl auf den Sprint-Umfang als auch auf das Sprint-Ziel haben. In jeder Sprint-Planung sollte auch nochmal ein prüfen-

Application Security Verification Standard - Authentifizierung

Level 1	Alle Seiten und Ressourcen erfordern eine Authentifizierung mit Ausnahme speziell für die Öffentlichkeit bestimmter Seiten/Ressourcen
	Alle Authentifizierungsprüfungen müssen auf Serverseite durchgeführt werden
	Bei Fehlern in der Authentifizierungsprüfung muss eine Anmeldung unterbunden werden
	Eine Passwortänderung verlangt immer das alte und neue Passwort und eine Bestätigung
Level 2	Alle Authentifizierungsaktionen werden ohne Speicherung sensibler Daten protokolliert
	Alle Passworte sind mit einem Salt-Wert gehasht und der Hash-Algorithmus ist langsam genug um entsprechende Brute-Force Angriffe abzuwehren
	Eine Multi-Faktor-Authentifizierung oder ein vergleichbarer Mechnismus wird angeboten um einen zusätzlichen Schutz gegen den Klau von Passworten zu bieten
Level 3	Geheime Daten wie u.a. API Keys oder Passworte dürfen nicht im Quellcode oder in Quellcode-Repositories abgelegt werden.

Abbildung 6: ASVS-Anforderungen aus dem Prüfbereich „Authentifizierung"

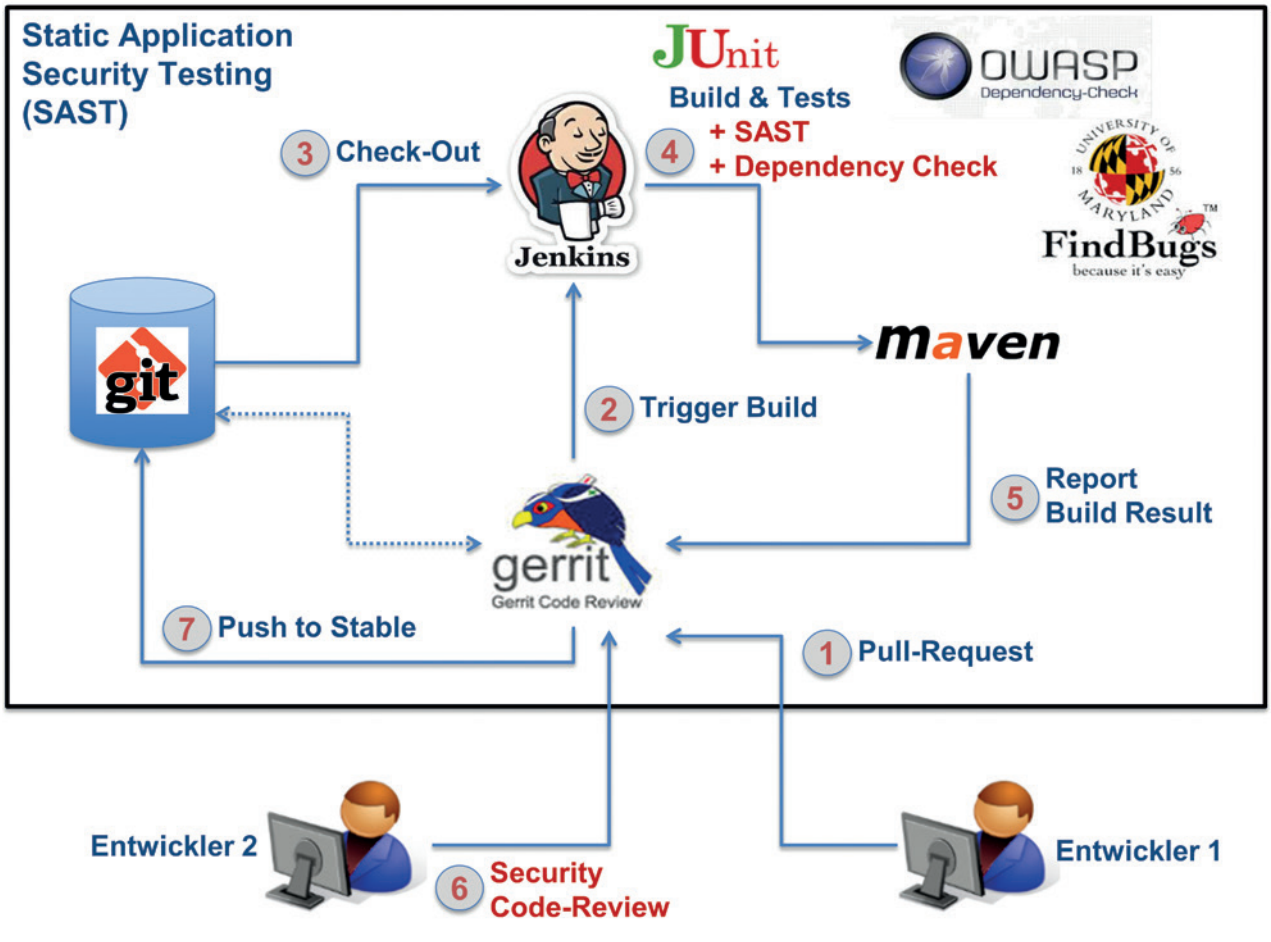


Abbildung 7: Continuous Integration Stage mit SAST, Abhängigkeitsprüfung und Code Reviews

der Blick auf das (hoffentlich) bereits vorhandene Threat-Modell geworfen und dieses je nach Bedarf angepasst werden.

Mit Beginn der Iteration werden in vielen Scrum-Teams die einzelnen User-Stories in Story-Kickoff-Meetings nochmals im Detail diskutiert und in einzelne kleinere Umsetzungsaufgaben zerlegt. Der Security-Officer achtet darauf, dass erforderliche sicherheitsrelevante Aufgaben hier nicht vergessen werden.

Gerade in der Entwicklung ist Security ein wichtiger Bestandteil. Hier sollte jeder Entwickler zumindest ein gewisses Maß an Basiswissen bezüglich Sicherheit besitzen. Geeignete Standard-Architekturen und Secure Coding Design Patterns *[6]* können hier von Anfang an Sicherheitslöcher minimieren. Unterstützend sollten nur erprobte Standard-Bibliotheken aus dem Sicherheitsbereich zum Einsatz kommen. Im Java-Enterprise-Umfeld sind dies zum Beispiel Spring Security oder Apache Shiro. In keinem Fall sollte man versuchen, eigene Hashing- oder Verschlüsselungs-Algorithmen zu implementieren. Versäumnisse in der Entwicklung sind später durch Security-Tests nur mit erheblichem Aufwand korrigierbar – sofern sie überhaupt rechtzeitig entdeckt werden.

Essenziell ist in dieser Phase ein wechselndes Pair-Programming der Entwickler mit dem Security-Officer. Dadurch ist neben der Erhöhung der Sicherheit im entstehenden Programmcode eine fortwährende Security-Weiterbildung der anderen Entwickler gewährleistet. Im Optimalfall werden damit auch weitere Security-Officer ausgebildet.

Um nun im Extremfall des Continuous Deployment mit jedem Commit ein neues sicheres Release der Software in Produktion zu bringen, ist die Test-Automatisierung ein überlebenswichtiger Baustein. In der ersten Stufe hat sich eine Continuous Integration Stage (CI) wie in *Abbildung 7* bewährt. Nachdem ein Entwickler einen Pull-Request mit einem neuen Satz Programmcode gestellt hat, werden die gesamte Anwendung neu gebaut und Unit- sowie Integrationstests ausgeführt. Diese Tests können bereits sicherheitsspezifische Anteile enthalten. Beispielsweise lässt sich heutzutage auch ein Autorisierungsmodell automatisch mit entsprechenden Tests auf gegebenenfalls vorhandene Lücken überprüfen. Der Test-Code sollte auch in keinem Fall als Code zweiter Klasse neben dem eigentlichen Produktiv-Code angesehen werden. Es wurden schon Fälle bekannt, in denen ein Insider-Angriff über eingeschleusten Malware-Code in Unit-Tests ausgelöst wurde.

Daran schließt sich die statische Analyse des Programm-Codes an (SAST: Static Application Security Testing). Im Java-Bereich bietet sich hierfür das Open-Source-Werkzeug FindBugs beziehungsweise dessen Weiterentwicklung SpotBugs *[7]* inklusive des Find-Security-Bugs-Plug-ins an *[8]*. Um das Entwicklerteam nicht mit unzähligen Fehlalarmen („False-Positives“) zu verwirren, sollte hier erst einmal mit einem kleinen Prüfungsumfang gestartet und dieser dann langsam erweitert werden. Ein weiterer oft vernachlässigter Bestandteil dieser CI-Stage ist die Prüfung von Fremdbibliotheken auf bekannte Sicherheitslöcher. Auch hier stehen mit OWASP Dependency Check *[9]* für Java oder Retire.js für JavaScript ausgereifte Open-Source-Werkzeuge zur Integration in die Build-Pipeline zur Verfügung.

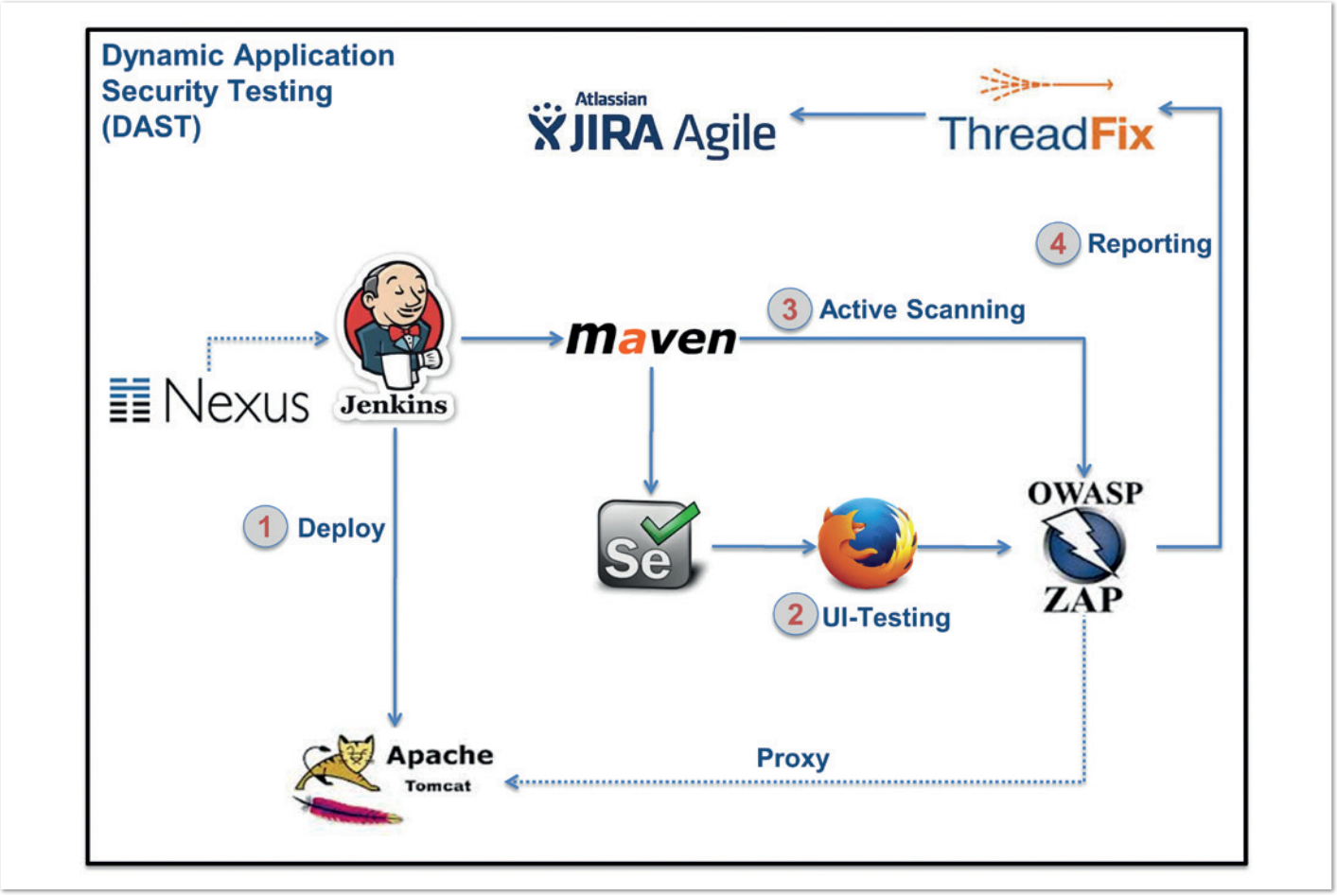


Abbildung 8: Dynamische Tests der Anwendungssicherheit mit Selenium und OWASP Zap

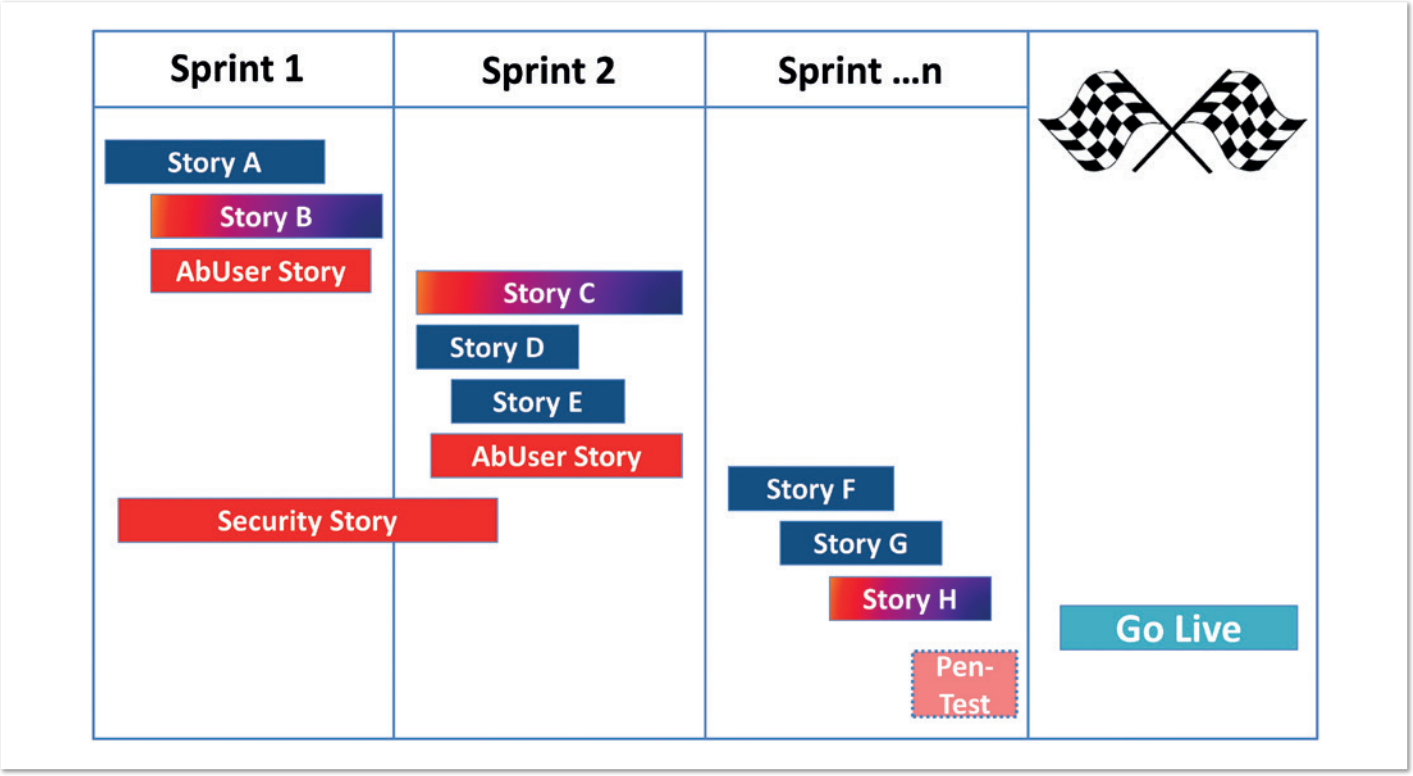


Abbildung 9: Optimale Integration von Security in die Sprints

Wird der automatisierte Prüfungsteil erfolgreich durchlaufen, greift der letzte integrale Bestandteil der CI-Stufe, das manuelle (Security) Code-Review durch einen zweiten Entwickler beziehungsweise den Security-Officer (beispielsweise mit Gerrit). Erst wenn auch diese letzte Hürde genommen ist, wird der neue Programmcode auf den Hauptzweig des Versionskontrollsystems eingecheckt und der Binärcode in ein gemeinsames Repository wie Nexus oder Artifactory hochgeladen.

Nun greift die nächste Stufe der Security-Testautomatisierung, der dynamische Sicherheitstest der Anwendung (DAST: Dynamic Application Security Testing). Beim dynamischen Sicherheitstest wird die Anwendung automatisiert mit tatsächlichen Angriffen wie SQL Injections oder Cross-Site-Scripting konfrontiert. Dazu gibt es mit OWASP Zap *[10]* ein sehr gutes Open-Source-Werkzeug, mit dem sich auch automatisierte dynamische Security-Tests per API in die DAST Stage integrieren lassen.

Abbildung 8 zeigt eine beispielhafte Pipeline für DAST. Zunächst wird der aus der ersten CI-Stage aus dem Repository (Nexus etc.) geladen und im Apache Tomcat Server eingerichtet. Anschließend werden mithilfe des Selenium-Frameworks Web-UI-Tests mit konfiguriertem OWASP Zap als Proxy zwischen dem Web-Browser und der Ziel-Anwendung ausgeführt und damit alle durchlaufenden Anfragen mit deren Adressen und Parametern aufgezeichnet. Sie werden dann in einem zweiten Schritt im OWASP Zap ein weiteres Mal durchlaufen, dabei allerdings einem aktiven Scan unterworfen. Die bekannten Anfragen werden also nun unter anderem mit entsprechenden SQL-Injection- oder XSS-Payloads befeuert.

Alle daraus resultierenden Ergebnisse werden in ThreadFix, einem spezifischen Reporting-Werkzeug für Security-Tests, aufgesammelt. Die Entwickler können nun mit beratender Mitwirkung des Security-Officers die gesammelten Reports bewerten, etwaige False Positives aussortieren und dann aus den verbleibenden echten Fehlern geeignete Tickets im Bugtracking-System (beispielsweise Jira) erstellen. Diese werden dann als technische Schulden vom Scrum-Team entsprechend der Priorität abgearbeitet.

Die Königsdisziplin bezüglich Security-Testing stellt das explorative Testen dar. In Zusammenarbeit mit dem Security-Officer wird durch ein sogenanntes „Charter" der Umfang des explorativen Tests abgesteckt. Ein Charter könnte zum Beispiel lauten: „Erforsche durch verschiedene URL-Adresseingaben im Browser Daten anderer Benutzer, um Sicherheitslücken in der Autorisierung aufzudecken." Durch diese Tests lernt man auch die eigene Anwendung aus neuen Blickwinkeln kennen und generiert neue Test-Ideen.

Nach dem Sprint ist vor dem nächsten Sprint

Der Sprint findet seinen Abschluss im Sprint Review. Dort sind die Stakeholder, also im Idealfall die Benutzer der Anwendung, aufgefordert, das neu erstellte Produkt-Inkrement unter die Lupe zu nehmen. Auch hier sollten alle Beteiligten durch den Security-Officer für die Sicherheit sensibilisiert werden. Eine gute Idee ist es beispielsweise, im Rahmen des Reviews die Stakeholder aufzufordern, selbst einmal ungültige Eingaben oder Wege in der Anwendung auszuprobieren. Eine vertrauensfördernde Maßnahme ist dabei auch, dem Kunden auf Wunsch das Recht einzuräumen, selbst einen Penetrationstest auf dem aktuellen Stand vorzunehmen.

In der Retrospektive ist der Scrum-Prozess durch Beobachten und Anpassen immer weiter zu verbessern. Sind im Sprint verbesserungswürdige sicherheitsrelevante Punkte aufgefallen, dann sind diese zu diskutieren. In Zusammenarbeit mit dem Security-Officer werden anschließend entsprechende Aktivitäten mit Verantwortlichkeiten für den nächsten Sprint definiert.

SecDevOps: Mehr als nur Netzwerksicherheit

In vielen Unternehmen ist der Betrieb von der Entwicklung immer noch strikt getrennt und damit weit entfernt von einer DevOps-Kultur. Häufig kümmert sich der Betrieb auch nur um die Absicherung der Netzwerke durch angemessene Netzwerk- und Webanwendungs-Firewalls sowie die Konfiguration der Transportsicherung durch HTTPS.

Daneben wird meist das Betriebssystem regelmäßig mit Sicherheitsupdates versorgt. Leider lässt die Patchfrequenz nach, je höher man den Stack hinaufgeht. Die Middleware (wie Applikationsserver) wird nur ungern mit dem neuesten Patch-Level versehen, da man je nach Testabdeckung anschließend häufig viele Anwendungen erneut, im schlimmsten Fall manuell, testen muss.

Hier spielt die Cloud ihre Vorteile aus. Je nach Cloud-Modell (IaaS, PaaS, SaaS) reduziert sich der Teil der noch selbst zu verwaltenden Systemanteile um bis zu 100 Prozent. Die Cloud-Provider haben das Patching ihrer Systeme mit den neuesten Sicherheitsfixes wesentlich zuverlässiger und schneller im Griff als die meisten On-Premises-Rechenzentren. Ein weiterer Vorteil ist hier, dass Zugangsdaten für verwendete Services wie Datenbanken oder Message-Broker nicht mehr selbst verwaltet werden müssen (und häufig unverschlüsselt in irgendwelchen lokalen Konfigurationsdateien herumliegen).

Im Rahmen von SecDevOps sollen Entwickler, Betrieb und Sicherheitsfachleute in einem Team zusammenarbeiten. Das bedeutet, neben dem Security-Officer sollten auch Betriebsverantwortliche in das Scrum-Team mit integriert sein. Der bisher separierte Betriebsteil wird so immer mehr in der Entwicklung aufgehen und die Aufgaben des Betriebs ändern sich entsprechend. Gerade die Cloud entlastet die Mitarbeiter von den lästigen Systemverwaltungsaufgaben. Stattdessen können sie zusammen mit den Entwicklern und dem Security-Officer ein proaktives Monitoring aller kritischen Anwendungsteile aufsetzen. Man geht hier auch mehr und mehr dazu über, die Anwendungen mit Laufzeit-Intrusion-Detection-Systemen zu verknüpfen. Damit sind diese in der Lage, selbst Angriffsmuster (wie auffällig viele Angriffe auf bestimmte URL-Adressen) zu erkennen und direkt darauf zu reagieren (etwa durch Blockieren einzelner IP-Adressen oder temporäres Deaktivieren einzelner Anwendungsteile).

Fazit

Die Anwendungssicherheit kann heutzutage nicht mehr allein durch die Experten aus der Informationssicherheit getragen werden. Stattdessen ist im agilen Umfeld jeder Entwicklungsbeteiligte für die Sicherheit der eigenen Anwendung mit verantwortlich. Wie im Verlauf dieses Artikels dargestellt, ist das Scrum-Framework an jeder Stelle offen für eine enge Integration von Security in die Entwicklung. Es wird damit auch ermöglicht, damit verbundene Aktivitäten in der Wertschöpfungskette immer weiter nach vorne zu verschieben und Sicherheitsaspekte bereits in den Anforderungen zu berücksichtigen.

Abschließend wird nochmal die Ausgangslage aus *Abbildung 2* aufgegriffen. In *Abbildung 9* ist die ursprüngliche Sprint-Planung hinsichtlich Sicherheit stark optimiert worden. Es sind nun neben User Stories ohne Security-Relevanz (blau) auch User Stories mit Security-spezifischen Akzeptanzkriterien (rot-blau) sowie AbUser Stories (rot) in den Sprints eingeplant. Auch die funktionelle Security-Story ist nun frühzeitig für die ersten Sprints mit vorgesehen. In jedem Sprint wird durch automatisierte Security-Tests die Erstellung eines sicheren Produkt-Inkrements gewährleistet, sodass ein eingeschränkter Penetrationstest gegebenenfalls nur noch für die wirklich harten Prüfungsfälle erforderlich ist.

Weitere Informationen

[1] *https://github.com/OWASP/Top10/tree/master/2017*
[2] Gene Kim et.al.: The DevOps Handbook, 2016, IT revolution Press, ISBN 9781942788003
[3] *http://agilemanifesto.org/iso/de/manifesto.html*
[4] *https://threatmodelingbook.com*
[5] *https://github.com/OWASP/ASVS*
[6] *https://www.owasp.org/index.php/OWASP_Proactive_Controls*
[7] *https://spotbugs.github.io*
[8] *https://find-sec-bugs.github.io*
[9] *https://github.com/jeremylong/DependencyCheck*
[10] *https://github.com/zaproxy/zaproxy*

Andreas Falk
andreas.falk@novatec-gmbh.de

Andreas Falk hat mehr als zwanzig Jahre Erfahrung in der Entwicklung von Unternehmensanwendungen im Java-Umfeld. Seit mehr als sieben Jahren ist er als Managing Consultant bei der NovaTec Consulting GmbH tätig. Seine Schwerpunkte sind die Architektur, Entwicklung und das agile (Security-) Testen von Anwendungen.

Security – auch in kleinen Entwicklerteams

Dr. Stefan Schlott, BeOne Stuttgart GmbH

Am Thema „Security" kommt inzwischen keiner mehr vorbei – das steht außer Frage. Große Firmen und Konzerne etablieren für ihre Entwicklungsabteilungen entsprechende Prozesse, die einzuhalten sind, und bieten zentrale Anlaufstellen für Rückfragen, Reviews oder Penetration Tests. Kleinere Firmen hingegen können sich den Aufbau solch großer Strukturen kaum leisten. Dennoch ist es möglich, sich in kleinen Schritten einem adäquaten Sicherheitsniveau zu nähern.

Natürlich macht kein Prozess eine Anwendung automatisch sicher und selbstverständlich kommt es auch immer auf den jeweiligen Anwendungsfall an, welchen Aufwand man in die Sicherheit einer Anwendung investieren sollte – es käme ja auch niemand auf die Idee, sich mit dem „1x1 gegen Wohnungseinbruch" für Fort Knox zufriedenzugeben. Aber zumindest die typischen Fehler und „Low Hanging Fruits" für Penetration-Tester und Angreifer lassen sich vermeiden.

Auf die wesentlichen Ziele beschränken

Die Bandbreite von Security-Maßnahmen reicht von vollkommener Ignoranz bis zur vollständigen Paranoia, und beide Extreme sind Projekten nicht zuträglich. Sinnvoll ist es, wenn wir uns auf die wesentlichen Sicherheitsziele einigen – ganz genauso, wie wir mit dem Kunden besprechen, welche Features besonders wichtig sind und welche erst in einer späteren Version umgesetzt werden. Hier geht es uns zunächst um das „Was": Vor was möchten wir uns schützen? Was sind Gefahren für den Anwender unserer Software beziehungsweise für unseren Kunden? Antworten können die Sorge vor finanziellen Verlusten sein (etwa der Verlust irgendwelcher Geschäftsgeheimnisse, drohende Strafzahlungen etc.), aber auch die Sorge um die Reputation (wer will schon mit einer Negativschlagzeile auf dem Titelblatt der Tageszeitung stehen) oder das Einhalten gesetzlicher Rahmenbedingungen (die DSGVO ist da häufig nur ein Aspekt).

Diese Diskussion lässt sich sehr untechnisch und damit für den Kunden sehr verständlich führen. Wenn man diese Sicherheitsziele später mit entsprechenden Gegenmaßnahmen und dahinter steckenden Aufwänden verknüpft, kann der Kunde mit üblichen Methoden des Risikomanagements entscheiden (vereinfacht: Kosten vs. potenzieller Schaden und dessen Eintrittswahrscheinlichkeit), welche der Sicherheitsziele umgesetzt werden sollen und welche Risiken er bereit ist, in Kauf zu nehmen.

Der nächste Schritt des Threat Modeling ist die Frage, mit welchen Mitteln ein Angreifer unsere Sicherheitsziele verletzen kann. Diese geben uns dann einen direkten Hinweis auf technische Gegenmaßnahmen. Eine gebräuchliche Klassifikation sind die „STRIDE"-Kategorien *(siehe Tabelle 1)*.

Bei komplexeren Systemen lohnt es sich, dies in seine groben Komponenten aufzugliedern (Application Decomposition). Durch Festlegen von Trust Boundaries wird nochmals klargestellt, innerhalb welcher Bereiche wir uns auf die Korrektheit welcher Informationen verlassen können und wo Angaben gegebenenfalls nochmals überprüft werden müssen. Hilfreich können hier Tools wie das kostenlose Microsoft Threat Modeling *(siehe „https://www.microsoft.com/en-us/download/details.aspx?id=49168")* sein. Es liefert nicht nur ansehnliche Grafiken für die Architektur-Dokumentation, sondern auch eine generische Liste von Sicherheitsrisiken (passend zur „STRIDE"-Klassifikation) für jede Komponente.

Wann was zu tun ist

Je früher man ein Problem erkennt, desto geringer ist der Aufwand, es zu beheben. Was für architekturelle Fehler oder fachliche Missverständnisse gilt, gilt ebenso für Security-Aspekte. In einer idealen Welt hätte man in jeder Phase der Entwicklung einen Vollblut-Security-Experten im Team – ein Wunsch, der sich leider nicht überall realisieren lässt. Was sich allerdings in jedem Team etablieren lässt, ist das Bewusstsein dafür, dass es zu jeder Phase der Software-Entwicklung entsprechende Sicherheitsaspekte gibt.

Viele Anforderungen sind mit einem der Sicherheitsziele verknüpft oder können mit den daraus abgeleiteten Schutzmaßnahmen in

Angriff	Vorgehen	Verletztes Prinzip
Spoofing	Vortäuschen einer Identität	Authenticity
Tampering	Verändern von Daten	Integrity
Repudiation	Abstreiten von Aktionen	Non-Repudiation
Information Disclosure	Unberechtigte Kenntnisnahme von Informationen	Confidentiality
Denial of service	Verhinderung der Dienste-Nutzung	Availability
Elevation of privilege	Unberechtigte Nutzung erhöhter Systemprivilegien	Authenticity

Tabelle 1

Verbindung gebracht werden. Umgekehrt gilt, dass neue oder geänderte Anforderungen in Bezug auf die Sicherheitsziele hinterfragt werden müssen. Gegebenenfalls sind die Sicherheitsziele und daraus folgende Maßnahmen zu ergänzen oder anzupassen. Im Scrum-Prozess sind dies Aufgaben, die bei der ersten Befüllung des Backlogs sowie bei der Backlog-Pflege anfallen.

Architektur-Entscheidungen sind ein weiterer Moment, an dem das Thema „Security" auf dem Radar erscheint: Hier sind auch konzeptionelle Security-Entscheidungen zu treffen – sei es bei der Strukturierung der Anwendung oder der Auswahl der verwendeten Frameworks und Bibliotheken. Aus Security-Sicht sind solche Bibliotheken zu bevorzugen, die zum einen gut gepflegt sind und zum anderen möglichst gute Voreinstellungen haben („Secure by default"). Es soll den Entwicklern später so leicht wie möglich gemacht werden, Dinge richtig zu tun.

Bleibt noch das eigentliche Doing, also Implementierung und Testing. Hier ist das Team so zu schulen, dass es die verwendeten Tools und Bibliotheken richtig (also sicher) verwendet und auch ansonsten handwerkliche Fehler vermeidet. Dies wird nur gelingen, wenn bei allen Entwicklern ein Bewusstsein für die Probleme vorhanden ist. Einen guten Anhaltspunkt für einen Einstieg und die brennendsten Themen liefert das Open Web Application Security Project (OWASP): Wirft man einen Blick auf das Top-10-Projekt *(siehe „https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project* und Vorjahre"), sind Probleme mit Injection-Angriffen und der Authentisierung am häufigsten. Cross-Site-Scripting (XSS) und Cross-Site-Request-Forgery (XSRF) spielen in der letzten Ausgabe der Top 10 in den hinteren Plätzen mit, bei Anwendungen mit großem Logik-Teil auf dem Browser (etwa bei der Anwendungsentwicklung mit Frameworks wie Angular) sollte ihnen jedoch ein besonderes Augenmerk zukommen.

Coaching mit Spaß

Ein Augenöffner ist häufig eine Demonstration, was mit Lücken wie SQL-Injection oder XSS möglich ist: Es ist schließlich eine Sache, ob man sich ein passwortloses Login erschleichen oder ein nerviges JavaScript-Browser-Popup öffnen kann – oder aber (mithilfe frei verfügbarer Tools) ein kompletter SQL-Dump extrahiert oder der Browser des Opfers als Angriffs-Trittbrett ins Firmen-Intranet oder auf weitere Webseiten genutzt wird.

Ebenfalls spannend ist der Wechsel in die Angreifer-Perspektive: Mit wenig Aufwand lassen sich Übungsumgebungen bereitstellen, mit denen man obige Angriffe einmal selbst ausprobieren kann. Entsprechend präparierte Anwendungen wie Damn Vulnerable Web Application (DVWA, siehe *„http://www.dvwa.co.uk")* oder OWASP WebGoat *(siehe „https://github.com/WebGoat/WebGoat")* lassen sich als Docker-Container rasch an den Start bringen. Wenn das Team daran etwas Spaß findet: Warum nicht einen kleinen Wettbewerb starten? Auf einem CTF-Server (Capture The Flag, z.B. FBCTF oder CTFd) sammeln Teams Punkte für sogenannte „Flags", die sie durch Ausnutzen von Lücken in einer anfälligen Anwendung finden. Der OWASP Juice Shop *(siehe „https://github.com/bkimminich/juice-shop")* bietet für eine solche Integration einen Extra-Startmodus.

Auch für die sehr trocken anmutende Phase des Threat Modeling gibt es spielerische Unterstützung: Ebenfalls auf den Seiten der OWASP findet man das Kartenspiel „Cornucopia" *(siehe „https://www.owasp.org/index.php/OWASP_Cornucopia").* Es ist ein einfaches Ablegespiel – möglicherweise nicht so spannend wie eine James-Bond-Poker-Runde, aber dennoch ein gutes Mittel, um die Diskussion über Sicherheitsanforderungen an ein Testprojekt (oder das eigene Projekt) anzuregen.

Fazit

Es wäre übertrieben zu erwarten, dass ein Entwicklerteam von heute auf morgen nur noch fehlerfreie Software schreibt; auch Security ist ein Lernprozess. Das Bewusstsein für die Probleme ist jedoch eine Grundvoraussetzung, die gelegt werden kann. Die hier vorgeschlagenen Methoden können dabei helfen, die ersten Schritte zu gehen.

Dr. Stefan Schlott
stefan.schlott@beone-group.com

Dr. Stefan Schlott ist Advisory Consultant bei der BeOne Stuttgart GmbH und dort als Entwickler, Architekt und Trainer im Java-Umfeld tätig. Security und Privacy gehören ebenso zu seinen Schwerpunkten wie skalierbare Architekturen und das breite Feld der verschiedenen Web-Technologien. In seiner Freizeit ist er auch als Dozent an der Dualen Hochschule Baden-Württemberg aktiv. Er begeistert sich für funktionale Programmierung sowie die Sprache Scala und ist überzeugter Open-Source-ler.

Containerisierte CI/CD-Pipelines mit OpenShift

Tobias Schneck, Loodse

Container erfreuen sich unter Entwicklern immer größerer Beliebtheit. Kein Wunder, denn sie federn Umgebungs-Unterschiede ab, verbessern die Vorhersehbarkeit und sind skalierbar. Dadurch vereinfachen und beschleunigen sie die Entwicklung und das Deployment neuer Features und verringern zusätzlich die Kosten für komplexe Umgebungen.

Um die Vorteile voll auszuschöpfen, braucht es spätestens im zweiten Schritt auch eine stabile und skalierbare Continuous-Integration/Continuous-Delivery-Umgebung (CI/CD). Diese sind seit jeher schwer aufzusetzen und zu pflegen; beim Aufbau einer containerisierten Build-Pipeline kommen zusätzlich neue Herausforderungen, aber auch Chancen hinzu. Dieser Artikel stellt einen der bislang gängigsten Ansätze bei der Nutzung von Jenkins-CI-Servern in Kubernetes-Clustern vor, die OpenShift-Build-Pipelines.

Für den Aufbau von flexibel skalierbaren CI/CD-Pipelines bietet sowohl Kubernetes mit dem Jenkins-Kubernetes-Plug-in als auch OpenShift mit den integrierten Jenkins-Build-Pipelines eine echte Open-Source-Option für einen containerisierten CI/CD-Server an. OpenShift ist eine Open-Source-Platform-as-a-Service-Lösung (PaaS) von Red Hat auf Kubernetes-Basis, die es Entwicklern ermöglicht, ihre Anwendungen zu bauen, zu testen und sie in private oder öffentliche Clouds zu deployen. Sowohl bei Plain Kubernetes als auch bei OpenShift kann der CI/CD-Server innerhalb oder außerhalb des Clusters betrieben werden.

Dem modernen „Infrastructure-as-a-Code"-Ansatz folgend, fokussiert sich dieser Artikel auf den Betrieb im Cluster selbst. Die größte Herausforderung dabei ist es, ein lauffähiges Container-Image dynamisch zu erstellen, zu testen und zu releasen, ohne die eigene Cluster-Infrastruktur zu verlassen. Dies ermöglicht einen durchgängig gelebten DevOps-Ansatz.

OpenShift integriert zwei Arten, wie Images gebaut werden können: native Docker-Builds mit Dockerfiles und sogenannte „Source-2-Image-Builds" (S2I, siehe *„https://github.com/openshift/source-to-image").* Zur Steuerung der beiden Build-Typen wurde in OpenShift ein auf Kubernetes-Anforderungen konfigurierter Jenkins-CI-Server als „Out-of-the-box"-Komponente hinzugefügt. Die durch das OpenShift-Client-Plug-in erweiterte Jenkins DSL unterstützt neben Java auch „Node.js"-Builds, die zur Laufzeit in spontan gestarteten Pods durchgeführt werden.

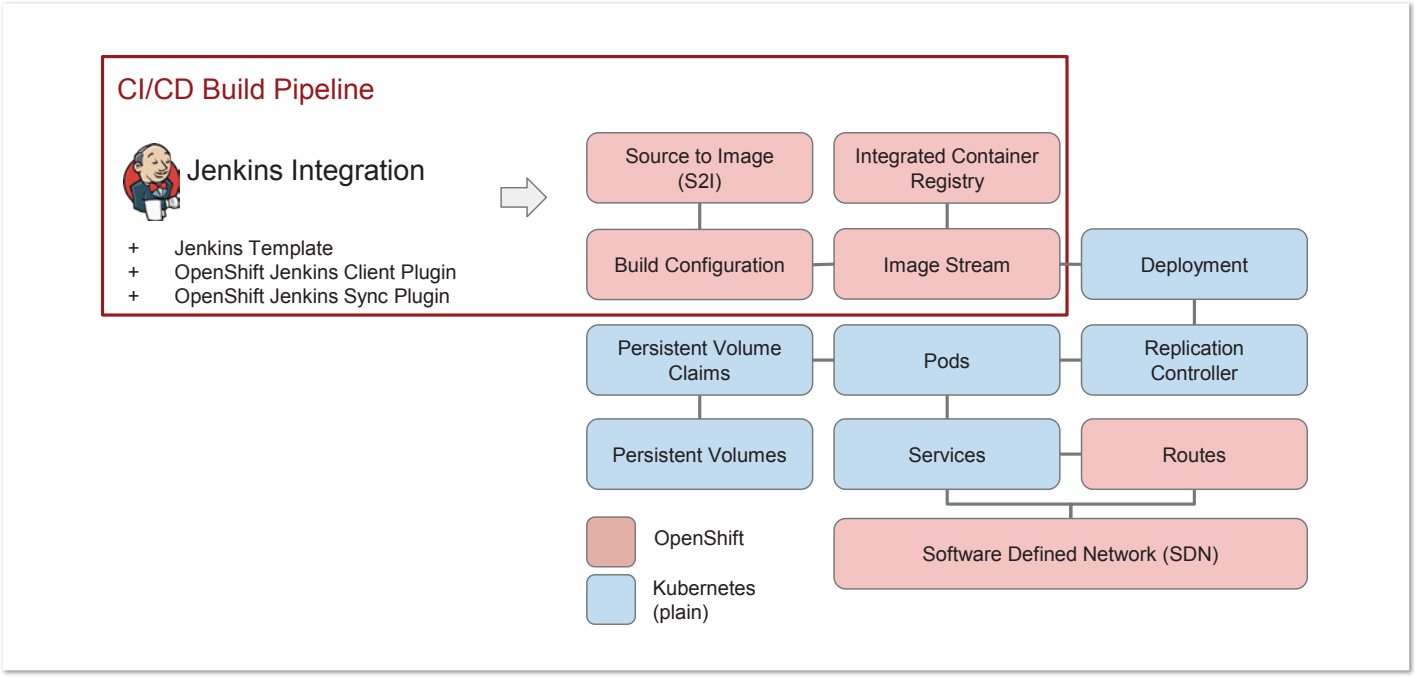


Abbildung 1: Die OpenShift-Komponenten

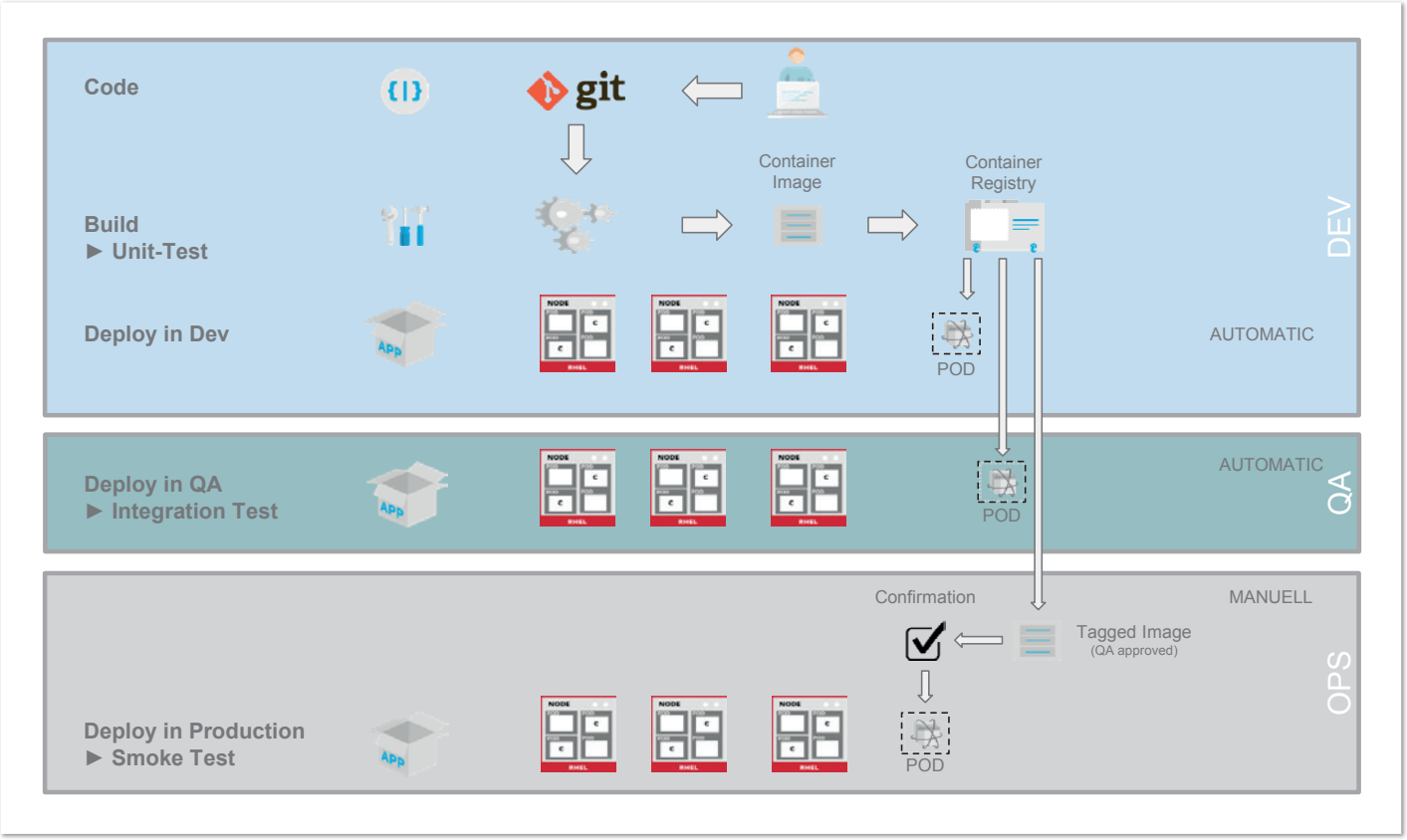


Abbildung 2: Build Workflow und Deployment Stages

Dieser Mechanismus ist erweiterbar, sodass bei Bedarf Unterstützung für weitere Sprachen hinzugefügt werden kann. Dadurch, dass Kubernetes-Pods als Ad-hoc-Build-Ressource genutzt werden, liegen bei hoher Nachfrage die Build-Limits allein bei den verfügbaren Cluster-Ressourcen. Die Jenkins-Slave-Nodes selbst skalieren automatisch, womit das manuelle Ressourcen-Management entfällt.

Die OpenShift-Komponenten

OpenShift ergänzt Kubernetes um einige hilfreiche Komponenten, die sowohl das Deployment selbst als auch den Aufbau einer Build-Pipeline vereinfachen:

- Routes, eine dynamische DNS-Komponente, die einfache Aufrufe der Applikation über eine fest definierbare URL erlaubt.

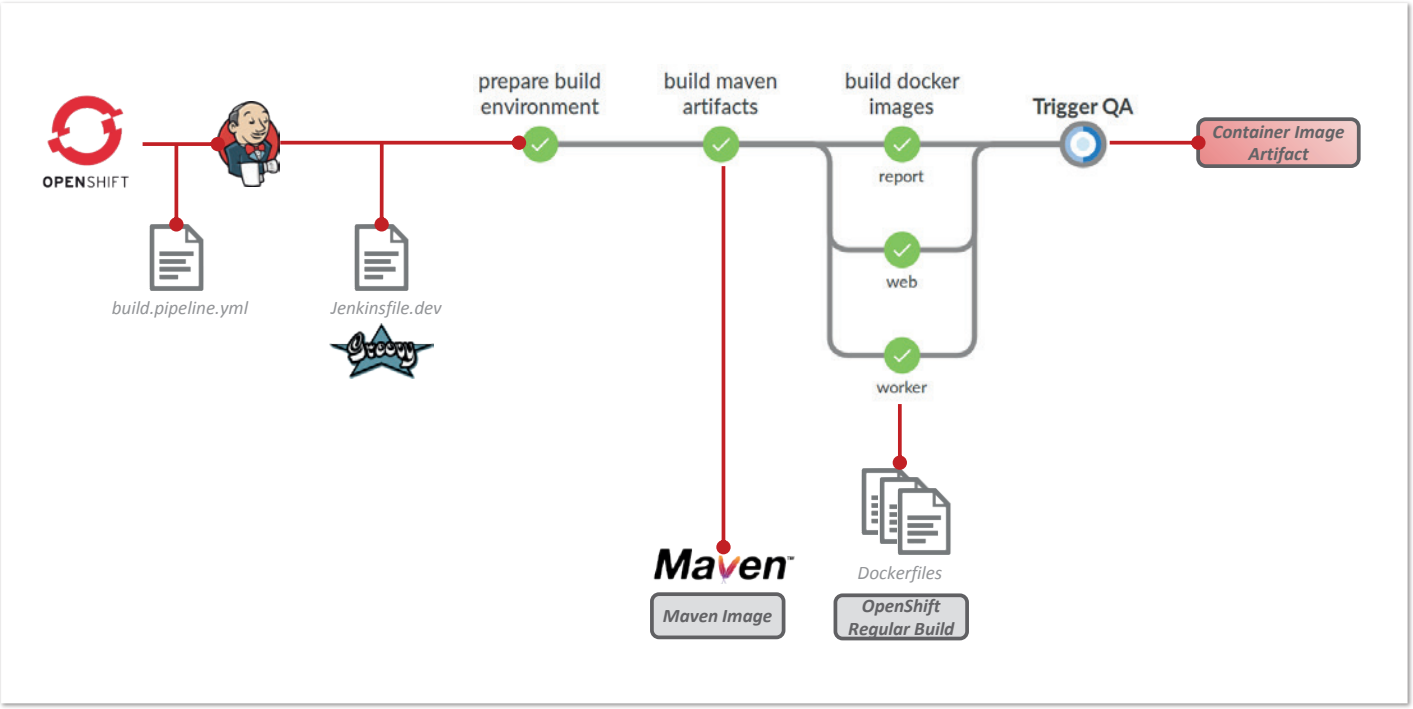


Abbildung 3: Jenkins-Build-Pipeline – DEV-Stage

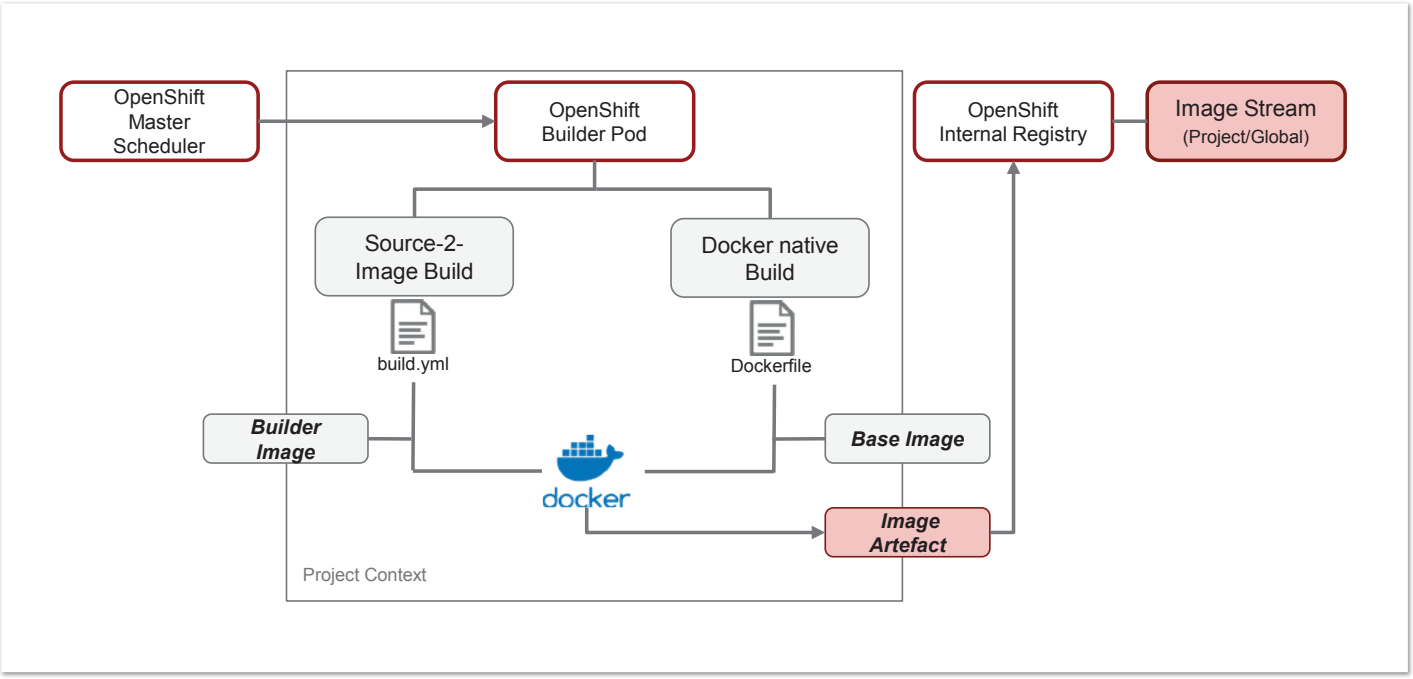


Abbildung 4: OpenShift-Regular-Builds (Source- und Docker-Strategie)

- Software Defined Network (SDN) als Netzwerk-Schicht zwischen OpenShift-Clustern und der Außenwelt.
- Image Stream als interne Objekt-Definition eines Docker-Image.
- Integrierte Container-Registry zum Speichern und Verteilen gebauter Container-Images.
- Build Configuration, die festlegt, welcher Build-Typ (Docker native, S2I, Jenkins-Pipeline) durchgeführt wird. Es definiert zudem die benötigte Build-Umgebung (wie Source Repository, Secrets und Webhook).
- „Source to Image"-Objekt, das einen fixen, vorkonfigurierten Build-Ablauf definiert, der lediglich das Sourcecode-Repository als Übergabe-Parameter benötigt. Die S2I-Builder-Images können selbst erstellt werden oder es werden die mitgelieferten Images wie für Node.js genutzt *(siehe „https://docs.openshift.com/container-platform/latest/using_images/s2i_images/index.html")*.

Damit verfügt OpenShift selbst noch nicht über ausführende CI/CD-Komponenten; sie werden erst mit der Jenkins-Integration ermöglicht. Diese besteht aus den folgenden Komponenten:

- Das Jenkins-Template legt fest, mit welchen Parametern ein Jenkins im OpenShift-Cluster eingerichtet wird. Am Template können Cluster-Admins Anpassungen durchführen und beispielsweise CPU, Memory, Secrets, Umgebungsvariablen oder Persistent Storage konfigurieren.

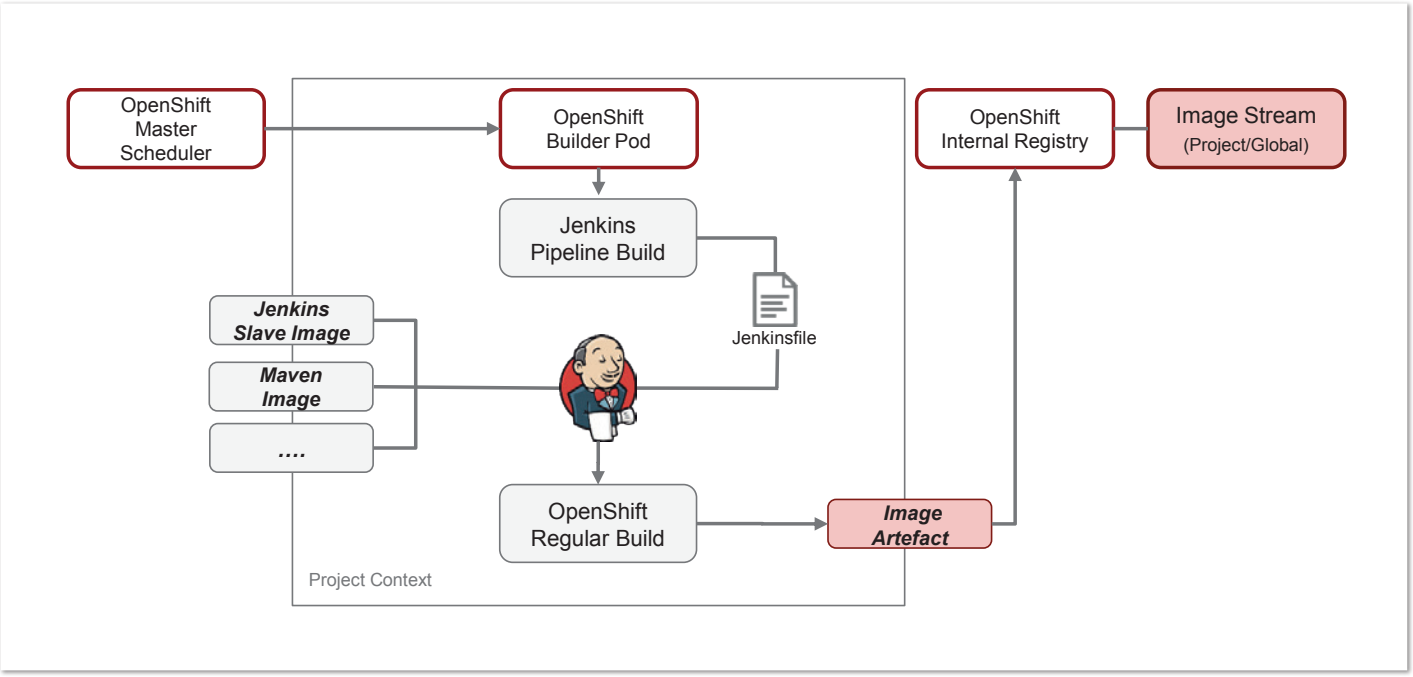


Abbildung 5: OpenShift-CI-Pipeline-Build (Jenkins-Pipeline)

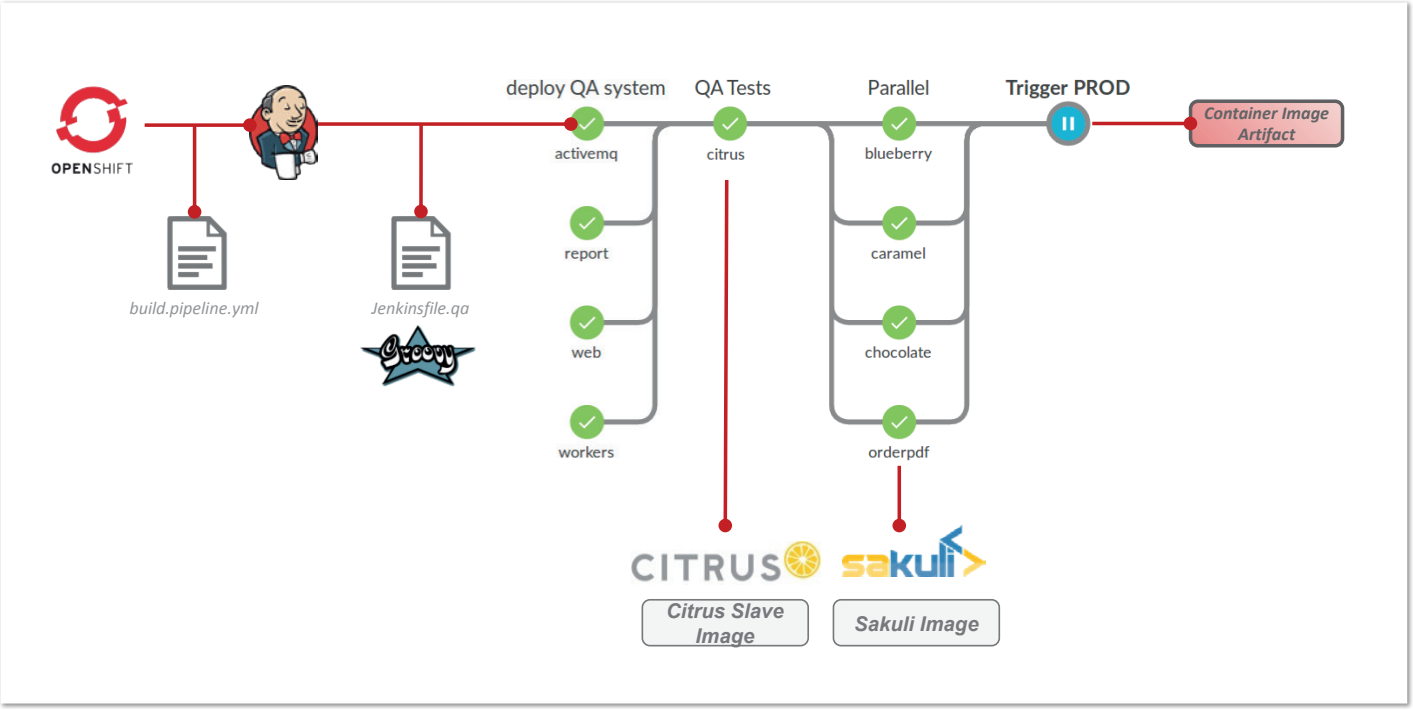


Abbildung 6: Jenkins-Build-Pipeline – QA Stage

- Das OpenShift-Jenkins-Client-Plug-in gewährleistet, dass innerhalb eines Jenkins-Builds direkt mit den OpenShift- und Kubernetes-Ressourcen kommuniziert werden kann, etwa um einen nativen OpenShift-Docker-Build anzustoßen oder ein Deployment auszulösen.
- Das OpenShift-Jenkins-Sync-Plug-in stellt sicher, dass das OpenShift-Dashboard und das entsprechende Build-Objekt über die Ergebnisse eines angestoßenen Jenkins-Jobs informiert werden.

Build Workflow und Deployment-Stages

Im OpenShift wird ein Build im Cluster als Objekt vom Typ „BuildConfig" gespeichert, im YAML-Format beschrieben und per Commandline oder UI im Cluster erzeugt. Das Objekt erwartet im Anschluss, um mit dem Build zu beginnen, ein Start-Event, das durch den sogenannten „Hook" oder einen CLI-Aufruf erzeugt wird. Der erzeugte Hook ist per Web-URL erreichbar und kann in Drittsystemen wie dem Versions-Verwaltungssystem (GitHub, GitLab oder ähnliche) hinterlegt sein. Dadurch wird bei jeder Änderung des Sourcecodes ein neuer Build angestoßen. Eine beispielhafte Konfiguration ist in diesem Projekt auf GitHub zu finden (siehe „toschneck/openshift-example-bakery-ci-pipeline-openshift.build.bakery.generic.yaml"). Ziel des automatisieren Build-Prozesses ist es, in drei Stages ein produktionsreifes Artefakt zu deployen *(siehe Abbildung 2)*:

- **Development Stage**
  Bau und Bereitstellung eines unveränderbaren Container-Artefakts
- **QA Stage**
  Automatisiertes Testen in einer produktionsnahen Umgebung durch geeignete Integration-Tests
- **Production Stage**
  Deployment auf das Produktivsystem und Smoke-Testing

Development Stage

Zum Bau eines unveränderbaren Artefakts wird zuerst der zugehörige Sourcecode ausgecheckt. Dann wird dieser mit einem Build-Tool wie Maven bei Java-Projekten in ein lauffähiges Applikations-Artefakt umgewandelt. Zu diesem Zeitpunkt sollten auch Unit-Tests durchgeführt werden, um auf Funktionsebene sicherzustellen, dass der Code keine Fehler enthält. Schlagen die Tests fehl, wird kein Artefakt erstellt und der Build abgebrochen. Das ist wichtig, um dem Entwickler möglichst schnell Feedback zu geben.

Ist der Test erfolgreich, wird die Applikation in ein Container-Image gepackt und mit einer eindeutigen ID in eine zentrale Container-Registry gepusht, um für die weiteren Stages zur Verfügung zu stehen. *Abbildung 3* zeigt, wie das in unserer Bakery-Beispiel-Applikation erfolgt.

Die „build.pipeline.yml"-Datei beschreibt, dass ein Build vom Typ „JenkinsPipeline" durchgeführt werden soll, der im Jenkinsfile „Jenkinsfile.dev" genauer beschrieben ist. Die zugehörigen Dateien sind auf GitHub unter „toschneck/openshift-example-bakery-ci-pipeline/openshift" einsehbar. Im „Jenkinsfile.dev" ist definiert, dass ein Node vom Typ „maven" das angegebene Java-Projekt kompiliert, testet und ein JAR-Artefakt erzeugt.

Da der Build innerhalb eines OpenShift-Clusters läuft, wird nun durch das OpenShift-Jenkins-Client-Plug-in ad hoc ein vorkonfiguriertes Maven-Container-Image als Pod gestartet. Im Vergleich zur alten Welt mit statischen Jenkins-Slaves ist es nun möglich, dynamische Workloads auf den Clustern auszuführen. Das ist vor allem bei großen Enterprise-Umgebungen interessant, da keine ungenutzten Compute-Ressourcen vorgehalten werden müssen.

Ist dieser Schritt erfolgreich, wird als Nächstes ein „Regular" OpenShift-Build-Objekt erzeugt, da Jenkins selbst keinen Zugang zum Docker-Socket hat und somit auch kein Docker-Image bauen kann. Hierbei ist es wichtig, die Unterschiede zwischen Regular-Builds und Pipeline-Builds zu verstehen.

Abbildung 4 zeigt, wie bei den Regular-Builds S2I- oder Docker-native Builds durchgeführt werden können. Diese haben als einzige Objekte im OpenShift-Cluster Zugriff auf den Docker-Daemon, können ein Container-Image erstellen und an die interne Container-Registry pushen. Dadurch wird ein neues Image-Stream-Objekt erzeugt beziehungsweise ein vorhandenes aktualisiert, was die weiteren Schritte in der Build-Pipeline triggert.

Bei den Pipeline-Builds wird hingegen anhand des angegebenen Jenkinsfiles ein normaler Jenkins-Build getriggert, der per Default keine Container-Images erzeugen kann. Um dies zu erreichen, ist ein weiteres Build-Objekt zu erzeugen, nämlich eines der beiden Regular-Builds. Die durch das OpenShift-Jenkins-Client-Plug-in bereitgestellten Credentials können dabei „out of the box" im Jenkinsfile genutzt werden, um per CLI oder Groovy-API den Regular-Build zu erzeugen. Technisch gesehen wird dafür ein angepasstes Jenkins-Slave-Image genutzt, das per Umgebungsvariablen die benötigten Cluster-Credentials gesetzt bekommt. Jenkins dient dabei mehr oder weniger als Wrapper, um das Java-Artefakt zu bauen und danach den eigentlichen Container-Build anzustoßen.

Aufgrund der derzeit genutzten Docker-Version in OpenShift sind keine Docker-Multi-Stage-Builds möglich, die diesen zusätzlichen Wrapper vermeiden könnten. Alternativ kann der Entwickler auch ein eigenes S2I-Builder-Image schreiben, bauen und in OpenShift einbinden, was allerdings zusätzlichen Pflegeaufwand bedeutet. Aus den genannten Gründen haben derzeit alle Wege eine gewisse Komplexität, die der Entwickler selbst lösen muss *(siehe Abbildung 5)*.

Zudem kann es nützlich sein, die gebauten Container-Artefakte nach jedem Build eines bestimmten Branch (etwa bei Feature-Branches) automatisch zu deployen. Wichtig ist, dass die genutzte Deployment-Konfiguration identisch mit der Produktions-Konfiguration ist. Dafür können bei OpenShift native YAML-Templates genutzt werden, die für die jeweilige Stage per Parameter angepasst werden. Beispielsweise kann das Feature-Branch-Deployment ein Präfix benutzen und steht danach unter einer eigenen URL *„feature-X.dev.bakery.mycluster"* zur Verfügung.

QA Stage

Nach dem erfolgreichen Durchlaufen der Development-Stage sind die erzeugten Artefakte nun ausgiebig zu testen. Je nach Projekt werden dafür manuelle oder automatische Aktionen notwendig. Im einfachsten Fall wird die Anwendung bei einer Code-Änderung im Haupt-Entwicklungspfad auf die QA-Umgebung eingerichtet und manuell getestet. Das sollte allerdings nicht das Ziel sein, da neue Features oder ein Bugfix möglichst schnell in Produktion gehen sollten. Besser ist es, abgeschlossene Features, die zum Beispiel durch einen Pull-Request gekennzeichnet sind, automatisiert mit einer Vielzahl von Integration-Tests zu testen. Im Idealfall ist die aufgebaute Testsuite so mächtig, dass auf manuelle Tests verzichtet werden kann.

Unsere Beispiel-Applikation beinhaltet sowohl API-Schnittstellen als auch eine grafische Web-Oberfläche. Beides sollte ausreichend getestet sein, um sicherzustellen, dass die Code-Änderung keine ungewollten Nebeneffekte hat. Da die Applikation in der Dev Stage durch Unit-Tests auf tieferer Ebene getestet wurde, können jetzt verschiedene Black-Box-Tests auf einem produktionsnahen System folgen. Der in *Abbildung 6* dargestellte Ablauf wurde dabei wie folgt umgesetzt:

- Deployment des Zielsystems mithilfe eines OpenShift-Deployment-Templates
- Testen des REST-API mithilfe von Citrus-Integration-Tests
- Testen der Web-UI und des PDF-Reports mit Sakuli-E2E-Tests

Zunächst muss das sogenannte „System Under Test" (SUT) eingerichtet sein. Wie bereits beschrieben, ist es wichtig, in ein nahezu produktionsnahes System zu deployen, das sich lediglich durch Parameter wie Verbindungsdaten (HTTP-Endpoints, Message-Broker-Host, Credentials) unterscheidet. Um die Durchlaufzeit der Pipeline zu reduzieren, ist das Deployment so zu optimieren, dass die Komponenten parallel gestartet werden können.

Besteht eine Abhängigkeit zu anderen Services, ist es hilfreich, das sogenannte „Init Containers"-Konstrukt in der Pod-Definition zu

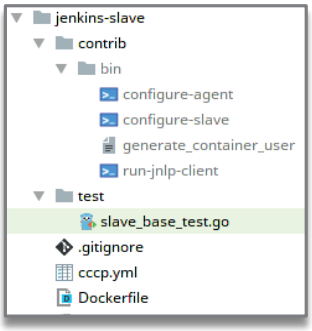
```
# Dockerfile
FROM consol/citrus:2.7.5

### sourced are copied from:
https://github.com/openshift/jenkins

# Copy the jenkins-slave entrypoint
ADD contrib/bin/* /usr/local/bin/

# Run the Jenkins JNLP client
ENTRYPOINT ["/usr/local/bin/run-jnlp-client"]
```

Used by



Jenkinsfile

```
podTemplate(label: "citrus",
    cloud: "openshift",
    inheritFrom: "maven",
    containers: [
        containerTemplate(name: "jnlp",
            image: "citrusframework//jenkins-slave",
        )
])
{
    node('citrus') {
        sh "echo execute oc citrus build"
        checkout scm
        sh "mvn install"
        junit 'citrus-tests/target/citrus-reports/**/*.xml'
        archiveArtifacts "citrus-tests/target/citrus-*/**/*"
    }
}
```

Abbildung 7: Custom-Jenkins-Slave-Image

nutzen. Dort kann mit einfachen Skripten überprüft werden, ob die konfigurierte URL oder der Host bereits erreichbar ist, ohne dass der eigentliche Container startet. Das vermeidet, dass beim Starten der eigentlichen Anwendungscontainer Ressourcen-hungrige Crash-Loops entstehen, da gerade Java-Anwendungen einen hohen Ressourcenverbrauch beim Start haben.

Sind alle Komponenten erreichbar, kann das API getestet werden. Hierfür wird im Beispiel das Citrus Integration Testing Framework *(siehe „https://citrusframework.org")* benutzt, das es ermöglicht, Unit-Test-artige Java-Tests zu schreiben, die sämtliche gängigen Message-Protokolle (HTTP REST, SOAP, JMS etc.) und Nachrichtenformate (XML, JSON, XSD etc.) beherrschen. Zudem bietet Citrus die Möglichkeit, nicht nur einfache Request-Response-Szenarien zu testen, sondern auch komplexe, Nachrichten-basierte Message-Workflows abzubilden, bei denen beispielsweise Message-Payloads über mehrere Nachrichten und Endpoints hinweg verarbeitet werden.

Um den von Citrus bereitgestellten Container möglichst nahtlos zu integrieren, kann die Groovy-DSL mit einem eigenen Jenkins-Slave-Image erweitert werden. Dieses Vorgehen bietet sich an, da die Tests wie die Unit-Tests selbst per Maven getriggert werden. Eine kurze Beschreibung steht in *Abbildung 7*.

Zunächst wird ein Jenkins-kompatibles Image benötigt. Um ein eigenes Image zu bauen, empfiehlt es sich, den Sourcecode des Maven-Slave-Image *(siehe „https://github.com/openshift/jenkins/tree/master/slave-maven")* zu kopieren und seine eigene Implementierung darauf aufzusetzen. Das Image lässt sich anschließend als neuer Node „citrus" ansprechen. Die darin aufgeführten Befehle wie „mvn clean install" werden dann innerhalb des zur Laufzeit gestarteten Containers ausgeführt. Sind alle Integration-Tests erfolgreich, wird im Anschluss die UI getestet.

Für die UI-Tests wurde im Beispiel Sakuli *(siehe „http://www.sakuli.org")* genutzt. Sakuli erlaubt es, sowohl Web- als auch native Oberflächen innerhalb eines Kontextes zu testen. Konkret wurden vier parallel laufende Testfälle definiert. In den drei Web-basierten Tests wurden unterschiedliche Produkte über die Web-Oberfläche bestellt und im Anschluss die erfolgreiche Abarbeitung auf dem Report-Server überprüft. Im vierten Fall wurde über die native Browser-Funktion „Als PDF speichern" eine PDF-Version des Produktions-Reports erzeugt und im Anschluss in einem nativen PDF-Viewer validiert.

Die Kombination von Web und OpenCV-basierter, visueller Testmethodik macht es möglich, komplexe End-to-End-Workflows zu testen. Damit ist sichergestellt, dass die Anwendung aus End-User-Perspektive wie gewünscht funktioniert. Diese High-Level-UI-Tests sind notwendig, da bei der Vielzahl der Technologien und Frameworks, die ein modernes Entwicklungsprojekt einsetzt, immer ein unvorhersehbarer Seiteneffekt auftreten kann, der nur beim finalen End-to-End-Szenario sichtbar wird. *Abbildung 8* zeigt, wie ein solcher Test mithilfe der von Sakuli bereitgestellten Container innerhalb eines OpenShift-Clusters ausgeführt wird und der Entwickler per Web-VNC den Testlauf beobachten kann.

Damit die Laufzeit optimiert ist, werden die Testfälle parallel ausgeführt. Bei größeren Testsuites lassen sich unter anderem verschiedene Browser in unterschiedlichen Versionen testen. Technisch gesehen wird innerhalb der Build-Pipeline dafür für jede Test-Instanz ein Pod im Cluster gestartet. Anschließend wird mit dem kleinen Helper-Skript „validate_pod-state.sh" überprüft, ob der Pod mit Status „Succeeded" oder „Failed" bereits fertig durchlaufen wurde. Der daraus resultierende Exitcode des Containers wird validiert und Logs sowie Screenshots zur eventuellen Fehler-Analyse in einen persistenten Speicher kopiert. Ist einer der parallelen Testläufe nicht erfolgreich, wird die QA-Build-Pipeline als fehlerhaft markiert, was im „Jenkinsfile.qa" konfiguriert wurde.

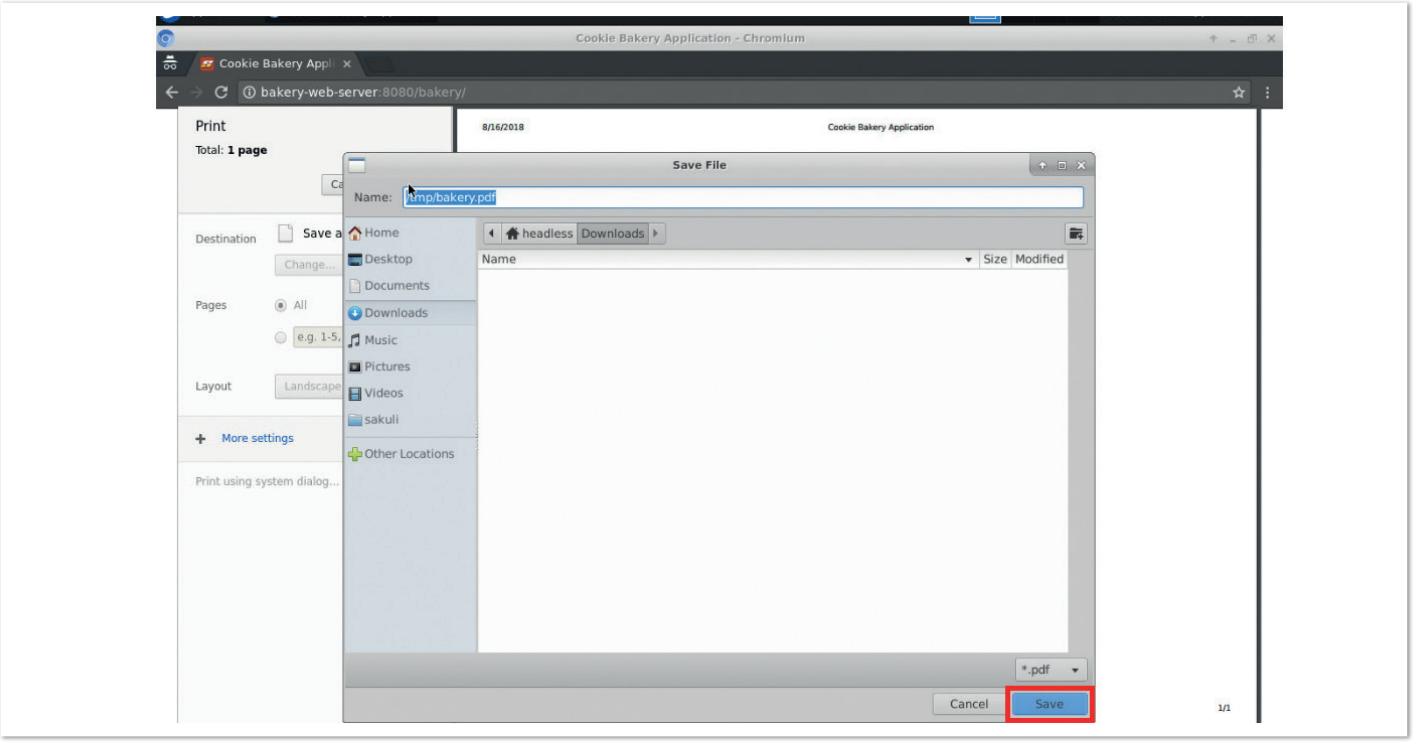


Abbildung 8: Sakuli-UI-Test in OpenShift

Die umfassenden automatisierten Tests können den Reifegrad des gebauten Image verifizieren. Ist alles in Ordnung, wird es für die nächste Stage freigeben. Dies kann entweder mit einem Update eines ImageStream-Objekts erfolgt oder mit dem Aufruf einer weiteren Build-Pipeline.

Production Stage

Wurde die QA Stage erfolgreich durchlaufen, kann das Deployment in das produktive System erfolgen. Meist wird dafür ein eigener Namespace (abgebildet als separates Projekt in OpenShift) oder ein separates Cluster genutzt, um die Umgebung entsprechend der Firmen-Policies abzusichern. Um Audit-Vorgaben für das produktive Deployment gerecht zu werden, lässt sich in die Jenkins-Pipeline eine User-Input-Aktion einbauen *(siehe Listing 1)*. Im Jenkins-Log ist dann zu sehen, welcher User das „OK" gesetzt hat. So kann ein hoher Automatisierungsgrad erreicht werden, ohne die Möglichkeit des manuellen Eingreifens zu nehmen.

Es ist ratsam, die Pipeline so zu gestalten, dass man das gleiche Deployment-Template wie in der QA Stage benutzen kann und nur Parameter anzupassen sind. Nach dem produktiven Deployment sollte ein sogenannter „Smoke-Test" durchgeführt werden, der nochmals sicherstellt, dass alles erfolgreich war. Je nach Komplexität können hier ein Subset der Integration-Tests oder ein kleiner Connection-Test wie ein sogenannter „Wait-for-it"-Container *(siehe „https://github.com/toschneck/wait-for-it")* benutzt werden. Das Ergebnis des gesamten Builds beziehungsweise des Deployments sollte per E-Mail oder Chat Notification dem verantwortlichen Team mitgeteilt werden, um im Fehlerfall schnell reagieren zu können.

Fazit

Beim Aufbau der Build-Pipelines mit OpenShift wird deutlich, dass Jenkins als statische Java-Anwendung nicht für den Bau und das Verwalten von Pods und Containern entwickelt wurde. Selbst mit den von OpenShift gelieferten Plug-ins ist es mühsam, das OpenShift-API direkt anzusprechen. Im Unterschied zum klassischen Jenkins kann bei der OpenShift-Integration das Jenkinsfile derzeit nicht einfach angepasst und ein neuer Build gestartet werden. Bei jeder Änderung am Jenkinsfile ist ein neuer Commit zu erzeugen, was wiederum bedeutet, dass die komplette Pipeline erneut bis zur eigentlichen Änderung ausgeführt wird. Das Resultat ist, dass das Schreiben und Pflegen der Pipeline sehr viel Aufwand verursacht. Statt das Groovy-API zu nutzen, empfiehlt sich als Workaround, direkt über Skripte mit dem „oc"-Client den API-Befehl für das Cluster abzusetzen. Der Vorteil ist, dass die einzelnen Schritte der Pipeline in Skripte verpackt sind und unabhängig voneinander ausgeführt beziehungsweise entwickelt werden können. Zudem ist der Entwickler unabhängiger vom Release-Zyklus des OpenShift-Jenkins-Client-Plug-ins und kann dieselbe OpenShift-Client-Version benutzen, in der der Cluster selbst betrieben wird.

Beim Arbeiten mit den Pipelines entsteht schnell der Eindruck, dass es zu viele Wrapper gibt, um letztendlich den eigentlichen Befehl für die Erzeugung der OpenShift-Objekte auszuführen. Zudem gibt es per Default kein einheitliches Vorgehen, wie auf die während des Builds erzeugten Laufzeitdaten wie Logs oder Screenshots zugegriffen werden kann. Hier ist deutlich sichtbar, dass Jenkins nicht dafür entwickelt wurde, auf dynamisch verteilten System-Landschaften wie OpenShift eingesetzt zu werden.

```
stage('Trigger PROD') {
  timeout(time: 2, unit: 'DAYS') {
    input message: '==> deploy PROD system?'
  }
 //… next steps
}
```

Listing 1

Das Hinzufügen eigener Node-Container ist zwar technisch möglich, aber eher als Workaround anzusehen als ein wirkliches durchdachtes Konzept, da es zu dem Thema keine Dokumentation gibt. Die Dokumente sind generell über viele Quellen verteilt: OpenShift, Kubernetes, Jenkins, Jenkins–Plug-in, Docker.

Als Gesamteindruck bleibt eher eine gefühlte Zwangsheirat als eine glückliche Ehe hängen. Mit viel Mühe und eigenen Workarounds lässt sich viel erreichen. Wer allerdings ein ready-to-use, Cloud-native denkendes Build-Tool erwartet, wird enttäuscht sein. Unabhängig von der eher trägen Performance des CI-Systems gibt es vor allem für das Handling der Build-Skripte im Entwickler-Alltag noch einiges an Verbesserungspotenzial.

Ziel muss es im Cloud-native-Kontext sein, Build-Pipelines schnell aufzubauen und zu verändern, was die Jenkins-OpenShift-Integration nur in einfachen Use Cases zulässt. Derzeit gibt es allerdings auch kein anderes Tool, das komplexe Use Cases standardmäßig abdecken kann. Als alternativ zu beobachtende Lösungen sind unter anderem das GitKube-Projekt *(siehe „https://github.com/hasura/gitkube")*, Skaffold von Google *(siehe „https://github.com/GoogleContainerTools/skaffold")*, GitLab CI *(siehe „https://gitlab.com")*, DroneCI/KubeCI *(siehe „https://www.kubeci.io")* und JenkinsX *(„https://github.com/jenkins-x/jx")* zu erwähnen. Es kann davon ausgegangen werden, dass sich beim CI/CD-Tooling für verteilte Systeme in den kommenden Monaten viel Spannendes tun wird und wir erst am Anfang einer langen Reise Richtung containerisierte CI/CD-Pipelines stehen.

Tobias Schneck
tobias.schneck@loodse.com

Tobias Schneck ist Senior Software Engineer bei Loodse. In seiner langjährigen Laufbahn hat sich Tobias auf Test-Automatisierungs- und CI-Projekte spezialisiert. Aktuell widmet er sich mit Leidenschaft den vielfältigen Anwendungsmöglichkeiten von Container-Technologien sowie Kubernetes und möchte diesen zum breiten Durchbruch verhelfen, um Entwicklern das Leben zu erleichtern. Tobias Schneck wurde von verschiedenen Fachkonferenzen als Sprecher eingeladen und organisiert das „Agile Testing @Munich"-Meetup.

iJUG Verbund

Alle Mitglieder auf einen Blick

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

www.ijug.eu

Impressum

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

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

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

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

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

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

Fotonachweis:
Titel: Original © jossdiim/123RF
S. 10: © Arunee Prasertsuk/123RF
S. 16: © Samantha Craddock/Fotolia
S. 24: © Aleksandar Pasaric/PEXELS
S. 30: © everythingpossible/123RF
S. 34: © pataralong/123RF
S. 35: © Andrew J. Russell, Bild aus dem Bestand der National Archives and Records Administration
S. 36: © US Central Intelligence Agency
S. 40: © Jakub Jirsak/123RF
S. 47: © inhabitant/123RF
S. 56: © deniskot/123RF
S. 59: © abscent/123RF

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

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

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

Inserentenverzeichnis

20. - 23. November in Nürnberg

2018.doag.org

PROGRAMM ONLINE
mit rund 450 Vorträgen

Eventpartner: AOUG SOUG swiss oracle user group iJUG Verbund ORACLE

JavaLand

Early Bird
bis 15. Jan. 2019

19. - 21. März 2019 in Brühl bei Köln

Ab sofort Ticket & Hotel buchen!

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu